

Module 1:
Introduction to Business Analysis



Thursday, 08th August



9:00 AM – 10:00 AM

Agenda:

- Course WBS
- General FAQs
- Business Analysis Basics
- 15 mins hands-on Practice session

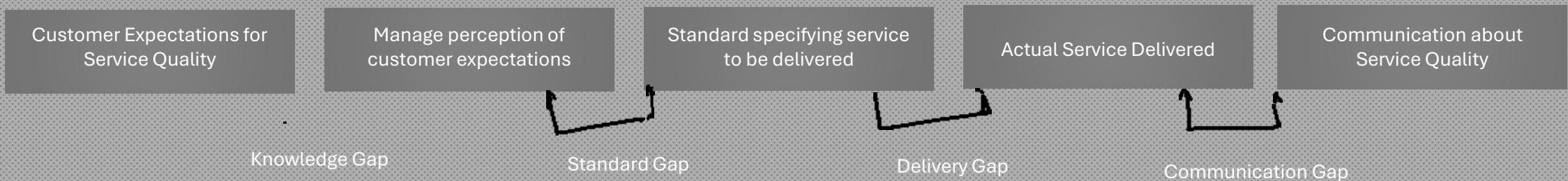


SAHIL DATERO
Business Analyst, Delivery

What it's Like to Be A Business Analyst?

- Business Analysts are always responsible for leading change, creating clarity, and driving alignment
- Business Analysts works with stakeholders across the organization to understand business objectives driving the change, define scope change, analyse and specify detailed requirements related to change and finally support implementation of change
- Business Analysts bridges multiple different Gaps among all stakeholders and ensures implementing change the generates as much value as possible

According to BABOK Guide® Version 3:
Business Analysis ultimately helps organization to understand the needs of the enterprise and why they want to create change, design possible solutions, and describe how those solutions can delivery value



What Projects will I Work On?

A number of software projects may benefit from the contributions of a business analyst. Common Types of projects includes the following - Business Process Improvement and Change, Business Intelligence/ Data warehousing, New product or service development, Customizations to existing products or service

What you need to know about Business Analysis?

Underlying Core Competencies

- Problem-Solving And Critical Thinking
- Analytical Thinking
- Relationship Building
- Self Management
- A thick Skin
- Driving Improvements
- Communication

		Iterative Model (Verification & Validation)	Agile / SCRUM	Waterfall Model
Collaboration Techniques	User Acceptance Testing	Requirements Specifications	Scope Statements	Functional Requirements
Product Backlog	User Stories and Acceptance Testing	User Interface Specifications	Traceability Matrices	Diagram and Visual Models

How to measure the success of a Business Analyst?

Project Success Measures: On-Time Delivery, On-Budget Delivery, On-Specification Delivery, ROI

Business Analyst efforts level: Minimized requirements change requested late in the project, Minimized requirements defects discovered after approval, Efficiency of the requirements effort, Increased business benefits (return part of ROI)

How to expand your Business Analysis Experience?

You must use deliberate practice to gradually get better at the skills – BA skills and related. It is steady accumulation of a related set of skills of your craft that makes you become truly great at what you do.

- ① Participate in requirements and use case review meetings
- ② Establish a new testing program to streamline quality of an aspect in system
- ③ Build Strong Product and stakeholder relationship
- ④ Learn how system/application/ product works

How working with different stakeholders looks like?

- | | |
|-------------------------------|--|
| Work with Business Analyst? | - Facilitate a meeting
- Take notes at a meeting |
| Work with Technical Roles? | - Collaborate with customers or Stakeholders
- Demo Software
- Become a Critical Consumer of Requirements
- Analyse A (Business Process)
- Help Select New Software
- Do Business Analyst Work
- Solve A new Problem or Create a New Benefit |
| Work with Business Functions? | - Become a Subject Matter Expert (SME) on A project
- Be A Facilitator SME
- Help Represent another Department on A project
- Lead a Project in your Department
- Facilitate A Process Improvement Session
- Help Conduct A Return On Investment (ROI) Analysis
- Become The Point of Contact for Technical Issues |

How to connect with Business Analysts Professionals?

Your goals for networking must include:

- ① Learn about Business Analysis Profession
- ② Understanding the state of the profession in your location or target location
- ③ Establishing mutually beneficial relationship with other professionals
- ④ Contributing to the professional community
- ⑤ Finding your next Position

Build your Online Information Routine

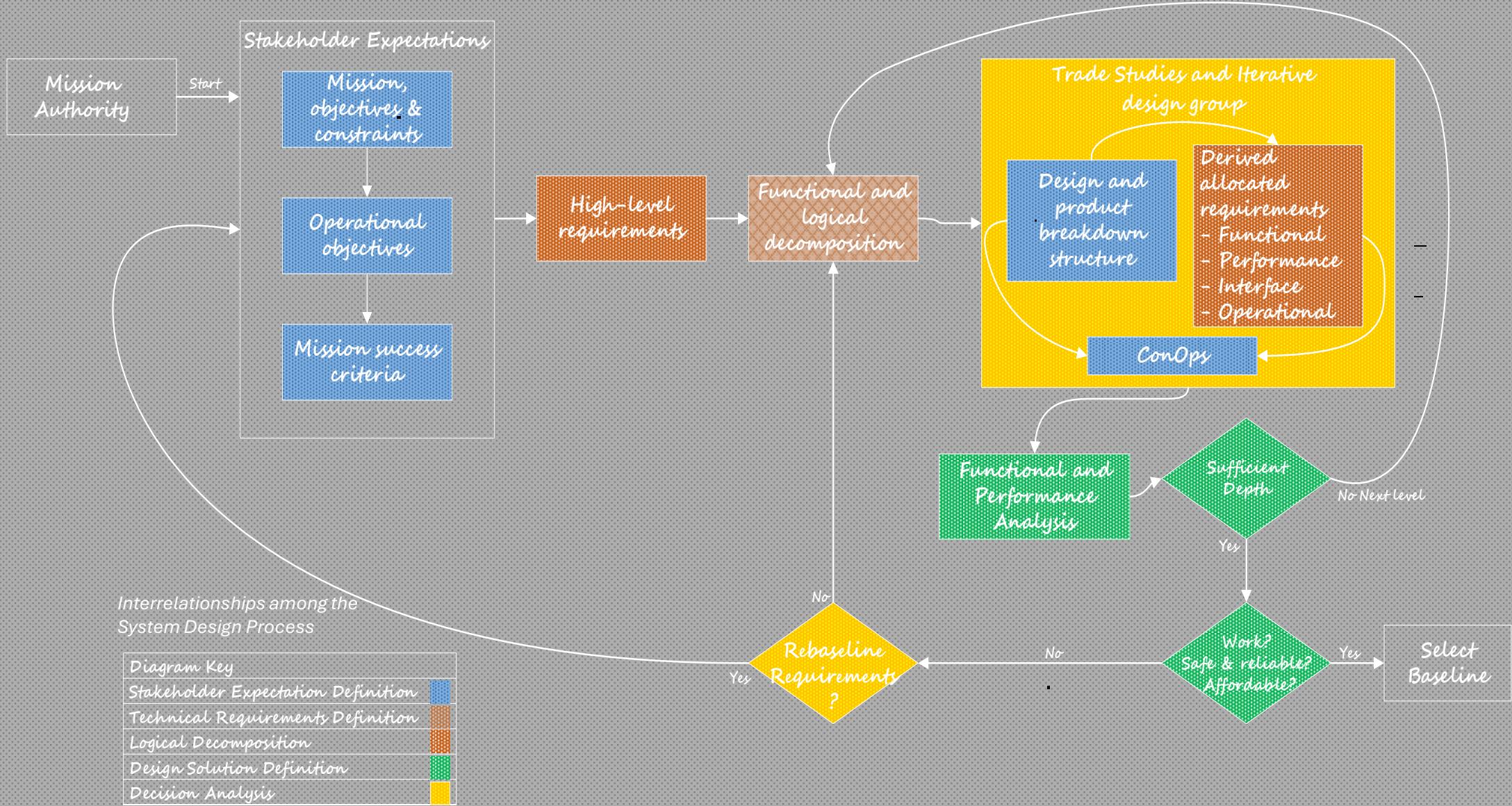
Begin Incorporating online business analyst activities into your daily or weekly routine. Refer to following sites to find more resources

- www.betimes.com
- www.thebacoach.com
- www.modernanalyst.com
- www.practicalanalyst.com

Begin researching from research papers and relevant books



System Design Process flow diagram





When poll is active respond at PollEv.com/sahildaterao222



Is there any difference in the meaning of words, 'Requirements' and 'Specifications'?

Yes, specifications include requirements but also contains other things such as blueprints, etc.

0%

No, they are essentially the same.

0%

'I am not sure'.

0%

Yes, requirements are the inputs to the design process, while specifications are the output.

0%

Requirements vs Specifications

Requirements describe what the product or system shall/should do:

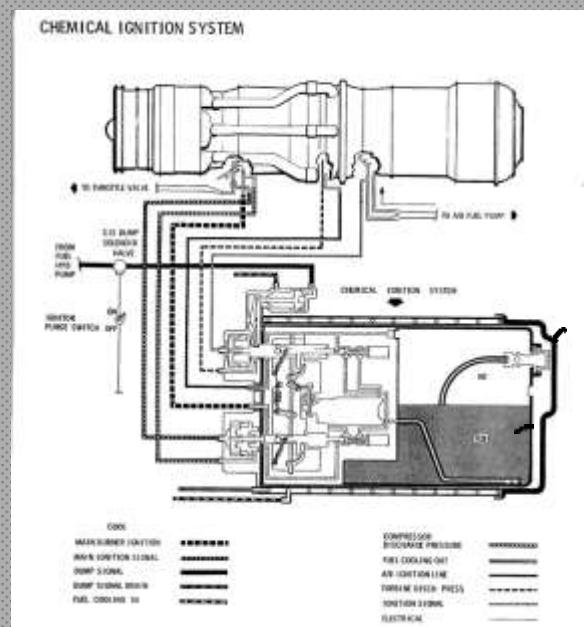
- ✓ Functions it shall perform
- ✓ How well it should perform these
- ✓ Degree of automation of the system (what operators must do)
- ✓ Compatible with other devices
- Open environment (Ideal)

"Temperature of oil in oil supply system, shall be reduced using, fuel-oil cooler, (use of cooler oil)"

Aircraft Dimensions	
Wing Span	55.62 ft.
Length (overall)	101.6 ft.
Height (to top of vertical stabilizer)	18.65 ft.
Thread (MLG center wheels)	16.67 ft.

Specifications describe how the system is built and works

- ✓ The form is made of..
 - Materials used in the system
 - Overall dimensions
 - Schematics, blueprints, etc
 - User Interface
 - Laboratory - Simulated



"Using Cameo Systems Modeler model shall be verified"

"If mixed fuel temperature < 265 ° F, all fuel shall be burned using afterburner, if > 265 ° F flow of hot fuel should be rerouted to tank 4, if tank 4 is full, fuel shall be rerouted to tank 5"

Assisting Engineers with Iterative Verification & Validation Testing Model

From conception to retirement, product goes through Phases i.e set of activities, milestones, set of reviews and audits. Conception stage, feasibility stage, development stage, utilization/support stage and retirement stage. contains series technical audit reviews for ***qualitative*** analysis.

In System Requirement review,

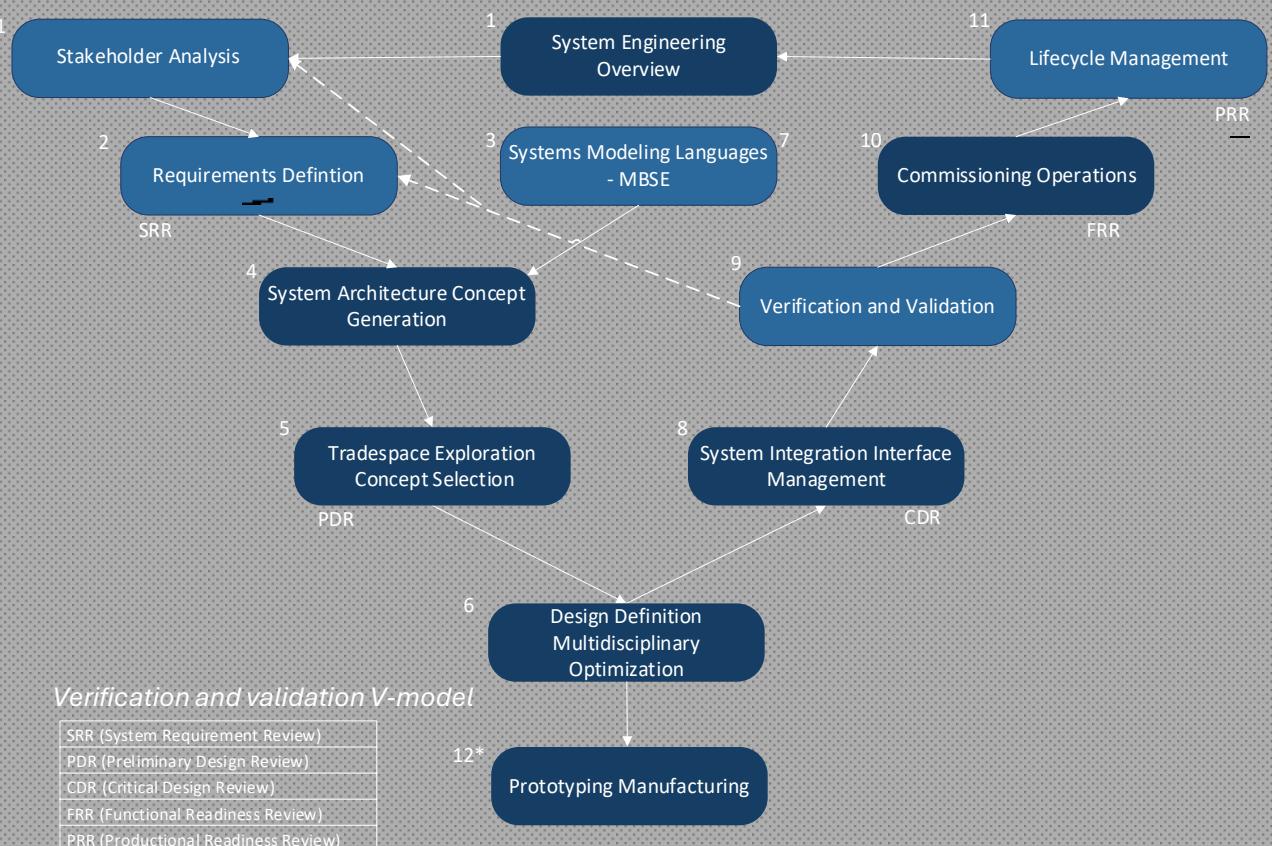
- Approve start of project
- Gather stakeholder's requirements
- High level Functional Architecture
- Conceptual Design
- Draft specs

In Preliminary Design review,

- Start component Testing
- Approve Performance Item Specs
- Approve Preliminary Design
- End of Requirement phase

In Critical Design review,

- Approve Final Design
- End of Design Phase



Assisting Engineers with Iterative Verification & Validation Testing Model

V - Model facilitates configuration audit... Integration tests workshops, test readiness review...

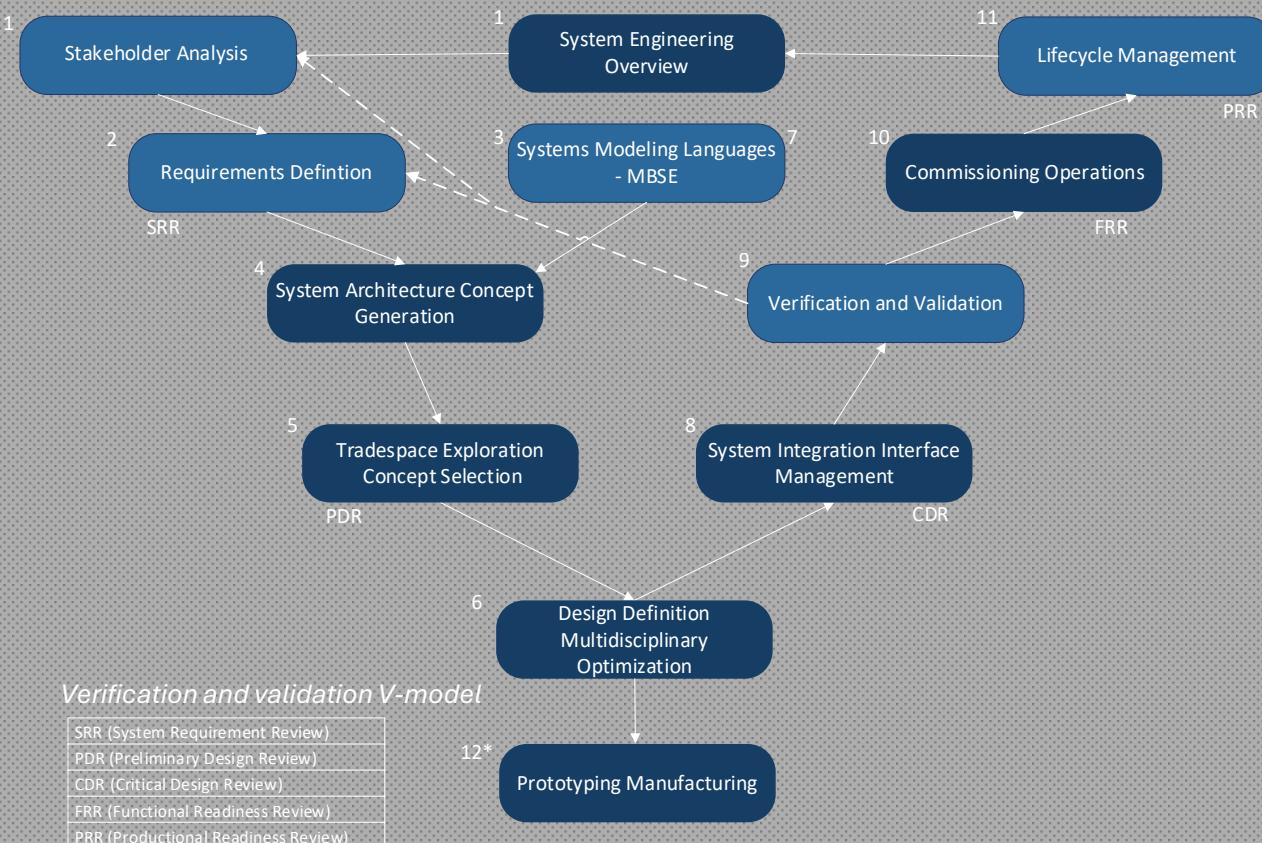
Objective of V-Model is failure **detection** during the execution of individual component testing, failure in interaction between the components or sub-system is detected, to check for deviations in existing behavior of system and requirements, and whether the services on which client and contractor agreed are provided.

In Functional Readiness review,

- Process integration specs meets customer requirements
- Audit first user acceptance
- End of testing

In Production Readiness/ Approval review,

- Formalizes (corrected) Product Baseline for Production
- Approve Start of Production



Assisting Engineers with Iterative Verification & Validation Testing Model

V - Model facilitates configuration audit... Integration tests workshops, test readiness review...

Objective of V-Model is failure **detection** during the execution of individual component testing, failure in interaction between the components or sub-system is detected, to check for deviations in existing behavior of system and requirements, and whether the services on which client and contractor agreed are provided.

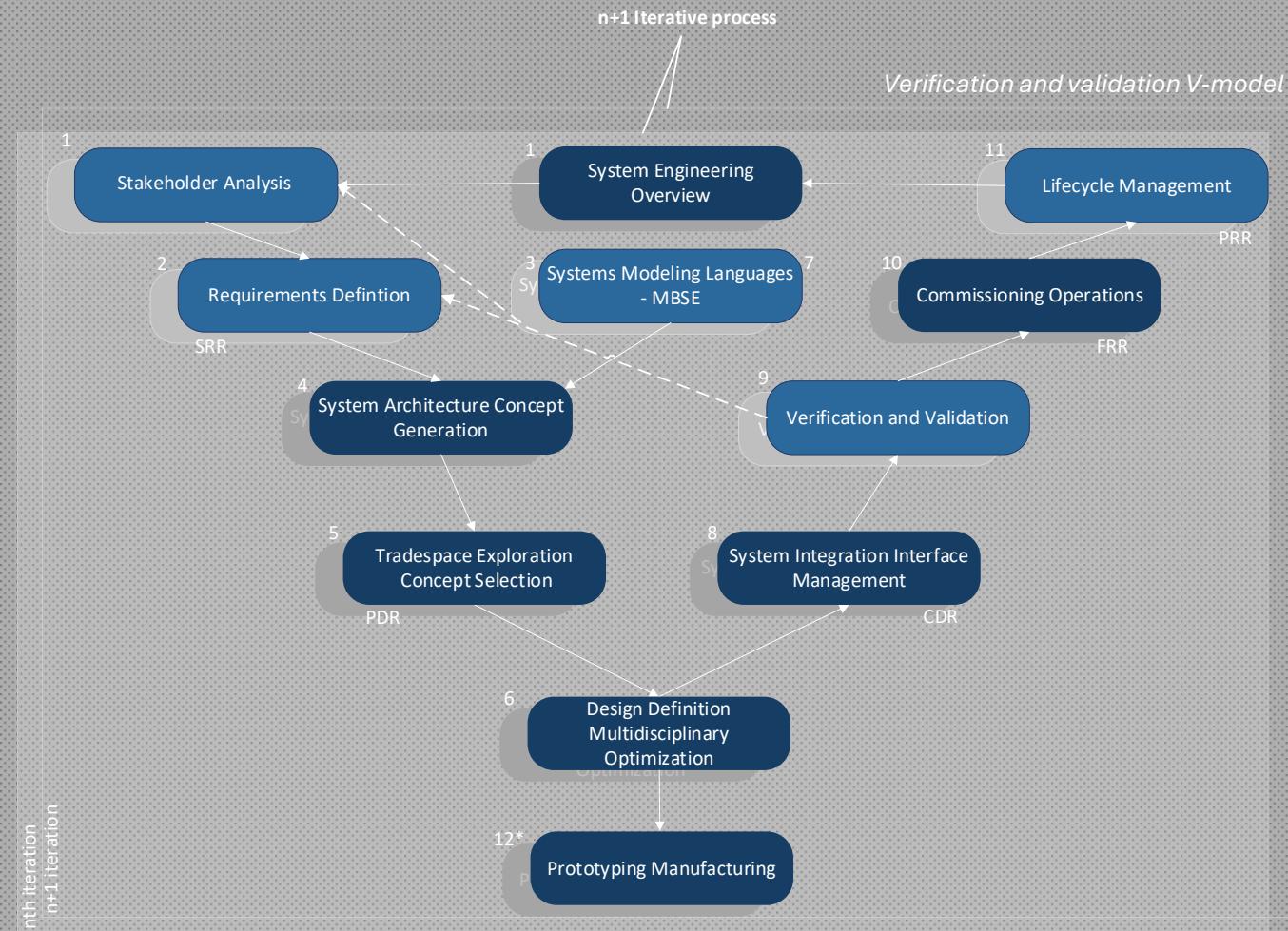
In Functional Readiness review,

- Process integration specs meets customer requirements
- Audit first user acceptance
- End of testing

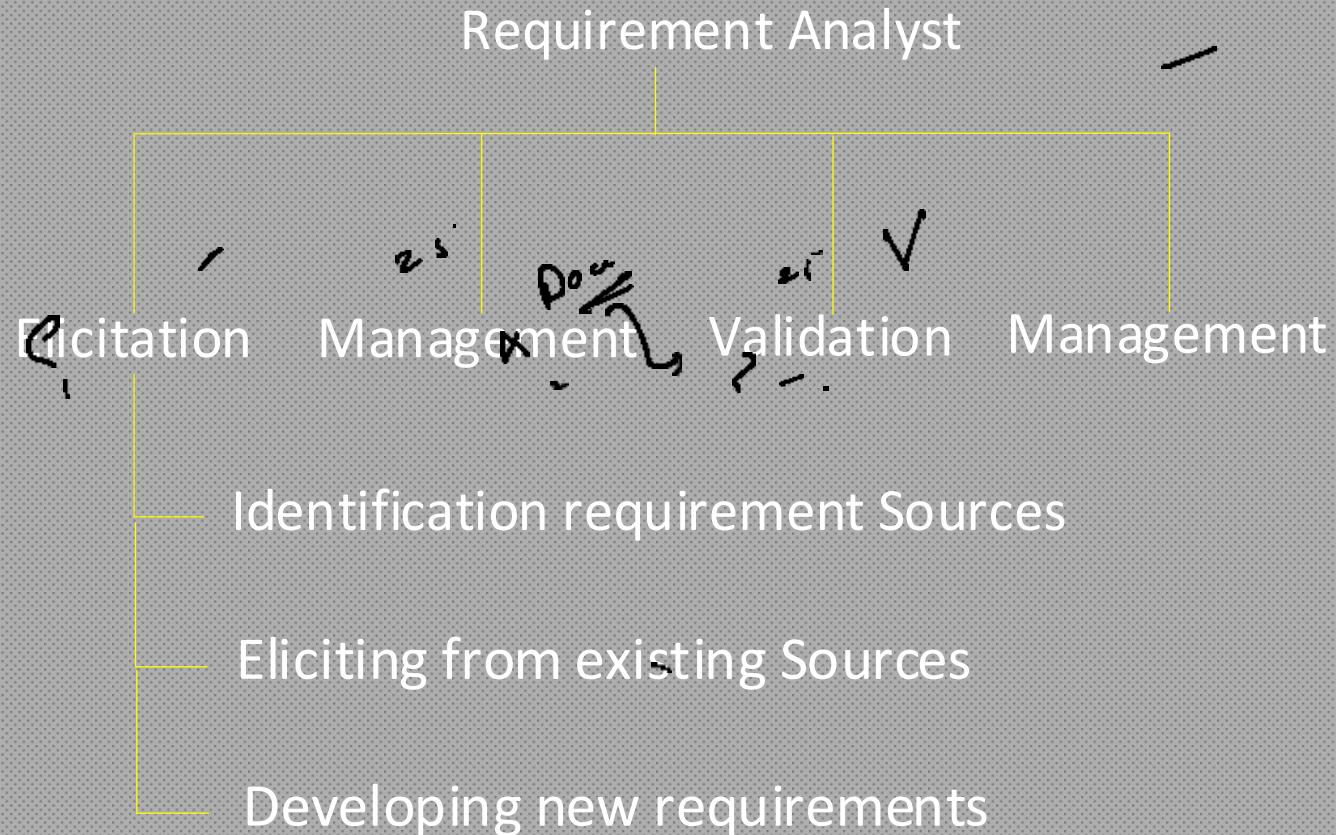
In Production Readiness/ Approval review,

- Formalizes (corrected) Product Baseline for Production
- Approve Start of Production

V-Model provisions **iterative** process for design of low-level requirement fulfilment review



Turning Stakeholder's interests, needs into logical requirements



Main goal of elicitation activity is elicitation of relevant requirement at required level of detail for system to be developed

Turning Stakeholder's interests, needs into logical requirements



For each technique, expert would know,

- ✓ Hints for preparation, execution and follow up work of the technique.
- ✓ The benefits of the technique for elicitation sub-activities.
- ✓ An effort Estimation for the technique.
- ✓ Critical Success factor of the technique.
- ✓ A checklist for important hints for applying the techniques.

I like the fact that business analysis work does not change as fast as software development but that I am Continuously learning.

~ Doug Goldberg, Senior Business Analyst

Module 2:
Requirement Analysis



Friday, 09th August



9:00 AM – 10:00 AM



SAHIL DATERO
Business Analyst, Delivery

Agenda:

- { • Requirement Definition – Types, Best Practices
- { • How to write good requirements?
- 15 mins hands-on Practice session
- Course WBS

The Term ‘Requirement’

- The IEEE 610.12-1990 standard defines the term “requirement” as follows

- ① A condition or capability needed by a user to solve a problem or achieve an objective
- ② A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents
- ③ A documented representation of a condition or capability

Definition: Requirements Artefact
“A Requirements artefact is a documented requirement”

Requirement Types

- Three main types of requirement

(See 830-1998; Lauesen 2002; Robertson and Robertson 2006; Sommerville 2007; Wiegers 2003]):

- Functional Requirements
- Quality Requirements
- Constraints

① Quality Requirements

- Quality requirements define quality properties of the system to be developed, e.g. performance of the system, its reliability, or its stability.
- Quality Requirements are often the architectural drivers

Performance Requirements

That needs to define how well the system needs to perform the function (latency rate, throughput rate, click per seconds)

Definition: Quality Requirement
“A Requirements defines a quality property of the entire system, or system component, service or function”

Types of Quality Requirements

Quality Property	Short Description
Availability	Availability refers to the percentage of time during which the system is actually available for use and fully operational
Efficiency	Efficiency is a measure of how well the system utilised hardware resources such as processor time, memory, or communication bandwidth
Flexibility	Flexibility indicates how much effort is needed to extend the system with new capabilities
Integrity	Integrity denotes how well the system is protected against unauthorised access, violations of data privacy, information loss, and infections through malevolent software
Interoperability	Interoperability indicates how easily the system can exchange data or services with other systems
Reliability	Reliability is the probability of the system executing without failure for a specific period of time
Robustness	Robustness is the degree to which a system or component continues to function correctly when confronted with invalid inputs, defects in connected systems or components, or unexpected operating conditions
Usability	Usability measures the effort the user requires to prepare input for, operate and interpret the output of the system
Maintainability	Maintainability indicates how easy it is to correct a defect or make a change in the system
Portability	Portability relates to the effort it takes to migrate a system or component from one operating environment to another
Reusability	Reusability indicates the extent to which a component can be used in systems other than one for which it was initially developed
Testability	Testability refers to the ease with which the software component or integrated system can be tested to find defects

② Functional Requirements

Definition: Functional Requirements

These {Functional Requirements} are statements of services the systems should provide, how the system should react to particular inputs and how the system should behave in particular situations). In some cases, the functional requirements may also state what the system shall not do

Interface Requirements

That describe conditions under which functional and performance shall be fulfilled

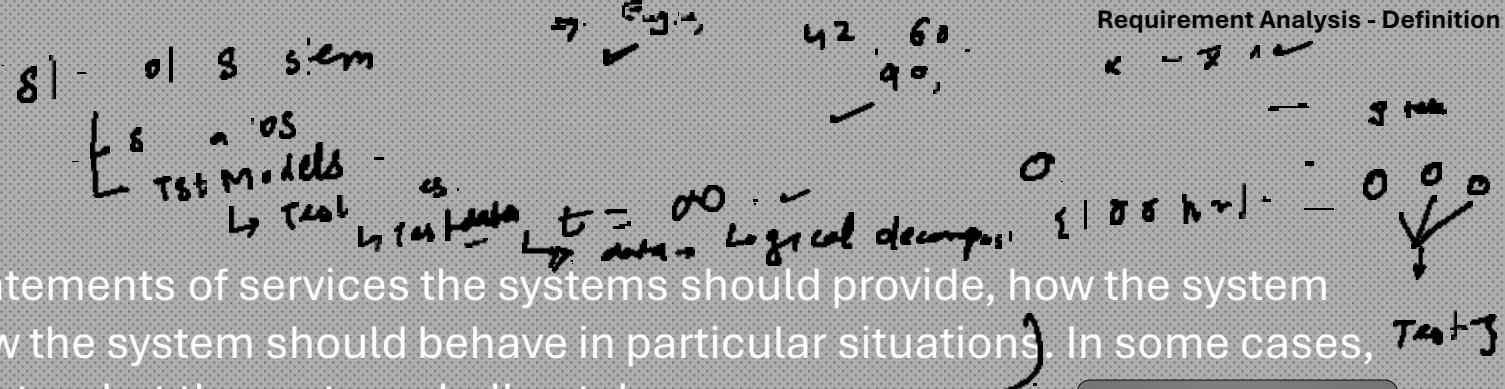
Environmental Requirements

That describe environmental conditions like user acceptance testing, atmospheric pressure, temperature, fluid turbulence

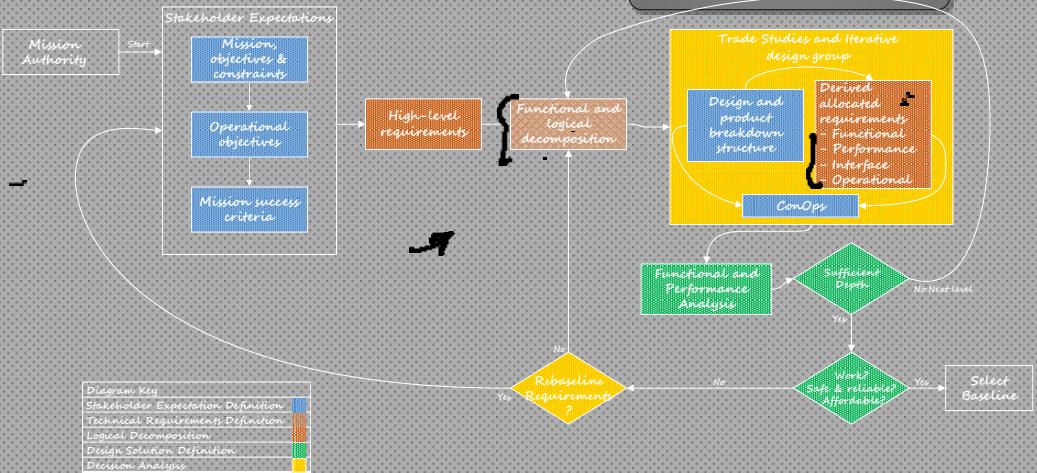
Rq

- Traditionally, functional requirements are documented using three complementary perspectives {The data perspective, the functional perspective, and the behavioural perspective}

- The requirements defining the data, functions, and behavior of the system are, in most cases {solution oriented requirements} since they are defined in a way that supports the realisation of the system. (In contrast to this goals (using English word 'should') are defined in a solution-neutral way)



[Sommerville 2007]



Mot 3
Modelling



~~so~~ → 70 - 80 % Req.

Definition: Constraints

'A Constraint is an organisational or technological requirement that restricts the way in which the system shall be developed'

30 day 35-25

③ Constraints

- Constraints either restrict the development process or the properties of the system to be developed.
- Constraints are typically superimposed by other organisation process (eg. Time, resource constraint from project management) or by environment or context in which system shall operate

Cultural Constraints

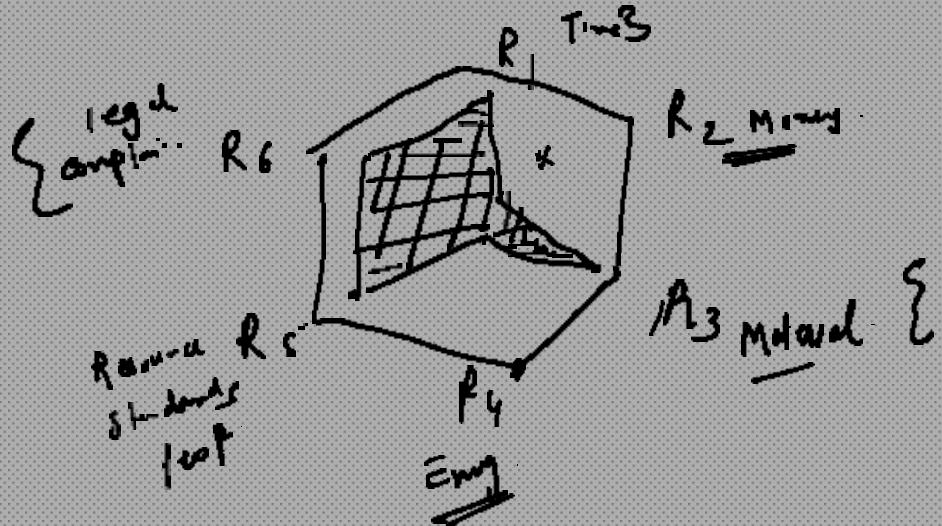
The constraint originating from the cultural background of the system user

Legal Constraint

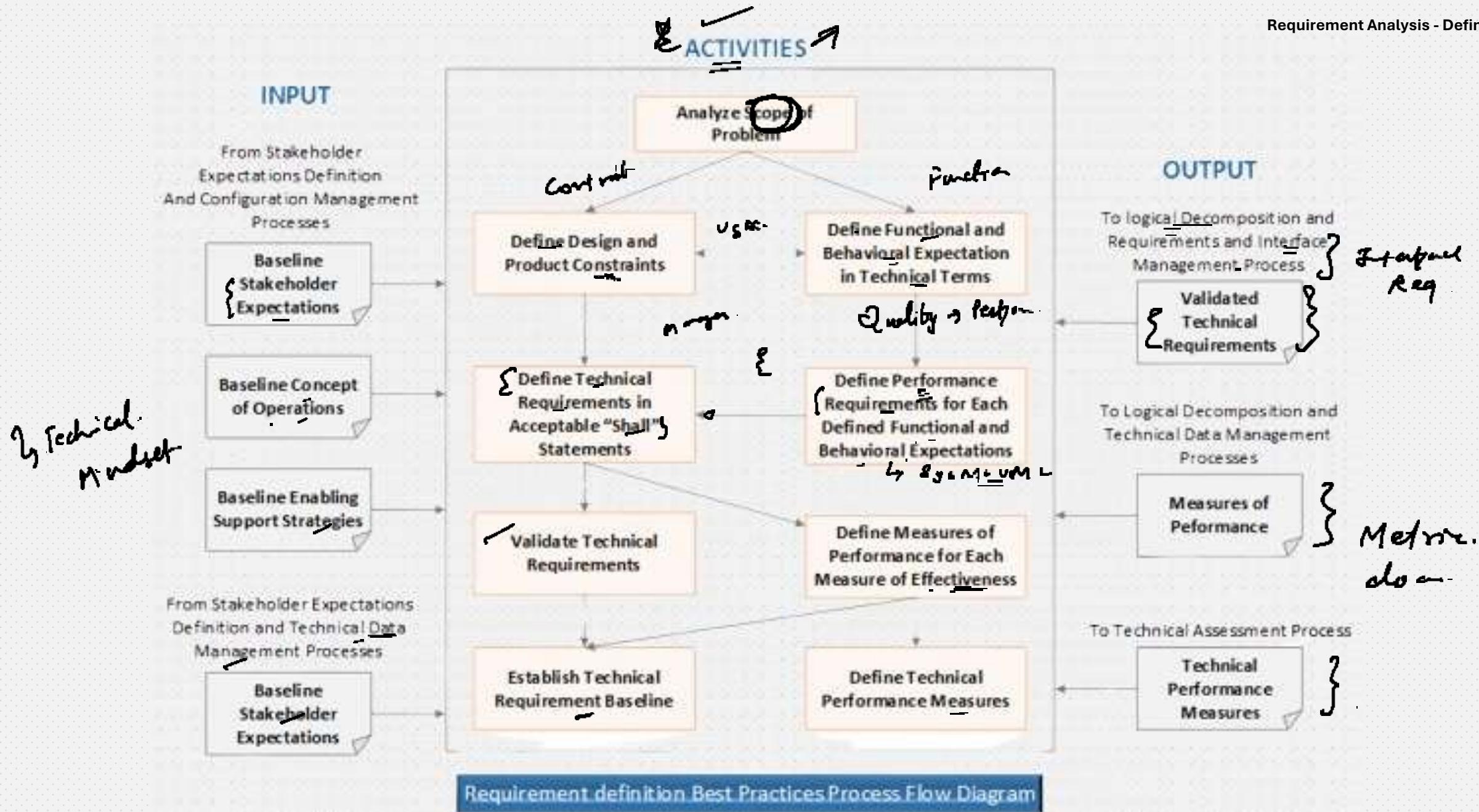
Constraints originating from laws and standards

Constraint affecting system itself

A constraint affecting the development process, eg. Architectural design process, implementation process, the test process, or several of these processes



- Range of realisation alternatives for requirements **without** considering constraints
- Range of possible realisation alternatives for requirements with the consideration of constraints
- Restricting the effect on a requirement



Writing understandable, verifiable, easy to read requirements

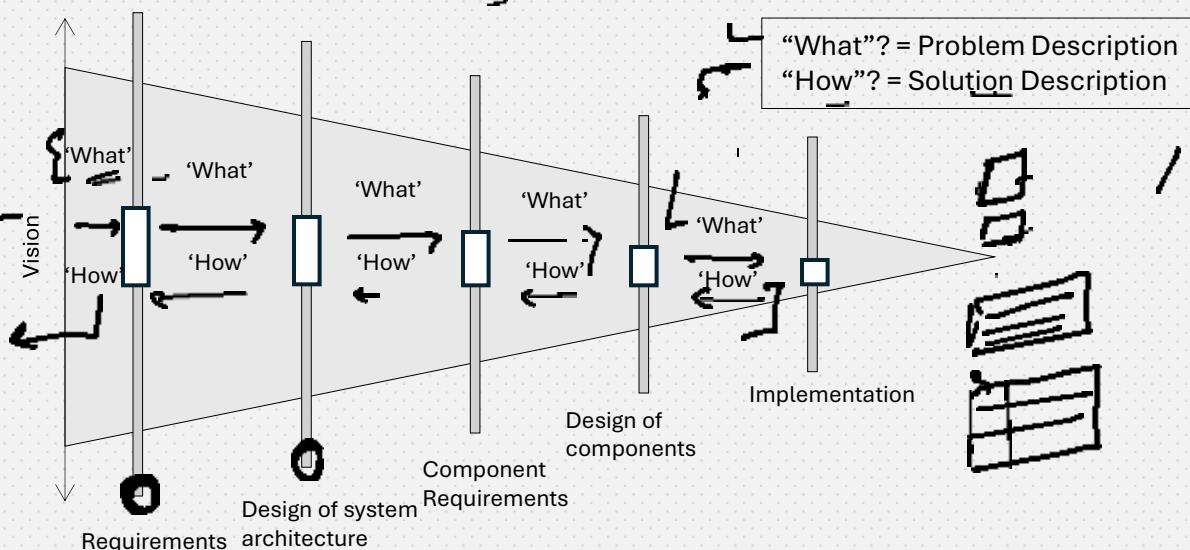
- Use keyword 'Shall' to identify requirements
- Explain how output responds to invalid input, valid inputs, and timing related events
- Make each requirement unique
- Assign unique identifier to each requirement
- Express requirements in terms of behavior
- Write for target audience – Design engineer and Validation engineer
- Describe "What" is done, not how it is done → Problem statement
- Use Description of outputs that can be observed and description of inputs that can be controlled

$t = 0$ ↗ fuel system
 $t = 30$ ↗
 Unique ID → Traceable ↗
 Scenario Based ↗
 Lab ready ↗ tests ↗
 System Test ↗ Time

Type

Problem

Solution



Reduction of solution space during the development process

Solution-oriented Approach I

Elements in a good requirement artefact

- Function Name
- Definition of terminology
- Trace links
- Justification for derived requirements
- Outputs – description – Units – Encoding
- Inputs affecting output
- Power-on behavior: Behavior of signal
- Reset response: State of response after reset
- Invalid Behavior

Continuous learning is the minimum requirement
for success in any field

~ Brian Tracy, Coach

Module 2:
Requirement Analysis - Management



Monday, 12th August



9:00 AM – 10:00 AM



SAHIL DATERO
Business Analyst, Delivery

Agenda:

- Three Aspects of managing requirements
- C4 Model for defining system context
- Dealing with Challenges with Requirement
- 15 mins Hands-on practice

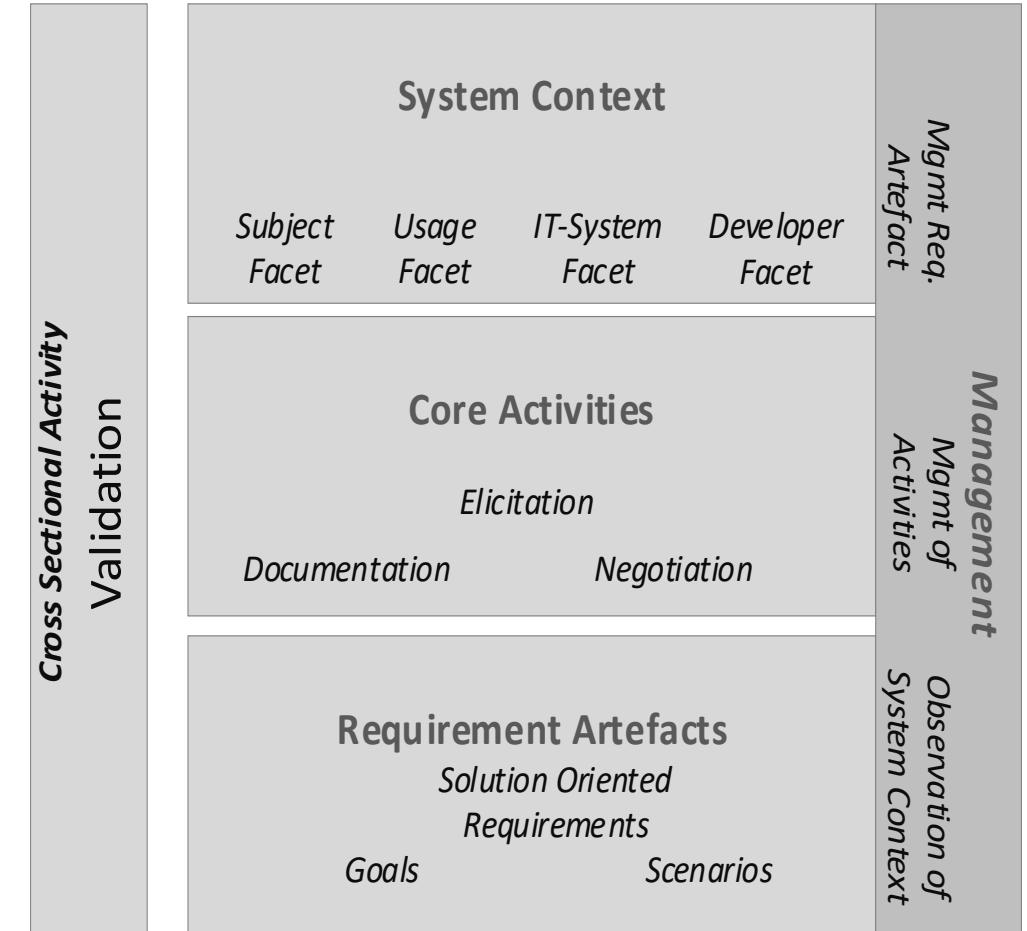
Three aspects considered in Requirement Management

While managing requirements we must consider three main aspects:

- The management of Requirement Artefacts
- The Identification and Management of Context Changes
- The management of Requirement Engineering Activities

I . Managing Requirement Artefacts

- Managing for large amount of requirement artefacts elicitation, documentation, negotiation, validation, allocation to system components, refinement, implementation and testing
- Managing Relation of these artefact with other requirement artefacts, design and test artefacts
- Aim of this management sub activity is to continuously keep track of all requirement artefacts, their relevant attributes, and relationships as well as their evolution



Five Activities of Managing Requirement Artefacts

① Definition of a requirement attribute scheme:

- Project specific and domain specific attributes identifier, name, requirement type, version, author, status, priority etc.

② Requirement Traceability

- Traceability from its origin i.e requirement source/ derived or refined requirement/ higher level requirement) to its realisation
- Requirements should be related to quality assurance artefacts to validate/verify the requirements.

③ Requirement Change Management

- Requirements undergo changes through-out development process caused by, changes in system context or by feedback from architectural design and testing.
- Change Management for requirements should ensure that changes are integrated consistently into existing requirement artefact.

④ Requirement Configuration Management

- Based on version management, fixes definition at point of time, manages evolution, groups set of related version together.

⑤ Requirement Prioritisation

- Prioritisation of requirements based on importance, risk, stability etc.

Elements in a good requirement artefact

- Function Name
- Definition of terminology
- Trace links
- Justification for derived requirements
- Outputs – description – Units – Encoding
- Inputs affecting output
- Power-on behavior: Behavior of signal
- Reset response: State of response after reset
- Invalid Behavior

II . Observing the System Context

Goal of this sub-activity is to identify changes in the context and estimate the impact of these changes.

Typical examples, of context changes are

- A new technology or a new competing product emerges
- A law or standard changes
- Evolution of stakeholder goals
- Change of an organisation policy
- Changes in the way that external actors (stakeholder or systems) use the system

① Techniques for context observation

- Context Scanning: Systematic search for changes in environment of system. Periodically or on occurrence of a special event.
- Context Monitoring: Continuous recording of evolution of the context. Tracking and interpretation of requirement changes.
- Context Prognosis: Forecasting context changes

② Structured Observation of the Context

A major challenge for context observation is to detect and anticipate changes in all four context facets as early as possible

- Observation of the subject facet
- Observation of the usage facet
- Observation of the IT system facet
- Observation of the Development facet

In principle, each type of context aspect (requirement sources, context objects, properties and relationships of context objects) can change. Changes in the context facet hence comprise:

- Changes of requirement sources, changes of context objects, changes in properties and relationships of the context objects
- C4 Model for defining system context (Code – Component – Container – Context)

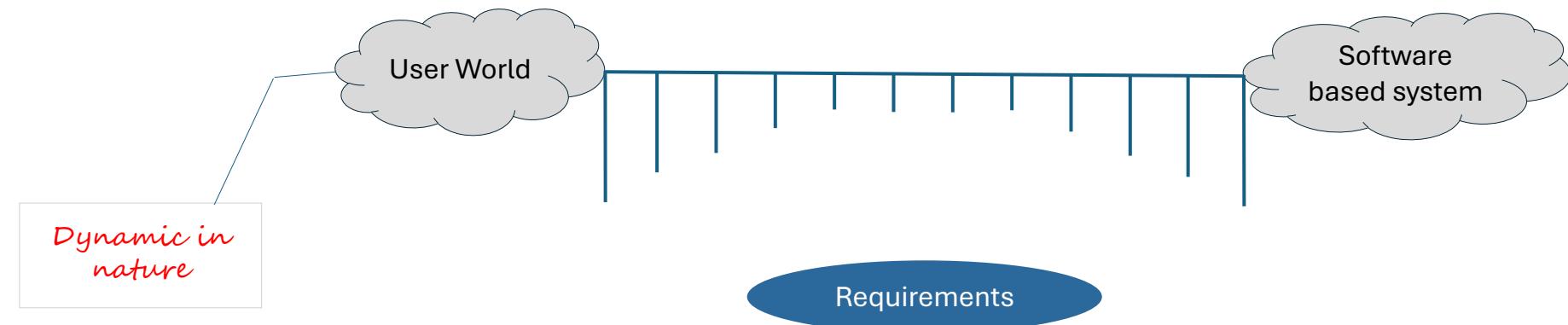
Dealing with Challenges with Requirement

Four major issues that affect requirements and requirement engineering processes

- Changing Requirements
- Differing Perspectives
- Lack of Standardizations
- People and Politics

① Changes in requirements

- System requirements reflects the world outside the system.
- As this is constantly changing then the requirements will inevitable also change
- It is often difficult to understand the implication of changes for the requirements as a whole



Range of sources that can affect overall requirements of the system

Technology changes

New Technologies may have to be incorporated into the system

Organizational Changes

The business structure and organization may change

Market Changes

The market for a system may change because of other systems introduced by competitive changed consumer expectations

Economic Changes

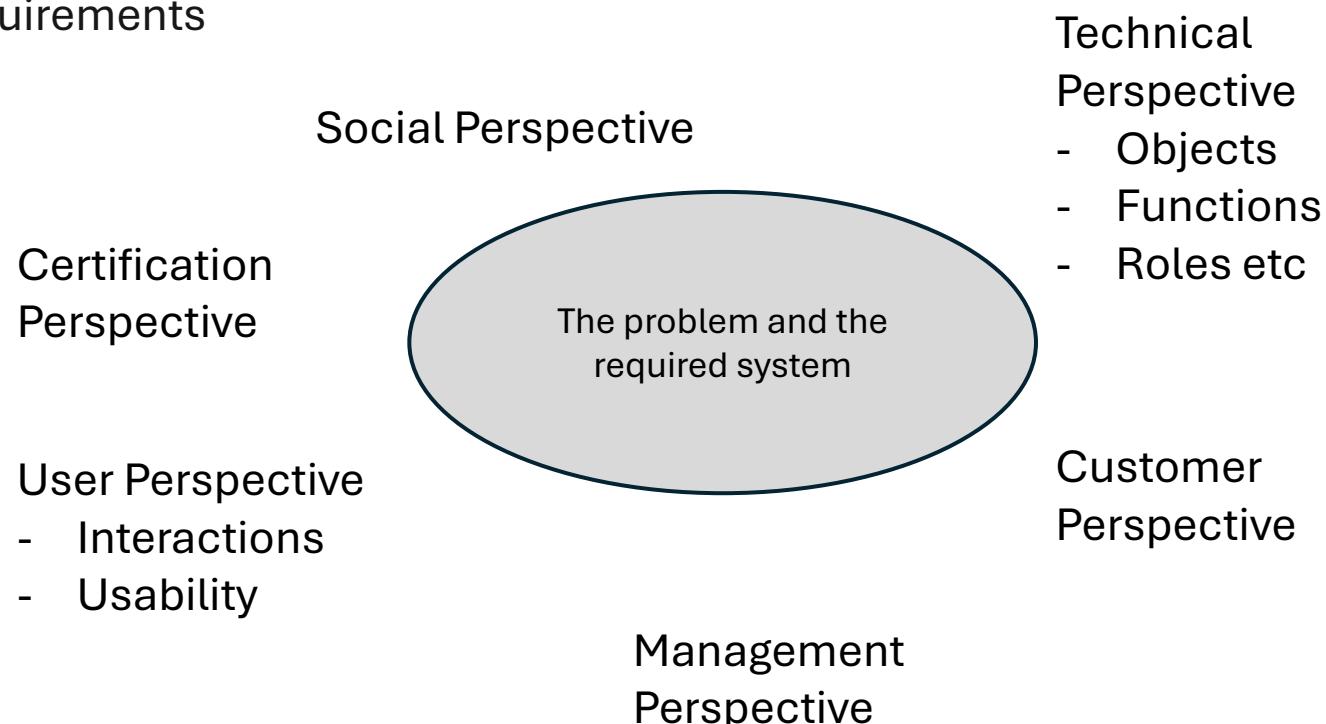
The Business may do better/ worse than before (scale-up/ scale down organization costs)

Political and Legal Changes

External events lead to a change in government policies or regulations

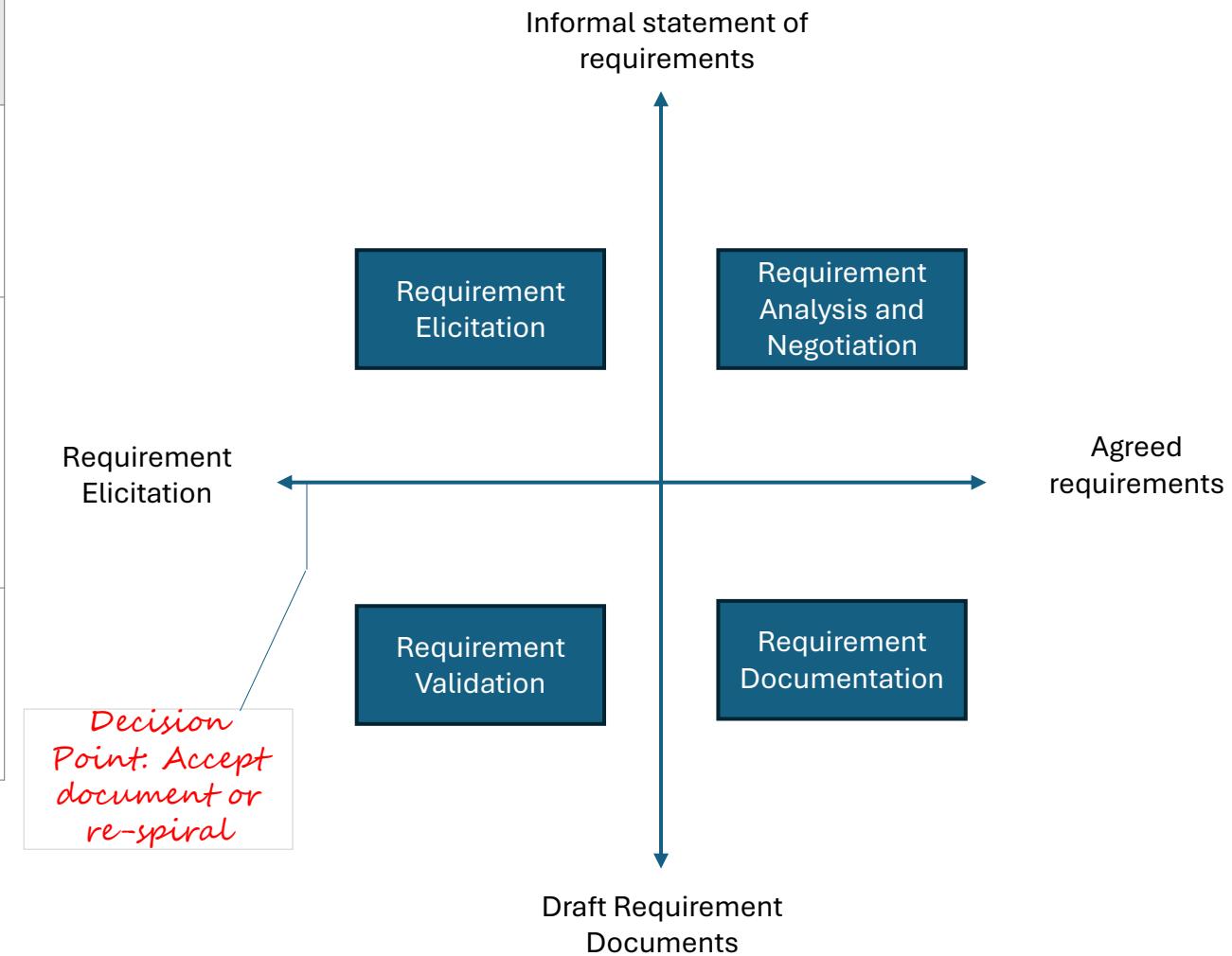
② Conflicting Stakeholder Perspectives

- Different Perspectives are not consistent and different stakeholders will want different thing from a system
- It is inevitable that some requirements will conflicting so that it is impossible to satisfy all stakeholder requirements without compromise
- Developing detailed requirements for future systems often cannot be given a high priority by senior people who will be affected by these requirements
- It is difficult to arrange meetings and stakeholder do not have time to think deeply about the system
- They therefore express their requirements as vague, high-level descriptions which have to be interpreted by engineers
- The level of detail required in a requirements specification differs greatly depending on type of product being developed, different way of expressing specifications



III. Managing requirement Engineering Activities

	+ (no deficiencies)	0 (minor deficiencies)	- (major deficiencies)
Content: Sample Criteria: 1) Completeness 2) Correctness	The artefact meets defined criteria	The artefact does not satisfy some of defined criteria	The artefact does not meet significant part of the defined criteria
Documentation Specification: Does the documentation/specification of the requirement comply with given documentation/specification rules	The artefact adheres to the defined documentation/specification rules	The artefact violates some of the defined documentation / specification rules	The artefact violates a significant number of defined documentation / specification rules
Agreement	Agreement about the artefact is established	There are relevant unresolved conflicts about the artefacts	There is not agreement about the artefact



Module 2:
Requirement Analysis - Management



Tuesday, 13th August



9:00 AM – 10:00 AM

Agenda:

- Requirement Engineering, Management - Benefits
- Requirement Landscape for Requirement Information Model
- Standards to refer



SAHIL DATERO
Business Analyst, Delivery

Defining Requirement Engineering and Management

Requirement Engineering is a systematic and disciplined approach to specification and management of requirements with the following objectives:

1. Knowing the relevant requirements, establishing consensus among stakeholders about requirements, documenting the requirements in compliance with given standard, and managing requirements systematically
2. Understanding and documenting stakeholder needs and desires
3. Specifying and managing requirements to minimize the risk that the system does not meet the stakeholder's desires and needs

Requirement Management is the process of managing existing requirements and requirement-based artefacts. In particular this includes documenting, changing and tracing requirements.

Defining Requirement Engineering and Management

Ebert [Ebert 2012] also defines the following:

“Requirement Management is a part of Requirement Engineering that is concerned with the maintenance, management and further development of requirements in the lifecycle”

Three main constraints [RUSO 2009] makes the requirement management tasks more complex

- ① Requirements have to be used by multiple persons
- ② Requirements are supposed to be reused
- ③ Requirements change

Requirement management simplifies Requirement Engineering:

- Structuring of requirement and the requirements documents (assigning attributes, filtering, sorting)
- Standardization of Terminology (glossary)
- Definition of clear processes and work steps to be performed (in the change process)

Goals and Benefits of Requirement Management

Amongst other things, requirement management provides answer to the following questions:

- What different types of requirement are there? Requirement Landscape
- To what level of details are requirements documented? Requirement Landscape
- Which requirements have already been accepted? Assignment of Attributes
- Which requirements comes from which source? Assignment of Attributes
- Which requirements are urgent and important and therefore candidates for the next release? Evaluation and Prioritization
- Which requirements generate costs that are high with few benefits? Evaluation and Prioritization
- Which requirements belong to a specific software baseline? Version Management
- Which version of requirement was implemented in the system? Versioning
- Who was the last person to change the requirement and why did they change it? Versioning
- Which technical component belong to which requirement? Traceability
- Which test cases belong to which requirement? Traceability
- Which requirement is part of system/product delivered? Traceability
- How do the two variants of product differ? Variant Management
- What Proportion of requirements has already been implemented and tested? Reporting
- How long does it take on average for a change request to be implemented? Reporting
- Has the requirement engineering process been improved in a specific measure? Reporting

Goals and Benefits of Requirement Management

Requirement Management is more difficult and more important

- the Greater the number of requirements that exists
- the Longer the estimated lifetime of the product is
- the Greater number of changes that are expected
- the larger the number of participants in Requirement Engineering process is
- the More difficult it is to reach or involve the stakeholders
- the higher the quality requirements of the system are
- the Greater the number of reused that is to be performed
- the more complex the development process is
- the more in-homogenous stakeholders opinions are
- the greater number of releases that will be developed
- the more important the use of standard is for project

Good Requirement Management...

- Increases the quality of the requirements, products and processes
- Reduces the project costs and project duration
- Makes it easier to monitor complex projects and processes
- Improves communication within and among team
- Increases customer satisfaction
- Reduces the project risk

Some important Standards viz. Requirement Engineering and Requirement Management

Standards	Describes
ISO 9000/ ISO 9001 – Quality Management Systems ISO 9001:2008	Improvement based on identifiability and traceability of requirements. Clause 7.5.3
ISO/IEC 12207: 2008 and 15288: 2008	“Software Lifecycle processes” and “Systems and software engineering” – “Systems Lifecycle Processes”, “System Requirement Analysis”, and “Software Requirement Analysis”, “Stakeholder Requirements Definition Processes”, “Requirement Analysis Process”
IEC 61508 [DIN 61508] “Functional Safety of Safety-related electrical/ electronic/ programmable electronic systems”	Standard for safety integrity levels that defines the risk
SOX (Sarbanes-Oxley Act) [USCo2002] US Federal Law	About who made what changes when, and thus relates to core concept of requirement management
VDI Guideline 2519, [VDI 2001]	German standard for knowing Procedure for creation of customer requirement specification, system requirement specification
IEEE 830-1998	“Recommended practice for software requirement specification”
ISO/IEEE/IEC 29148: 2011	“Systems and Software engineering – Lifecycle processes, Requirement Engineering”
IEEE Standard 1233	“Guide for developing System Requirement Specification”
ISO/IEC 14102:1995	Evaluation and Selection of CASE tools
ISO/IEC 25010:2011	“System and software engineering – System and Software Quality Requirements and REValuation [SQuaRe]”
ISO 29110 Lifecycle process standard for very small and medium entities	System Lifecycle for people less than 25 people approx.
European Standard ISO 9241 for human computer interaction	Quality requirements that a user friendly software must contain and development and testing process for such software
CMMI (Capability Maturity Model Integration) v1.3 [SEI 2010]	Considers ‘requirement development and requirement management’

Describing requirements Information Model with help of Requirement Landscape

When documenting requirements, we repeatedly encounter a number of basic questions, that have nothing to do with specific content of requirements, but which must be defined at early stages for ex:

- Q. What are different types of requirements that exist and that have to be considered?
- Q. How can requirements be classified according to their solution dependency?
- Q. How should these requirements be documented and presented?
- Q. To what level of detail must requirement be described?

Defining Requirement Landscape:

Requirement Landscape is the specification of

- Classification to be used for the types of requirements
- Classification to be used for independence of requirements from a solution
- Required levels of abstraction (detailing levels) for each type of requirements artefact
- Forms of presentation to be used for each type of requirements artefact

Describing requirements Information Model with help of Requirement Landscape

Basic Principles (Classification of Requirements)

Classifying requirements according to following dimensions

- The Type of Requirement
- Independence of requirement from a solution
- The Level of detail (or abstraction level) of a requirement

“What types of requirements have to be considered during collection and documentation?”

Functional Requirements: Functional Requirements describe the functionality that the planned system should provide. These requirements describe what the planned system should be able to do. Functional Requirement is a requirement concerning a result of behavior that shall be provided by a function of the system.

Quality Requirements: Quality Requirements describe desired qualities of the planned system and thereby influence the system architecture. This class describes for example requirement related to system's performance, reliability, robustness

Constraints: Constraints are organizational, legal, or technical specifications for realisation of the planned system. A constraint is a requirement that limit solution space beyond what is necessary for meeting the given functional requirements and quality requirements.

Wealth of types of requirements that have become established in recent years:

ROSU 2009	Functional Requirements, Technology Requirements, Quality Requirements, requirements for user interface, requirements for other components delivered, requirements for activities, requirements for other components delivered, requirements for activities to be performed, legal contractual requirements
WIBE 2013	Business requirements, business rules, constraints, interface requirements, features, functional requirements, non-functional requirements, quality requirements, system requirements, user requirements
RORO 2014	Functional requirements, Non-Functional requirements, constraints
Youn2014	Business Requirements, User Requirements, Product Requirements, Environmental requirements, System Requirements, Functional requirements, Performance requirements, Interface Requirements etc

Requirement Classification based on the dependence of requirement on a Solution

To consciously differentiate between the requirements with a different solution dependency, classifying requirement artefact dependent on reference to solution or dependence on solution

① Goal Oriented Descriptions (Goals)

Goal-Oriented Description document the intention of the system without addressing the implementation (solution). They are therefore the most abstract form of a documented requirement.

Goal can be described for example, in Natural language in pure text form, with and-or trees, independent notations, regardless of form of presentation. The main goal is to achieve a system understanding to thus recognize the required added value of the planned system.

② Scenario-Oriented Descriptions (Scenarios)

Solution-Oriented Description document the desired process to be supported from the user perspective, system's perspective. Scenario thus describe possible sequences of interactions to fulfil one or more goals. They often supply context required for the requirement describing the process usually covering atomic requirements.

Scenarios can be described using structured templates, in pure text form as a type of story, or based on models using activity diagram, business process models (BPMN), Sequence Diagrams ("Requirement Modelling")

③ Scenario-Oriented Descriptions (Solution-Based Requirements)

Solution-Oriented Description document (using Solution-Based Requirements) document specific requirements. They describe the data, functions, system behavior, statuses.

Solution-Oriented requirements can be described in either natural language as classic textual requirements, or via model-based notations (UML). Solution oriented requirement cover all requirements for the classic system views: data, functions, behavior of the planned system.

Levels of Detail for Requirement-Twin Peak Model

In practice, detailing requirement is only a strict sequential (waterfall Process) that begins with rough requirement artefact and then turn these step by step into requirement artefact at a fine level of detail and in next step are used as basis for the creation of a system architecture.

In real life, at the beginning of the process, there are usually requirements with very different level of detail on the one hand, and on the other hand an early interaction and system architecture which means that there are mutual influences between the system architecture or solution decisions and the requirements

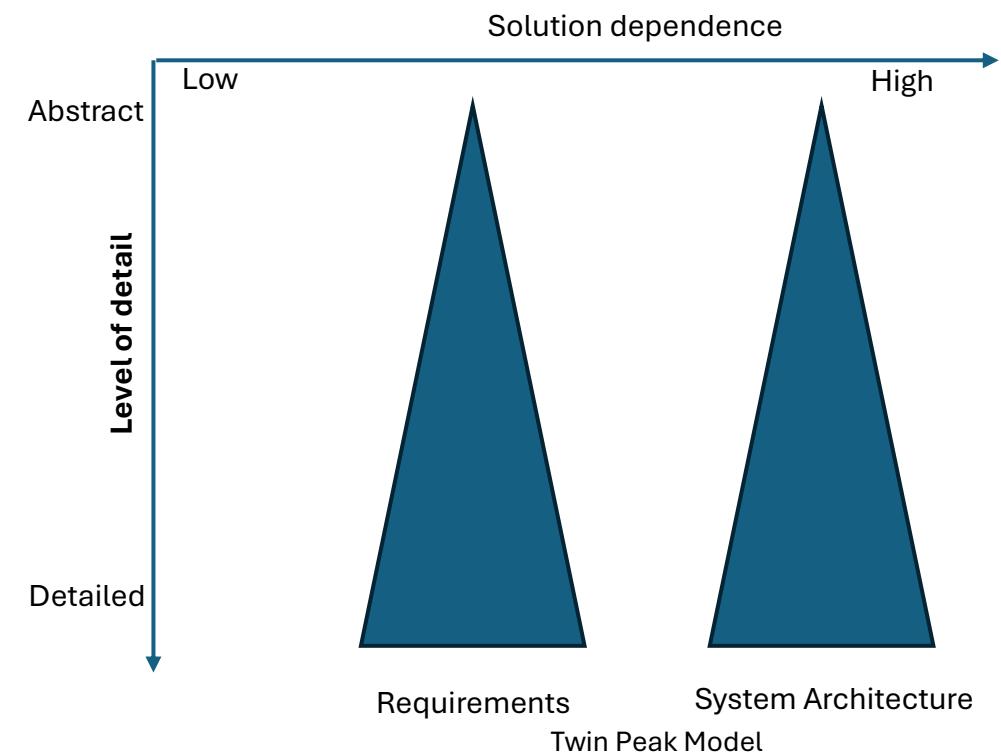
The required level of details depends on many factors:

- ① System Context and Domain
- ② Expertise and Proximity of Stakeholders
- ③ Level of Freedom in implementation

To comply with above factors, requirements should be detailed

Till an extent that

- ① All stakeholders have reached common understanding
- ② Detailed to an accepted residual risk
- ③ Subsequent solution is clearly verifiable



Forms of Presentation for documenting Requirements

The form of document used for documenting requirements, depends on various factors

- ① Purpose of documentation: (e.g formal check, review, description)
- ② Recipient of the Information (e.g Product Manager, architect, tester, developer)
- ③ Classification of the requirement (e.g Use case, Performance requirement)

Different forms of presentation for documentation:

- Textual Presentation of requirement using natural Language.
 - ① Pure Phrase
 - ② Structuring Template
- Model Based Presentation using Modelling Languages
 - ① Unified Modelling Language (UML)
 - ② Business Process Model and Notation (BPMN)
 - ③ Event Driven Process Chain (EPC)
 - ④ System Modelling language (SysML)
 - ⑤ Entity Relationship Model (ERM)
 - ⑥ Petri Nets
- Formalized Representation of Requirement using formal languages
 - ① Mathematical – Algebraic Description
 - ② Set theory forms of description
 - ③ Logical Descriptions and operators

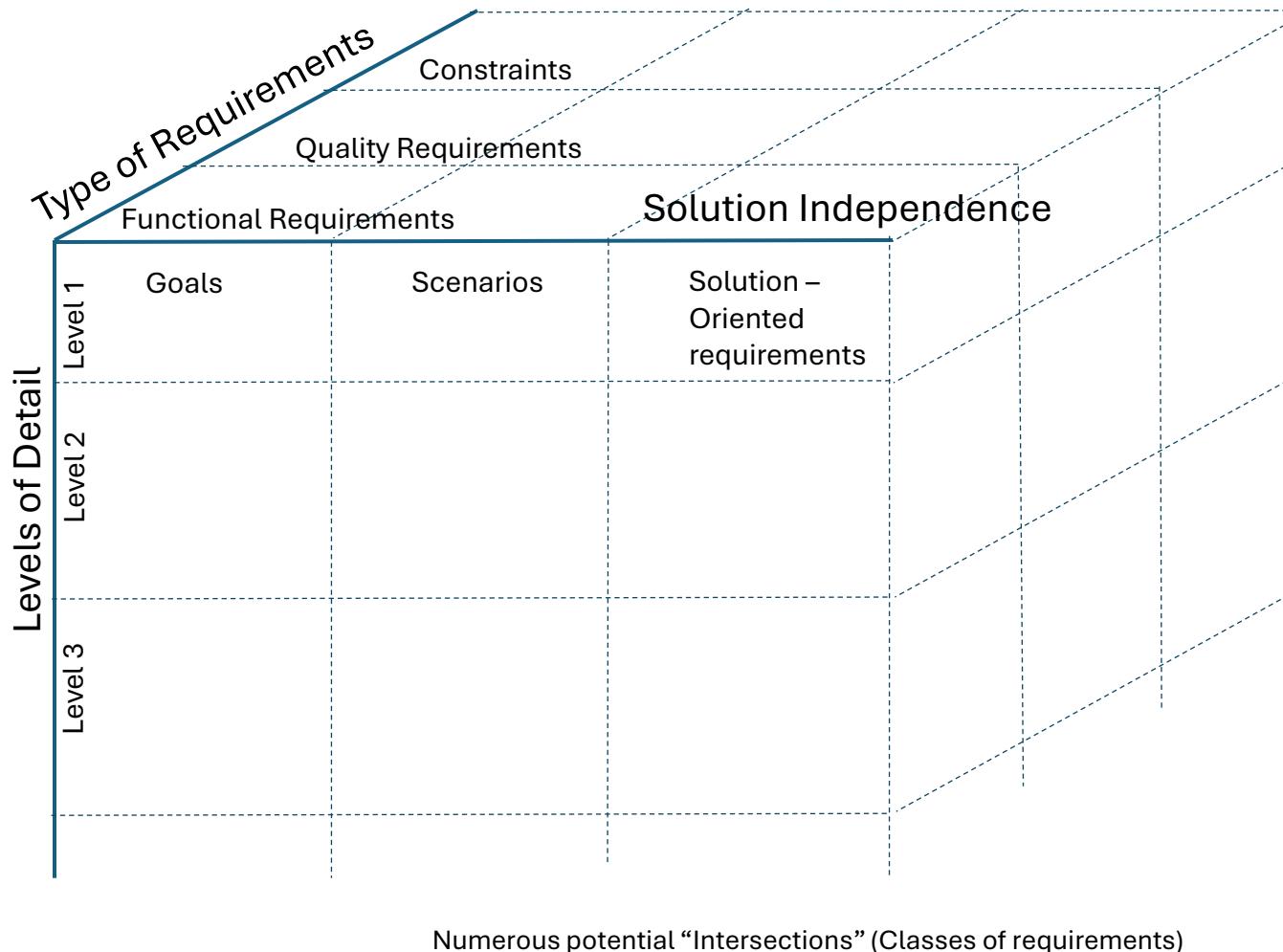
Describing Information Model

Solution Dependency				
Level of Detail	Requirement Type	Low (Goal)	Medium (Scenario)	High (Solution-Based Req.)
Level 1: Business Level	Constraint	Business Goal (Textual)	Not Relevant	Not Relevant
	Quality Requirement	Service Quality (Textual)	Not Relevant	Not Relevant
	Functional Requirement	Not Relevant	Business Process (BPMN)	Business Rule Notation
Level 2: User Rule	Constraint	Usability Goal	Not Relevant	Not Relevant
	Quality Requirement	Not Relevant	User Interface (mock-up)	Not Relevant
	Functional Requirement	Not Relevant	User use case (use case diagrams, templates)	User Requirement (Textual, ER Models)
Level 3: System Level	Constraint	Not Relevant	Not Relevant	Interface guidelines (Textual)
	Quality Requirement	System-Quality Goal (Textual)	Not Relevant	System Quality (Textual)
	Functional Requirement	Not Relevant	Medium, Use Case (MSC, AD)	Interface Requirement (Textual, MSC)

Further aspects can be added in information model

- Which attributes are used for which types of artefacts?
- Which views are supported?
- Which evaluation criteria are planned for requirements?
- Which roles are responsible for maintenance and change?
- Which traceability relationships between requirements artefacts and upstream and downstream are documented?
- How are variant of requirements documented?

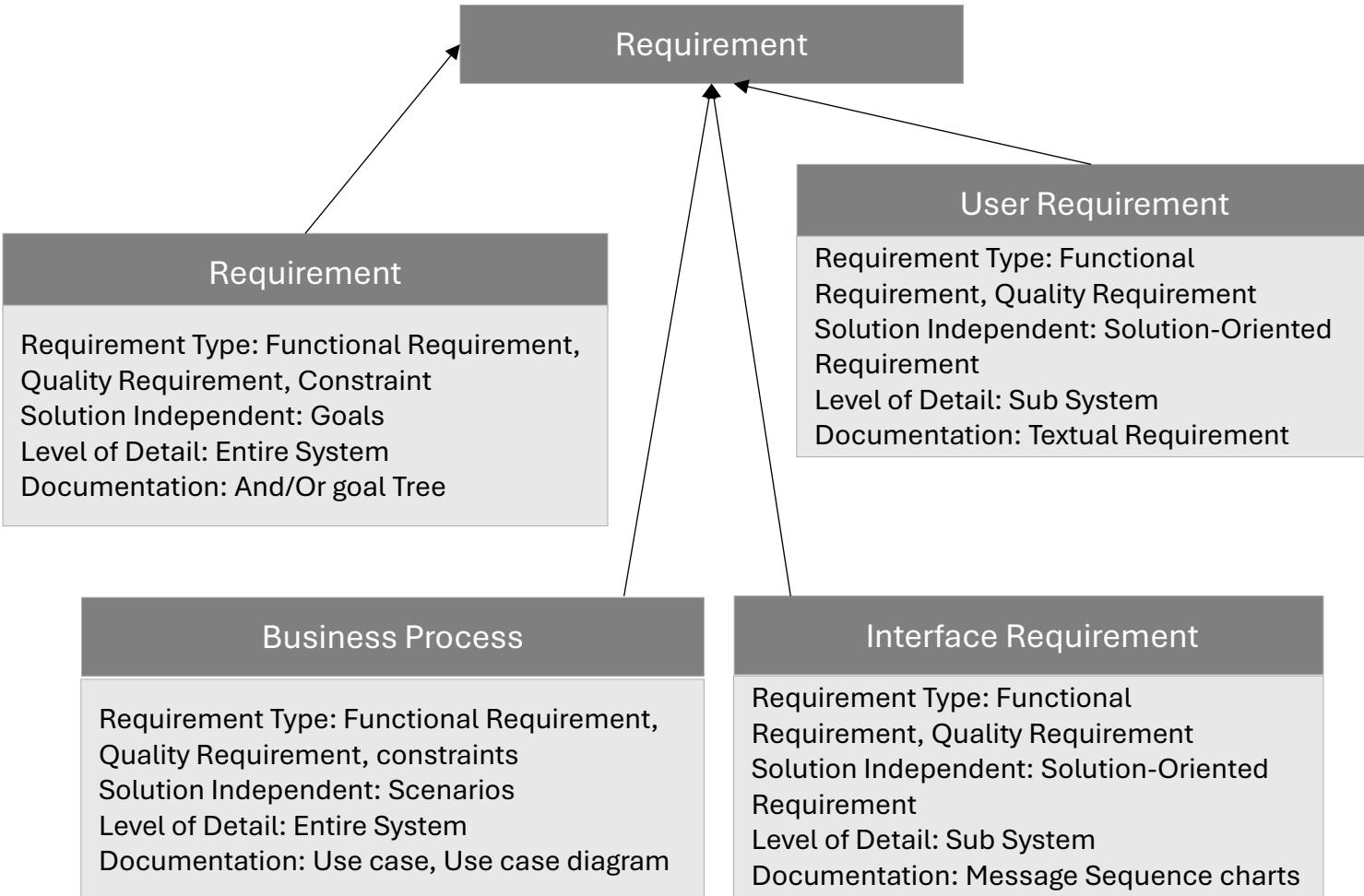
Describing Information Model



Five levels of Specifications

- ① Specification Level 0: roughly describes the overall project and its goals
- ② Specification Level 1: describes the use cases and business processes of the business areas that fulfil the goals (functional specification)
- ③ Specification Level 2: Details of the business processes and business requirements of the business areas at specification level 1 (functional specification)
- ④ Specification Level 3: Describes detailed user requirements with division into sub-systems and a description of interfaces (functional specification)
- ⑤ Specification Level 4: Describes the technical specification with a separation into hardware, and other components

Describing Information Model



Example of a specific Requirement Information Model (RIM)

In addition to the specification of requirements types, independence from solutions, abstraction level and form of representation already defined, the requirements information model should be supplemented by further aspects:

- Which **attributes** will be used where?
- Which **views** will be supported?
- Which **evaluation criteria** are planned for requirement?
- Which **roles** are responsible for maintaining and changing which information?
- Which **traceability relationships** among requirements and upstream and downstream artefacts will be documented?
- How will **variants** of requirements be documented?

Five levels of Specifications

- ① Specification Level 0: roughly describes the overall project and its goals
- ② Specification Level 1: Describes the use cases and business processes of the business areas that fulfil the goals (functional specification)
- ③ Specification Level 2: Details of the business processes and business requirements of the business areas at specification level 1 (functional specification)
- ④ Specification Level 3: Describes detailed user requirements with division into sub-systems and a description of interfaces (functional specification)
- ⑤ Specification Level 4: Describes the technical specification with a separation into hardware, and other components

Module 2:
Requirement Analysis - Management



Wednesday, 14th August



9:00 AM – 10:00 AM

Agenda:

- Describing information model
- Assigning attributes and views for requirements
- Designing a attribute schema
- Change Management for attribute schemas



SAHIL DATERO
Business Analyst, Delivery

Assigning attributes and views for requirements

Definition: Attribute

“An attribute is a characteristic property of unit”

Attribute Type	Meaning
Identifier	Short, unique identification of a requirement artefact in the set of requirements under consideration
Name	Unique characteristic name
Description	Describes the content of the requirement in compact form
Version	Current version of the requirement
Author	Designates the author of the requirement
Source	Designates the sources of the requirement
Justification	Describes why this requirement is important for the planned system
Stability	Designates the probable stability of the requirement here, stability is the scope of changes expected for this requirement in the future; possible differentiation: ‘Stable’; ‘Volatile’
Criticality	In the sense of an estimation of the level of damage and probability of occurrence
Priority	Designates the priority of the requirement with regard to the selected features for prioritization (‘Importance of acceptance on the market’, ‘Order of implementation’, ‘Damage or opportunity costs of non-realization’)
Owner	Designates the person, Stakeholder group, or organizational (unit) responsible for the content of this requirement
Requirement Type	Designates the type of requirement
Status of the content	Designates the current status of the validation – ‘Un-checked’, ‘in-valuation’, ‘checked’, ‘failed’, ‘To be corrected’
Status of agreement	Designates the current status of the agreement – ‘Not Agreed’, ‘Agreed’, ‘Conflicts’
Effort	Forecast/ Actual effort for this requirement
Release	Indicates the degree of legal liability (‘Must’, ‘Recommended’, ‘Optional’)

Some more attributes categories with possible attributes, Identification, Context Relationship, Documentation Aspect, Content Aspects, Agreement Aspects, Validation Aspects, Management Aspects

Assigning attributes and views for requirements

Definition: Attribute Schema

"The set of all defined attributes for a class of requirement."

In Requirement Management, providing an attribute schema (Template-based) for requirement brings the following advantage,

- Accurate and **consistent definition** of the required information: A pre-defined schema defines which information or attributes for requirements must be entered and which values are allowed for this information
- **Gap Detection:** It is possible to detect gaps in the elaboration of requirements if certain attributes are empty
- **Support for employee training:** Employees who have already worked on same attribute schema can quickly find the necessary information and where information on the requirement should be documented.
- Finding the same information in the same place: As all requirements within a project are documented on the basis of the same attribute schema, there is a clear specification where which requirements – such as author – can be found for a requirement

Attribute support a number of requirement management tasks as well as other management tasks:

- **Views**
- **Prioritization**
- Project Management
- Release Management
- Risk Management
- **Traceability**
- Variant Management
- **Reporting**

Designing an attribute schema

To design an attribute schema for use in a specific project we recommend the following steps,

- Identify sources of attributes
- Select the attributes
- Define permitted attribute values and properties of attributes
- Define dependencies between attributes and their values
- Provide dependencies between attributes and their values
- Document the attribute schema

Designing an attribute schema

② Selecting Attributes

For seven categories to be used as attributes for requirements,

- Check which of the proposed attribute has already been selected for the project?
- For each category and attribute proposed, check and evaluate the benefit of attribute for current project
- Weigh up whether a new attribute should be added to reference schema

③ Defining permitted attribute values and properties of attributes

The table shows the attribute schema for our example, bank.

④ Defining Dependencies and their values

- Dependencies like – Stability, Release, Status

⑤ Providing support for recording data

- Support like, Default Value, Description

⑥ Documenting the attribute schema

- Attribute schemas are presented in tabular form or information model

Attribute Type	Example
Identifier	Ex. Project Code + Sequential Number, ex. JMA1001
Name	Free Text
Description	Free Text
Version	The format of versioning is still to be defined
Author	Only following persons are allowed to write requirements: Name Last_Name
Source, Owner	List of all stakeholders
Criticality, Priority, Priority for customer	“Low”, “Medium”, “High”
Requirement Type	“Functional Requirement”, “Constraint”, “Quality Requirement”
Solution Dependency	“Goal”, “Scenario”, “Solution-Oriented Requirement”
Level of Detail	Level 1: Business Scope, Level 2: User Scope, Level 3: IT Scope
Status of the content	‘In-Progress’, ‘in-evaluation’, ‘released’, ‘Changed’, ‘Rejected’, ‘Detailed’, ‘Implemented’, ‘Tested’, ‘Completed’, ‘Un-checked’, ‘in-validation’, ‘checked’, ‘failed’, ‘To be corrected’
Status of agreement	‘Not Agreed’, ‘Agreed’, ‘Conflicts’
Effort	Only whole or half values
Release	“Must”, “Recommended”, “Optional”

Change management for Attribute schemas

Retrospective changes to an attribute schema during course of project should be avoided

- Adding, changing or deleting an attribute
- Adding, changing or deleting possible attribute values (value range)
- Adding, changing, or deleting relationships between attributes
- Changing default values for the attribute types
- Changing binding character of attribute (“Mandatory Field” and “Optional Field”)

① Identifying Sources of Attributes

To select attributes, firstly identify the relevant sources for the attributes. Sources that can be used to select attribute includes:

- An attribute schema from similar project
- A reference schema of the organization or another standard
- Organization rules that determine, for example which attributes must be used in all attribute schema in all projects
- Stakeholders of the requirement engineering process

Goal and Type of Views

Definition: View

A view is a goal-oriented abstraction of the requirements that covers only those requirements and associated information that are relevant for the respective purposes, (e.g stakeholder, decision, requirement). From a Technical perspective, however, a view is a predefined, reusable combination of filter and sorting settings as well as abstractions and aggregations.

The following are examples of views you can create through filtering:

- All released requirements
- All requirements that belong to a specific release
- All requirements that already have been tested
- All requirements for which specific person is responsible
- All requirements that developer has to take in account when implementing a specific component
- Presentation of requirement in order of their criticality
- Distribution of work across the team member

Module 2:
Requirement Analysis - Management



Friday, 16th August



9:00 AM – 10:00 AM

Agenda:

- Motivation for Requirement Traceability
- Classes of traceability relationship types
- Presentation of Traceability Information
- Stakeholder Guidance



SAHIL DATERO
Business Analyst, Delivery

Motivation for requirements traceability

Aspect	Use of Traceability Information
Verifiability and acceptance	Traceability supports the validation that a requirement was considered (correctly and completely) during the implementation of the system, supports acceptance of system since it provides evidence that the stakeholder's requirement have been implemented in the system
Gold Platting	By using traceability information, functions or qualities of the system can be uncovered that were not specified in the requirements. The development and integration of such functions and qualities is referred to as gold plating. In the same way, requirements with no justification for their existence can be uncovered .
Change management	Traceability allows for analysing, in the case of change, which other artefacts (e.g. requirements, components, or test cases) are affected by a change. Traceability provides estimation of efforts required for integrating the change
Quality assurance, maintenance, and repair	Traceability facilitates the identification of the causes and the impact of errors, and identification of parts of the system affected by an error and the prognosis of the effort required for correcting the error
Re-Engineering	Traceability supports Re-Engineering of legacy systems by relating the functions of legacy system to the requirements for the new system and by documenting which components of the new system realise these requirement
Reuse	Traceability supports the reuse of development artefacts related to a requirement. When a (set of) requirement(s) is reused in the new system, the corresponding artefacts which realise this requirement(s) in the old system can be identified. These artefacts are potential reuse candidates and can be checked for whether they require adaptation or if they can be reused without adaptation.
Project Traceability	Traceability information supports tracing the project and the current project status.
Risk management	Traceability between requirements and other artefacts (e.g components) supports risk management by facilitating the identification of development artefacts that are potentially affected by a risk or threat
Accountability	Traceability information can be used to assign development effort to individual requirements.
Process improvement	Traceability supports process improvement. Traceability information can be used to trace problems in the development process back to their causes.

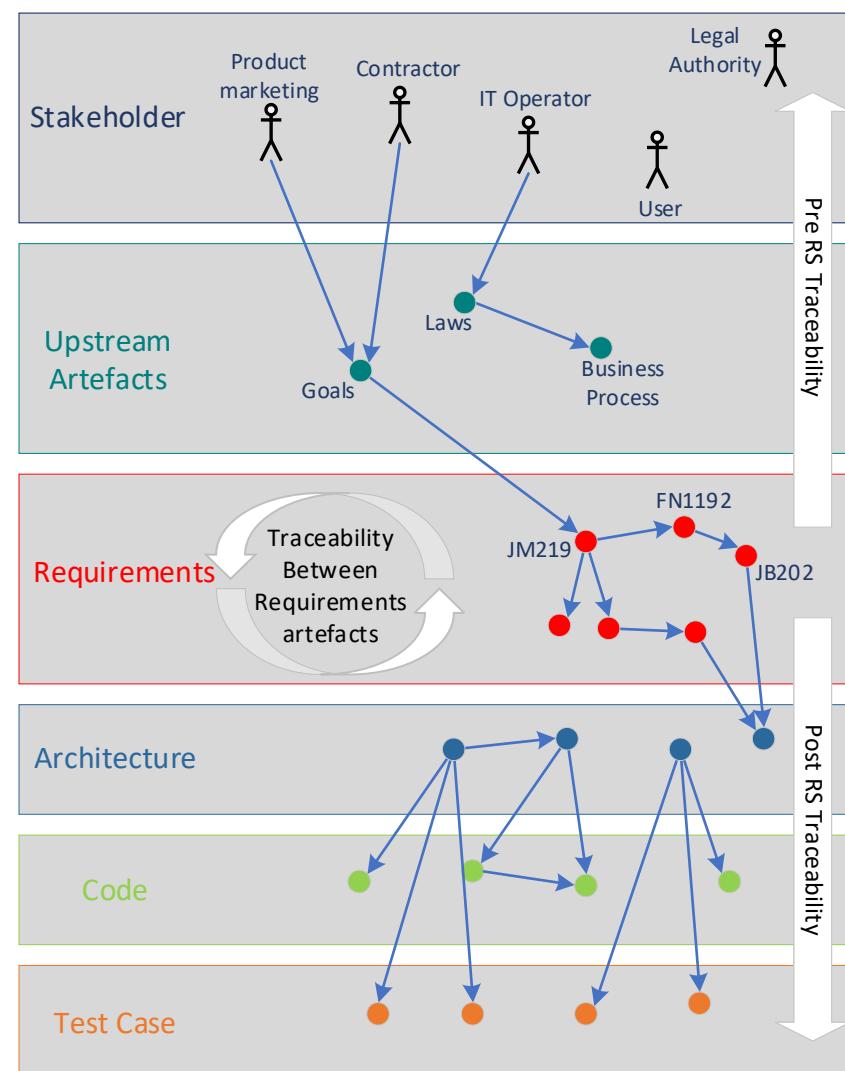
Definition: Requirement Traceability
 "Requirement Traceability refers to the ability to describe and follow the life of a requirement, in both a forwards and backwards direction (i.e from its origins through its development and specification, to its subsequent deployment and use, and through all periods of on-going refinement and iteration in any of these phases" - [Gotel and Finkelstein 1994]

Impact analysis, source analysis, coverage analysis, earned value analysis

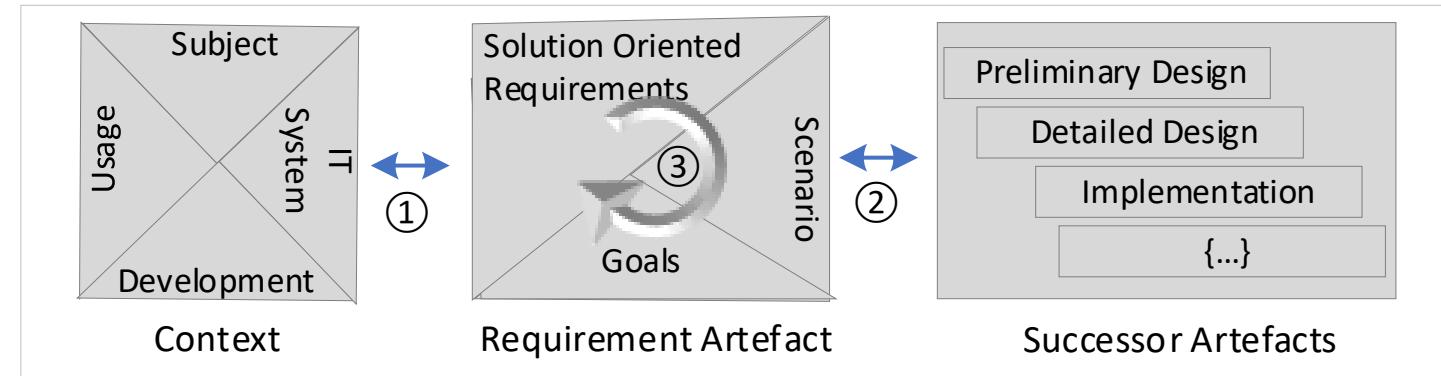
Different Traceability Views

Requirements traceability can essentially be distinguished by the following dimensions

- Traceability among requirements at same level of detail:
This type of traceability describes for example, content-related dependencies between functional requirements
- Traceability among requirements at different level of detail:
This type of traceability describes, for example, the detailing of legal requirements into system requirements
- Traceability between versions of requirements:
This type describes the traceability of the evolution of a requirement over time. A particularity of this view is that there is only one valid version at a given time
- Traceability among requirements and downstream development artefacts:
This type of traceability describes, for example, dependencies that document the implementation / realization of a requirement as a system component or test case.
- Traceability among requirements and upstream artefacts:
This type of traceability describes, justification or source of a requirement
- Pre-Requirement Specification Traceability:
Is traceability of requirement to their origin. For example to the upstream goals and visions or other sources of requirements from the system context, or existing documents and stakeholders.
- Post-Requirement Specification Traceability:
Is traceability of requirement to subsequent development artefacts, such as architectural design, implementation, test cases



Different Traceability Views



① Pre - Traceability of requirements:

Comprises relationships from requirement artefacts (i.e goals, scenarios, and solution-oriented requirements) to aspects in four context facets

② Post - Traceability of requirements:

Comprises relationships between the requirement artefacts and their successor artefacts

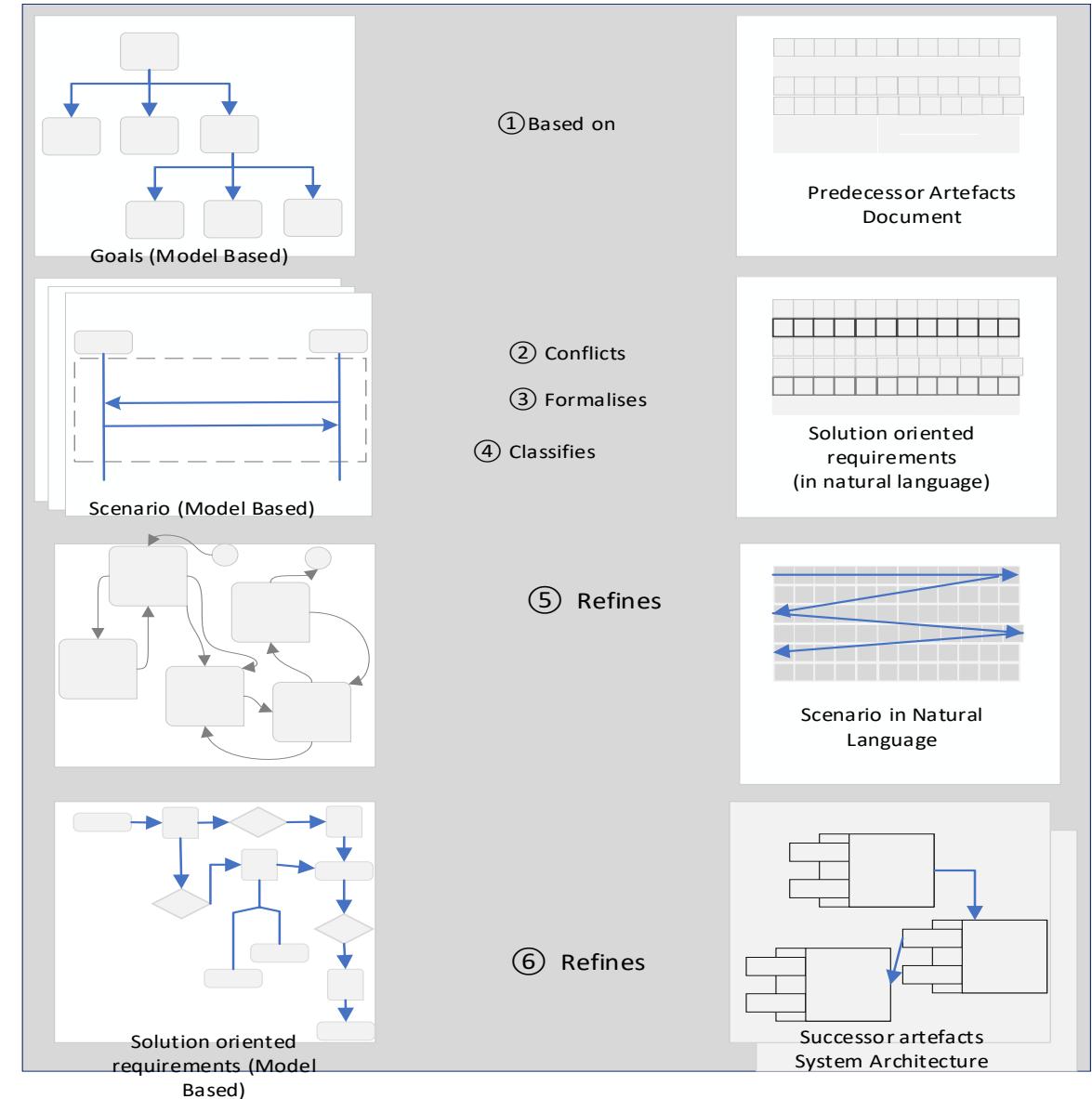
③ Traceability among different requirements artefacts:

Comprises relationships between requirement artefacts as well as traceability between requirements and design artefacts

Traceability Relationship Types

In the following we focus on traceability of requirement artefacts and distinguish five classes of traceability relationship types

- Condition
 - Constraint
 - Precondition
- Content
 - Similar
 - Compares
 - Conflicts
- Abstraction
 - Classifies
 - Aggregates
 - Generalises
- Evolution
 - Replaces
 - Satisfies
 - Based on
 - Formalises
 - Refines
 - Derived
- Miscellaneous
 - Example of
 - Verifies
 - Rationale
 - Responsible for
 - Background
 - Comment



Presentation of Traceability information

I . Text-Based References

Test Case	Reference to Requirement	Test Case Description	Priority

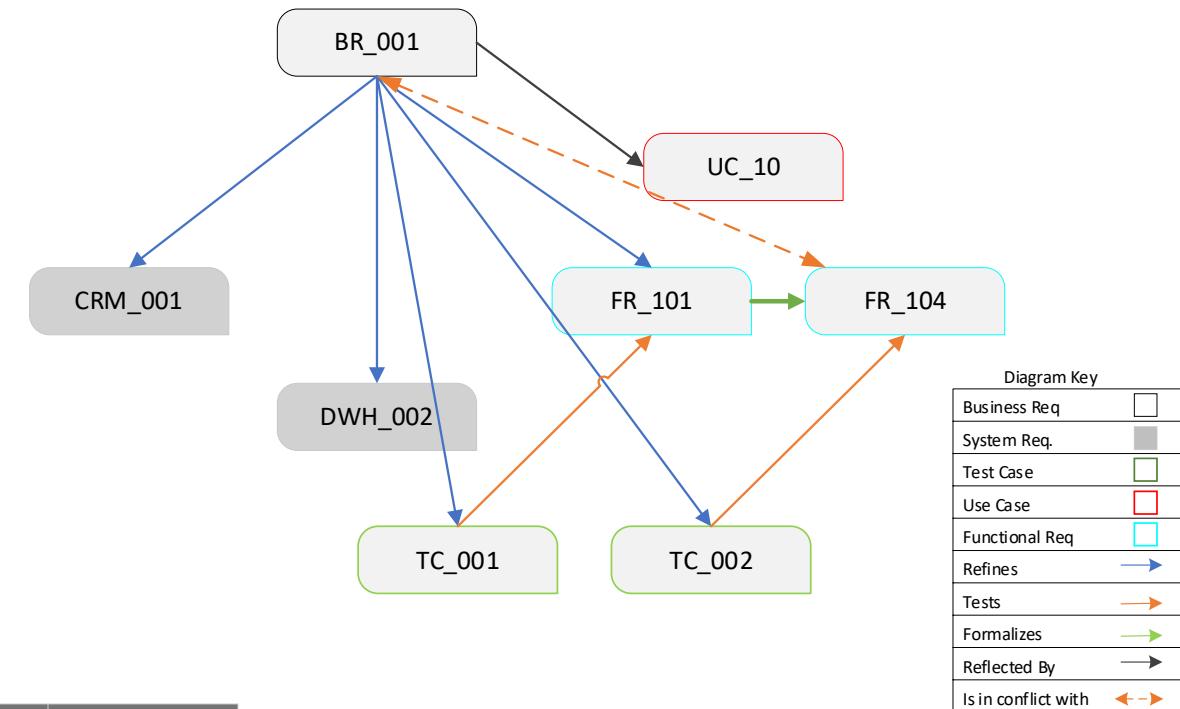
II . Traceability Matrices

Source Artefact	Target Artefact		
	UC_10	UC_101	UC_20
BR_01	Satisfies		
BR_01		Conflicts	

III. Traceability Tables

Business Requirement	Use Case	Functional Requirement	System Specification	Gui Element	Test Case
BR_01	UC_10	FR_101	CRM_001	GUI_001	TC_001
BR_02	UC_11	FR_104	DWH_002		TC_002

III. Traceability Graphs



Traceability model - Example

Requirement Identification	Requirement Description	Priority	...
FR_3131	<p>"If mixed fuel temperature < 265 ° F, all fuel shall be burned using afterburner, if > 265 ° F flow of hot fuel should be rerouted to tank 4, if tank 4 is full, fuel shall be rerouted to tank 5"</p> <p>...is tested by TC_0021</p> <p>...is formulated by FR_0014</p>	High	

Fig: Traceability by means of textual references in the requirement text

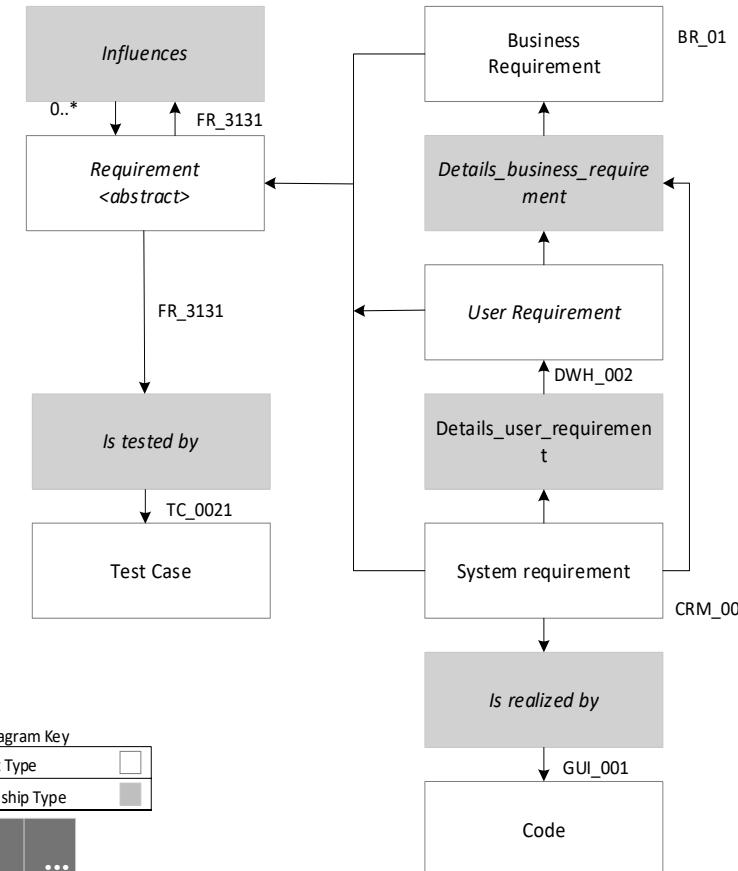
Reference to Test Case	Reference to Requirement	Requirement Identification	Requirement Description	Priority	...
TC_0021	FR_0021	FR_3131	<p>"If mixed fuel temperature < 265 ° F, all fuel shall be burned using afterburner, if > 265 ° F flow of hot fuel should be rerouted to tank 4, if tank 4 is full, fuel shall be rerouted to tank 5"</p>	High	

Fig: Traceability by means of textual references with separate attribute

Reference to Test Case	Reference to Requirement	Requirement Identification	Requirement Description	Priority	...		
TC_0021	FR_0021		"If mixed fuel temperature < 265 ° F, all fuel shall be burned using afterburner, if > 265 ° F flow of hot fuel should be rerouted to tank 4, if tank 4 is full, fuel shall be rerouted to tank 5"				
			Reference to Requirement	Test Case Identification	Requirement Description	Priority	...
			FR_3131	TC_0021	When velocity > 60 km/hr energy should be supplied from fuel	Medium	

Completes...

Fig: Traceability via hyperlinks



Module 2:
Requirement Analysis - Management



Thursday, 19th August



9:00 AM – 10:00 AM

Agenda:

- Principle of Requirement Prioritisation
- Ad-Hoc prioritisation techniques
 - One – Criterion Classification
 - Ranking and Top-Ten technique
 - Kano Classification



SAHIL DATERO
Business Analyst, Delivery

Is it ok not to use every attribute?

For optional attributes whose entry is not mandatory, following conclusions can be drawn.

- The attribute was not used in either view or a report:

This indicates that the attribute in question does not support a specific goal and is probably not of interest to any stakeholders

- The attribute is always populated with the same value:

In this case, there does not seem to be any real distinction between the different requirements in relation to this attribute, which also means that the proposed list of values for selection is not suitable.

- The attribute is never populated:

The reason for attributes not being populated often lies in the fact that users are not aware of the definition or the benefit of the attribute, or do not directly see any benefit.

- The attribute is only filled for a few requirements:

If an attribute is important it can be declared as mandatory field, which forces entry in the future.

- The attribute is not populated in individual cases:

It must first be determined whether this attribute is still relevant for the project. If yes Business Analyst should complete relevant requirements.

Attribute Type
Identifier
Name
Description
Version
Author
Source, Owner
Criticality, Priority, Priority for customer
Requirement Type
Solution Dependency
Level of Detail
Status of the content
Status of agreement
Effort
Release

Principles of evaluation

In all activities of requirement engineering, requirements are evaluated on the basis of various criteria.

For example, during elicitation, requirements are categorized according to Kano criteria, during documentation requirements are **evaluated** according to criticality because of guidelines and standards to be followed.

The task of requirements management is to document these evaluation results in an appropriate form and to foresee the **consequence of the evaluation** for the Business analysis Process

A requirement that has been evaluated as particularly safety-criticality could for example, be subjected to more detailed quality control during testing than a requirement that is considered non-critical

Possible evaluation criteria are for example:

- The legally – binding nature of a requirement
- Implementation effort/costs
- Criticality
- Stability
- Degree of Motivation

Sources for evaluation criteria are e.g:

- Project Management Standards
- Guidelines and Standards
- The requirements attribute schema
- The sub-sequent development disciplines

Prioritizing Requirements

In principle, prioritization should always follow the following process:

For example, during elicitation, requirements are categorized according to **Kano criteria**, during documentation requirements are evaluated according to **criticality** because of guidelines and standards to be followed.

- Determining the goals of prioritization
- Determining the prioritization criteria (importance, cost, damage, duration, risk, volatility)
- Determining the stakeholders involved in prioritization
- Determining the requirements to be prioritized
- Selecting the prioritization technique
- Adjusting the attribute schema, if necessary
- Performing the prioritization
- Regular checking and if necessary, reprioritisation of requirements

First step is to determine which decisions are to be made on the basis of prioritization results. Thus, when deciding which functions of the system should be specified in detail first, other prioritization criteria must be considered as compared to when determining the order of implementation.

Depending on the pursued goal, it is necessary to determine which evaluation or prioritization criterion, or which combination of criteria, is used to determine the requirement's priority.

When selecting requirements to be prioritized, it should be noted that the requirements should at the same level of detail to avoid distorting the result of the prioritization.

Ad Hoc Prioritization Techniques

The following ad-hoc prioritization techniques work well in practice

"In projects, Ad-Hoc prioritization techniques are a pragmatic approach for effective prioritization of requirements. [Pohl 2020]

Number of complexity of requirements	Large number of requirements/ high complexity of prioritisation					
	Low number of requirements / high complexity of prioritisation					
	Large number of requirements / low complexity of prioritisation					
	Low number of requirements/ low complexity of prioritisation					
Techniques	Effort					
Ranking Top Ten	Low	***	***	**	*	
One-Criterion Classification	Very Low	***	***	**	*	
Kano Classification	Low	**	**	***	**	
Wiegers' prioritisation Matrix	Medium	*	*	***	**	
Cost-Value Approach	High	*	*	***	**	

① Ranking and Top-Ten Technique

- Ad hoc ranking: When ad hoc ranking is applied, the requirements are ranked by individual stakeholders or by a group of stakeholder with regard to a chosen criterion such as risk
- Top-Ten Technique: When the top-ten technique is applied a fixed number of requirements artefact is selected (Typically ten) as top-priority requirements artefacts. The selection is determined by a single selection criterion. Then, the selected requirements are ranked according to some criterion, for example the importance of the requirements for the success of the system

② One-Criterion Classification

A one-criterion classification prioritises artefacts based on a single criterion. A common criterion for a single-critical classification is degree of necessity of a requirement artefact.

[IEEE Std 830-1998]). [IEEE Std 830-1998] suggests the following three priority classes for prioritisation with respect to the degree of necessity of a requirements artefacts

- Essential: Implies that the system will be acceptable unless these requirements are fulfilled in an agreed manner
- Conditional: Implies that these requirements artefacts would enhance the system, but would not make it unacceptable if they are absent
- Optional: Implies a class of requirement artefacts that may or may not be worthwhile

③ Kano Classification

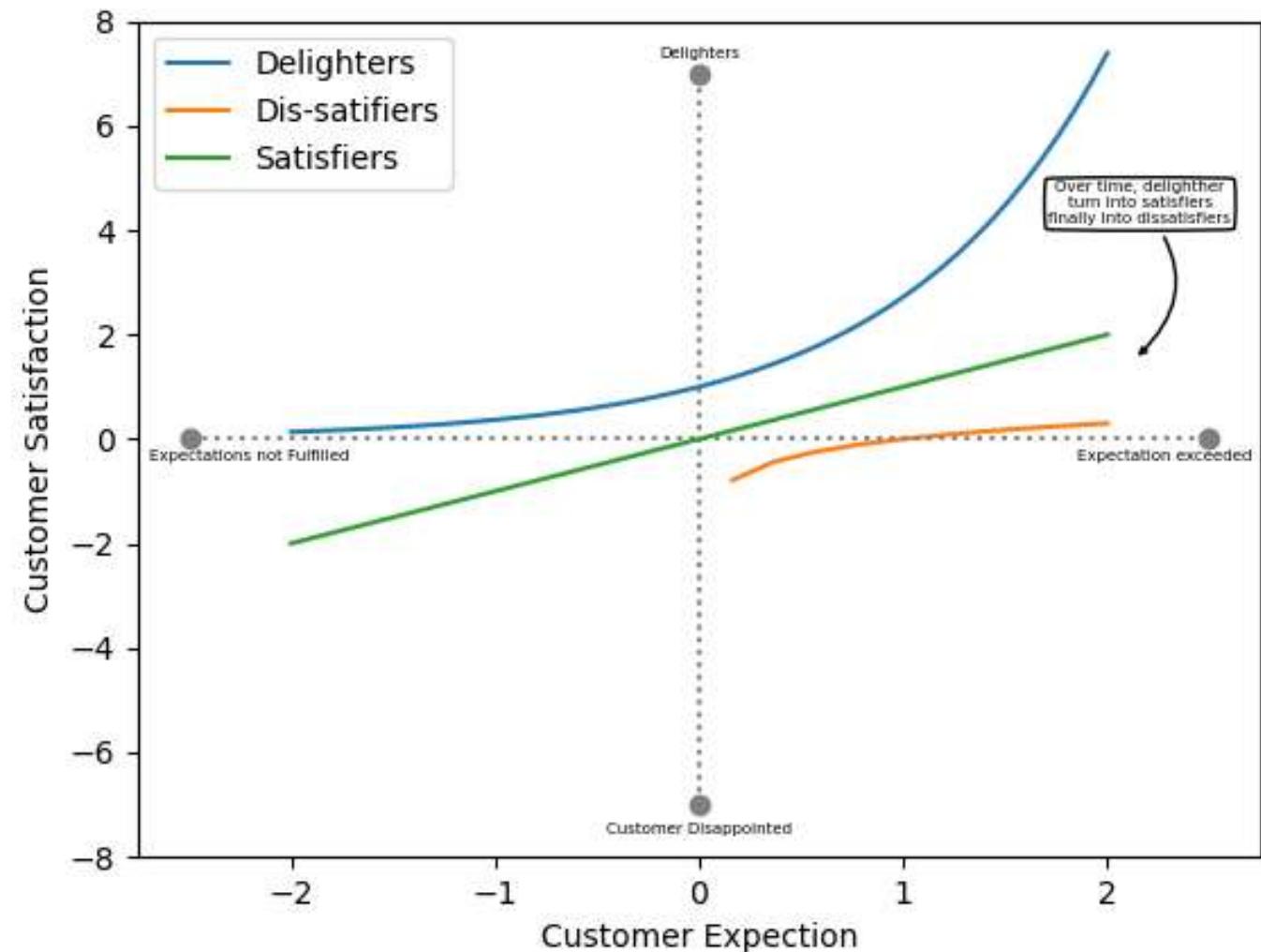
The starting point of the Kano classification is the classification of system features (or requirements artefacts) into three classes [Walden 1993]

- Dissatisfier (also called a must-be requirement) A requirement is a dissatisfier if the system must realise this requirement to enable market entry
- Satisfier (also called a one-dimensional customer requirement): A requirement is a satisfier if the customers consciously demand the realisation of this requirement in the system. Satisfiers positively influence the degree of customer satisfaction. i.e an increased amount of realised satisfiers generally results in increased customer satisfaction.
- Delighter (also called an attractive requirement): A requirement is a delighter if the customers are not aware of this requirement or if they do not expect the realisation of the requirement in the system. Customer satisfaction increases disproportionately, if the system realises such a requirement.

③ Kano Classification

Steps:

- 1) Identify a set of system features to be classified
- 2) Create a questionnaire containing a question for each system feature to determine how a potential customer would feel if
 - a) the feature is realised in the system (functional question)
 - b) If the feature is not realised in the system (dysfunctional function)
- 3) Analyse the answers to the questionnaire and calculate the average values
- 4) Identify the feature class of each system feature



④ Wieger's Prioritisation Matrix

Step 1: Determine the weights of the four calculation parameters of benefit, penalty cost and risk.

Step 2: List all requirements to be prioritised in the prioritisation matrix

Step 3: Estimate the relative benefit of each requirement with regard to customer satisfaction or achievement of business objective.
Scale (from 1 (lowest) to 9 (highest))

Step 4: Estimate for each requirement penalty that would occur if the requirement were not realised in system Scale (from 1 (lowest penalty) to 9 (highest penalty))

Step 5: Calculate for each requirement, value of R_i

$$\text{Value } (R_i) = \sum_{i=1}^n \text{Benefit } (R_i) * \text{WeightBenefit} + \text{Penalty}(R_i) * \text{WeightPenalty}$$

Step 6: Estimate for each requirement relative cost related to realisation of requirement. Scale (1 (lowest) to 9 (high))

Step 7: Estimate for each requirement relative risk related to realisation of requirement. Scale (1 (lowest) to 9 (high))

Step 8: Calculate individual priorities based on predicted and calculated values using,

$$\text{Priority } (R_i) = \frac{\text{Value \% } (R_i)}{\text{Cost\% } (R_i) * \text{WeightCost} + \text{Risk\%}(R_i) * \text{WeightRisk}}$$

Step 9: Rank the requirements in descending order

④ Wieger's Prioritisation Matrix

Estimated values

Relative Weight		2 (Weight Benefit)		1 (Weight Penalty)		1 (Weight cost)		0.5 (weight risk)			
Requirement	Relative benefit	Relative Penalty	Total	Value%	Relative Cost	Cost%	Relative Risk	Risk %	Priority	Rank	
R1	5	3	13	16.8	2	13.3	1	9.1	0.941	1	
R2	9	7	25	32.5	5	33.3	3	27.2	0.692	3	
R3	5	7	17	22.1	3	20	2	18.2	0.759	2	
R4	2	1	5	6.5	1	6.7	1	9.1	0.577	4	
R5	4	9	17	22.1	4	26.7	4	36.4	0.489	5	
Total	25	27	77	100	15	100	11	100			

$$\text{Value } (R_i) = \sum_{i=1}^n \text{Benefit } (R_i) * \text{WeightBenefit} + \text{Penalty}(R_i) * \text{WeightPenalty}$$

$$\text{Priority } (R_i) = \frac{\text{Value } \% (R_i)}{\text{Cost\% } (R_i) * \text{WeightCost} + \text{Risk\%}(R_i) * \text{WeightRisk}}$$

⑤ Cost-Value Approach

Step 1: Business Analyst review the candidate requirements to ensure that requirements are complete and clearly defined

Step 2: Customers and users determine the relative value of each requirement using pairwise comparison

Step 3: The relative cost of implementing each requirement is estimated by experienced software engineers using the pairwise comparison method

Step 4: The relative value and cost of each requirement is calculated, a cost-value matrix is created
Value on y axis and cost on X axis

Step 5: The cost-value diagram is used by the stakeholders as conceptual map for analysing and discussing the requirements

Relative Intensity	Explanation
1	If equal value – two requirements are of equal value
3	Slightly more value – experience slightly favours one requirement over another
5	Essential or strong value – Experience strongly favours one requirement over another
7	Very Strong value – A requirement is strongly favoured and its dominance is demonstrated in practice
9	Extreme value – The evidence favouring one over another is of the highest possible order of affirmation
2,4,6,8	Intermediate values between two adjacent judgements – when compromised is needed

⑤ Cost-Value Approach

Relative Intensity	Explanation
1	If equal value – two requirements are of equal value
3	Slightly more value – experience slightly favours one requirement over another
5	Essential or strong value – Experience strongly favours one requirement over another
7	Very Strong value – A requirement is strongly favoured and its dominance is demonstrated in practice
9	Extreme value – The evidence favouring one over another is of the highest possible order of affirmation
2,4,6,8	Intermediate values between two adjacent judgements – when compromised is needed

	R_1	R_2	R_3	Sum	Estimated priority vector
R_1	1	1/4	5	0.9	0.3
R_2	4	1	3	1.73	0.58
R_3	1/5	1/3	1	0.36	0.12

Requirement	Cost(%)	Value(%)
R_1	6	17
R_2	13	6
R_3	9	26
R_4	6	4
R_5	4	7
R_6	16	23
R_7	24	2
R_8	22	15

Module 2:
Requirement Analysis - Management



Friday, 20th August



9:00 AM – 10:00 AM

Agenda:

- Systematic approach to Requirement Prioritisation
- Weiger's Classification
- Versioning of requirements
- Introduction to Change management



SAHIL DATERO
Business Analyst, Delivery

Change Management – Educational Objectives

- Knowing Version Control Activities
- Knowing the characteristics of Requirements Configuration
- Knowing the development tasks supported by Requirement Baselines
- Knowing the necessity and disadvantages of Requirement Branching
- Knowing the main reason for requirement changes
- Knowing Types of requirement changes
- Knowing Heuristics for evaluating the stability/ volatility of Requirements
- Knowing the Goal and Tasks of a Change Control Board
- Mastering and using the Change Management Process

Versioning: Tracking Requirements Throughout its Lifecycle

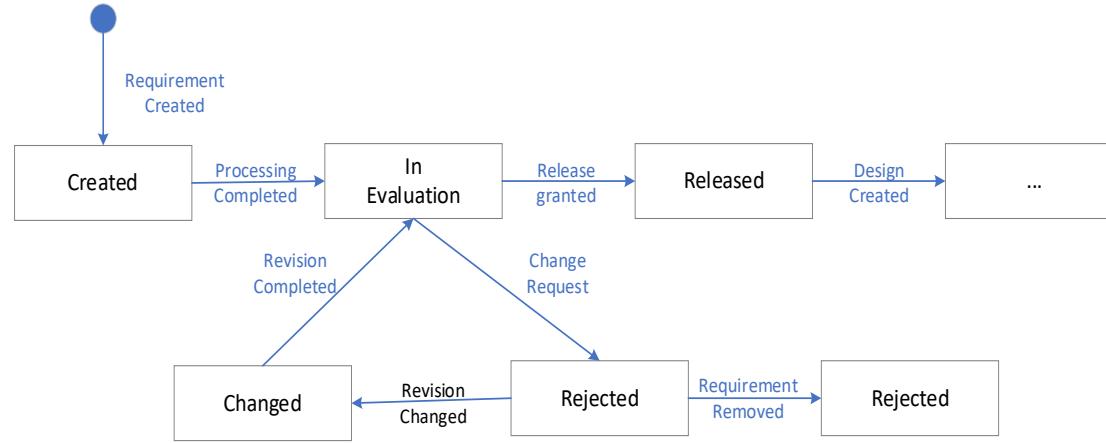


Fig: Statuses and status transitions of a requirement

Versioning can place at different levels

- In the case of versioning at the document level, every change to the content of a **document** (e.g., a change to one or more requirement within the document) must lead to new version
- In the case of versioning at the requirement level, every change to the content of **requirement artefact** must lead to new version

Definition: "Version Control"

Version Control (or a version control system) is used to document, manage, and restore documents, files, and individual artefacts. Version Control allows you to trace changes to documents and artefacts and to revise changes made so that you can return to old versions. Version Control therefore enables a sequential consideration of the evolution of a document or artefact over its entire life.

Definition: "Status"

According to [RuSo 2009]: "**Statuses** specify the progress, of the requirement. If we compare the life of a requirement with a project plan, then the statuses of the requirement often correspond to the milestones in the project plan."

Definition: "Version"

A **version** is a specific content status of a requirements artefact or document at a specific point in time. Versioning allows you to trace the history of a requirements artefact or document back without any gaps and reset it to an earlier version. Changes to content always lead to new version.

Making Requirements Traceable, Versioning Requirement Documents and Artefacts

To **document** a change, it is important that a document has document history so performed changes can be recognized at a glance. The document history should always contain at least the following information:

- The new version number of the document
- The date on which the change was performed
- The person who made the change
- The changes that were made
- The reason for the change

When you **change requirements** (i.e you create new requirement version). At the minimum the following information must be documented to describe the change compared to the previous version:

- The new version number of the requirement (whole number or increment)
- The change action performed compared to last baseline (e.g deletion)
- The change made to the content of the requirement
- The reason for the change (i.e., what or who was decisive for the change)
- The name and role of the person who performed the change
- The time of the change (date + time)

Facilitating Performance measures, Requirement Configuration

According to [PoRu2015], requirements configuration have the following properties:

- Logical connection:** The selected requirements version of a configuration are connected logically and are selected for a specified purpose.
- Consistency:** The combined requirements and requirements documents are consistent and belong together logically
- Uniqueness:** The configuration for the selected requirements versions has an identifier that identifies it uniquely
- Unchangeability:** The configuration is based on specific version status of the requirements. Changes to these requirement versions lead to new versions that can be used in new configurations.
- Basis for reset:** Configuration offer defined statuses to allow requirements to be reset to an older, consistent requirement status (version status)

Definition: "Requirement Configuration"

A **requirement configuration** comprises a defined set of logically related requirements, whereby at most one version of each requirement is contained in the requirements configuration

Definition: "Configuration Management"

Configuration Management, from [ISO29148], The purpose of Configuration Management Process is to establish and maintain integrity of all identified outputs of a project or process and make them available to concerned parties.

Frozen Documentation Status, Requirement Baseline

Requirements baselines support three essential activities in the development process

- They form the basis for planning delivery increments (releases) because for customers, they represent a visible configuration of stable requirements versions.
 - They are used to **estimate the implementation costs** of a particular release.
 - They enable a **comparison with competing products** on the market with the defined release

Definition: “Requirement Baseline”

Requirement baselines are selected, formally checked, and released requirements configurations that cover stable requirements artefacts and often reflect a fixed development and delivery status for a product. (ex. for a specific product release)

Definition: “Release Management”

Release Management, is concerned with bundling requirements, for a product, with the scheduling for the manufacture, and ultimately, the delivery of a finished system, [...] For a release, all current configuration elements are usually managed under one common label and the software is then created from the configuration thus created.

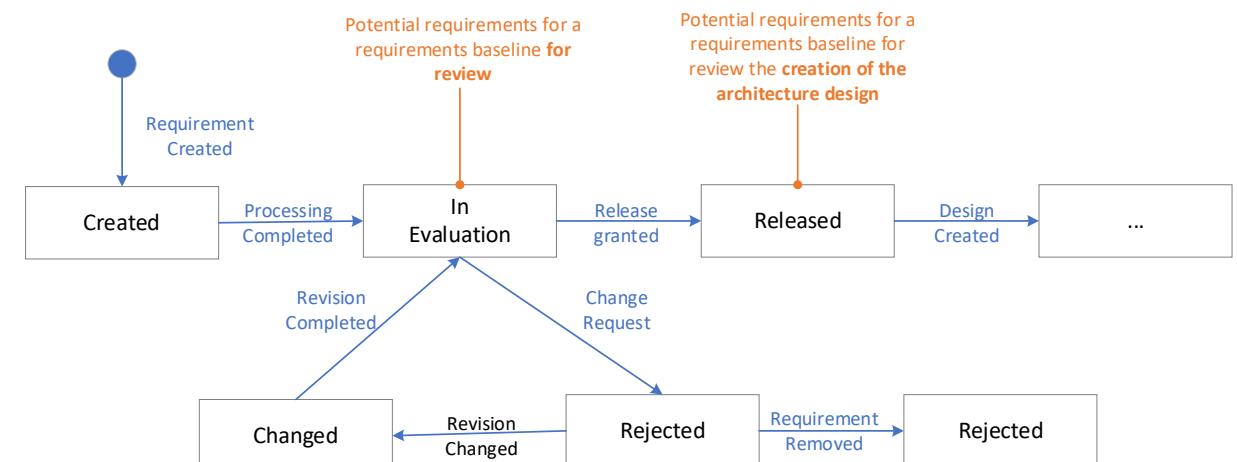


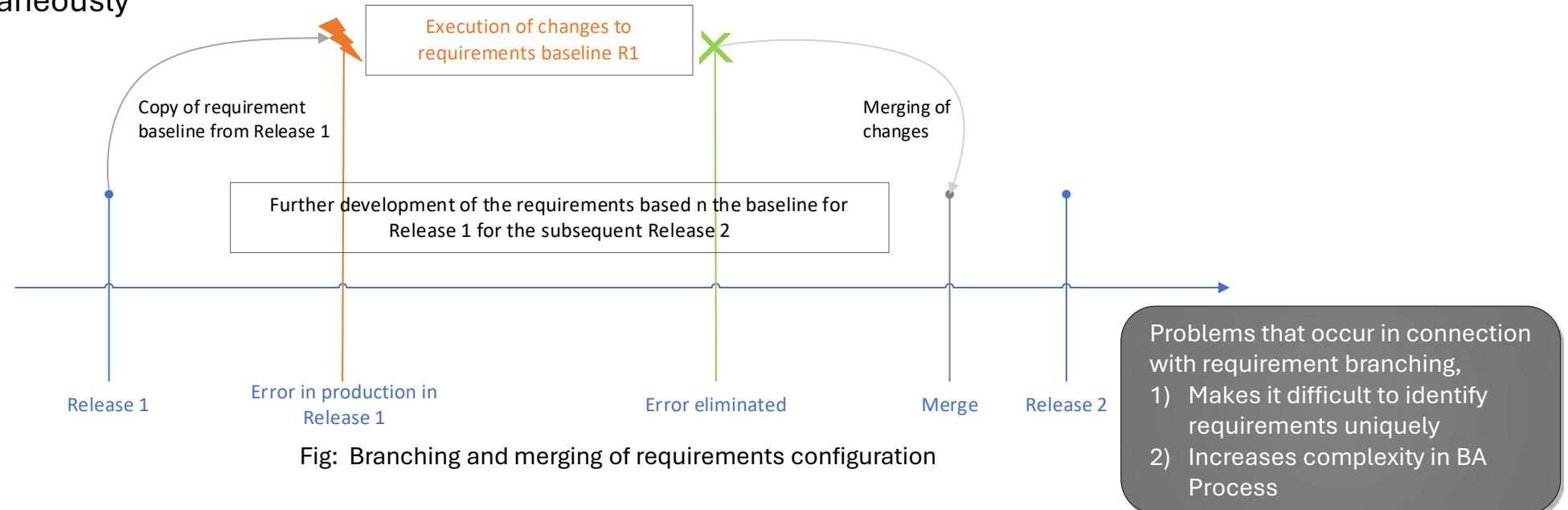
Fig: Possible milestones for requirement configurations and requirements baselines

Enabling parallel development, Branching Requirements

The term “Branching” originates from configuration management and allows the **parallel development** of systems in different development Branches.

The Two development branches are then generally **merged** again before the next main release so that, for example, the errors that have already been corrected in the production branch do find their way back into production with the new release

In contrast to versioning requirements, branching requirements allows **Multiple versions** of requirements to be **valid** in parallel simultaneously



Change management for Requirements

The changes in requirements originate in following sources:

- Incident Management:** They form the basis for planning delivery increments (releases) because for customers, they represent a visible configuration of stable requirement versions.
- Department and product management:** These group of people usually create new requirements for the system which improve the use of the system or reflect new facets of the system.
- Developers:** This group of people generally defines change requests relating to the technical implementations of the system. These changes do not directly influence the user functionality of a system.
- Testers:** These people generally defines changes at rectifying errors that exist in the system (due to faulty or incomplete requirements)
- Errors in ongoing system operation:** Changes due to incorrect system behavior that are reported as an incident by the user or application operation. These changes to requirements result from incorrect or missing requirements, and not from an incorrect implementation of the requirement
- Context Diagrams:** Changes that result from changes to constraints in the system context.

Definition: "Change Management"

Change Management regulates the further development of the product by monitoring change requests for the product and processes of this change request across following steps: Creation, evaluation, realization, testing, and acceptance

Module 2:
Requirement Analysis - Management



Wednesday, 21th August



9:00 AM – 10:00 AM

Agenda:

Understanding of,

- Responsibilities Change management board
- Documenting Change Requests
- Classification of Change Requests

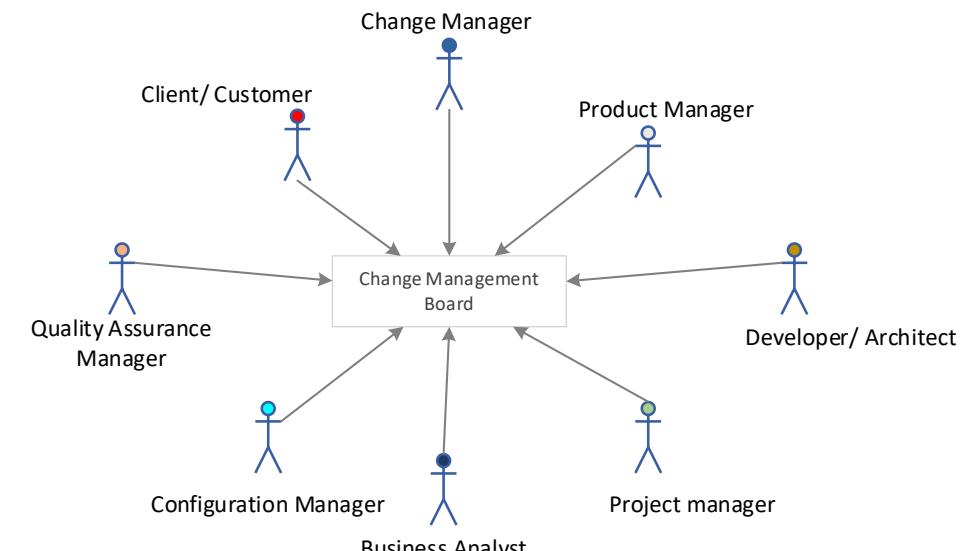


SAHIL DATERO
Business Analyst, Delivery

Quorum for Change management board.

The changes control board typically has the following responsibilities [Wiegers 1999]

- Classification of incoming change requests:** The change control board analyses each incoming change request and, based on the analysis, assigns the changes in different categories (e.g. corrective, adaptive, exceptional change)
- Effort estimation for change integration:** The change control board estimates the effort required for integrating the change required for integrating the change by conducting an impact analysis, often assigned to 3rd Party
- Evaluation of change requests and decision making:** The change control board evaluates the change requests, ex. efforts, benefits, accept - reject
- Prioritisation of accepted change request:** The change control board prioritises the change requests, that have been accepted and assigns each request to system release



Possible Composition of change control board

Supporting Decision making of Change Management Board - Documenting Change Requests

Template for the documentation of a Change Requests

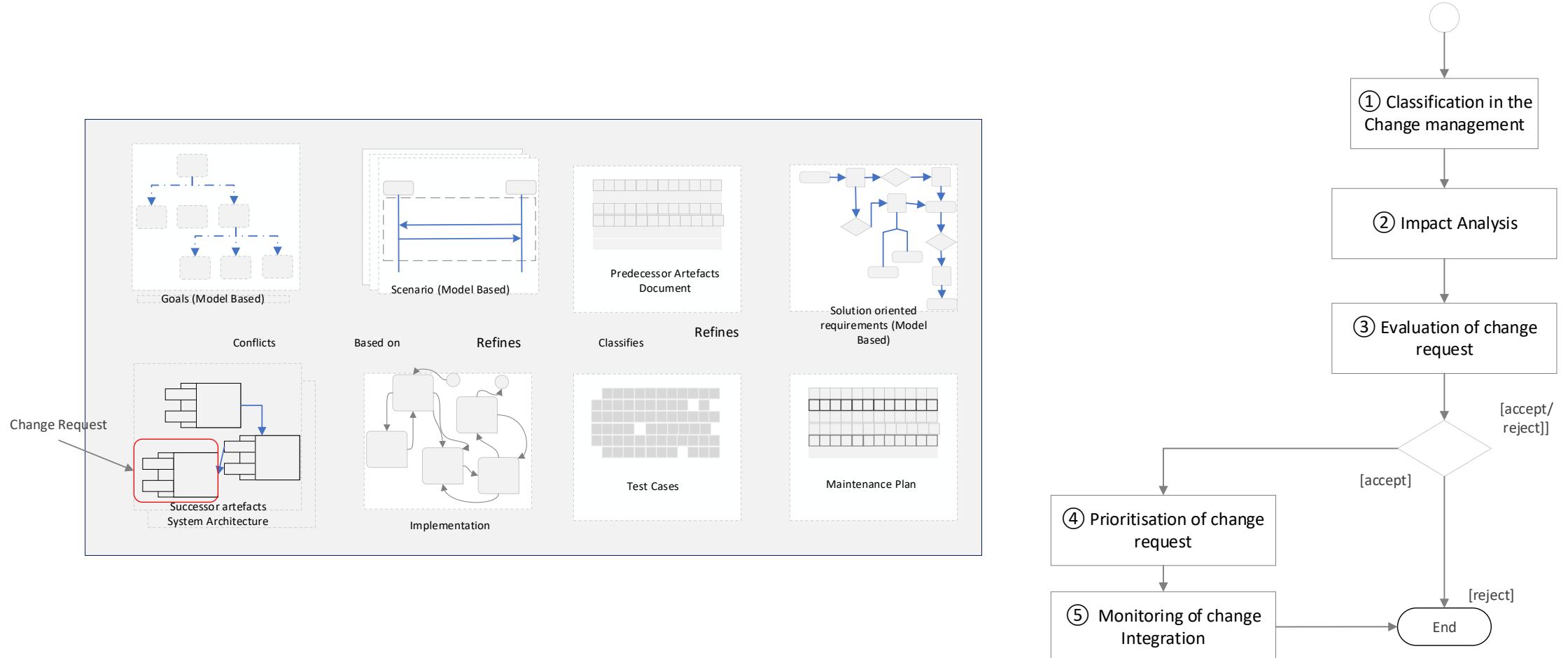
Content	Description
Project name	Name of the project to which the change applies
Request No.	Consecutive number of the change request
Title	Title of the change request
Date	Date of the change request
Originator	Name of the originator
Origin	Origin of the change (ex. marketing, customer, architectural design) from which the change originates
Status	Current status of the change request (ex. submitted, evaluated, rejected, to be integrated, verified, finalised)
Originator's Priority	Priority of the change request (defined by the originator of the change)
Verifier of the change	Name of the person responsible for verifying the change after its integration
Date of last update	Date of the last update of the change request
Release	Assignment to the release in which the change can be integrated
Integration effort	Estimated effort required for change integration
Description of the change request	Textual description of the change request
Comments	Comments relating to the change request, ex from other stakeholders, the change control board etc.

Referring to Requirements baseline, to Define type of Change Requests

Change requests for requirements typically refer to requirements of an established requirements baseline

- Integration of new requirement:** A new requirement has been elicited and shall be integrated into the requirements baseline
- Removal of an existing requirement:** An existing requirements artefact is invalid and shall therefore be removed from the requirements baseline
- Extension of an existing requirement:** An existing requirement shall be extended by particular aspects, e.g., attributes neglected so far. These extensions can be integrated into the requirements baseline
- Change of an existing requirement:** A requirement is changed in a way that can be classified neither as a single extension nor as a reduction and the change has to be integrated into the requirements baseline. Such a change can for example, modify the assignment of the output values to the input values of a function

Process of requirements change management



Module 2:
Requirement Analysis - Management



Thursday, 22nd August



13:00 PM – 14:00 PM

Agenda:

Hands on practise,

- Real life project handling as a BA
- Requirement Management modules questionnaire
- Documentation Practise (CR, RTM, RACI etc)



SAHIL DATERO
Business Analyst, Delivery

Module 2:
Requirement Analysis - Management



Friday, 23rd August



09:00 AM – 10:00 AM

Agenda:

Hands on practise,

- Real life project handling as a BA
- Requirement Management modules questionnaire
- Documentation Practise (CR, RTM, RACI etc)



SAHIL DATERO
Business Analyst, Delivery

Module 2:
Requirement Analysis - Management



Monday, 26th August



09:00 AM – 10:00 AM

Agenda:

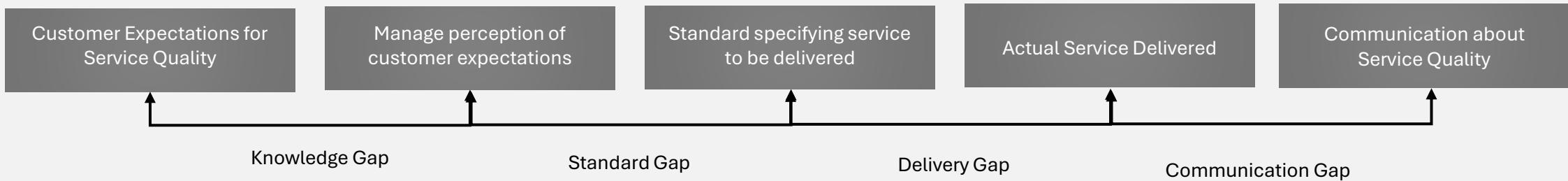
- Tasks of a Business Analysts
- An insight into Best practices for BA
- COCOMO II Model for Estimation



SAHIL DATERO
Business Analyst, Delivery

What it's Like to Be A Business Analyst?

- Business Analysts are always responsible for leading change, creating clarity, and driving alignment
- Business Analysts works with stakeholders across the organization to understand business objectives driving the change, define scope change, analyse and specify detailed requirements related to change and finally support implementation of change
- Business Analysts bridges multiple different Gaps among all stakeholders and ensures implementing change the generates as much value as possible



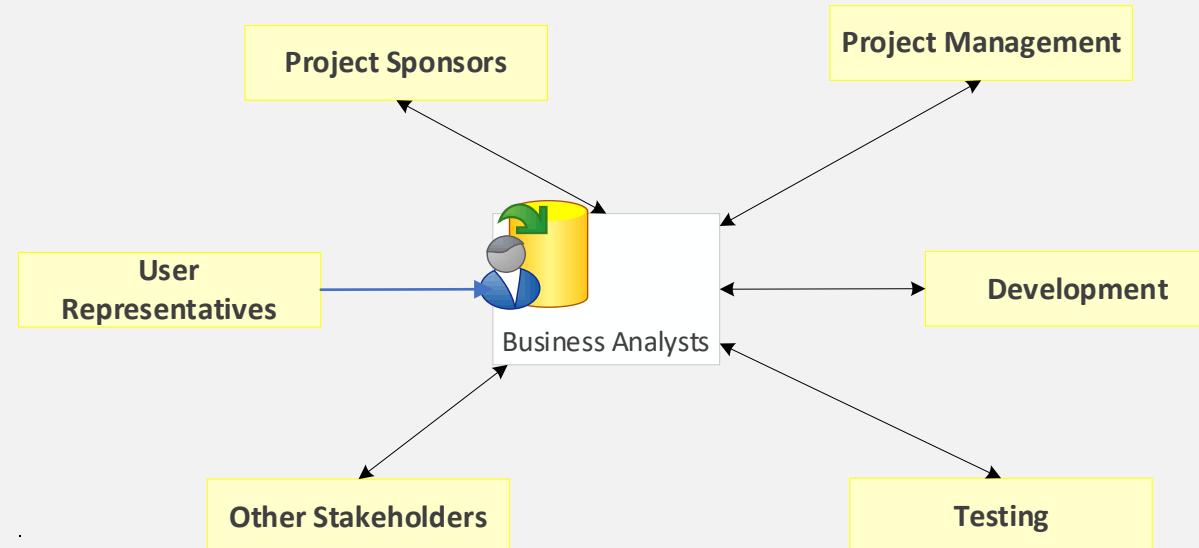
A number of software projects may benefit from the contributions of a business analyst. Common Types of projects includes the following - Business Process Improvement and Change, Business Intelligence/ Data warehousing, New product or service development, Customizations to existing products or service

According to BABOK Guide® Version 3:

Business Analysis ultimately helps organization to understand the needs of the enterprise and why they want to create change, design possible solutions, and describe how those solutions can delivery value

What it's Like to Be A Business Analyst?

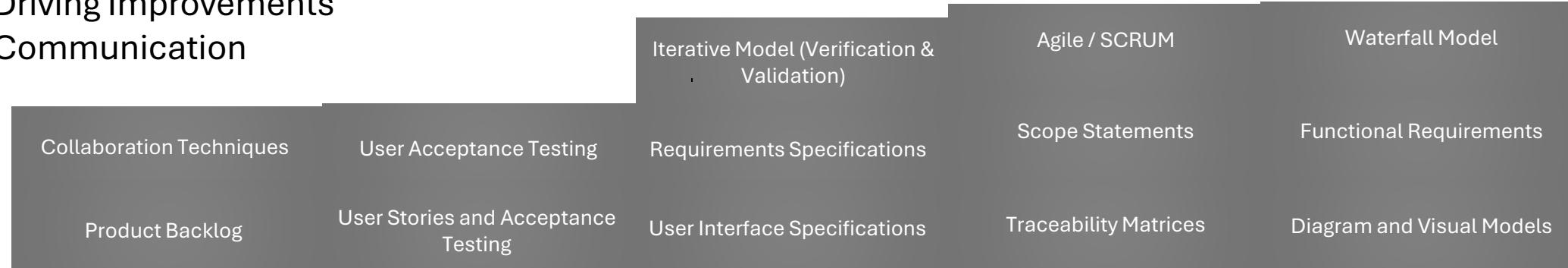
- Business Analysts is the individual who has the primary responsibility to elicit, analyse, document, and validate the needs of stakeholders. The analyst serves as a principle interpreter through which requirements flow between consumer community and the software development team.



Trap: Don't assume that any talented developer or knowledgeable user can automatically be an effective business analyst without training, resource materials and coaching

Underlying Core Competencies

- Problem-Solving And Critical Thinking
- Analytical Thinking
- Relationship Building
- Self Management
- A thick Skin
- Driving Improvements
- Communication



How to measure the success of a Business Analyst?

Project Success Measures: On-Time Delivery, On-Budget Delivery, On-Specification Delivery, ROI

Business Analyst efforts level: Minimized requirements change requested late in the project, Minimized requirements defects discovered after approval, Efficiency of the requirements effort, Increased business benefits (return part of ROI)

Essential Business Analyst's skills

Skills provided in this section are important. Young (2004) provides a comprehensible table of skills that are appropriate for Junior-level, mid-level, senior-level business analysts

Listening Skills: To become effective in two-way communication, learn how to listen effectively. Active listening involves eliminating distractions, maintaining attentive posture, and eye contact, and restating key points to confirm your understanding. You need to grasp what people are saying and also read between the lines to detect what they might be hesitant to say. Learn how your collaborators prefer to communicate, avoid imposing your personal filter of understanding what you hear from the customers.

Interviewing and questioning skills: Most requirements input comes through discussions, so a BA must be able to interact with diverse individuals and groups about their needs. You need to ask right questions to surface essential requirements information. For example, users naturally focus on system's normal, expected behaviours. However, much code gets written to handle exceptions. Therefore, you must also probe to identify error conditions and determine how the system should respond. It is with experience that you will become skilled in art of asking questions that reveal and clarify uncertainties, disagreements, assumptions and unstated expectations (Gause and Weinberg 1989)

Thinking on your feet: Business Analysts always need to be aware of the existing information and to process new information against it. They need to spot contradictions, uncertainties, vagueness and assumptions so they can discuss them in a moment if appropriate. You can try to script the perfect set of questions. You need to draft good questions, listen carefully to responses and quickly come up with the next smart thing to say or ask.

Analytical Skill: An effective business analyst can think both in high levels and low levels of abstraction. Sometimes you will need to drill down from high-level information into details, on other side you will need to generalize from a specific need that one user described to a set of requirements that will satisfy multiple stakeholders. BAs need to understand complex information coming from many sources and to solve hard problems related to that information. They need to critically evaluate the information to reconcile conflicts, separate users ‘wants’ from the underlying true needs, and distinguish solution ideas from requirements.

Systems Thinking Skills: Although a business analyst must be detail-oriented, he must also see the big picture. The BA must check requirements against what he knows about the whole enterprise, the business environment, and the application to look for inconsistencies and impacts. The BA needs to understand the interactions and relationships among the people, processes, and technology related to the system to judge whether the requirements affects other parts of the system.

Learning skills: Analysts must learn new material quickly, whether it is about new requirements approaches or the application domain. They need to be able to translate that knowledge into practice efficiently. Analysts should be efficient and critical reader. You don't have to be an expert in that domain, so don't hesitate to ask clarifying questions. Be honest about what you don't know, it's ok to not know it all but it's not ok to hide your ignorance.

Facilitation skills: The ability to facilitate requirements discussions, elicitation workshops is a vital analyst capability. Facilitation is the art of leading a group towards success. Facilitation is essential when collaboratively defining requirements, prioritizing needs, and resolving conflicts. A neutral facilitator who has a strong questioning, observational and facilitation skills can help a group build trust.

Leadership skills: A strong analyst can influence a group of stakeholders to move in a certain direction to accomplish a common goal. Leadership requires understanding a variety of techniques to negotiate agreements among project stakeholders, resolve conflicts, and make decisions. The analyst should create a collaborative environment, fostering trust among the various stakeholder groups who might not understand each other's motivations, needs and constraints.

Observational skills: An observant analyst will detect comments made in passing that might turn out to be significant. By watching a user perform her job or use a current application, a good observer can detect subtleties that user might not think to mention. Strong observational skills sometimes expose new areas for elicitation techniques, thereby revealing additional requirements.

Communication skills: The principle deliverable from requirements development is a set of written requirements that communicates information effectively among customers, marketing managers and technical staff. The analyst need a solid command of the language and the ability to express complex ideas both in written form and verbally. You must be able to write for multiple audience, including customers who have to validate the requirements and developers who need clear, precise environments for implementation. A BA needs to be able to summarize and present information at level of detail the target audience needs.

Organizational skills: BAs must contend with a vast array of jumbled information gathered during elicitation and analysis. Coping with rapidly changing information and structuring all the bits into a coherent whole demands exceptional organizational needs and the patience and tenacity to make sense from ambiguity and disarray. As an analyst you need to be able to set up an information architecture to support the project information as it grows through-out the project (Beatty and Chen 2012)

Modeling skills: Modeling ranging from the venerable flowchart through structured analysis models (data flow diagrams, entity-relationship diagram, and similar diagrams) to Unified Modelling Language (UML Notations) should be part of every analyst's repertoire (Beatty and Chen 2012). The BA will need to know when to select specific models based on how they add value. Also he'll need to educate other stakeholders on the value of using these models and how to read them.

Interpersonal skills: Analysts must be able to get people with competing interests to work together as a team. An analyst should feel comfortable talking with individuals in diverse job functions and at all levels of the organizations. A BA should speak the language of the audience she is talking to, not using technical jargon with business stakeholders. She might need to work with virtual teams whose members are separated geographically, time zones, cultures or native languages. A BA should be easy to communicate with and be clear and consistent when communicating with team members.

Creativity: The BA is not merely a scribe who records whatever customer say. The best analysts invent potential requirements for customers to consider (Robertson 2002). They conceive innovative product capabilities, imagine new markets and business opportunities, and think of ways to surprise and delight their customers. A really valuable BA finds creative ways to satisfy needs that users didn't even know they had. Analyst should be careful to avoid gold-plating the solution (don't simply add new requirements to specification without customer approval)

The business analyst's tasks

This section describes some of the typical activities that you might perform while wearing an analyst's hat.

Define business requirements: Your work as BA begins when you help the business or funding sponsor, or marketing manager define the business requirements. You might suggest a template for a vision and scope document and work with those who hold the vision to help them express it clearly.

Plan the requirements approach: The analyst should develop plans to elicit, analyse, document, validate, and manage requirements throughout the project. Work closely with the project manager to ensure these plans align with the overall project plans and help them achieve the project goals.

Identify project stakeholders and user classes: Work with the business sponsors to select appropriate representatives for each user class, enlist their participation, and negotiate their responsibilities. Explain what you would like from your customer collaborators and agree on an appropriate level of engagement from each one.

Elicit requirements: A proactive analyst helps user to articulate the system capabilities they need to meet their business objectives by using a variety of information- gathering techniques.

Analyse requirements: Look for derived requirements that are a logical consequence of what the customers requested and for implicit requirements that the customers seem to expect without saying so. Use requirements models to recognize patterns, identify gaps in the requirements. Reveal conflicting requirements and confirm that all requirements specified are within scope. Work with stakeholders to determine the necessary level of detail for specifying user and functional requirements.

Document Requirements: The analyst is responsible for documenting requirements in a well organized and well written manner that clearly describes the solution that well address customer's problem. Using standard template accelerates requirements development by reminding the BA of topics to discuss with the user representatives.

Communicate requirements: You must communicate the requirements effectively and efficiently to all parties. The BA should determine when it is helpful to represent requirements by using methods other than text, including various types of analysis models, tables, mathematical equations. Communication is not simply a matter of putting requirements n paper and tossing them over a wall. It involves ongoing collaboration with the team to ensure that they understand the information you are communicating

Lead requirements validation: The BA must ensure that requirement statements possess desired characteristics, “Writing excellent requirements” and that a solution based on requirements will satisfy stakeholder needs. Analysts are the central participants in reviews of requirements. You should also review designs and tests that were derived from the requirements to ensure that the requirements were interpreted correctly. If you are creating from the requirements to ensure that the requirments were interpreted correctly. If you are creating acceptance tests in a place of detailed requirements on an agile project, those should also be reviewed.

Facilitate requirements Prioritization: The analyst brokers collaboration and negotiation among the various stakeholders and the developers to ensure that they make sensible priority decisions in alignment with achieving business objectives

Manage requirements: A business analyst is involved throughout the entire software development life cycle, so she should help create, review, and execute the project's requirements management plan. After establishing a requirements baseline for a given product release or development iteration, the BA's focus shifts to tracking the status of those requirements, verifying their satisfaction in the product, and managing changes to the requirements baseline. With input from various colleagues, the analyst collects traceability information that connects individual requirements to other system elements.

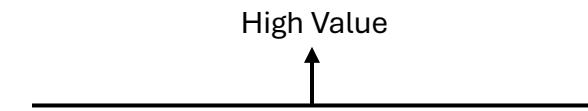
Getting started with new practices

This section groups the Business analysis good practices, by relative value they can contribute to most projects and relative difficulty of implementation

- Define a requirements engineering process
- Base plans on requirements
- Renegotiate commitments
- Train business analysts
- Plan Requirements approach
- Select product champions
- Identify user requirements
- Hold elicitation interviews
- Specify nonfunctional requirements
- Prioritize requirements
- Define vision and scope
- Establish a change control process
- Review the requirements
- Allocate requirements to subsystems
- Use a requirements management tool
- Record business rules

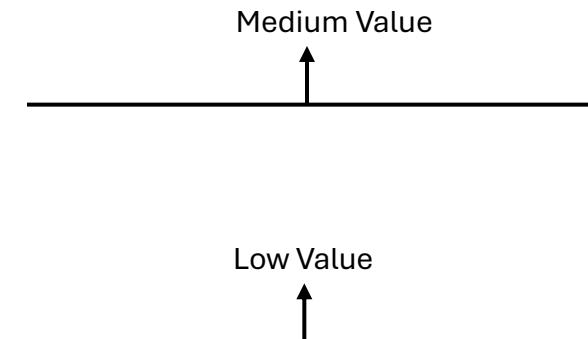
High Difficulty	<input checked="" type="checkbox"/>
Medium Difficulty	<input type="checkbox"/>
Low Difficulty	<input checked="" type="checkbox"/>

- Educate developers about application domain
- Adopt requirement document templates
- Identify user classes
- Model the application environment
- Identify Requirement origins
- Establish baselines and control versions of requirements sets
- Identify requirements decision makers
- Maintain a requirements traceability matrix
- Hold facilitated elicitation workshops
- Estimate requirements efforts
- Reuse existing requirements
- Educate stakeholders about requirements
- Conduct focus groups
- Create prototypes
- Analyse feasibility
- Define acceptance criteria
- Model the requirements
- Analyse interfaces
- Perform change impact analysis
- Select an appropriate life cycle



High Difficulty	<input checked="" type="checkbox"/>
Medium Difficulty	<input type="checkbox"/>
Low Difficulty	<input type="checkbox"/>

- Identify system events and responses
- Review past lessons learned
- Track requirements effort
- Create a data dictionary
- Observe users performing their jobs
- Test the requirements
- Track requirements status
- Perform document analysis
- Track requirements issues
- Uniquely label each requirement
- Create a glossary
- Distribute questionnaires
- Maintain change history
- Simulate the requirements
- Examine Problem reports



High Difficulty	<input type="checkbox"/>
Medium Difficulty	<input type="checkbox"/>
Low Difficulty	<input checked="" type="checkbox"/>

- COCOMO II Model for Estimation

Module 3:
Requirement Analysis – Modelling



Tuesday, 27th August



09:00 AM – 10:00 AM

Agenda:

- Overview of Model Based System Engineering (MBSE)
- Three pillars of MBSE
- The Myths of MBSE
- Overview of SyML Diagrams



SAHIL DATERO
Business Analyst, Delivery

MBSE alternative to document-based approach through system design lifecycles

- With the document-based approach, systems engineers manually generate some subset of following artefacts:

- Concept of operations (ConOps) documents
- Requirements Specifications
- Interface Definition Documents (IDDs)
- Requirement Traceability and Verification Matrices
- Matrices of Structural interfaces (N^2 Charts)
- Architecture Description Documents (ADDs)
- System Design Specifications
- Test Case Specifications

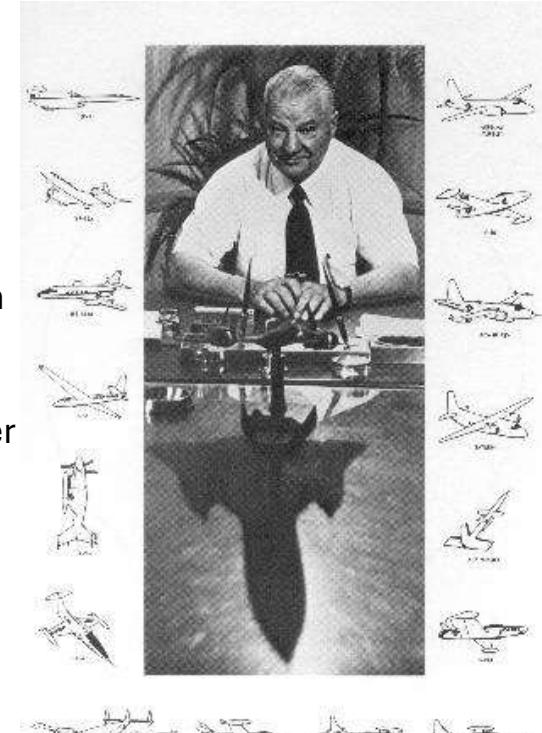
□ Problem is this: The document – based system engineering is expensive, it is time consuming and error prone

□ In Model-Based System Engineering, the system model serves as central repository for decisions; each decision is captured as a model element (or relationship between elements) in a single place within the system model.

□ With MBSE approach all diagrams and autogenerated text artefacts are merely views of the underlying model; they are not the model itself. And that distinction is the root of the return on investment (ROI) that MBSE offers over traditional approach.

□ MBSE Promises increased quality and affordability for one simple reason:
The cheapest defect to fix is the one you prevented.

MBSE is the formalized application of modeling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing through development and later life cycle phases



The three pillars of MBSE – Modelling Languages, Modelling Methods and Modelling Tools

① Modelling Languages

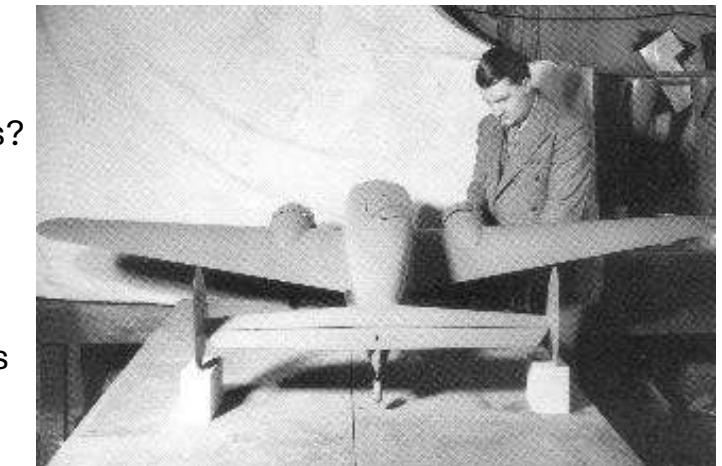
When you create a model you are speaking a language. It's not the natural language you learned as a child at home or in school. MBSE Practitioners commonly use the System Modelling Language (SysML) to construct models of a system's structure, behavior, Requirements and constraints. Other than SysML, other graphical modelling languages are (e.g., UML, UPDM, BPMN, MARTE, SoaML, IDEFx, etc) other text modelling languages are (e.g Verilog, Modelica). A modelling language defines a grammar: a set of rules that determines whether a given model is well formed or ill formed.

② Modelling Methods

A Modelling Method is like a roadmap; It's a documented set of design tasks, that a modelling team performs to create a system model. It's documented set of design tasks that ensures that everyone on the team is building the system model consistently and working toward a common end point. Like all projects, an MBSE project requires a plan. And every plan begins with a purpose.

Your team will begin with answering the following questions for finding purpose:

- Why are you modelling?
- What are the expected results of the modelling efforts?
- Are you creating a model only to serve as the central record of authority for all design decisions?
- Do you need to autogenerate text artefacts from the model for review and approval?
- Will you use the model to manage requirements traceability and perform Downstream impact analysis?
- Will you use the model to perform trade studies of alternative configurations?
- Will the system model be integrated with dedicated equation-solving tools and simulation tools to execute the model directly?
- Will the model itself be an input for the work of downstream design and development team such as hardware/ software/ reliability/ availability/ performance analysis?
- Will the model contain the integration and acceptance test cases that will verify system assembly after development?



The three pillars of MBSE – Modelling Languages, Modelling Methods and Modelling Tools

Once your team has defined that purpose, you can then answer a new set of questions:

- How much of the external environment of your proposed system needs to be modelled?
- Which parts of your system need to be modelled?
- Which behaviours needs to be modelled?
- Which detail needs to be in the model? How deeply you need to decompose the internal structures and behaviours?
- Which details can be in model and which needs to be omitted?

The answers to these questions determine the Scope of the system model your team needs to build.

The scope of the model also determines the modelling method that your team will follow. Focus of this course is SysML, but here is a list of some well known modelling methods:

- Method: INCOSE Object-Oriented Systems Engineering Method (OOSEM)

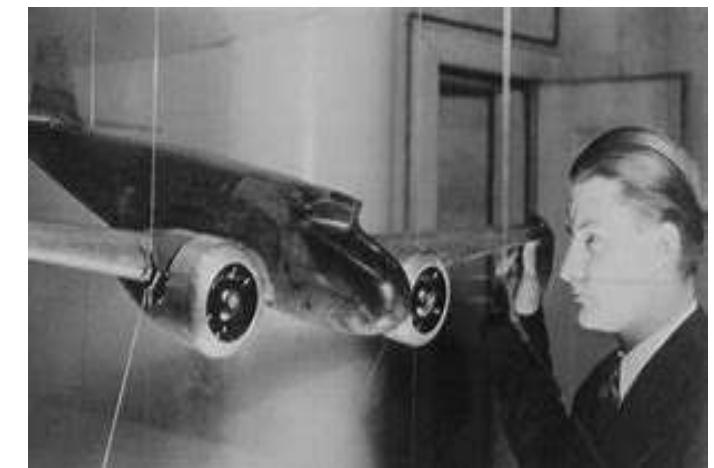
Reference: Friedenthal, Sanford, et al., A Practical Guide to SysML, Second Edition: The Systems Modeling Language (Boston: MK/ OMG Press, 2011)

- Method: Weilkiens System Modeling (SYSMOD) method

Reference Weilkiens, Tim, Systems Engineering with SysML/UML: Modeling, Analysis, Design (Boston: MK/OMG Press, 2008)

- Method: IBM Telegonic Harmony-SE

Reference: Hoffmann, Hans-Peter, “Harmony-SE/SysML Desk-book: Model-Based Systems Engineering with Rhapsody,” Rev. 1.51, Telelogic/I-Logix white paper (Telelogic AB, May 2006)



The three pillars of MBSE – Modelling Languages, Modelling Methods and Modelling Tools

③ Modelling Tools

Developing Proficiency with a modelling tool is the third pillar of MBSE. Modelling Tools are a special class of tools that are designed and implemented to comply with the rules of one modelling languages, enabling you to construct well-formed models in those languages. Modelling tools are distinct from diagramming tools such as Visio, Schematic, SmartDraw, ProcessOn, and others. With a Diagramming Tool, you create diagrams – Shapes on a page. There is no model underlying those diagrams that ensure automated consistency between them. In contrast, with a modelling tool, you create a model – a set of elements and relationships between the elements, and optionally a set of diagrams that serve as views of the underlying model.

The following are commercial-grade modelling tools:

- Agilian (vendor: Visual Paradigm)
- Artisan Studio (vendor: Atego)
- Enterprise Architect (vendor: Sparx Systems)
- Cameo Systems Modeler (vendor: No Magic)
- Rhapsody (vendor: IBM Rational)
- Umodel: (vendor: Altova)

The following are free modeling tools,

- Modelio (creator: Modeliosoft)
- Papyrus (creator: Atos Origin)

Bursting the Myths of MBSE

- MBSE does not (and cannot) eliminate the difficult work of architecting a designing a system well

The myth of MBSE arises among stakeholders – external customers and internal downstream design and development teams – who know about MBSE but don't practice themselves. The myth they harbor deep inside is that MBSE is an Easy Button, they believe incorrectly that MBSE makes every engineering task easier and reduces cost at every point in the life cycle

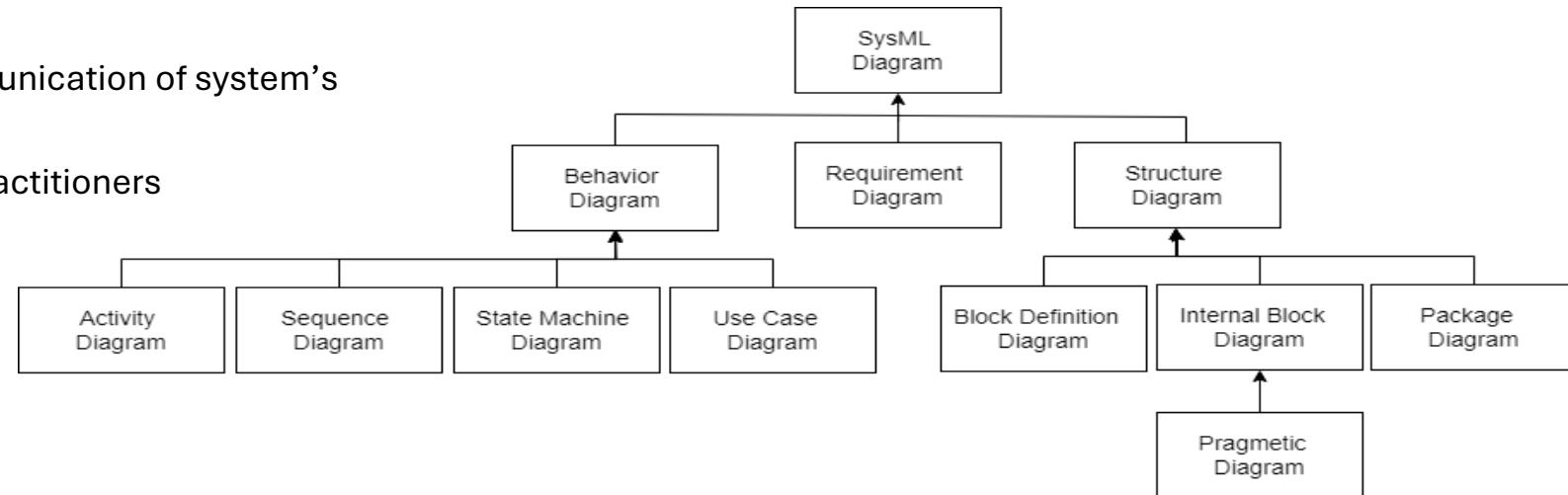
- Modelling well is difficult. Designing well is difficult.

It's possible to create a bad model. And it's possible to create a good model of a poorly designed system. Creating a good model of a well-designed system to the degree of breadth, depth and fidelity required to satisfy the model's purpose takes time and hard work and discipline.

- MBSE Delivers ROI when change happens

When new design decisions are made and stakeholder's needs evolve throughout the system life cycle. And change of-course inevitably happens.

- Purpose of SysML is visualization and communication of system's design among stakeholders
- SysML is the language "Spoken" by MBSE Practitioners
- SysML is a Graphical language
- Vocabulary of SysML is Graphical Notations
- SysML is owned and Published by Object Management Group Inc (OMG)



Brief summary of the purpose of each kind of diagram

Diagram	Description
Block Definition Diagram (BDD)	BDD Is used to display elements such as blocks and value types (elements that define the types of things that exist in an operational system) and the relationships between those elements. Common uses for a BDD include displaying system hierarchy trees and classification trees
Internal Block Diagram (IBD)	IBD is used to specify the internal structure of a single block. More precisely, an IBD shows the connections between the internal parts of a block and the interfaces between them.
Use Case Diagram	The Use Case Diagram is used to convey the use cases a system performs and the actors that invoke and participate in them. A use case diagram is a black-box view of the services that a system performs in collaboration with its actors
Activity Diagram	The Activity diagram is used to specify a behavior with a focus on the flow of control and the transformation of inputs into outputs through a sequence of actions. Activity diagrams are commonly used as an analysis tool to understand and express the desired behavior of a system
Sequence Diagram	The Sequence Diagrams is used to specify a behavior with a focus on how the parts of block interact with one another via operation calls and asynchronous signals. Excellent mechanism for specifying test cases
State Machine Diagram	The State machine diagrams are used to specify behavior with a focus on set of states of a block and the possible transitions between those states in response to event occurrences. Serves as input to development stage of the life cycle
The Parametric diagram	The Parametric diagram is used to express how one or more constraints – specifically, equations and inequalities are bound to the properties of a system, It supports trade studies of candidate physical architecture
Package Diagram	The package diagram is used to display the way a model is organized in the form of a package containment hierarchy.
Requirements Diagram	The requirement diagram is used to display text-based requirements, relationships between requirements and relationship between requirements and the other model elements that satisfy, verify and refine them

Module 3:
Requirement Analysis – Modelling



Wednesday, 28th August



09:00 AM – 10:00 AM



SAHIL DATERO
Business Analyst, Delivery

Agenda:

- Block Definition Diagrams
- Purpose
- Elements and Relationships

Block Definition Diagrams

The most common kind of SysML Diagram is the block definition diagram. You can display various kinds of **model elements** and relationships on a BDD to express information. You can also adopt **design techniques** for creating extensible system structures, a practice that reduces time and cost to change your design as stakeholder's need evolve.

Purpose

The model elements that you display on BDDs – Blocks, actors, value, types, constraint blocks, flow specifications, and interfaces – serve as types for other model elements that appear on BDDs as elements of **definition**. Elements of definition are important; the structural relationships among them – associations, and dependencies – are arguably more important.

When should you create a BDD?

You should create a BDD often. BDDs are not tied to any system life cycle or level of design. You and your team will create them when you perform following business analysis activities – stakeholder needs analysis, requirements definition, architectural design, performance analysis, test case development, and integration. And you often create a BDD in conjunction with other SysML diagrams to provide a complementary view of an aspect of your system of interest.

Block Definition Diagrams - Frame

The BDD Frame

The model element type that the diagram frame represents can be any of the following:

- Package
- Model
- modelLibrary
- View
- Block
- constraintBlock

The model element type that the diagram represents serves as the namespace for the other elements shown on the diagrams. A **namespace** is simply a model elements that's allowed to contain other model elements; that is, it can have other elements nested under it within the model hierarchy. A name space, therefore, is a concept that has meaning only within your system model; it has no meaning within an instance of your system. A **package**, is however is the most common kind of namespace for the various elements of definition that appear on BDDs. Therefore, the element that's named in the header of a BDD typically is a package you've created somewhere in the model hierarchy.

Block Definition Diagrams

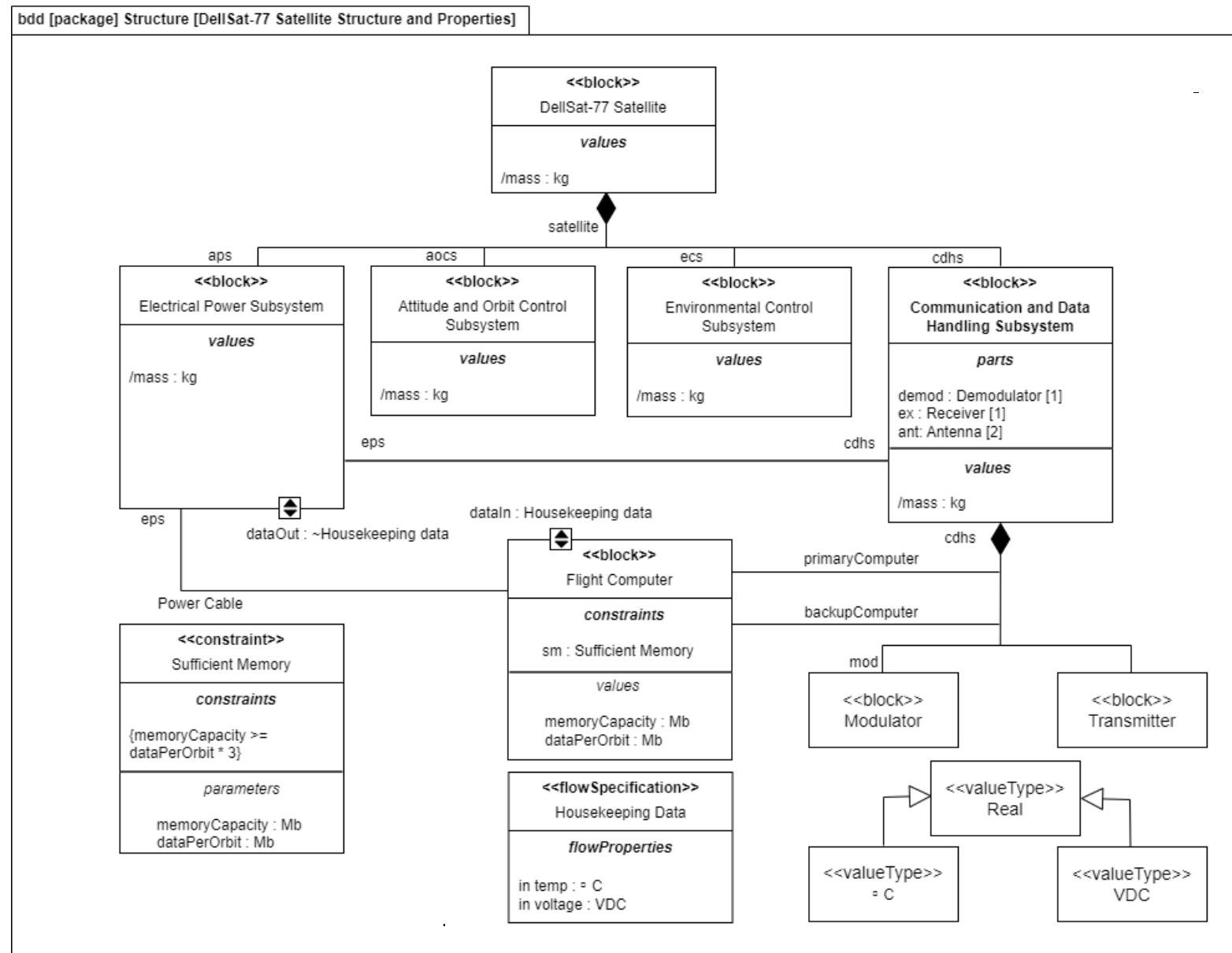


Fig: Sample block definition diagram

Block Definition Diagrams - Blocks

□ Blocks – Basic unit of structure in SysML

Note the distinction between *definition* and *Instantiation* (which SysML refers to as “usage”)

Some kind of model elements (e.g types, blocks, value types, constraint blocks) represent **definitions** of types; other kinds of model elements (e.g part properties, value properties, constraint properties) represent **instances** of those types.



Fig: Illustration to represent analogy of ‘definition’ and ‘instances’

Furthermore, Block would represent an entity, that defines a set of properties.

To distinguish between elements of definition and elements have usage - Elements of usage have a name only (e.g DesktopWorkstation)
Elements of usage have a name and a type, separated by a colon (e.g., SDX1205LJD : DesktopWorkstation)

Features come in two varieties: structural features (also known as properties_ and behavioural features.

Here are the optional compartments you can display:

- Parts
- References
- Values
- Constraints
- Receptions
- Standard ports
- Flow ports
- Full ports
- Proxy ports
- Flow properties
- Structure (You can display in this compartment all the same notations you can display in Internal block diagram (IBD))

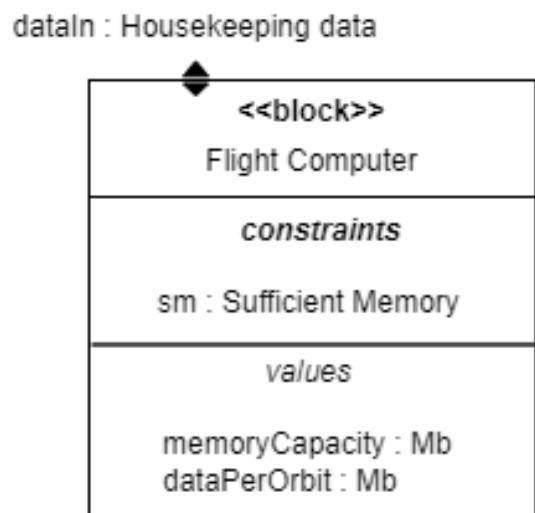


Fig: A block

Block Definition Diagrams – Properties

■ Structural Properties

There are five kinds of structural features (also known as **Properties**) that a block can own:

- Part properties
- Reference properties
- Value properties
- Constraint properties
- Ports

▫ Part Properties

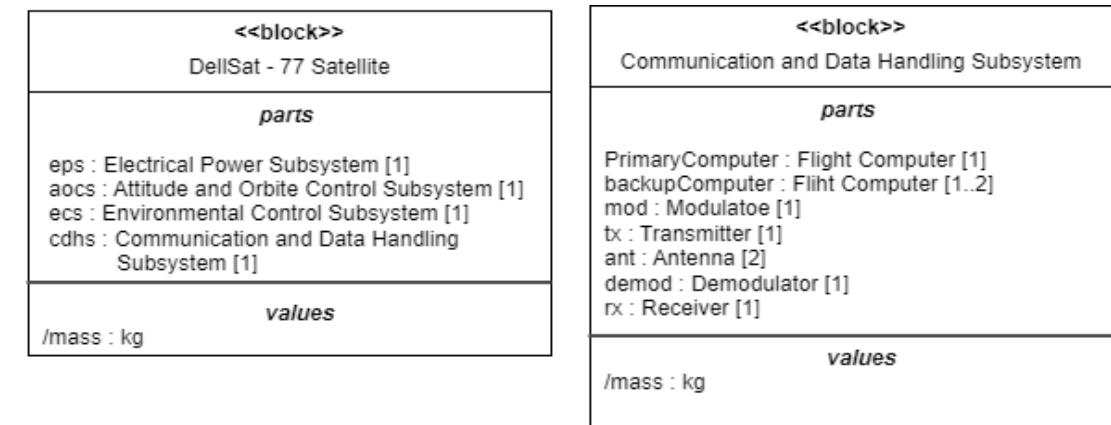
A part property represents a structure that's internal to a block. Stated differently, a block is composed of its part properties.

This relation conveys ownership, however, differs in essence. In hardware domain, ownership typically refers to a physical Composition. SysML states definitively that ownership means that a part property can belong to only one composite structure at a time.

When you list a part property in the parts compartment of block, it appears as a string with the following format:

<part name> : <type> [<multiplicity>]

Fig: Block with part properties



Block Definition Diagrams – Properties

▫ Reference Properties

A reference property represents a structure that's external to a block. Unlike a part property, reference property does not convey ownership.

When you list a reference property it appears as a string with the following format:

<reference name> : <type> [<multiplicity>]

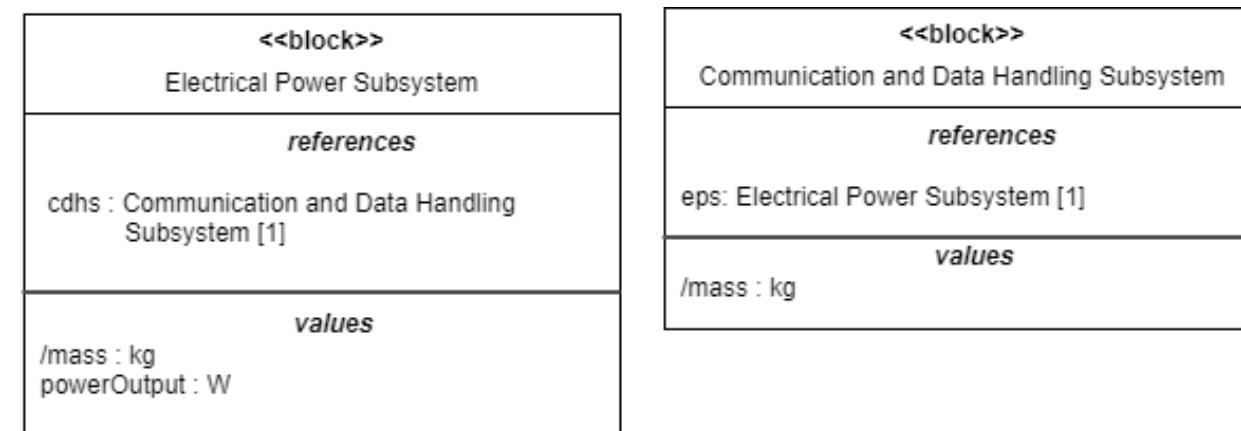


Fig: Block with reference properties

Block Definition Diagrams - Properties

▫ Value Properties

A value property represents a quantity, a Boolean, or a string. Value properties is something you can assign number to.
More on this during, “Parametric Diagrams”

When you list a value property it appears as a string with the following format (default field is an optional piece of information):
<value name> : <type> [<multiplicity>] = <default value>

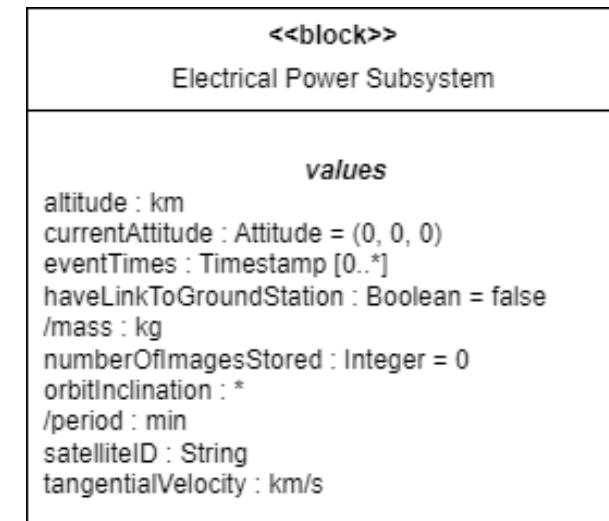


Fig: Block with value properties

Block Definition Diagrams - Properties

▫ Constraint Properties

A Constraint property represents a mathematical relationship (an equation or inequality), that is imposed on a set of value properties. This is a higher level of model fidelity than is required on most modelling projects.

When you list a constraint property in the constraints compartment of a block it appears as a string with the following format:

<constraint name> : <type>

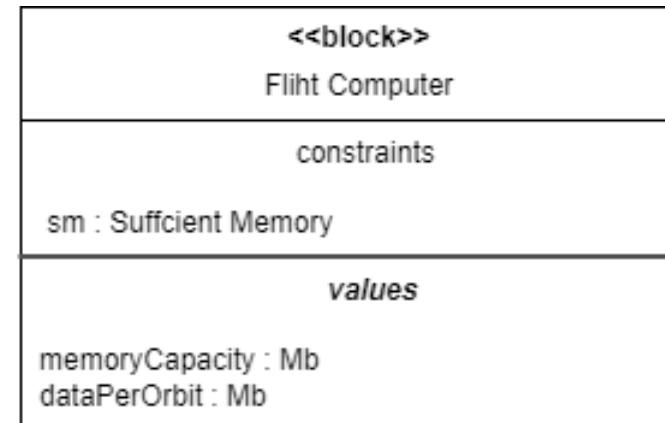


Fig: Block with constraint properties

Block Definition Diagrams - Ports

Ports

A port is a kind of property that represents a distinct interaction point at the boundary of a structure through which external entities can interact with that structure, either to provide or request a service or to exchange matter, energy or data. When you add a port to a block, You hide all the internal implementation from its client. Those clients only know the structure's interface (the structure's interface, the services it provides and required, and the types of matter, energy or data that can flow in and out). A **standard port** lets you specify an interaction point with a focus on the services that a block provides or requires. A **flow port** lets you specify an interaction point with a focus on types of matter, energy or data that can flow in and out of a block.

Each flow property has a direction, a name, and a type, which are displayed as a string in the following format:

<direction> <name> : <>

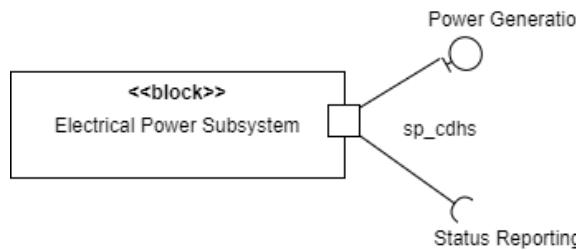


Fig: Blocks with standard ports

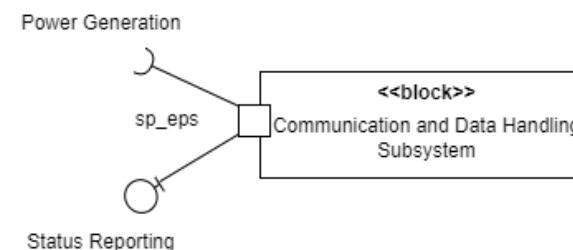


Fig: Interfaces

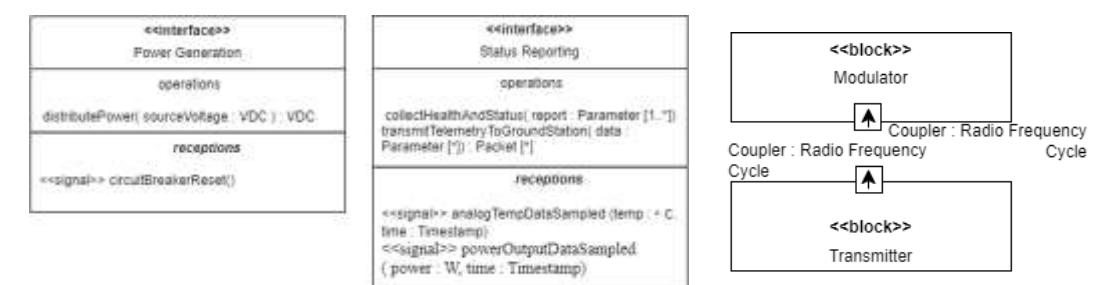


Fig: Blocks with atomic flow ports

Block Definition Diagrams – Behavioural Features

- Behavioral Features

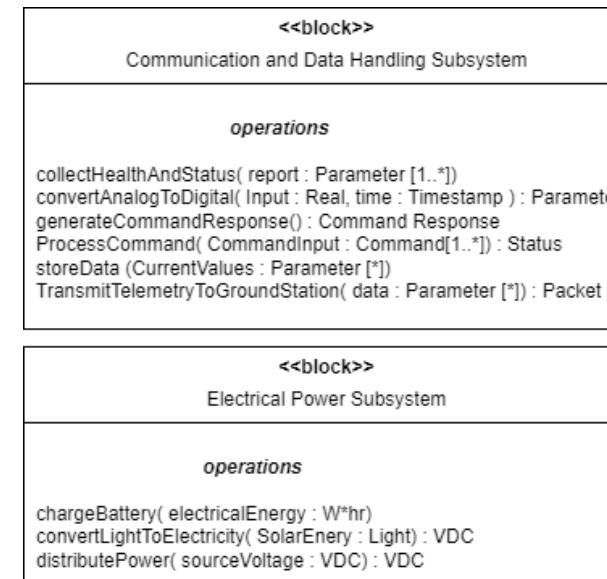
SysML offers two kinds of behavioral features: Operations and receptions.

- Operations

An operation represents a behavior that a block performs when a client calls it. Stated formally, an operation is invoked by a call event. To be seen detail when we study, “Activity Diagrams, Sequence Diagrams, State Machine Diagrams”

You display an operation on a BDD as a string in the operations compartment of a block. The string has following format
 <operation name> (<parameter list>) :<return type> [<multiplicity>]

Fig: Block with operations



Block Definition Diagrams - Behavioural Features

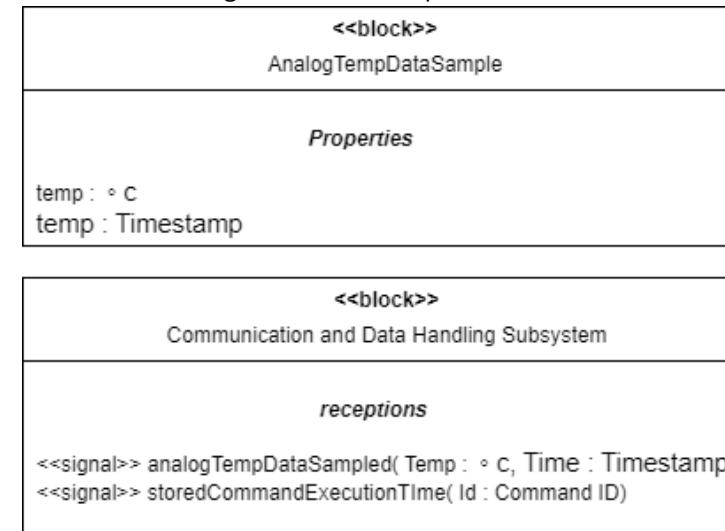
□ Reception

A reception represents a behavior that a block performs when a client sends a single that triggers it. Started formally, a reception is invoked by a **single event**. The key distinction between a reception and an operation is that a reception always represents an asynchronous behavior. Another key point is that a **signal** is itself a model element. You can use a signal to represent any type of matter, energy, or data that one [art of a system sends to another part – generally for the purpose of triggering a behavior on the receiving end.

You display a display a reception in the receptions compartment of a block. The string has following format

<signal> = <reception name> (<return type>)

Fig: Block with receptions



Module 3:

Requirement Analysis – Modelling



Friday, 30th August



09:00 AM – 10:00 AM

Agenda:

- Block Definition Diagrams
- Relationships – Associations
- Internal Block Diagrams
- Introduction



SAHIL DATERO
Business Analyst, Delivery

Block Definition Diagrams - Associations

'Blocks' focuses on blocks and various properties than blocks can own. Blocks and relationships are very important for system model.

There are three types of relationships

- Associations
- Dependencies
- Generalizations

While studying 'Structural Features', we ideate, **reference property** and **part property** as two associations we create between blocks

- A reference property represents a structure that's external to a block – A structure that the block needs to be connected for some purpose
- A part property instead represents a structure that's internal to a block – in other words, a structure that block is composed of.

Reference Property and Part Property different notation, is called 'Reference association' and 'composite association' respectively.

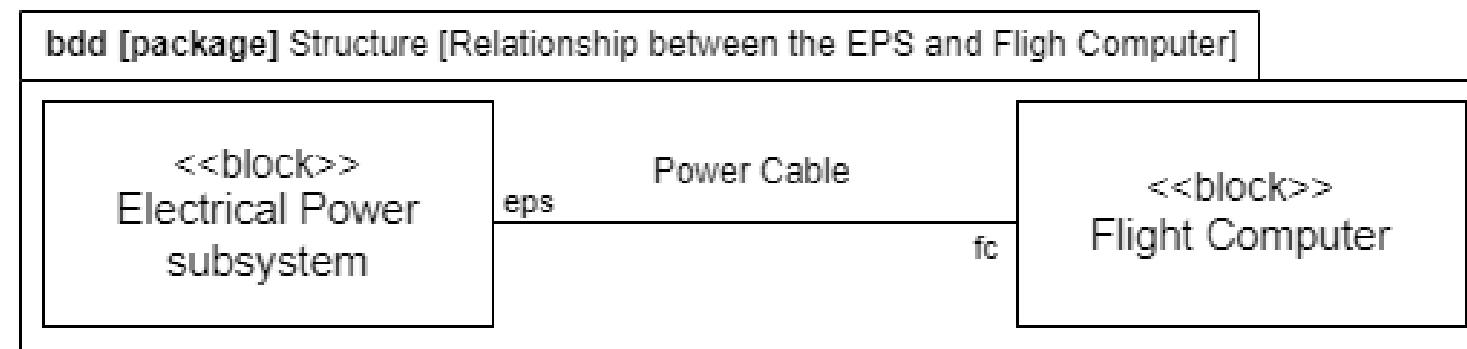
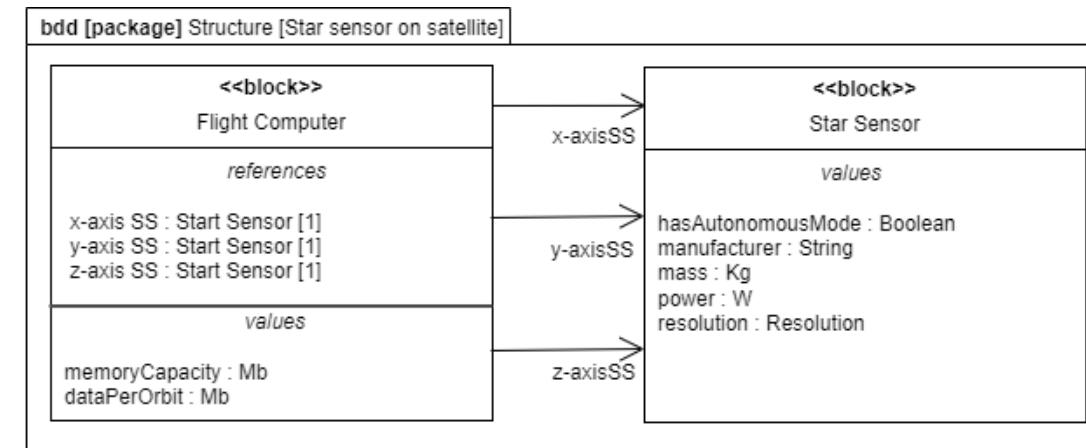


Fig: Reference associations

Block Definition Diagrams - Associations

- Another factor you intend in your decision is the need to specify a type for a connector on an IBD.

Reference Association



Composite Association

Fig: Using reference associations to specify multiple reference properties of the same type

Block Definition Diagrams – Composite Association conveys structural decomposition

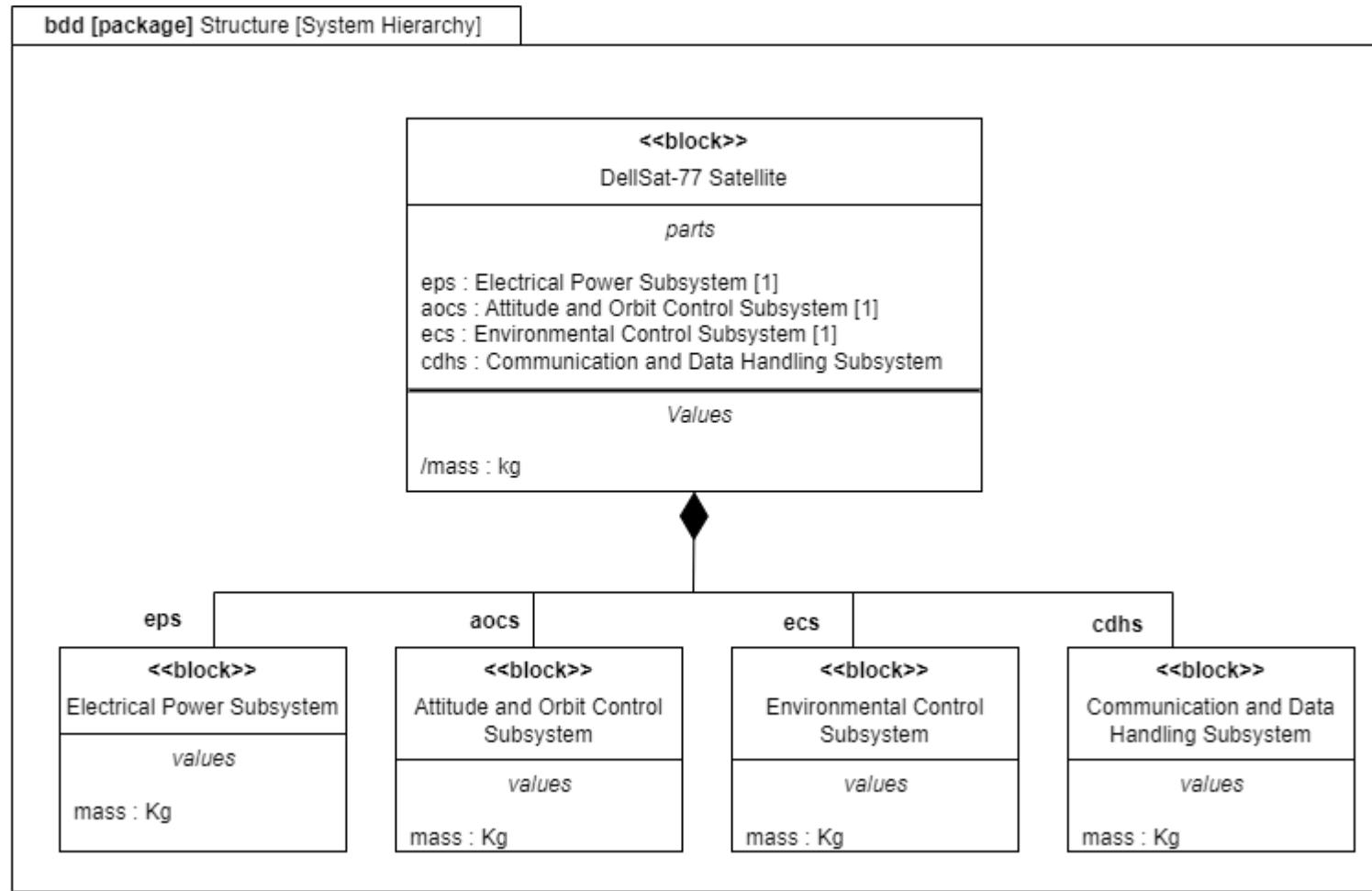


Fig: Composite relation and part properties

Block Definition Diagrams - Associations

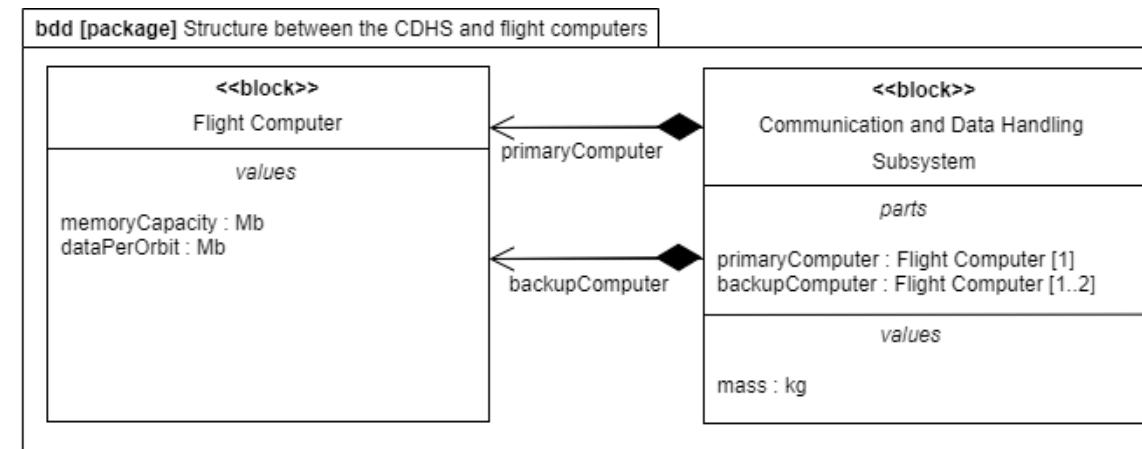


Fig: Using reference associations to specify multiple reference properties of the same type

Block Definition Diagrams - Associations

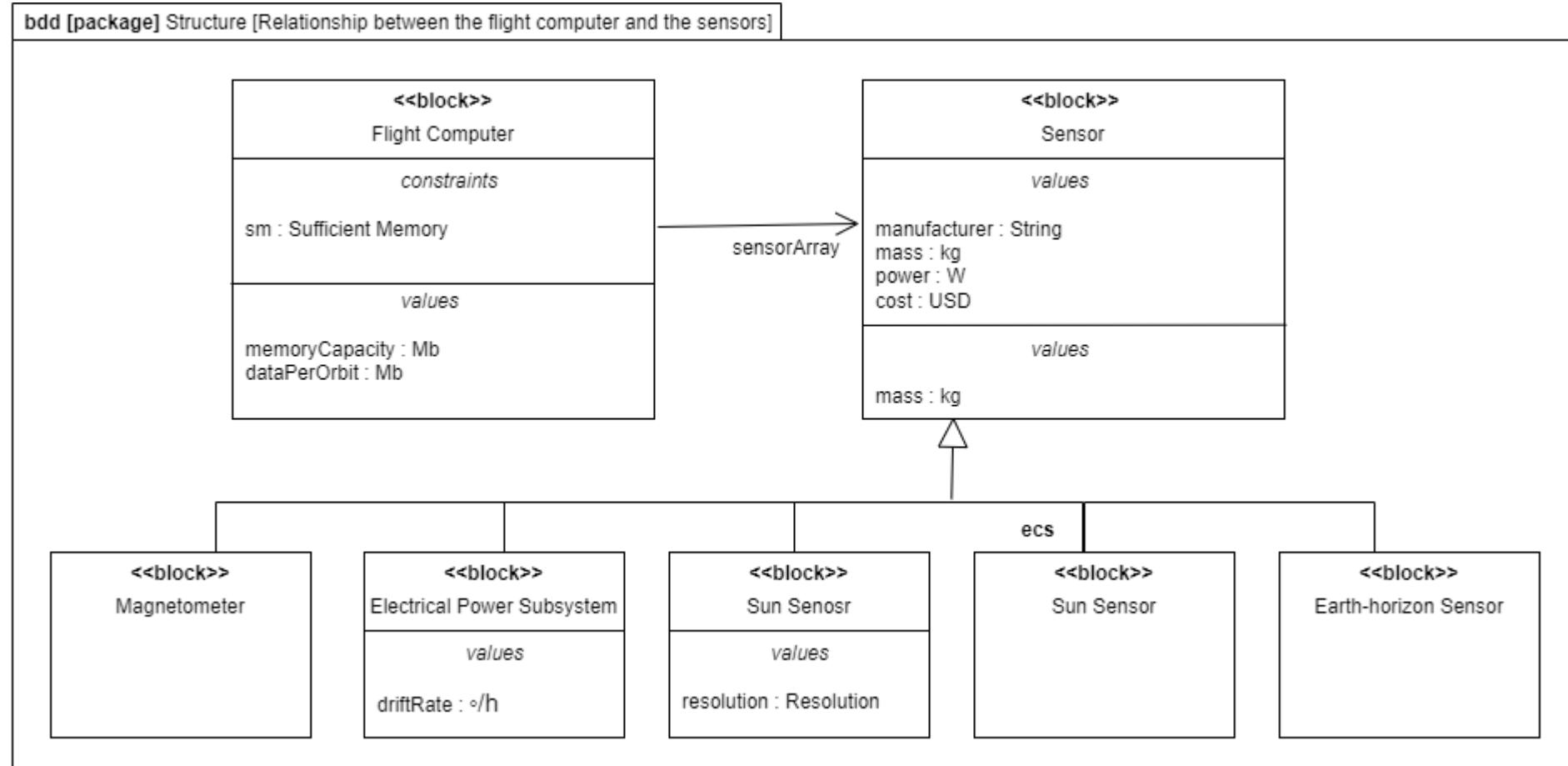
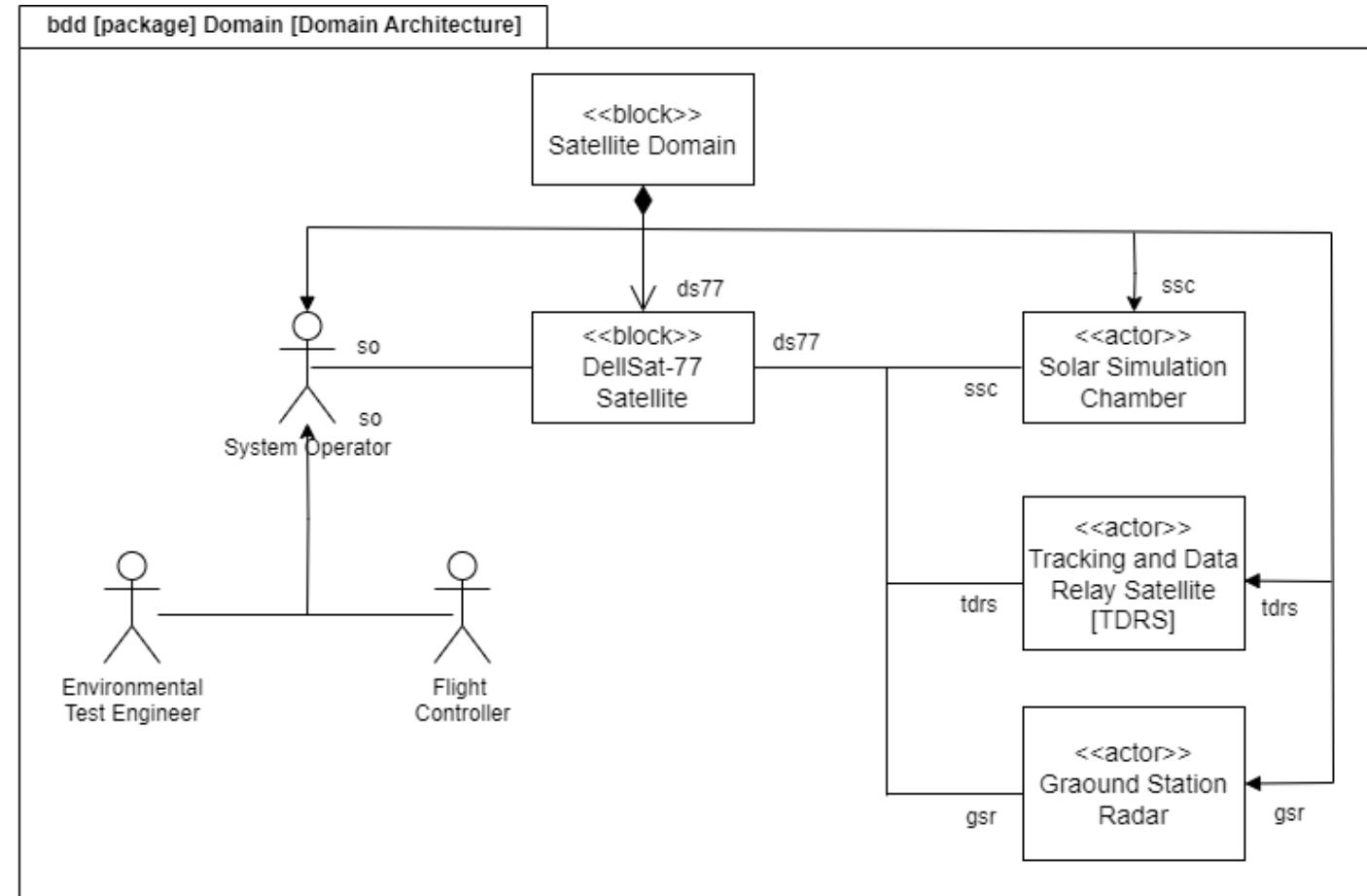


Fig: Designing to an abstraction

Block Definition Diagrams - Associations

Fig: Actors on a *BDD*

Block Definition Diagrams - Dependencies

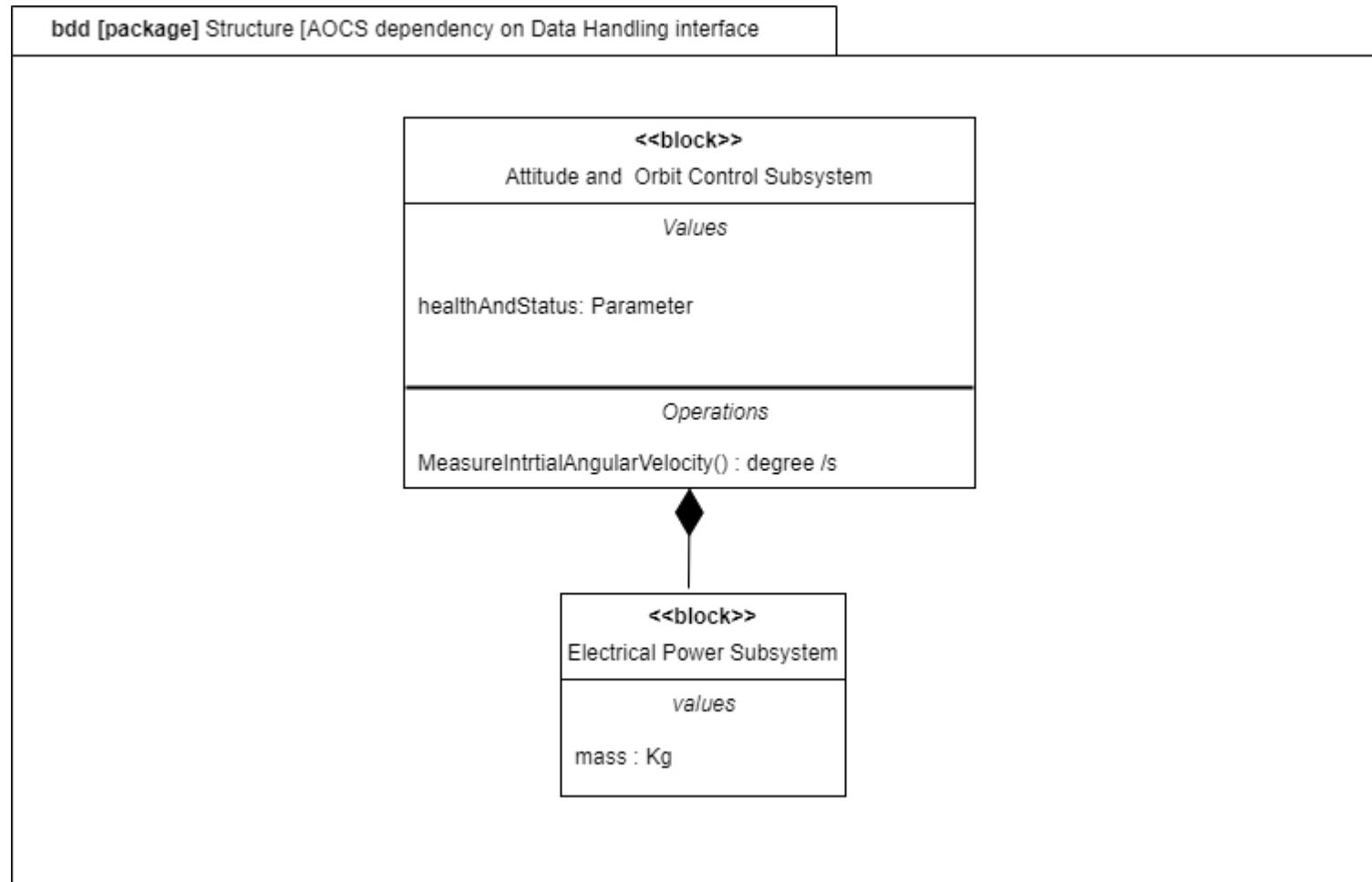


Fig: Dependencies on a *BDD*

Block Definition Diagrams – Value Types

Values Properties can appear as the types of the following:

- Atomic flow ports on blocks and actors
- Flow properties in flow specifications
- Constraint parameters in constraint blocks
- Item flows and item properties on connectors
- Return Types of Operations
- Parameters of operations and receptions
- Object nodes, pins and activity parameters within activities

There are three kinds of value types

- 1) Primitive: String, Boolean, Integer, real
- 2) Structured: has a specific structure
- 3) Enumerated: Literals (legal values)

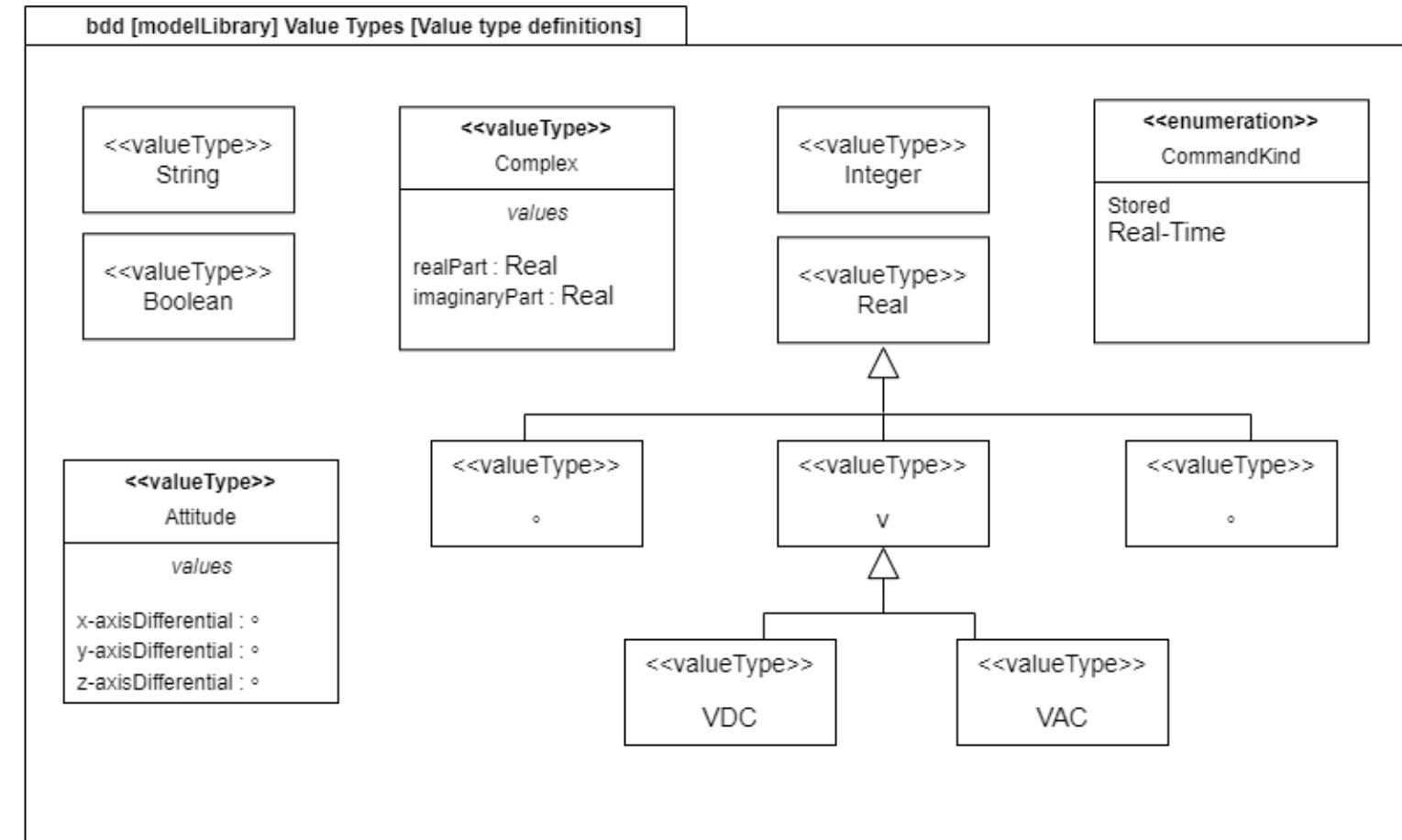


Fig: Value Types

Block Definition Diagrams – Value Types

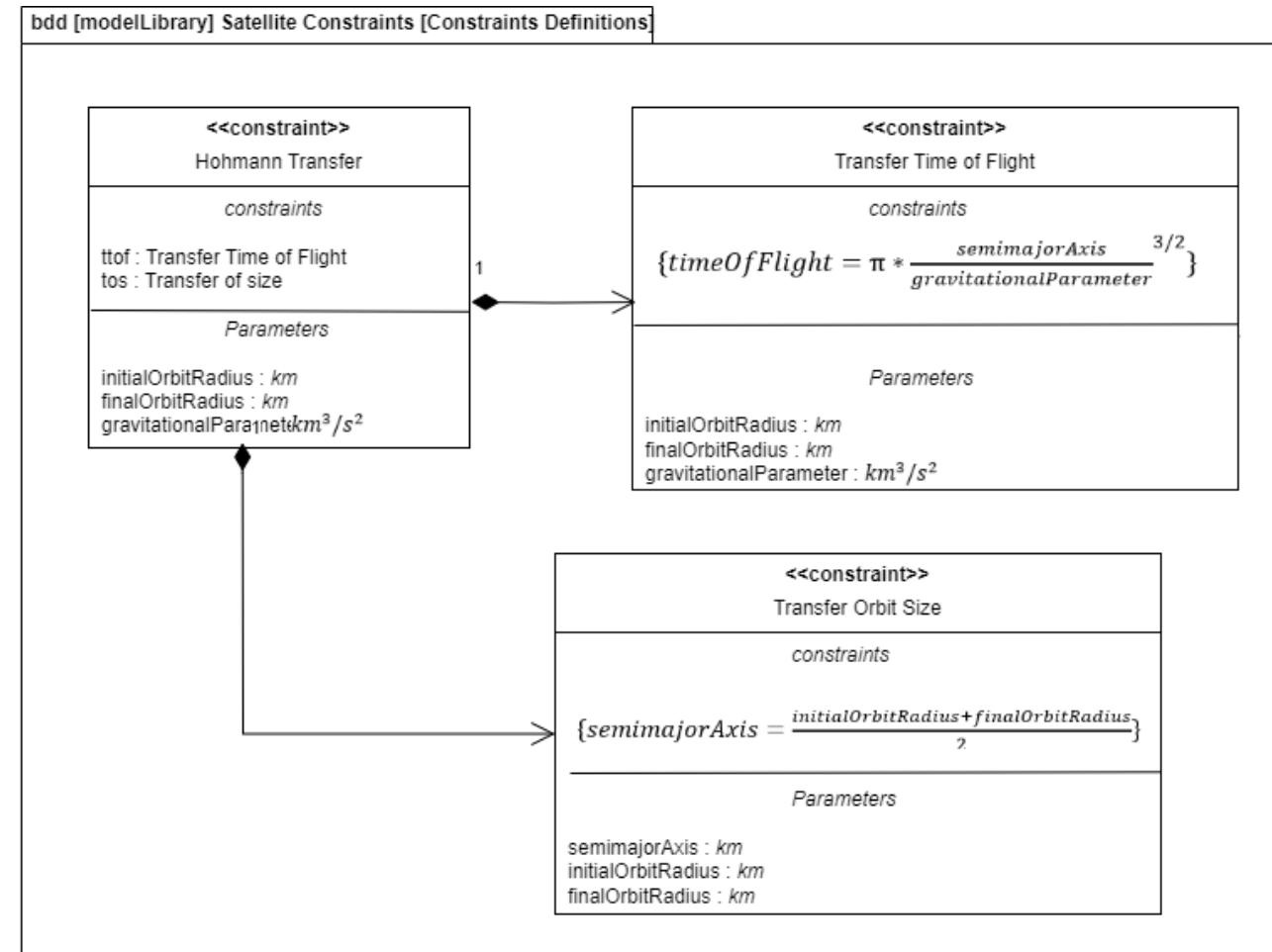


Fig: Relationships between constraint blocks

Internal Block Diagrams

Internal Block Diagram (IBD) Has a close relationship to the BDD.

- You can create an IBD to specify the internal structure of a single block.
- Unlike a BDD, an IBD does not display blocks; it displays usages of blocks, i.e part properties and reference properties.
- An IBD conveys how the parts of a block must be assembled to create a valid instance of the block.
- It shows how an instance of that block must be connected to external entities (reference properties) to create a valid instance of the system as a whole.

When should you create an IBD?

IBDs and BDDs provide complementary views of a block. A BDD lets you first define a block and its properties. Then you can use an IBD to display a valid configuration of that block – a specific set of connections among the block's properties. Because of this close relationship, you often create IBDs and BDDs for various stakeholders at different points in the system life cycle.

Blocks, revisited – an IBD of Communication and Data Handling Subsystem block

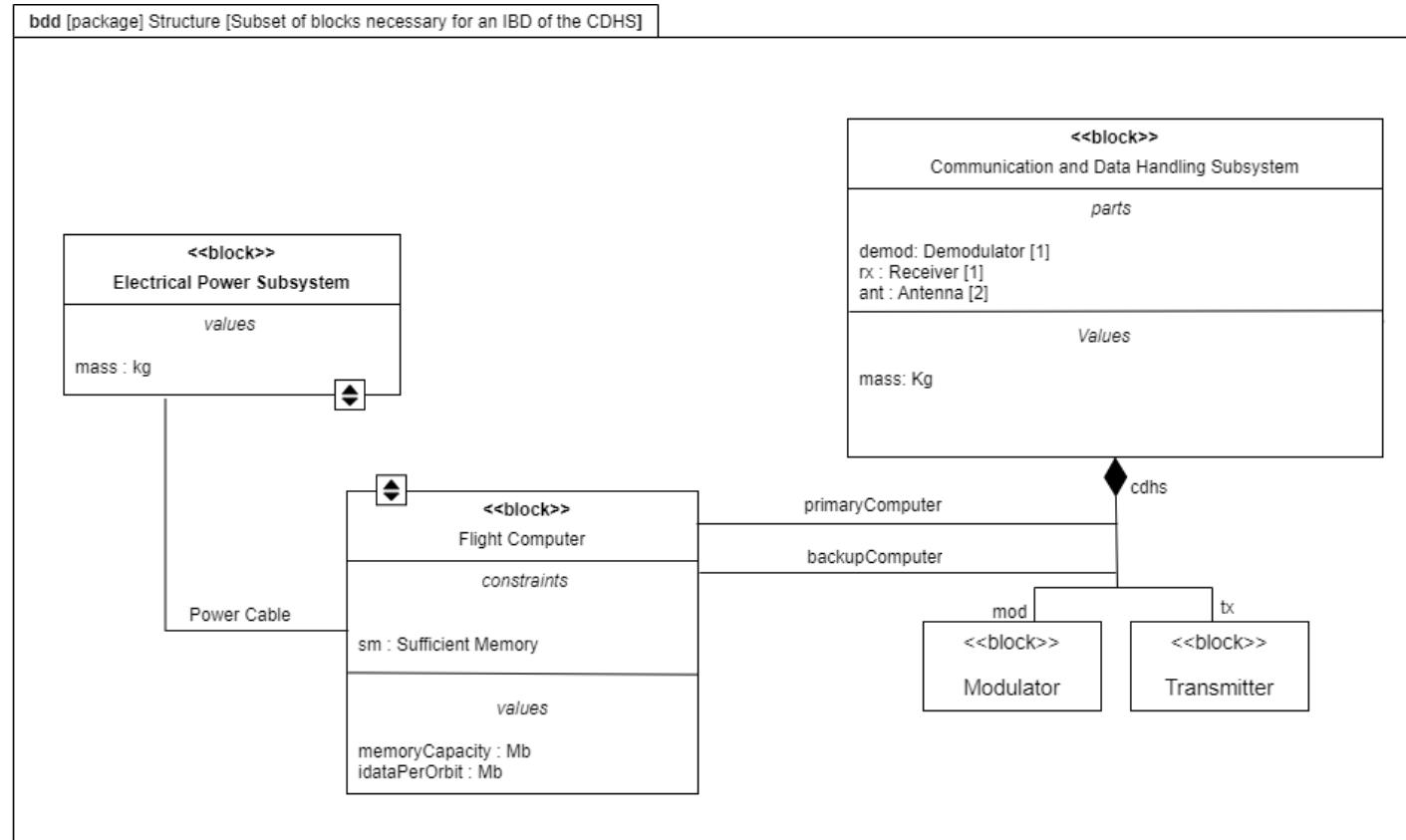


Fig: The blocks necessary for an IBD of the Communication and Data Handling Subsystem block

Module 3:
Requirement Analysis – Modelling



Monday, 02nd September



12:00 PM – 01:00 PM

Agenda:

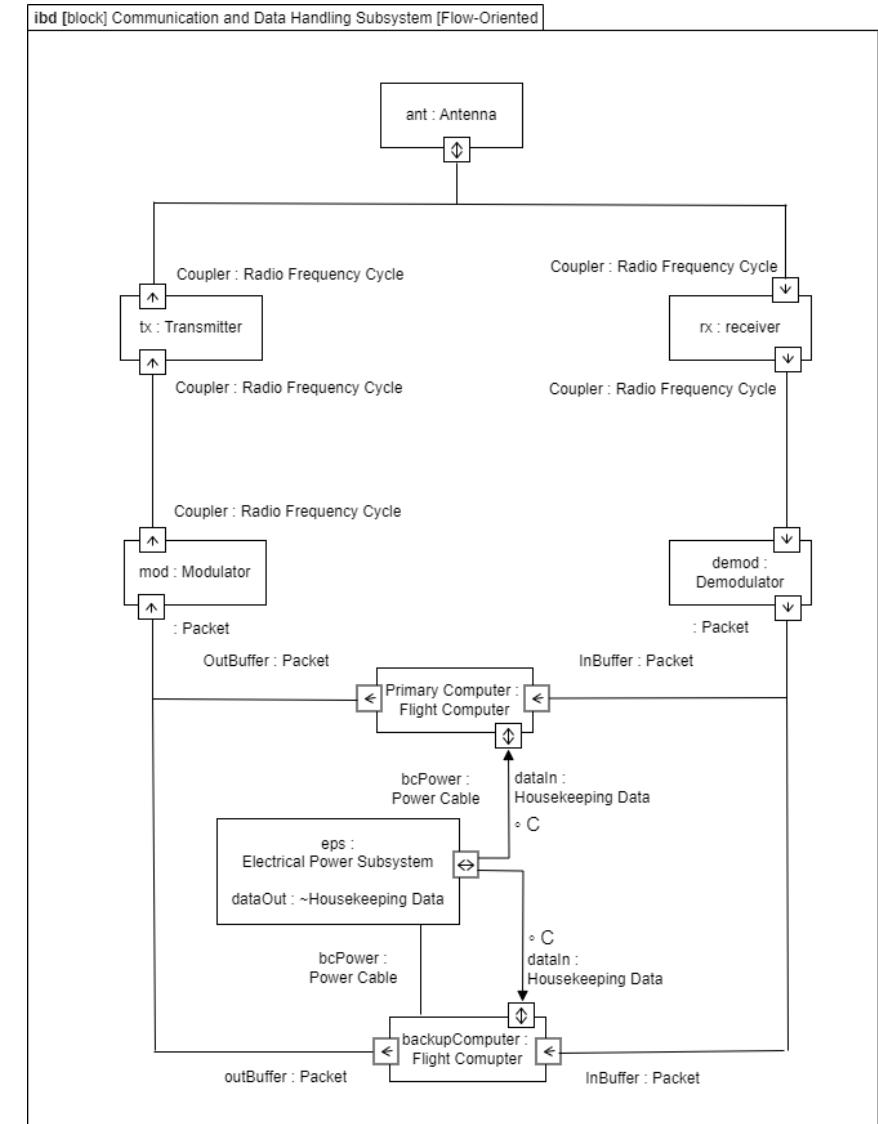
- Internal Block Diagrams
- Use Cases
- Use case Specifications
- Use Case Diagrams



SAHIL DATERO
Business Analyst, Delivery

Sample IBD Diagram

The BDD conveys that the Communication and Data Handling Subsystem block has seven Part Properties (Internal to block)
It has one reference property (system external to block)



IBD Diagram with a port on the frame

This diagram shows that these standard ports have a required interface, Light Source. This model conveys that the satellite's electrical power subsystem requires a light source, which it will access from the external environment via two solar panels on the satellite boundary.

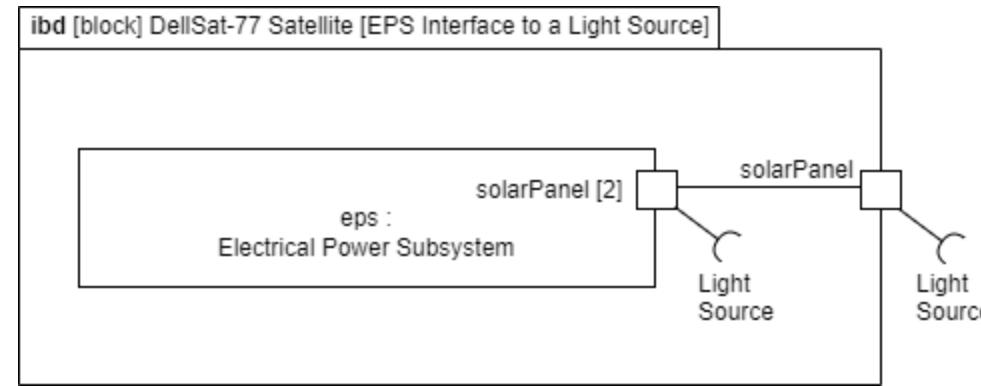


Fig: An IBD with a port on the frame

IBD Diagram with Dot Notation

Dot notation enables you to express structural hierarchy compactly in the form of a text string. An example is shown in the property at the top of the IBD, The string sensorPayload.xaxisSS : Star Sensor conveys several pieces of information:

- The DellSat-77 Satellite block owns a part property name sensor-Payload.
 - The part property sensorPayload, in turn, owns a property named x-axisSS
 - The property x-axisSS is typed by the block named Star Sensor.
 - The multiplicity of x-axis is 1..1 (the default, because multiplicity is shown

The Dot Notation have some drawbacks, The string `sensorPayload.x-axisSS` : Standard Sensor does not convey following pieces of information:

- The name of the block that types the part property sensorPayload
 - The multiplicity of the part property sensor Payload.

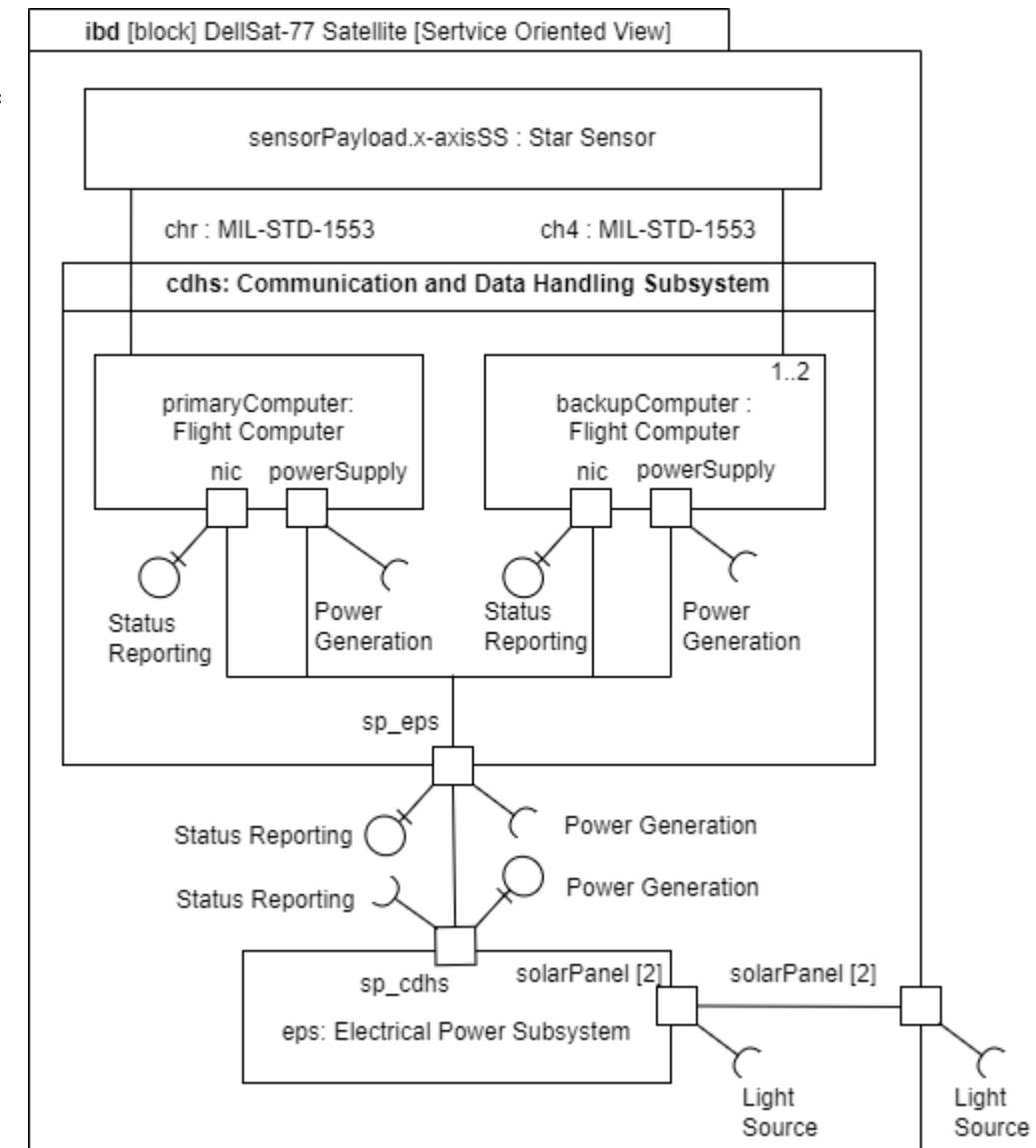


Fig: An IBD with nested Properties

Use Case Diagrams

A use case diagram concisely conveys a set of use cases – the externally visible services that a system provides – as well as the actors that involve and participate in those use cases.

A use case diagram is a block box view, it is therefore well suited to serve as a system context diagram.

When should you create a Use Case Diagram?

A use case diagram is an analysis tool and is generally created early in the system life cycle. System analysts may enumerate use cases and create use case diagrams during the development of the system concept of operations (ConOps).

In some methodologies, analysts create use cases in form of text-based functional requirements during the requirements elicitation and specification stage of the system life cycle. System Architects later analyse system-level use cases to derive and allocate subsystem and component level use cases during the architectural design stage.

Wait! What's a Use Case?

“Specification of sequences of actions, including variant sequences and error sequences, that a system, subsystem or class can perform by interacting with outside objects to provide a service of value.”

The UML Reference Manual, James Rumbaugh, Ivar Jacobson, and Grady Booch define a Use case as a

“A use case captures, a contract between the stakeholders of a system about its behavior.

The Use case describes the system’s behavior under various conditions as it responds to a request from one of the stakeholders, called the primary actor.

The primary actor initiates an interaction with the system to accomplish some goal.

The system responds, protecting the interests of all the stakeholders.

Different sequences of behavior, or scenarios, can unfold, depending on the particular requests made and the conditions surrounding the requests.

The use case gathers those different scenarios together.

Alistair Cockburn explains,

Key ideas to keep in mind when you're enumerating your system's use cases:

- A use case is a service – a behavior – that your system will perform. The use case name, therefore is always a verb phrase (such as ‘Send Command’)
- Not every behavior your system performs is a use case. Rather, use cases are the subset of system behaviours that external actor can directly invoke or participate in
- An actor can be a person or an external system that interfaces with your system.
- The actors that invoke a use case are called primary actors. The actors that participate in the use case are called secondary actors.
A primary actor can also be a secondary actor.
- Each use case should represent a primary actor’s goal. Write the use case name – the verb phrase – from the perspective of the actor and not your system.
- The use case name does not convey a lot of information. You will create a use case specification for each use case to narrate how your system and its actors collaborate to achieve the use case goal.

Use case specifications, unfolding narrative when primary actor invokes the use case

- Use case name: A verb Phrase
- Scope: The entity that owns (provides) the use case (for example, the name of an organization, system, subsystem, or component)
- Primary actor: The actor that invokes the use case (the actor whose goal the use case represents)
 - The actors that invoke a use case are called primary actors. The actors that participate in the use case are called secondary actors.
A primary actor can also be a secondary actor.
- Stakeholder: Someone or something with a vested interest in the behaviour of the system.
- Pre-conditions: The conditions that must be true for this use case to begin
- Guarantees (Postconditions) : The conditions that must be true at the end of the use case
- Trigger : the event that gets the use case started
- Main success scenario: The scenario (the sequence of steps) in which nothing goes wrong
- Extensions (alternative branches): Alternatives sequences of steps branching off the main success scenario.
- Related Information : Whatever your project needs for additional information.

Use case diagram

System Boundary

The notation for the system boundary is a rectangle.

Actors

There are two notations for an actor:

1) A stick figure

2) A rectangle with keyword <>actor>>

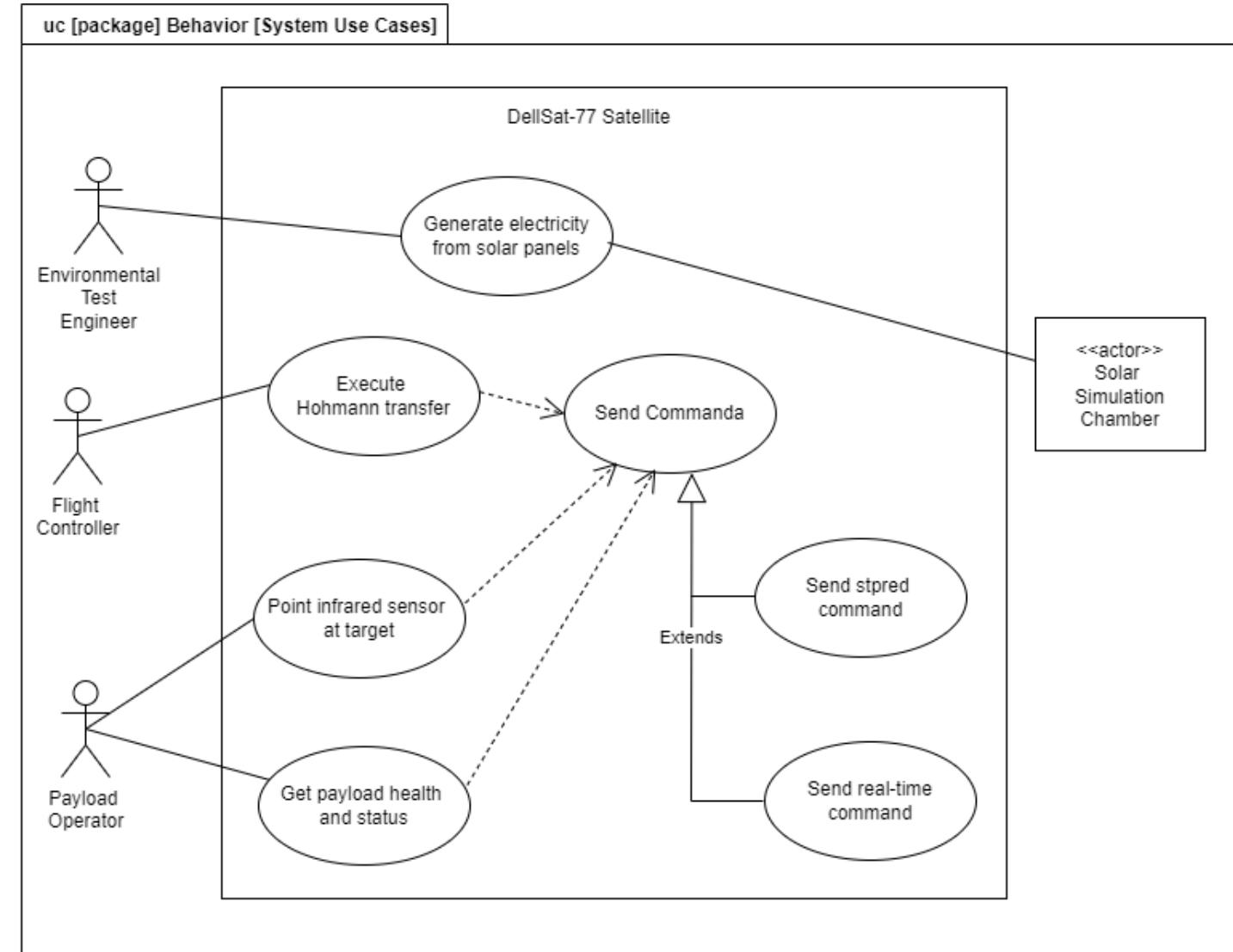
Associating Actors with use cases

- You cannot create a composite association between an actor and a use case

- You cannot create an association (of any kind) between two actors

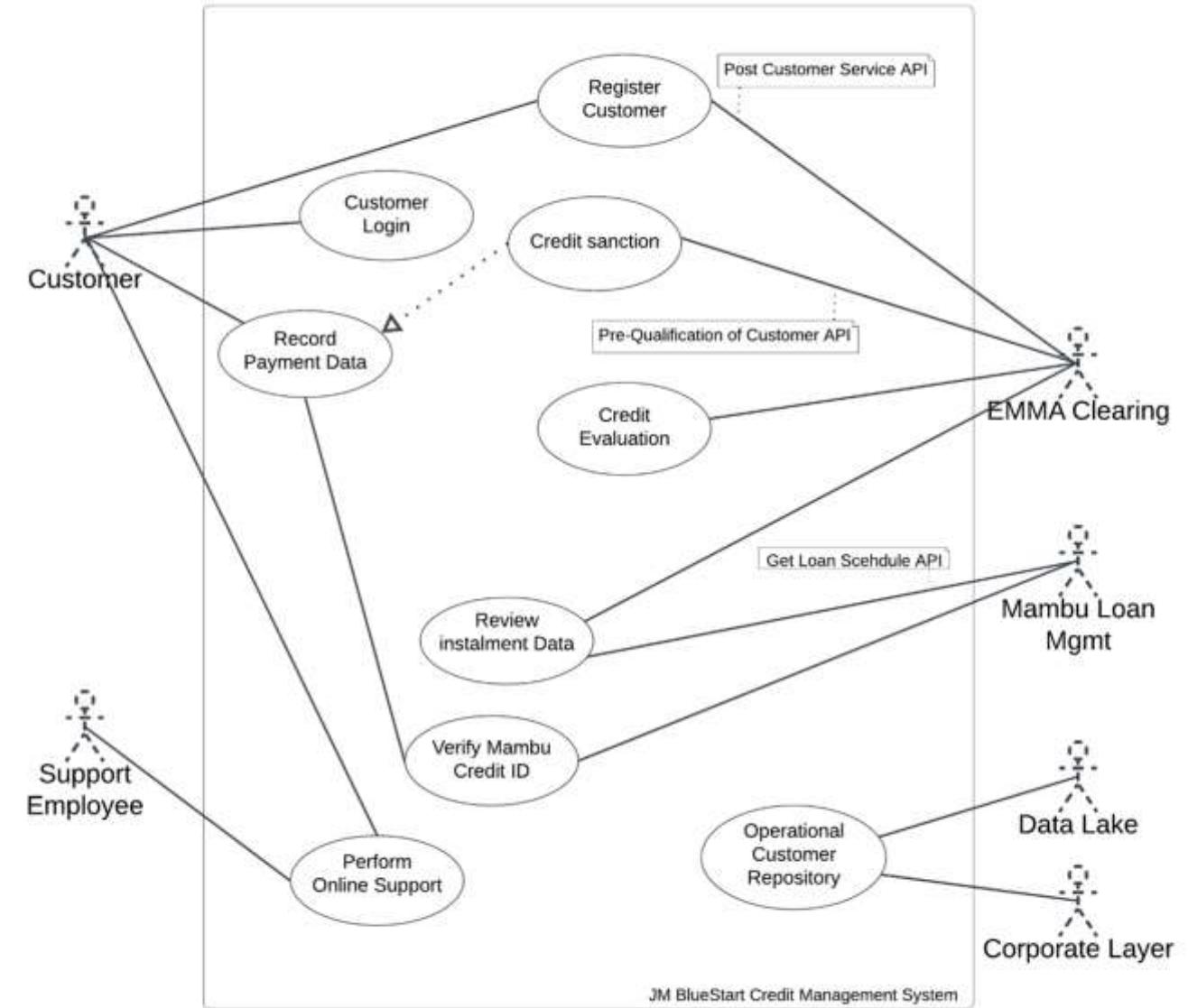
- You cannot create an association (of any kind)

- Between two use cases



Use case diagram of JM Bluestart Project

- Use case name:
- Scope:
- Primary actor:
- Stakeholder:
- Pre-conditions:
- Guarantees (Postconditions) :
- Trigger :
- Main success scenario:
- Extensions (alternative branches):
- Related Information :



Module 3:
Requirement Analysis – Modelling



Monday, 03rd September



09:00 AM – 10:00 AM

Agenda:

- Activity Diagram



SAHIL DATERO
Business Analyst, Delivery

Purpose of Activity Diagram

An activity diagram is kind of behavior diagram; it's dynamic view of the system that expresses sequences of behaviours and event occurrences over time.

This is in contrast to structure diagrams (BDDs, IBDs, and parametric diagrams), which are static views that convey no sense of the passage of time or change within the system and its environment.

An activity diagram is particularly good at expressing the flow of objects—matter, energy, or data—through a behavior, with a focus on how the objects can be accessed and modified in an execution of that behavior during system operation. And activity diagrams are uniquely capable of expressing continuous system behaviors.

Activity diagrams express the order in which actions are performed, and they can optionally express which structure performs each action. They do not, however, offer any mechanism to express which structure invokes each action.

Sample Activity Diagram

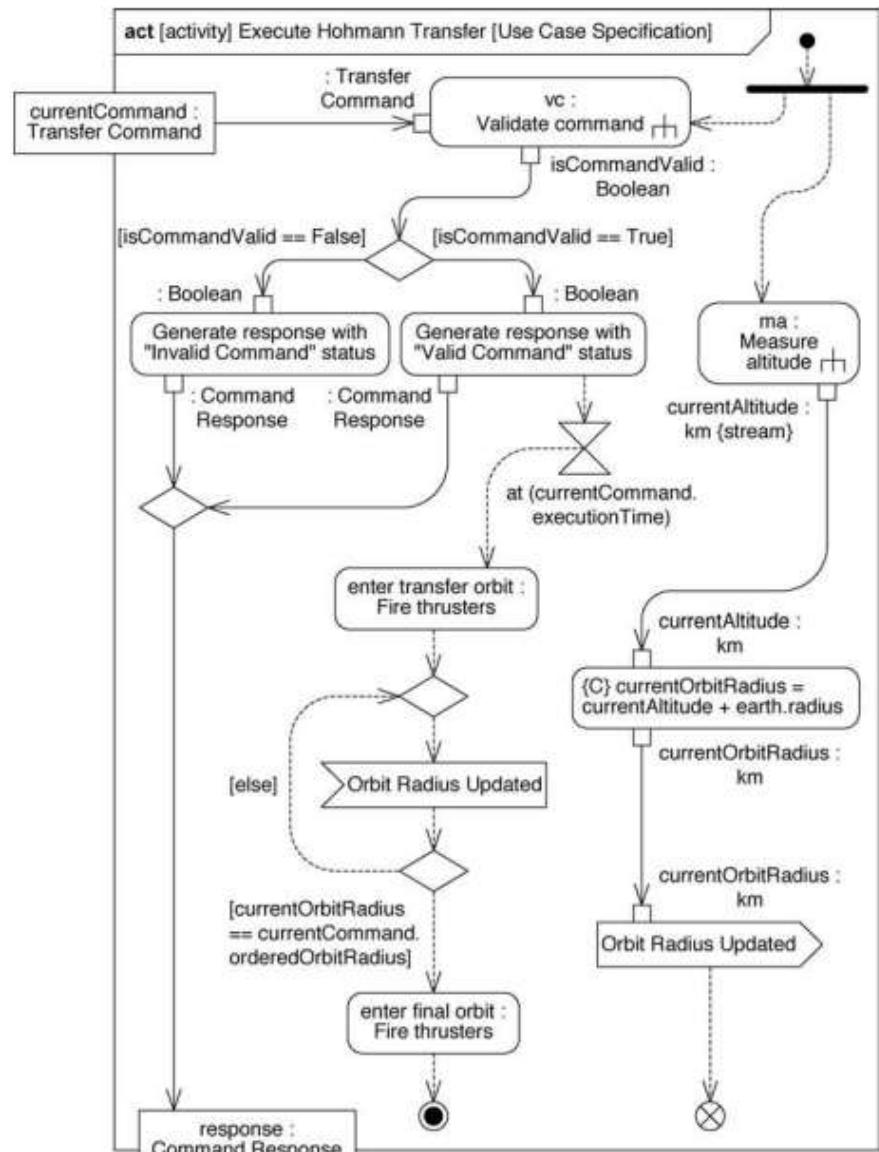
When should you create activity diagram?

When you need to communicate with stakeholders and capture the expected behaviors of the system and its actors and capture the expected behaviors of the system's internal parts.

Activity Diagram Frame

An activity diagram is particularly good at expressing the flow of objects—matter, energy, or data—through a behavior, with a focus on how the objects can be accessed and modified in an execution of that behavior during system operation. And activity diagrams are uniquely capable of expressing continuous system behaviors.

Fig: Sample Activity Diagram



Activity Diagram – Actions and nodes

Actions – the basics

An action is one kind of node that can exist within an activity; it's a node that models a basic unit of functionality within the activity



Fig: Actions with natural language expressions

Object Node

An object node, another kind of node that can exist within an activity, models the flow of object tokens through an activity (where each object token, again, represents an instance of matter, energy, or data). An object node most often appears between two actions to convey that the first action produces object tokens as outputs, and the second action consumes those object tokens as inputs

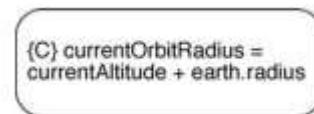


Fig: Actions with opaque language expressions

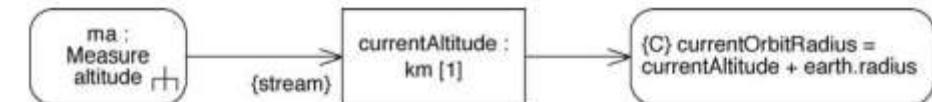


Fig: Object Nodes between two actions

Activity Diagram – Pins and Activity Params

Pins

A pin is a specialized kind of object node. You attach a pin to an action to represent an input or output of the action. The notation for a pin is a small square attached to the boundary on the outside of an action.

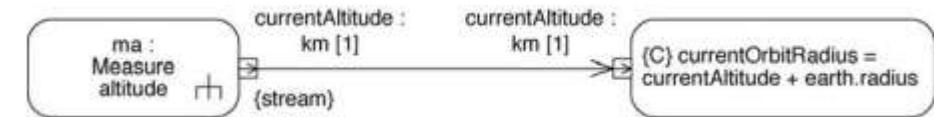


Fig: Actions with Pins

Activity Parameters

An activity parameter is another specialized kind of object node. You attach it to the frame of an activity diagram to represent an input or an output of the activity as a whole

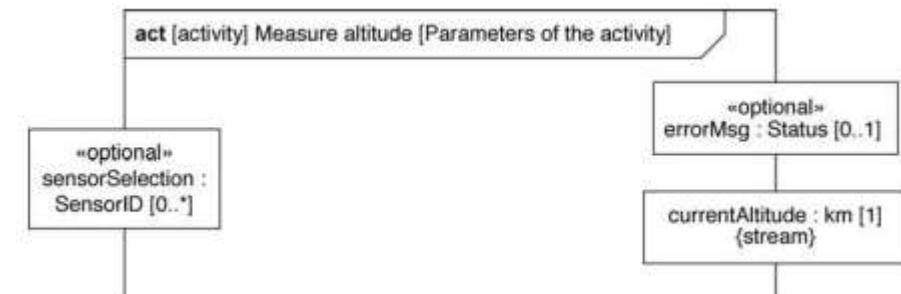


Fig: An activity diagram frame with activity parameters attached

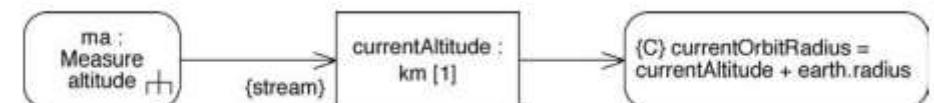


Fig: Object Nodes between two actions

Activity Diagram – Object Flows and Object Nodes

Object Flows

An object flow is the kind of edge that transports object tokens.

You use object flows to convey that instances of matter, energy, or data flow through an activity from one node to another when the activity executes during system operation

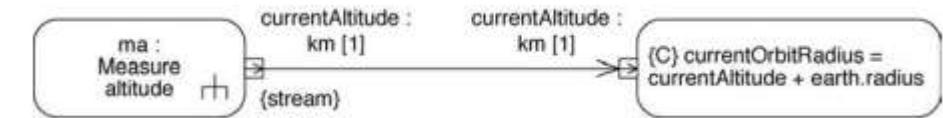


Fig: An activity diagram frame with activity parameters attached

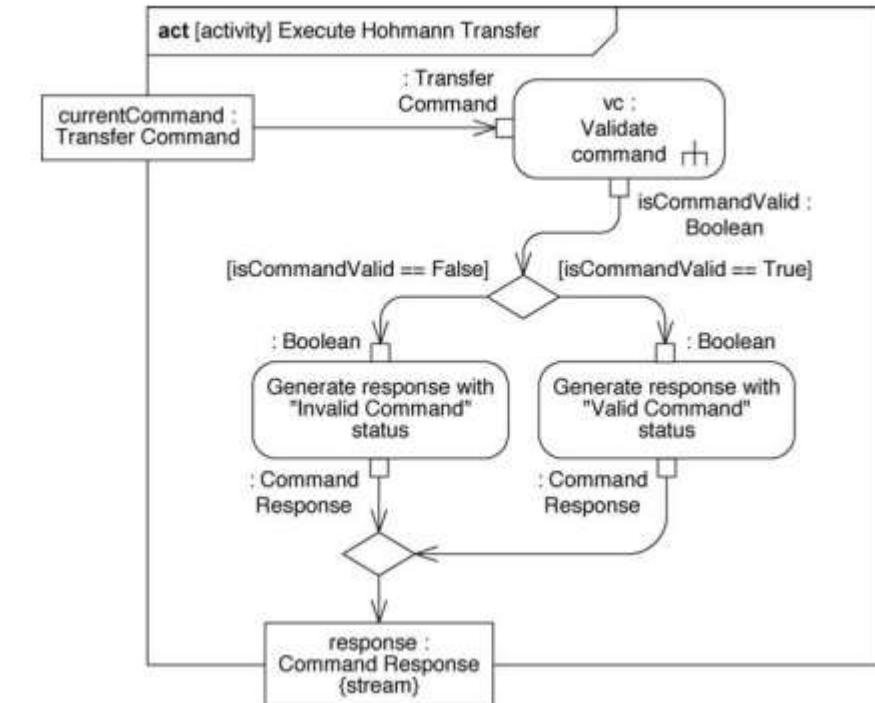


Fig: Object flows

Activity Diagram – Flows and Actions

Control Flows

A control flow is the kind of edge that transports control tokens. And the arrival of a control token enables an action that's waiting for one.

Actions

Three conditions must be satisfied for an action to start:

- The activity that owns the action is currently executing
- A control token arrives on each of the incoming control flows.
- A sufficient number of object tokens arrive on each of the incoming object flows to satisfy the lower multiplicity of the respective input pin.

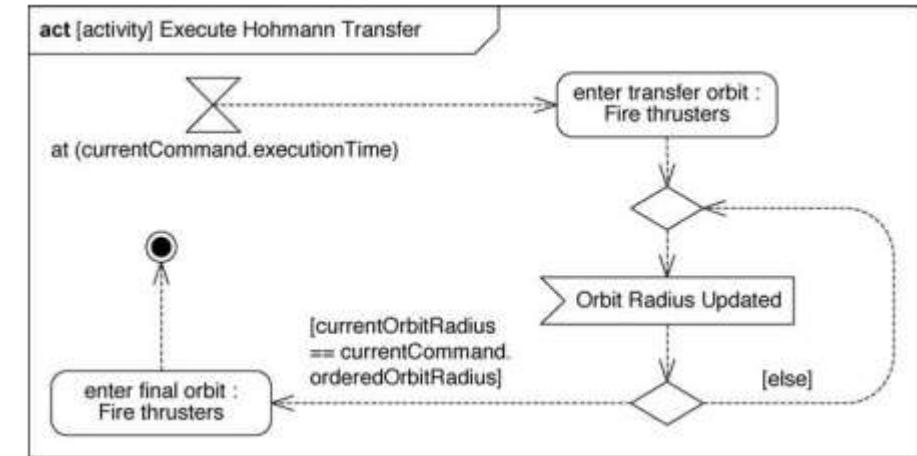


Fig: Control flows

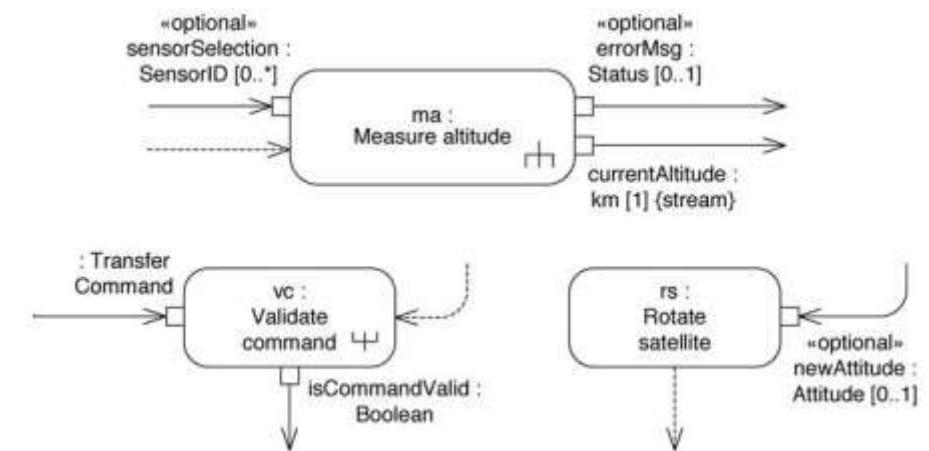


Fig: Actions with incoming edges

Nodes: Decision and Merge

Decision Nodes

A decision node marks the start of alternative sequences in an activity. The notation is a hollow diamond. A decision node must have a single incoming edge and generally has two or more outgoing edges. Each outgoing edge is labeled with a Boolean expression called a guard, which is displayed as a string between square brackets (e.g., `isCommandValid == False`)

Merge Nodes

A merge node marks the end of alternative sequences in an activity. The notation is the same as the one for a decision node: a hollow diamond. You can distinguish one from the other by the number of incoming and outgoing edges; a merge node has two or more incoming edges and a single outgoing edge. When a token—either an object token or a control token—arrives at a merge node via any of its incoming edges, the token is immediately offered to the outgoing edge

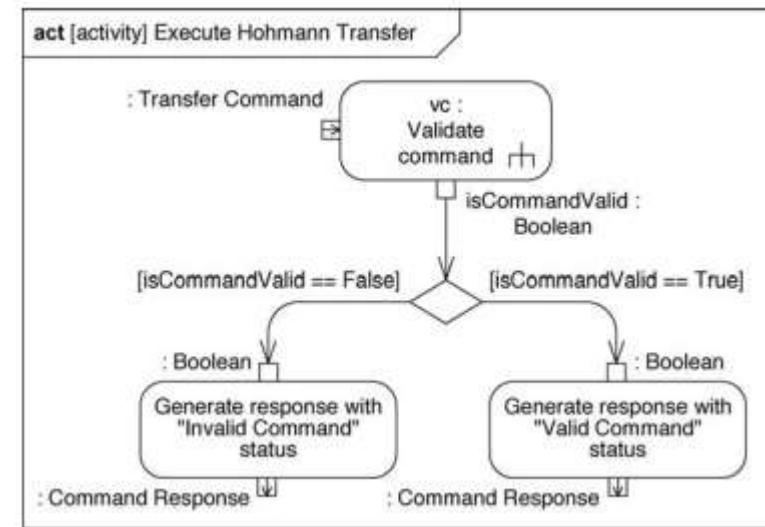


Fig: An activity fragment containing a decision node

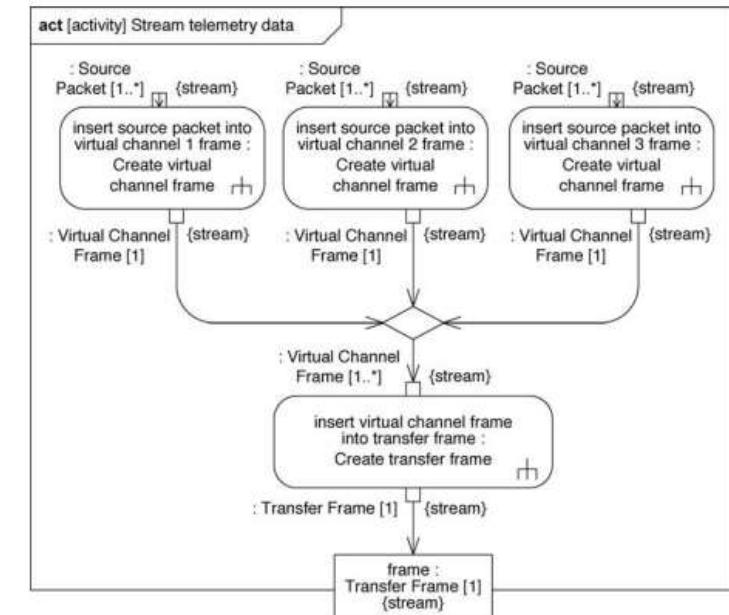


Fig: Using a merge node to model the interleaving of token

Dataflow Sample Activity Diagram

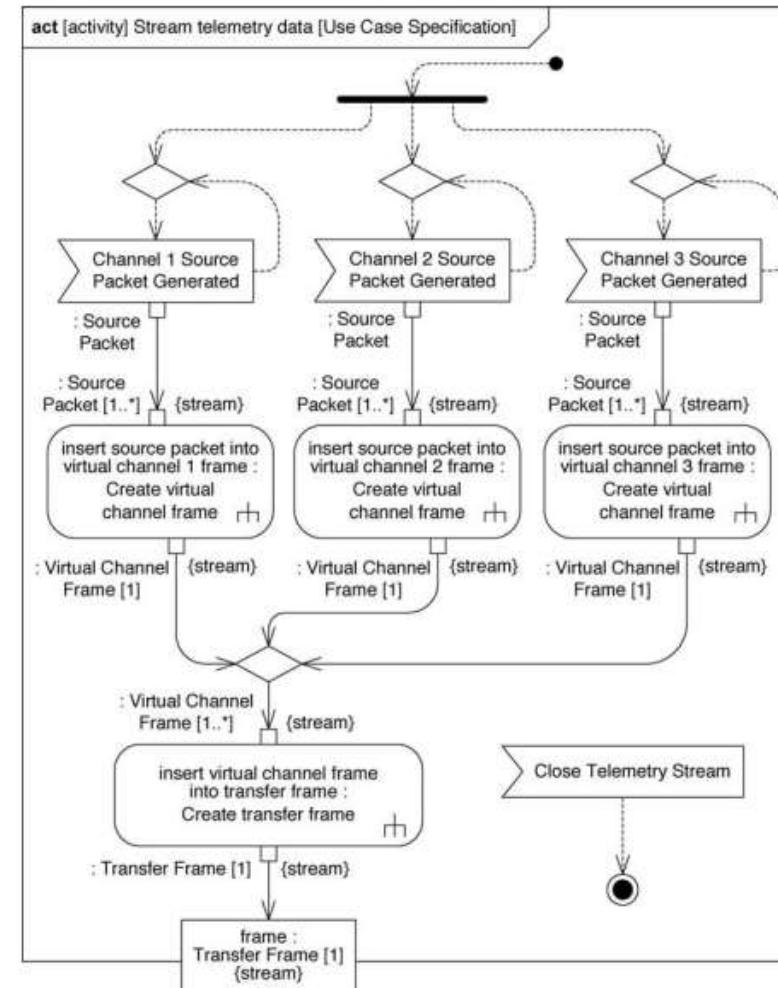


Fig: Dataflow of telemetry data

Allocating Behavior to structure

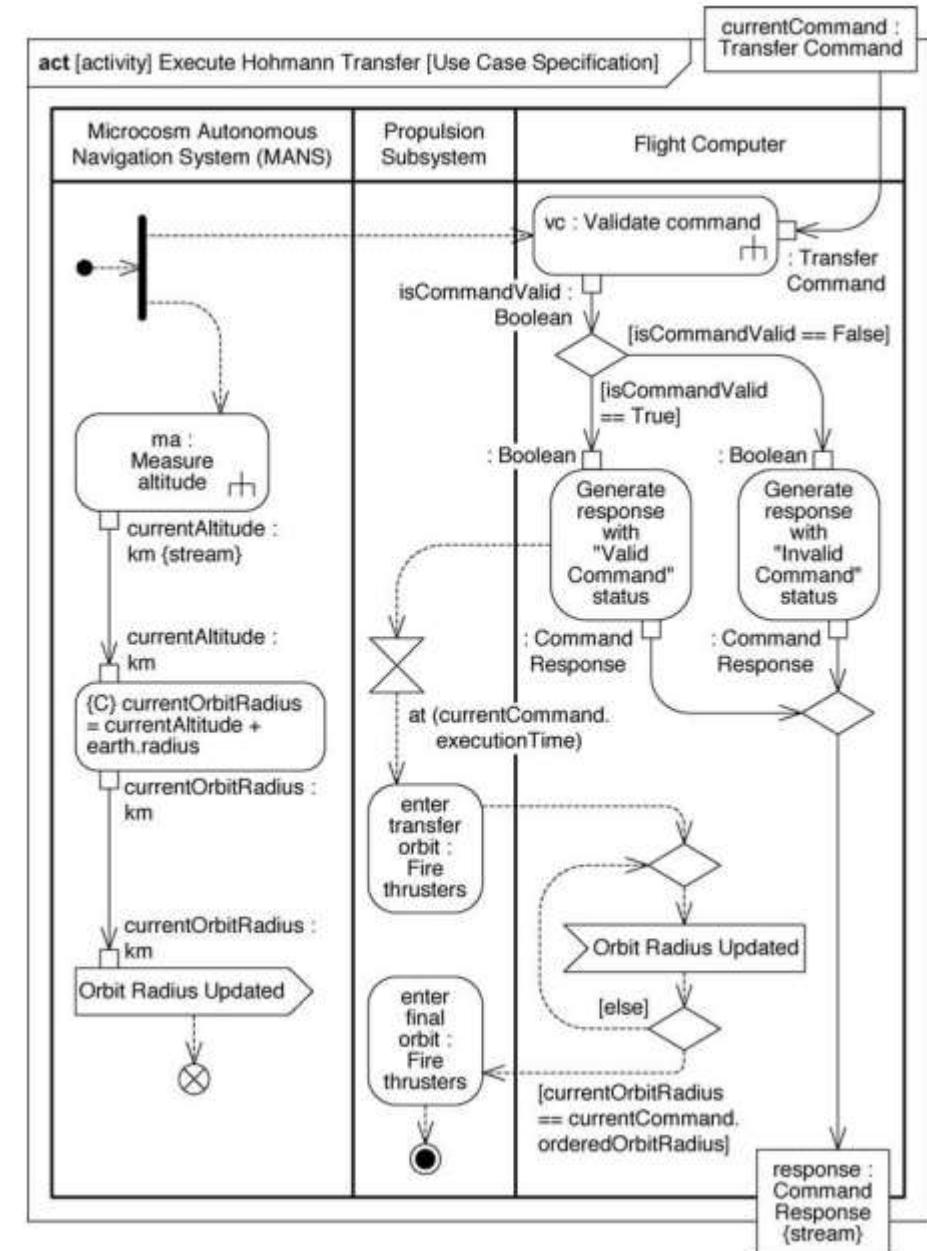


Fig: Using activity partitions to allocate action to structure

Module 3:
Requirement Analysis – Modelling



Monday, 04th September



09:00 AM – 10:00 AM



SAHIL DATERO
Business Analyst, Delivery

Agenda:

- Sequence Diagram – Purpose, uses
- Sequence Diagram - Frame

Purpose of Sequence Diagram

- Sequence diagrams are a second kind of SysML diagram that you can use to express information about a system's dynamic behavior.
- You can use elements called lifelines to model the participants in a system behavior and then use messages between lifelines to model interactions among those participants.
- You can specify time constraints and duration constraints on interactions.
- You can also use various kinds of interaction operators to steer the execution of an interaction. Interaction uses let you model behavioral decomposition among a set of interactions.
- A sequence diagram is a good choice for specifying a behavior when you want the focus to be on how the parts of a block interact with one another via operation calls and asynchronous signals to produce an emergent behavior. That behavior is formally called an interaction.
- A sequence diagram is a precise specification of a behavior. It is therefore well suited to serve as a detailed design artifact—an input into development.
- This is possible because sequence diagrams convey the three requisite pieces of information needed to automate a transformation into source code:
 - The order in which behaviors are performed,
 - Which structure performs each behavior, and
 - which structure invokes each behavior.

When should you create a sequence diagram?

- You can create one to specify a behavior at any level in the system hierarchy.
- Sequence diagrams are useful early in the life cycle during ConOps development to specify the intended interactions between the system of interest and the actors in its environment.
- You can use sequence diagram instead of traditional text based test case specification, that sometimes is in parallel with requirement specification activity

The Sequence Diagram Frame

- Interactions - The only allowable model element type for a sequence diagram
- Like an activity, a block, and a package, an interaction is also a kind of namespace. It can therefore contain a set of named elements (such as lifelines, event occurrences, and messages) within the model hierarchy.

The Sequence Design Frame

Lifelines

A lifeline is an element that represents a participant in an interaction

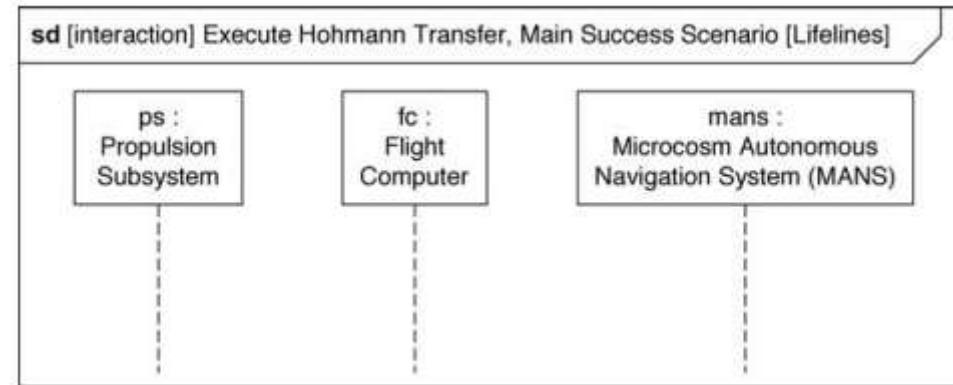


Fig: Lifelines

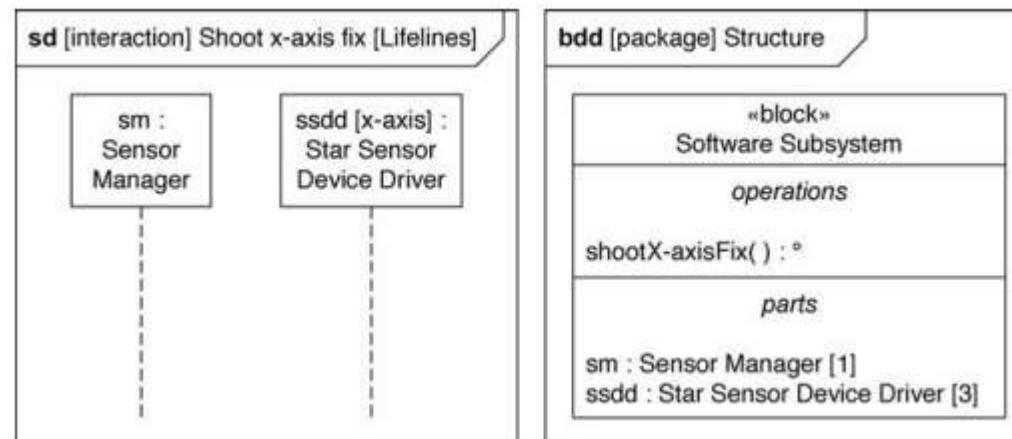


Fig: lifeline with a selector expression

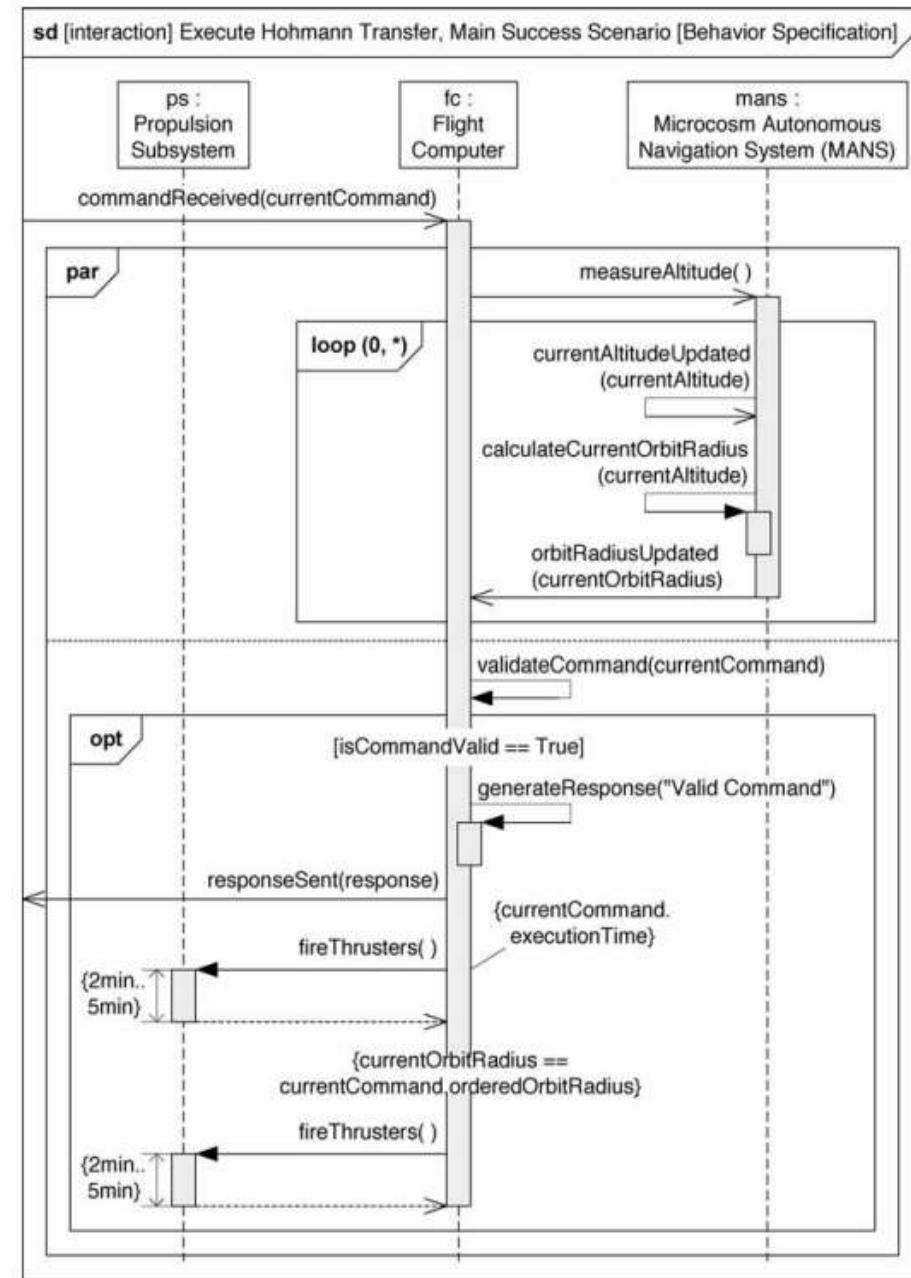


Fig: A Sample Sequence Diagram

The Sequence Design Frame - Lifelines

The set of lifelines in an interaction convey ordered sequences of event occurrences, and those sequences form the narrative of the interaction.

There are six kinds of event occurrences that can appear on lifelines:

- Message send occurrences
- Message receive occurrences
- Lifeline creation occurrences
- Lifeline destruction occurrences
- Behavior execution start occurrences
- Behavior execution termination occurrences

A message represents a communication between a sending lifeline and a receiving lifeline.



The Sequence Design Frame - Messages

Message and Message Occurrences

Message Types

① Asynchronous Message:

An asynchronous message represents a communication between a sending lifeline and a receiving lifeline wherein the sender immediately proceeds with its own execution after sending the message.

<message name> (<input argument List>)

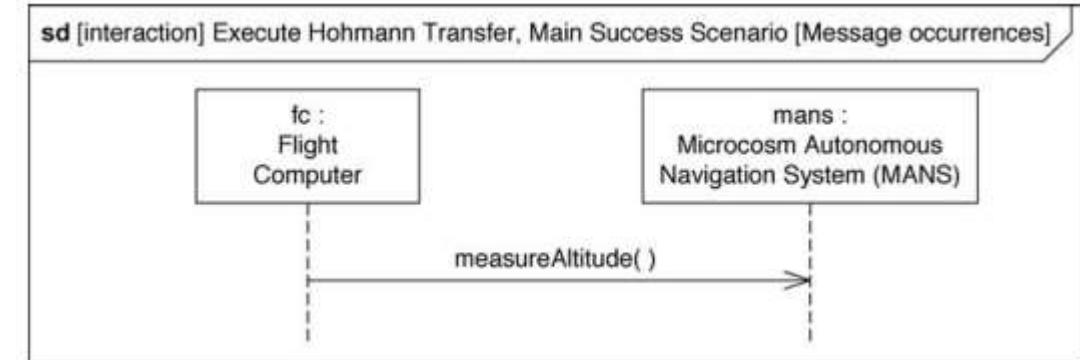


Fig: A message send occurrence and a message receive occurrence

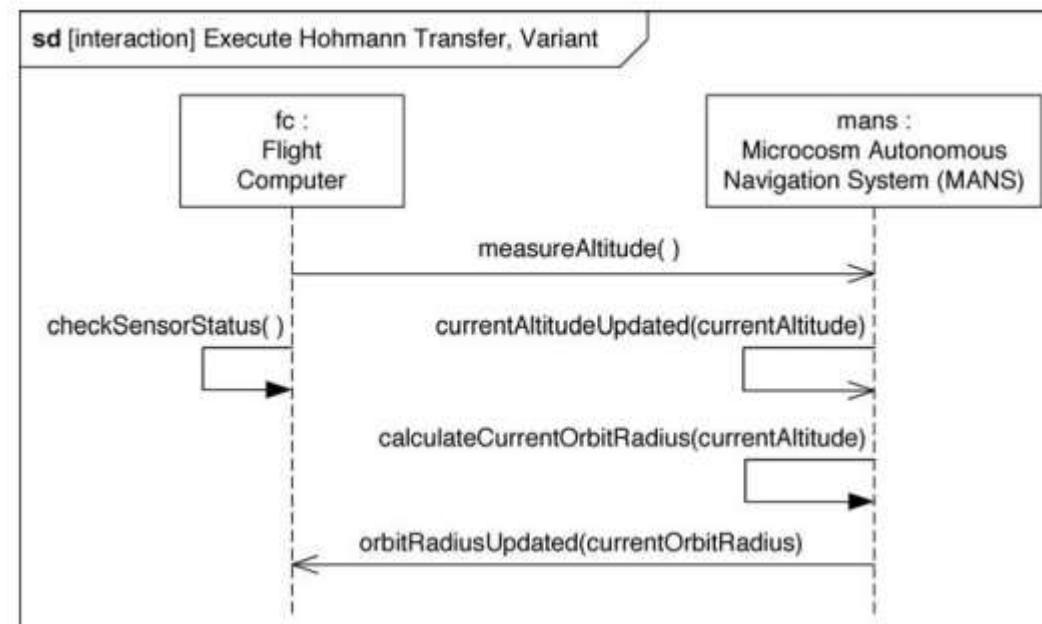


Fig: Asynchronous messages in an interaction

The Sequence Design Frame - Messages

□ Message and Message Occurrences

② Synchronous Message:

A synchronous message represents a communication between a sending lifeline and a receiving lifeline wherein the sender waits for the receiver to finish executing the invoked behavior and send a reply message before the sender can proceed with its own execution.

<Message name> (<input argument list>)

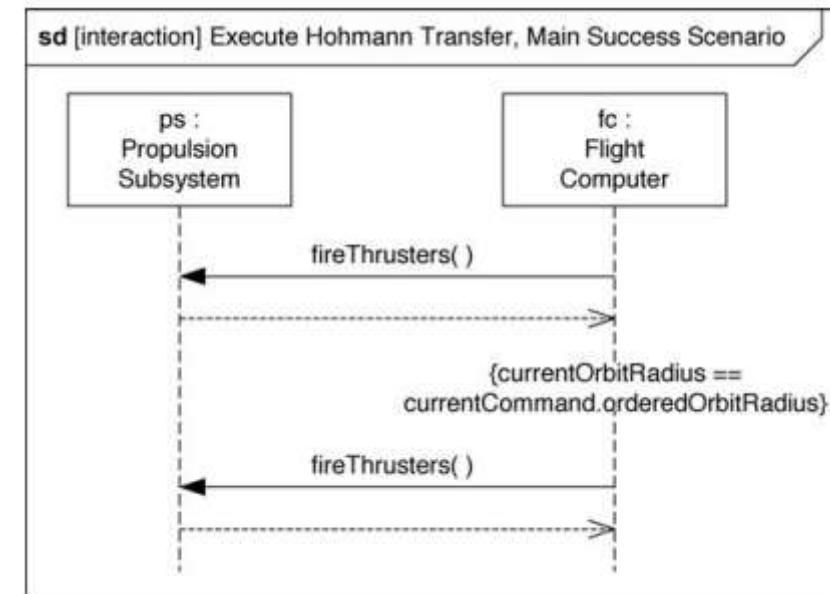


Fig: Synchronous messages in an interaction

The Sequence Design Frame - Messages

□ Message and Message Occurrences

③ Reply Message:

A reply message represents a communication that marks the end of a synchronously invoked behavior. It's always sent from the lifeline that performed the behavior to the lifeline that invoked the behavior (via a synchronous message earlier in the interaction)

That label has the following format: <Assignment target> = <message name>

(<output argument list>) : <value specification>

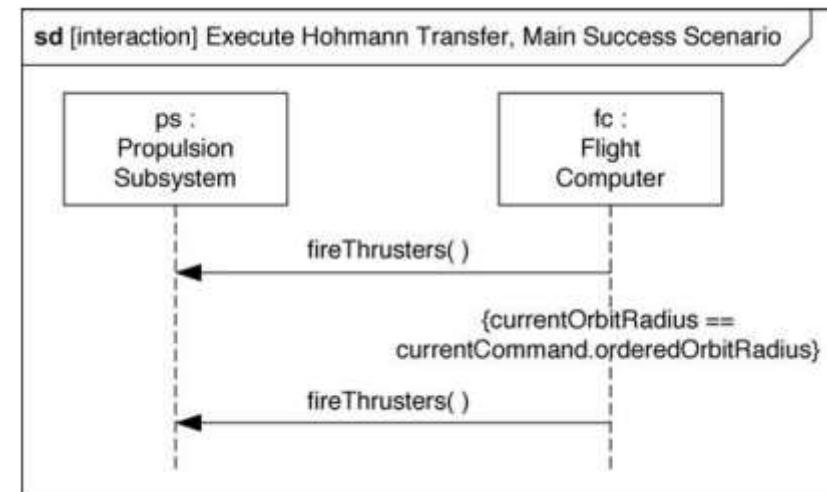


Fig: Synchronous messages with implicit reply messages

The Sequence Design Frame - Messages

□ Message and Message Occurrences

④ Create Messages:

A create message represents a communication that creates a new instance within a system—an instance that then participates in the interaction.

That label has the following format: <Assignment target> = <message name>

(<output argument list>) : <value specification>

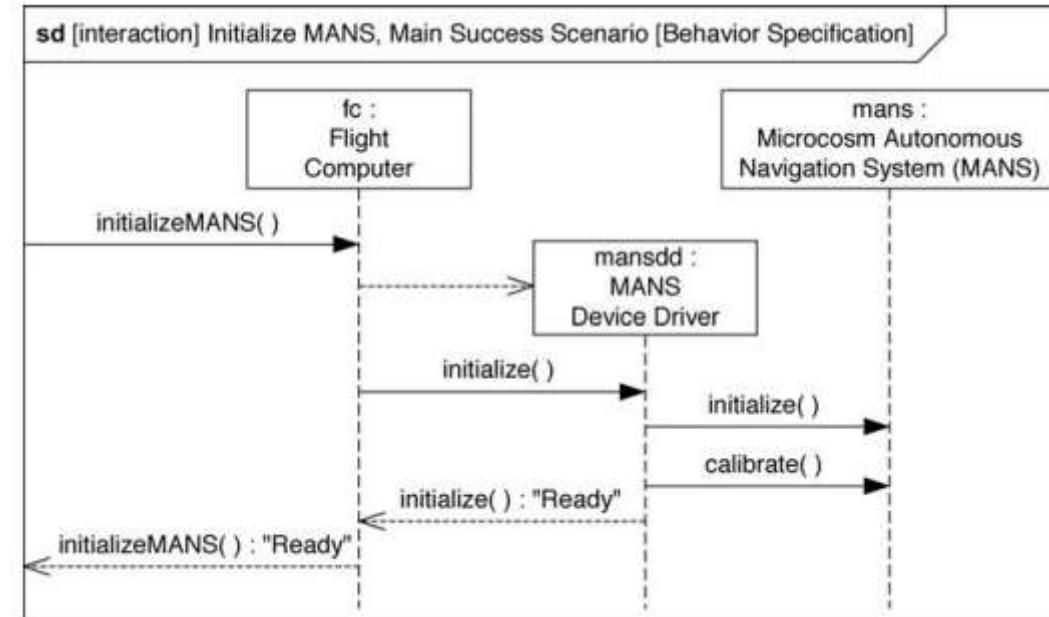


Fig: A create message in an interaction

The Sequence Design Frame – Lifeline Creation

Lifeline Creation

. A lifeline creation occurrence exists on a lifeline at the point where the arrowhead end of a create message meets the head of that lifeline. (A message receive occurrence also exists at that point; it is therefore coincident with the lifeline creation occurrence.)

Lifeline Destruction

The fourth kind of event occurrence that can appear on a lifeline is a lifeline destruction occurrence (or destruction occurrence, for short). A destruction occurrence represents the termination of a lifeline and the destruction of the instance within a system that the lifeline represents.

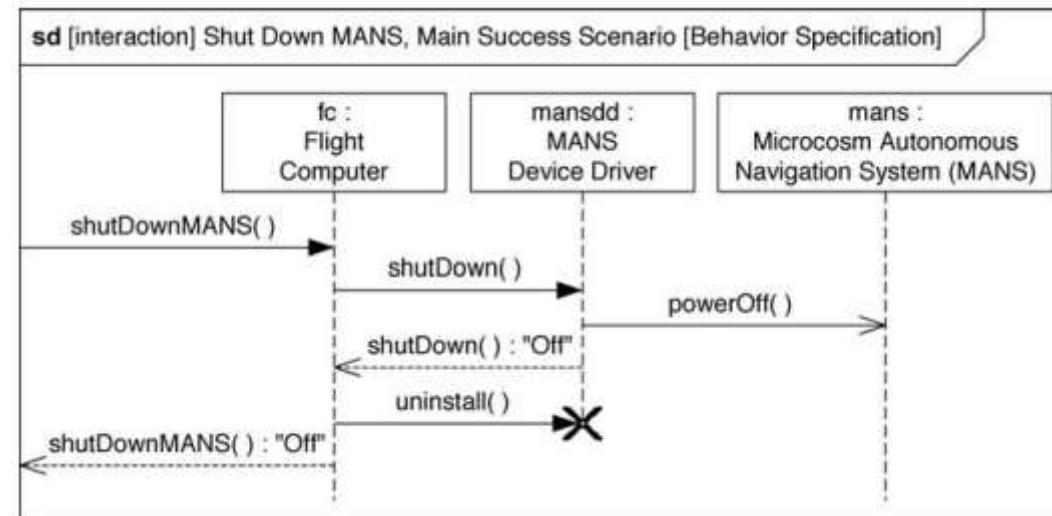


Fig: A destruction occurrence in an interaction

The Sequence Design Frame – Execution Specifications

The last two kinds of event occurrences that can appear on a lifeline are behavior execution start occurrences and behavior execution termination occurrences. A behavior execution start occurrence is generally implicit at the point where a **lifeline** receives a synchronous or asynchronous message. A behavior execution termination occurrence is generally implicit at the point where a lifeline sends a reply message

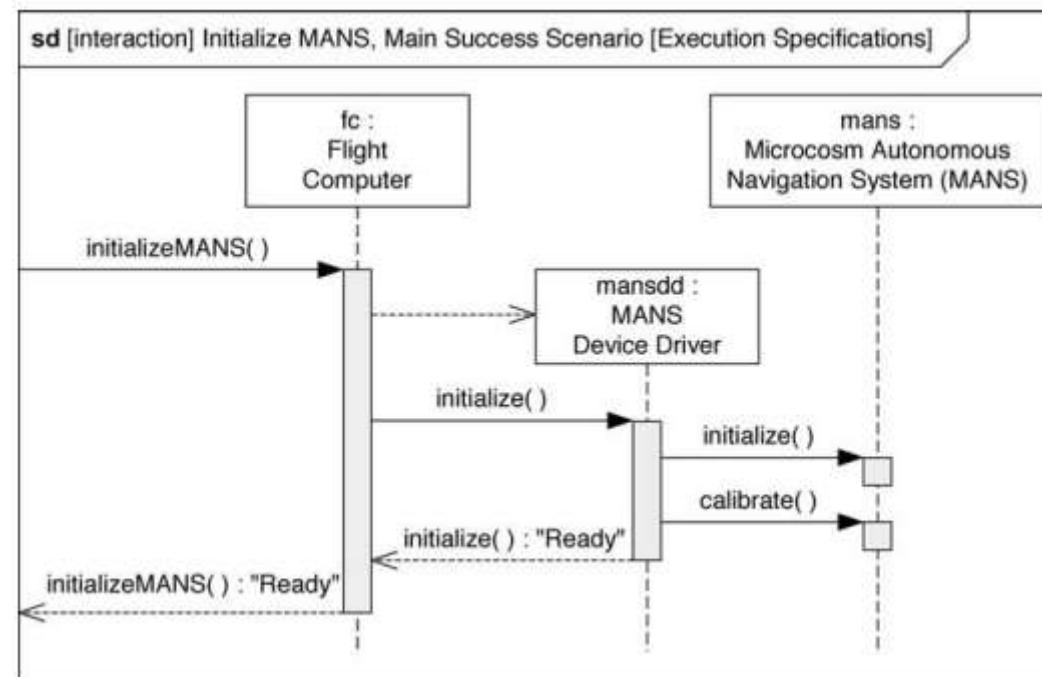


Fig: An interaction with execution specifications displayed

The Sequence Design Frame – Constraints

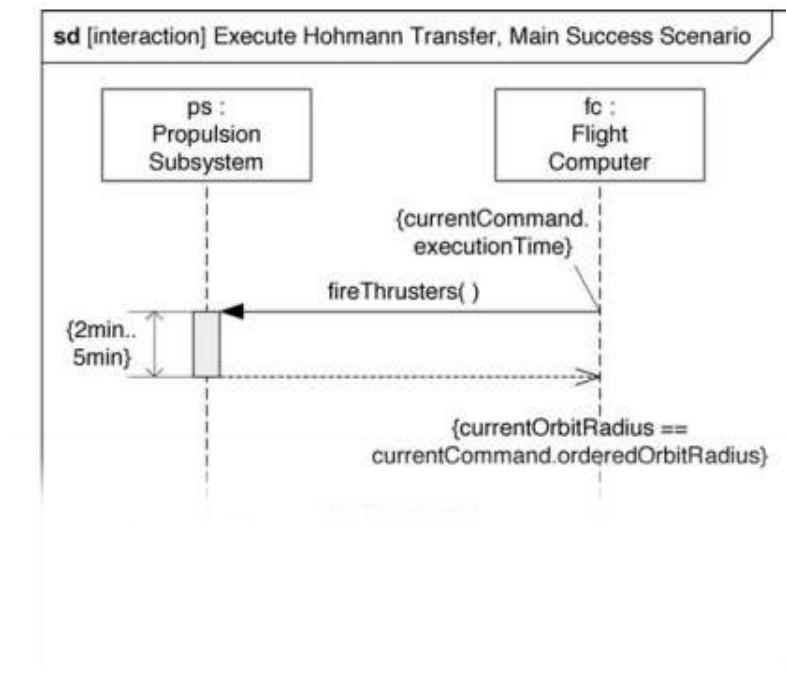
In the context of interactions, there are three kinds of constraints you will use in daily practice:

- Time constraints,
- Duration constraints, and
- State invariants

① Time Constraints

A time constraint specifies a required time interval for a single event occurrence. That time interval may be a single time value (that is, min. and max. are equal) or even a property that holds a time value.

The key idea here is, When the interaction executes during system operation, it's considered to be a valid execution only if that event occurrence happens within the time interval specified by the time constraint



The Sequence Design Frame – Constraints

② Duration Constraints

A duration constraint specifies a required time interval for a pair of event occurrences. Again, that time interval may be a single time value or a property that holds a time value.

The key idea here is, When the interaction executes during system operation, it's considered a valid execution only if the lapse between that pair of event occurrences falls within the time interval specified by the duration constraint.

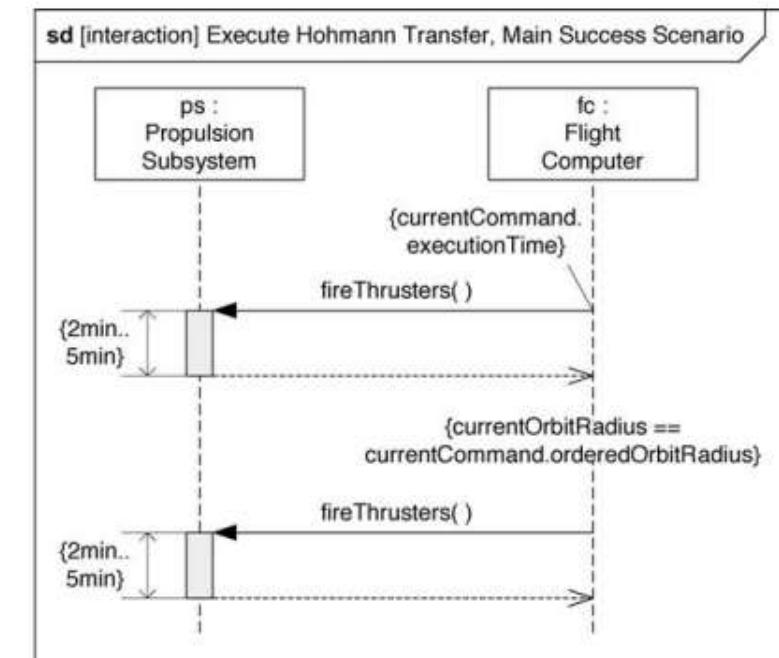


Fig: Specifying constraints in an interaction

The Sequence Design Frame – State Variants

A state invariant is a condition that you apply to a specific lifeline at a point preceding (immediately above) a particular event occurrence.

That condition must hold true for that lifeline at the moment of that event occurrence in a valid execution of the interaction

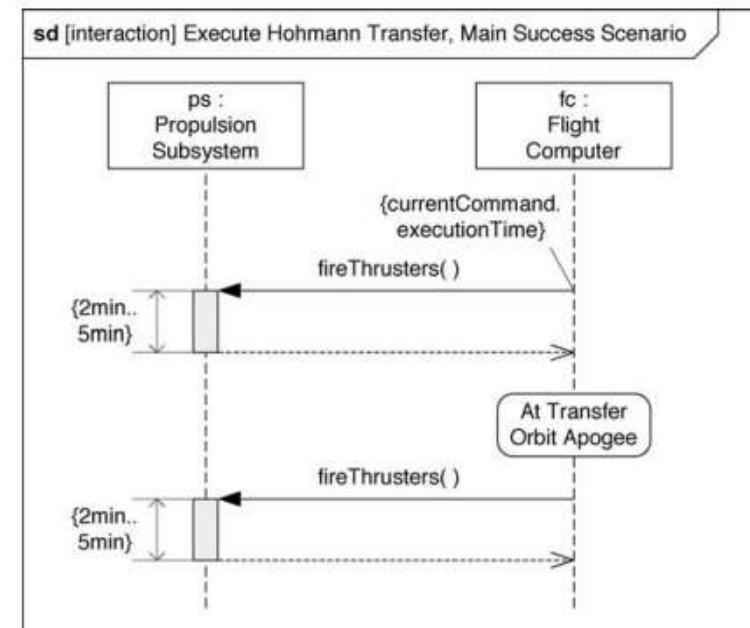


Fig: Specifying a state invariant using the state notation

The Sequence Design Frame – Combined Fragments

A combined fragment is a mechanism that allows you to add control logic (such as decisions, loops, parallel behaviors) to an interaction. We call that string an interaction operator, SysML defines 11 interaction operators.

There are four, however, that you're likely to use in your daily modeling work:

- opt,
- alt,
- loop, and
- par.

A combined fragment with an opt interaction operator represents an optional set of event occurrences that could happen during an execution of the interaction if a condition—called the guard—evaluates to true.

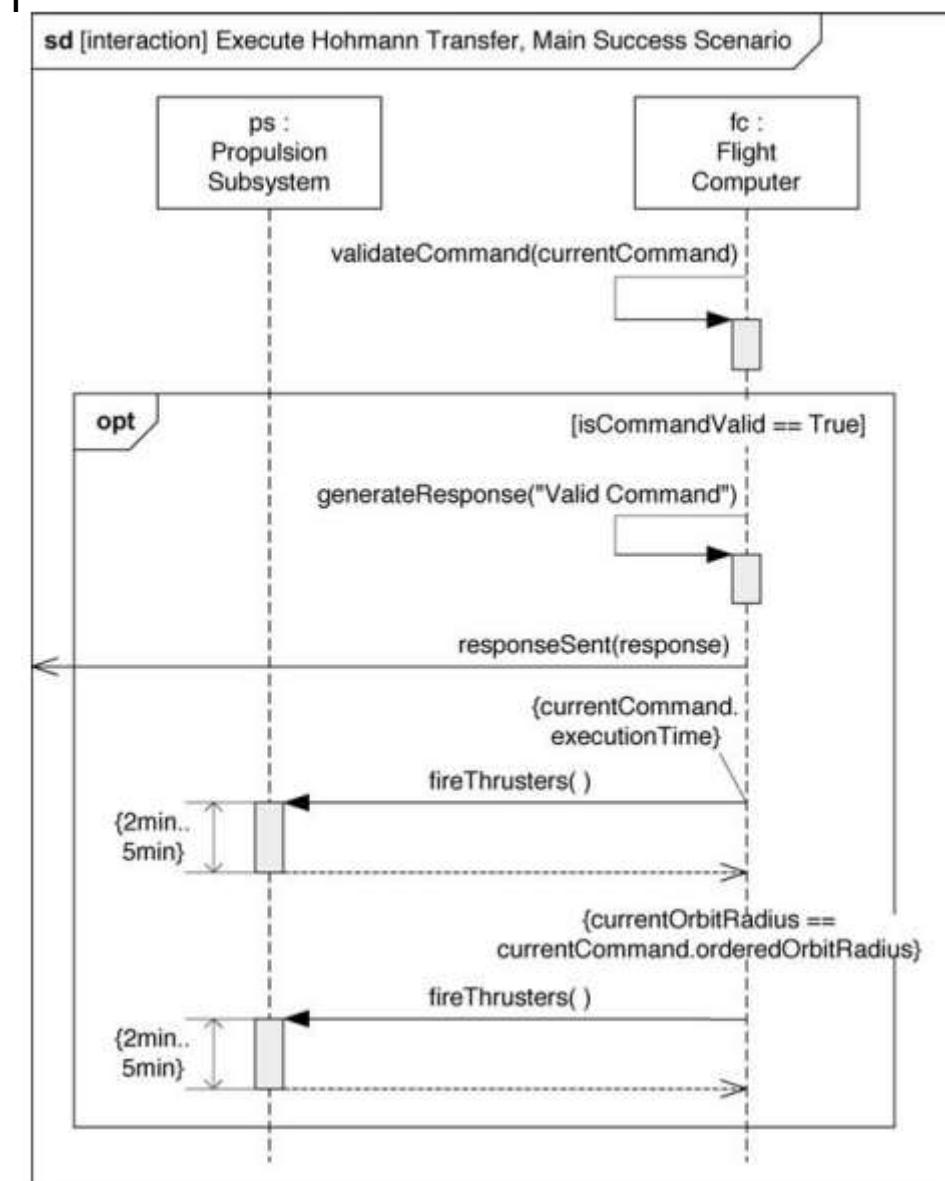


Fig: An opt combined fragment in an interaction

The Sequence Design Frame – Combined Fragments

Alt Operator

An alt combined fragment must have two or more operands (regions) that contain those alternative sets of event occurrences.

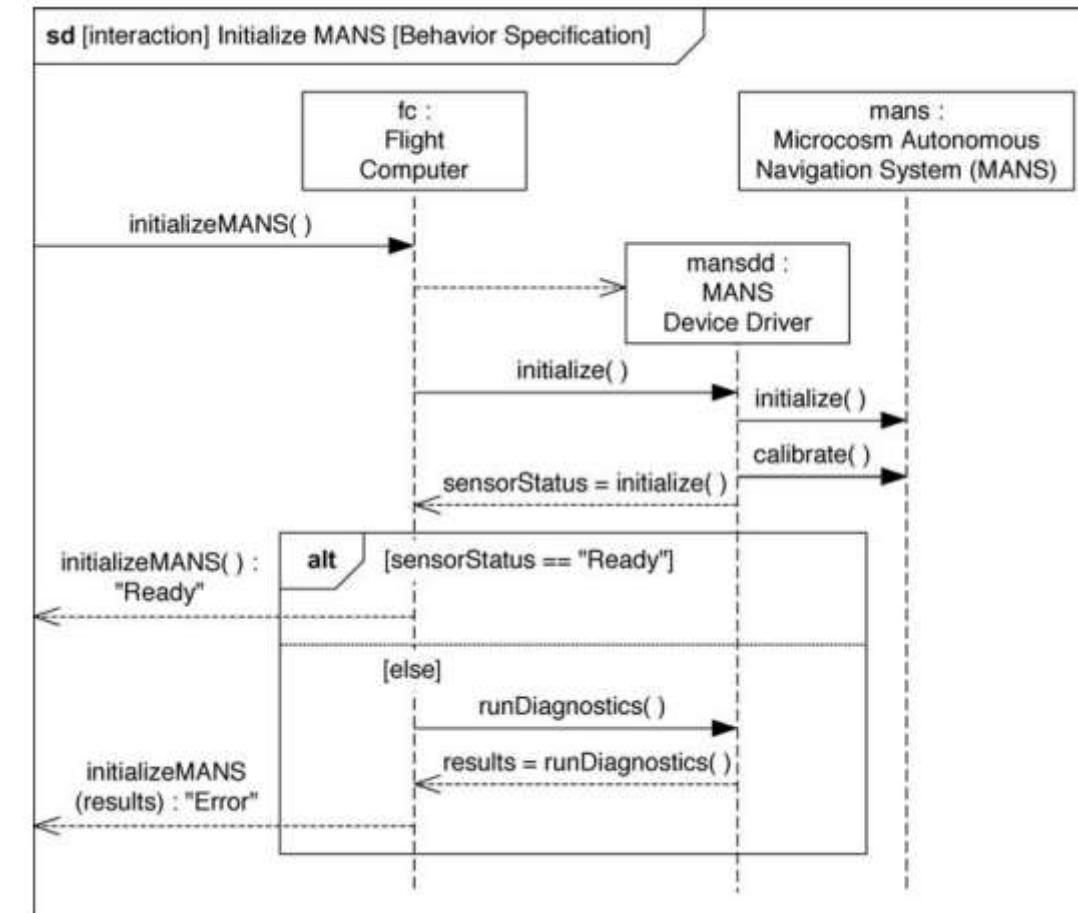


Fig: An alt combined fragment in an interaction

The Sequence Design Frame – Combined Fragments

Loop Operator

A combined fragment with a loop interaction operator represents a set of event occurrences that could happen multiple times during a single execution of an interaction

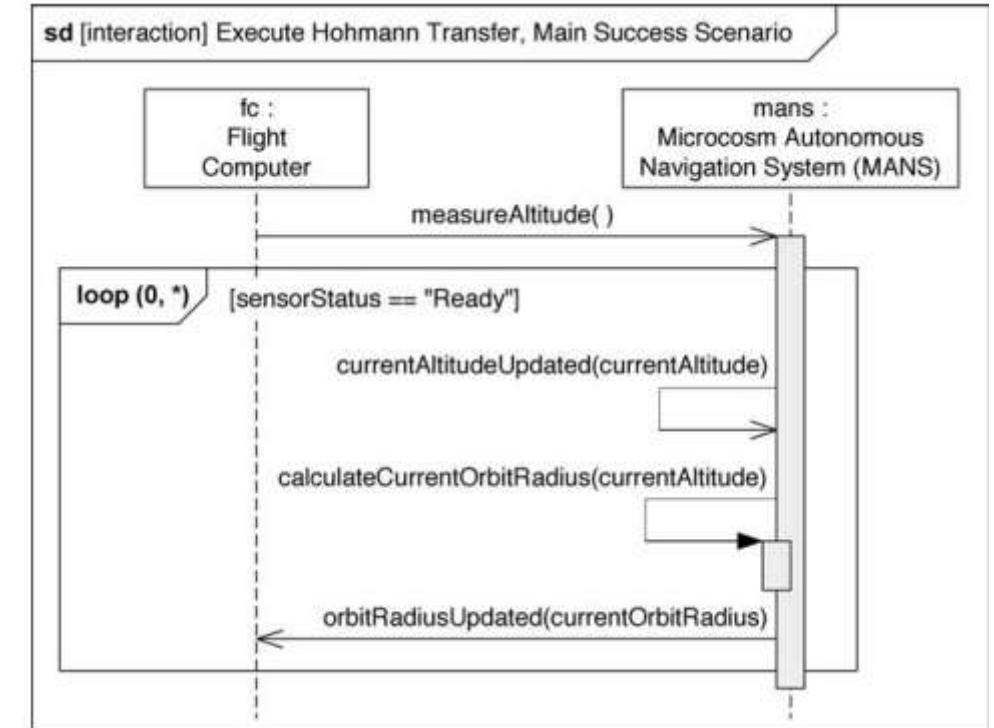


Fig: A loop combined fragment in an interaction

The Sequence Design Frame – Combined Fragments

Par Operator

A combined fragment with a par interaction operator represents two or more sets of event occurrences that happen in parallel with each other during an execution of an interaction

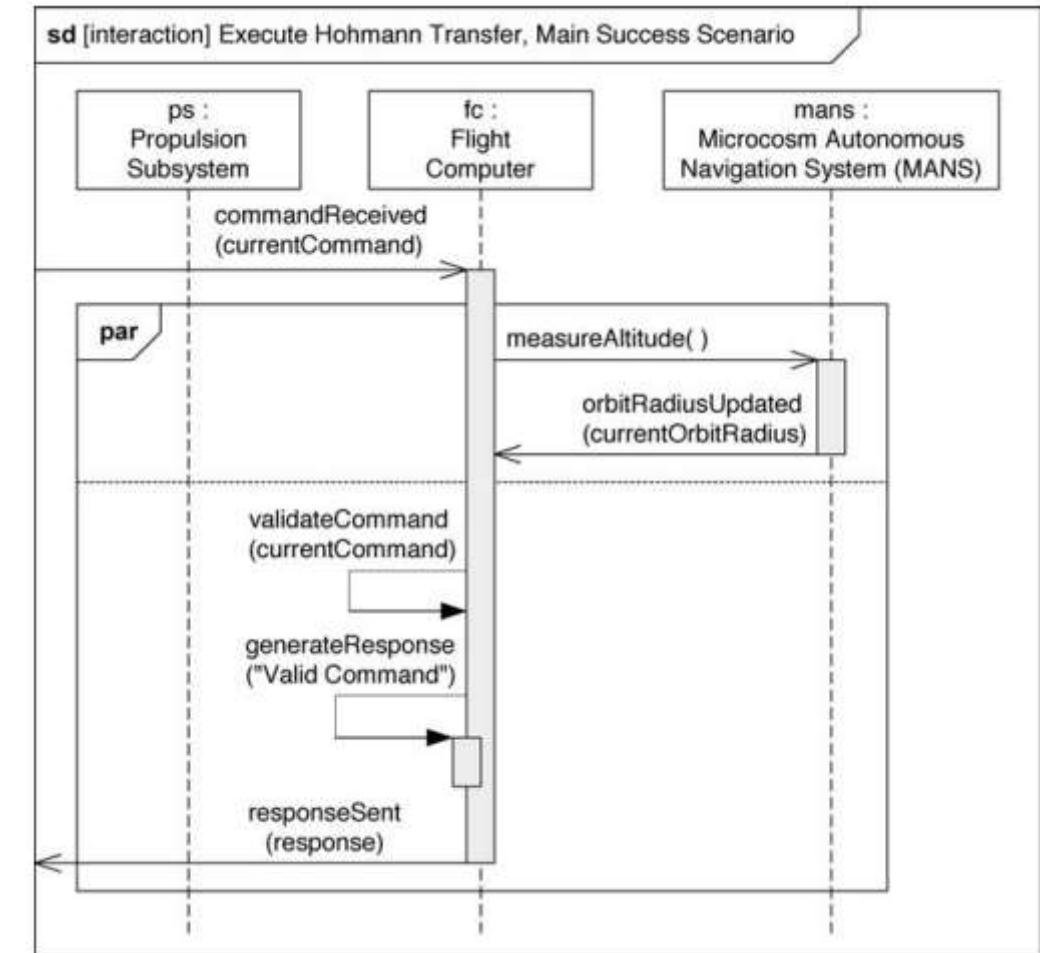


Fig: A lpar combined fragment in an interaction

Module 3:
Requirement Analysis – Modelling



Monday, 06th September



09:00 AM – 10:00 AM



SAHIL DATERO
Business Analyst, Delivery

Agenda:

- State Diagram – Purpose, uses
- State Diagram – Frame
- Package Diagram – Purpose, uses
- Package Diagram – Frame

Purpose of State Diagram

Unlike an activity diagram and a sequence diagram, a state machine diagram focuses attention on how a structure within a system **changes state** in response to event occurrences over time.

A state machine diagram is well suited to serve as a detailed design artifact (that is, an input into development).

Many commercial-grade modeling tools let you autogenerate production-quality source code based on the behavior displayed on a state machine diagram.

The drawback to a state machine diagram is that its use is limited to describing the behavior of blocks that have **defined states** (that is, they exhibit state-based behavior in response to event occurrences). Not all blocks have such defined states.

Like a sequence diagram, a state machine diagram is a precise and unambiguous specification of **behavior**.

When should you create a State diagram?

- You can potentially create a state machine diagram at any point in the system life cycle.
- Because a state machine behavior most often serves as a block's classifier behavior, you can create a state machine diagram to describe the behavior of a block at any level in the system hierarchy (such as the system of interest itself, a subsystem, or a single component).

The State Diagram Frame

- stateMachine - The only allowable model element type for a sequence diagram
- Like an interaction, an activity, a block, and a package, a state machine is also a kind of namespace. The elements that appear within the frame—the vertices and transitions—are contained within (nested under) this state machine within the model hierarchy.
- You're not required to specify on the state machine diagram the name of the block that the state machine is associated with.

The State Machine Diagram Frame - States

States

- One of the simplest examples is a lamp that turns on and off via a pull chain. The lamp has two defined states: On and Off.



The state that it's in at a given moment determines how it will **respond** to event occurrences (such as unscrewing the light bulb, pulling the chain, or knocking the lamp over).

- Software objects, too, can have a defined set of states. A file, for example, can exist in the following states: Open, Closed, Modified, Unmodified, Encrypted, Unencrypted, and others.

Sometimes states have meaning only in the context of other states.

- For example, Modified and Unmodified are meaningful only when a file is in the Open state.

Formally, we would refer to Open as a **composite state**; Modified and Unmodified would be **substates** of the Open state.

A state that has no substates is called a **simple state**. Another common kind of state is the **final state**.

The Orbit Insertion, Acquisition, Slew, and Safe Mode states are examples of simple states.

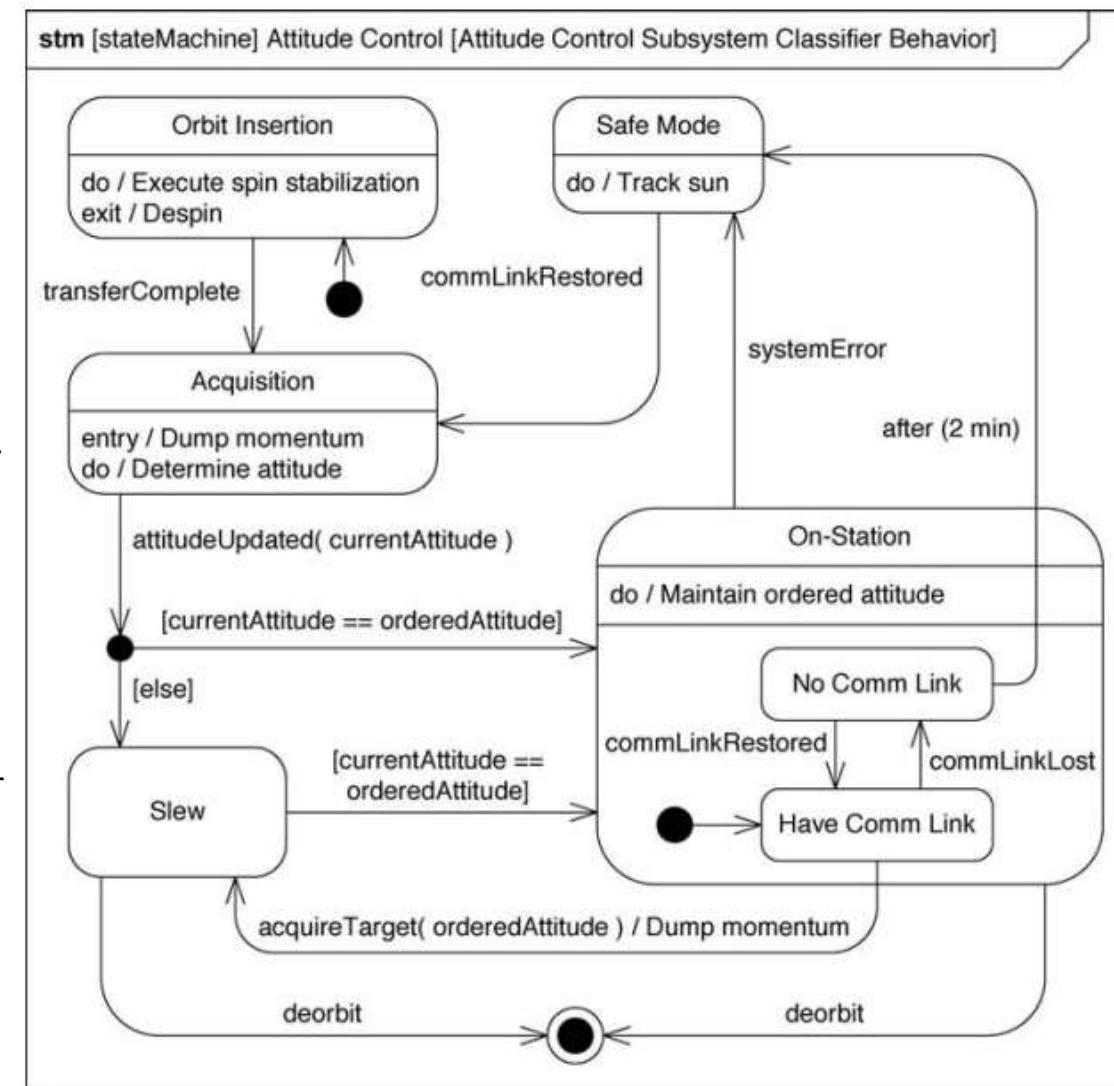


Fig: A Sample State Machine Diagram Diagram

The State Machine Diagram Frame – Simple States (Entry-Exit)

A simple state may optionally display a second compartment that lists its internal behaviors and internal transitions. SysML defines three internal behaviors that a state can perform: entry, exit, and do.

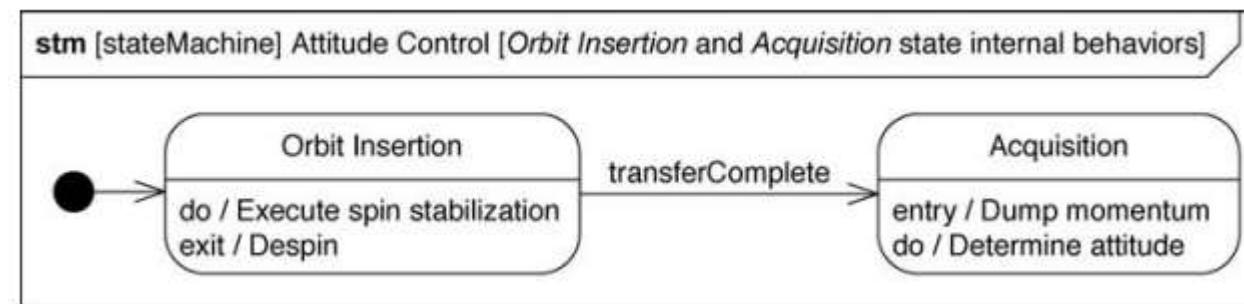


Fig: Entry, exit and do behaviours in simple states in a state machine

A state's entry behavior, if present, is the first **behavior executed** upon entering that state. The entry behavior is regarded as atomic (uninterruptible). This means that it's guaranteed to finish executing before the state machine can process a new event occurrence (and potentially transition out of that state).

A state's exit behavior, if present, is **the last behavior** executed before leaving that state, which happens when an event occurrence causes the state machine to transition to a new state. Like an entry behavior, an exit behavior is regarded as atomic. It's guaranteed to finish executing; no new event occurrence can interrupt its execution.

The State Machine Diagram Frame – Simple States (Do)

It's important to understand that a state machine may rest in a given state for some nondeterministic period before transitioning to a new state. This means that other behaviors can execute in between a state's entry and exit behaviors.

A state's do behavior, if present, begins executing upon entering the state, immediately following the state's entry behavior. The key difference between the do behavior and the entry behavior is that the do behavior is nonatomic; its execution can be interrupted by a new event occurrence that causes a transition to a new state.

If a do behavior is interrupted by an event occurrence, the do behavior is aborted and the exit behavior for that state is executed (right before leaving that state). If a **do behavior terminates** on its own before the next event occurs, one of two things could happen:

- The state machine could continue to rest in that state while waiting for the next event occurrence, if all of the outgoing transitions require a trigger.
- The state machine could immediately transition to a new state, if there's an outgoing transition that doesn't require a trigger.

The State Machine Diagram Frame – Composite States

The difference in Composite and Simple state is that a composite state has **nested substates**, which you would display in a third compartment (below the second compartment).

Like Simple States, When a composite state is inactive, all its substates are inactive, too. When a composite state is active, **exactly one** of its substates is also active.

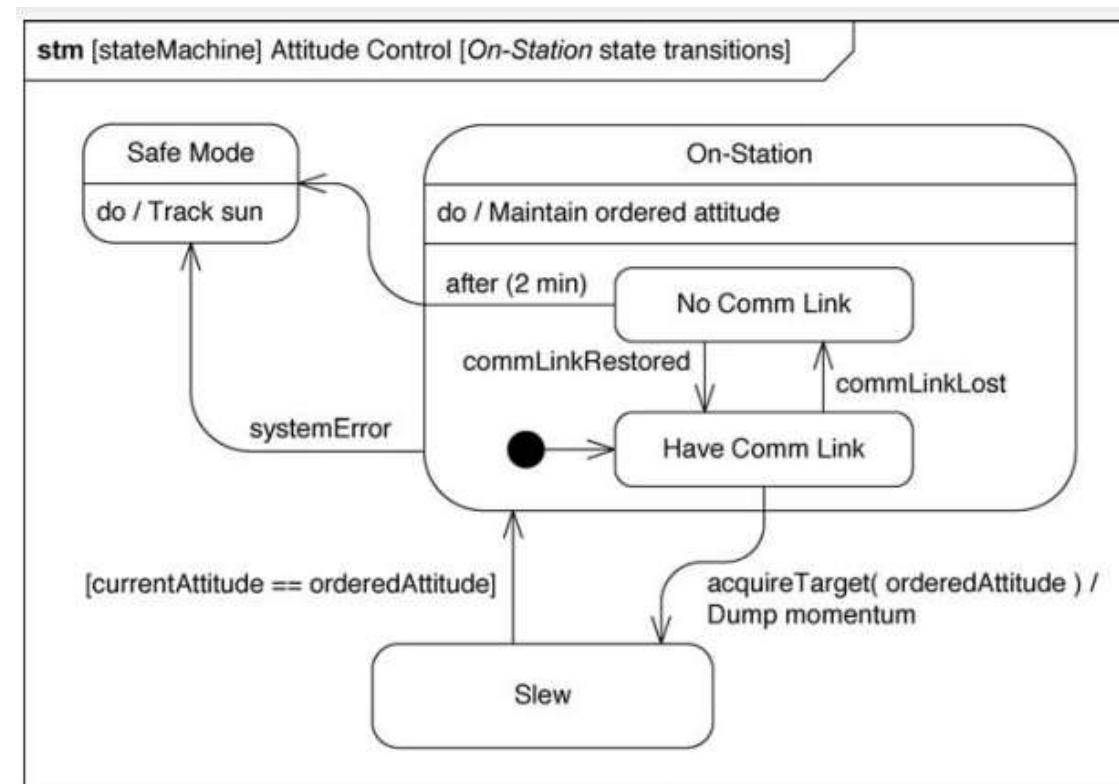


Fig: Composite State with two substate in a state machine

The State Machine Diagram Frame – Final States

This represents completion of the state machine behavior as a whole; it will **not react** to any new event occurrences from that point forward.

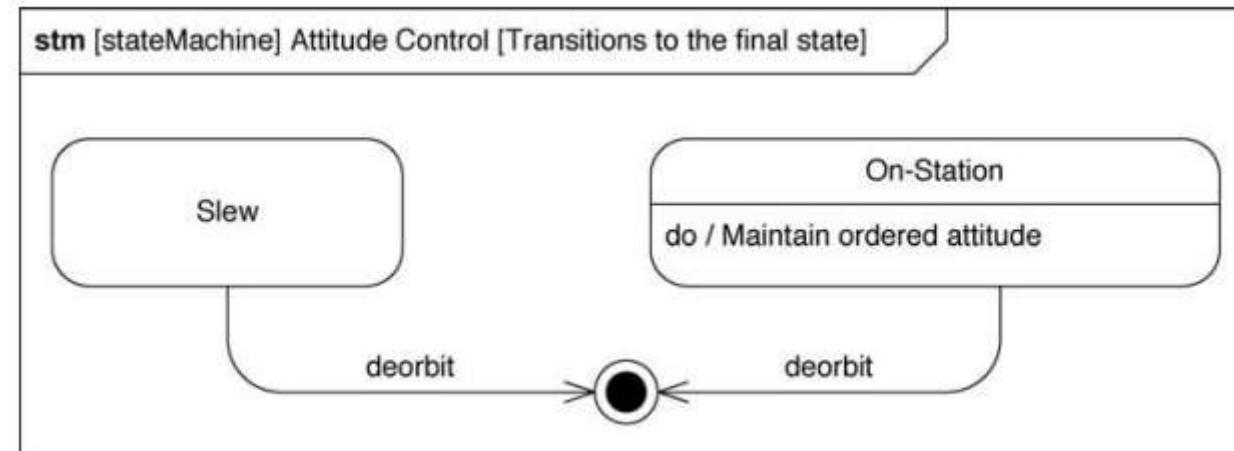


Fig: Final state in a state machine

The State Machine Diagram Frame – Transitions

Each transition can specify three optional pieces of information: a trigger, a guard, and an effect. The format that, string follows:
 <trigger> [<guard>] / <effect>

The trigger must match the name of an event that you've defined in your system model. SysML defines four types of events: signal events, call events, time events, and change events,

For now, it's sufficient to know that an instance of an event (during system operation) is called an event occurrence, and an event occurrence can trigger a transition between states.

The guard is a Boolean expression always expressed between square brackets – True or False

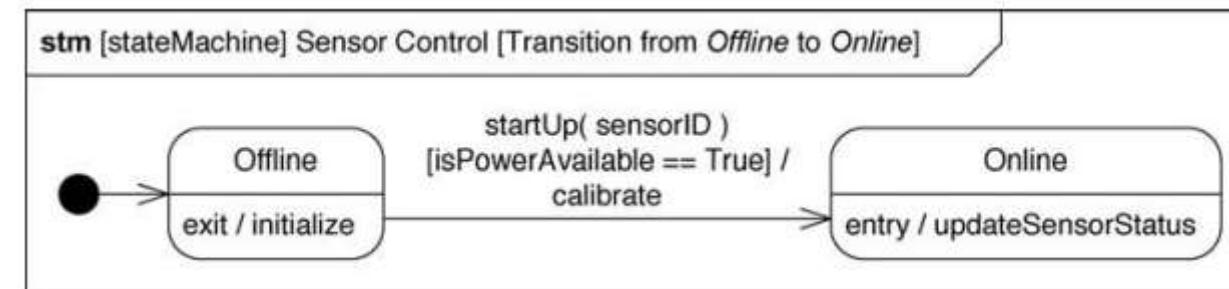


Fig: A transition with a trigger, a guard, and an effect

The key point is that this effect gets executed when the transition fires

The State Machine Diagram Frame – Transitions

A transition effect is part of a larger sequence of behaviors that executes when a transition fires. We call that sequence of behaviors the **run-to-completion step**. The run-to-completion step consists of the following behaviors in the order listed:

- The exit behavior of the source state
- The effect specified for the transition itself
- The entry behavior of the target state

This entire sequence of behaviors is regarded as atomic and instantaneous. It's guaranteed to finish executing; no new event occurrence can interrupt any of the behaviors in this sequence.

Systems engineers know that no real-world system behavior truly executes in zero time. However, when you choose to model a behavior as part of a run-to-completion step—as an entry behavior, a transition effect, or an exit behavior—you are asserting to your stakeholders that the system will be implemented so that the behavior will be un-interruptible and that no other behavior will run concurrently during its execution, creating the illusion that it runs in zero time.

The State Machine Diagram Frame – Transitions (External vs Internal)

- Whenever you see an arrow drawn from one state to another (or from one state back to itself), you're looking at an external transition.

The string format for an internal transition is the same as for an external transition. However, the string for an internal transition is displayed in the second compartment of a state (along with the three optional internal behaviors); in contrast to an external transition, the string for an internal transition is not displayed next to an arrow.

When an internal transition in a state fires, neither the exit behavior nor the entry behavior for that state, if present, gets executed. In short, when an internal transition fires, the only behavior that gets executed is the effect specified for that internal transition.

The trigger for this internal transition is a **change event**

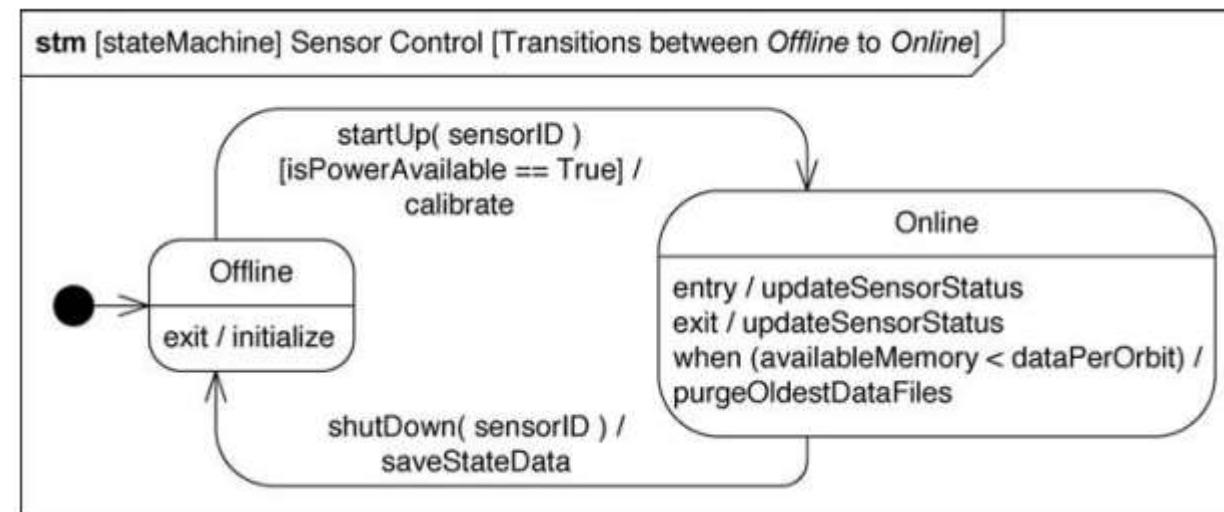


Fig: A state with an internal transition

The State Machine Diagram Frame – Effect Type

SysML defines four types of events:

- Signal Events
- Call Events
- Time Events
- Change Events

Signal Events

A signal event represents the receipt of a signal instance by a target structure that's receptive to it; the target in this context would be the structure that's executing a state machine behavior.

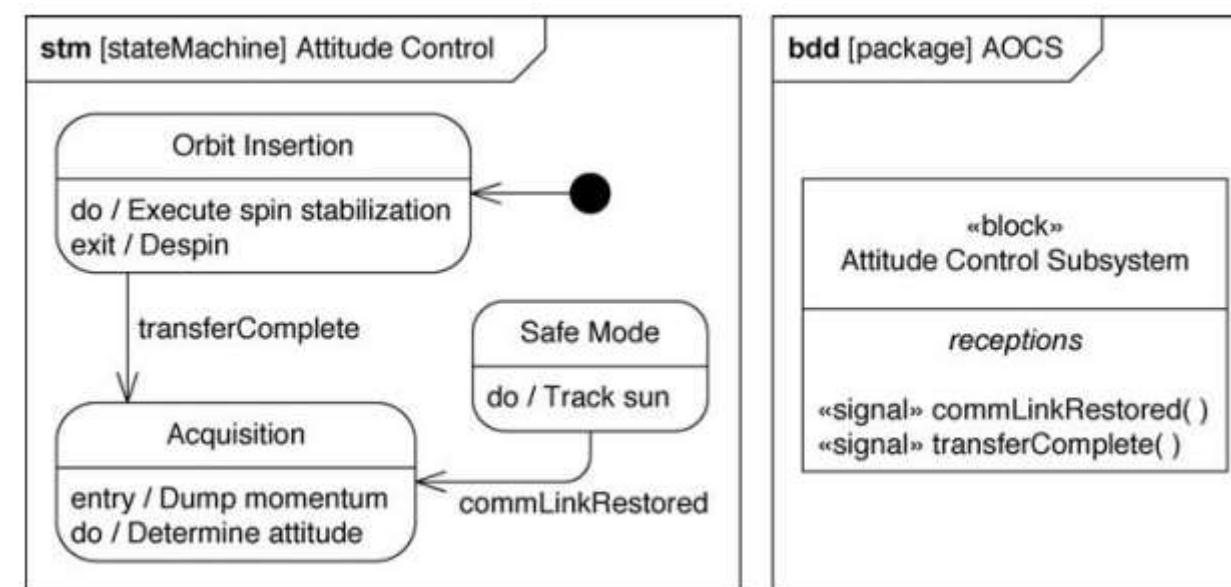


Fig: Signal event triggers and corresponding receptions

The State Machine Diagram Frame – Effect Type

Call Events

A call event represents the receipt of a request to invoke an operation in a target structure—a request that's sent from a calling structure. The target in this context would be the structure that's executing state machine behavior.

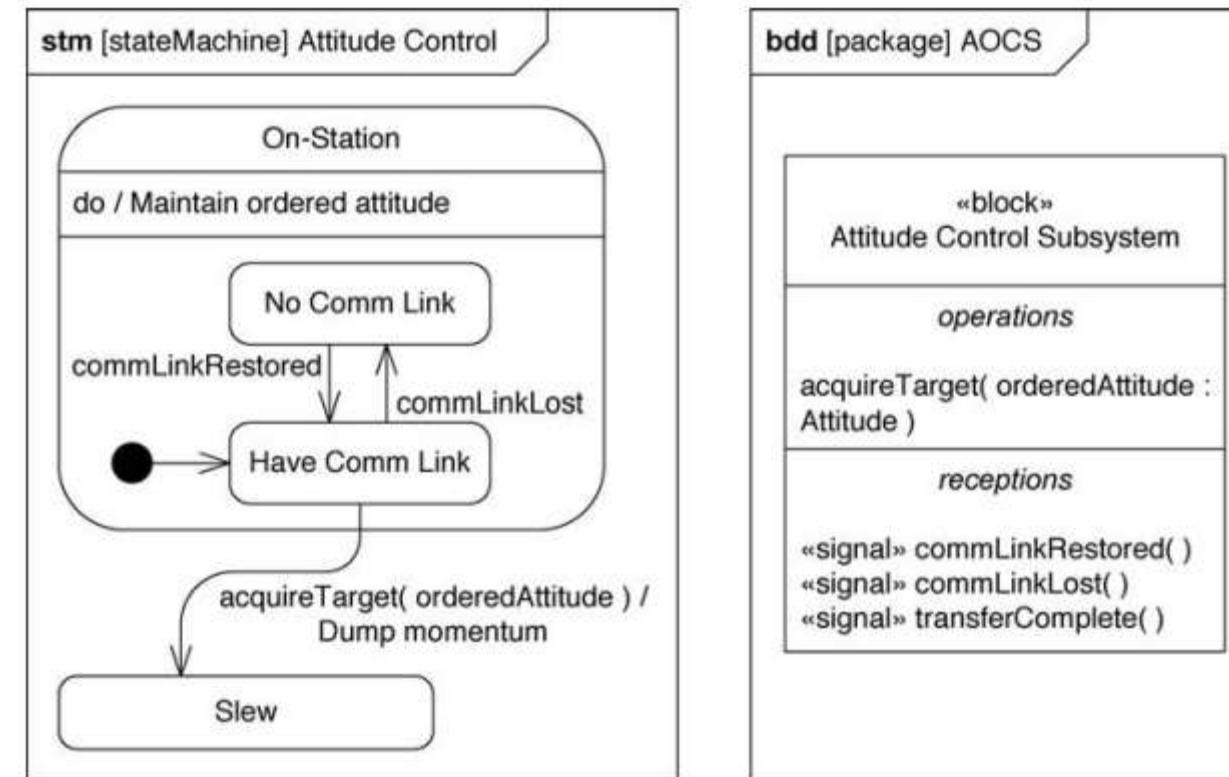


Fig: Signal event triggers and corresponding receptions

The State Machine Diagram Frame – Effect Type

Time Events

Time events represents an instant in time. There are two types of time events: relative and absolute. A relative time event trigger always begins with the keyword after. An absolute time event trigger always begins with the keyword at.

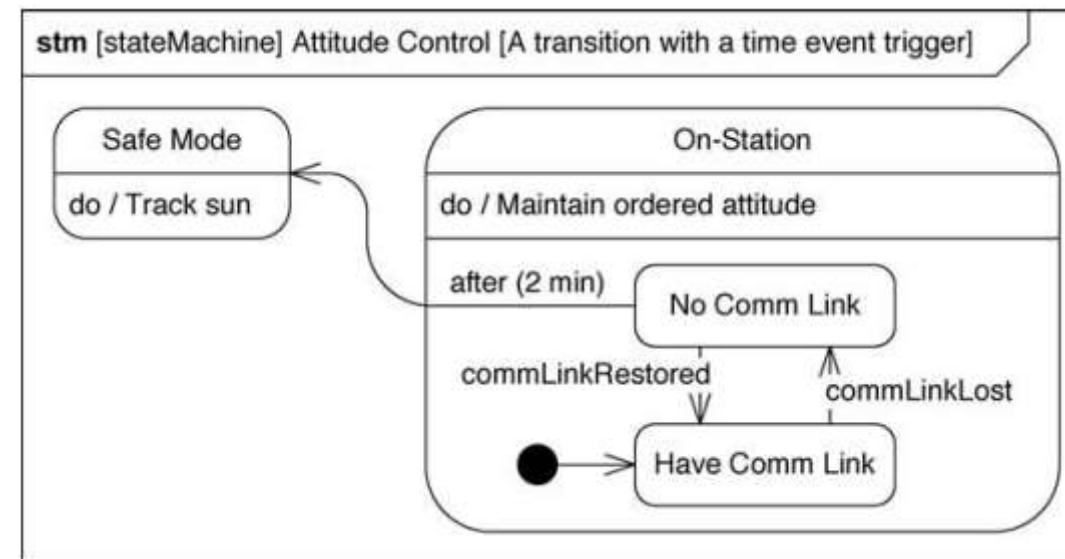


Fig: A time event trigger in a state machine

The State Machine Diagram Frame – Effect Type

Change Events

A change event is defined as a Boolean expression. A defined change event occurs during system operation each time the specified Boolean expression toggles from false to true. As with a time event trigger, it's easy to identify a change event trigger; it always begins with the key-word when, followed by a Boolean expression in parentheses.

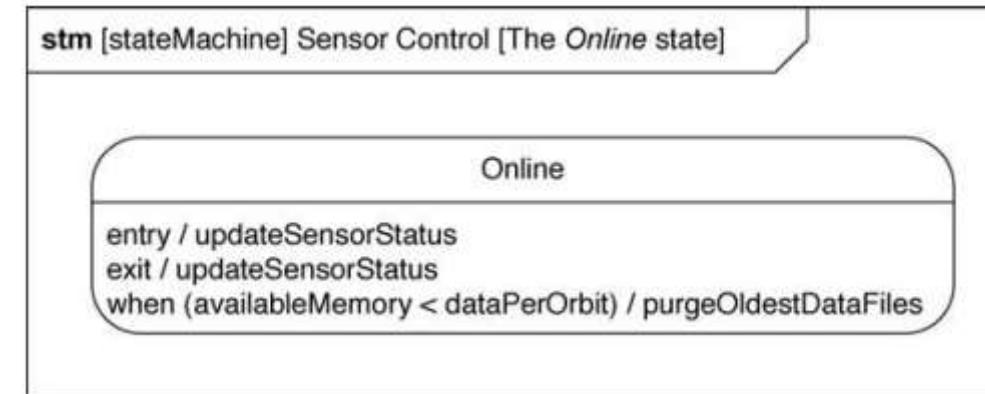


Fig: A change event trigger in a state machine

Purpose of Package Diagram

A **Package Diagram** is the kind of diagram you create when you need to display the organization of the system model.

Depending on your project's unique goals, you will structure modelt's useful to create package diagrams to provide your stakeholders an easily understood view of that structure.

Package Diagrams can display packages nested within packages to convey the containment hierarchy of the model

When should you create a Package diagram?

- You would create new package diagrams when you modify the model structure in significant ways (as determined by the concerns of your stakeholders).
- As the design stage of the life cycle progresses, you will decompose higher-level structural elements into lower-level structural elements.
- As the design stage of the life cycle progresses, you will decompose higher-level structural elements into lower-level structural elements.

The Package Diagram Frame

The model element type that the diagram frame represents can be any of the following:

- package
- model
- modelLibrary
- view
- profile

The Package Diagram Frame

A model is one of four specialized kinds of packages; it's the kind of package that serves as the root of a containment hierarchy.

A change in the supplier element (at the arrowhead end) may result in a change to the client element (at the tail end).

Fig. conveys that a change to the contents of the Requirements package may result in a change to the contents of the Test Cases, Behavior, and Structure packages.

Importing Packages is represented with **Package Import** Relationship

A package diagram expresses information about the structure of a system model (a package containment hierarchy); this is in **contrast** to BDDs and IBDs, which convey information about the structure of a system you're designing. Modelers

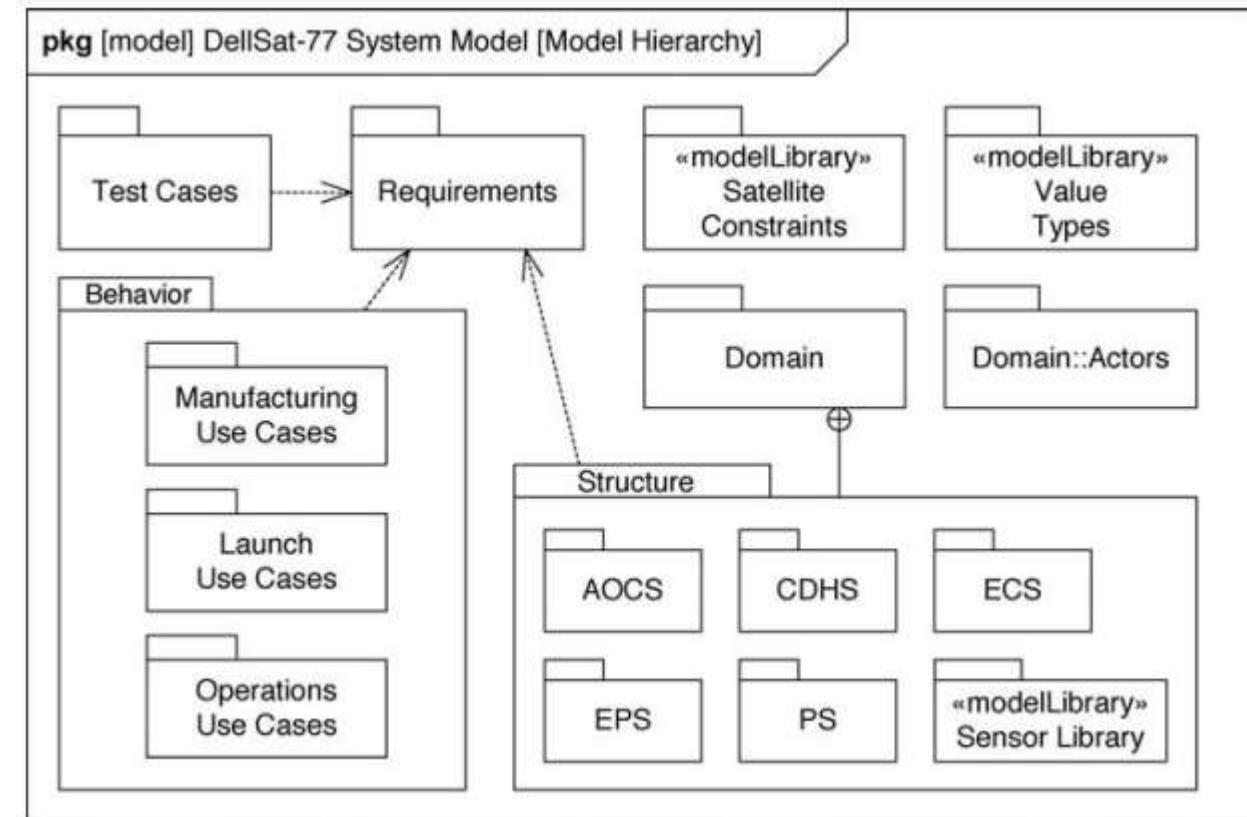


Fig: A Sample Package Diagram

The Package Diagram Frame

A model is the kind of package that serves as the root of a containment hierarchy. A model library is the kind of package that contains a set of elements that you intend to reuse in multiple models. A profile is the kind of package that contains a set of stereotypes—extensions to an existing modeling language that define a new modeling language that's better suited for a particular design domain. A view is the kind of package that contains a filtered subset of a model—a subset that conforms to a defined viewpoint and addresses specific stakeholder concerns. You can create package diagrams to display these specialized kinds of packages and the dependencies that exist among them.

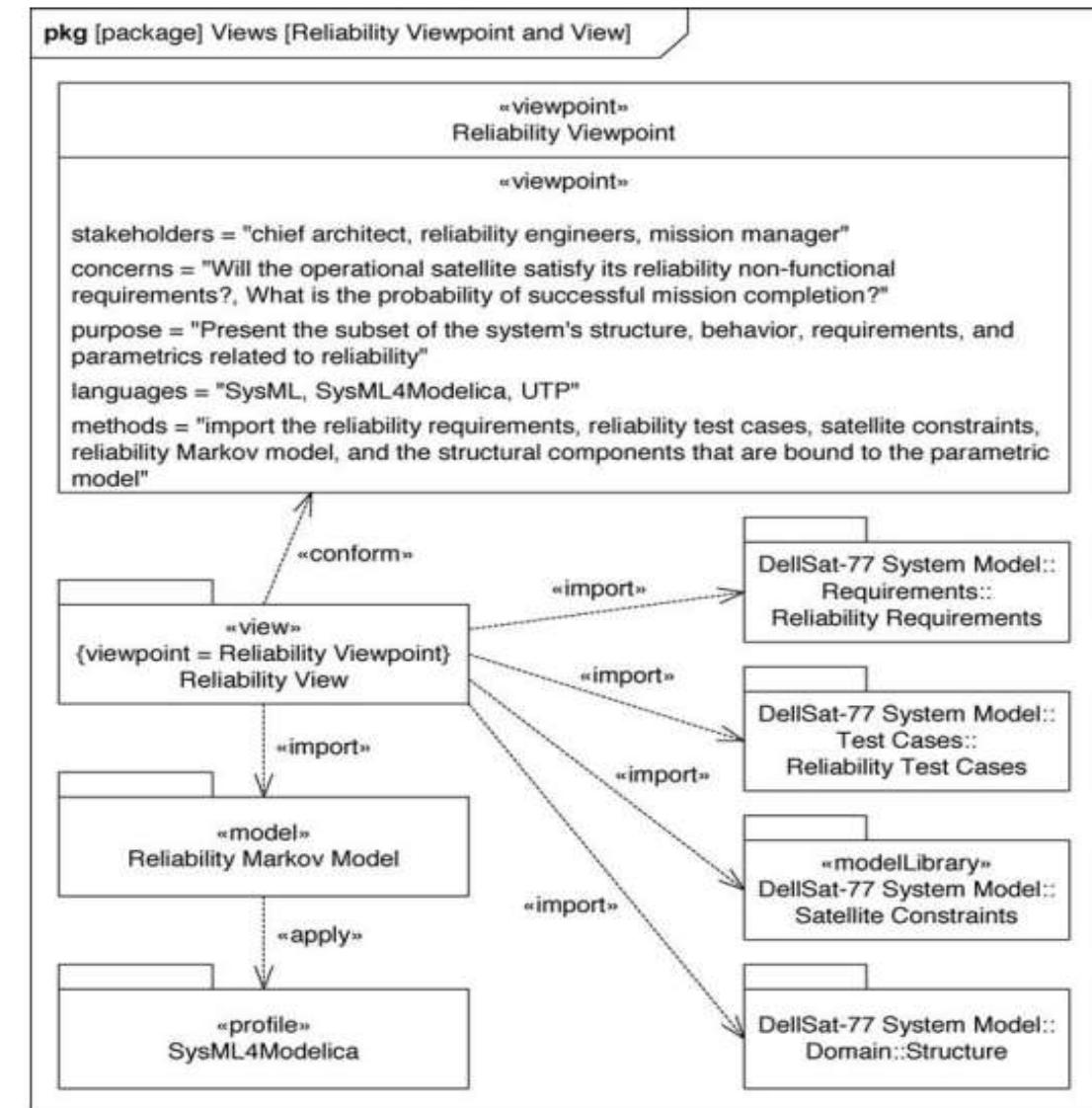


Fig: A viewpoint and a conforming view

Module 3:
Requirement Analysis – Modelling



September, Monday, 09th



09:00 AM – 10:00 AM

Agenda:

- Parametric Diagram – Purpose, uses
- Parametric Diagram - Frame
- Requirement Diagram – Purpose, uses
- Requirement Diagram - Frame



SAHIL DATERO
Business Analyst, Delivery

Purpose of Parametric Diagram

Parametric diagram is a unique kind of SysML diagram, one that's used to express information about a system **constraints**.

These constraints generally take form of mathematical models that determine set of valid **values** within a running system. A parametric diagram has unique capability to convey these mathematical models to the stakeholders

What do you achieve by imposing a fixed mathematical relationship on block's value properties:

- To specify assertions about valid system within an operational system (and therefore detect exceptional conditions when they occur)
- To use the blocks in your system model to provide the inputs for (and capture the outputs of) engineering analyses and **simulations** during design phase.

Where do parametric diagrams fit all into all this?

- To display the **bindings** between constraints parametric in different constraints expressions to create a composite system of equations (or inequalities)
- To display the bindings between constraints parametric and value properties to apply a constraint expression to a block (and, in so doing, impose a fixed mathematical relationship on a set of value properties)

When should you create a Parametric diagram?

- Just as IBDs and BDDs provide complementary views of blocks, Because of this close relationship, you can potentially create para-metric diagrams during any stage of the system life cycle.
- Like an IBD, a parametric diagram displays the internal structure of a block—but with a focus on the bindings between value properties and constraint parameters.

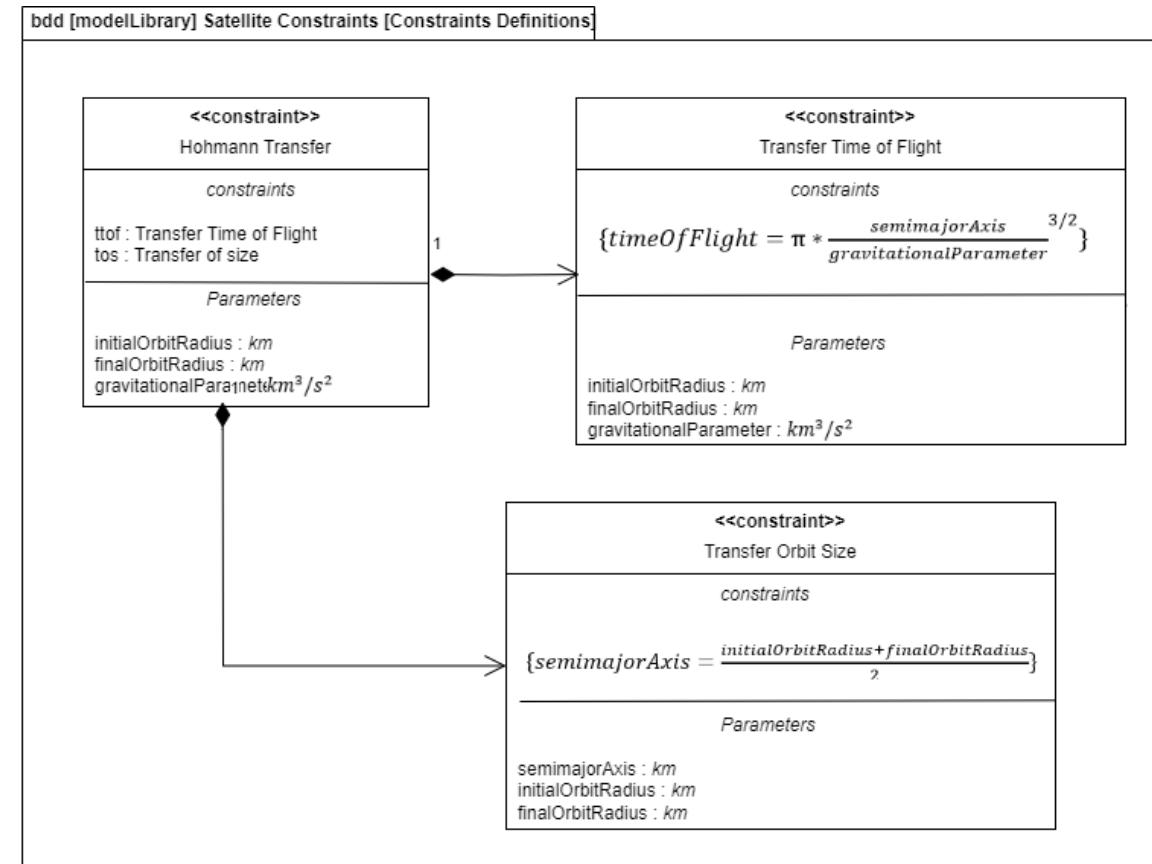


Fig: Block definitions needed to create a parametric diagram for Hohmann Transfer

Blocks revisited, Parametric Diagram provisioning, mathematical model constraints

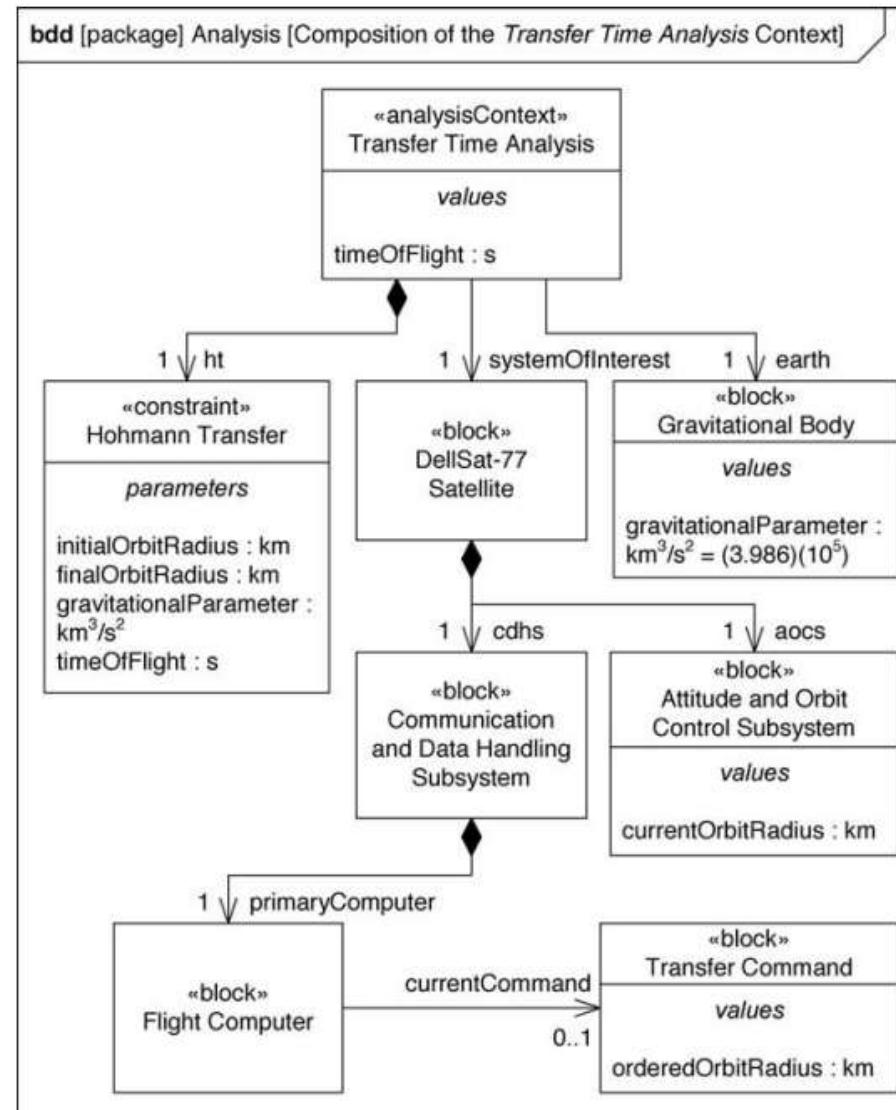


Fig: Block definitions needed to create a parametric diagram for Transfer Time analysis

The Parametric Diagram Frame

The model element type that the diagram frame represents can be any of the following:

- block
- constraintblock

When a parametric diagram represents a constraint block, the diagram displays only the constraint properties and bindings that form the internal structure of that constraint block.

When a parametric diagram represents a block, it primarily displays the bindings between the block's value properties and constraint properties that contain nested value properties of interest.

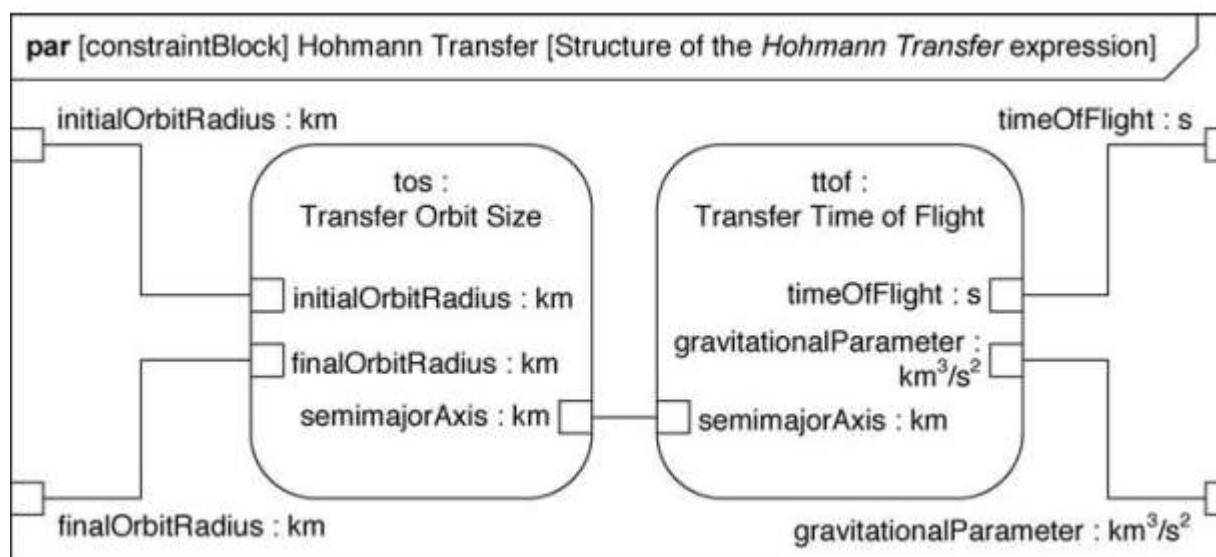


Fig: Parametric diagram for the Hohmann Transfer constraint Block

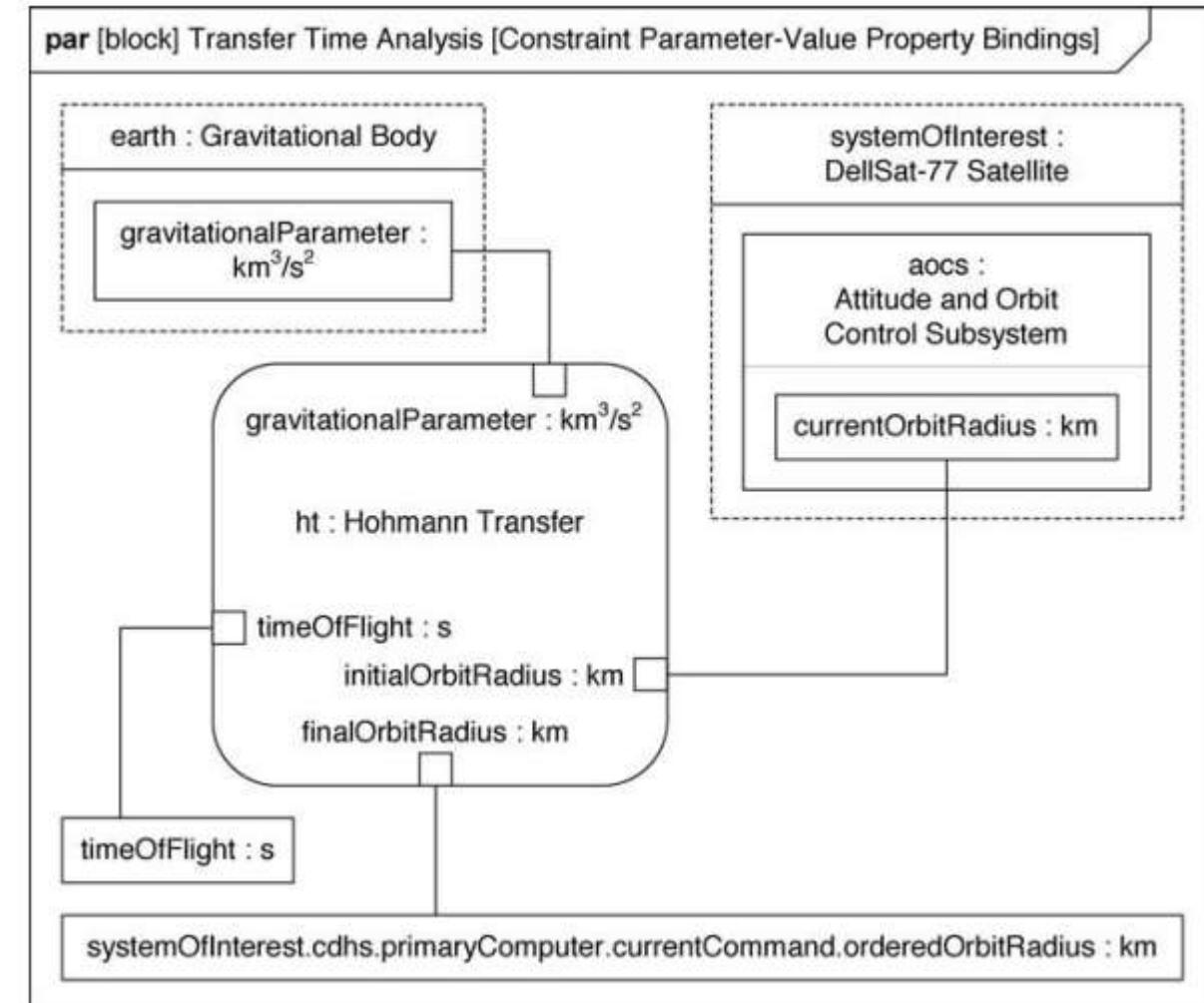


Fig: Parametric diagram for the Transfer Time Analysis block

The Parametric Diagram Frame

`<constraint name> : <type>`

Constraint Parameter

`<parameter name> : <type> [<multiplicity>]`

Value property

`<value name> : <type> [<multiplicity>] = <default value>`

Binding connector

- A binding connector can appear only on a parametric diagram.
- Note that binding connectors convey no notion of direction.
- Simply represents an equality relationship between the two elements attached at either end.
- That value is then available to the constraint property that owns the constraint parameter, and one of two things can happen:
 - ① The constraint property evaluates to true or false (if values are supplied to all of the parameters in the expression).
 - ② A value is calculated for the constraint parameter that didn't receive a value across its binding connector (if values are supplied to all of the others).

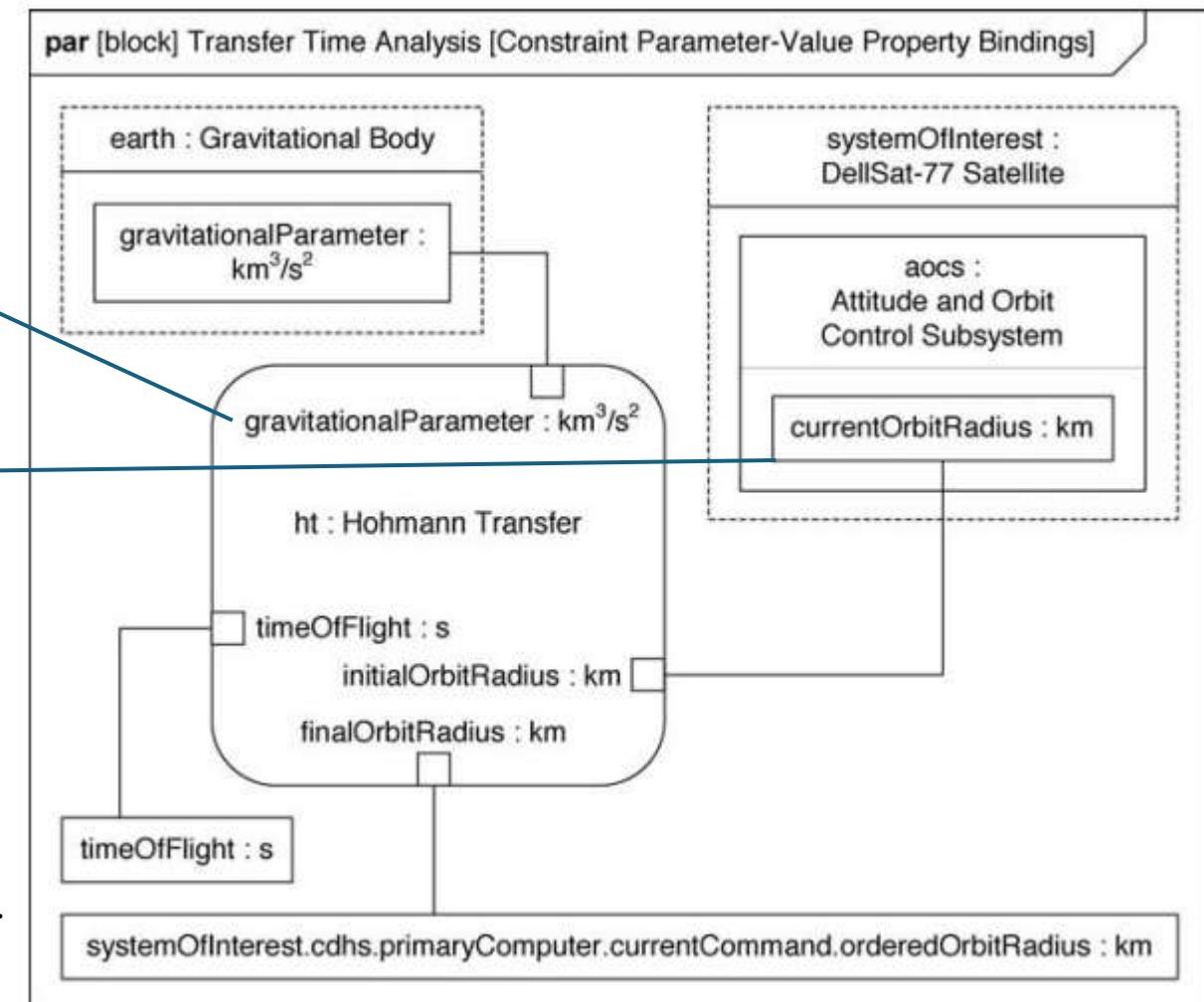


Fig: Parametric diagram for the Transfer Time Analysis block

Purpose of Requirement Diagram

Modelers typically use six kinds of relationships to establish traceability among requirements as well as traceability from requirements to structures and behaviors in the system model.

You can use several notations on requirements diagrams to express these relationships, and each has strengths and weaknesses.

A system's requirements inform every other aspect of its design. The requirements diagram is the primary medium in SysML for conveying this kind of information to your stakeholders.

Not that all methodologies require text-based requirements or that your project team must create them.

An increasingly widespread technique is to create use cases (and their associated use case narratives) in lieu of text-based functional requirements, and constraint expressions in lieu of text-based nonfunctional requirements.

If your project team does write text-based requirements, however, the requirements diagram is the kind of SysML diagram you would create when you need to display those requirements and their relationships to other model elements.

This diagram is particularly valuable when your target audience needs to see the traceability from the requirements to the elements in your system model that are dependent on them.

When should you create a Requirements diagram?

As you add new elements to the model, you will create relationships from those elements back to the requirements that drove the need for their creation.

Establishing requirements traceability in this manner is an ongoing activity throughout design and development.

And you may need to create a requirements diagram to display those relationships at any point during this work.

The Requirement Diagram Frame

The model element type that the diagram frame represents can be any of the following:

- package
- Model
- ModelLibrary
- View
- requirement

You will see that only two elements on this diagram—Mission Requirements Specification and DellSat-77 System Requirements Specification—are contained within the default namespace, the Requirements package. All the other elements shown are nested elsewhere in the model hierarchy.

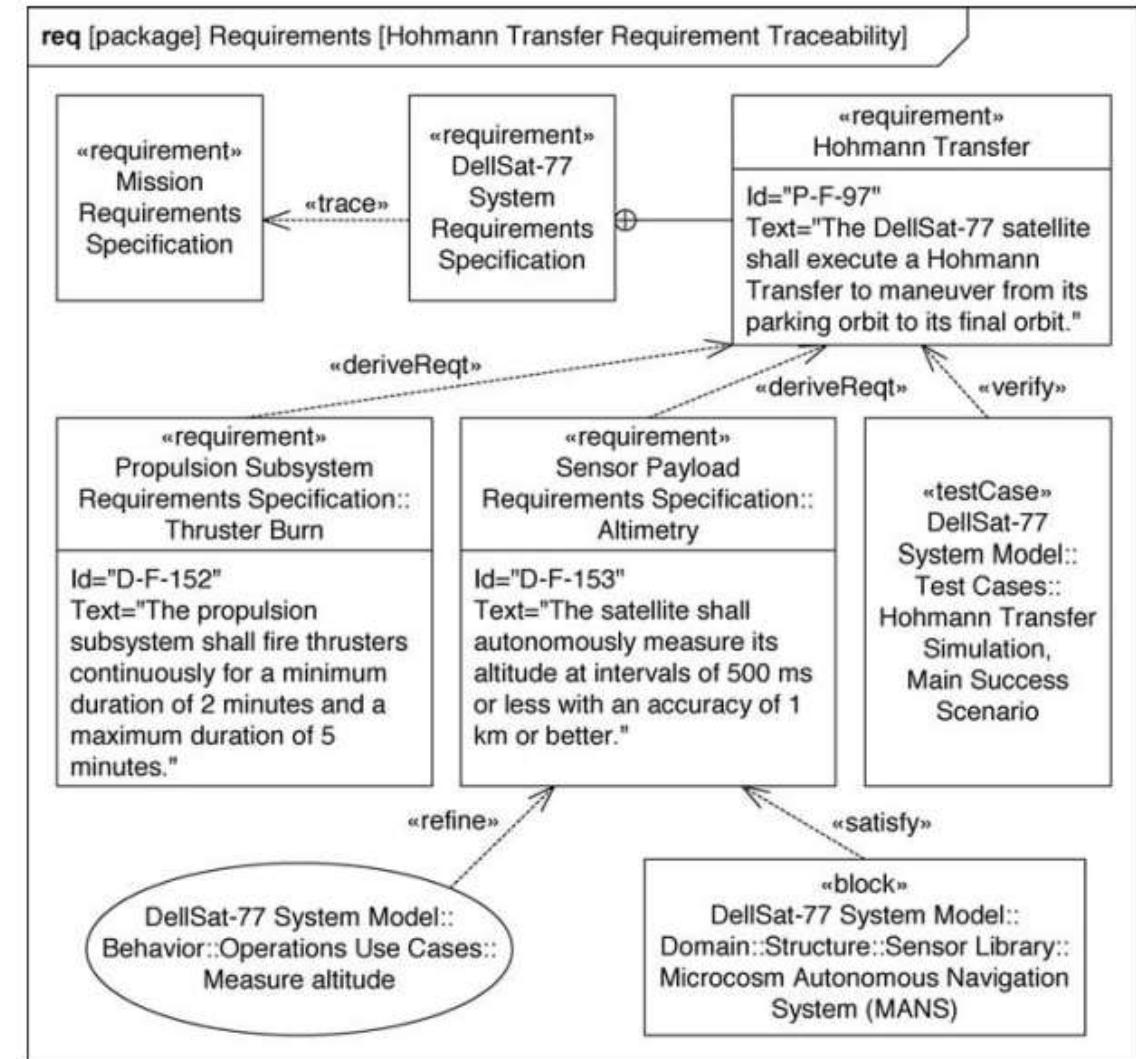


Fig: Sample requirements diagram

Requirement Relationships

Capturing relationships in your system model is useful. However, the greater value lies in relationships you create among the requirements and other model elements. There are six kinds of requirements relationships that you're likely to use in your modelling work: Containment, trace, derive requirements, refine, satisfy, and verify and copy (that will be rarely used in daily practice)

These relationships establish requirements traceability within the system model, which is commonly a process requirement in systems engineering organizations.

Practically speaking, however, capturing these relationships in your model will enable you to use your modeling tool to autogenerate requirement traceability and verification matrices (RTVMs) and perform automated downstream impact analysis when the requirements change (as they inevitably will).

These capabilities are huge time-savers, and that directly translates into cost savings.

Containment Relationships Diagram

Crosshair notation in Fig is used to convey that the DellSat-77 System Requirements Specification element contains the Hohmann Transfer requirement.

Qualified name string notation is used to convey that the Propulsion Subsystem Requirements Specification element contains the Thruster Burn requirement.

A qualified name string conveys that the Sensor Payload Requirements Specification element contains the Altimetry requirement.

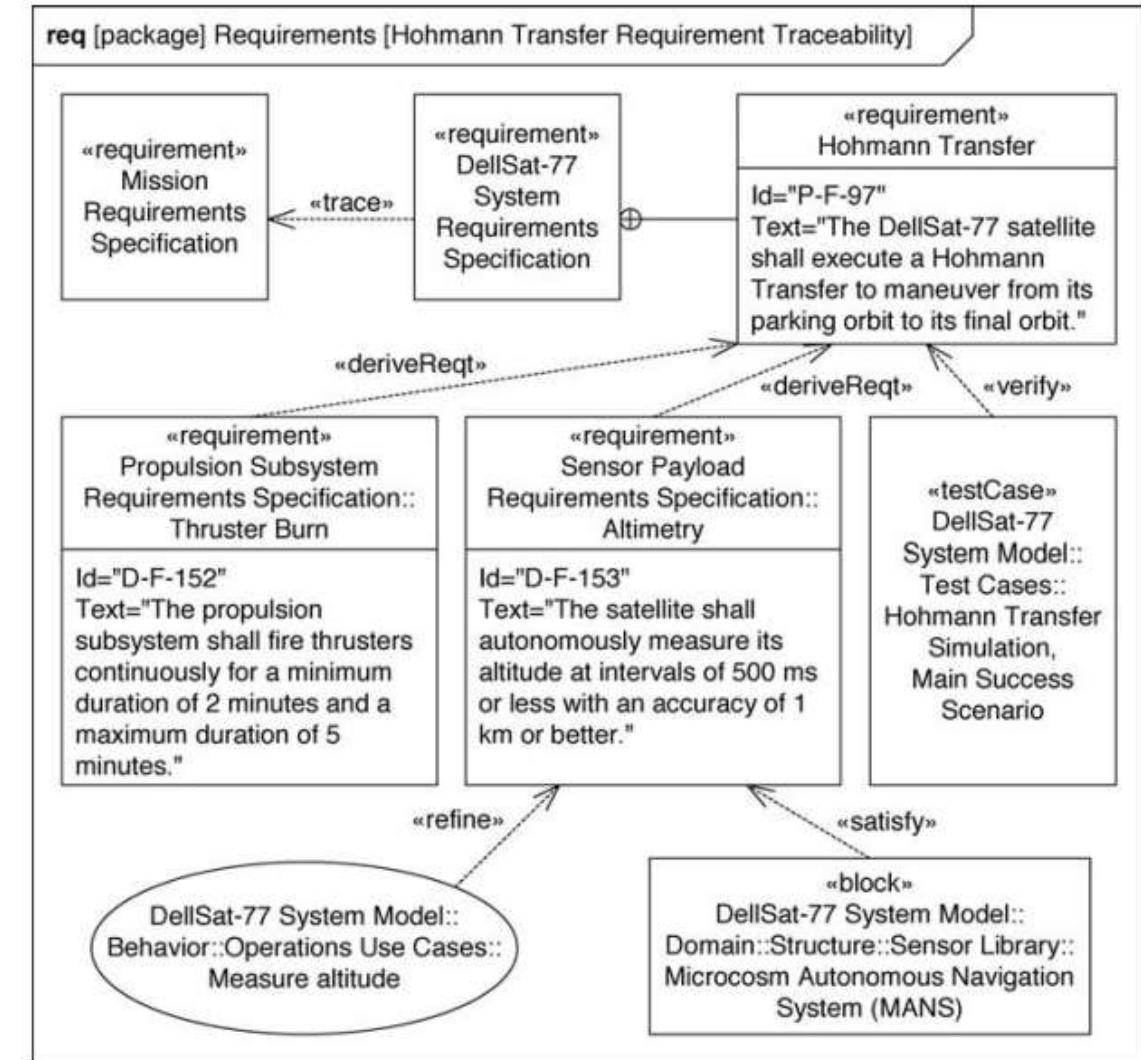
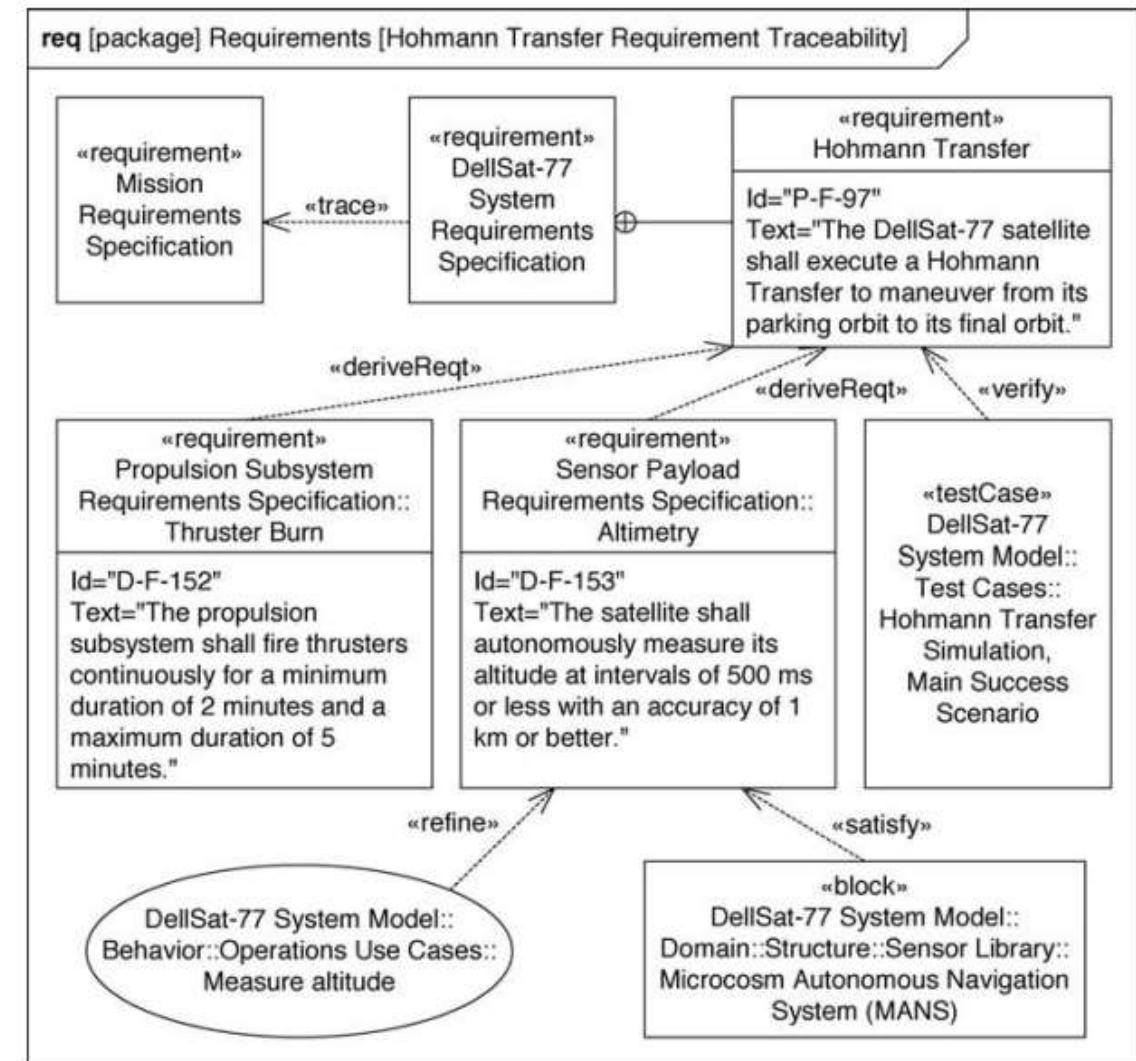


Fig: Sample requirements diagram

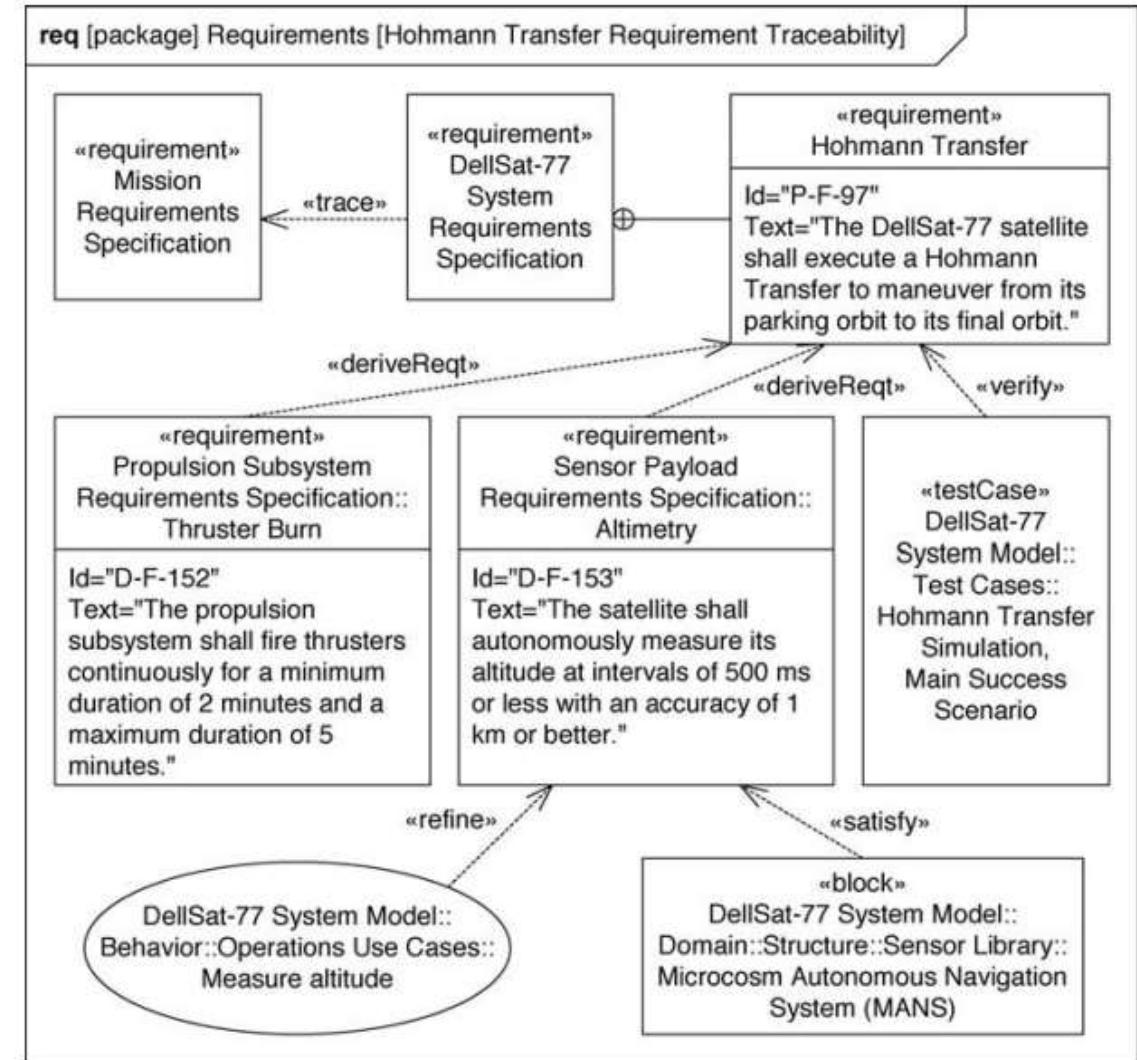
Trace Relationships



Trace relationship is used the trace relationship in Fig to convey that the DellSat-77 System Requirements Specification traces back to the Mission Requirements Specification.

Fig: Sample requirements diagram

Derive Requirement Relationships Diagram

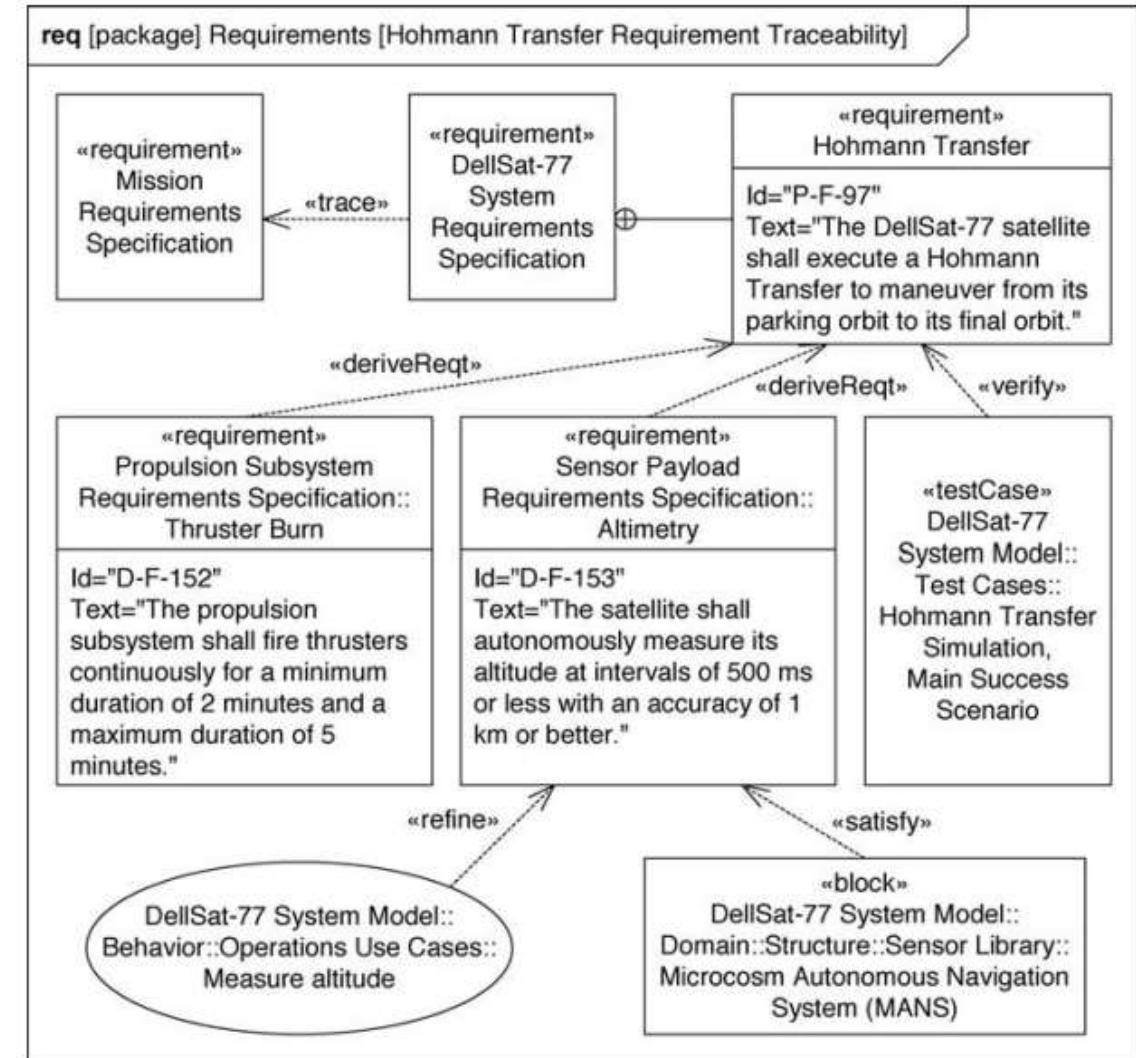


In the requirements diagram in Fig displays two derive requirement relationships. This model conveys that the Thruster Burn and Altimetry requirements are derived from the Hohmann Transfer requirement.

Thruster Burn and Altimetry are the client elements in the relationships shown; Hohmann Transfer is the supplier.

Fig: Sample requirements diagram

Refine Relationships Diagram



In the requirements diagram in Fig conveys that the Measure altitude use case refines the Altimetry requirement. The text use case specification (or the activity diagram) that accompanies the use case contains a detailed sequence of steps that specifies the required behavior of the system more precisely than the Altimetry requirement does by itself.

Fig: Sample requirements diagram

Verify Relationships Diagram

By convention, however, the client element is always a test case. A test case is simply a behavior that you define somewhere in your model—one that you create for the purpose of invoking a particular structure's functionality to verify that it satisfies one or more of the requirements allocated to it. A test case can be any one of the three kinds of behaviors:

activity, interaction, or state machine.

However, a test case is most often modeled as an interaction (and displayed on a sequence diagram).

The requirements diagram in Fig conveys that the test case named Hohmann Transfer Simulation, Main Success Scenario verifies the Hohmann Transfer requirement.

This test case is a behavior that, when executed, will prove that the system implementation actually satisfies the Hohmann Transfer requirement. (Of course, a system model typically contains many sets of test cases to fully verify all system and component requirements.)

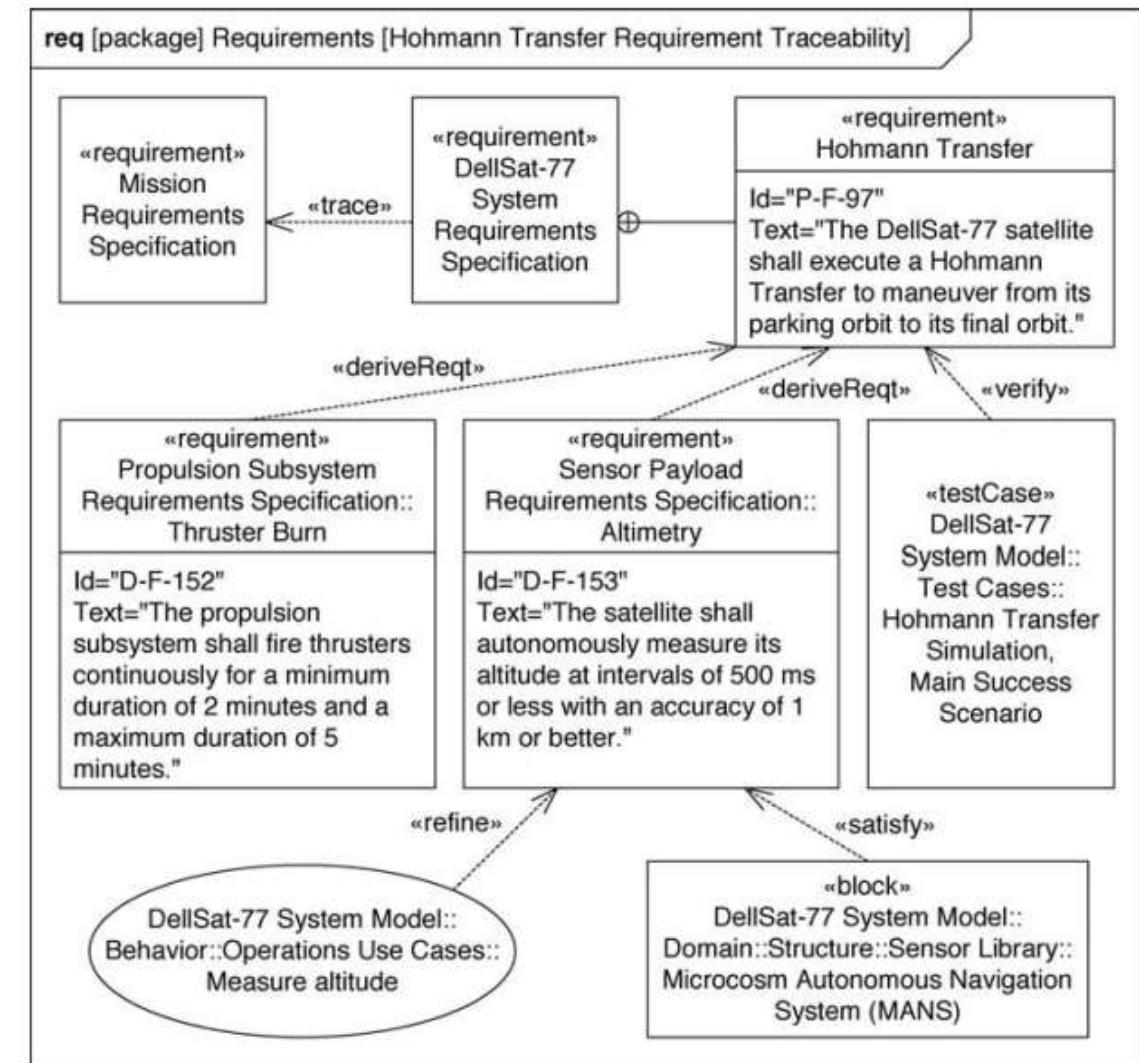


Fig: Sample requirements diagram

Satisfy Relationships Diagram

The requirements diagram in Figure 11.1 conveys that the Microcosm Autonomous Navigation System (MANS) block satisfies the Altimetry requirement.

The satisfy relationship is an assertion that an instance of the block at the client end will fulfill the requirement at the supplier end.

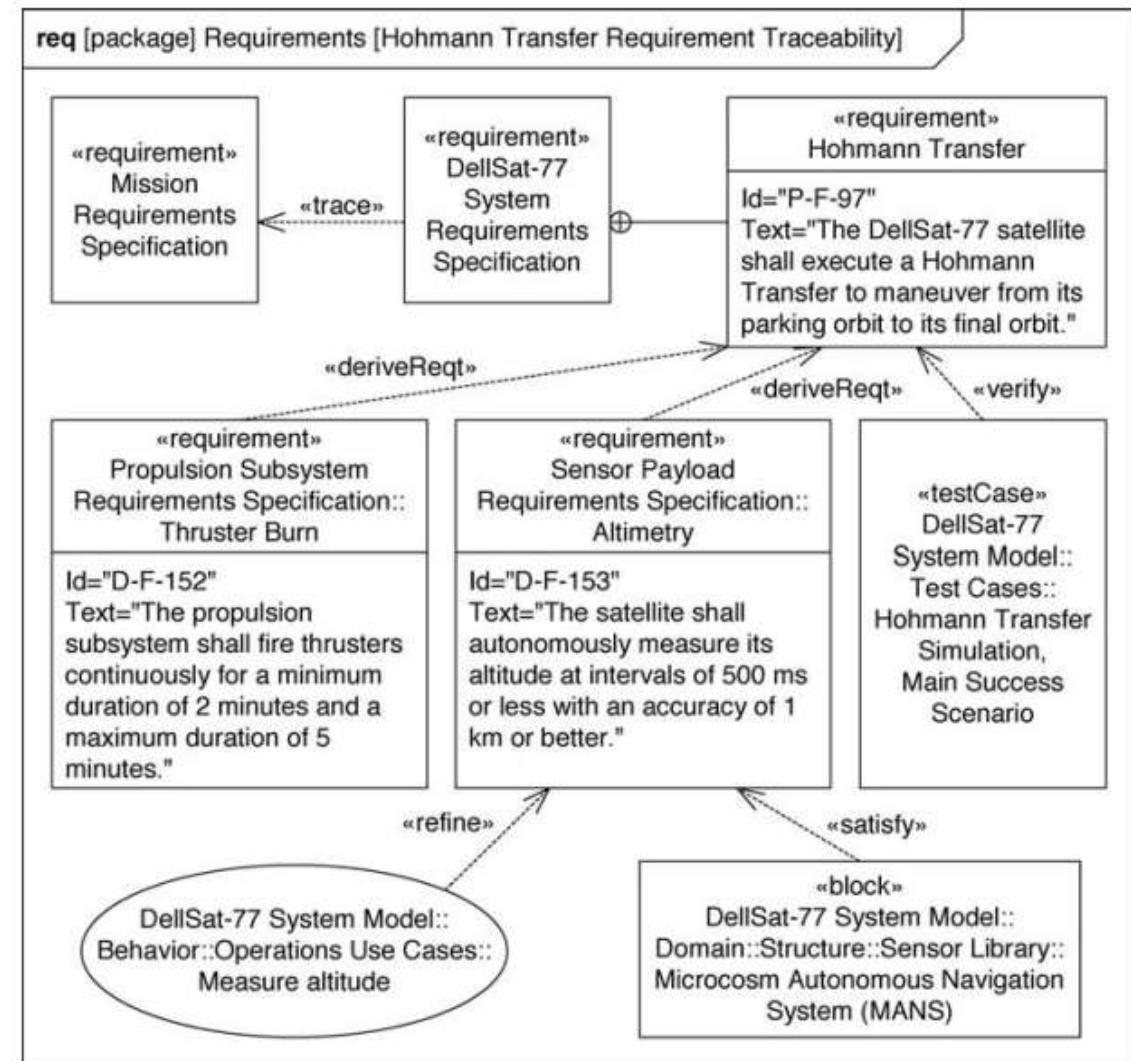


Fig: Sample requirements diagram

Notation for Requirement Relationships

Direct Notation

Dashed Line with arrow head

Compartment Notation

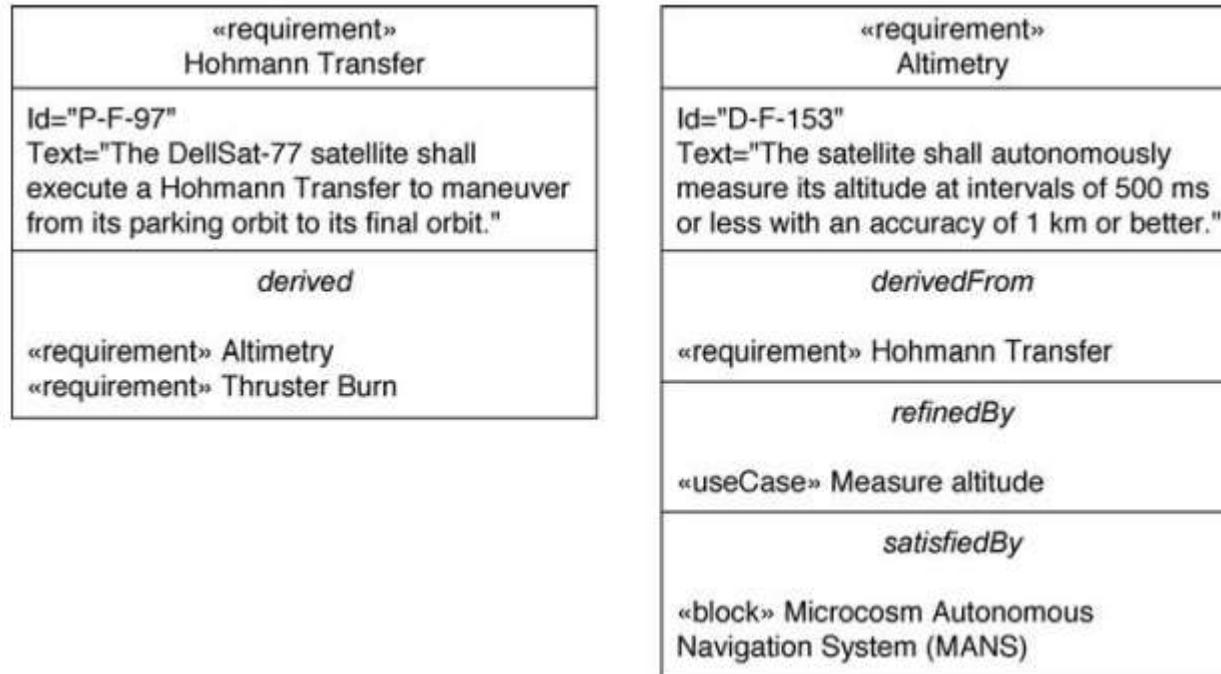


Fig: Displaying requirements relationships using compartment notation

Notation for Requirement Relationships

Callout Notation

Refers to comment anchored to an element

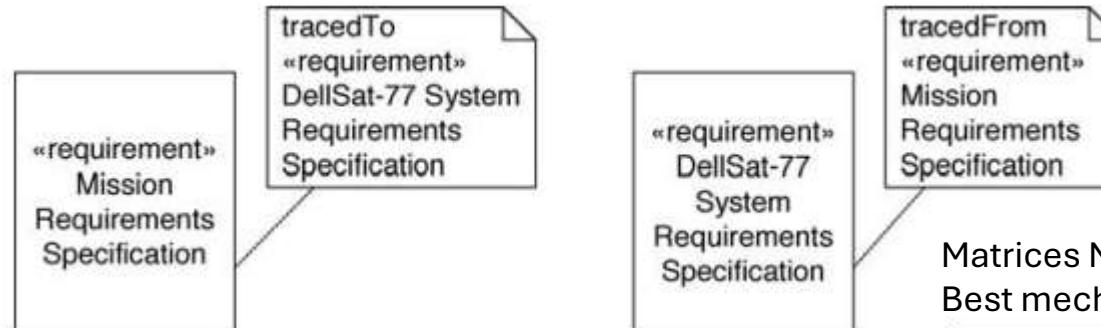


Fig: Displaying requirements relationships using callout notation

Matrices Notation
Best mechanism to display many relationships in least amount of space

Client	Supplier		
	«requirement» Mission Requirements Specification	«requirement» Hohmann Transfer	«requirement» Altimetry
«requirement» DellSat-77 System Requirements Specification	trace		
«requirement» Thruster Burn		deriveReqt	
«requirement» Altimetry		deriveReqt	
«testCase» Hohmann Transfer Simulation, Main Success Scenario		verify	
«useCase» Measure altitude			refine
«block» Microcosm Autonomous Navigation System (MANS)			satisfy

Fig: Displaying requirements relationships using matrix notation

Notation for Requirement Relationships

Tables

Commercial-grade modeling tools let you define your own custom table format, including the insertion of extra columns for any properties that you need to display. Keep in mind, however, that a table becomes cumbersome to read in printed documentation when its width spans multiple pages.

id	name	relation	id	type	name
SRS-1081	DellSat-77 System Requirements Specification	trace	MRS-1016	requirement	Mission Requirements Specification
P-F-97	Hohmann Transfer	verifiedBy		testCase	Hohmann Transfer Simulation, Main Success Scenario
D-F-152	Thruster Burn	deriveReqt	P-F-97	requirement	Hohmann Transfer
D-F-153	Altimetry	deriveReqt	P-F-97	requirement	Hohmann Transfer
		refinedBy		useCase	Measure altitude
		satisfiedBy		block	Microcosm Autonomous Navigation System (MANS)

Fig: Displaying requirements relationships using table notation

SysML Notation Desk Reference

Element Kind	Notation	Slide No.
Block	<pre> <<block>> Block Name Parts partPropertyName : Type [multiplicity] references referencePropertyName : Type [multiplicity] values valuePropertyName : Type [multiplicity] = default value constraints constraintPropertyName : Type operations operationName(parameterName : Type, ...) : returnType [multiplicity] receptions <<single>> receptionName(parameterName : Type, ...) </pre>	

Table A.1 Graphical Notations for Block Definition Diagrams (BDD)

SysML Notation Desk Reference

Element Kind	Notation	Slide No.
Actor	 <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <<actor>> Actor Name B </div>	
Flow Specification	<div style="border: 1px solid black; padding: 10px;"> <p style="text-align: center;"><<flowSpecification>></p> <p style="text-align: center;">Flow Specification Name</p> <hr/> <p style="text-align: center;"><i>flowProperties</i></p> <p>direction flowPropertyName : Type</p> </div>	
Constraint Block	<div style="border: 1px solid black; padding: 10px;"> <p style="text-align: center;"><<constraint>></p> <p style="text-align: center;">Constraint Block Name</p> <hr/> <p style="text-align: center;"><i>constraints</i></p> <p>{constraint expression}</p> <hr/> <p style="text-align: center;"><i>parameters</i></p> <p>parameterName : Type [multiplicity]</p> </div>	

Table A.1 Graphical Notations for Block Definition Diagrams (BDD) (continued)

SysML Notation Desk Reference

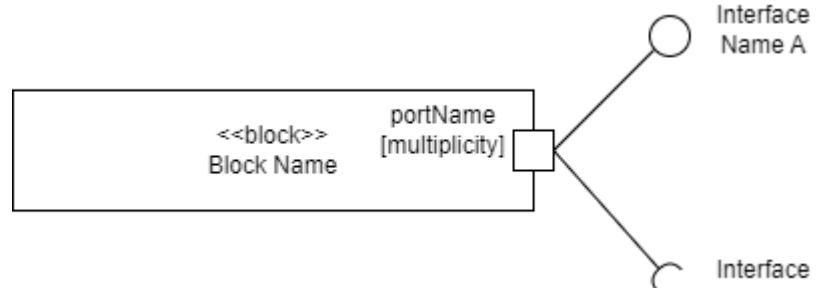
Element Kind	Notation	Slide No.
Interface	<pre> <<interface>> Interface Name operations operationName(parameterName : Type, ...) : returnType [multiplicity] receptions <<signal>> receptionName(parameterName : Type, ...) </pre>	
Signal	<pre> <<signal>> Signal Name Properties propertyName : Type [multiplicity] </pre>	
Standard Port (with Provided and Required Interfaces)	<pre> <<block>> Block Name portName [multiplicity] </pre>  <p>The diagram shows a rectangular block labeled "Block Name" with a port labeled "portName [multiplicity]". Two lines extend from the port to two circular nodes labeled "Interface Name A" and "Interface Name B".</p>	

Table A.1 Graphical Notations for Block Definition Diagrams (BDD) (continued)

SysML Notation Desk Reference

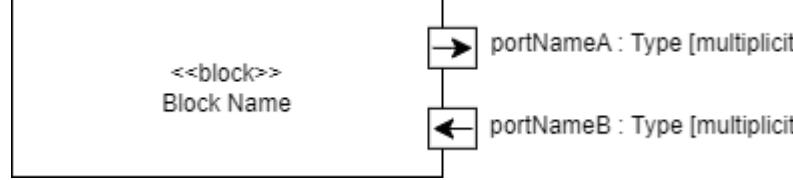
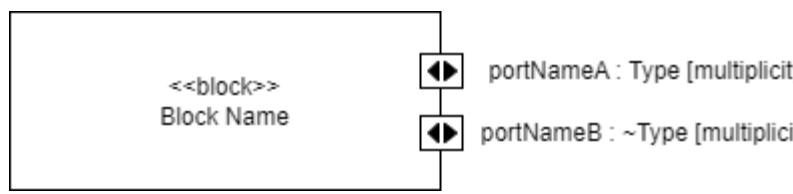
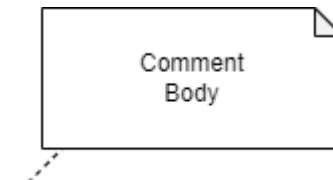
Element Kind	Notation	Slide No.
Atomic Flow Port		
Nonatomic Flow Part		
Comments		

Table A.1 Graphical Notations for Block Definition Diagrams (BDD) (continued)

SysML Notation Desk Reference

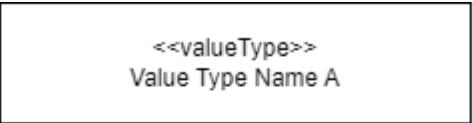
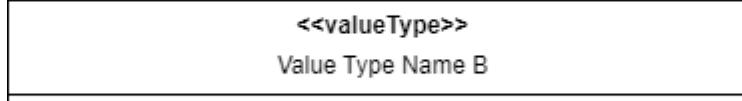
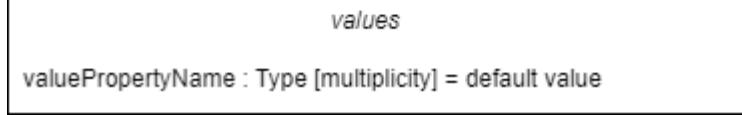
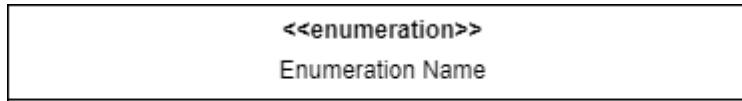
Element Kind	Notation	Slide No.
Value Type	  	
Enumeration	 	

Table A.1 *Graphical Notations for Block Definition Diagrams (BDD) (continued)*

SysML Notation Desk Reference

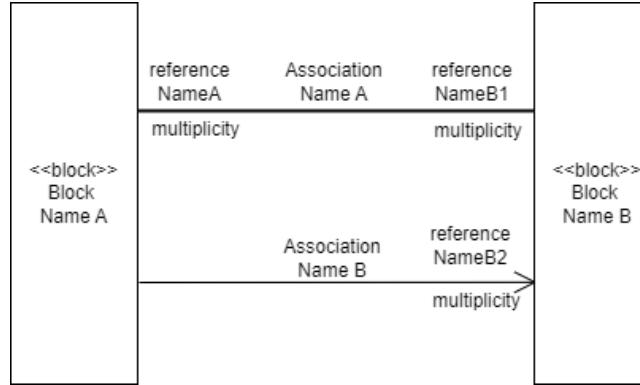
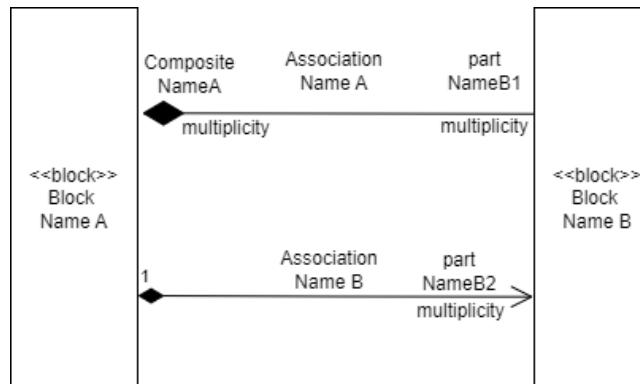
Element Kind	Notation	Slide No.
Reference Association	 <p>The diagram shows two rectangular blocks. The left block is labeled '<<block>> Block Name A' and the right block is labeled '<<block>> Block Name B'. Between them is a horizontal association line. Above the line, 'reference NameA' is connected to 'Association Name A' and 'reference NameB1' is connected to 'multiplicity'. Below the line, 'multiplicity' is connected to 'Association Name B' and 'reference NameB2' is connected to 'multiplicity'.</p>	
Composite Association	 <p>The diagram shows two rectangular blocks. The left block is labeled '<<block>> Block Name A' and the right block is labeled '<<block>> Block Name B'. Between them is a horizontal association line. Above the line, 'Composite NameA' is connected to 'multiplicity' (with a black diamond symbol) and 'part NameB1' is connected to 'multiplicity'. Below the line, 'multiplicity' is connected to 'Association Name B' (with a black diamond symbol) and 'part NameB2' is connected to 'multiplicity' (with a black diamond symbol). Additionally, there is a multiplicity '1' at the start of the line from Block A.</p>	

Table A.1 Graphical Notations for Block Definition Diagrams (BDD) (continued)

SysML Notation Desk Reference

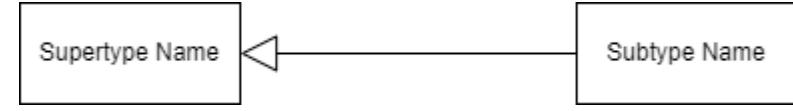
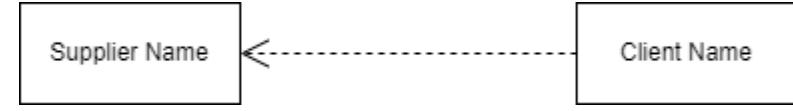
Element Kind	Notation	Slide No.
Reference Association		
Dependency		

Table A.1 *Graphical Notations for Block Definition Diagrams (BDD) (continued)*

SysML Notation Desk Reference

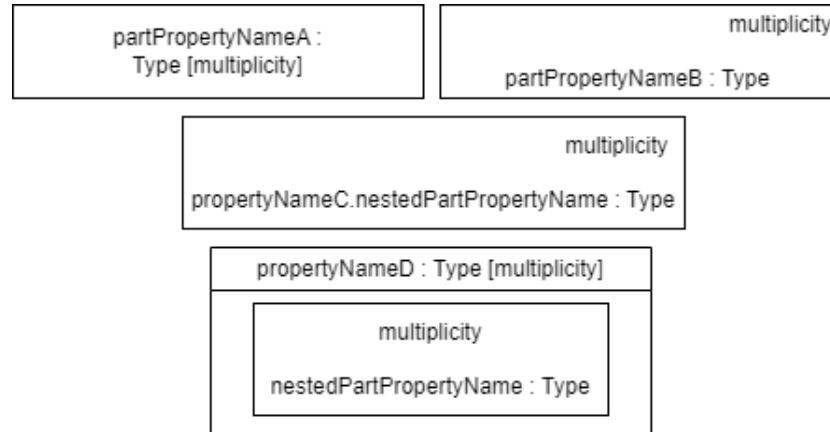
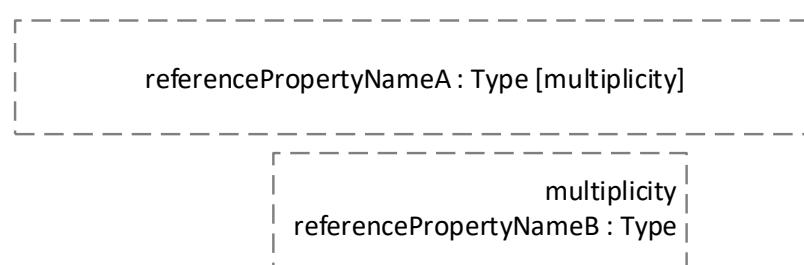
Element Kind	Notation	Slide No.
Part Property		
Reference Property		

Table A.2 Graphical Notations for Internal Block Diagrams (IBD)

SysML Notation Desk Reference

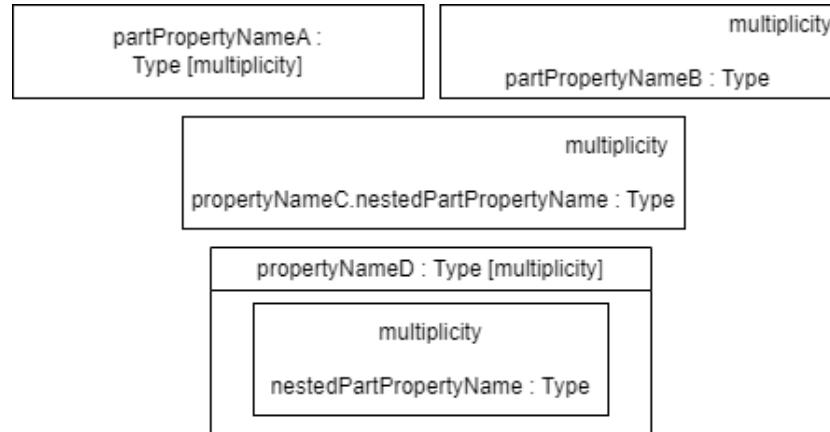
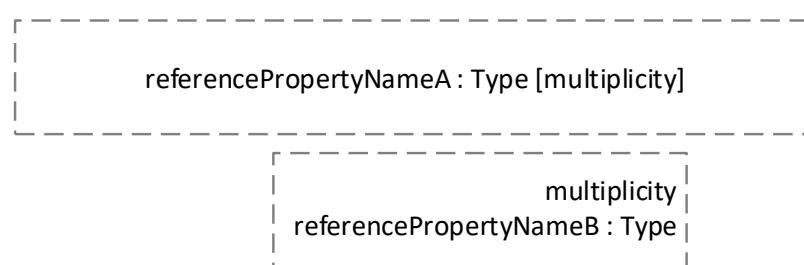
Element Kind	Notation	Slide No.
Part Property		
Reference Property		

Table A.2 Graphical Notations for Internal Block Diagrams (IBD) (continued)

SysML Notation Desk Reference

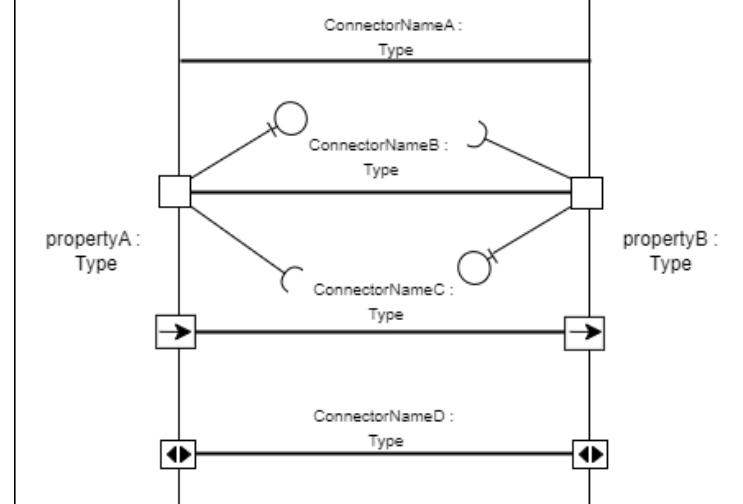
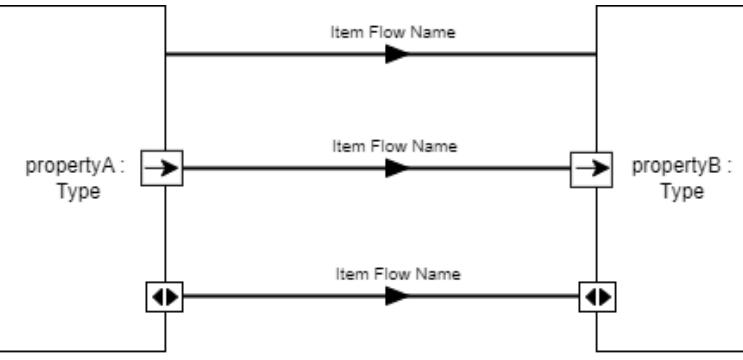
Element Kind	Notation	Slide No.
Connectors	 <p>The diagram illustrates four connectors (ConnectorNameA through ConnectorNameD) connecting two blocks. Connector A is a bidirectional connector between propertyA and propertyB. Connector B is a bidirectional connector between propertyA and propertyB. Connector C is a unidirectional connector from propertyA to propertyB. Connector D is a bidirectional connector between propertyA and propertyB.</p>	
Item Flow	 <p>The diagram illustrates three item flows (Item Flow Name) connecting two blocks. Item Flow Name A is a bidirectional item flow between propertyA and propertyB. Item Flow Name B is a unidirectional item flow from propertyA to propertyB. Item Flow Name C is a bidirectional item flow between propertyA and propertyB.</p>	

Table A.2 Graphical Notations for Internal Block Diagrams (IBD) (continued)

SysML Notation Desk Reference

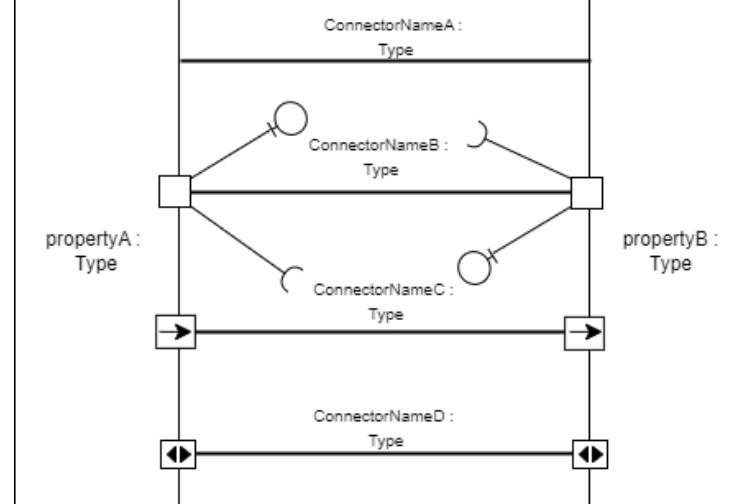
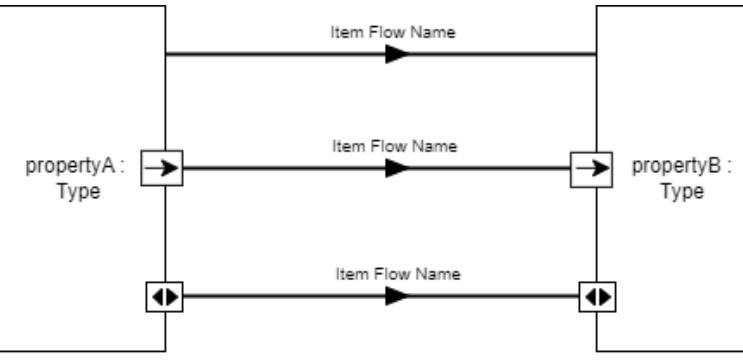
Element Kind	Notation	Slide No.
Connectors	 <p>ConnectorNameA : Type</p> <p>ConnectorNameB : Type</p> <p>ConnectorNameC : Type</p> <p>ConnectorNameD : Type</p> <p>propertyA : Type</p> <p>propertyB : Type</p>	
Item Flow	 <p>Item Flow Name</p> <p>Item Flow Name</p> <p>Item Flow Name</p> <p>propertyA : Type</p> <p>propertyB : Type</p>	

Table A.2 Graphical Notations for Internal Block Diagrams (IBD) (continued)

SysML Notation Desk Reference

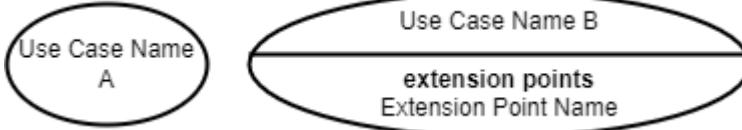
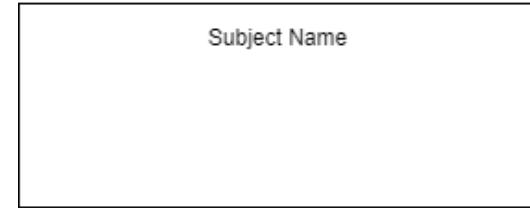
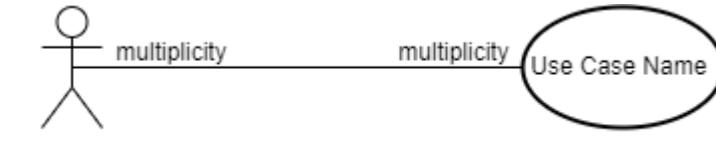
Element Kind	Notation	Slide No.
Use Case		
Actor		
System Boundary (for Subject)		
Reference Association		

Table A.3 Graphical Notations for Use Case Diagrams

SysML Notation Desk Reference

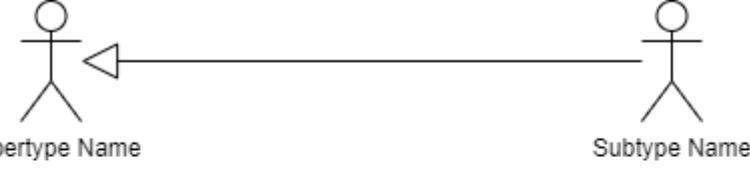
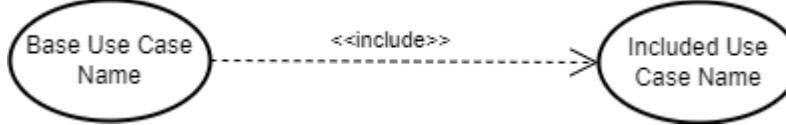
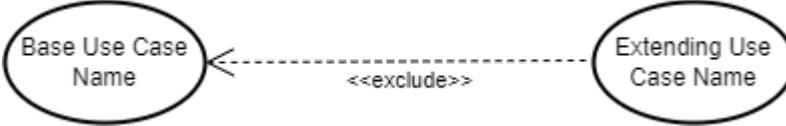
Element Kind	Notation	Slide No.
Generalization	 	
Include		
Exclude		

Table A.3 Graphical Notations for Use Case Diagrams (continued)

SysML Notation Desk Reference

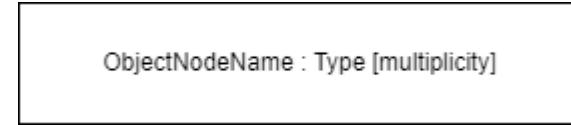
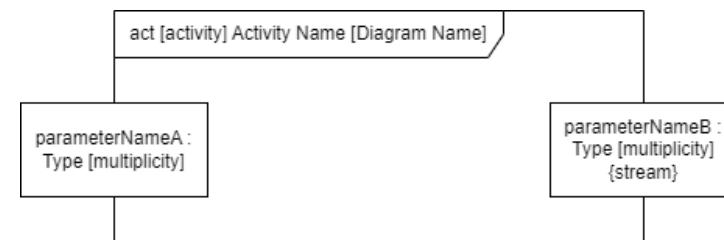
Element Kind	Notation	Slide No.
Basic Action		
Object Node		
Pin	<p>pinNameA : Type [multiplicity]</p> <p>pinNameB : Type [multiplicity]</p>  <p>{stream}</p>	
Activity Parameter	<p>act [activity] Activity Name [Diagram Name]</p> <p>parameterNameA : Type [multiplicity]</p> <p>parameterNameB : Type [multiplicity] {stream}</p> 	

Table A.4 Graphical Notations for Activity Diagram

SysML Notation Desk Reference

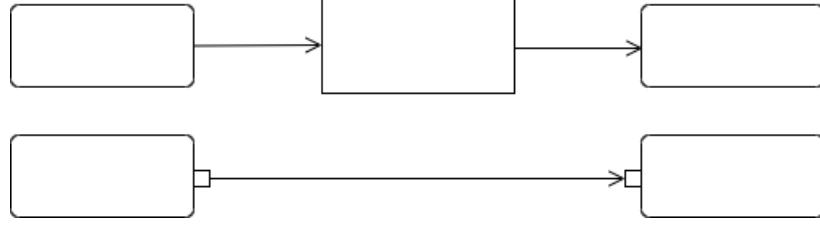
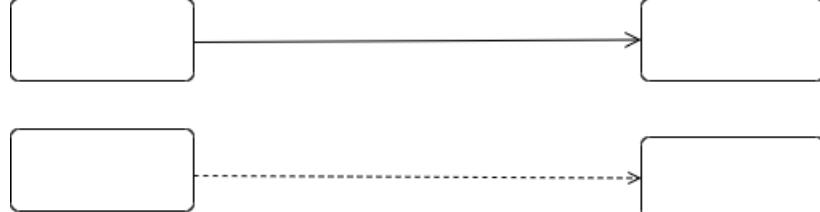
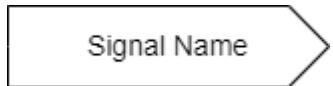
Element Kind	Notation	Slide No.
Object Flow		
Control Flow		
Call Behavior Action		
Send Signal Action		

Table A.4 Graphical Notations for Activity Diagram (continued)

SysML Notation Desk Reference

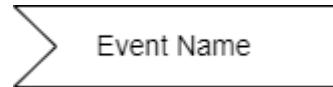
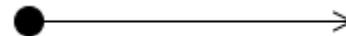
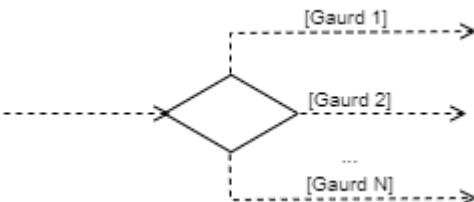
Element Kind	Notation	Slide No.
Accept Event Action		
Wait Time Action	 at (Absolute Time Expression)  after (After Time Expression)	
Initial Node		
Activity Final Node		
Decision Node		

Table A.4 Graphical Notations for Activity Diagram (continued)

SysML Notation Desk Reference

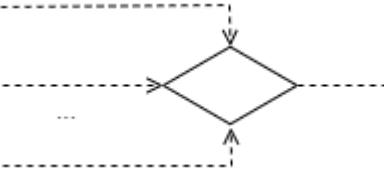
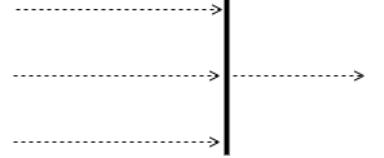
Element Kind	Notation	Slide No.
Merge Node		
Join Node		
Activity Partition		

Table A.4 Graphical Notations for Activity Diagram (continued)

SysML Notation Desk Reference

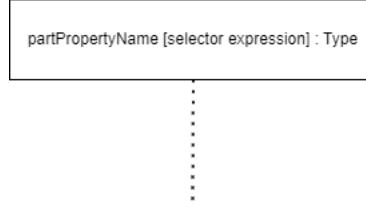
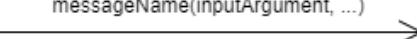
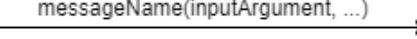
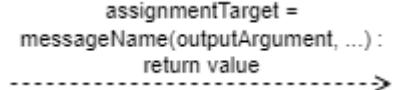
Element Kind	Notation	Slide No.
Lifeline		
Asynchronous Message		
Synchronous Message		
Reply Message		

Table A.4 Graphical Notations for Activity Diagram (continued)

SysML Notation Desk Reference

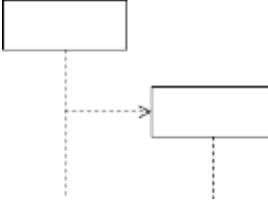
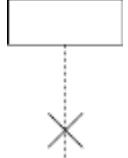
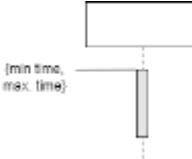
Element Kind	Notation	Slide No.
Create Message		
Destruction Occurrence		
Execution Specification		
Time Constraint		

Table A.5 Graphical Notations for Sequence Diagram

SysML Notation Desk Reference

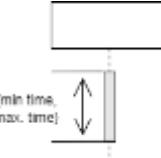
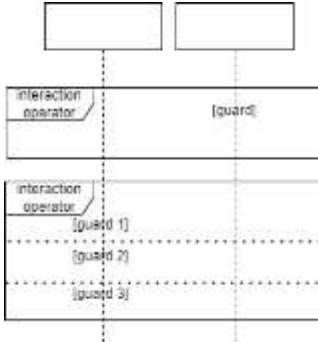
Element Kind	Notation	Slide No.
Duration Constraint		
State Invariant		
Combined Fragment		

Table A.5 Graphical Notations for Sequence Diagram (continued)

SysML Notation Desk Reference

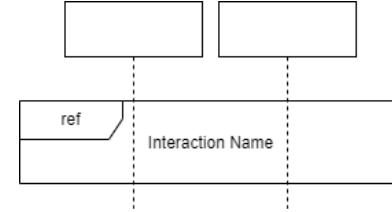
Element Kind	Notation	Slide No.
Interaction Use		

Table A.5 *Graphical Notations for Sequence Diagram (continued)*

SysML Notation Desk Reference

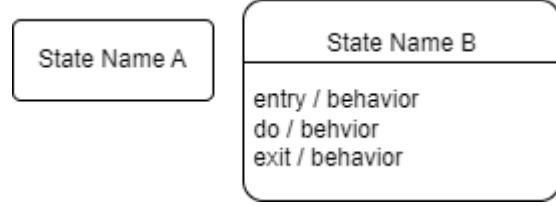
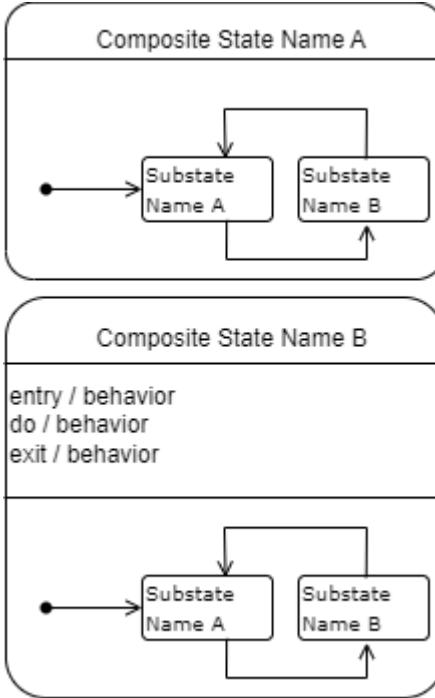
Element Kind	Notation	Slide No.
Simple State		
Composite State		

Table A.6 Graphical Notations for Sequence Diagram (continued)

SysML Notation Desk Reference

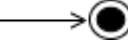
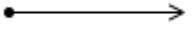
Element Kind	Notation	Slide No.
Final State		
External Transition	<p><u>signalName(argument, ...)</u> [guard] / effect →</p> <p><u>operationName(argument, ...)</u> [guard] / effect →</p> <p><u>at (Absolute Time Expression)</u> [guard] / effect →</p> <p><u>after (Relative Time Expression)</u> [guard] / effect →</p> <p><u>when (Boolean Expression)</u> [guard] / effect →</p>	
Internal Transition	<div style="border: 1px solid black; padding: 10px; width: fit-content;"> <p style="text-align: center;">State Name</p> <p>signalName(argument,...) [guard] / effect operationName(argument, ...) [guard] / effect at (Absolute Time Expression) [guard] / effect when (Boolean Expression) [guard] / effect</p> </div>	
Internal Pseudo state		

Table A.6 Graphical Notations for Sequence Diagram (continued)

SysML Notation Desk Reference

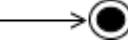
Element Kind	Notation	Slide No.
Final State		
Junction Pseudostate		

Table A.6 *Graphical Notations for Sequence Diagram (continued)*

SysML Notation Desk Reference

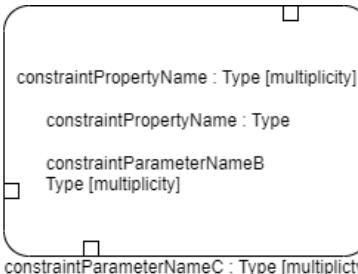
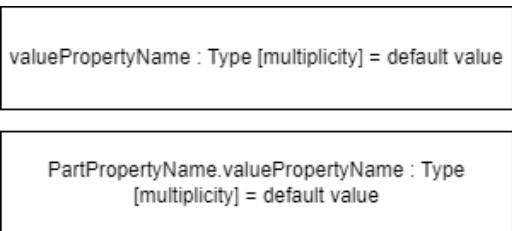
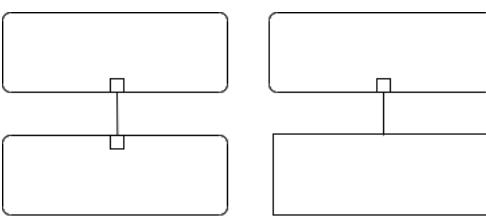
Element Kind	Notation	Slide No.
Constraint Property	 constraintPropertyName : Type	
Constraint Parameter	 constraintPropertyName : Type [multiplicity] constraintPropertyName : Type constraintParameterNameB : Type [multiplicity] constraintParameterNameC : Type [multiplicity]	
Value Property	 valuePropertyName : Type [multiplicity] = default value PartPropertyName.valuePropertyName : Type [multiplicity] = default value	
Binding Connector		

Table A.7 Graphical Notations for Sequence Diagram (continued)

SysML Notation Desk Reference

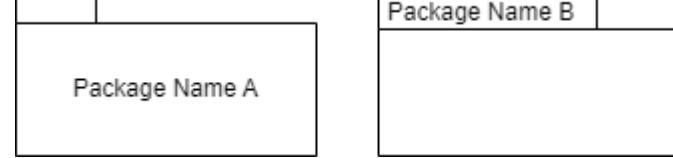
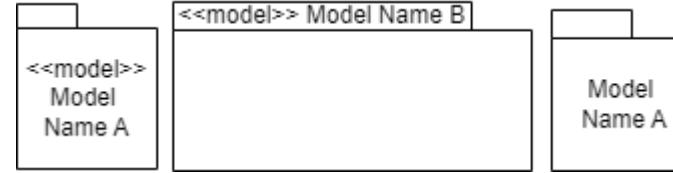
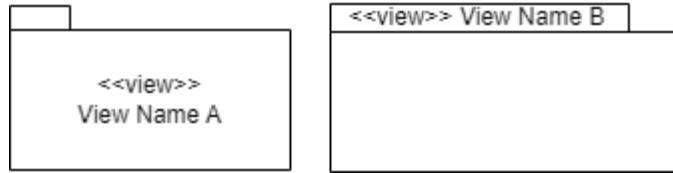
Element Kind	Notation	Slide No.
Package		
Model		
Model Library		
Profile		

Table A.8 Graphical Notations for Package Diagram

SysML Notation Desk Reference

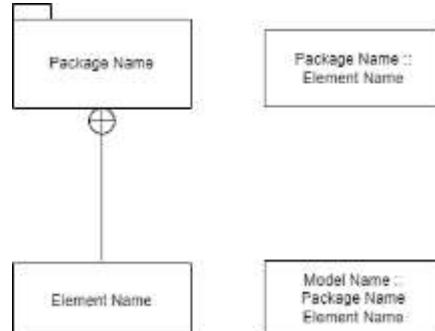
Element Kind	Notation	Slide No.
Viewpoint	<pre> <<viewpoint>> Viewpoint Name <<viewpoint>> stakeholders = "text string" concerns = "text string" purpose = "text string" languages = "text string" methods = "text string" </pre>	
Namespace Containment		
Dependency		

Table A.8 Graphical Notations for Package Diagram (continued)

SysML Notation Desk Reference

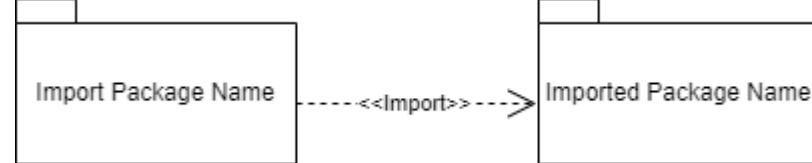
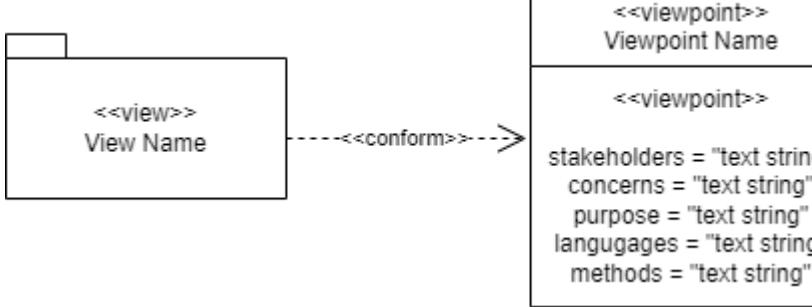
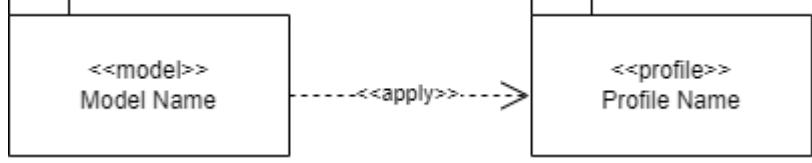
Element Kind	Notation	Slide No.
Imported Package	 <pre> graph LR IP[Import Package Name] -- "<<Import>>" --> IP_IMPORTED[Imported Package Name] </pre>	
Conform	 <pre> graph LR V[<<view>> View Name] -- "<<conform>>" --> VP[<<viewpoint>> Viewpoint Name stakeholders = "text string" concerns = "text string" purpose = "text string" languages = "text string" methods = "text string"] </pre>	
Profile Application	 <pre> graph LR M[<<model>> Model Name] -- "<<apply>>" --> P[<<profile>> Profile Name] </pre>	

Table A.8 Graphical Notations for Package Diagram (continued)

SysML Notation Desk Reference

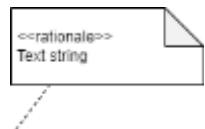
Element Kind	Notation	Slide No.
Requirement	<pre> <<requirement>> Requirement Name Id = "text string" text = "text string" tracedTo <<elementKind>> Element Name A tracedFrom <<elementKind>> Element Name B derived <<requirement>> Requirement Name A derivedFrom <<requirement>> Requirement Name B refinedBy <<elementKind>> Element Name C satisfied <<block>> Block Name verifiedBy <<testCase>> Test Case Name </pre>	
Rationale		

Table A.9 Graphical Notations for Requirement Diagrams

SysML Notation Desk Reference

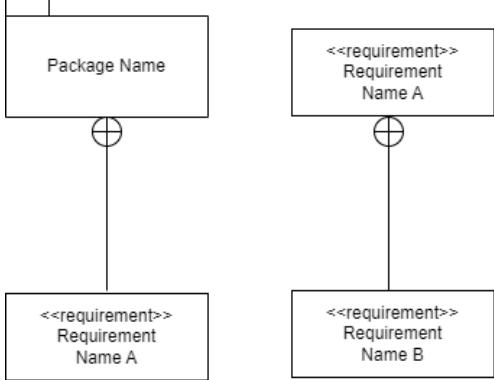
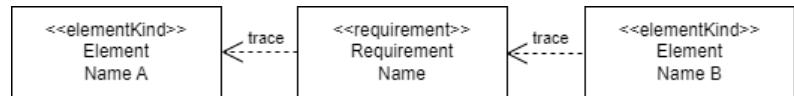
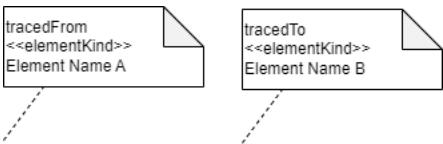
Element Kind	Notation	Slide No.
Namespace Containment	 <pre> graph TD PN[Package Name] --- RA1["<<requirement>> Requirement Name A"] PN --- RA2["<<requirement>> Requirement Name B"] RA1 --- P RA2 --- P </pre>	
Trace (Callout Notation)	 <pre> graph LR EA["<<elementKind>> Element Name A"] -- trace --> ER["<<requirement>> Requirement Name"] ER -- trace --> EB["<<elementKind>> Element Name B"] </pre>	
Derive Requirement (Direct Notation)	 <pre> graph LR AF["tracedFrom
<<elementKind>> Element Name A"] -.-> AT["tracedTo
<<elementKind>> Element Name B"] </pre>	

Table A.9 Graphical Notations for Requirement Diagram (continued)

SysML Notation Desk Reference

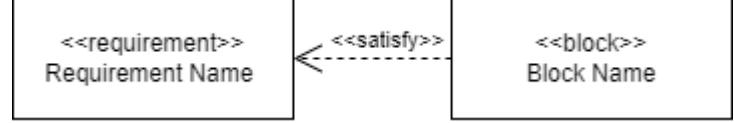
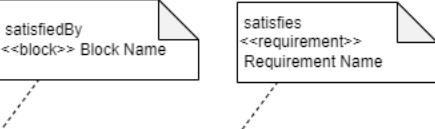
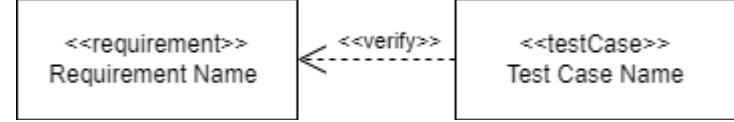
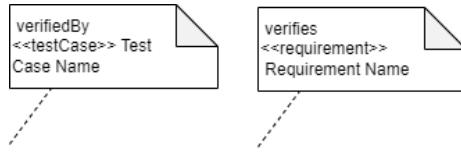
Element Kind	Notation	Slide No.
Satisfy (Callout Notation)		
Satisfy (Callout Notation)		
Verify (Direct Notation)		
Verify (Callout Notation)		

Table A.9 Graphical Notations for Requirement Diagram (continued)

SysML Notation Desk Reference

Element Kind	Notation	Slide No.
Allocation (Direct Notation)	<pre> <<elementKind>> Element Name A -----> <<elementKind>> Element Name B </pre>	
Allocation (Compartment Notation)	<pre> <<elementKind>> Element Name A -----> <<elementKind>> Element Name B -----> <<elementKind>> Element Name A -----> <<elementKind>> Element Name A </pre>	
Allocation (Callout Notation)	<pre> allocatedTo <<elementKind>> Element Name B -----> allocatedFrom <<elementKind>> Element Name A </pre>	
Allocation Activity Partition	<pre> <<allocate>> Element Name -----> <<allocate>> Element Name : Type </pre>	
Rationale	<pre> rationale Text string -----> <<elementKind>> Element Name A -----> <<elementKind>> Element Name B </pre>	

Table A.10 Graphical Notations for Allocations

Module 4:
Requirement Based Testing



Tuesday, 10th & 11th September



09:00 AM – 10:00 AM



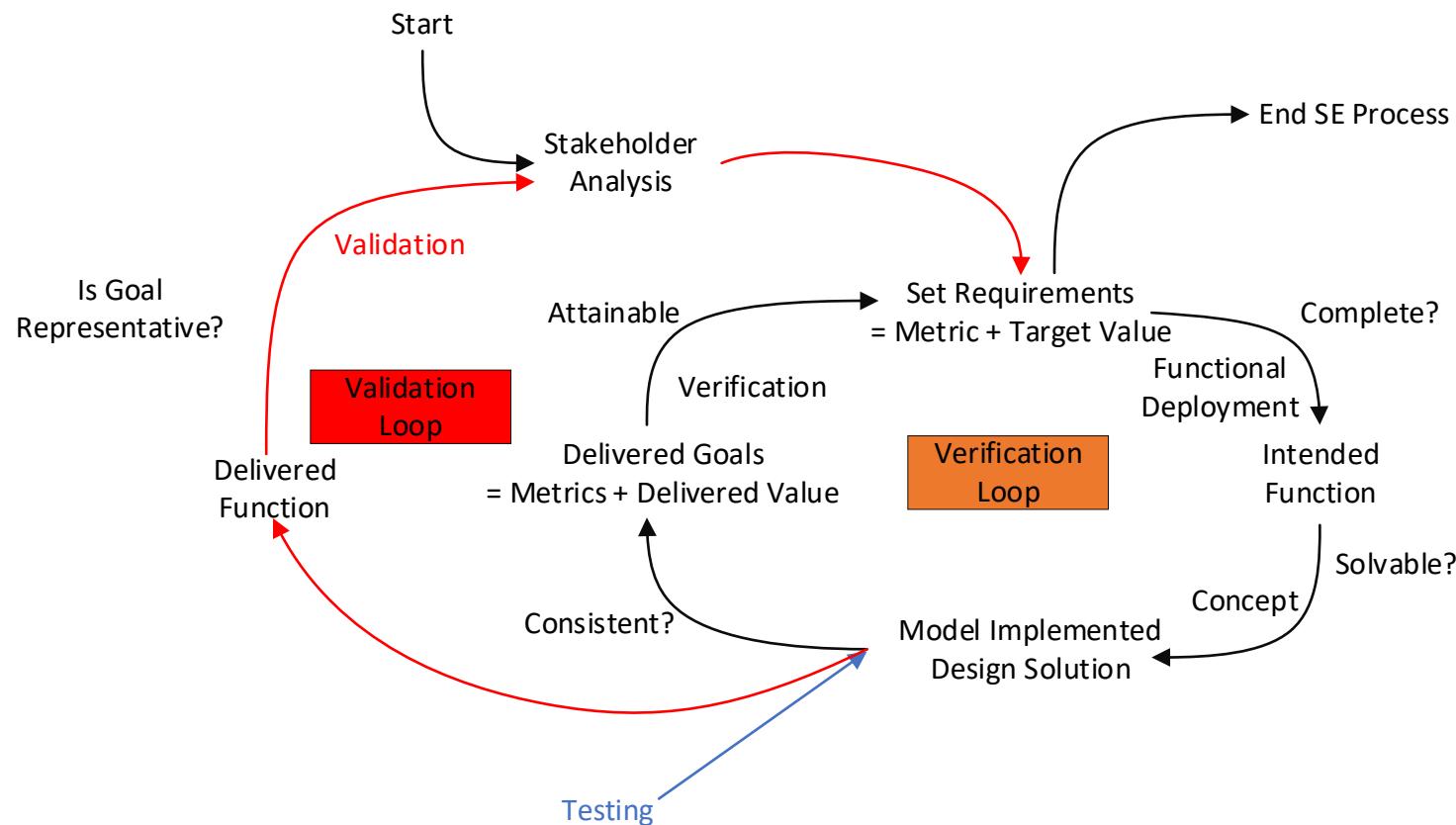
SAHIL DATERO
Business Analyst, Delivery

Agenda:

- Where does testing come into picture?
- Product Verification Best Practices
- Introduction to V & V Reviews
- Testing Validation Checklist
- Relationship between test cases and use cases

Verification and Validation

Testing aims to detect deviations from the specification and to check whether the system satisfies the defined acceptance criteria



*Was the end product realized right?
Did you implement it correctly?*

Verification

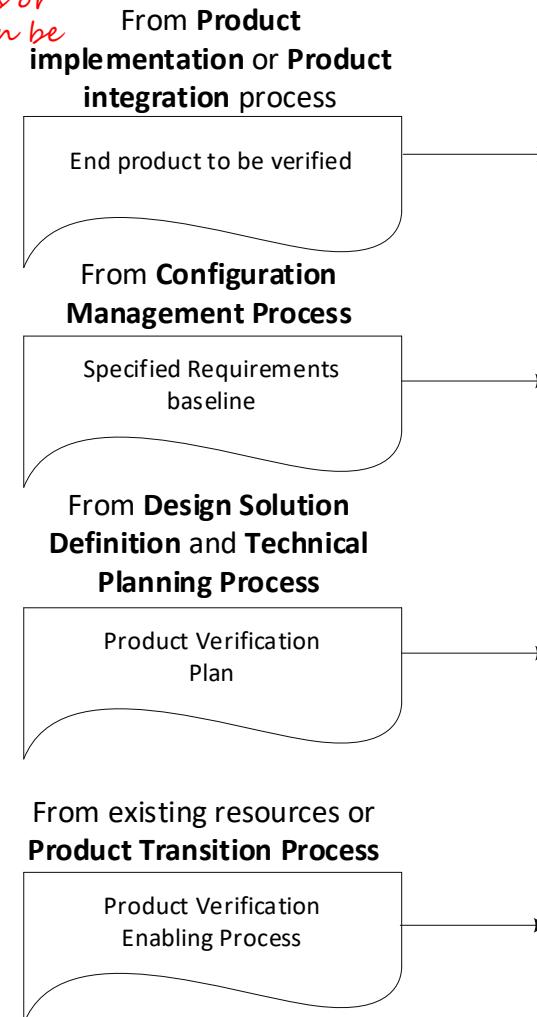
- During Development
- Check if requirements are met
- Typically in the laboratory
- Component/ Subsystem centric

*Was the right end product realized?
Did you deploy the right solution?
Did you actually build the right thing?*

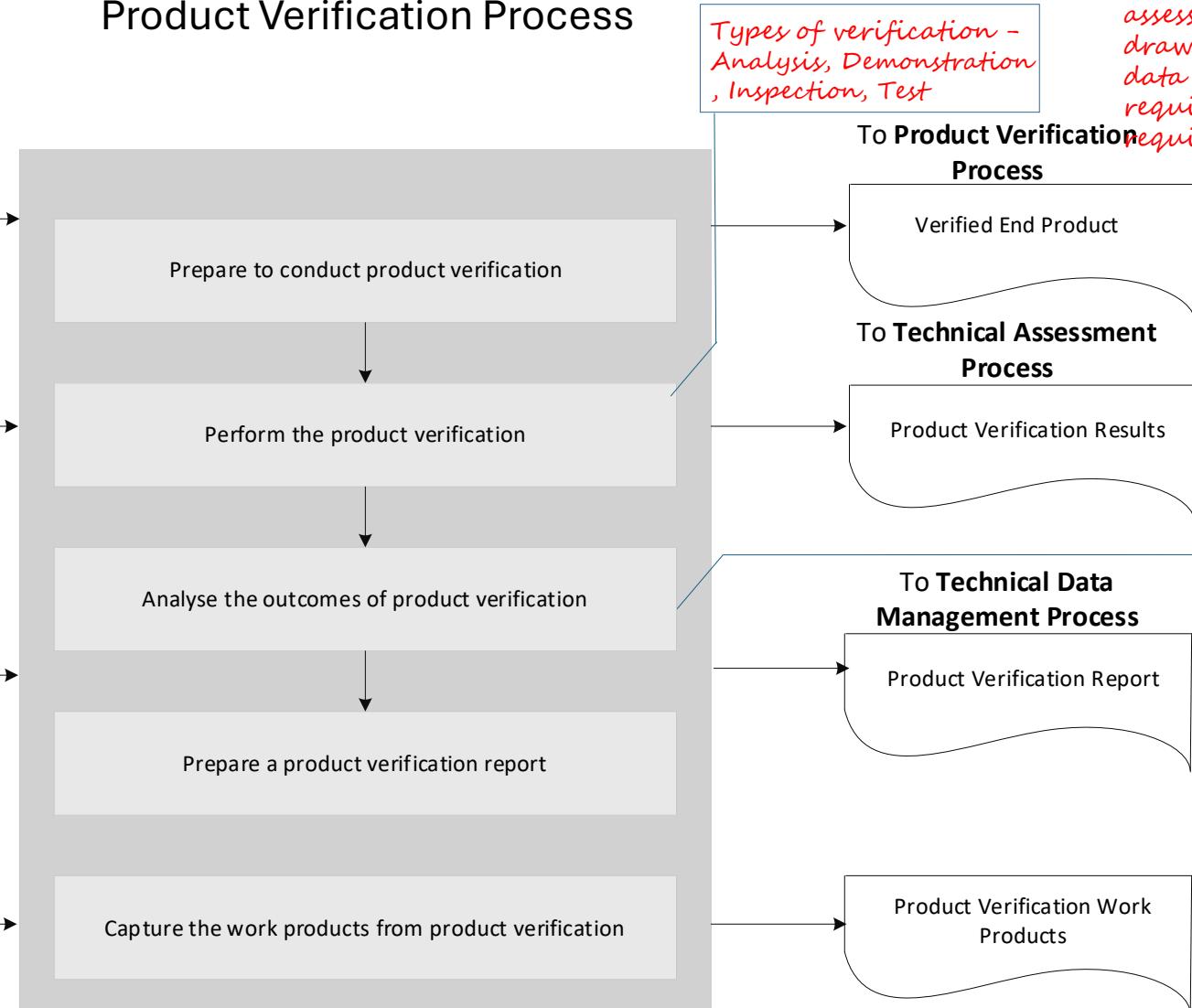
Validation

- During or after integration
- Typically in real or simulated mission environment
- Check if stakeholder intent is met
- Full up system

Analysis - a quantitative and qualitative (can be both) assessment of a requirement or group of requirements against measurable standards or data / results that can be correlated to said requirements.



Product Verification Process



Inspection - a qualitative assessment to verify a drawing or document or data against a requirement or group of requirements

Outputs - Discrepancy Reports, Verified Product, Compliance Documentation

Simulation - Quantitative testing that can provide functionally equivalent results of the requirement or set of requirements by mimicking in aircraft operational environments.

Testing - only method by which behavior of actual requirements and/or device and/or system can be observed and quantitatively measured.

Lifecycle Reviews

Review	Title	Purpose
P/SRR	Program Requirement Review	The P/SRR is used to ensure that the program requirements are properly formulated and correlated with the Agency and mission directorate strategic objectives
P/SDR	Program Definition Review, or System Definition Review	The P/SDR ensures the readiness of the program for making a program commitment agreement to approve project formulation startups during program Implementation Phase
MCR	Mission Concept Review	The MCR affirms the mission need and examines the proposed mission's objectives and the concept for meeting those objectives
SRR	System Requirement Review	The SRR examines the functional and performance requirements defined for the system and the preliminary program or project plan and ensures that the requirements and the selected concept will satisfy the mission
MDR	Mission Definition Review	The MDR examines the proposed requirements, the mission architecture, and the flow down to all functional elements of the mission to ensure that the overall concept is complete, feasible, and consistent with available resources
SDR	System Definition Review	The SRR examines the functional and performance requirements defined for the system and the preliminary program or project plan and ensures that the requirements and the preliminary program or project plan and ensures that the requirements and the selected concept will satisfy the mission
MDR	Mission Definition Review	The MDR examines the proposed requirements, the mission architecture, and the flow down to all functional elements of the mission to ensure that the overall concept is complete, feasible and consistent with available resources
SDR	System Definition Review	The SDR examines the proposed system architecture and design and the flow down to all functional elements of the system
PDR	Preliminary Design Review	The PDR demonstrates that the preliminary design meets all system requirements with acceptable risk and within the cost and schedule constraints and establishes the basis for proceeding with detailed design. It will show that the correct design options have been selected, interfaces have been identified, and verification methods have been described
CDR	Critical Design Review	The CDR demonstrates that the maturity of the design is appropriate to support proceeding with full scale fabrication, assembly, integration, and test. CDR Determines that the technical effort is on track to complete the flight and ground system development and mission operations, meeting mission performance requirements within the identified cost and schedule constraints.
PRR	Production Readiness Review	A PRR is held for FS&GS projects developing or acquiring multiple or similar systems greater than three or as determined by the project. The PRR determines the readiness of the system developers to efficiently produce the required number of systems. It ensures that the production plans; fabrication, assembly, and integration enabling products; and personnel are in place and ready to begin production

Lifecycle Reviews

Review	Title	Purpose
SIR	System Integration Review	An SIR ensures that system is ready to be integrated, segments, components, and subsystems are available and ready to be integrated into system. Integration facilities, support personnel, and integration plans and procedures are ready for integration
TRR	Test Readiness Review	A TRR ensures that the test article (hardware/ software), test facility, support personnel and test procedures are ready for testing and data acquisition, reduction and control
SAR	System Acceptance Review	The SAR verifies the completeness of the specific end products in relation to their expected maturity level and assesses compliance to stakeholder expectations. The SAR examines the system, its end products and documentation, and test data and analyses that support verification. It also ensures that the system has sufficient technical maturity to authorize its shipment to the designated operational facility or launch site.
ORR	Operational Readiness Review	The ORR examines the actual system characteristics and the procedures used in the system or end product's operation and ensures that all system and support (flight and ground) hardware, software, personnel, procedures, and user documentation accurately reflect the deployed state of the system.
FRR	Flight Readiness Review	The FRR examines tests, demonstrations, analyses and audits that determine the system's readiness for a safe and successful flight or launch and for subsequent flight operations. It also ensures that all flight and ground hardware, software, personnel, and procedures are operationally ready.
PLAR	Post-Launch Assessment Review	A PLAR is a post-deployment evaluation of the readiness of the systems (spacecraft) to proceed with full, routine operations. The review evaluates the status, performance, and capabilities of the project evident from the flight operations experience since launch. This can also mean assessing readiness to transform responsibility from the development organization to the operations organization. The review also evaluates the status of the project plans and the capability to conduct the mission with emphasis on near-term operations and mission-critical events. This review is typically held after the early flight operations and initial checkout.
CERR	Critical Event Readiness Review	A CERR confirms the project's readiness to execute the mission's critical activities during flight operation.
PFAR	Post-Flight Assessment Review	A PFAR evaluates the activities from the flight after recovery. The review identifies all anomalies that occurred during the flight and mission and determines the actions necessary to mitigate or resolve the anomalies for future flights
DR	Decommissioning Review	A DR confirms the decision to terminate or decommission the system and assesses the readiness of the system for the safe decommissioning and disposal of system assets.

Test Plan and Validation Checklist

Appendix I : Test Plan Sample Outline

1. Introduction
 - 1.1 Purpose and Scope
 - 1.2 Responsibility and Change Authority
 - 1.3 Definitions
2. Applicable and Reference Documents
 - 2.1 Applicable Documents
 - 2.2 Reference Documents
 - 2.3 Order of Procedure
3. System X Description
 - 3.1 System X Requirement Flow Down
 - 3.2 System X Architecture
 - 3.3 End Item Architecture
 - 3.4 Other Architecture Description
4. Verification and Validation Plan
 - 4.1 Verification and Validation Management Responsibilities
 - 4.2 Verification Methods
 - 4.2.1 Analysis
 - 4.2.2 Demonstration
 - 4.2.3 Inspection
 - 4.2.4 Test
 - 4.2.4.1 Qualification Testing
 - 4.2.4.2 Other Testing
 - 4.3 Validation Methods
 - 4.4 Certification Process
 - 4.5 Acceptance Testing
 5. Verification and Validation Implementation
 6. System X End Item Verification and Validation

Validation Aspects	Levels		
	Level 1 Minimal Validation	Level 2 Standard Validation	Level 3 Advanced Validation
CONTEXT CONSIDERATION			
Wrong Context Information		✓	
Incomplete Requirement Sources		✓	
Insufficiently considered context info		✓	
ARTEFACTS-CONTENT DIMENSION			
Requirement biased by a specific solution		✓	
Missing traceability information	✓		
Inconsistencies in requirements	✓		
Incorrect requirements	✓		
Untestable requirements		✓	
ARTEFACTS-DOCUMENTATION DIMENSION			
Requirements in the wrong documentation format		✓	
Broken documentation rules	✓		
ARTEFACTS - AGREEMENT DIMENSION			
Requirements not agreed	✓		
Requirements not agreed anew after modification		✓	
Unresolved conflicts		✓	
Undetected conflicts		✓	
ACTIVITY EXECUTION			
All relevant stakeholders involved in activity execution		✓	
All prescribed activities performed		✓	
All prescribed outputs created		✓	
Execution of the activities documented correctly		✓	
All relevant inputs considered for each activity		✓	

I Subject Facet

- Have all relevant objects been identified and documented?
- Have all identified objects been captured completely and correctly?
- Have all quality requirements of objects been captured?
- Have all relevant legal requirements for the representation of objects in the system (eg protection of data privacy) been considered?
- Have all relevant requirement sources been incorporated in the identification of objects and their properties and relationships?

II IT System Facet

- Have the relevant properties of test articles with which system interacts captured completely and correctly?
- Have all required system interfaces and the relevant protocols at each interface been documented complexly and correctly?
- Have all relevant IT Strategies been considered appropriately?
- Have all relevant IT policies such as installation, update, and backup policies been considered and documented?
- Have all relevant requirement sources of the IT system facet been involved in the requirements elicitation activities?

III Usage Facet

- Have the interaction with environment been captured completely?
- Have quality requirements for interaction with environment been considered (eg desired performance, robustness, incorrect inputs?)
- Have the specifics of different user groups been considered?
- Have usage goals of relevant sources been elicited completely?

IV Development Facet

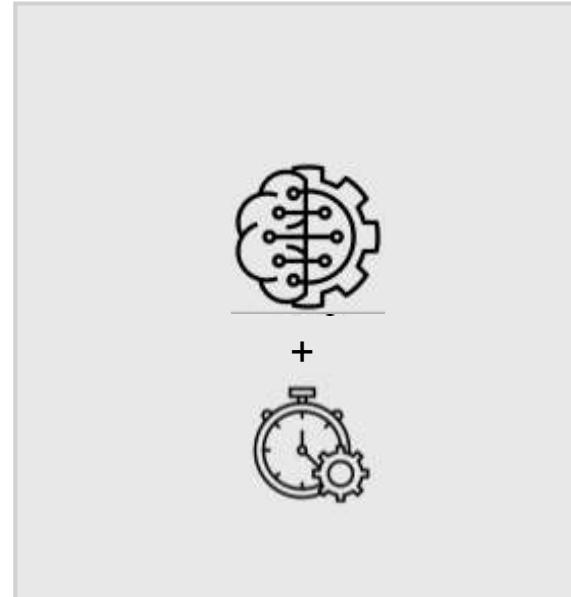
- Have all requirements pertaining to the languages and development tools to be used been captured completely, correctly?
- Have the requirements for the development process and development standard, including development rules and guidelines with which the project must comply been captured completely?
- Have all development artefacts to be provided to client identified?
- Have targeted project duration and cost have been agreed on?

Requirement Based Testing

Testing aims to detect deviations from the specification and to check whether the system satisfies the defined acceptance criteria



Execute software intensive systems
In a controlled environment
to detect defects



Test Case Definition
Test Execution

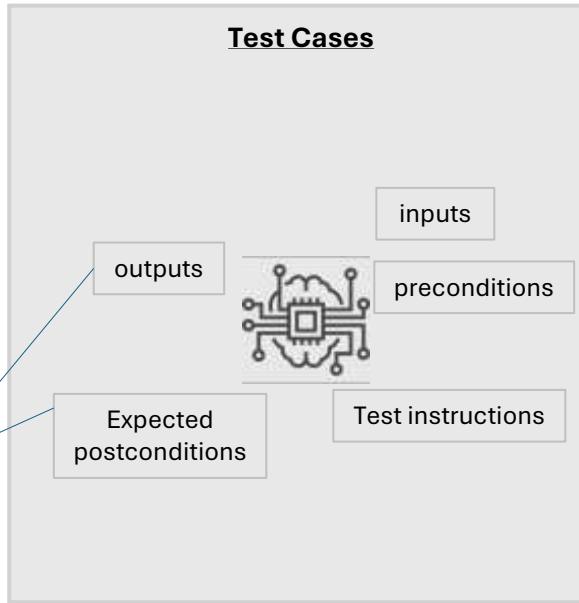


Defect Detection
Better requirement artefacts

*Failures denote fault in
test objects*

Main Concepts behind testing

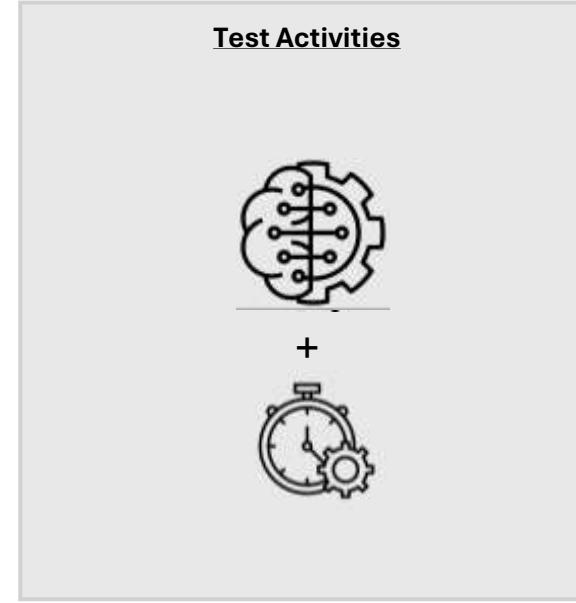
Following is introductory to main concepts behind testing software intensive systems that are relevant to requirement based testing



Failure or passing of test object

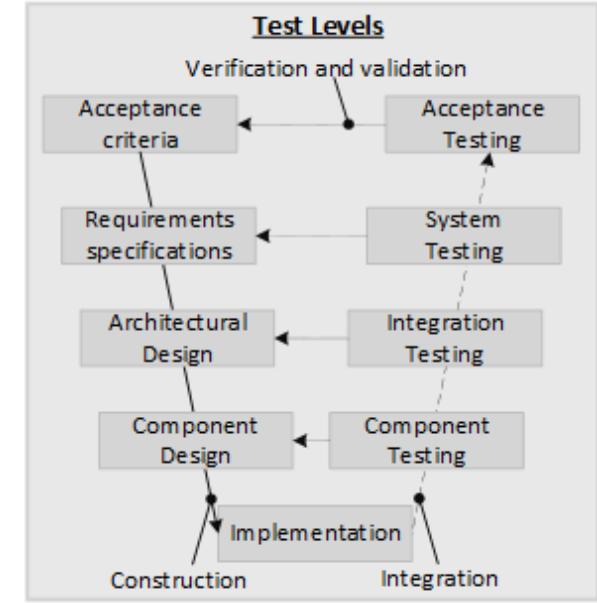
Specifies information needed for test Execution

Testers define which properties of the system to be tested shall be checked and how they shall be checked? The result of this activity is a set of test cases



Test Case Definition
Test Execution

The system is executed for each previously defined test case and the deviation of the observed output from the expected output (as specified in test case) is evaluated



Pre-release version of software
Acceptance of different user Contract as per acceptance criteria

Requirement based definition of test cases

Deriving test cases from requirements artefacts

Requirements – based definition of Test case



Requirement artefact are basis for test case derivation

Missing realisations can be detected during test execution

Undetected faults in requirements artifacts can be found during test case derivation.

Model – based definition of Test case

No. of tested states / No. of all states

No. of tested transitions / No. of all transitions

No. of tested events / No. of all events

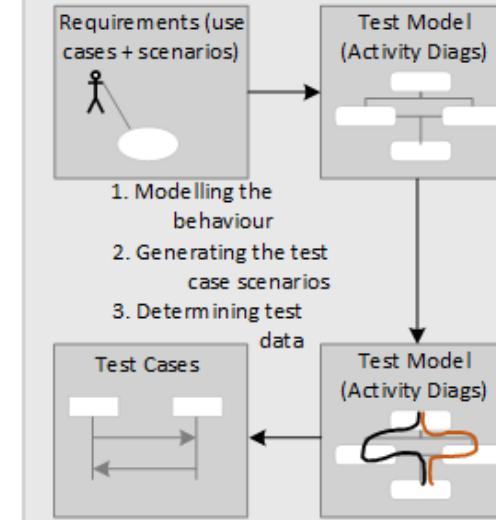
No. of tested activities / No. of all activities

No. of tested paths through test modes/ No. of all possible paths through test models

SysML, UML serve as basis for derivation of test models

Selection of test cases according to test coverage criteria

ScenTED Approach



Modelling the behavior: The complete system behavior is modeled in a test model based on use cases and their scenarios. UML 2, Activity diags

Generating Test-case scenarios:
Coverage Criteria

Tracing Test cases: Concrete Inputs and outputs

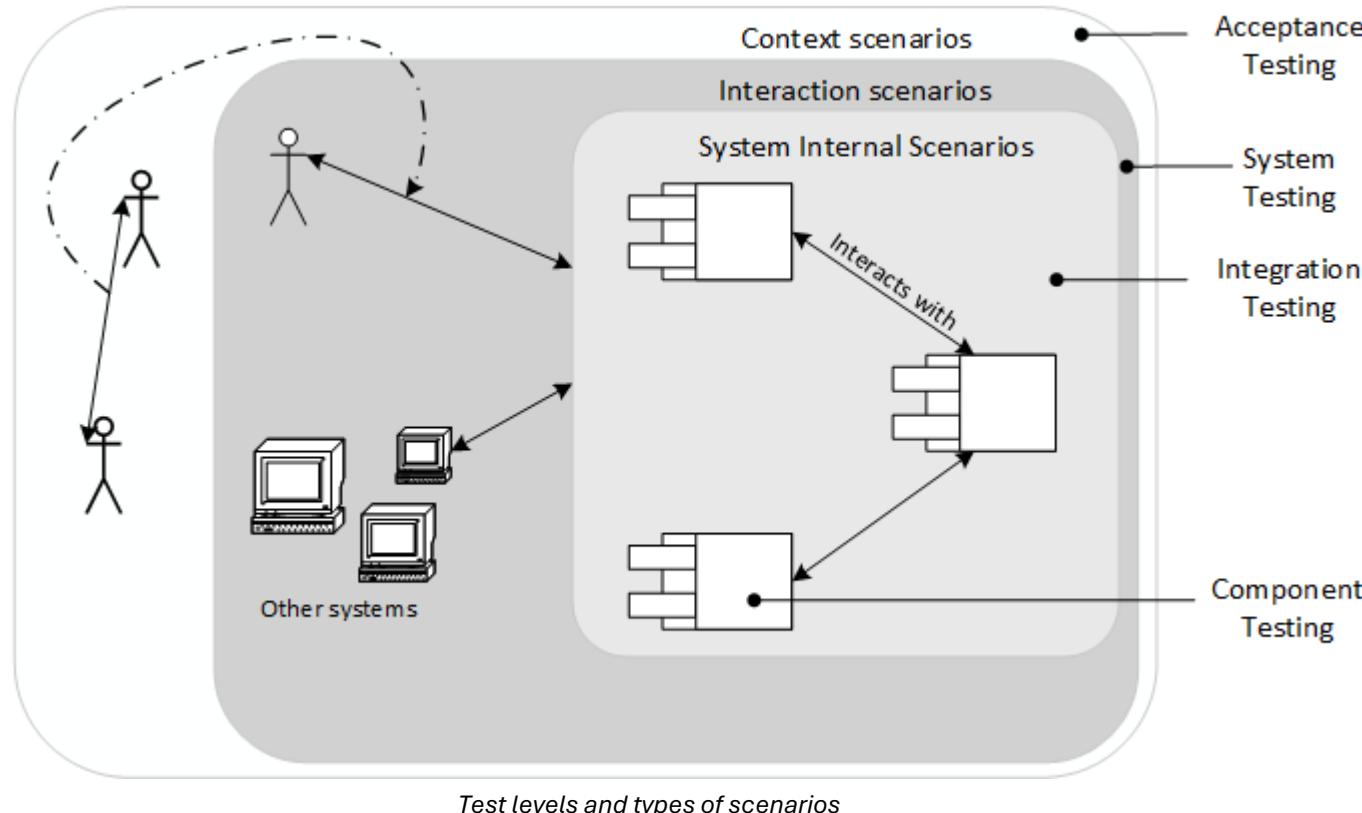
Main Concepts behind testing

Following is introductory to main concepts behind testing software intensive systems that are relevant to requirement based testing

<u>Test Case Definition</u>			
Code based Testing Structure of program code serves as test reference 'White-Box tests' since definition of test cases relies on internal structure of test objects	Specification based Testing Test object's specifications serve as test reference 'Black-Box tests' since internal structure (implementation of test object) is not considered during test case definition	Applicability to test levels Source-code based testing is suited for component testing, integration testing For identifying test cases for Testing Component – Component specification Integration – Design Documentation System – Requirement Specification Acceptance – Acceptance criteria	Coverage of test object during testing Source-code based testing is advantageous since complete source code is known during test case definition Specification based testing is advantageous since whole specification can be covered, helps to detect missing implementation of requirement

Role of scenario in Testing

Execution of test case comprises a sequence of interactions between elements in the context of the test object and test objects



A test case scenario is a type scenario that specifies types of inputs, types of expected outputs, and the interactions between types of actors which are involved in the test

- System Internal Scenarios: The test case scenario for component testing are system-internal scenario. Since the individual components of the systems are tested. The test-case scenario of integration testing are also system internal scenarios
- Interaction Scenarios: The test case scenario of system testing are interaction scenarios, since the expected interactions between the users and the system or between the system and other systems are specified in the system test-case scenarios
- Context Scenarios: During acceptance testing also, context scenario play a major role. For example: during a field test, the software-intensive system is tested by selected users under realistic conditions, i.e., the system is tested in the context of its actual use.

Creating Scenario and Test Cases

Scenario Creation Checklist

- Provide the reviewers with all requirements artefacts that are relevant to creating the scenario
- Provide a (reduced) scenario template
- Select only those sections of the scenario template that are relevant for the validation goal
- Provide checklist for detecting defects
- Provide rules for documenting scenarios
- Ensure reviewers document all detected defects
- Take care that both original requirements artefacts as well as the created scenarios are corrected
- Validate scenario created in a larger group of stakeholders by applying one of the validation techniques
- Scenario Documentation template:

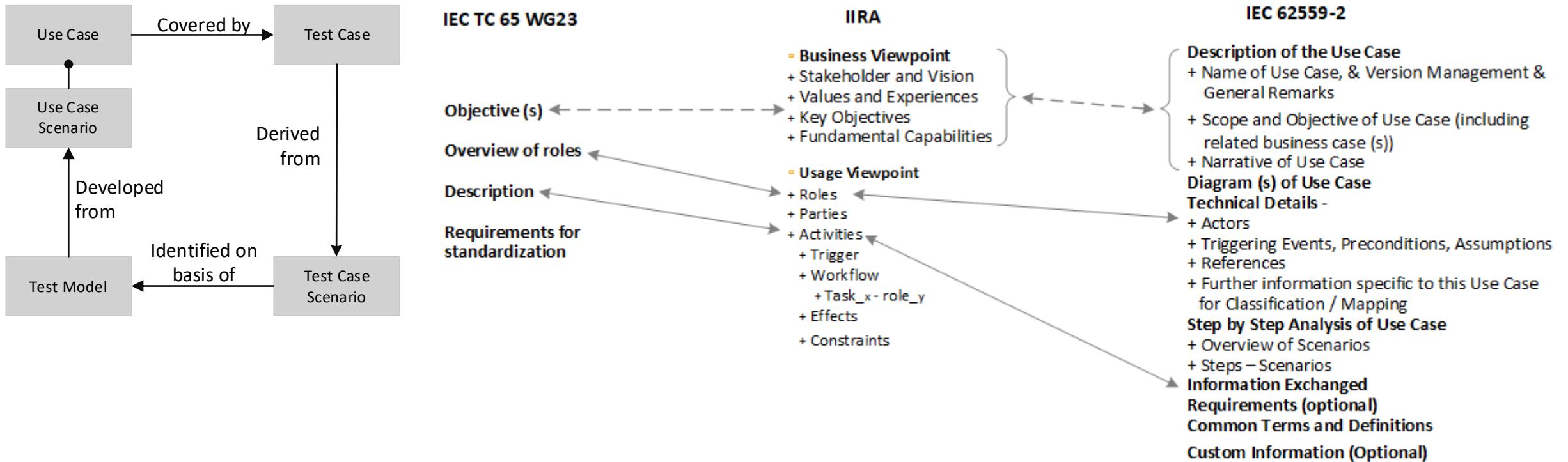


Test Case Creation Checklist

- Collect all requirements that are needed as input for creating the test cases
- Provide a template for the test cases
- Provide detailed rules for deriving test cases from requirement artefacts.
- Provide checklists to support defect detection
- Ensure that the reviewers apply the rules for creating test cases correctly
- Take care that the detected defects and issues are documented
- Ensure that creator corrects the defects in the requirements artefacts that were detected during the creation of the test cases
- Validate the created test cases to detect further requirements defects
- Test case Documentation template

[Link to Use Case Repository - Europa](#)

Relationship between Use Case and Test Cases



As a test Engineer, I would want to

- Generate Test Cases, test data and test schedules automatically
- Visualize test cases, test data and test schedulers
- Specify and reuse test environments
- Capture test execution results for further test evaluations
- Specify matching mechanisms for actual and expected responses
- Specify arbitration rules for verdict calculation
- Generate executable test scripts and test results for a dedicated target platform
- Produce test reports in a desired format
- Specify test (automation) architectures in a technology-independent manner

... so that comprehensibility and communication among stakeholders are improved, important knowledge is preserved, and the degree of automation is increased

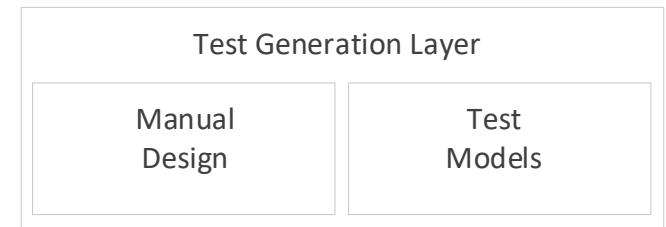
UML Testing Profile -> Domain specific standards

SysML, UML, IEEE:829, SoaML, ISO 26262, ETSI TPLan, ISTQB TAE, TestIF, ISO 29119-4, ISTQB CMBT, ETSI TDL, ISO/IEC 61508, ISO 29119-1, UTP-1, ISO 29119-5, EN 50129, ISO 29113-3, Do-178C, ETSI TTCN-3, IEEE:ATML, ETSI MBT, ISO 29119-3, EN 50129

Building Test Architecture

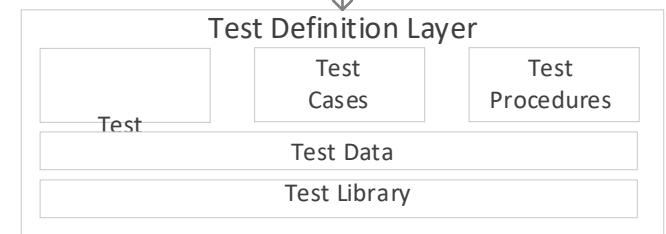
- **Test Generation Layer**

Manual / Automated Design of test cases / Test Data



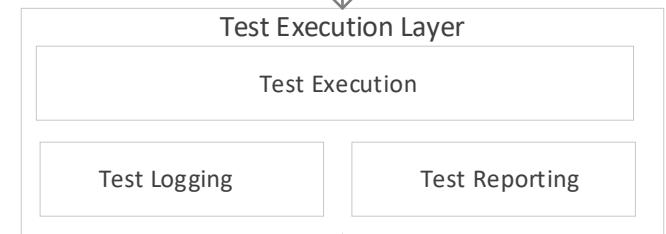
- **Test Definition Layer**

Specification of test Cases, test data, test procedures



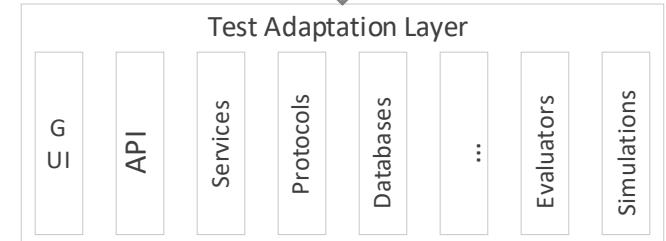
- **Test Execution Layer**

Execution of test cases, logging of test execution, test evaluation & verdict arbitration



- **Test Adaptation Layer**

Establishing communication with the system under test in order stimulate and observes it



How Use Cases Help You Define Software Requirements

Tips on writing good use case

Agreement on use case structure and processes is essential to achieving quality and consistency

Don't forget you are writing use case for end user

Sequencing the events in a flow is not always necessary to achieve clarity

Removing CRUD creates a simpler document

Selecting a single choice for an actor, and creating alternative flows for other choices can simplify the main flow

Readability can suffer if there are if statements in a flow because it usually means multiple requirements

References indicate what caused an alternative flow to start and what the system does in response

References define beginning and end of an alternative flow

Use cases are not design documents they are requirement documents

Alternate flow explain deviation from main flow

Use cases must have single main flow and alternative main flow

Establishing a use case style makes writing and reading them faster and easier

People are most import use case element

Diagrams provide a visual reference for the use case

A common mistake is to confuse requirements with design specification

Use Case describe both functionality and results

Use case transform shall statement into groups that provide observable value and context organized from a user perspective

Use cases are formal requirements that clearly define the resultant value

A use case is the story of how the business or system and the user interact

Involve Users
Directly (in-call)

Develop your use
Case iteratively

Verify your use
cases

Depict your Use
Case Visually

Use your Use Cases
to Harvest
Nonfunctional
requirements

Trace your Use
Cases

Prioritize your
use cases and Use
Case scenarios

Use Case defining Unit Tests vs User Acceptance Test

	Unit Tests	User Acceptance Test
Scope	Small isolated tests, that focus on smallest testable parts of an application, typically individual functions or methods. Often written during or after development	UAT is high level testing done after the system has been developed and integrated
Purpose	To validate that each specific function or piece of code works as needed.	To validate whether the system meets business requirements and whether it works for end user or clients
Focus	On Internal logic, verifying whether a particular a piece of code works as intended	On the overall functionality from User's POV ensuring the application fulfils the user's needs
Automation	They are typically automated and run frequently during the development process	Typically conducted by client representative or end users
Level	Low level testing, targeting specific code segment or components	High level testing, covering full processes and workflow

Benefits of Unit Tests

Early Bugs Identification -> Significant Reduction in Production Bugs -> Makes Complex code easy to understand -> Provides Documentation -> Save Development Time -> Easier to change and refactor code -> Developers Become Confident

Roy Osherove's Naming Convention for **Unit Tests**
[UnitOfWork_StateUnderTest_ExpectedBehavior]

Name of Method being tested

Input value for the Method

What Method returns for specified input

Module 4:
Requirement Based Testing



Wednesday 12th September



12:00 PM – 01:00 PM



SAHIL DATERO
Business Analyst, Delivery

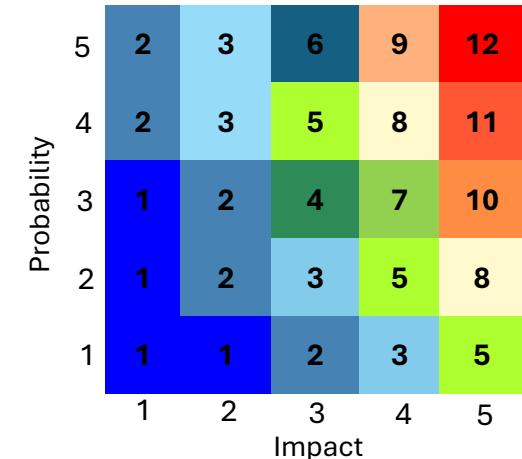
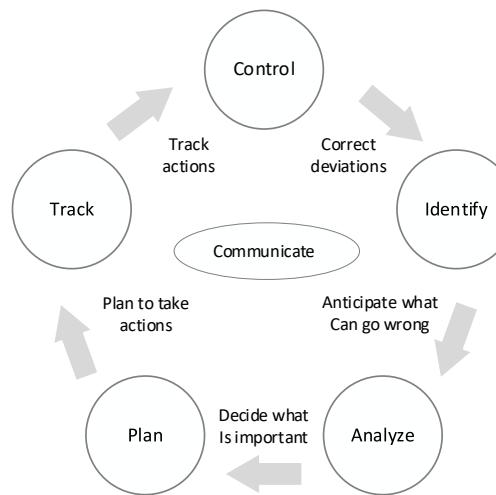
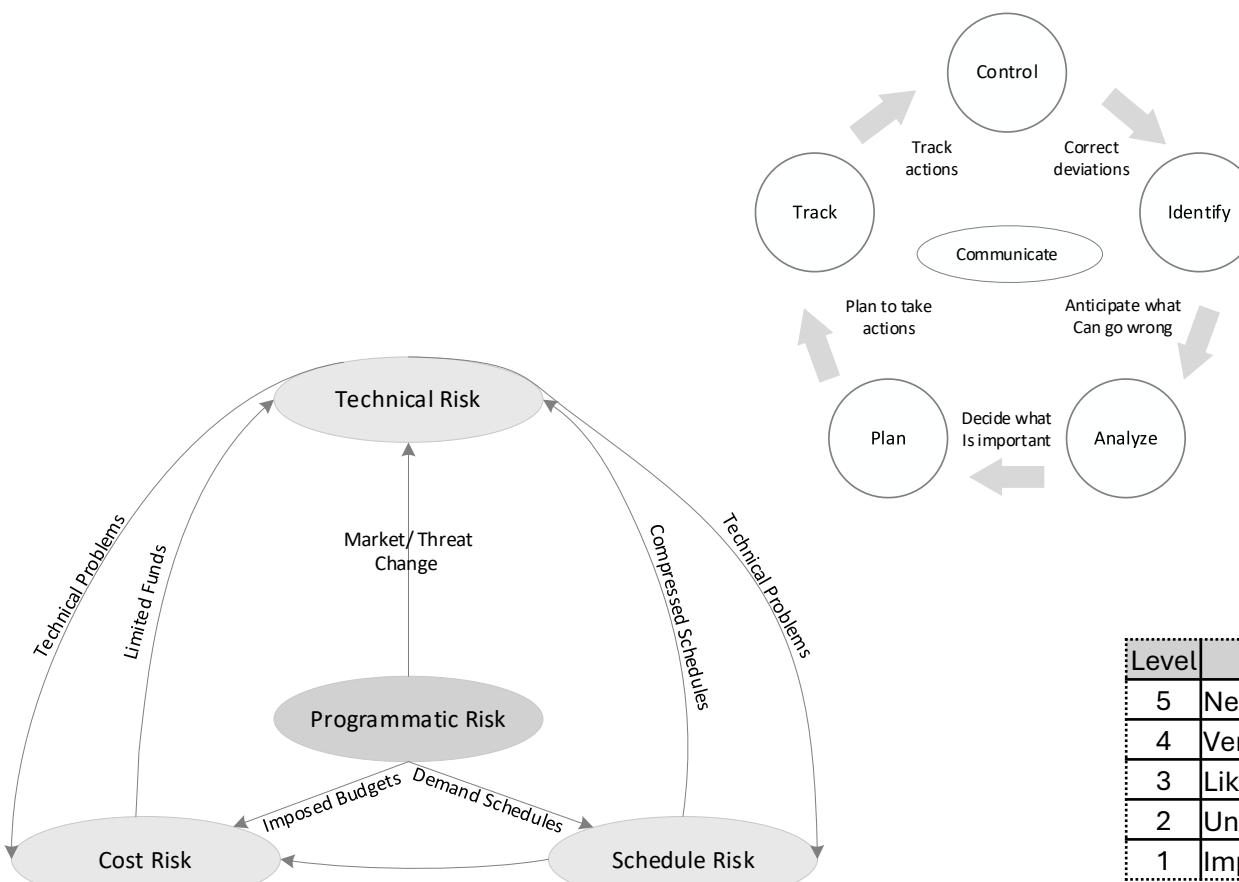
Agenda:

- Risk Management Best Practices
- Lifecycle models
- Technical Review Audits
- System Lifecycle Process
- ISO 27001, ISO 23002, ISO 20000, Best Practices

What is Risk?

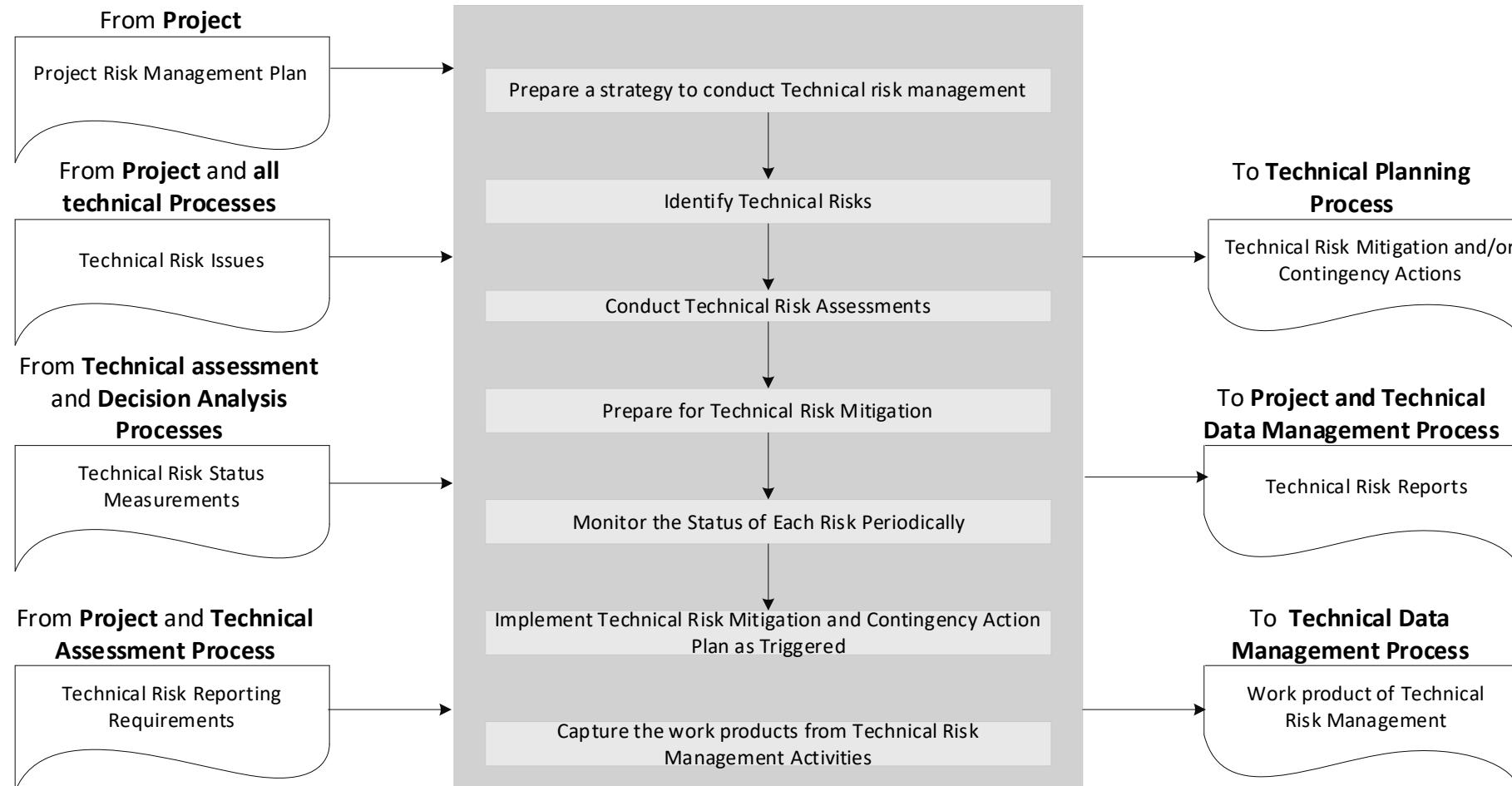
- Risk is a measure of **future uncertainties** in achieving program technical performance goals within defined cost and schedule constraints
- Risk can be associated with **all** aspects of technical effort eg threat, technology maturity, supplier capability, design maturation, performance against plan, etc, as these aspects relate within the systems structure and with interfacing products.
- Risk has three components
 1. Future **root cause**
 2. Probability or **likelihood** of that future root cause occurring
 3. **Consequences** (or effect) of that future occurrence
- Risk is defined as the combination of:
 - The **probability** that a program or project will experience an undesired event
 - The **consequences**, impact, or severity of the undesired event, were it to occur)
- The undesired event might come from **technical** or **programmatic** sources (eg cost overrun, schedule slippage, safety mishap, health problems, malicious activities, environmental impact, or failure to achieve a needed scientific or technological objective or success criteria
- Technical Risk Management is an unorganized, systematic risk-informed decision-making discipline that proactively identifies, analyses, plans, tracks, controls, communicates, documents, and manages risk to increase the likelihood of achieving project goals.

Risk Categories – «Iron» Triangle



Level	Value	Criteria
5	Near certainty	Everything points to this becomes a problem, always has
4	Very Likely	High chance of this becoming a problem
3	Likely (50/50)	There is an even chance this may turn into a problem
2	Unlikely	Risk like this may turn into a problem once in a while
1	Improbable	Not much chance this will become problem

Technical Risk Management Best Practices – Process flow diagram



What is a Life Cycle?

- **Life Cycle:** Describes how a product goes from conception to retirement
- **Life Cycle Model (LCM):** Generally a set of phases (with entry/exit milestones) that describes how a product goes from conception to retirement
- Each phase of a life Cycle is generally characterized by a set of:
 - **Activities**
 - **Milestones**
 - **Artefacts**
 - **Review**
- Examples:
 - Classic Waterfall LCM (1970)
 - Improved Waterfall LCM (1970)
 - “V” LCM (1991)
 - Spiral Development LCM (1981)
 - Iterative Development (1999)
 - Block / Incremental Development LCM

What is a Development Life Cycle/ Process?

Generally it is the **portion of the lifecycle** dealing with development

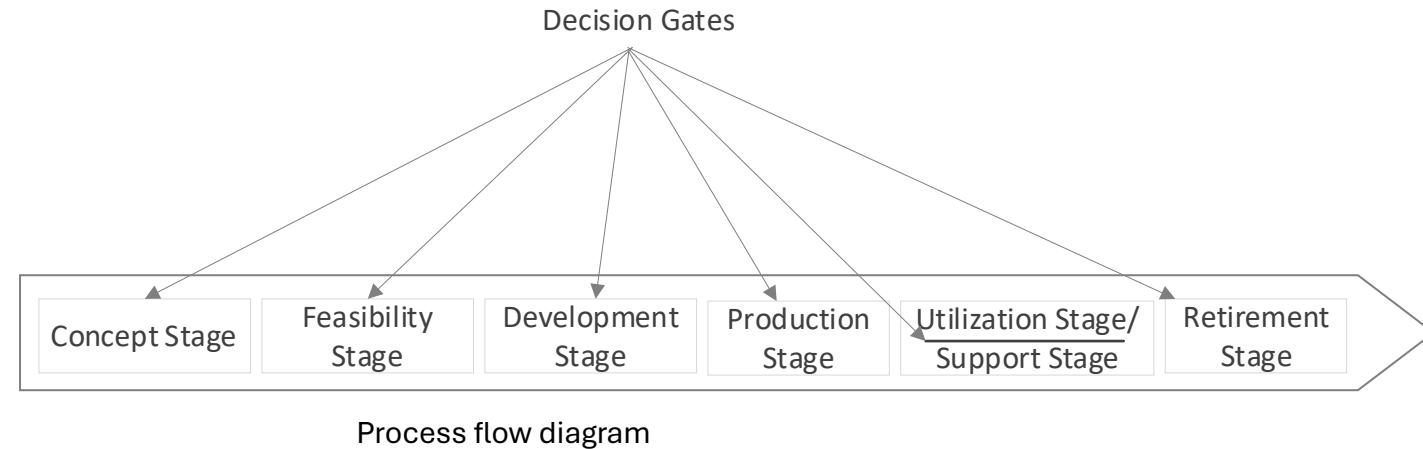
Generally a (Software) Development Life Cycle/ Process includes:

- Requirements
- Design
- Implementation/ Coding/ Unit Testing
- Integration Testing/ Verification

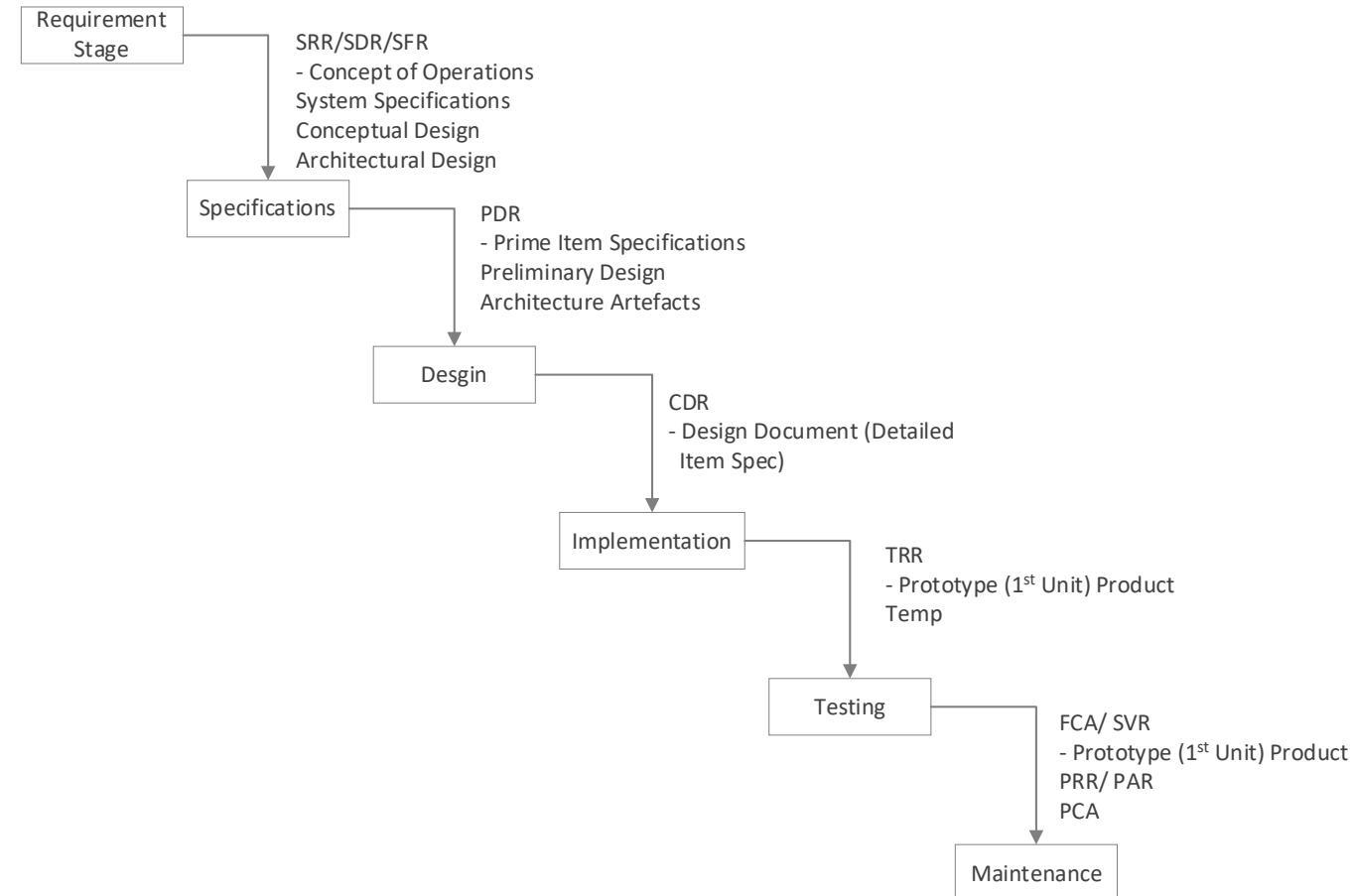
Note that it **generally excludes**:

- Manufacturing
- Deployment
- Operations
- Retirement

LifeCycle Model with Decision Gates



Classic Waterfall Model (Royce 1970)



Technical Reviews and Audits

Alternative System/ Concept Review (ASR/ ACR)
- Select Preferred System Concept
- Approve/ Kick off acquisition

System Requirement Review (SRR)
- Approve / Kickoff start of project
- PMP / SEMP / Temp
- Customer Requirements
- Top Level Functional Architecture
- Top Level Conceptual Design
- Draft Specs

System Design / Definition / Functional Review (SDR / SFR)
- Approve System Spec:
- Approve Conceptual Design
- End of Concept / Architecture phase

Preliminary Design Review (PDR)
- Approve Performance Item Spec
- Approve Preliminary Design
- End of Requirement Phase

Critical Design Review (CDR)
- Approve Final Desing (DI Specs)
- End of Design Phase

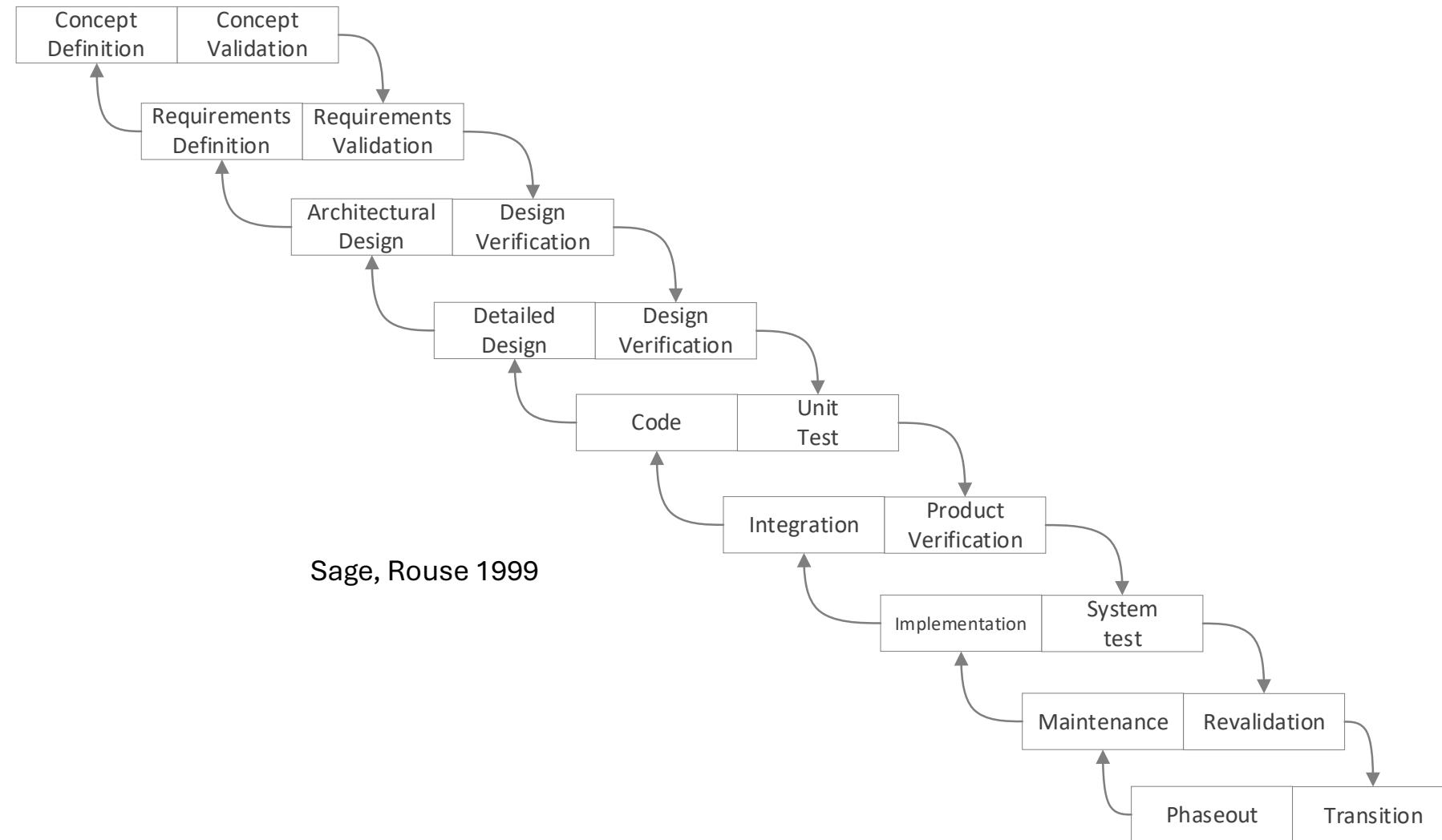
Software Specifications Review (SRR): See PDR

Functional Configuration Audit / System Verification Review (FCA / SVR)
- [1st Unit Acceptance]
- Verifies that PI Specs meets Customer requirements
- Verifies that DI Specs meets PI Spec
- End of testing

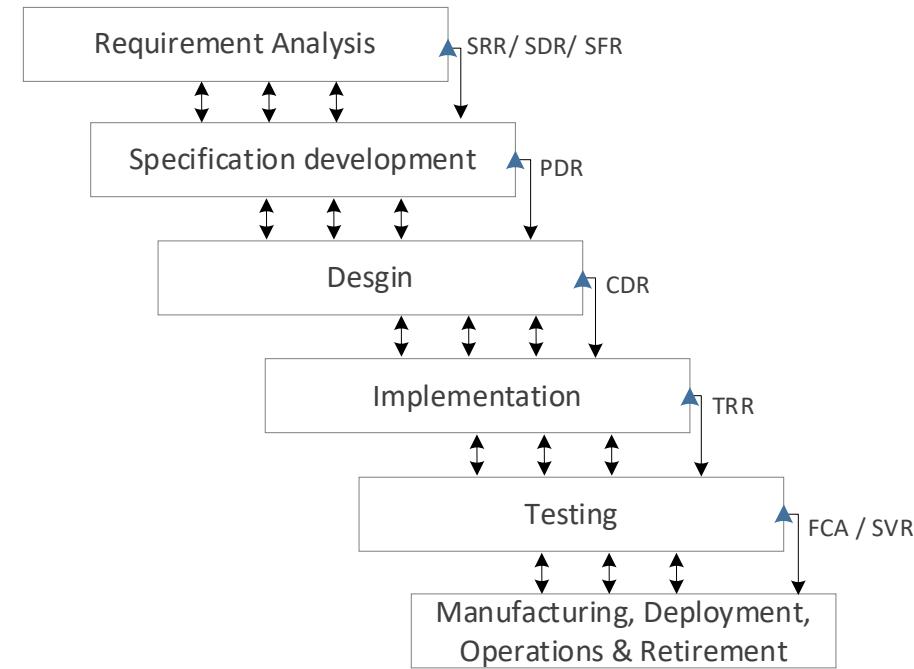
Production Readiness Approval Review (PRR/PAR)
- Approve Start of unit production

Physical Configuration Audit (PCA)
- Formalizes (corrected) product baseline or production
- Follows PRR/ PAR

Duplicate Waterfall Model (Boehm 1981)



Concurrent Waterfall

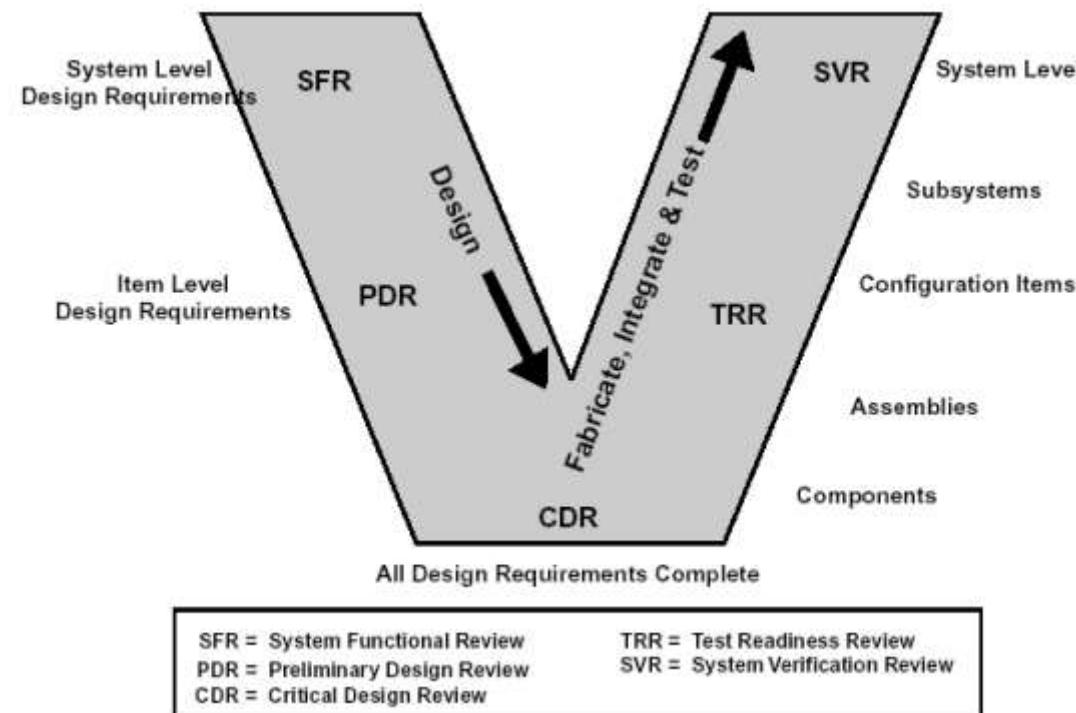


Modified Waterfall Life Cycle Models

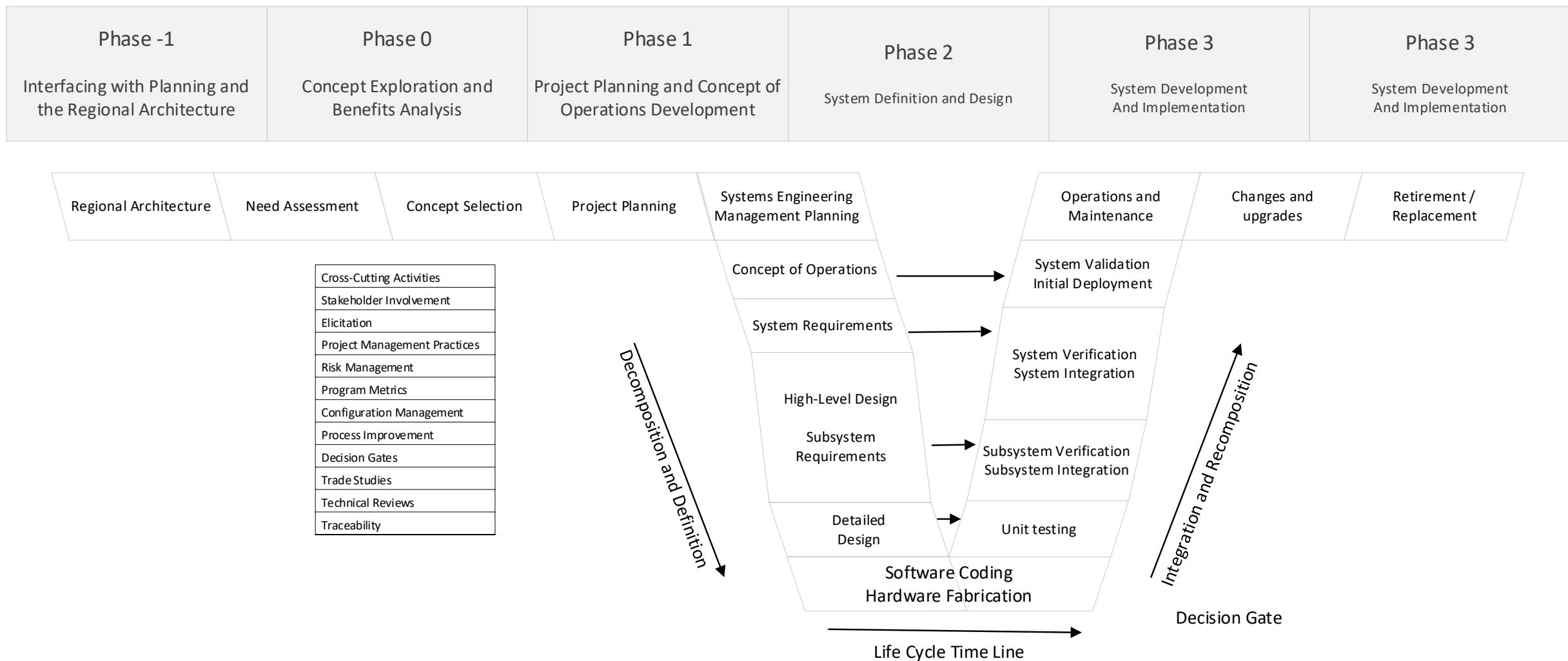
- Waterfall with Concurrent Design / Development
- Evolutionary Prototype
- Staged Delivery / Desing to Schedule
- Evolutionary Delivery

“V” Model from SEF (Quld, 1990)

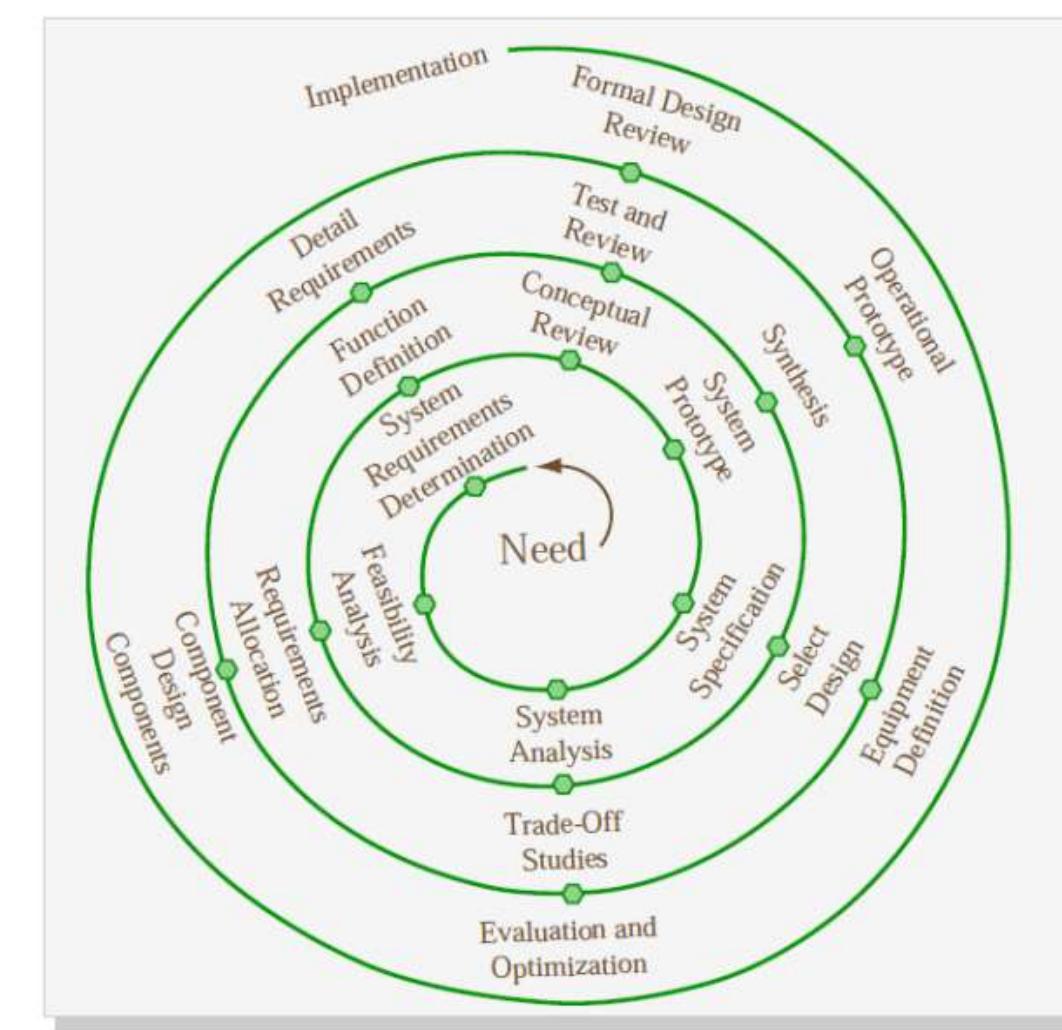
Separation Specification and Design



V Model



Spiral Model (Boehm, 1981)



Life Cycle Models and Processes

Managing life cycle models

- A small number of generic life cycle models are required for running the business of the organization / enterprise
- Life cycle Models are systems and must be life cycle managed via the life cycle management process

Process management

- Also managed according to the life cycle management process
- Tailoring applied to provide processes relevant for various types of the system
- 1) A life cycle model is defined in terms of stages and the contributions they make
- 2) Individual life cycle stages that influence the fulfilment of an agreement to supply a system product or service are described
- 3) Modified or new system life cycle processes are defined

Life Cycle Processes

This process provides life cycle policies, processes, models, and procedures that are consistent with the organization's objectives, they are defined, adapted, improved and maintained to support individual project needs within the context of the organization, and that are capable of being applied using effective, proven methods and tools.

Outcomes

- As a result of successful implementation of the life cycle model management process:
 - a) Policies and Procedures for the management
 - b) Responsibilities, accountability, and authority of the life cycle management are defined
 - c) Life cycle processes, models, and procedures for use by the organization are defined, maintained and improved
 - d) Prioritized process, models , and process improvements are implemented

Agile vs Waterfall Details

How is Scrum different from Waterfall?

Scrum is an Agile methodology that emphasizes **iterative development**, collaboration and flexibility.

In contrast, the waterfall model is a more traditional, linear approach to project management where each phase must be completed before moving on to the next.

Scrum allows for **changes** and adaptations through the process.

While waterfall is more rigid and less responsive to changing requirements

What methodology is good for my project?

The choice between Agile (such as Scrum) and Waterfall depends on the nature of your projects.

Agile is generally better suited for projects with uncertain or **changing** requirements, where flexibility and rapid iteration are important.

Waterfall may be more appropriate for projects with well-defined, **stable** requirements and a clear timeline.

Consider the specific needs and constraints of your project when deciding which methodology to use.

Module 4:
Requirement Based Testing



Saturday 14th September



05:00 PM – 06:00 PM

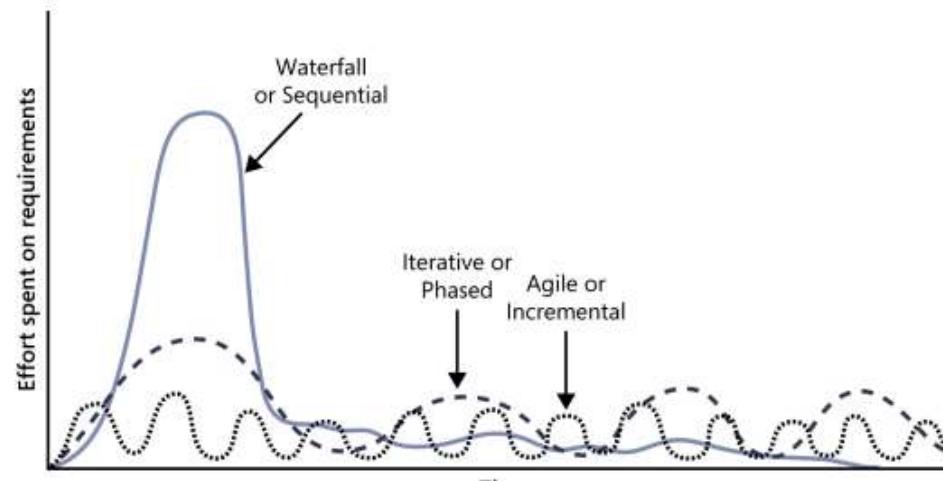
Agenda:

- Agile vs Waterfall methodology
- SCRUM Methodology
- SCRUM Ceremonies / Rituals
- SCRUM Terms and Definition
- ISO 27001, ISO 23002, ISO 20000, Best Practices



SAHIL DATERO
Business Analyst, Delivery

Agile vs Waterfall details



How is Scrum different from Waterfall?

Scrum is an Agile methodology that emphasizes **iterative development**, collaboration and flexibility.

In contrast, the waterfall model is a more traditional, linear approach to project management where each phase must be completed before moving on to the next.

Scrum allows for **changes** and adaptations through the process.

While waterfall is more rigid and less responsive to changing requirements

What methodology is good for my project?

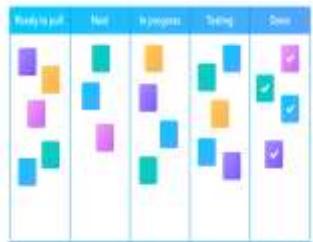
The choice between Agile (such as Scrum) and Waterfall depends on the nature of your projects.

Agile is generally better suited for projects with uncertain or **changing** requirements, where flexibility and rapid iteration are important.

Waterfall may be more appropriate for projects with well-defined, **stable** requirements and a clear timeline.

Consider the specific needs and constraints of your project when deciding which methodology to use.

What are the other Agile methodologies? Apart from Scrum



Kanban methodology

Kanban is a visual workflow management system that emphasizes just-in-time delivery, **limiting** work in progress, and continuous improvement



Lean Development

Lean Development focuses on maximizing **customer value** while minimizing waste. It emphasizes continuous improvement, flexibility, and **eliminating** non-existent activities.



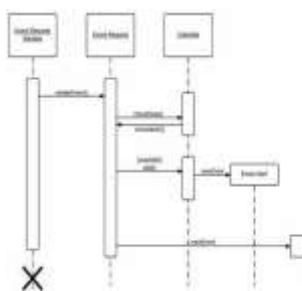
Feature Driven Development

Feature Driven Development (FDD) is a client-centric, architecture centric, and iterative software development process, it focuses on developing **well-defined** features based on client requirements



Extreme Programming

Extreme Programming (XP) is a software development methodology that emphasizes customer satisfaction, rapid feedback, and continuous improvement through **short** development cycles.

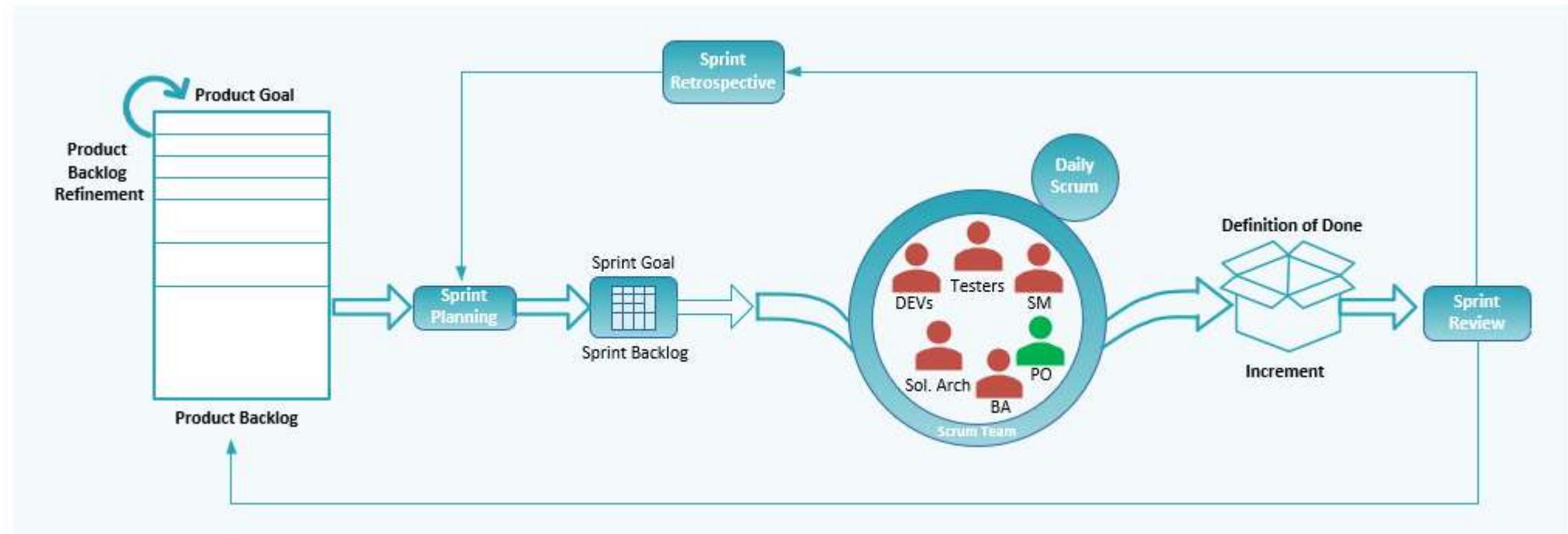


Behavior Driven Development

Behavior Driven Development (BDD) is a software development process that focuses on defining and validating the expected **behavior** of a system through collaboration between cross-functional teams.

How does Scrum Work?

Scrum is an Agile framework for managing work with an emphasis on software development. It is designed to deliver value to the customer **quickly** and effectively. The Scrum process involves a series of events, roles and artefacts that work to help teams structure and manage their work.



SCRUM Ceremonies or Rituals

Ceremonies	Responsibilities of BA
Sprint planning (duration)	<ul style="list-style-type: none"> - Clarifying Requirements: Ensure that user stories are well defined, understood, and prioritized for the sprints - Facilitating Communication: Act as a bridge between stakeholders and the development team to ensure shared understanding of goals - Story Sizing: Collaborate with the team to estimate efforts and complexity of user stories - Prioritizing Deliverables: Work with the Product Owner to confirm that high-value items are prioritized and included in the sprint backlog
Daily scrum	<ul style="list-style-type: none"> - Tracking Progress: Monitor how requirements are progressing and address any blockers related to the business objectives - Facilitating Clarification: Answer any questions the team may have regarding requirements or user stories during the sprint - Ensuring Focus: Keep the team aligned with the sprint goals by reiterating priorities if needed. - Reporting to the stakeholders: Communicate daily updates to stakeholders ensuring visibility of the team's progress
Sprint Review	<ul style="list-style-type: none"> - Validating Deliverables: Ensure that the delivered features meet the acceptance criteria and stakeholder expectations. - Gathering Feedback: Facilitate discussion between the team and stakeholders to gather feedback of completed work - Demonstrating Value: Highlight the business value and delivered during the sprint through effective presentation of completed work - Aligning on Next Steps: Collaborate with the product owners and stakeholders to identify potential improvements or adjustments for future sprints
Sprint Retrospective	<ul style="list-style-type: none"> - Identifying Process Improvements: Help the team analyse what went well and what can be improved from the business analysis perspective - Facilitating Collaboration: Encourage open discussions that foster team collaboration and process optimization - Capturing Lessons learned: Document insights related to requirement gathering communication, or any analysis challenges faced. - Aligning on Action Items: Ensure that actionable improvements for business analysis practices are identified and agreed upon
Backlog Refinement	<ul style="list-style-type: none"> - Clarifying User Stories: Ensure that the User stories are detailed, clear, and aligned with business goals. - Prioritizing Backlog Items: Work closely with the Product Owner to prioritize items based on business value and feasibility. - Breaking down Epics: Decompose larger epics or features into smaller, manageable stories for the team. - Defining Acceptance Criteria: Ensure that acceptance criteria are clearly defined and understood by both the development team and stakeholders

Roles in Scrum Team



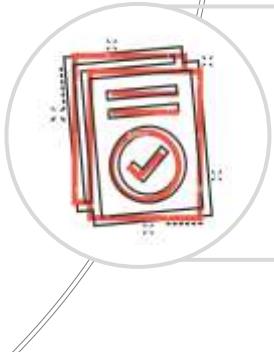
Scrum Master: Responsible for making the process **run** smoothly, removing obstacles, and organizing meetings

- 1) Sprint Burn Down Chart: A visual representation of progress of the team, showing **remaining work** in the sprint backlog over time
- 2) Impediment Log: A document tracking any **obstacles** or blockers encountered and how they are being solved
- 3) Retrospective Action Items: Documentation of Key insights
- 4) Team metrics/ Velocity Reports: Team's performance and productivity **measures**



Product Owner: Responsible for maximizing product value and maintenance of Product backlog

- 1) Product Backlog: A prioritized list of all features, enhancements and bug fixes
- 2) Definition of Done: A shared **understanding** within team about what constitutes to 'Done'
- 3) Technical Documentation: Any necessary **documentation** related to the design, architecture, or technical decisions during sprint
- 4) Release Plan: A timeline or plan specifying when product increments or releases are expected



Development Team: Responsible for delivering the product increments and related technical artefacts

- 1) Sprint Backlog: A list of tasks, user stories, and work items the team has **committed** to completing during the sprint.
- 2) Increments (Working software): A tangible, shippable product meeting the 'Definition of Done'
- 3) Definition of Done: A shared understanding within the team that defines when a user story or feature is considered as **complete** for release
- 4) Technical Documentation: Same as above

SCRUM Terms and Definition

Scrum Cycle

- Also Known as Sprint, is a fixed period of time (typically 2-4 weeks) during which a team works to **complete** a set of pre- defined tasks or user stories

Story

- A story is a small, well-defined piece of work that can be completed within a **single Sprint**. Stories are written from the perspective of the end-user and describe the desired functionality

Epic

- An Epic is a **large user story** that can be broken down into smaller stories. It represents a significant feature or functionality that needs to be delivered

Story Points

- A story point is a **unit of measurement** used to estimate the effort required to complete a story. It is a relative measure, not an absolute one.

Story Point to hour ratio

- There is no fixed ratio between story points and hours, as it depends on the team's velocity and the complexity of the work. Generally, one story point is equivalent to 'X' hours of work, where 'X' is 6.5

SCRUM Terms and Definition, conti..

Tasks and Subtasks

- A task is a specific piece of work that needs to be completed as part of a larger project or story. Subtasks are smaller, more granular pieces of work that make up a task. Tasks can be estimated in terms of hours, but the Scrum methodology uses story points instead.

Story Points vs Hours

- Story points are a relative measure of the effort required to complete a task or story, whereas hours are an absolute measure of time. The Scrum team estimates tasks in story points to avoid getting bogged down in precise time estimates, which can be difficult to predict accurately, especially for complex or uncertain work.

Program Increment (PI)

- A Program Increment (PI) is a longer-term planning period in the Scaled Agile Framework (SAFe) that encompasses multiple Scrum cycles or sprints. During a PI, the Scrum team works towards a set of larger, strategic objectives that span multiple sprints.

Scrum Cycles in a PI

- The number of Scrum cycles or sprints that occur within a Program Increment can vary, but is typically between 4 and 6. This allows the team to work towards the PI's objectives through a series of shorter, iterative cycles that build upon each other.

SCRUM Terms and Definition

DOD

- The Definition of Done is a clear and agreed-upon set of criteria that must be met for an item to be considered complete. This definition is crucial for ensuring that the Increment, at the end of each Sprint, is potentially shippable.

Spike

- "SPIKE" refers to an activity aimed at exploring or investigating a particular aspect of a product or project. SPIKES are used when there's uncertainty or a lack of clarity about how to implement a certain feature or address a specific technical challenge. They are often conducted to gather information, perform research, or prototype potential solutions.

DOR

- The Definition of Ready is a set of criteria for an item which must be satisfied before it can be pulled into a Sprint for implementation. It ensures that the team has a clear understanding of the work and that all necessary prerequisites are fulfilled before committing to it.

**Ad-hoc
Tasks**

- A task that is introduced into a scrum cycle in the middle of a scrum cycle (not at the beginning)

Spillover

- In Scrum, spillover occurs when unfinished sprint backlog items are moved back to the product backlog, instead of being completed by the end of the sprint.

Ideal size of Scrum team... 5-9 team members,
Ideal duration of a sprint cycle ... 2 weeks,
Ideal duration of a Product Increment 6 weeks

SCRUM Terms and Definition

EPICS

- Do not have estimates associated with it

Stories

- Should always be **estimated** in terms of story points

Tasks and Sub-tasks

- which come user the stories should always be estimates in terms of hours.

Velocity

- Amount of story points your team can deliver per sprint

Capacity

- How many **hours** can an **individual** commit to, in a sprint (considering holidays, sick leaves etc..)

Individual Velocity

- Number of "DONE" **stories** an **individual** can deliver is individual velocity.

Team Velocity

- Number of "DONE" **stories** a **team** can deliver, collectively, is team capacity.

Individual Capacity vs Team Capacity

- How many **hours** can the team **commit** to, in total, including holidays and sick leaves.

Product Backlog vs Sprint Backlog

Understanding the differences between the product backlog and sprint backlog is crucial effective Agile Planning and Execution



Product Backlog

The ordered list of all features, enhancements, and bug fixes that need to be developed for the product.



Sprint Backlog

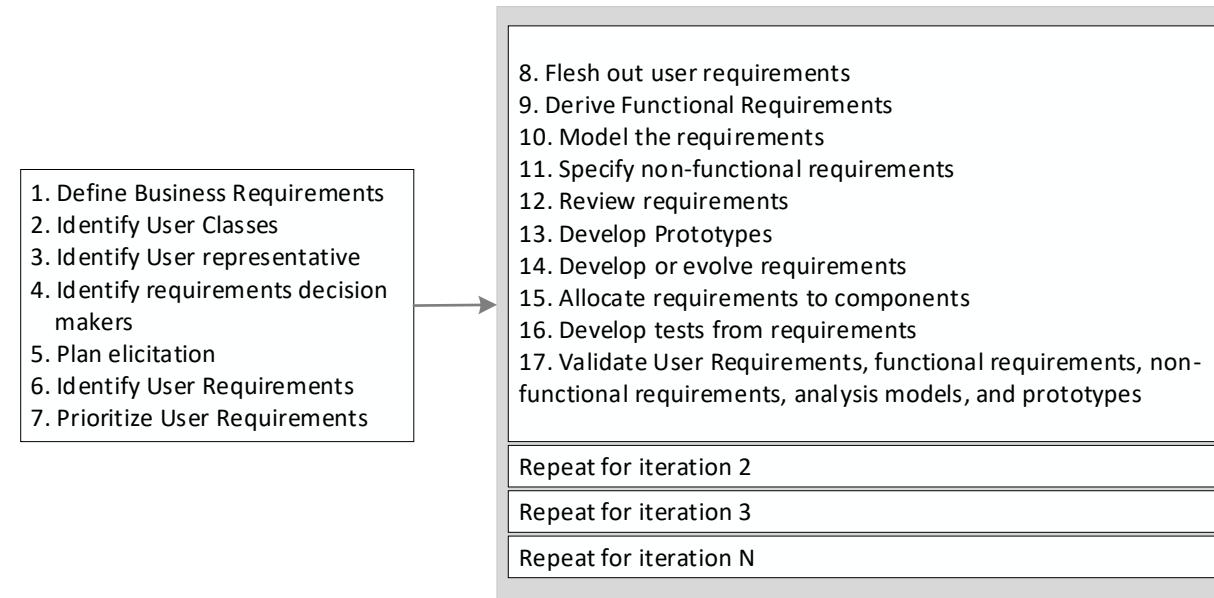
The subset of Product Backlog items selected for a specific sprint, with a detailed plan for delivering them.



What's the connection
connection between the
two?

The Sprint Backlog is derived from the Product Backlog, focusing on the high-priority items for the current sprint.

Requirement Development Process



Minutes of meet structure (for every company/project it will differ).

- 1.0 Date
- 2.0 Attendees
- 2.1 Who could not attend
- 3.0 Next Steps
- 3.1 Sr. number
- 3.2 Who (assigned)
- 3.3 (What (Action items in Detail))
- 3.4 When (Due Time)
- 3.5 Comments/ Updates
- 3.6 Status
- 4.0 Summary of discussion

Structuring BRD, FRD, CR, User manuals

Module 5:
Requirement Elicitation



Wednesday, Thursday 18th, 19th September



05:00 PM – 06:00 PM

Agenda:

- Responsibilities of BA In Elicitation
- Different Techniques of Requirement Elicitation

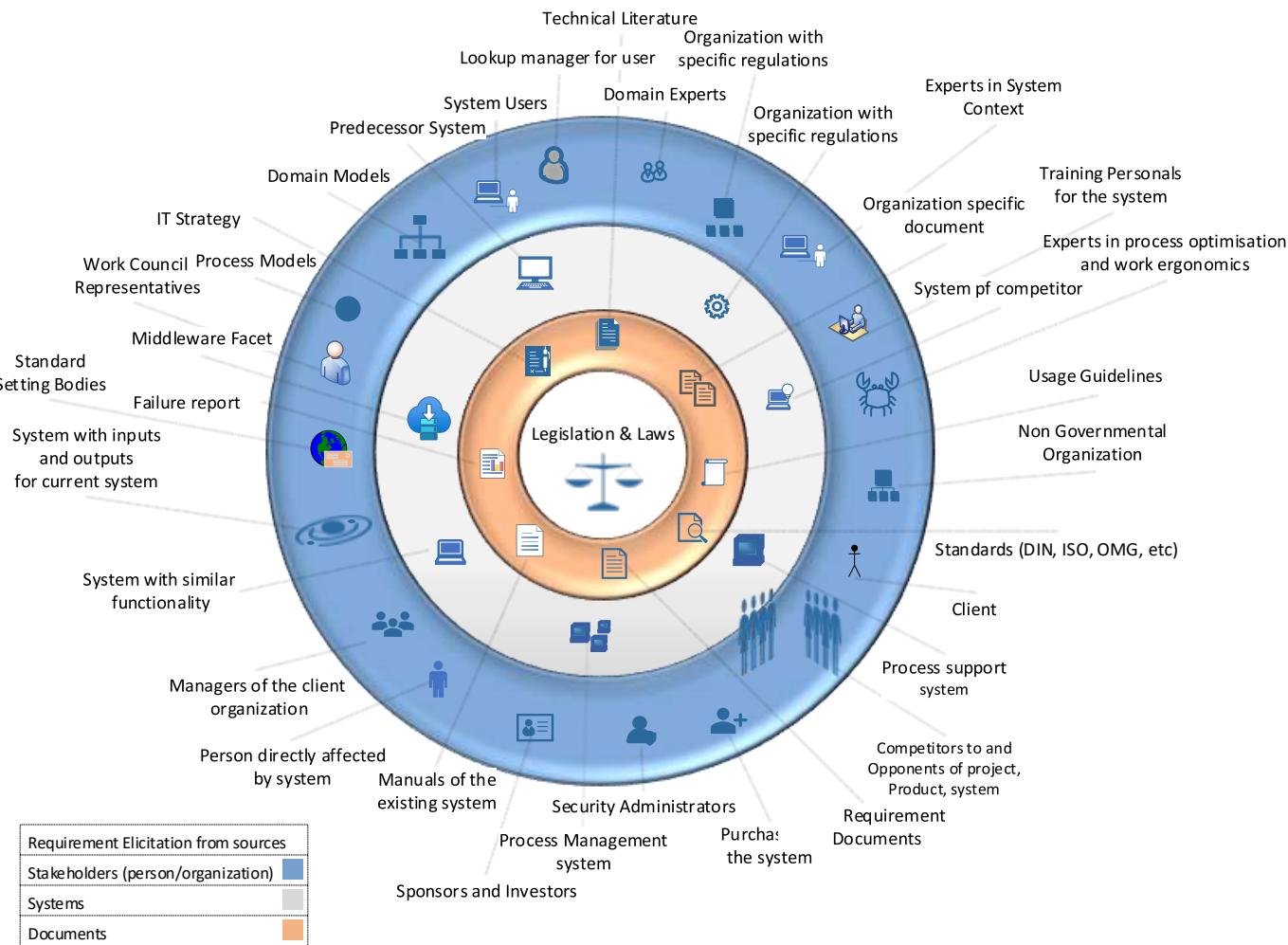


SAHIL DATERO
Business Analyst, Delivery

Elicitation helps to turn stakeholder's needs - interests into functional, operational and performance requirements

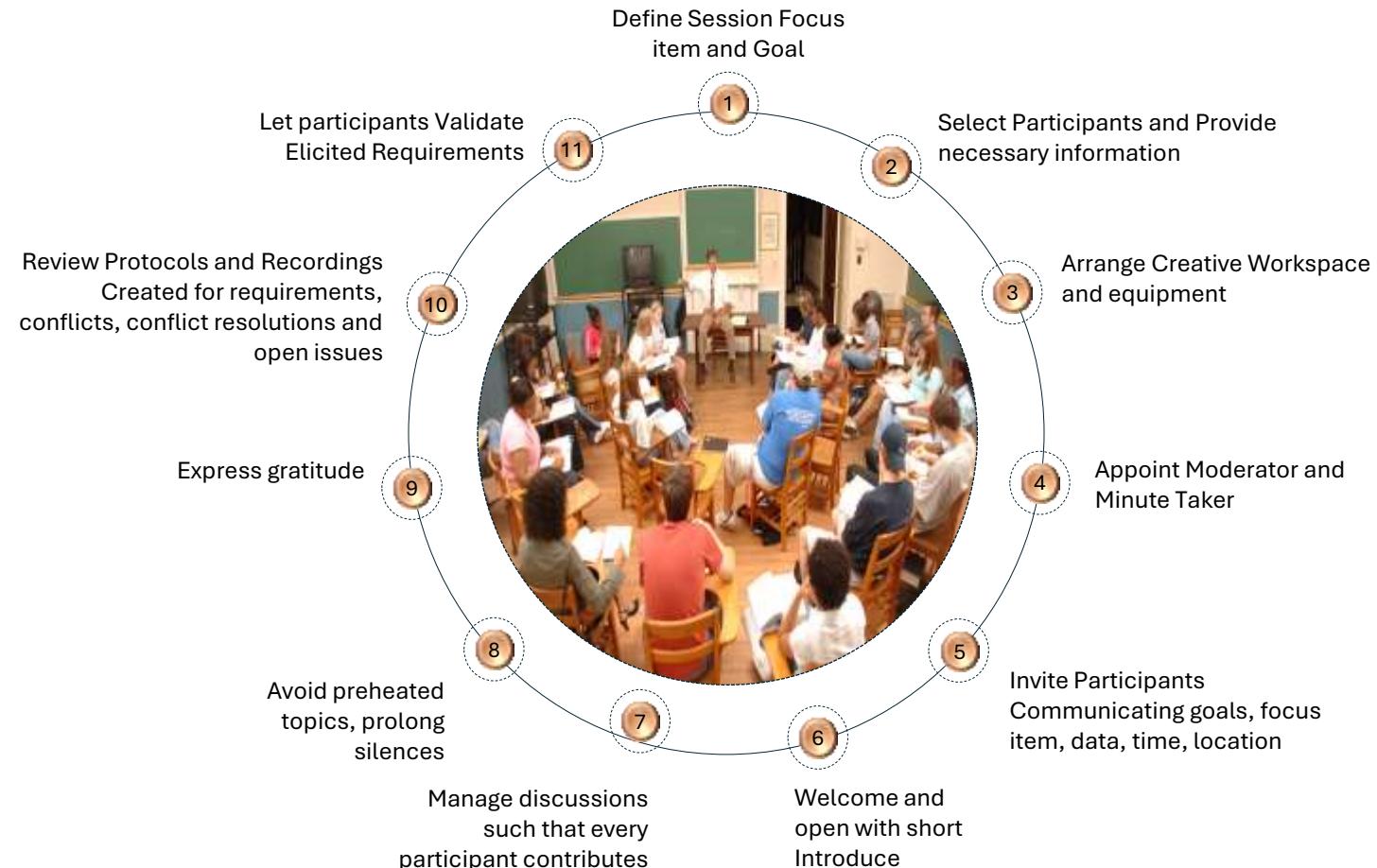
	Relevant Sources	Stakeholder Collaboration	Stakeholder's needs
Preparation (Set up) Phase	<ul style="list-style-type: none"> • Use pragmatic and systematic approach to Identify sources • Select relevant elicitation technique 	<ul style="list-style-type: none"> • Identify all stakeholders, based on How much new system affects them • Reserve time and budget for conflicts • Consider process pattern to plan activities 	<ul style="list-style-type: none"> • Provide elicitation objective, desired result quality, selected sources, • Get an overview of business case, project situation • Plan for system context analysis
Execution Phase	<ul style="list-style-type: none"> • Combine elicitation activities that address same requirement source • Define Elicitation activity goals and agenda, scope of change, output integration, specifics of usage facet 	<ul style="list-style-type: none"> • Elicitation as Time Boxed <u>activity</u> • Retrospect plan after each activity • Schedule considering long & short term activities • Parallelize independent activities 	<ul style="list-style-type: none"> • Search for conflicts and react to them according to agreed strategy • Document existing information, analysis approach, outcomes for stakeholder engagement
Confirm Results	<ul style="list-style-type: none"> • Compare elicitation results against source information • Compare elicitation results against other elicitation results 	<ul style="list-style-type: none"> • Using requirement modelling tools provide insights of outcome of newly elicited requirement to map with stakeholder's desired outcome, to assess risk 	<ul style="list-style-type: none"> • Using Process modelling, data modelling, interface analysis confirm relevance of elicited requirements
Achieve Consensus	<ul style="list-style-type: none"> • Establish communication to form common base with stakeholders on information and elicited requirement 	<ul style="list-style-type: none"> • Identify encountered indicators in communication and documentation 	<ul style="list-style-type: none"> • State your position clearly in some cases revealing unexpected problems or existing conflicts
Manage Collaboration	<ul style="list-style-type: none"> • Keep handy for document as source, name, location, relevance, etc 	<ul style="list-style-type: none"> • Gain agreement on commitments, monitor stakeholder engagement 	<ul style="list-style-type: none"> • Facilitate Stakeholder Engagement Approach and Measures for BA Performance Assessment

Effective Requirement Elicitation: Engaging Stakeholders, Analyzing Documentation, and Leveraging System Insights



- Elicit and develop goals with various stakeholders
 - Define scenarios for the identified goals by documenting concrete examples of goal satisfaction. Consider also scenarios in which the goals are not satisfied
 - Analyse the scenarios and identify possible new goals does the scenarios fulfil? Are these goals already known? Do these goals complement, contradict, or refine an already existing goal?
 - Define scenarios for the goals and thereby establish an iterative goal-scenario development cycle
 - Derive solution-oriented requirements from the defined goals and scenarios
 - Document solution-oriented requirements from the defined goals and scenarios: Which properties must the system have in order to satisfy in order to satisfy goals as well as scenarios?

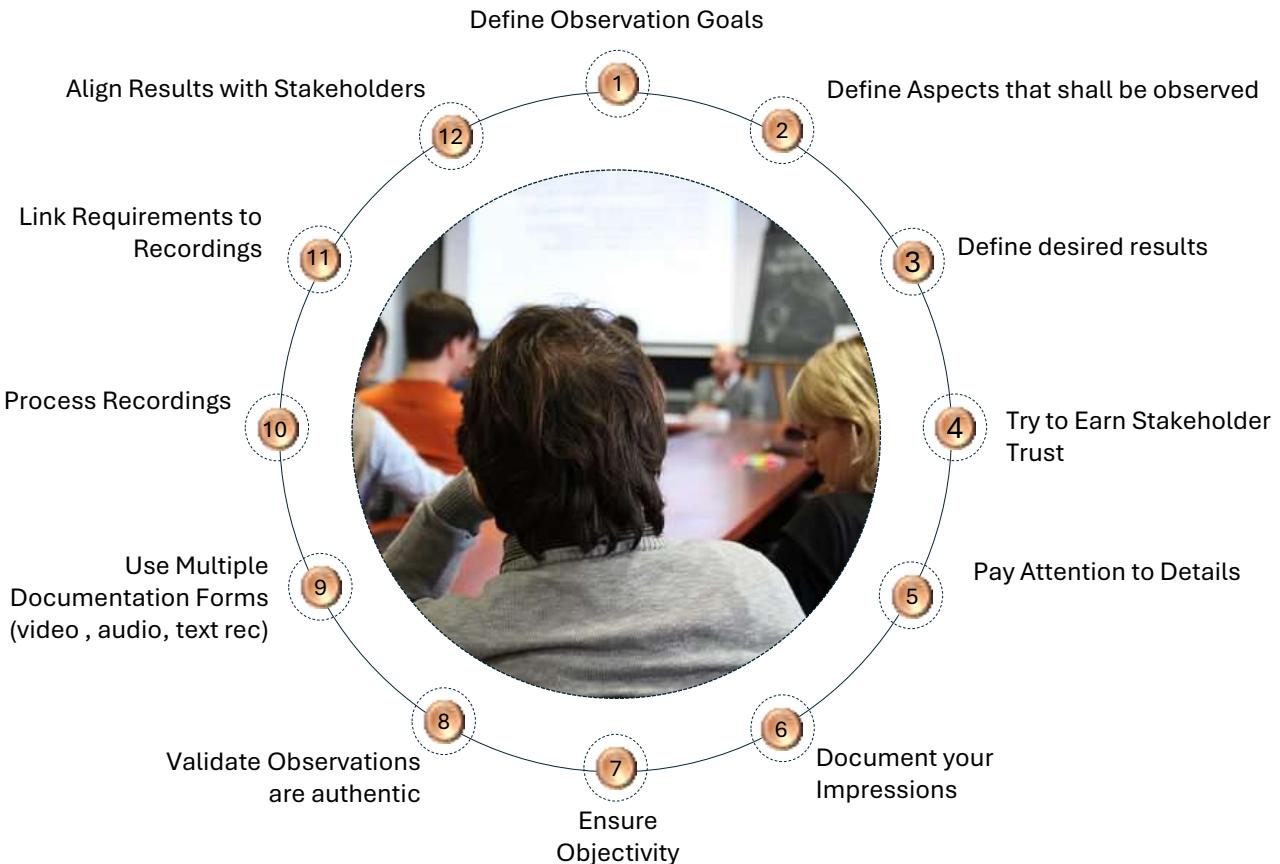
Focus Groups



Critical Success Factors:

- Participant's background and experience
- Participant presentation and motivation
- Experienced moderator to guide the discussion
- Groupthink effect
- Comfortable and creative environment

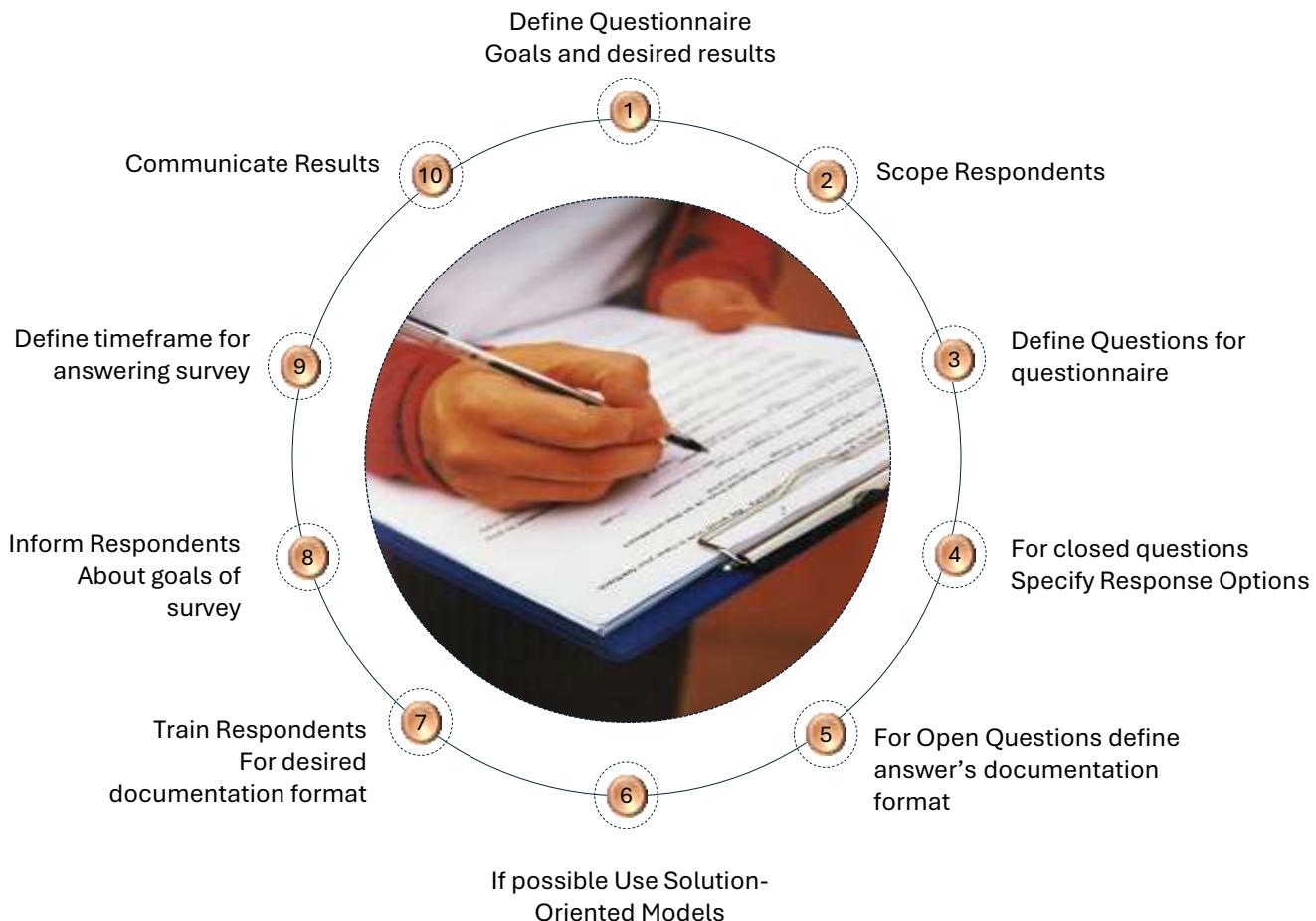
Observation



Critical Success Factors:

- Willingness of stakeholders to cooperate
- Processing of results
- Objectivity of the observer
- Observability

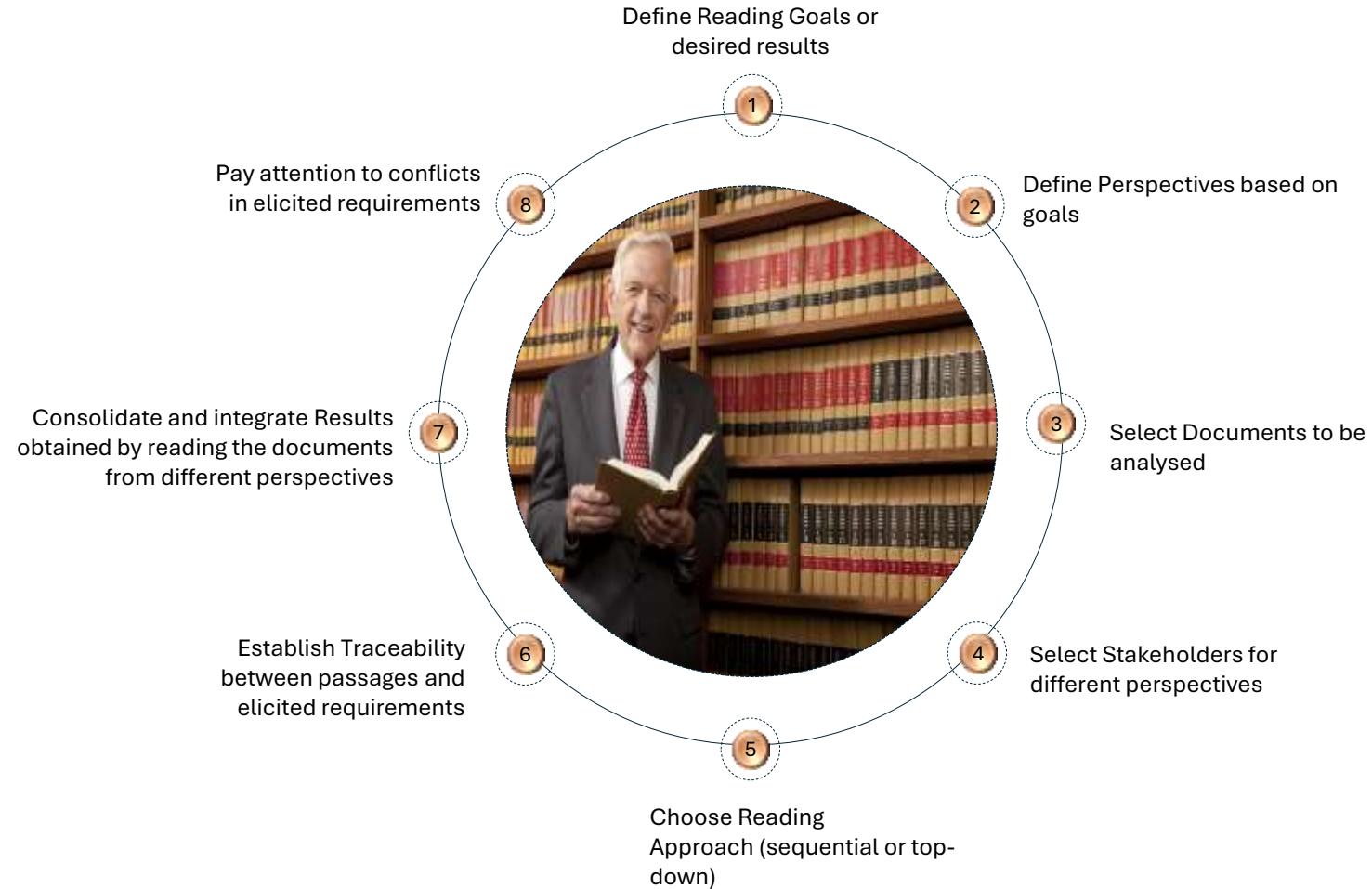
Questionnaires



Critical Success Factors:

- Comprehensible questions
- Clearly defined answers
- Training of stakeholders (If required)
- Selection of the right documentation format

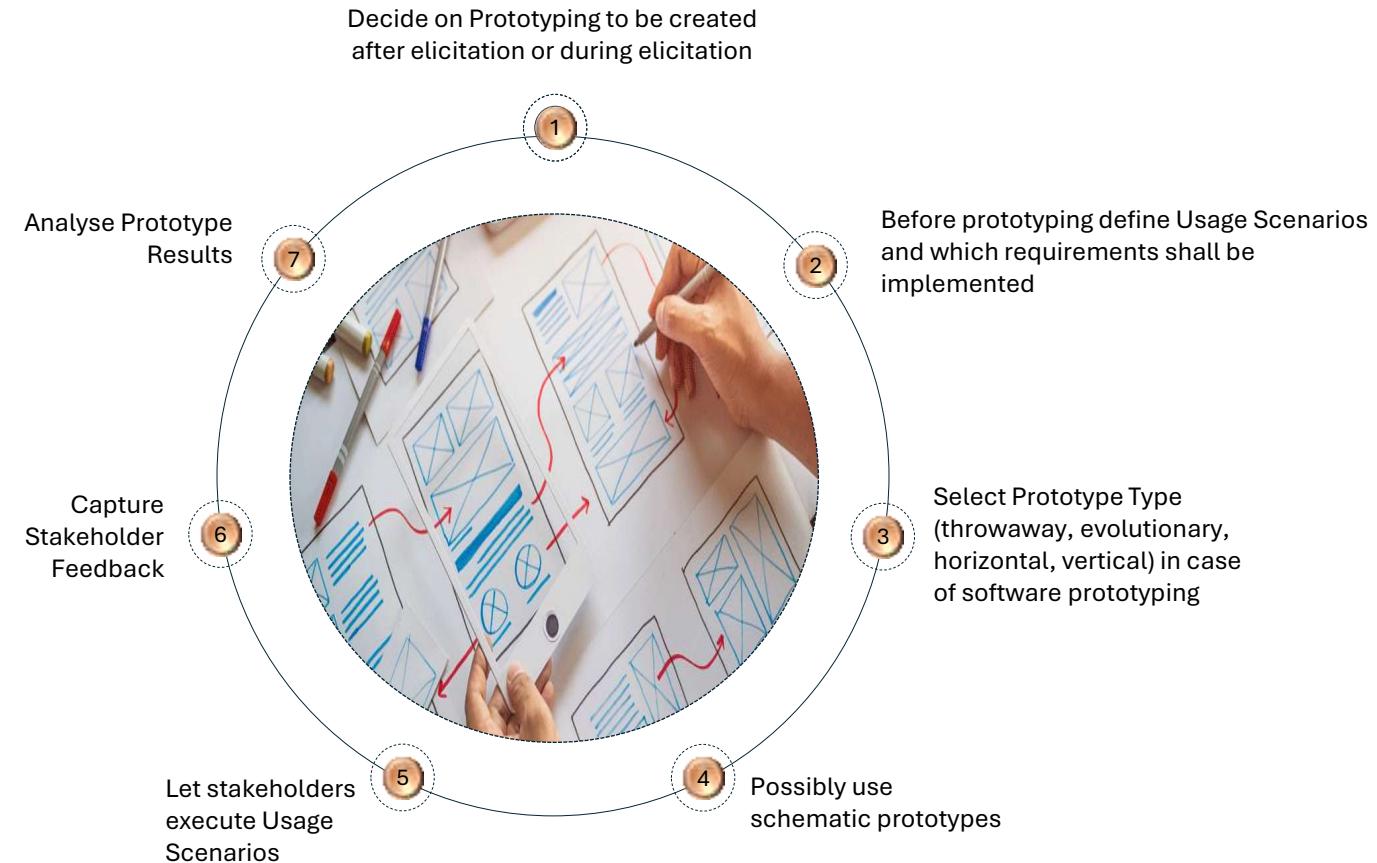
Perspective-based reading



Critical Success Factors:

- Selection of perspective and documents
- Good structure
- Clearly Defined Perspectives

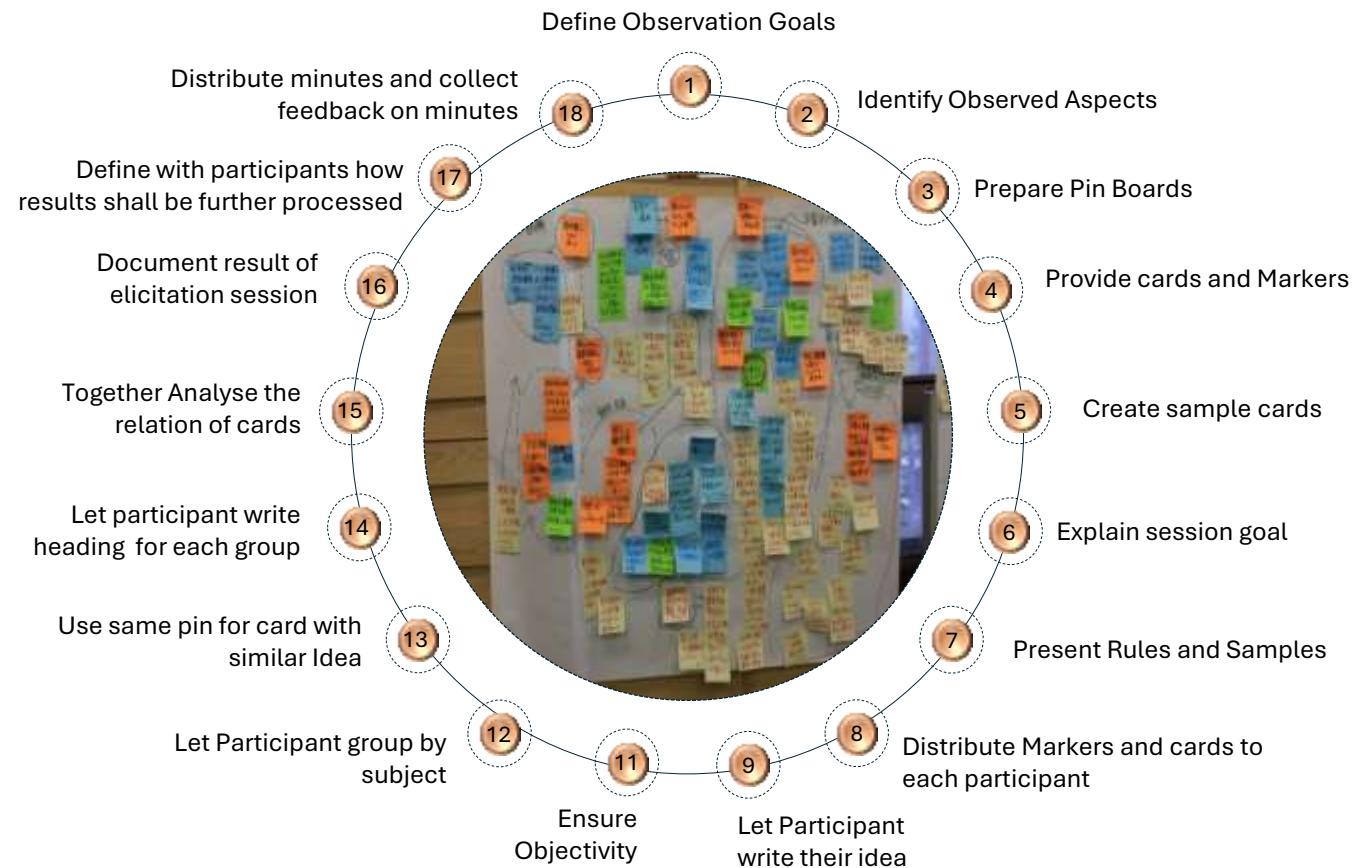
Prototyping



Critical Success Factors:

- Selection of requirements
- Limitations of the prototype

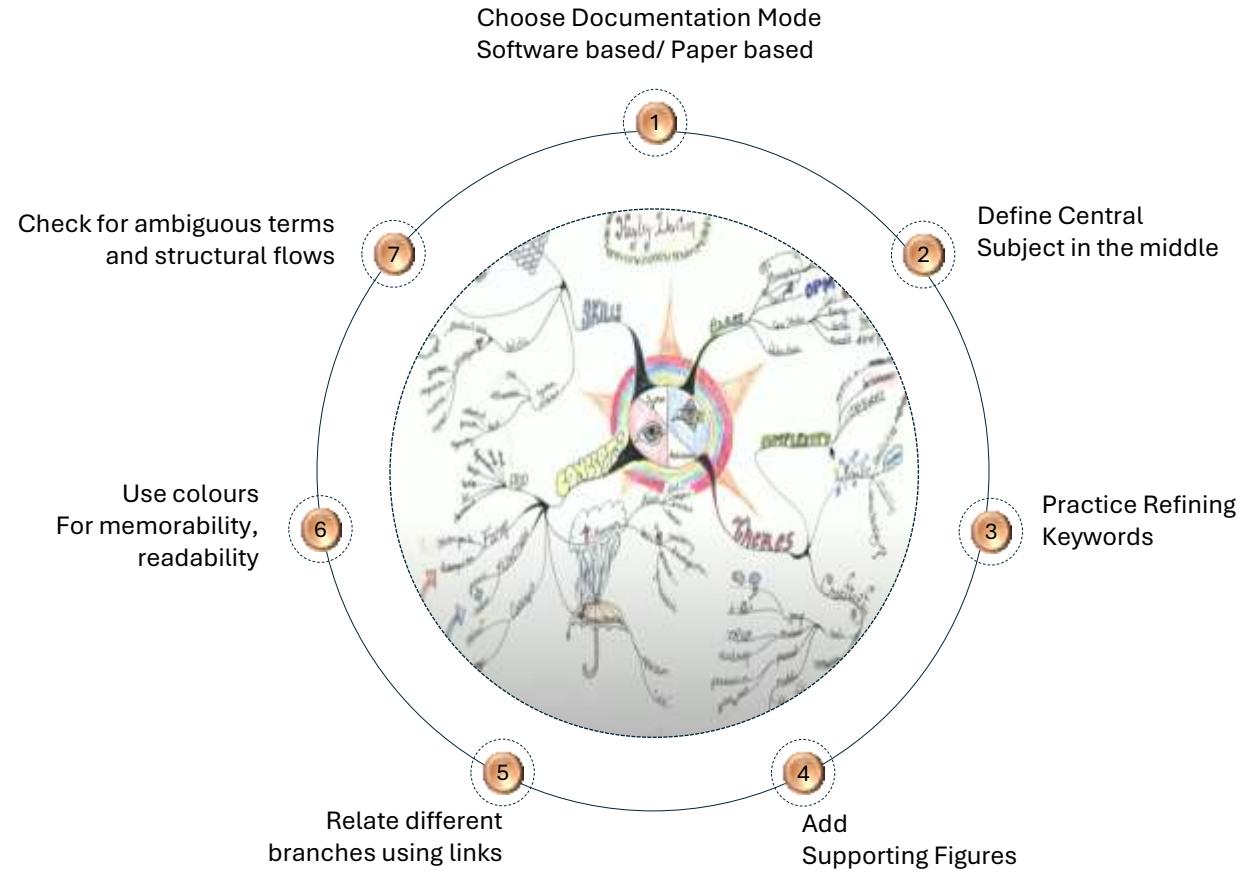
KJ Method



Critical Success Factors:

- Clearly define goal
- Size of the groups
- Reducing effort in large groups

Mind Mapping



Critical Success Factors:

- Good structure and visualization

Brainstorming

- Define the subject or problem
- Select the stakeholders under consideration of the context facets
- If required, focus the brainstorming on one context facets
- Appoint a room and time
- Invite the participants
- Provide visualization media
- Appoint a moderator and a minute taker
- Quantity over quality
- Free association and visionary thinking are explicitly desired
- Taking on and combining expressed ideas is allowed and desired
- Criticizing other participants is forbidden even if idea seems to be absurd
- Questions for qualifications are allowed
- Even at longer-lasting deadlocks do not abort immediately – overcome atleast two longer-lasting deadlocks
- Wait until the brainstorming comes to a natural end
- Assign each idea to a category
- Discard unusable ideas
- Define how to proceed with the usable ideas
- Create the minutes to document the ideas and the procedures for further processing of ideas

Critical Success Factors:

- Observing the brainstorming rules
- Size the group influences creative performance

Osborne Checklist

- Put to other uses? (New ways to use as is? Other uses if modified?)
- Adapt? (What else is like this? What other ideas does this suggest?)
- Modify? (Change meaning, color, motion, sound, odor, taste, form, shape? Other changes?)
- Minify? (What to subtract? Eliminate? Smaller? Lighter? Slower? Split up? Less Frequent?)
- Substitute? (Who else instead? What else instead? Other place? Other time?)
- Rearrange? (Other Layout? Other sequence? Change pace?)
- Reverse? (Opposites? Turn it backward? Turn it upside down? Turn it inside out?)
- Combine? (How about a blend, an assortment? Combine purposes? Combine ideas?)

Critical Success Factors:

- Quality of questions
- Unambiguous and comprehensible statements
- Up-to-date questions

Interview

Interview Goal:

- Explicitly define the goal of the interview

Interview Participants:

- Choose the participants based on the goal of the interview
- Invite all participants in due time
- Communicate the goal and rationale of the interview to participants

Interview Location:

- Choose a location for the interview that provides an undisturbed environment and accommodates all participants

Interview Questions:

- Use open as well as closed interview questions
- Write all questions with as concrete a context as possible.
- Avoid Leading Questions

Preparation of the interviewer:

- Establish common understanding of interview questions
- Familiarize yourself with the interview partners
- Learn the terminology of the participants

Work Element:

- Ensure that elicited information is correct by summing it up for the interviewee and clarifying unclear issues

- Provide additional information beyond that provided in the invitation
- Start the interview with Introductory question

Opening:

- At the beginning, sum up the goal and rationale of the interview for the interviewees

- Create models and/or scenarios during the interview in order to get immediate feedback from the interviewee

- Document the results of the interview

Finalization:

- Sum up the elicited knowledge

- Point out to the interviewees the importance of their contributions
- Finalize the interview minutes
- Document the elicited requirements
- Revise the created models and scenarios
- Collect open issues in a to-do list
- Distribute the results to the interviewees
- Ask the interviewees to check and confirm the results
- Identify requirements conflicts in the results of the interview and resolve them using conflict resolution techniques

Critical Success Factors:

- Communication Skills
- No leading questions
- Clearly defined and communication goals
- Terminology

Workshop

Defining the workshop goal

- Defining the goal of the workshop explicitly.

Defining expected results and a work procedure:

- Define the intended results explicitly
- Define a procedure for attaining the workshop goal and producing the expected results
- Sum up the work procedure in an agenda

Selecting and inviting the participants:

- Pay attention to the workshop goal when selecting participants
- Ensure that the selection of participants is representative (e.g wrt each of four facets)
- Invite the participants in due time
- Agree upon the workshop goal with the participants

Workshop Location

- Provide a suitable room that accommodates all participants and fosters communication.
- Provide an undisturbed working atmosphere
- Provide appropriate technical equipment (whiteboard, projector, flipchart, etc)

Moderator and Minute-Taker

- Invite and appoint an external moderator, if possible

Opening

- Present the workshop goal, expected results, and the agenda at the beginning of the workshop
- Allow for a discussion of the workshop goal, the expected results, and the agenda
- Explain the techniques to be applied during the workshop

- Define and present the conversation rules on workshop

- Let participants vote on each rule

Work Element

- The moderator takes care that the participants adhere to the agenda
- The moderator observes that participants observe the conversation rules
- The minute-taker is responsible for providing documentation of all intermediate and the final-results.
- Pay attention to documenting conflicts identified during the workshop
- Take care that decisions are explicitly documented
- Try to resolve conflicts using the resolution techniques

Finalisation

- Ensure that all open issues have been documented
- Define the procedure for resolving each open issue
- Allow the participants to provide feedback on the workshop

Follow-up

- Consolidate the work results
- Ask each participant to approve the workshop minutes

Critical Success Factors:

- The ‘right’ participants
- Accepted Goal, Right Participants
- Motivation of participants
- Moderator, Groupthink Effort
- Undisturbed workshop location

Heuristics for conflict analysis

1) Checking for a data conflict

- Let the stakeholders explain the conflicting requirements
- Does one of the explanations deviate from the actual requirement? If so, a data conflict exists

2) Asking for the stakeholder's interests

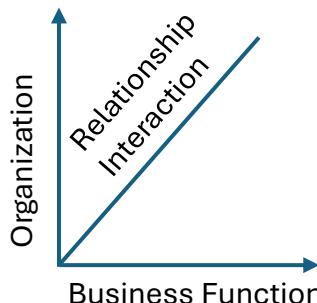
- Ask the stakeholders for their goals with regard to the conflicting requirements
- Check the resulting goals for contradictions. If there is a contradiction an interest conflict exists

3) Clarifying the stakeholders' evaluation backgrounds:

- Ask the stakeholder why he/she evaluates the conflicting requirements in the way he/she does.
- Check the evaluation backgrounds of the different stakeholders for differences that may cause the detected conflict. If there is such a difference value conflict exists

Harvard model of negotiation is based on four principles,

- ✓ Separate the person from the issue
- ✓ Negotiate not position focused but interest oriented
- ✓ To develop success criteria that a solution must fulfil
- ✓ You should have different options to choose from



B – Best
A – Alternative
T – To
N – Negotiated
A - Agreement

Situation	Goal	Possible Situation	Achieving Situation
Win – win	To make all stakeholders winners	If all conflicting parties achieve goals completely or partially in a conflict	Raise Adequate Expectations Understand how stakeholders wants to win Understand stakeholder's viewpoints Define Objective criteria Expectations shall be oriented towards expectations
Win – lose	To make atleast one party achieve its goals	If some stakeholder achieve their goals in a conflict at expense of other stakeholder	Wrong or unrealistic expectations are identified and altered
lose – lose	-	If no conflicting party achieves its goal in a conflict	-

Suitable condition for particular Elicitation Technique

Attribute	Suitable in the following situations
Conversational	When (relevant) stakeholders are available for oral information exchange
Questioning	If relevant questions can be formulated upfront; if some form of communication with stakeholders is possible; if complicated subject matter is concerned
Observational	If stakeholders cannot be addressed directly or if they are unable to state their needs and actions (detailed enough)
Artifact-based	to improve understanding of the project and of the domain
Creativity stimulating	If innovation is needed
Experiencing	If users and usability are key aspects of the project

Module 5:

Business Analysts: Best Practices



Friday 20th September



10:00 AM – 11:00 AM

Agenda:

- Responsibilities of BA In Elicitation
- Different Techniques of Requirement Elicitation



SAHIL DATERO
Business Analyst, Delivery

Candidate Resume, Interview Prep

Sahil Daterao, Business Analyst

9860-838-003 - sahildaterao0611@gmail.com - [Sahil Daterao | LinkedIn](#) - [Sahil Daterao | GitHub](#)

A seasoned candidate with 2 years-experience as business analyst and master's degree in analytical discipline. Optimizes requirement lifecycle and adheres to requirement engineering best practices resulting to delivery process with zero escalations and un-matched customer affection. Strengthens Boehm's V-Model for iterative development, fulfilment of operational objectives resulting to 100% defect free UAT delivery. Increase awareness of staff around complex managerial concepts like configuration management, COSMOD-Re, requirement modelling, Goal-Scenarios model, ScenTED approach, Kano model, UML, SysML etc. Exacting standards like ISO 27001 resulting to in-time detection of dangers, risks

- Assisting scrum team with Data, Behavioral and Functional perspective solution oriented attributes
- Conducting system readiness, functional readiness, critical design - review before production testing
- Drafting database dictionary, ERD to assist developers documenting Architecture Design Document
- Eliciting new, innovative and existing requirement from documents, stakeholders and legacy system
- Formulating strategy, plan and test cases for iterative verification and validation during system design
- Forecast tolerance limit, disruption impact, risk, recovery time for change request as per ISO 22301
- Good experience in handling and delivering multiple projects simultaneously within given constraints
- Mapping Goals & user scenarios with epic for logical decomposition of product breakdown structure
- Resolving conflicts by resetting client interests, evaluation methods, and requirement comprehension
- Retrospect sprints, estimate capacity, refine backlog, facilitate team's progress with burndown chart
- Skillful in creating BRD, FRD, SRS documents within scope of IT, usage, development, process facets
- Supporting scrum team with fine-tuned tools for requirements introducing, evaluating and modeling
- Using SysML, UML, traceability matrices to Prioritize, model, trace, manage changes in requirements
- Writing effective use cases covering functional, performance, interface, environment - requirements

Narsee Monjee Institute of Management and Studies, Mumbai Apr 2016 – May 2022

Masters of Business Administration in Technology – Business Intelligence Analytics

- **Honors:** 2.68 CGPA / 67 %
- **Relevant Course work:** Transforming inputs – Big data analytics, Neural networks, Pattern discovery - Cluster analytics, Supervised, Unsupervised learning, Quantitative techniques, Numerical methods Exploring data sources, Building predictive models, Organization behavior, Operations management

Bachelors in Technology – Mechanical Engineering

- **Honors:** 2.58 CGPA / 64.5 %
- **Relevant Course work:** Engineering Mathematics, Vibration and Materials Engineering, Heat Transfer, Fluid Machinery, Additive Manufacturing, Mechatronics and Control, Design of Machine Elements, AutoCAD Solid Modelling, Refrigeration and air cooling, Fluid Machinery, Strength of Machines

Skillset

Management and Technical

Azure / Jira	Change Management Support	Database Analytics
Defect Incident Management	ISO 27001, 22301, 20000	Microsoft excel, visio, word, ppt
Preparing BRD, FRD, SRS, SOW	Requirement Elicitation	Requirement Prioritization
Stakeholder engagement	Scope Definition	Traceability Management
Use Case Definition	User Interface Specification	Gap Analysis
CFD Simulation - Ansys	Data Science - MATLAB	Data Modelling - Talend
Prompt Engineering	Programming - Python	Predictive Analytics - SAS EM
Secure CI/CD - Azure DevOps	UML, SysML - BPMN	

IGT Infoglobaltech, Bangalore 2022 – present

Associate Business Analyst 2023 - present
Intern, Business Analyst 2022 – 2023

As 8th person to join business analysts' team in 100+ person infrastructure service provider in software industry created from scratch fully functional requirement engineering practices for ongoing projects. In two years, enabled IGT to become seasoned expert for Caribbean market retail banking industry system development in technologies like SQL Server 2016, dot. Net, Jenkins, DevSecOps. Achieved employee of year award and established policies as per ISO 27001 for credit collections department. Enhanced requirement - elicitation, validation, documentation, modelling, and prioritization practices.

- Business Continuity: Documenting software specific procedures that guide to respond, recover, resume and restore to a pre-defined level of operation resulting to a fruitful disruption compliant to ISO 22301
- Conflict Resolution: Identifying conflicts, classify into data, value, interests and structural resulting to instant conflict resolution using technique - BATNA turned near to dying project to 140% profitable
- C4 Model Use case: Spearhead abstraction level documentation of code, component, container and context resulting for testers, programmers, architects a reference to operational objectives of mission
- Diagramming with code: Implementing 'abstraction-first' approach to software architecture diagramming resulting to scalable, universal method to design developer friendly architecture using Visio, PlantUML
- Database Modelling: Draft data dictionary, data flow diagram of 'As-is' and 'To-be' workflows, catering trade study alternative choices to stakeholders resulting to change management as per ITIL/ISO 20000
- Defect Resolution: Helping scrum team resolve issues with tri-faceted insights, and provisioning to client data library, code scripts, process flow diagrams, requirements baseline, meeting briefs
- Exacting Standards: Providing curated ISMS Scope document, Architecture design document template, Standard for documentation structures resulting to in-time delivery of technical reference document
- Elicitation: Curate action plan resulting to knowing stakeholder's needs, desires and requirements
- Focus Validation: Establish iterative V-model testing focusing on all abstraction levels like system, component and integration resulting to user acceptance of CoSaCs (dot. NET) retail banking software
- Modelling using Visio: Develop using PlantUML in less than 25 days, 120+ SysML diagrams representing classes, interfaces, objects, functions, process models within DR Reporting application source code
- Review and Validation: Engaging with SMEs and consultants to update Apache Service Mix ESB from 3.4 to 7.0 resulting to technological upgrade of software defined hardware accelerated banking software
- Sprint refinement with Azure: Allocating technical requirements in measure of capacity, story point, efforts, performance and time resulting to establishment of distinct product breakdown structure
- Stakeholder Engagement: Establishing effective communication resulting to common consensus of requirements with all stakeholders across multiple facets using decision matrix, stakeholder map
- Streamline Migration: Conduct data mapping for integration nodes of CoSaCs Sales POS function and Oracle Xstore resulting to seamless functional and logical decomposition till necessary depth
- Test Planning: Defining scope, long and short term action plan, data models for components of banking software resulting in good faith with client due to its consistency in functionality post version update
- Testing and Quality Guarantee: Assess migration change request risk of CoSaCs Sales POS function trade workflow resulting to tested and quality assured impact analysis prior to system deployment

Achievements and Side Projects

- Certification of Completion - Python Training (shareable)
- Introduction to AI in Data Center NVIDIA (shareable)
- DevSecOps with Azure DevOps: Secure CI/CD (shareable)
- Using Cameo Tool streamline system design - MBSE Execution (under review)
- MATLAB Programming for Numerical Computation (is active)
- Working Group contributor of 5 IEEE Committees in fields of AI / ML, LLM, SAR (is active)
- CI/CD pipeline for Data Science, AI projects using tensorflow and SQL (is active)
- Documentation practices compliant to ISO 27001, ISO 22301, ISO 45001, ISO 14001 (is active)

<https://youtu.be/Tt08KmFfIYQ?si=mYj-sq5lmxvA0HdX>

Microsoft Edge
PDF Document

IIBA
IREB
INCOSE

Page 2 of 3

Candidate Resume, Interview Prep

- 1) Tell me about yourself (education, experience, why are you wanting to job change? What value can you bring to the organization?)
- 2) Tell me about your project (business Functions 3-4-7 minutes, 3-4 lines about BA role in project)
- 3) What were your responsibilities as BA in your project (Showcase skills from your CV are relevant Job Description – ChatGPT 10JDs.)
- 4) Why you want to leave prior company?
- 5) What have you researched so far about this company?

Task List

Youtube -> Profile optimization for

- 1) [LinkedIn](#)
- 2) [Naukri, Monster jobs](#)
- 3) [CV Optimization \(Job Description + ChatGPT\)](#)
- 4) [Interview Prep](#)

+ [Skillset](#) Learn in public: [\(5\) savinder puri – YouTube](#)

Project example:

- 1) SysML Diagram in this ppt to draw.io
 - 2) Structuring BRD, FRD, SRS
 - 3) Youtube Data analyst projects (SQL, Python, Word Doc)
- [Learn in Public](#)

