ARCADIA REFERENCE: WORKFLOW AND ACTIVITIES

Arcadia Engineering Activities
Contents & Orchestration

Jean-Luc Voirin

Table of Contents

1 Scope of this document

ARCADIA is a tooled method devoted to systems & architecture engineering, supported by Capella modelling tool.

It describes the detailed reasoning to

- *understand the real customer need,*
- *define and share the product architecture among all engineering stakeholders,*
- *early validate its design and justify it,*
- *ease and master Integration, Validation, Verification, Qualification (IVVQ).*

It can be applied to complex systems, equipment, software or hardware architecture definition, especially those dealing with strong constraints to be reconciled (cost, performance, safety, security, reuse, consumption, weight…).

It is intended to be used by most stakeholders in system/product/software or hardware definition and IVVQ as their common engineering reference and collaboration support.

ARCADIA stands for ARChitecture Analysis and Design Integrated Approach.

This document provides a detailed view of engineering activities defined and supported by Arcadia, their relations and the way they interact.
The high level tasks and activities described in Arcadia User Guide are duplicated in this document, but each of them is further decomposed and detailed, down to elementary activities, including lower level figures.

2 Arcadia Reference Documents

An in-depth introduction and description of Arcadia, with explanations on the method, on the language, illustrated by detailed examples of application, can be found in the Arcadia reference book:

Jean-Luc Voirin, 'Model-based System and Architecture Engineering with the Arcadia Method', ISTE Press, London & Elsevier, Oxford, 2017

A presentation of Arcadia main principles and concepts can be found in the following online documents, including this one:

- Arcadia Engineering Landscape: an introduction to Engineering as supported by Arcadia
- Arcadia User Guide: a first level description of Arcadia approach and main engineering Tasks
- Arcadia Reference - Activities: an in-depth description of Arcadia tasks and activities
- Arcadia Reference - Data Model: data created and exploited by these activities
- Arcadia Reference - Capabilities: main processes supporting engineering
- Arcadia Language - MetaModel: a more formal description of Arcadia language concepts
- Arcadia Q&A: real life questions and answers on deploying Arcadia

See table 'Summary of reference Documents Contents' next page.

For easier navigation capabilities (including in diagrams, between activities and data, etc.), a web version can be browsed here.

Advanced practitioners in modelling and Arcadia can also access the Arcadia-compliant Capella model of Arcadia, from which this material is automatically extracted, here.

Summary of reference Documents Contents		Book	Landscape	User Guide	Reference - Activities	Reference - DataModel	Reference - Capabilities	Language - MetaModel	Q&A
History	Why was the method created and tooled? For which purpose? With which benefits?	✓							
Philosophy	What are its objectives and expected scope? What are its specificities?	✓	(✓)						
	How does it address Engineering Issues and Challenges?	✓		(✓)			(✓)		
	What kind of major levers does it use to address them?	✓		(✓)			(✓)		
Principles and approach	What are the drivers of each core perspective ....? How to build each of them?	✓		(✓)	(✓)				
	How to address Major engineering Issues using Arcadia and these perspectives?	✓					✓		
Details for implementation	What are the detailed processes to build each of the core perspectives?	(✓)			✓				
	How and where are engineering data elaborated and used to address major engineering challenges?	✓			(✓)	(✓)	✓		
	What is the formal definition of the Arcadia language & concepts?	✓				(✓)		✓	
	Examples and samples of models?	✓							
Hints for Deployment	Which major questions arise in projects applying Arcadia?								✓

✓ : fully detailled (✓) : simplified or partial

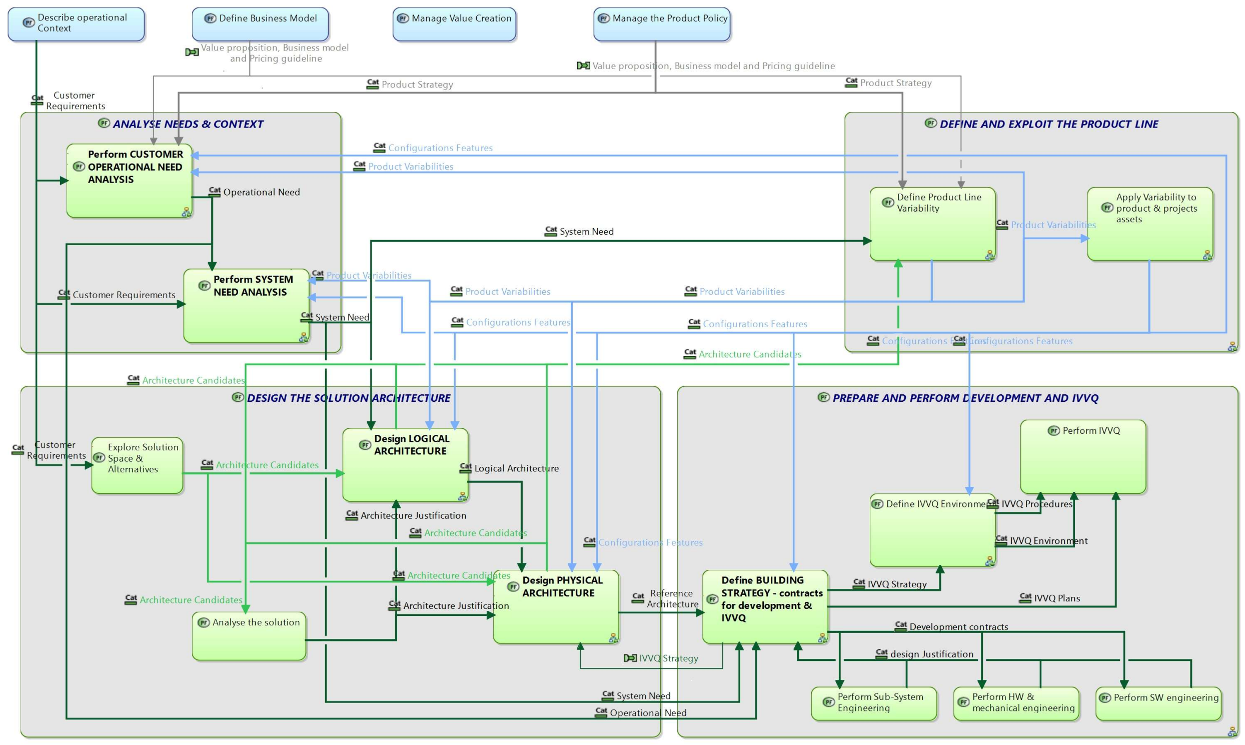
3 Arcadia Workflow at a glance

This figure provides a synthetic view of Arcadia tasks and major categories of interactions between these tasks.

REMINDER:

Iterations and loops are necessary in real life conditions, yet they are not represented here for sake of simplicity.
Although the workflow described here appears to be straightforward, activities may be carried out in a different order; however, for best quality of engineering results, each activity should not be fully completed without having checked its outcome against its expected entries for consistency.

The links mentioned here are to be considered as dependency links, but not necessarily time-related ordering of steps & tasks.
While preserving these dependencies, any process or order can be used:
- top-down or waterfall approach,
- bottom-up and reuse-driven approaches,
- iterative or incremental processes,
- ...



Each high-level task and its detailed activities are described below.

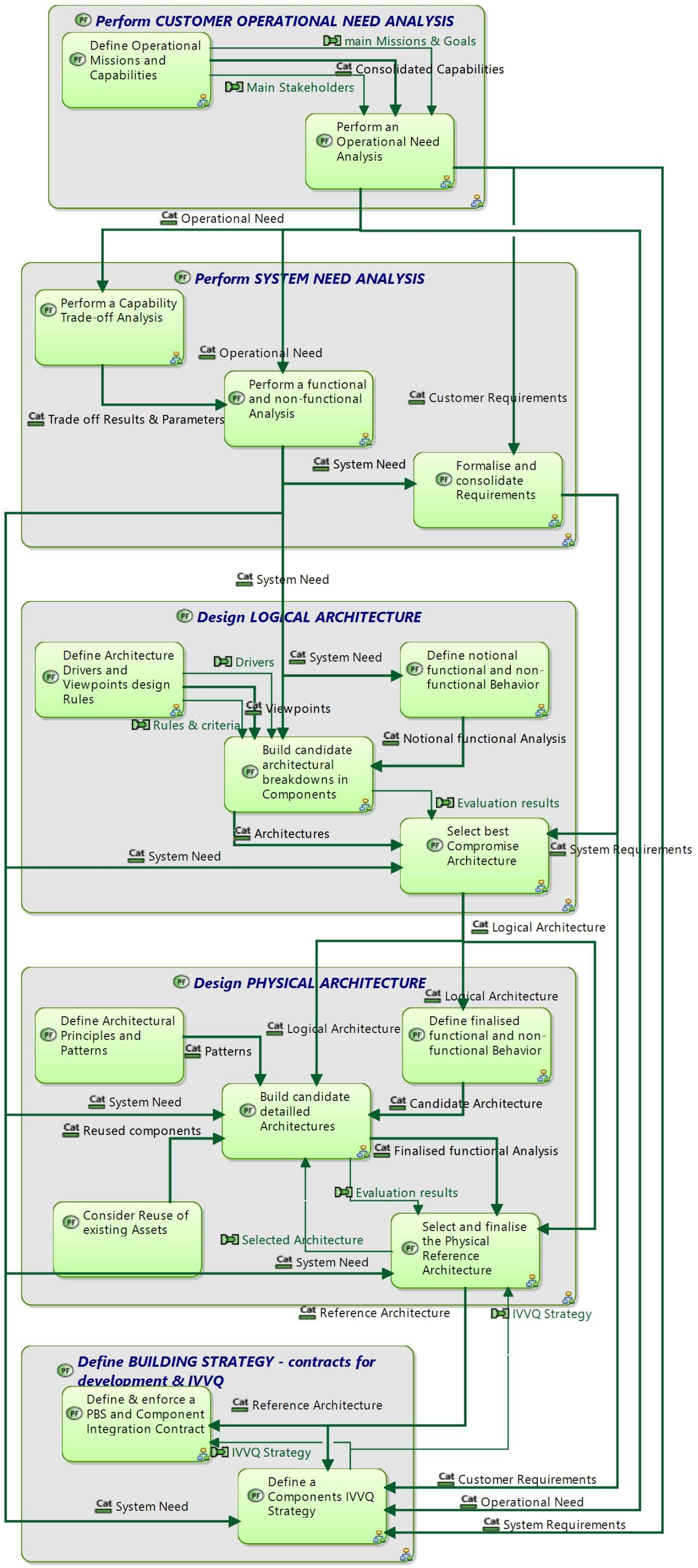
4 Focus on model-building activities

This figure introduces the major core aspects (called perspectives) of Arcadia framing for engineering, each producing engineering data useful for others. It focuses on core perspectives structuring both definition and collaboration, giving one level of detail for each perspective.

NOTES:

Iterations and loops are necessary in real life conditions, yet they are not represented here for sake of simplicity.
Although the workflow described here appears to be straightforward, activities may be carried out in a different order; however, for best quality of engineering results, each activity should not be fully completed without having checked its outcome against its expected entries for consistency.

The links mentioned here are to be considered as dependency links, but not necessarily time-related ordering of steps & tasks.
While preserving these dependencies, any process or order can be used:
- top-down or waterfall approach,
- bottom-up and reuse-driven approaches,
- iterative or incremental processes,
- …


Perform CUSTOMER OPERATIONAL NEED ANALYSIS
Define Operational Missions and Capabilities
main Missions & Goals
Consolidated Capabilities
Main Stakeholders
Perform an Operational Need Analysis
Operational Need
Perform SYSTEM NEED ANALYSIS
Perform a Capability Trade-off Analysis
Operational Need
Perform a functional and non-functional Analysis
Customer Requirements
Trade off Results & Parameters
System Need
Formalise and consolidate Requirements
System Need
Design LOGICAL ARCHITECTURE
Define Architecture Drivers and Viewpoints design Rules
Drivers
System Need
Define notional functional and non-functional Behavior
Viewpoints
Rules & criteria
Notional functional Analysis
Build candidate architectural breakdowns in Components
Evaluation results
Architectures
Select best Compromise Architecture
System Requirements
System Need
Logical Architecture
Design PHYSICAL ARCHITECTURE
Logical Architecture
Define Architectural Principles and Patterns
Logical Architecture
Define finalised functional and non-functional Behavior
Patterns
System Need
Build candidate detailled Architectures
Candidate Architecture
Reused components
Finalised functional Analysis
Evaluation results
Consider Reuse of existing Assets
Selected Architecture
Select and finalise the Physical Reference Architecture
System Need
Reference Architecture
IVVQ Strategy
Define BUILDING STRATEGY - contracts for development & IVVQ
Define & enforce a PBS and Component Integration Contract
Reference Architecture
IVVQ Strategy
Define a Components IVVQ Strategy
Customer Requirements
Operational Need
System Need
System Requirements

5 Arcadia Tasks and Activities

Each activity is described below, along with its inputs and outputs, other activities interacting with it, and its sub-activities.

For key activities, the way to elaborate data is also described in each activity description field, as indicated by **<Data elaboration description>** tag. Engineering data mentioned in this description (identified as **data**) are described in another document.

5.1 ANALYSE NEEDS & CONTEXT

Understand and analyse the global need of stakeholders down to customer and system requirements

See sub-tasks description

5.1.1 Perform CUSTOMER OPERATIONAL NEED ANALYSIS

The first perspective on system engineering brought by Arcadia focuses on analyzing the stakeholders needs and goals, their expected missions and activities, far beyond (and often before) customer requirements. This analysis also contributes ensuring adequate system need understanding with regard to its real operational use and IVVQ conditions, but it does not consider the solution or system per se.

Outputs of this engineering activity mainly consist of an "operational architecture" which describes and structures the stakeholders need in terms of actors/users, their operational capabilities and activities (including operational use scenarios with dimensioning parameters, and operational constraints such as safety, security, lifecycle, etc.).

See **<Data elaboration description>** below

Engineering goals

- Understand the real Customer Need to address, in terms of Tasks to be completed by users
- Check the Need Consistency, Completeness

- Collect Material for future technical Trade-offs, Optimisations, Negotiations with Customer
- Ensure realism/relevance of IVVQ operational scenarios & tests.

Tasks to be completed during this step

- Define Operational Missions & Capabilities
- Perform an operational Need Analysis

Stop Criteria

This step is achieved when agreement with higher level stakeholders (incl. Customer if possible) is obtained on the description of the operational need.

Contributors & Competencies

Major competencies required to complete this step are suggested below:

Required Competencies	Major Expected Contribution	Possible Contributors (*)
Domain knowledge		• Chief architect • Customer • Operational expert • Systems engineering manager • Specialty engineering expert • IVVQ manager • Product line manager • Simulation expert • Program manager
Operational Domain	Operational Use & constraints	
Product & Technical Domain (incl. Product line)	Product line constraints, capability gaps	
Systems engineering - Design		

Operational Analysis	OA modeling
Modeling & viewpoint engineering techniques	Modeling techniques
Value Engineering	Customer & Stakeholders Stakes & expected value
Systems engineering - Specialties	
Technico-operational simulation	Realism of operational scenarios & OA
Specialties engineering (safety, perf, RAMST…)	Operational constraints & cases
Systems engineering / IVVQ	
Test & Trials Strategy Plan	Relevant operational scenarios
Integration means definition	Operational test capabilities

(*) Depending on each organisation, competencies may be allocated to different actors; the following contributors list is therefore just an example to be adapted to each organisation:

<Data elaboration description>

The **Operational Mission Analysis** identifies what the end users expect to carry out, starting with **Users Missions & Capabilities** : Missions being major goals of main stakeholders, Operational Capabilities being services that end users should be able to realise to fulfil mission.

Then it identifies **Operational Entities/actors** involved in the mission, the **Operational Activities** they are expected to perform, and "user stories" described as **Operational Processes** (or tasks) involved in each capability, and time-related **Operational Scenarios** involving **Operational Activities**.

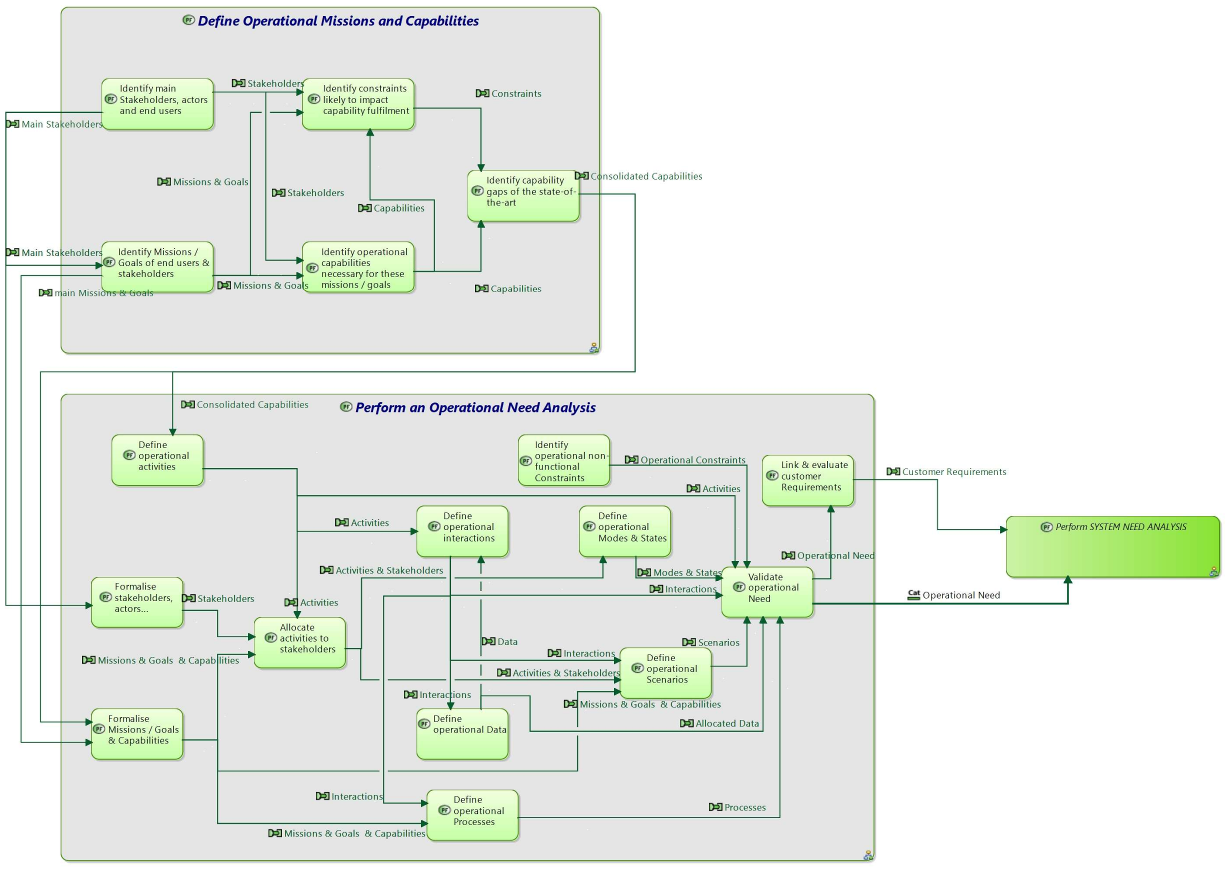
This process may lead to early defining constraints on the expected system itself, that can be captured in an initial version of **System Need Specification**, and **Textual Requirements**.

<Data elaboration description end>

This figure provides an inner view of the task Perform CUSTOMER OPERATIONAL NEED ANALYSIS, in the context of Arcadia core perspectives.

For each of the first level tasks presented here, only links with other core perspectives are displayed.

For a complete view of links with other engineering tasks and activities, see the figure dedicated to this first level task.


Define Operational Missions and Capabilities
Identify main Stakeholders, actors and end users
Stakeholders
Identify constraints likely to impact capability fulfilment
Constraints
Main Stakeholders
Missions & Goals
Stakeholders
Capabilities
Identify capability gaps of the state-of-the-art
Consolidated Capabilities
Main Stakeholders
Identify Missions / Goals of end users & stakeholders
Identify operational capabilities necessary for these missions / goals
Missions & Goals
main Missions & Goals
Capabilities
Consolidated Capabilities
Perform an Operational Need Analysis
Define operational activities
Identify operational non-functional Constraints
Operational Constraints
Link & evaluate customer Requirements
Customer Requirements
Activities
Activities
Define operational interactions
Define operational Modes & States
Perform SYSTEM NEED ANALYSIS
Operational Need
Activities & Stakeholders
Modes & States
Validate operational Need
Interactions
Formalise stakeholders, actors...
Stakeholders
Activities
Operational Need
Allocate activities to stakeholders
Data
Scenarios
Missions & Goals & Capabilities
Interactions
Define operational Scenarios
Activities & Stakeholders
Interactions
Missions & Goals & Capabilities
Formalise Missions / Goals & Capabilities
Define operational Data
Allocated Data
Interactions
Define operational Processes
Processes
Missions & Goals & Capabilities

5.1.1.1 Define Operational Missions and Capabilities

Identify what the end users expect to carry out,

- Missions, major goals of main stakeholders
- Operational Capabilities the system/SW should contribute to

Here, capabilities should be seen as quantified overall operational goals, results, services that are expected from end users[1].
e.g. "localise", "follow route", "track"; or more extensively: “ability to detect/locate a given kind of target in such area in less than such time”.

Assess qualitative but also quantitative metrics or parameters to quantify expected capabilities (e.g. precision of localisation).

Identify any constraints likely to impact capability fulfilment, such as

- Concepts & Doctrine,
- Organisation functioning,
- Infrastructures,
- Equipment (including system/SW),
- Whole system life cycle cost,
- Logistics, Deployment, Sustainability,
- Human Factors,
- Competencies, Training, …

Note: This analysis is often known as "DOTMLPF" (Doctrine, Organization, Training, Materiel, Leadership and Education, Personnel, and Facilities).

Identify capability gaps (capability analysis[2]) between expressed capability need above , and existing systems (including systems of previous generations, and competitors).

[1] Capability: "The ability to execute a specified course of action. (A capability may or may not be accompanied by an intention.)" US DoD;

"The appropriation combination of competent people, knowledge, money, technology, physical assets, systems and structures necessary to deliver a specified level of performance in pursuit of the organisation's objectives, now and/or in the future" NZ Gov.(

[2] "Capability Analysis : A tool of statistical measurement used to determine capability by comparing a process's actual performance with customer expectations." www.surveymethods.com

"Capability analysis is a set of calculations used to assess whether a system is statistically able to meet a set of specifications or requirements." www.capability-analysis.com

Input:

- Customer Requirements, non formal or textual description.
- Other Customer documents, including Use Cases, Scenarios, Domain Analysis, Capabilities Analysis, Operational and first System models,

- Customer and end users interviews.
- Existing previous generation systems.

Output:

- Missions, goals & capabilities.
- Capability gaps of existing solutions.

Target documents:

- System Segment Specification (SSS)
- Operational Concept Document (OCD) – for more detailed description if needed.

Verification and Consistency checks:

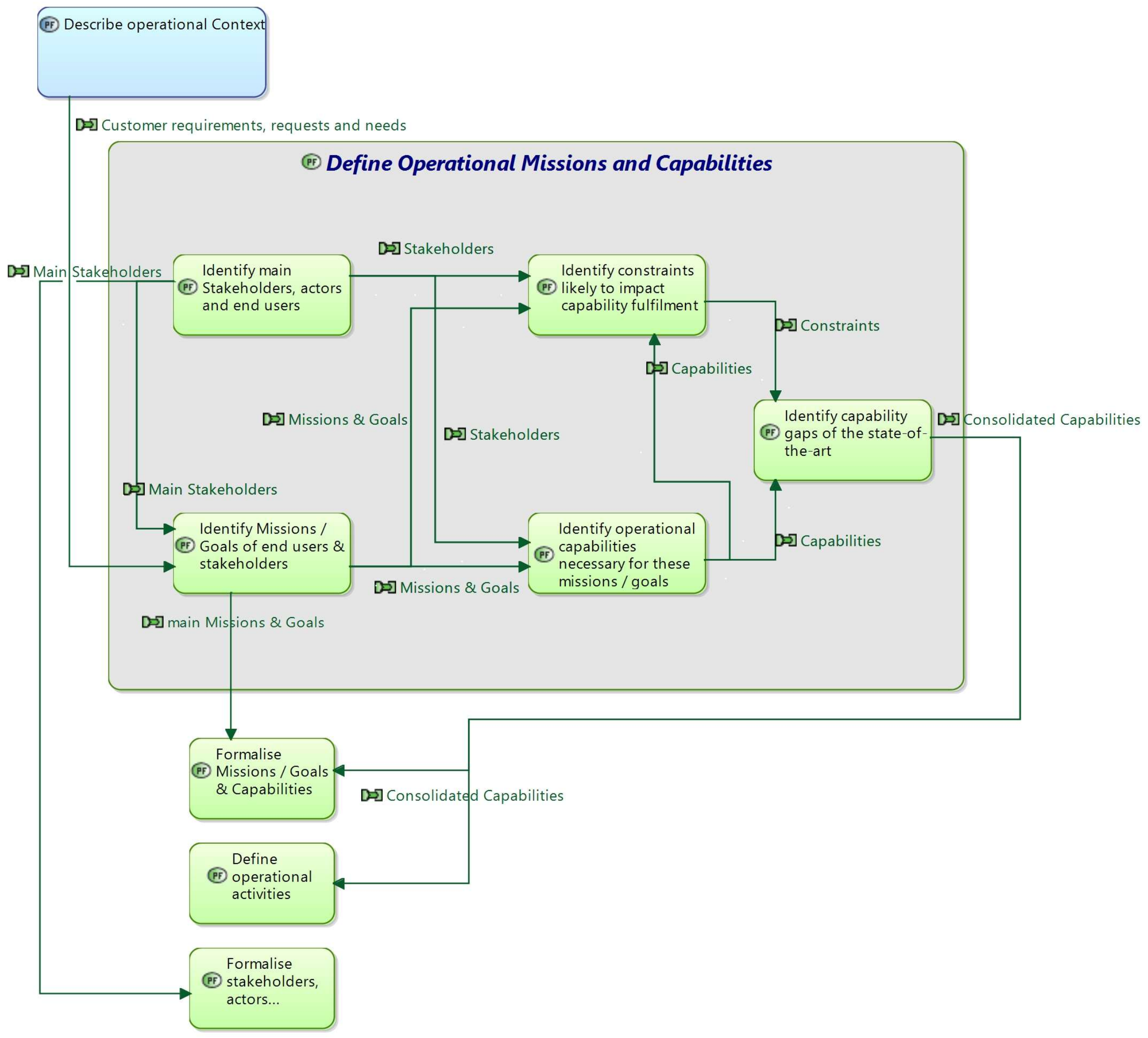
External consistency:

- Between customer documents, and capability analysis products (including models).

Internal consistency:

- Between outputs of the analysis.
- Verify the Need Description: coherent, complete, relevant: no contradiction, no gap, no inaccuracy.

This figure describes the interactions of the considered task with other engineering activities.


Describe operational Context
Customer requirements, requests and needs
Define Operational Missions and Capabilities
Main Stakeholders
Identify main Stakeholders, actors and end users
Stakeholders
Identify constraints likely to impact capability fulfilment
Constraints
Capabilities
Missions & Goals
Stakeholders
Identify capability gaps of the state-of-the-art
Consolidated Capabilities
Main Stakeholders
Identify Missions / Goals of end users & stakeholders
Identify operational capabilities necessary for these missions / goals
Capabilities
Missions & Goals
main Missions & Goals
Formalise Missions / Goals & Capabilities
Consolidated Capabilities
Define operational activities
Formalise stakeholders, actors...

5.1.1.1.1 Identify main Stakeholders, actors and end users

including expected users of the system, operational entities, organisations and actors that should interact with end users

5.1.1.1.2 Identify Missions / Goals of end users & stakeholders

both high level objectives and contextual ones.

including unexpected or hostile stakeholders goals.

5.1.1.1.3 Identify constraints likely to impact capability fulfilment

such as

·Concepts & Doctrine,

·Organisation functioning,

·Infrastructures,

·Equipment (including system/SW),

·Whole system life cycle cost,

·Logistics, Deployment, Sustainability,

·Human Factors,

·Competencies, Training, …

Note: This analysis is often known as "DOTMLPF" (Doctrine, Organization, Training, Materiel, Leadership and Education, Personnel, and Facilities).

5.1.1.1.4 Identify operational capabilities necessary for these missions / goals

Here, capabilities should be seen as quantified overall operational goals, results, services that are expected from end users .

e.g. "localise", "follow route", "track"; or more extensively: "ability to detect/locate a given kind of target in such area in less than such time".

5.1.1.1.5 Identify capability gaps of the state-of-the-art

between expressed capability need above , and existing systems (including systems of previous generations, and competitors), organisations, etc.

(capability analysis)

5.1.1.2 Perform an Operational Need Analysis

Based on the former missions, goals & capabilities of end users, define the organisation, behaviour and results expected from them:

- Operational Context & Stakeholders: organisation using the system, actors, geographical or organisational nodes / operational entities
- Operational processes (or tasks), necessary for each expected capability

- Activities being necessary to perform each operational process, allocated to actors, operational entities
- Relationships & interchanged data/information between activities (and actors); including required standards, interfaces
- Operational modes & states (e.g. mission phases…)
- Operational Scenarios illustrating superimposition of processes for a given situation, orchestration of operational activities.

Identify all major dimensioning needs, and [non-functional] constraints, relating them to the appropriate missions, activities, actors… and associated scenarios (including worst case or feared events for each of them), and relate them to operational scenarios: e.g.:

- Performance issues, reactivity/latency constraints…
- Parallelism in activities
- Human factors
- Safety-related issues, concerning actors and neighbourhood/environment
- Security issues
- …

Evaluate operational importance/value of each customer requirement (how much it contributes to reaching expected goals and capabilities, operational activities).

Input:

- Customer Requirements, non formal or textual description.
- Other Customer documents, including Use Cases, Scenarios, Domain Analysis, Capabilities Analysis, Operational and first System models,
- Customer and end users interviews.
- Existing previous generation systems.

Output:

- Missions, goals & capabilities.

- Operational Analysis results: Consolidated Use Cases, Scenarios, operational processes and Activities, data and exchanged data flows, organisational actors and nodes/organisations, operational modes & states.

Target documents:

- System Segment Specification (SSS)
- Operational Concept Document (OCD) – for more detailed description if needed.

Verification and Consistency checks:

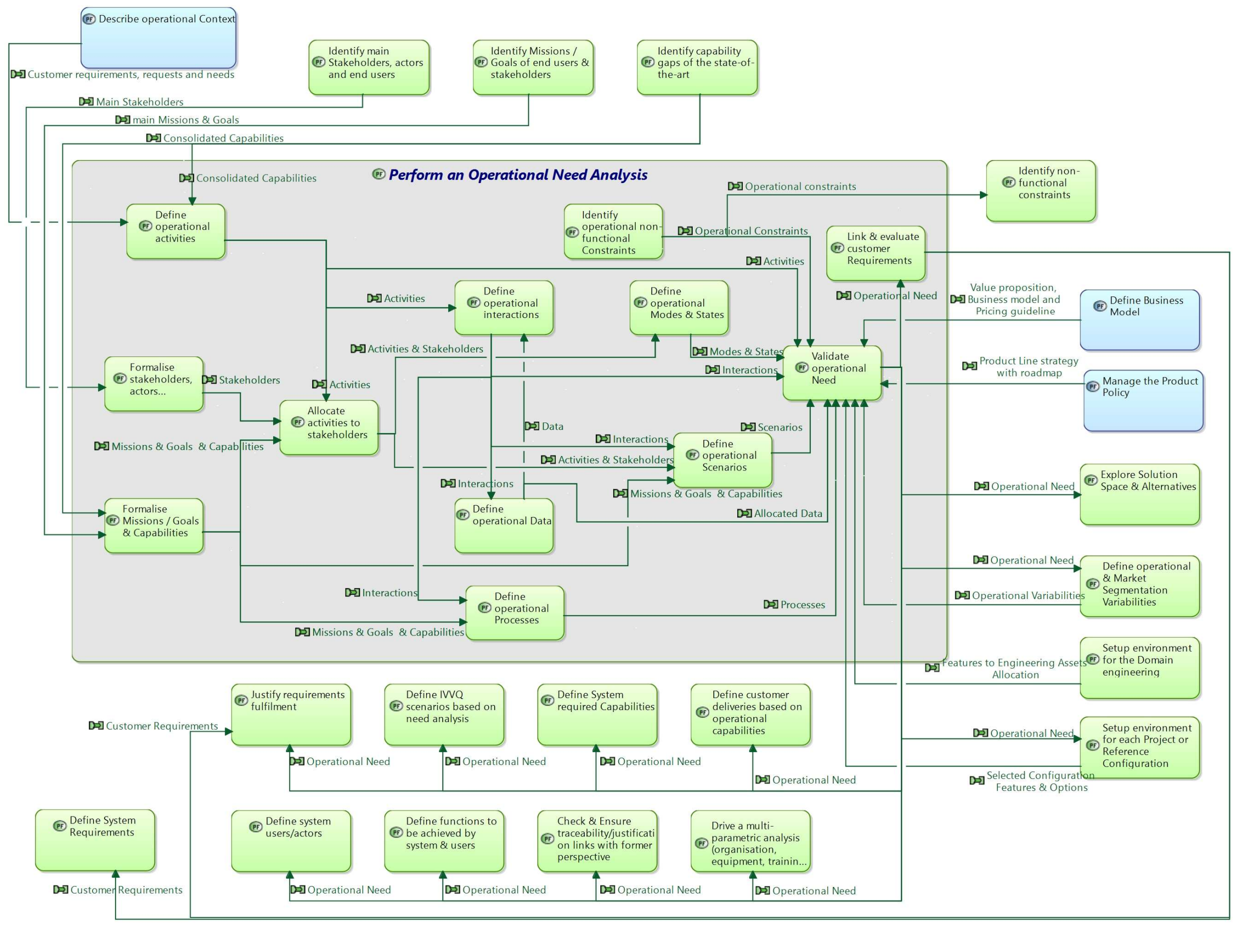
External consistency:

- Between customer documents, and operational analysis products (including models).
- Checking need completeness (e.g. a requirement not relatable to any operational activity might denote a lack in need analysis, and vice versa),

Internal consistency:

- Between outputs of the analysis.
- Verify the Operational Need Description: coherent, complete, relevant: no contradiction, no gap, no inaccuracy.

This figure describes the interactions of the considered task with other engineering activities.


Describe operational Context
Customer requirements, requests and needs
Identify main Stakeholders, actors and end users
Identify Missions / Goals of end users & stakeholders
Identify capability gaps of the state-of-the-art
Main Stakeholders
main Missions & Goals
Consolidated Capabilities
Perform an Operational Need Analysis
Consolidated Capabilities
Operational constraints
Identify non-functional constraints
Define operational activities
Identify operational non-functional Constraints
Operational Constraints
Link & evaluate customer Requirements
Activities
Activities
Define operational interactions
Define operational Modes & States
Operational Need
Value proposition, Business model and Pricing guideline
Define Business Model
Activities & Stakeholders
Modes & States
Validate operational Need
Formalise stakeholders, actors...
Stakeholders
Activities
Interactions
Product Line strategy with roadmap
Manage the Product Policy
Allocate activities to stakeholders
Data
Scenarios
Missions & Goals & Capabilities
Interactions
Define operational Scenarios
Activities & Stakeholders
Interactions
Missions & Goals & Capabilities
Operational Need
Explore Solution Space & Alternatives
Formalise Missions / Goals & Capabilities
Define operational Data
Allocated Data
Operational Need
Define operational & Market Segmentation Variabilities
Interactions
Define operational Processes
Processes
Operational Variabilities
Missions & Goals & Capabilities
Features to Engineering Assets Allocation
Setup environment for the Domain engineering
Justify requirements fulfilment
Define IVVQ scenarios based on need analysis
Define System required Capabilities
Define customer deliveries based on operational capabilities
Customer Requirements
Operational Need
Setup environment for each Project or Reference Configuration
Operational Need
Operational Need
Operational Need
Operational Need
Selected Configuration Features & Options
Define System Requirements
Define system users/actors
Define functions to be achieved by system & users
Check & Ensure traceability/justification on links with former perspective
Drive a multi-parametric analysis (organisation, equipment, trainin...
Customer Requirements
Operational Need
Operational Need
Operational Need
Operational Need

5.1.1.2.1 Formalise Missions / Goals & Capabilities

later linked to operational scenarios and processes involved

5.1.1.2.2 Formalise stakeholders, actors...

Operational Context & Stakeholders

-organisation using the system,

-actors,

-geographical or organisational nodes / operational entities

5.1.1.2.3 Define operational activities

either for each stakeholder, or for each mission/capability

5.1.1.2.4 Allocate activities to stakeholders

activities are expected to contribute to missions and capabilities

5.1.1.2.5 Define operational interactions

interactions as exchanges, communications, shared information, events or commands between activities and therefore between stakeholders

interactions can be allocated to physical communication means between stakeholders

5.1.1.2.6 Define operational Data

Data manipulated by stakeholders, to be exchanged between them;

Relationships & interchanged data/information between activities (and actors); including required standards, interfaces

5.1.1.2.7 Define operational Processes

Operational processes (or tasks), necessary for each expected capability / mission, defined by chaining different activities

5.1.1.2.8 Define operational Scenarios

expressing the time-related interactions between activities and between stakeholders, using operational interactions previously defined; , necessary for each expected capability / mission

5.1.1.2.9 Define operational Modes & States

Allocated to stakeholders

Example: missions modes or phases, context for capability use, for operational process involvement...

including activities available in each mode or state

5.1.1.2.10 Identify operational non-functional Constraints

all major dimensioning needs, and [non-functional] constraints, relating them to the appropriate missions, activities, actors… and associated scenarios (including worst case or feared events for each of them), and relate them to operational scenarios: e.g.

·Performance issues, reactivity/latency constraints…

·Parallelism in activities

·Human factors

·Safety-related issues, concerning actors and neighbourhood/environment

·Security issues

·…

5.1.1.2.11 Validate operational Need

against all former elements.

5.1.1.2.12 Link & evaluate customer Requirements

Formalise operational requirements using activities, processes, scenarios, modes & states... and link them to eachother.

Evaluate operational importance/value of each customer requirement (how much it contributes to reaching expected goals and capabilities, operational activities), thanks to these links.

5.1.2 Perform SYSTEM NEED ANALYSIS

This perspective focuses on the system itself, in order to define how it can contribute to satisfy the former operational needs, along with its expected behavior and qualities. The main goal at that point is to check the feasibility of stakeholders requirements (cost, schedule, technology readiness, etc.) and if necessary, to provide means to renegotiate their content.

Outputs of this engineering activity mainly consist of system functional need descriptions (system capabilities, functions or services, functional chains, and scenarios), interoperability and interaction with the users and external systems (functions, exchanges plus non-functional constraints).

See **<Data elaboration description>** below

Engineering goals

- Define functional and non-functional need/expectations for system/SW

- Check Feasibility of Requirements (tech., cost, schedule, ...)
- Find most structuring/constraining Requirements for operational purpose
- Evaluate their impact on design & integration – therefore their cost range
- Confront Requirements with Reuse opportunities
- Get technical Material to support Negotiation by evaluating operational added value of each requirement

Tasks to be completed during this step

- Perform a Capability Trade-off Analysis
- Perform a functional and non-functional Analysis
- Formalise and consolidate Requirements

Stop Criteria

This activity is achieved when are obtained

- risk mitigation on System/SW definition (requirements consistency, functional need validity, cost estimation, adequacy to operational need)
- and sufficient definition for decision making to proceed with further design (early architecture & EPBS, requirements, system/SW functional need)
- it usually requires agreement with higher level stakeholders (incl. customer).

Contributors & Competencies

Major competencies required to complete this step are suggested below:

Required Competencies	**Major Expected Contribution**	**Possible Contributors (*)**
Domain knowledge		• Chief architect • Customer • Systems engineering manager

		• Functional analyst • Specialty engineering expert • IVVQ manager • Product line manager • Simulation expert • Program manager
Product & Technical Domain (incl. Product line)	Product line constraints, capability gaps	
Systems engineering - Design		
Functional & non functional Analysis	System Analysis	
Modeling & viewpoint engineering techniques	Modeling techniques	
Value Engineering	Cost effectiveness of functions	
Systems engineering - Specialties		
Technico-operational simulation	Adequacy to operational scenarios & OA	
Technical Simulation	validity of functional behaviour	
Specialties engineering (safety, perf, RAMST…)	Non-functional constraints	
Systems engineering / IVVQ		
Test & Trials Strategy Plan	Relevant non-functional scenarios	
Integration means definition	Functional/non-functional test capabilities	

(*) Depending on each organisation, competencies may be allocated to different actors; the following contributors list is therefore just an example to be adapted to each organisation:

<Data elaboration description>

Based on User/Customer **Textual Requirements**, and on **Operational Mission Analysis**, the **System Need Specification** is performed to scope expectations on the system.

Specified Functions or services required from the system to contribute to each **Operational Activities** are identified;
 Spec. Functional Exchanges between the system and **External systems/actors** (with which it is expected to interact) are defined, along with the **Spec. Exchange contents** to be exchanged between them.

The use cases describing the way the system should be used are captured by **Specified Capabilities**, illustrated by **Spec. Functional Chains** and **Specified Scenarios**; each of them involves **Specified Functions** to express expectations on the system. These use cases should also be inspired by **Operational Processes** and **Operational Scenarios**, and traced towards them.

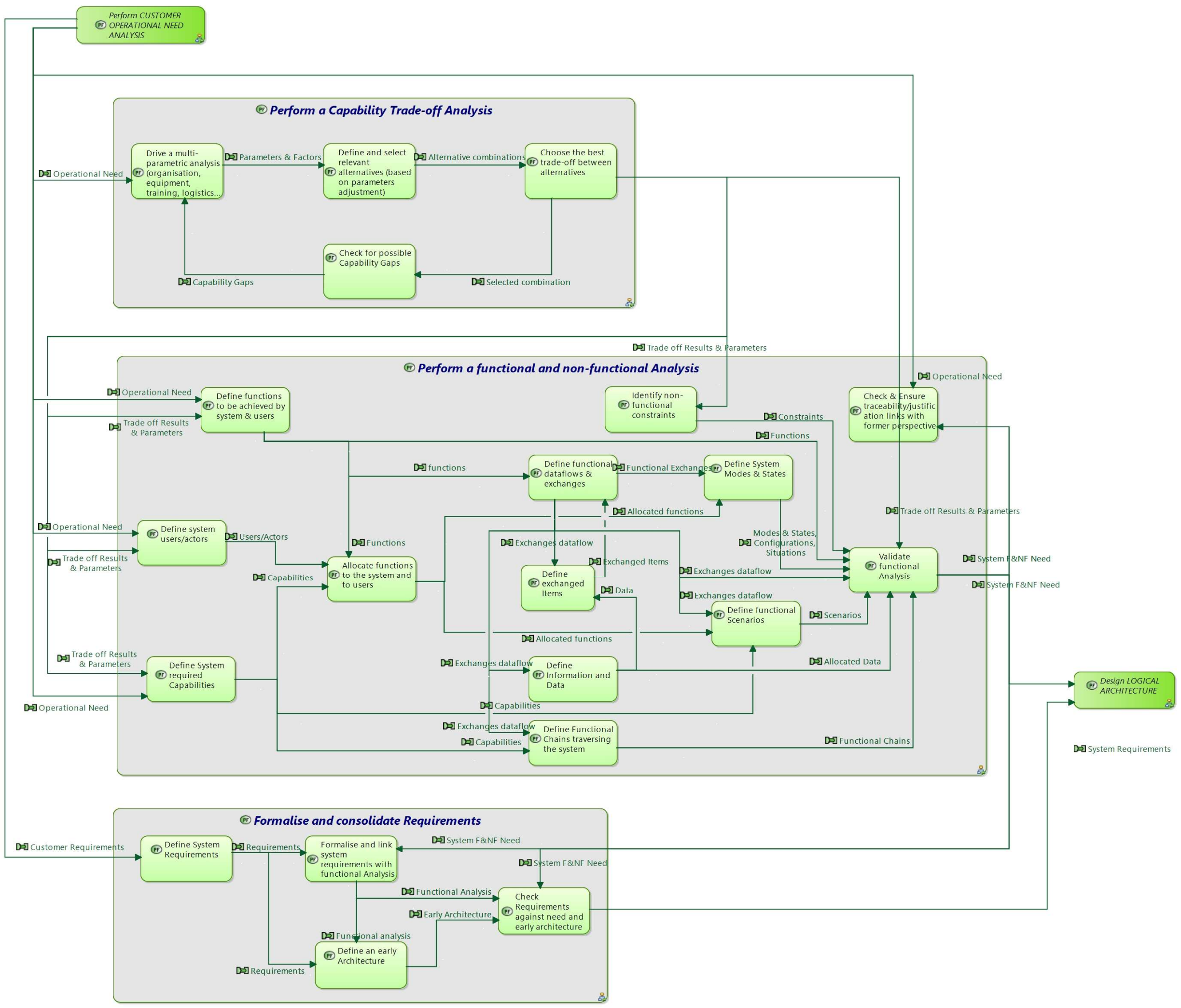
All these elements are related to user **Textual Requirements**, and can be also complemented by system-level **Textual Requirements**.

<Data elaboration description end>

This figure provides an inner view of the task Perform SYSTEM NEED ANALYSIS, in the context of Arcadia core perspectives.

For each of the first level tasks presented here, only links with other core perspectives are displayed.

For a complete view of links with other engineering tasks and activities, see the figure dedicated to this first level task.


Perform CUSTOMER OPERATIONAL NEED ANALYSIS
Perform a Capability Trade-off Analysis
Operational Need
Drive a multi-parametric analysis (organisation, equipment, training, logistics...
Parameters & Factors
Define and select relevant alternatives (based on parameters adjustment)
Alternative combinations
Choose the best trade-off between alternatives
Check for possible Capability Gaps
Capability Gaps
Selected combination
Trade off Results & Parameters
Perform a functional and non-functional Analysis
Operational Need
Define functions to be achieved by system & users
Trade off Results & Parameters
Identify non-functional constraints
Constraints
Functions
Check & Ensure traceability/justification links with former perspective
Operational Need
functions
Define functional dataflows & exchanges
Functional Exchanges
Define System Modes & States
Allocated functions
Trade off Results & Parameters
Operational Need
Define system users/actors
Users/Actors
Functions
Exchanges dataflow
Modes & States, Configurations, Situations
Trade off Results & Parameters
Capabilities
Allocate functions to the system and to users
Define exchanged Items
Exchanged Items
Exchanges dataflow
Validate functional Analysis
System F&NF Need
System F&NF Need
Data
Exchanges dataflow
Define functional Scenarios
Scenarios
Allocated functions
Trade off Results & Parameters
Define System required Capabilities
Exchanges dataflow
Define Information and Data
Allocated Data
Operational Need
Capabilities
Design LOGICAL ARCHITECTURE
Exchanges dataflow
Capabilities
Define Functional Chains traversing the system
Functional Chains
System Requirements
Formalise and consolidate Requirements
Customer Requirements
Define System Requirements
Requirements
Formalise and link system requirements with functional Analysis
System F&NF Need
System F&NF Need
Functional Analysis
Check Requirements against need and early architecture
Early Architecture
Functional analysis
Define an early Architecture
Requirements

5.1.2.1 Perform a Capability Trade-off Analysis

Drive a multi-parametric analysis, in order to identify which parameters impact mainly the expected capabilities, in a wider scope than just system intrinsic performance; these parameters are selected based on operational capability definition above:

- Concepts & Doctrine,
- Organisation functioning,
- Infrastructures,
- Equipment (including system/SW),
- Whole System life cycle cost,
- Logistics, Deployment, Sustainability,
- Human Factors,

- Competencies, Training, …

(Known as "DOTMLPF" analysis (Doctrine, Organization, Training, Materiel, Leadership and Education, Personnel, and Facilities)).

Then choose the best trade-off between all the former parameters, to elect the best combination.

Functional and non-functional analysis of the system/SW will then be applied to this selected trade-off results, that will orient the analysis.

Some capability gaps may then appear (unreachable capacities or performance), that must affect system use, deployment, requirements, and definition; iterate with operational need analysis if needed.

Operational gap with product line and reused assets may also be identified here if needed.

Input:

- Operational Analysis outputs

Output:

- Results of multi-parametric analysis
- Description of the trade-off solution orientation regarding the different parameters considered.

Target documents:

- System/Segment Specification (SSS)
- Operational Concept Document (OCD) – for more detailed description if needed.

Verification and Consistency checks:

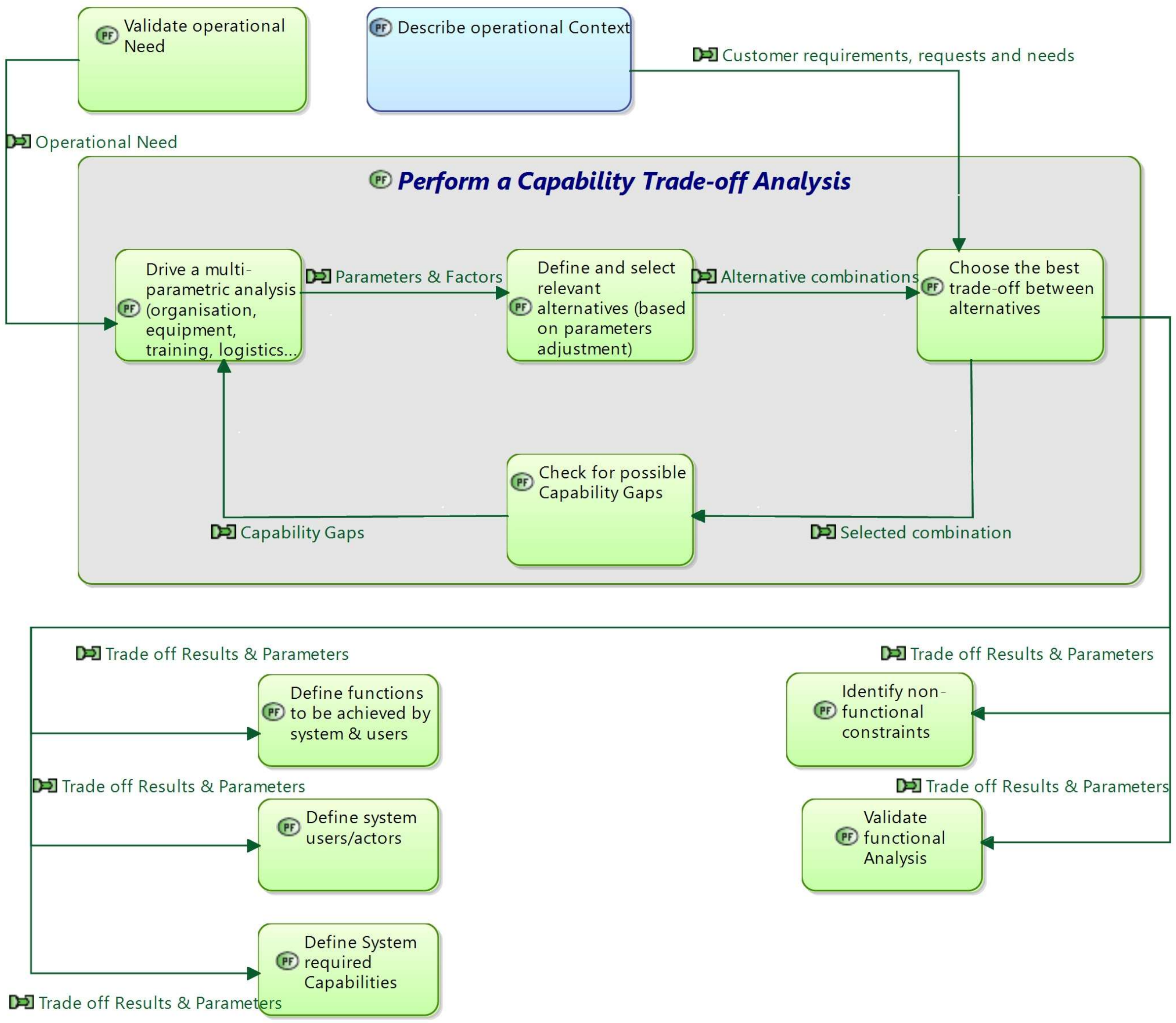
External consistency:

- Between Capabilities / operational analysis, and trade-off results

Internal consistency:

- Between candidate solutions, and between the elected solution and multi-parameters quantification.
- Verify the Capability Description: coherent, complete, relevant: no contradiction, no gap, no inaccuracy.

This figure describes the interactions of the considered task with other engineering activities.


Validate operational Need
Describe operational Context
Customer requirements, requests and needs
Operational Need
Perform a Capability Trade-off Analysis
Drive a multi-parametric analysis (organisation, equipment, training, logistics...
Parameters & Factors
Define and select relevant alternatives (based on parameters adjustment)
Alternative combinations
Choose the best trade-off between alternatives
Check for possible Capability Gaps
Capability Gaps
Selected combination
Trade off Results & Parameters
Define functions to be achieved by system & users
Trade off Results & Parameters
Identify non-functional constraints
Trade off Results & Parameters
Define system users/actors
Trade off Results & Parameters
Validate functional Analysis
Define System required Capabilities
Trade off Results & Parameters

5.1.2.1.1 Drive a multi-parametric analysis (organisation, equipment, training, logistics...)

in order to identify which parameters or factors impact mainly the expected capabilities, in a wider scope than just system intrinsic performance; these parameters or factors are selected based on operational capability definition above:

·Concepts & Doctrine,

·Organisation functioning,

·Infrastructures,

·Equipment (including system/SW),

·Whole System life cycle cost,

·Logistics, Deployment, Sustainability,

·Human Factors,

·Competencies, Training, …

(Known as "DOTMLPF" analysis (Doctrine, Organization, Training, Materiel, Leadership and Education, Personnel, and Facilities)).

5.1.2.1.2 Define and select relevant alternatives (based on parameters adjustment)

select the most likely and viable combinations of parameters or factors in order to achieve expected capabilities

5.1.2.1.3 Choose the best trade-off between alternatives

find the optimum between parameters to fulfil expected capabilities

5.1.2.1.4 Check for possible Capability Gaps

unreachable capacities or performance, that may affect Capability fulfilment, system use, deployment, requirements, and definition; iterate with operational need analysis if needed

5.1.2.2 Perform a functional and non-functional Analysis

Define System/SW required System Functions (functions of the system directly driven by the operational need), shared information and data exchanges/interfaces, to satisfy operational Need.

More precisely, from operational activities description and capability trade-off above plus customer requirements,

- Identify activities that should be supported by the system/SW and its users

- Identify functions required to satisfy and support all these activities
- Complement them with functions that could fill the capability gaps detected above
- Allocate these functions respectively to system/SW Vs users (incl. Human Factors)
- List and detail information, data flows, managed, exchanged and required by all these functions (internal or external to system); including required standards & interfaces
- Identify functional chains traversing the system/SW in order to support operational processes and capabilities (traversing functions & data flows)
- Identify system/SW modes & states, relate them to functions
- Allocate operational scenarios to system/SW and users, functional chains, modes & states, therefore defining system scenarios; enrich them if needed
- Create and maintain traceability links with operational analysis (e.g. functions wrt activities, functional chains wrt operational processes, system scenarios wrt operational scenarios).

Identify all major dimensioning needs, and [non-functional] constraints, relating them to the appropriate fonctions, functional chains, actors… and associated scenarios, and relate them to system scenarios: e.g.

- Identify non-functional constraints (performance, safety…) and relate them to concerned functions, functional chains…
- Identify industrial constraints not coming from customer/user: ability to produce, to test, to maintain, to sub-contract…
- When intending to reuse existing assets, check this functional/non-functional analysis against these assets for compatibility.
- Enrich system scenarios with non-functional & industrial constraints
- Identify and select main (non functional) viewpoints (concerns) (*) susceptible to impact the Architecture Definition & breakdown.
- Each viewpoint should emphasise a specific set of constraints or expected behaviour, quality, respect of non-functional properties… At least one viewpoint should be dedicated to Reuse and Product Policy.
- Try to order them in terms of importance, relative priority.

Ensure traceability/justification links between operational and functional/non-functional analyses, and check consistency/coherency between them.

If necessary, envisage retroaction on operational need analysis (e.g. to change actors role for safer behaviour…).

Input:

- Operational Analysis outputs
- Customer Requirements

Output:

- Functional & non-functional analysis result (System functional breakdown + dataflow, functional chains, non functional constraints, scenarios…)
- Traceability between Operational & System analyses
- List of relevant /critical viewpoints for the target system architecture

Target documents:

- System/Segment Specification (SSS)

Verification and Consistency checks:

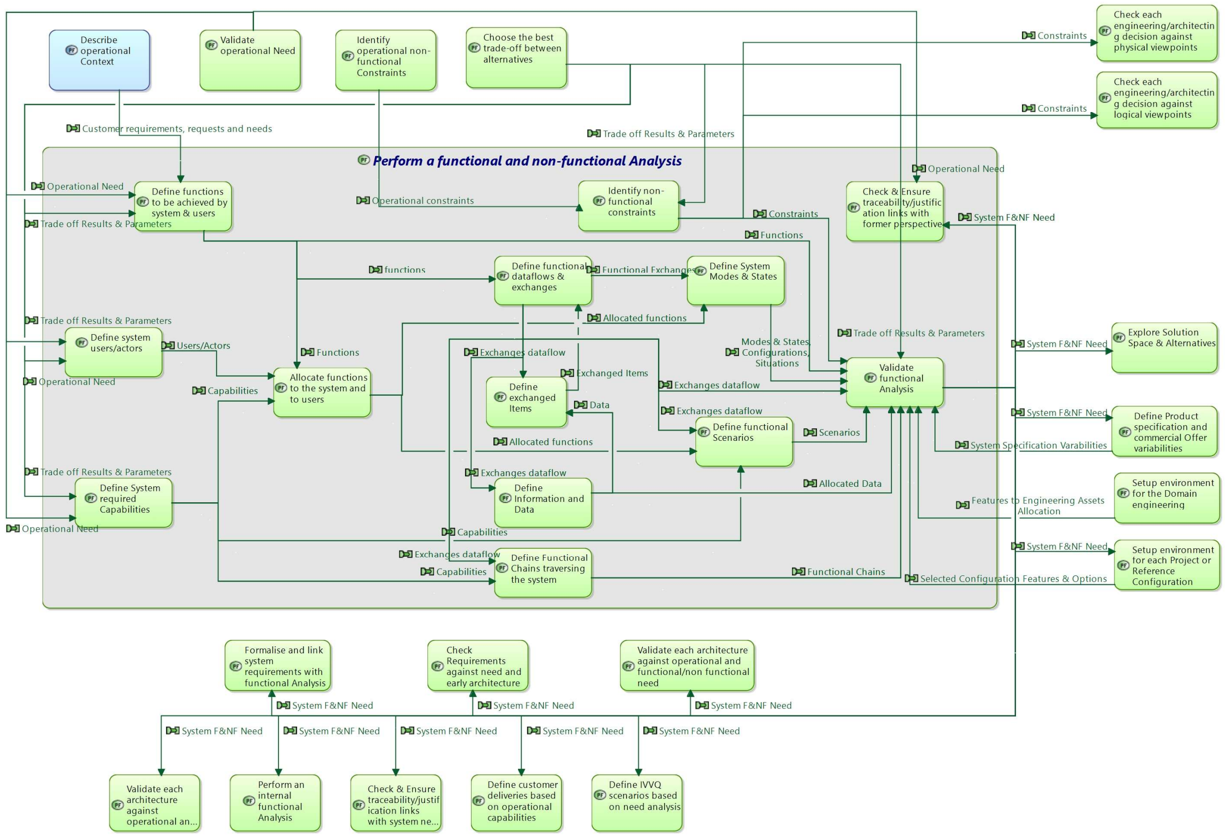
External consistency:

- Between Operational activities/data and System functions/data…

Internal consistency:

- Between all functional & non-functional elements
- Verify the functional/non-functional Need Description: coherent, complete, relevant: no contradiction, no gap, no inaccuracy.

This figure describes the interactions of the considered task with other engineering activities.


Describe operational Context
Validate operational Need
Identify operational non-functional Constraints
Choose the best trade-off between alternatives
Check each engineering/architecting decision against physical viewpoints
Check each engineering/architecting decision against logical viewpoints
Constraints
Constraints
Customer requirements, requests and needs
Trade off Results & Parameters
Perform a functional and non-functional Analysis
Operational Need
Define functions to be achieved by system & users
Trade off Results & Parameters
Operational constraints
Identify non-functional constraints
Constraints
Functions
Check & Ensure traceability/justification links with former perspective
Operational Need
System F&NF Need
functions
Define functional dataflows & exchanges
Functional Exchanges
Define System Modes & States
Allocated functions
Trade off Results & Parameters
Define system users/actors
Users/Actors
Functions
Exchanges dataflow
Exchanged Items
Modes & States, Configurations, Situations
Trade off Results & Parameters
Validate functional Analysis
System F&NF Need
Explore Solution Space & Alternatives
Operational Need
Capabilities
Allocate functions to the system and to users
Define exchanged Items
Data
Exchanges dataflow
Exchanges dataflow
Define functional Scenarios
Scenarios
System F&NF Need
Define Product specification and commercial Offer variabilities
System Specification Varabilities
Allocated functions
Trade off Results & Parameters
Exchanges dataflow
Define System required Capabilities
Define Information and Data
Allocated Data
Setup environment for the Domain engineering
Features to Engineering Assets Allocation
Operational Need
Capabilities
Exchanges dataflow
Define Functional Chains traversing the system
Capabilities
Functional Chains
System F&NF Need
Setup environment for each Project or Reference Configuration
Selected Configuration Features & Options
Formalise and link system requirements with functional Analysis
Check Requirements against need and early architecture
Validate each architecture against operational and functional/non functional need
System F&NF Need
System F&NF Need
System F&NF Need
System F&NF Need
System F&NF Need
System F&NF Need
System F&NF Need
System F&NF Need
Validate each architecture against operational an...
Perform an internal functional Analysis
Check & Ensure traceability/justification links with system ne...
Define customer deliveries based on operational capabilities
Define IVVQ scenarios based on need analysis

5.1.2.2.1 Define system users/actors

among operational Actors, those interacting with the system, including system users

5.1.2.2.2 Define System required Capabilities

based on operational capabilities

5.1.2.2.3 Define functions to be achieved by system & users

based on activities allocated to those actors that are users of the system, Identify functions required to satisfy and support all these activities;

Complement them with functions that could fill the capability gaps detected above

5.1.2.2.4 Allocate functions to the system and to users

considering different alternatives and possible contributions of the system, and workshare between the system and external actors.

including workshare between operators/users own work, human tasks assisted by the system, and fully automated tasks.

5.1.2.2.5 Check & Ensure traceability/justification links with former perspective

between functions and operational activities, functional chains and operational processes, data and information...

including checking consistency/coherency between them.

5.1.2.2.6 Define functional dataflows & exchanges

managed, exchanged and required by all functions (internal or external to system); including required standards & interfaces

5.1.2.2.7 Define exchanged Items

notably defining which sets of data (or information, or material…) are to be considered as a whole for this exchange (at the same time, coherently…).

5.1.2.2.8 Define Information and Data

managed, exchanged and required by all functions (internal or external to system); including those exchanged with users or external systems

5.1.2.2.9 Define Functional Chains traversing the system

in order to support operational processes and capabilities (succession of functions and functional exchanges traversing data flows);

in order to express non-functional constraints such as latency, criticity, etc

5.1.2.2.10 Define functional Scenarios

Allocate operational scenarios to system and users, functional chains, modes & states, therefore defining system scenarios; enrich them if needed

5.1.2.2.11 Define System Modes & States

including functions available in each mode or state

5.1.2.2.12 Identify non-functional constraints

all major dimensioning needs, and [non-functional] constraints, relating them to the appropriate fonctions, functional chains, actors… and associated scenarios, and relate them to system scenarios: e.g.

·Identify non-functional constraints (performance, safety…) and relate them to concerned functions, functional chains…

·Identify industrial constraints not coming from customer/user: ability to produce, to test, to maintain, to sub-contract…

·When intending to reuse existing assets, check this functional/non-functional analysis against these assets for compatibility.

·Enrich system scenarios with non-functional & industrial constraints

·Identify and select main (non functional) viewpoints (concerns) (*) susceptible to impact the Architecture Definition & breakdown.

Each viewpoint should emphasise a specific set of constraints or expected behaviour, quality, respect of non-functional properties… At least one viewpoint should be dedicated to Reuse and Product Policy.

·Try to order them in terms of importance, relative priority.

5.1.2.2.13 Validate functional Analysis

against all former elements.

5.1.2.3 Formalise and consolidate Requirements

Define system/SW requirements

Define Requirements to implement the former functions, data exchanges, non-functional constraints… and complement customer-originated requirements.

Maintain bi-directional traceability between Requirements and system/SW Need functions, data flows, interfaces, scenarios…

When Reuse is expected, compare and map requirements with existing components to be reused.

Define an early architecture

Build an early architectural View of the System/SW, based on previous capability engineering choices & results,

- Focusing on main constraints impacting design & IVVQ (performance, critical parts, dynamic behaviour, real-time issues, system modes & states, development & ownership cost… and reuse of existing assets);
  Note: restrict early architecture to most significant and risky aspects and parts of the system/SW
- Allocating system/SW Need functions, data flows …to components of this architecture
- Dealing with first non-functional requirements (Quality of Service, industrial constraints, subcontracting, modularity, Product Line approach, design to cost…)
- In conformity with operational Need.

The approach to build this early architecture is the same as logical/physical architecture design described later in this document, and should not be restricted to a functional breakdown.

Check (internal) Requirements against early architecture and need analysis.

This should at least lead to evaluate, for each requirement:

- the importance of its contribution to operational need
  by following links from requirement to functions implementing it, then links from functions towards operational activities
- its feasibility (against early architecture; see above)
  by following links from requirement to functions implementing it, then from functions to components of architecture,
  and consideration of non-functional constraints and viewpoints
- its qualitative cost range (through complexity to map on architecture, integration issues, complexity of validation scenarios, of preselected technologies when significant…).

When a particular requirement is not achievable (cost, feasability, …), return to the initial operational need in order to see if the requirement can be relaxed.

Note that requirements analysis may lead to modify/improve early architecture; on the other side, requirements refinement should stop when not relevant to (not impacting) early architecture.

Input:

- System/SW functional & non-functional analysis outputs
- Customer requirements

Output:

- System Requirements formalizing System definition,
- Consolidated early Architecture
- Allocation of System functions to architecture components
- Traceability links between requirements, system functional/non-functional analysis and early architecture

Target document:

- System/Segment Specification (SSS)

Verification and Consistency checks:

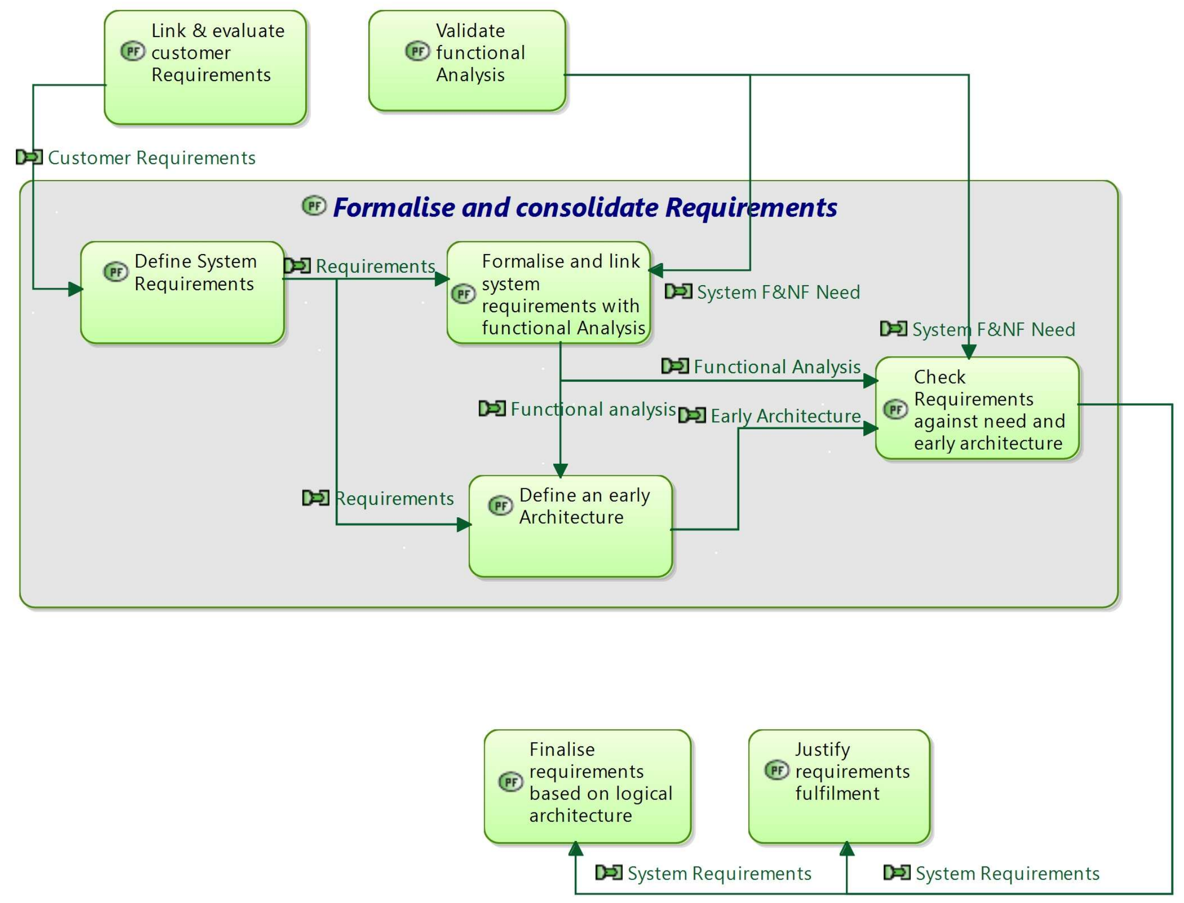
External consistency:

- Between System requirements and User Requirements
- Between system requirements and functional/non-functional analysis

Internal consistency:

- Between system requirements and early architecture
- Verify the Requirements Description: coherent, complete, relevant: no contradiction, no gap, no inaccuracy.

This figure describes the interactions of the considered task with other engineering activities.


Link & evaluate customer Requirements
Validate functional Analysis
Customer Requirements
Formalise and consolidate Requirements
Define System Requirements
Requirements
Formalise and link system requirements with functional Analysis
System F&NF Need
System F&NF Need
Functional Analysis
Check Requirements against need and early architecture
Functional analysis
Early Architecture
Requirements
Define an early Architecture
Finalise requirements based on logical architecture
Justify requirements fulfilment
System Requirements
System Requirements

5.1.2.3.1 Define System Requirements

Based on customer requirements and operational/functional analyses

Define Requirements to implement the former functions, data exchanges, non-functional constraints… and complement customer-originated requirements.

5.1.2.3.2 Formalise and link system requirements with functional Analysis

Maintain bi-directional traceability between Requirements and system/SW Need functions, data flows, interfaces, scenarios…

5.1.2.3.3 Define an early Architecture

Build an early architectural View of the System/SW, based on previous capability engineering choices & results,

·Focusing on main constraints impacting design & IVVQ (performance, critical parts, dynamic behaviour, real-time issues, system modes & states, development & ownership cost… and reuse of existing assets);

Note: restrict early architecture to most significant and risky aspects and parts of the system/SW

·Allocating system/SW Need functions, data flows …to components of this architecture

·Dealing with first non-functional requirements (Quality of Service, industrial constraints, subcontracting, modularity, Product Line approach, design to cost…)

·In conformity with operational Need.

The approach to build this early architecture is the same as logical/physical architecture design described later in this document, and should not be restricted to a functional breakdown.

Please refer to logical and physical architecture steps

5.1.2.3.4 Check Requirements against need and early architecture

to evaluate, for each requirement:

·the importance of its contribution to operational need

by following links from requirement to functions implementing it, then links from functions towards operational activities

·its feasibility (against early architecture; see above)

by following links from requirement to functions implementing it, then from functions to components of architecture,

and consideration of non-functional constraints and viewpoints

·its qualitative cost range (through complexity to map on architecture, integration issues, complexity of validation scenarios, of preselected technologies when significant…).

5.2 DESIGN THE SOLUTION ARCHITECTURE

Analyse elements shaping the definition of the solution down to a reference architecture

See sub-tasks description

5.2.1 Explore Solution Space & Alternatives

<Data elaboration description>

Most part of the work is done using and producing **Free Description Documents** and possibly complementary **Textual Requirements**, along with other kinds of more or less formalised orientation and decision making support material.

However, some first elements of need and solution sketching may be produced, such as **Operational Mission Analysis**, **System Need Specification**, **Designed Solution Architecture**, especially to outline and compare different notional, high level alternatives, as preliminary logical architecture candidates. See 'Evaluate & Verify Architecture Choices' for more details.

<Data elaboration description end>

5.2.2 Design LOGICAL ARCHITECTURE

This perspective aims at building a notional component breakdown of the system at a coarse-grain level. Based on solution-oriented functional and non-functional analysis describing the designed behavior (functions, interfaces, capabilities, functional chains & scenarios, modes & states…), build one or several decompositions of the system into logical components. Its limited complexity level helps in exploring the solution alternatives.

All major (non-functional) constraints (safety, security, performance, IVV, cost, non-technical, Etc.) are taken into account and compared to each other so as to find the best trade-off. This approach is viewpoint-driven, where viewpoints formalize the way these constraints impact the system architecture.

Outputs of this engineering activity consist of the selected logical architecture which is described by components and justified interfaces definition, functional behavior, scenarios, modes and states, formalization of all viewpoints and the way they are taken into account in the components design. Since the architecture has to be validated against the need analysis, links with requirements and operational scenarios are also to be produced.

See **<Data elaboration description>** below

Engineering goals

- Build a coarse-grained breakdown of the system/software in components,
 Convenient to structure further engineering and development while managing system/software complexity,
 Near optimum Compromise between all major Requirements, Stakes &

Constraints(including non-functional, industrial, performance…),
Therefore not likely to be challenged later in the development.

- Early define and validate properties of the architecture and the system/software with regards to non functional constraints
- Iterate on previous early architecture and consolidate its definition
- Get technical Material to support Negotiation by evaluating operational added value of each requirement

Tasks to be completed during this step

- Define Architecture Drivers and Viewpoints design Rules
- Define notional functional and non-functional Behavior
- Build candidate architectural breakdowns in Components
- Select best Compromise Architecture

Stop Criteria

This activity is achieved when an architecture can be considered – and proven - as the best compromise according to multi-viewpoint analysis, and it integrates all major constraints allocated to the System/SW at this level.

This activity should not reach a level of detail dealing with technical/technological constraints or choices from lower engineering levels, unless they affected or challenged the considered breakdown and viewpoints reconciliation.

Contributors & Competencies

Major competencies required to complete this step are suggested below:

Required Competencies	**Major Expected Contribution**	**Possible Contributors (*)**
Domain knowledge		• Chief architect • Systems engineering

		manager • Functional analyst • Specialty engineering expert • System engineer • Product line manager • Simulation expert • Program manager • Others
Product & Technical Domain (incl. Product line)	Product line constraints, especially reuse	
Systems engineering - Design		
Functional & non functional Analysis	System Analysis refinement/complement	
Architecture definition (logical, physical)	Architectural design & compromise	
Modeling & viewpoint engineering techniques	Modeling techniques, viewpoint building	
Value Engineering	Cost effectiveness of solutions	
Systems engineering - Specialties		
Technical Simulation	validity of component behaviour	
Specialties engineering (safety, perf, RAMST…)	Adequacy of Architecture to specialty	

(*) Depending on each organisation, competencies may be allocated to different actors; the following contributors list is therefore just an example to be adapted to each organisation:

<Data elaboration description>

The first decisions shaping architecture start from **System Need Specification** and **Textual Requirements** : a notional designed behaviour is elaborated, being described by means of high-level **Design Functions**, **Design Functional exchanges** etc., that should implement and fulfil **Specified Functions** and **Spec. Functional Exchanges**.

The structuration of the system into notional logical **Components performing Functions** is performed in relation with the functional analysis above. Components group or segregate **Design Functions** according to architectural drivers.

This preliminary, conceptual architecture is completed and verified by defining the way **Specified Capabilities** are implemented in each notional architecture candidate **Designed Solution Capabilities**.

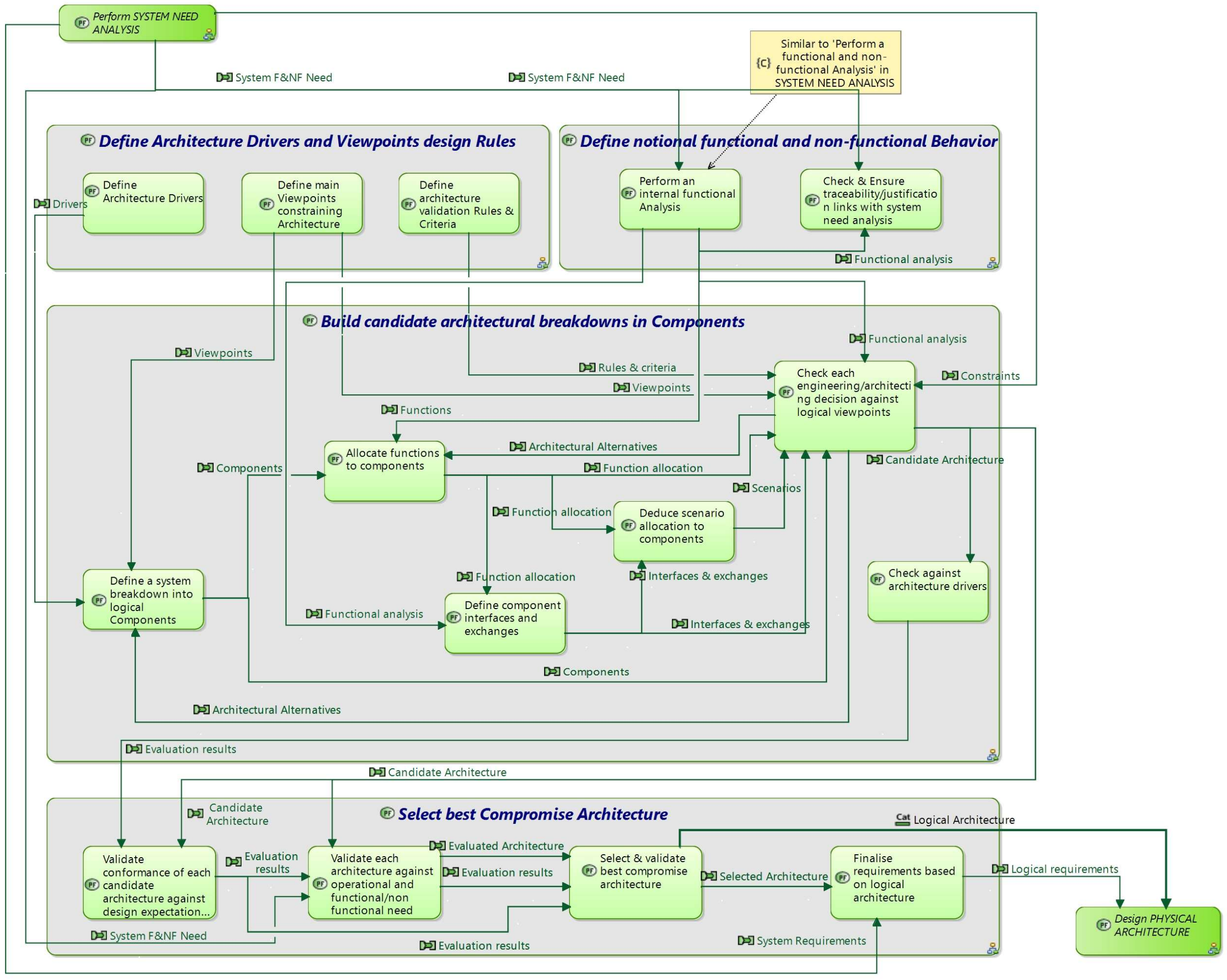
For this purpose, logical **Design Functional Chains** and **Design Scenarios** are defined, inspired by **Spec. Functional Chains** and **Specified Scenarios**. They are applied to **Design Functions** and **Components performing Functions**, and contribute to the global coherency check of each candidate architecture.

<Data elaboration description end>

This figure provides an inner view of the task Design LOGICAL ARCHITECTURE, in the context of Arcadia core perspectives.

For each of the first level tasks presented here, only links with other core perspectives are displayed.

For a complete view of links with other engineering tasks and activities, see the figure dedicated to this first level task.


Perform SYSTEM NEED ANALYSIS
System F&NF Need
System F&NF Need
Similar to 'Perform a functional and non-functional Analysis' in SYSTEM NEED ANALYSIS
Define Architecture Drivers and Viewpoints design Rules
Define Architecture Drivers
Define main Viewpoints constraining Architecture
Define architecture validation Rules & Criteria
Drivers
Define notional functional and non-functional Behavior
Perform an internal functional Analysis
Check & Ensure traceability/justification links with system need analysis
Functional analysis
Build candidate architectural breakdowns in Components
Functional analysis
Viewpoints
Rules & criteria
Viewpoints
Check each engineering/architecting decision against logical viewpoints
Constraints
Functions
Architectural Alternatives
Allocate functions to components
Components
Function allocation
Candidate Architecture
Scenarios
Function allocation
Deduce scenario allocation to components
Function allocation
Interfaces & exchanges
Check against architecture drivers
Define a system breakdown into logical Components
Functional analysis
Define component interfaces and exchanges
Interfaces & exchanges
Components
Architectural Alternatives
Evaluation results
Candidate Architecture
Candidate Architecture
Select best Compromise Architecture
Logical Architecture
Evaluated Architecture
Validate conformance of each candidate architecture against design expectation...
Evaluation results
Validate each architecture against operational and functional/non functional need
Evaluation results
Select & validate best compromise architecture
Selected Architecture
Finalise requirements based on logical architecture
Logical requirements
Design PHYSICAL ARCHITECTURE
System F&NF Need
Evaluation results
System Requirements

5.2.2.1 Define Architecture Drivers and Viewpoints design Rules

Define architecture drivers

Architecture drivers are major Stakes & Properties that architecture should favour, depending on the domain and product policy.

e.g. ease of evolution, real-time constraints, ease of separate development & sub-contracting, scalability, portability, certification, 24x7 availability...

These are design priorities that will orient and constrain architecture definition, when having to make choices among various possibilities, in order to ease and secure development and / or system behaviour.

As an example, favouring real time constraints may hinder modularity, or loose coupling between components; portability may prevent from using advanced features of the underlying platform…

Define main Viewpoint Rules & Criteria

Define and associate to each viewpoint, viewpoint design rules (constitution and checking rules) in order to express how to build, how to test architecture against each viewpoint.

Define also criteria to confront and reconcile all viewpoints (at least, priority between viewpoints).

Each architecture design decision should further be checked against architecture drivers compliance.

Each design and development choice impacting these drivers should also be justified and checked (e.g. middleware technology threatening modularity or performance…)

Input:

- output of operational and system/SW need analyses
- List of predefined viewpoints rules to analyse the architecture

Output:

- Architecture drivers to be applied to system architecture
- Checklists to confront design choices to architecture drivers
- Viewpoint-analysis rules

Target documents:

- System/Segment Design Document (SSDD)

Verification and Consistency checks:

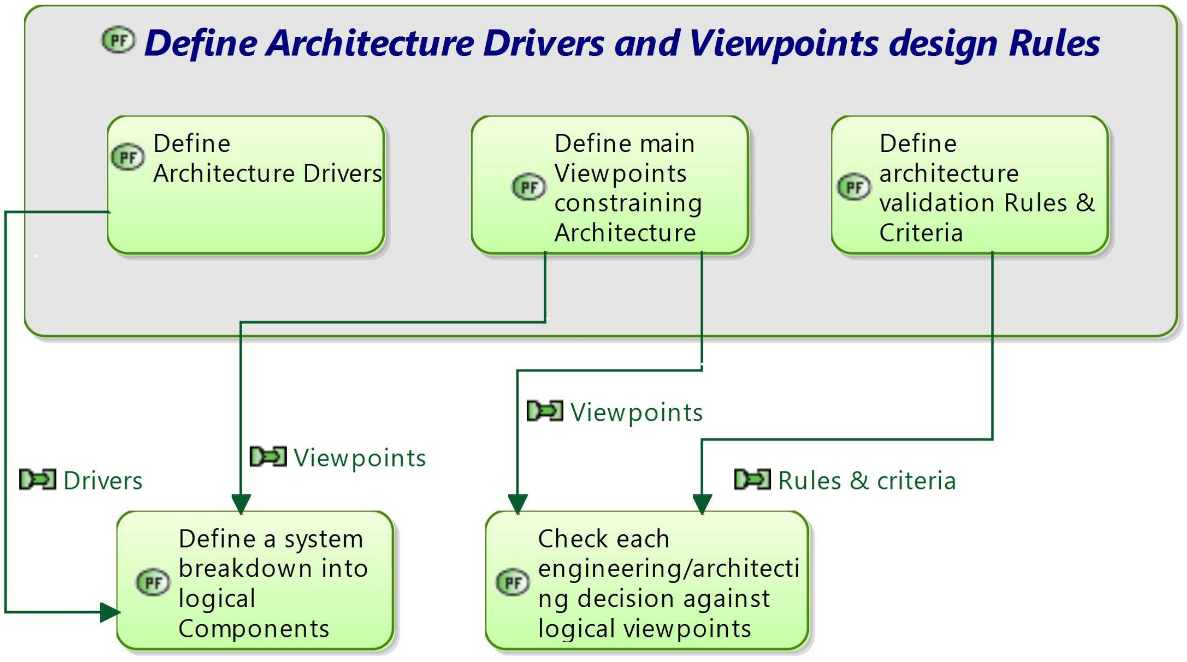
External consistency:

- Between User Requirements, industrial constraints (reuse, product line…) and Architecture Drivers

Internal consistency:

- Between Architecture drivers and selected Viewpoints & rules

This figure describes the interactions of the considered task with other engineering activities.


Define Architecture Drivers and Viewpoints design Rules
Define Architecture Drivers
Define main Viewpoints constraining Architecture
Define architecture validation Rules & Criteria
Drivers
Viewpoints
Viewpoints
Rules & criteria
Define a system breakdown into logical Components
Check each engineering/architecting decision against logical viewpoints

5.2.2.1.1 Define Architecture Drivers

major Stakes & Properties that architecture should favour, depending on the domain and product policy.

e.g. ease of evolution, real-time constraints, ease of separate development & sub-contracting, scalability, portability, certification, 24x7 availability...

These are design priorities that will orient and constrain architecture definition, when having to make choices among various possibilities, in order to ease and secure development and / or system behaviour.

As an example, favouring real time constraints may hinder modularity, or loose coupling between components; portability may prevent from using advanced features of the underlying platform…

5.2.2.1.2 Define main Viewpoints constraining Architecture

main (non functional) concerns susceptible to impact the architecture breakdown (e.g. performance, safety, interface management, product line & reuse, cost).

Priority and relative importance of these viewpoints with respect to eachother is also to be defined.

5.2.2.1.3 Define architecture validation Rules & Criteria

know-how to be used in order to help the solution emerge :

·design rules for each viewpoint (e.g. how to group functions, how to minimize interfaces complexity...): constitution and checking rules to test architecture against each viewpoint.

·compromise criteria, to reconcile viewpoints (e.g. should performance issues be prioritary as compared to reuse issues)

·verification criteria to check that the final compromise architecture meets all requirements, operational and industrial needs.

5.2.2.2 Define notional functional and non-functional Behavior

This task is similar to System Need Analysis 'Define notional functional and non-functional Behavior'.

Define a functional behaviour that should fulfil former functional analysis, addressing:

- design & description of solution behaviour instead of need expression
- first design decisions reagrding behaviour

Build and maintain justification and traceability links with System Need Analysis functions, functional chains, scenarios, modes &states, data etc.

More precisely,

- Identify functions required to satisfy and implement all system need analysis functions
- Complement them with necessary functions that were not identified in need analysis
- List and detail information, data flows, managed, exchanged and required by all these functions (internal or external to system); including required standards & interfaces

- Identify functional chains traversing the system/SW in order to implement need defined functional chains (traversing functions & data flows); similarly define functional scenarios implementing those defined at need level;
  enrich them if needed in order to appropriately define and check solution behaviour
- Identify system/SW modes & states, relate them to functions; enrich them if needed
- Create and maintain traceability links with system need analysis (e.g. between functions, between functional chains, between scenarios).

Identify all major dimensioning needs, and [non-functional] constraints, relating them to the appropriate fonctions, functional chains, actors… and associated scenarios, and relate them to system scenarios: e.g.

- Identify non-functional constraints (performance, safety…) and relate them to concerned functions, functional chains…
- Identify industrial constraints not coming from customer/user: ability to produce, to test, to maintain, to sub-contract…
- When intending to reuse existing assets, check this functional/non-functional analysis against these assets for compatibility.
- Enrich system scenarios with non-functional & industrial constraints
- Identify and select main (non functional) viewpoints (concerns) (*) susceptible to impact the functional analysis.
  Each viewpoint should emphasise a specific set of constraints or expected behaviour, quality, respect of non-functional properties… At least one viewpoint should be dedicated to Reuse and Product Policy.
- Try to order them in terms of importance, relative priority.

Ensure traceability/justification links between system need and notional and functional/non-functional analyses, and check consistency/coherency between them.

Input:

- System Need Analysis outputs
- Customer Requirements

Output:

- Functional & non-functional analysis result (System functional breakdown + dataflow, functional chains, non functional constraints, scenarios…)
- Traceability between notional & System Need analyses
- List of relevant /critical viewpoints for the target system architecture

Target documents:

- System/Segment Design Document (SSDD) (preliminay)

Verification and Consistency checks:

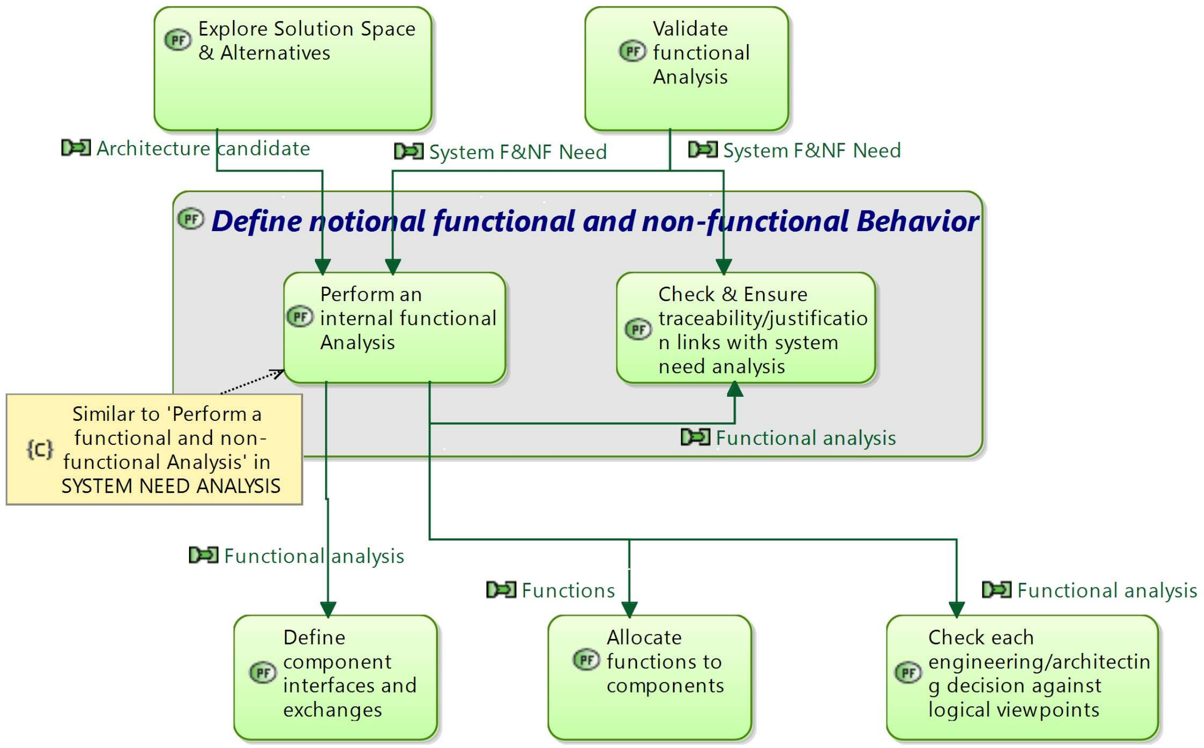
External consistency:

- Between System Need and notional functional Analysis functions/data…

Internal consistency:

- Between all functional & non-functional elements
- Verify the functional/non-functional Behaviour Description: coherent, complete, relevant: no contradiction, no gap, no inaccuracy.

This figure describes the interactions of the considered task with other engineering activities.


Explore Solution Space & Alternatives
Validate functional Analysis
Architecture candidate
System F&NF Need
System F&NF Need
Define notional functional and non-functional Behavior
Perform an internal functional Analysis
Check & Ensure traceability/justification links with system need analysis
Similar to 'Perform a functional and non-functional Analysis' in SYSTEM NEED ANALYSIS
Functional analysis
Functional analysis
Functions
Functional analysis
Define component interfaces and exchanges
Allocate functions to components
Check each engineering/architecting decision against logical viewpoints

5.2.2.2.1 Perform an internal functional Analysis

Detail and re-factor external functional analysis (esp. functions) addressing

·greater level of detail resolving ambiguities of definition

·and design decisions choosing among various implementation options.

Build and maintain justification and traceability links with external analysis functions.

*** Please refer to System Need Analysis / Perform a functional & non-functional analysis ***

5.2.2.2.2 Check & Ensure traceability/justification links with system need analysis

between logical architecture functions and system need functions, functional chains, data and information, scenarios...

including checking consistency/coherency between them.

5.2.3 Build candidate architectural breakdowns in Components

Starting from previous functional & non-functional analysis results (functions, interfaces, data flows, behaviours…), build one or several decompositions of the system/software into logical components[1]. This may lead to considering several alternatives (functional, structural or both) that should be evaluated and compared to others.

Logical components will later tend to be the basic decomposition for development/sub-contracting, integration, reuse, product and configuration management item definitions (but other criteria will be taken into account to define the boundaries for these items).

The component building process consists in

- Grouping functions together in a consistent way (see viewpoints below) by allocating them to components
- While "inheriting" component interfaces and exchange data flows, from functional data flows between functions
- And deducing requirement and system scenarios allocation to components, based on functions allocation & traceability links
- Ensuring traceability and justification links with former steps (functions, components, scenarios…).

This building process has to deal with each Viewpoint & associated design Rules, either by

- Grouping functions close to each other in the considered viewpoint (e.g. dealing with the same operational activity, having same hard real-time constraints, sharing complex interfaces…)
- Or segregating / separating functions that must not be grouped (e.g. functions of different criticality/certification levels, functions heavily consuming platform resources…)
- Or mixed.

Yet all viewpoints will not suggest a given breakdown in components:

- functional consistency, modularity, interfaces confinement, resource consumption, safety/dependability, … favour and allow components outlining in the viewpoint scope: each of them may suggest a breakdown in components from its own constraints
- maintainability, cost management, human factors, time-critical Paths, system modes & states…, are more likely to influence other viewpoints components outlining, rather than allowing their own viewpoint components definition.

Therefore, all these viewpoints need to be confronted and reconciled with each others: this can be initiated in this task, in order to reduce the number of candidate architectures, but must anyway be formalised and completed in the next engineering task below.

Note that in some cases (e.g. performance viewpoint with limited resources), limited parts of the logical architecture may have to be described as an early physical architecture in order to validate against viewpoints rules (e.g. performance issues according to available computing power hypotheses).

During this component building process, the former system need functional analysis may have to be reworked: among others,

- To detail / refine some functions in order to fit components breakdown (e.g. split one function into processing, user interaction, data management; or to implement redundant functional paths)
- To optimise design by defining common use functions, generic ones
- to deal with non-functional viewpoints constraints (e.g. rearrange functions for a more secure behaviour, or check performance constraints against the use of security methods such as cryptography)
- To add functions necessary for design description (e.g. technical services, communication support, monitoring, download…).

[1] Note that the word 'Component' should be understood in a general manner, as a constituent of the system/SW at this level; it will later turn to either a sub-system, an equipment, a piece of software, a hardware board or function…

Input:

- Selection of relevant/critical viewpoints to describe the architecture
- system/SW functional and non-functional analysis

Output:

- Logical Architecture candidates including components functional contents, interfaces and dataflows, allocated non-functional constraints

Target documents:

- System/Segment Design Document (SSDD) (preliminary)

Verification and Consistency checks:

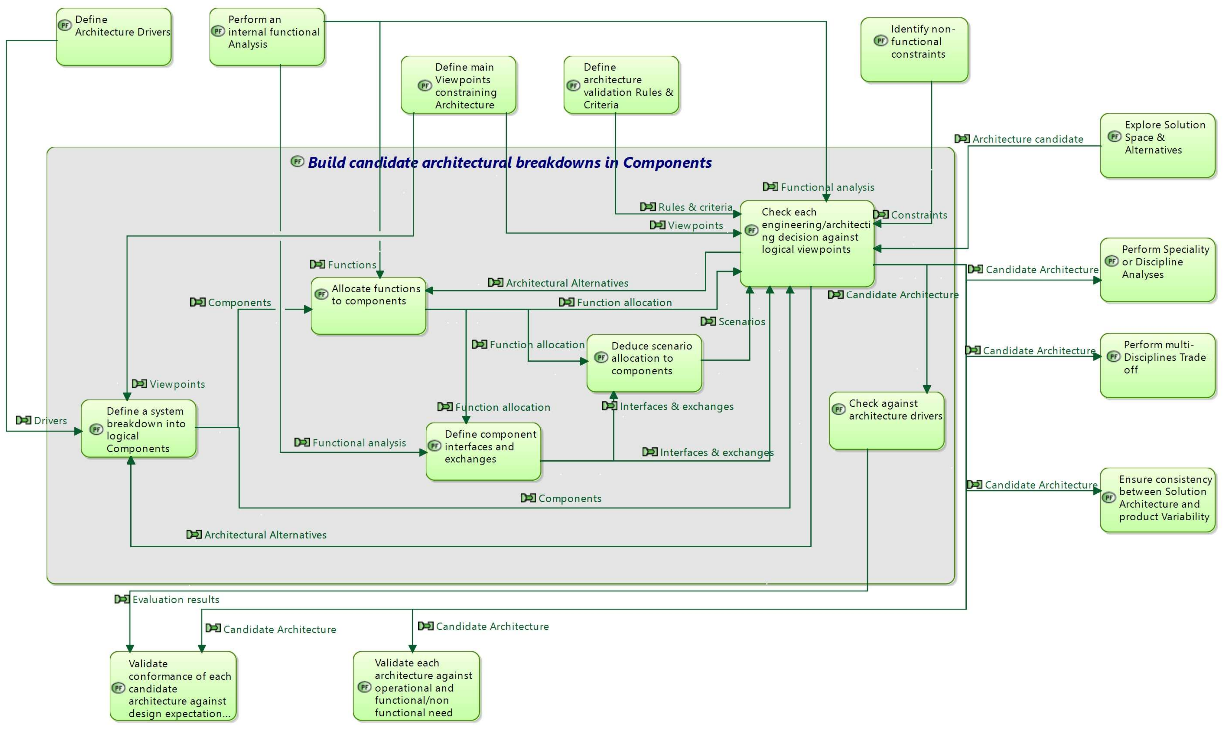
External consistency:

- Between User Requirements and system/SW need analysis, industrial constraints (reuse, product line…) and selected viewpoints

Internal consistency:

- Between Architecture drivers, viewpoints and Logical components and/or interfaces
- Between functions to components allocation, and architecture drivers/viewpoints rules
- Verify each logical architecture Description: coherent, complete, relevant: no contradiction, no gap, no inaccuracy.

This figure describes the interactions of the considered task with other engineering activities.


Define Architecture Drivers
Perform an internal functional Analysis
Define main Viewpoints constraining Architecture
Define architecture validation Rules & Criteria
Identify non-functional constraints
Explore Solution Space & Alternatives
Architecture candidate
Build candidate architectural breakdowns in Components
Functional analysis
Rules & criteria
Viewpoints
Constraints
Check each engineering/architecting decision against logical viewpoints
Perform Speciality or Discipline Analyses
Candidate Architecture
Functions
Allocate functions to components
Architectural Alternatives
Function allocation
Components
Scenarios
Deduce scenario allocation to components
Perform multi-Disciplines Trade-off
Viewpoints
Drivers
Define a system breakdown into logical Components
Interfaces & exchanges
Check against architecture drivers
Functional analysis
Define component interfaces and exchanges
Ensure consistency between Solution Architecture and product Variability
Evaluation results
Validate conformance of each candidate architecture against design expectation...
Validate each architecture against operational and functional/non functional need

5.2.2.3.1 Define a system breakdown into logical Components

Starting from previous functional & non-functional analysis results, build one or several decompositions of the system/software into logical components .

Logical components will later tend to be the basic decomposition for development/sub-contracting, integration, reuse, product and configuration management item definitions (but other criteria will be taken into account to define the boundaries for these items).

Initial beakdown is usually initiated by considering how functions should be either groued or segregated according to functional consistency and viewpoints ocnstraints.

5.2.2.3.2 Allocate functions to components

This building process has to deal with each Viewpoint & associated design Rules, either by

·Grouping functions close to each other in the considered viewpoint (e.g. dealing with the same operational activity, having same hard real-time constraints, sharing complex interfaces…)

·Or segregating / separating functions that must not be grouped (e.g. functions of different criticality/certification levels, functions heavily consuming platform resources…)

·Or mixed.

5.2.2.3.3 Define component interfaces and exchanges

based on allocated functions and deduced from functional exchanges between them;

functional exchanges should here be allocated to component exchanges.

5.2.2.3.4 Deduce scenario allocation to components

based on functions allocation & traceability links, create interaction scenarios between components and with external actors

5.2.2.3.5 Check each engineering/architecting decision against logical viewpoints

.check how much this architecture satisfies or infringes each viewpoint design rules, and expected non-functional properties

.analyse the impact of this architecture on allr key viewpoints (e.g. safety, performance), to detect discrepancies and "distortions" between viewpoints

.correct and iterate as needed.

5.2.2.3.6 Check against architecture drivers

verifying that architecture drivers are applied and that design rules are satisfied.

5.2.2.4 Select best Compromise Architecture

This tasks consists in finding and justifying the best compromise between constraints driven by each viewpoint, in a process called "Viewpoints weaving". This task is to be performed on each architectural alternative.

Evaluate candidate architecture(s) against each selected main viewpoint, in order to check how much this architecture impacts each viewpoint, satisfies or infringes the viewpoint design rules, and expected non-functional properties. Validate & justify it by its impact on each Viewpoint.

A general approach (to be adapted to each domain) might be:

1. select most important viewpoint to structure the system/sw (e.g. functional grouping, or safety level…)
2. check this grouping against most important architecture drivers in order to detect inconsistencies
3. analyse the impact of this grouping on other key viewpoints (e.g. safety, performance), to detect discrepancies and "distortions" between viewpoints
4. correct and iterate as needed.

It is recommended to preliminary define reconciliation criteria & rules between viewpoints, depending on each domain (e.g. "first define software partitions according to DO178B safety levels, then privilege functional grouping, then confront with time-critical chains; in case of conflict, privilege the critical chains").

Validate architecture against operational and system/SW need :

- how it supports operational activities,
- how it deals with functional behaviour (functional contents of each component, contribution to functional chains), interfaces, data flows & data models…,

- how it satisfies expected properties & constraints,
- how it implements/responds to operational scenarios & capabilities.

Finalise requirements based on this logical architecture.

Input:

- Logical Architecture candidates

Output:

- Logical Architecture description through viewpoints & reconciliation views
- Consolidated Requirements
- Issues and decisions (justifications) statement

Target documents:

- System/Segment Design Document (SSDD) (preliminary)

Verification and Consistency checks:

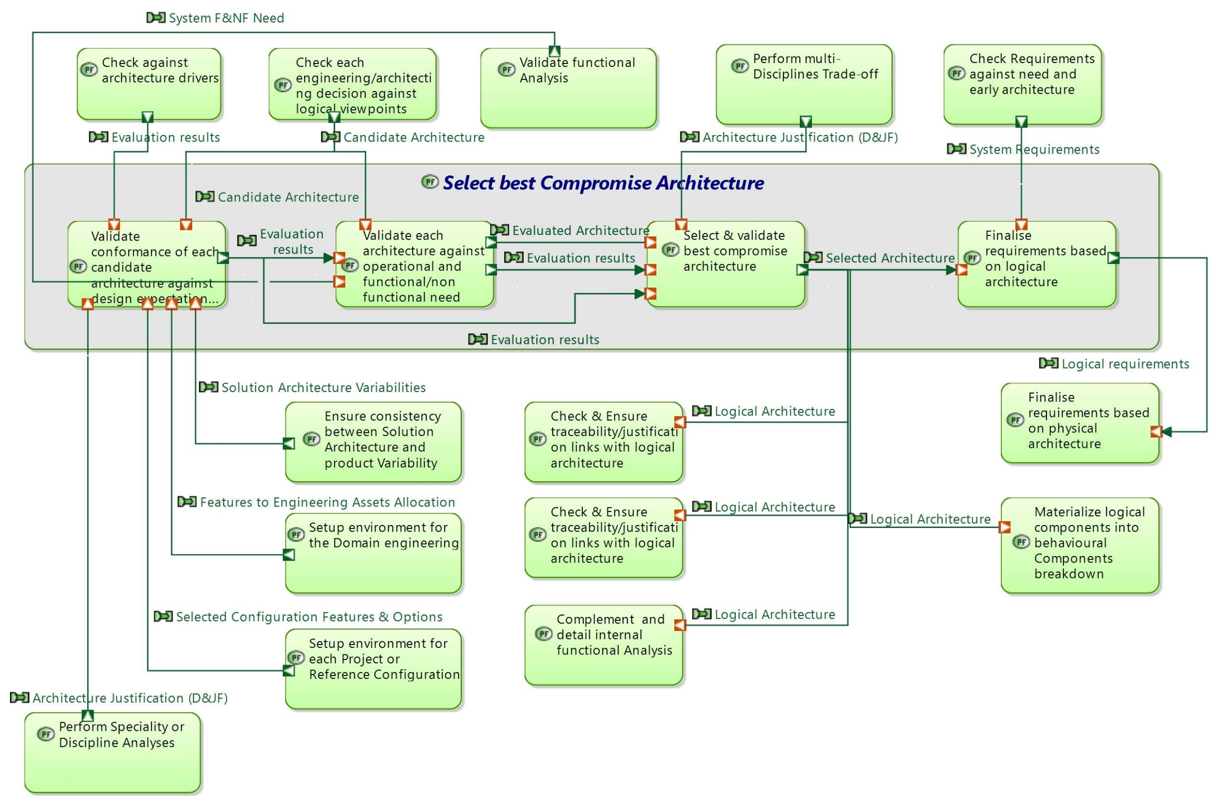
External consistency:

- Between User Requirements and system/SW need analysis, industrial constraints (reuse, product line…) and final logical architecture

Internal consistency:

- Between Architecture drivers and functional behaviour, Logical components and/or interfaces; between these and viewpoints and final logical architecture
- Verify the logical architecture Description: coherent, complete, relevant: no contradiction, no gap, no inaccuracy.

This figure describes the interactions of the considered task with other engineering activities.


System F&NF Need
Check against architecture drivers
Check each engineering/architecting decision against logical viewpoints
Validate functional Analysis
Perform multi-Disciplines Trade-off
Check Requirements against need and early architecture
Evaluation results
Candidate Architecture
Architecture Justification (D&JF)
System Requirements
Select best Compromise Architecture
Candidate Architecture
Validate conformance of each candidate architecture against design expectation...
Evaluation results
Validate each architecture against operational and functional/non functional need
Evaluated Architecture
Evaluation results
Select & validate best compromise architecture
Selected Architecture
Finalise requirements based on logical architecture
Evaluation results
Logical requirements
Solution Architecture Variabilities
Ensure consistency between Solution Architecture and product Variability
Check & Ensure traceability/justification on links with logical architecture
Logical Architecture
Finalise requirements based on physical architecture
Features to Engineering Assets Allocation
Setup environment for the Domain engineering
Check & Ensure traceability/justification on links with logical architecture
Logical Architecture
Logical Architecture
Materialize logical components into behavioural Components breakdown
Selected Configuration Features & Options
Setup environment for each Project or Reference Configuration
Complement and detail internal functional Analysis
Logical Architecture
Architecture Justification (D&JF)
Perform Speciality or Discipline Analyses

5.2.2.4.1 Validate conformance of each candidate architecture against design expectations incl. viewpoints

A general approach (to be adapted to each domain) might be:

1.select most important viewpoint to structure the system (e.g. performance, or safety level…)

2.check how much this architecture satisfies or infringes the viewpoint design rules, and expected non-functional properties

3.analyse the impact of this architecture on other key viewpoints (e.g. safety, performance), to detect discrepancies and "distortions" between viewpoints

4.correct and iterate as needed.

5.2.2.4.2 Validate each architecture against operational and functional/non functional need

·how it supports operational activities,

·how it deals with functional behaviour (functional contents of each component, contribution to functional chains), interfaces, data flows & data models…,

·how it satisfies expected properties & constraints,

·how it implements/responds to operational scenarios & capabilities.

5.2.2.4.3 Select & validate best compromise architecture

based on how each architecture fulfils need and main viewpoints analysis results

5.2.2.4.4 Finalise requirements based on logical architecture

based on requirement - function links, and functions to component allocation, allocate requirements to components and check that architecture fulfils requirements.

Complement requirements as needed.

5.2.3 Design PHYSICAL ARCHITECTURE

This perspective defines the "final" detailed architecture of the system at this level of engineering, ready to be developed according to implementation, technical and technological constraints and choices. The tradeoff around resources (e.g. power, communication, computation etc.) is addressed by introducing hosting physical components for implementation.

The same viewpoint-driven, functional-based approach as for logical architecture building is used. The model at that point is considered ready to develop by downstream engineering teams.

Outputs of this engineering activity consist of the selected physical architecture which includes global behavior, components to be produced, formalization of all viewpoints and the way they are taken into account in the components design. Links with requirements and operational scenarios are also produced.

See **<Data elaboration description>** below

Engineering goals

- Manage engineering complexity through a structuring architecture,
  easing separation of (technical) concerns,
  favouring safe and separate development of components,
  securing and allowing an efficient IVVQ
- Favour Reuse of legacy Assets, and Product Policy through relevant Design
- Early validate some key features of the solution

Tasks to be completed during this step

- Define Architectural Principles and Patterns
- Define finalised functional and non-functional Behavior
- Consider Reuse of existing Assets
- Build candidate detailled Architectures
- Select and finalise the Physical Reference Architecture

Stop Criteria

This activity is achieved when one architecture can be considered – and proven - as the best compromise according to multi-viewpoint analysis, if it integrates all major constraints allocated to the System/SW at this level, and is sufficiently refined to be developed by lower level component providers.

Contributors & Competencies

Major competencies required to complete this step are suggested below:

Required Competencies	Major Expected Contribution	Possible Contributors (*)
Domain knowledge		• Chief architect • Sub-contractors • Systems engineering manager • Specialty engineering expert • System engineer • Software/hardware specialists • IVVQ manager • Product line manager • Simulation expert

		• Program manager • Configuration manager • Others
Product & Technical Domain (incl. Product line)	Product line constraints, especially reuse	
Systems engineering - Design		
Functional & non functional Analysis	System Analysis refinement/complement	
Architecture definition (logical, physical)	Architectural design & compromise	
Modeling & viewpoint engineering techniques	Modeling techniques, viewpoint building	
Value Engineering	Cost effectiveness of solutions	
Systems engineering - Specialties		
Technical Simulation	Validity of technical choices	
Specialties engineering (safety, perf, RAMST…)	Adequacy of Architecture to specialty	
Technical choices & TRL	Selection of technologies & derisking	
Configuration Management **Systems engineering / IVVQ**	Validity of component breakdown	
Test & Trials Strategy Plan	Definition of integration constraints	
Integration means	Realism of integration means	

definition	

(*) Depending on each organisation, competencies may be allocated to different actors; the following contributors list is therefore just an example to be adapted to each organisation:

<Data elaboration description>

The approach is similar to the one performed in 'Evaluate & Verify Architecture Choices', with more in-depth analysis and technical, design choices, up to the detail required to specify subsystems and IVVQ.

More detailled description is done of **Design Functions**, **Design Functional exchanges**,
 adding detailled **Design Exchange contents** to **Design Functional exchanges** to describe interactions between functions,
 and **Components Exchanges** to design interfaces between components accordingly.

The structuration of the system into finalised **Components performing Functions** is performed in coherency with the former notional logical architecture, and allocating **Design Functions** above to these **Components performing Functions** with the functional analysis above.

Physical resources hosting the **Components performing Functions** are defined as **Physical Hosting Components** and **Physical Component Links** between them, to which the **Components Exchanges** will be allocated. This completes the components interface definition.

Each of the former **Components performing Functions**, **Physical Hosting Components** and **Physical Component Links** is the source of **Configuration Items** definition, resulting in a first version of the **PBS**.

This detailled architecture is also completed and verified by defining the way **Specified Capabilities** are implemented in the **Designed Solution Capabilities** of the detailled, finalised **Designed Solution Architecture**.

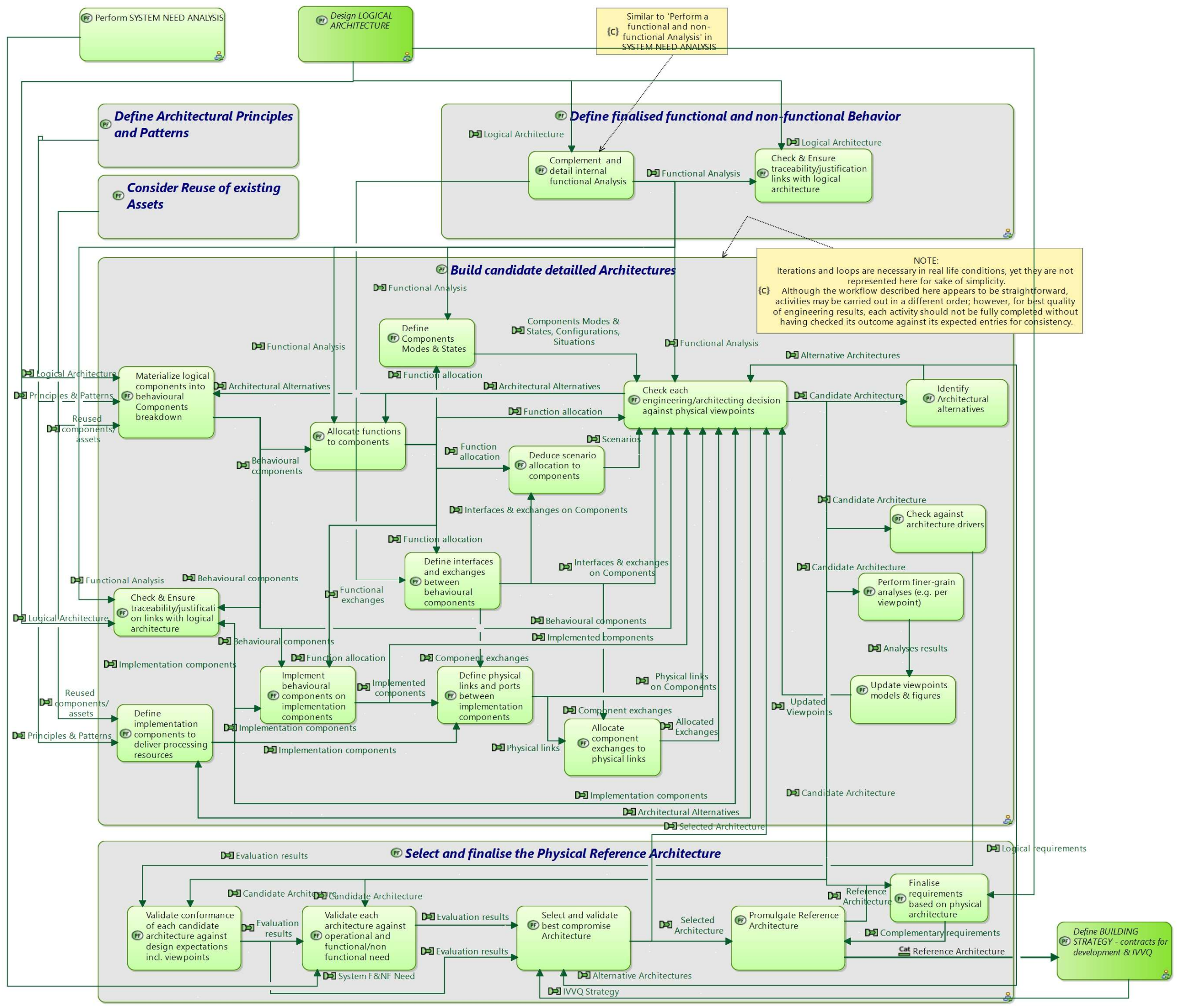
For this purpose, **Design Functional Chains** and **Design Scenarios** are defined, inspired by **Spec. Functional Chains** and **Specified Scenarios**. They are applied to **Design Functions** and **Components performing Functions**, and contribute to the global coherency check of the final architecture.

<Data elaboration description end>

This figure provides an inner view of the task Design PHYSICAL ARCHITECTURE, in the context of Arcadia core perspectives.

For each of the first level tasks presented here, only links with other core perspectives are displayed.

For a complete view of links with other engineering tasks and activities, see the figure dedicated to this first level task.


Perform SYSTEM NEED ANALYSIS
Design LOGICAL ARCHITECTURE
Similar to 'Perform a functional and non-functional Analysis' in SYSTEM NEED ANALYSIS
Define Architectural Principles and Patterns
Consider Reuse of existing Assets
Define finalised functional and non-functional Behavior
Complement and detail internal functional Analysis
Check & Ensure traceability/justification links with logical architecture
NOTE: Iterations and loops are necessary in real life conditions, yet they are not represented here for sake of simplicity. Although the workflow described here appears to be straightforward, activities may be carried out in a different order; however, for best quality of engineering results, each activity should not be fully completed without having checked its outcome against its expected entries for consistency.
Build candidate detailled Architectures
Define Components Modes & States
Materialize logical components into behavioural Components breakdown
Allocate functions to components
Deduce scenario allocation to components
Check each engineering/architecting decision against physical viewpoints
Identify Architectural alternatives
Check against architecture drivers
Perform finer-grain analyses (e.g. per viewpoint)
Update viewpoints models & figures
Define interfaces and exchanges between behavioural components
Check & Ensure traceability/justification on links with logical architecture
Implement behavioural components on implementation components
Define physical links and ports between implementation components
Allocate component exchanges to physical links
Define implementation components to deliver processing resources
Select and finalise the Physical Reference Architecture
Validate conformance of each candidate architecture against design expectations incl. viewpoints
Validate each architecture against operational and functional/non functional need
Select and validate best compromise Architecture
Promulgate Reference Architecture
Finalise requirements based on physical architecture
Define BUILDING STRATEGY - contracts for development & IVVQ
Logical Architecture
Functional Analysis
Principles & Patterns
Reused components/assets
Architectural Alternatives
Alternative Architectures
Candidate Architecture
Function allocation
Scenarios
Components Modes & States, Configurations, Situations
Behavioural components
Interfaces & exchanges on Components
Functional exchanges
Implemented components
Implementation components
Component exchanges
Physical links
Physical links on Components
Allocated Exchanges
Updated Viewpoints
Analyses results
Selected Architecture
Evaluation results
System F&NF Need
IVVQ Strategy
Reference Architecture
Complementary requirements
Logical requirements

5.2.3.1 Define Architectural Principles and Patterns

Identify architecture Invariants (common and generic behaviours, interfaces, functions, services, …) that simplify definition, implementation, development and integration of the system/software, by reducing diversity and heterogeneity of features in the architecture. These are called architectural Patterns.

- Rationalisation components & functions: search for similarities in logical architecture analysis (e.g. a data server simplifying communications of a highly shared data set, common and generic hardware computing core, data transformation library…)

- Technical functions and services: in order to support common behaviours and properties expected from the architecture (e.g. communication services, components lifecycle, supervision/surveillance, reconfiguration means, test & observability probes…)

- Technological patterns: driven by technology and state-of-the-art in architecting and hardware/software technology (e.g. component model, client-server paradigm, hardware FFT computing resource).

These modelling concepts may be considered as catalogue elements providing parts of models of efficient architectural solutions to be adapted to target components. The use of architectural patterns is of great benefit for easing separation of concerns, unifying behaviour, supporting internal standards e.g. component-based design, communication means, system-level common services…

Standards compliance should be sought as much as possible, while checking their real adequacy to the planned use in the system/software.

Input:

- state-of-the-art architectural patterns (e.g. components (containers), services..., real-time architectures & concepts (RMA, queuing networks…)
- Standards

Output

- Architectural patterns applicable to the target architecture (including interfaces)
- Selection of Architectural patterns to be applied to logical/physical components and/or their interfaces

Target documents:

- System/Segment Design Document (SSDD)

Verification and Consistency checks

External consistency:

- Between domain (User Requirements, non-functional constraints) and applicable standards / patterns

Internal consistency:

- Between Architectural patterns & standards

5.2.3.2 Consider Reuse of existing Assets

Consider Reuse of existing assets, e.g. COTS, middleware, legacy components, hardware functions components & boards, frameworks, execution platforms…

In each case, opportunity to reuse should be considered at least for each main viewpoint identified above:

not only in terms of functional or technical contents, but also interfaces proximity, dynamic behaviour, operational use (environmental conditions, performance, operational use scenarios), platform features & resource consumption, and more.

This analysis should lead to identifying gaps between initial development and reuse conditions, and therefore to feasibility and cost of required adaptations.

Input:

- Existing assets complying or not with standards (ideally, including their reusable formalisation and/or models)

Output:

- Identification of existing reusable assets applicable to the system/software

- Required adaptation developments

Target documents:

- System/Segment Design Document (SSDD)

Verification and Consistency checks:

External consistency:

- Between Asset available description (spec, interfaces documents, applicable standards…) and asset repository

Internal consistency:

- between assets features, properties and corresponding required functions in the system/software

5.2.3.3 Define finalised functional and non-functional Behavior

This task is similar to Logical Architecture 'Perform a functional and non-functional Analysis'.

Define a detailled functional behaviour that details and concretises former notional functional analysis, addressing:

- ready-to-develop description of designed behaviour
- greater level of detail resolving ambiguities of definition
- and design decisions choosing among various implementation options
- enrichment/confrontation with reused assets
- functions required for technical and technological implementation constraints.

Build and maintain justification and traceability links with Logical Architecture functions, functional chains, scenarios, modes &states, data etc.

More precisely,

- Identify functions required to satisfy and implement all Logical Architecture notional functions

- Complement them with necessary functions that were not identified Logical Architecture
- List and detail information, data flows, managed, exchanged and required by all these functions (internal or external to system); including required standards & interfaces
- Identify functional chains traversing the system/SW in order to implement need defined functional chains (traversing functions & data flows); similarly define functional scenarios implementing those defined at Logical Architecture level; enrich them if needed in order to appropriately define and check solution behaviour
- Identify system/SW modes & states, relate them to functions; enrich them if needed
- Create and maintain traceability links with Logical Architecture (e.g. between functions, between functional chains, between scenarios).

Identify all major dimensioning needs, and [non-functional] constraints, relating them to the appropriate fonctions, functional chains, actors… and associated scenarios, and relate them to system scenarios: e.g.

- Identify non-functional constraints (performance, safety…) and relate them to concerned functions, functional chains…
- Identify industrial constraints not coming from customer/user: ability to produce, to test, to maintain, to sub-contract…
- When intending to reuse existing assets, check this functional/non-functional analysis against these assets for compatibility.
- Enrich system scenarios with non-functional & industrial constraints
- Identify and select main (non functional) viewpoints (concerns) (*) susceptible to impact the functional analysis.
  Each viewpoint should emphasise a specific set of constraints or expected behaviour, quality, respect of non-functional properties… At least one viewpoint should be dedicated to Reuse and Product Policy.
- Try to order them in terms of importance, relative priority.

Ensure traceability/justification links between notional and finalised functional/non-functional analyses, and check consistency/coherency between them.

Input:

- Notional logical functional and non-functional analysis
- Reusable assets functional & non functional description

Output:

- Functional & non-functional analysis result (System functional breakdown + dataflow, functional chains, non functional constraints, scenarios…)
- Traceability between notional & finalised functional analyses
- List of relevant /critical viewpoints for the target system architecture

Target documents:

- System/Segment Design Document (SSDD)

Verification and Consistency checks:

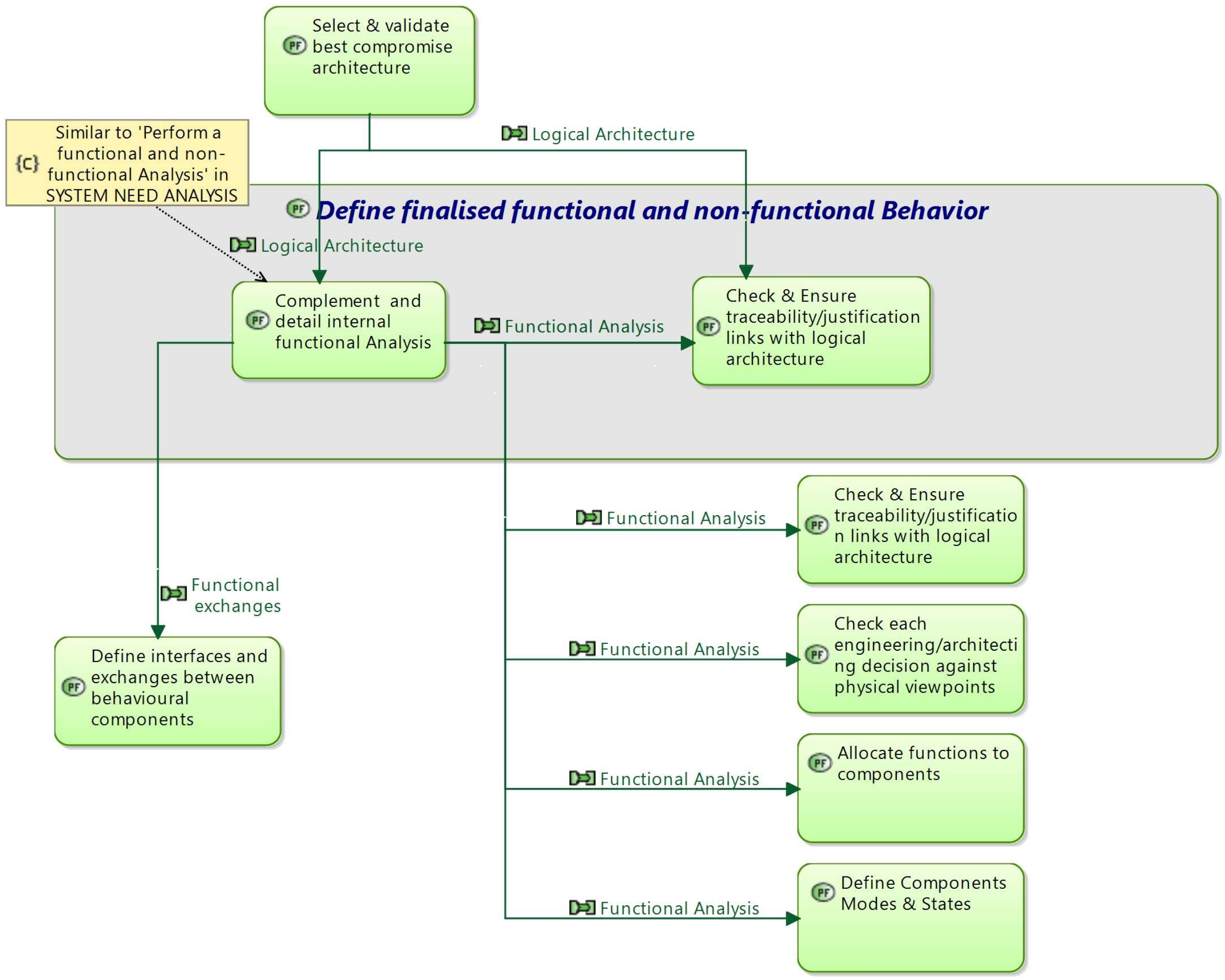
External consistency:

- Between finalised and notional functional Analysis functions/data…

Internal consistency:

- Between all functional & non-functional elements
- Verify the functional/non-functional Behaviour Description: coherent, complete, relevant: no contradiction, no gap, no inaccuracy.

This figure describes the interactions of the considered task with other engineering activities.



Select & validate best compromise architecture
Logical Architecture
Similar to 'Perform a functional and non-functional Analysis' in SYSTEM NEED ANALYSIS
Define finalised functional and non-functional Behavior
Logical Architecture
Complement and detail internal functional Analysis
Functional Analysis
Check & Ensure traceability/justification links with logical architecture
Functional Analysis
Check & Ensure traceability/justification links with logical architecture
Functional exchanges
Functional Analysis
Check each engineering/architecting decision against physical viewpoints
Define interfaces and exchanges between behavioural components
Functional Analysis
Allocate functions to components
Functional Analysis
Define Components Modes & States

5.2.3.3.1 Complement and detail internal functional Analysis

Detail and re-factor internal functional analysis from logical architecture (esp. functions) addressing

·greater level of detail resolving ambiguities of definition

·design decisions choosing among various implementation options

·enrichment/confrontation with reused assets

·functions required for technical and technological implementation constraints.

Build and maintain justification and traceability links with previous functional analysis.

*** Please refer to System Need Analysis / Perform a functional & non-functional analysis ***

5.2.3.3.2 Check & Ensure traceability/justification links with logical architecture

between physical architecture functions and logical architecture functions, functional chains, data and information, behavioural components and logical components, functional allocation to components, scenarios...

including checking consistency/coherency between them.

5.2.3.4 **Build candidate detailled Architectures**

Starting from the logical architecture above, use the same approach of component building (please refer to logical architecture 'Build candidate architectural breakdowns in Components'), in order to finalise implementation decisions and consequences on architecture.

The component breakdown definition should go in deeper detail by detailing and refining as necessary, especially identifying

- Behavioural components carrying functional contents (mainly derived from logical architecture ones, e.g. software components, programmable logic device programming, hardware processing functions, or equipment) to which functions are to be allocated
- Implementation components giving resources for behavioural components execution (e.g. processors, programmable logic devices such as FPGA but also middlewares and operating systems if needed) through allocation links
- Behavioural component interfaces and exchanges, deduced from functional data flows (e.g. by grouping)
- Implementation components interfaces and physical links (e.g. bus, network, power line) on which behavioural exchanges will be allocated
- Architectural patterns that optimise and rationalise the architecture, applied to each relevant architecture element
- Technology-originated architectural patterns to implement the selected design/development technologies
- Reused assets.

Remember to check any architecture & design decision against selected architecture drivers and viewpoints.

Note that a few candidate physical architectures may be defined, compared with each other in order to elect the best one through multi-viewpoint analysis.

Traceability and justification links with former steps is to be maintained at function level, component level (with logical components & functions), requirements, scenarios (allocated to components) etc.

Input:

Logical architecture, architecture drivers

Architectural patterns

Reusable assets

Logical architecture Requirements

Output:

- Candidate physical architecture
- Physical architecture Requirements

Target documents:

- System/Segment Design Document (SSDD)

Verification and Consistency checks:

External consistency:

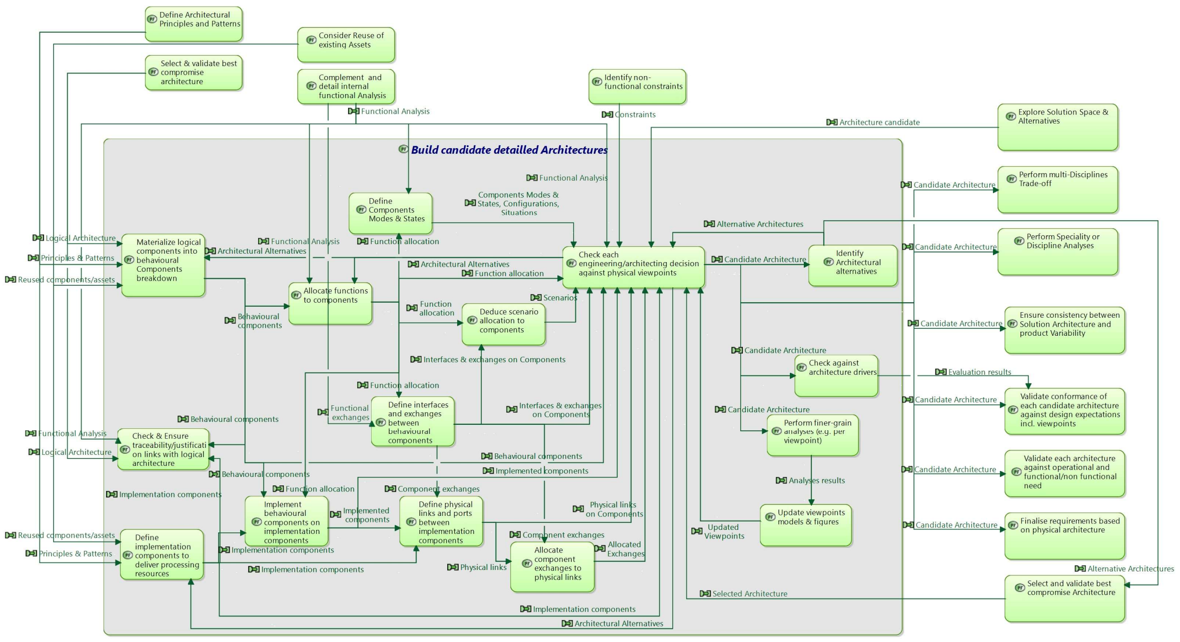
- Between Physical Architecture and Logical components and Logical interfaces, viewpoints…
- Between Physical Architecture components requirements and Logical Architecture requirements
- Between Operational & System/SW need analyses and Physical Architecture

Internal consistency:

- Between Physical components & interfaces and reusable Assets & Architectural patterns and their implementation in physical architecture

- Verify the reference physical architecture Description: coherent, complete, relevant: no contradiction, no gap, no inaccuracy.
- Between Physical Architecture components requirements & justifications and Reference physical architecture
- Verify the requirements Description: coherent, complete, relevant: no contradiction, no gap, no inaccuracy.

This figure describes the interactions of the considered task with other engineering activities.


Define Architectural Principles and Patterns
Consider Reuse of existing Assets
Select & validate best compromise architecture
Complement and detail internal functional Analysis
Functional Analysis
Identify non-functional constraints
Constraints
Architecture candidate
Explore Solution Space & Alternatives
Build candidate detailled Architectures
Functional Analysis
Define Components Modes & States
Components Modes & States, Configurations, Situations
Logical Architecture
Principles & Patterns
Reused components/assets
Materialize logical components into behavioural Components breakdown
Functional Analysis
Architectural Alternatives
Function allocation
Architectural Alternatives
Function allocation
Check each engineering/architecting decision against physical viewpoints
Alternative Architectures
Candidate Architecture
Identify Architectural alternatives
Candidate Architecture
Perform multi-Disciplines Trade-off
Candidate Architecture
Perform Speciality or Discipline Analyses
Allocate functions to components
Behavioural components
Function allocation
Deduce scenario allocation to components
Scenarios
Candidate Architecture
Ensure consistency between Solution Architecture and product Variability
Candidate Architecture
Check against architecture drivers
Evaluation results
Interfaces & exchanges on Components
Function allocation
Functional exchanges
Define interfaces and exchanges between behavioural components
Interfaces & exchanges on Components
Candidate Architecture
Validate conformance of each candidate architecture against design expectations incl. viewpoints
Candidate Architecture
Perform finer-grain analyses (e.g. per viewpoint)
Behavioural components
Functional Analysis
Logical Architecture
Check & Ensure traceability/justification on links with logical architecture
Behavioural components
Implemented components
Behavioural components
Candidate Architecture
Validate each architecture against operational and functional/non functional need
Analyses results
Implementation components
Function allocation
Component exchanges
Implement behavioural components on implementation components
Implemented components
Define physical links and ports between implementation components
Physical links on Components
Update viewpoints models & figures
Updated Viewpoints
Candidate Architecture
Finalise requirements based on physical architecture
Reused components/assets
Principles & Patterns
Define implementation components to deliver processing resources
Implementation components
Component exchanges
Allocate component exchanges to physical links
Allocated Exchanges
Implementation components
Physical links
Alternative Architectures
Select and validate best compromise Architecture
Selected Architecture
Implementation components
Architectural Alternatives

5.2.3.4.1 Materialize logical components into behavioural Components breakdown

Behavioural components carrying functional contents (mainly derived from logical architecture ones, e.g. software components, programmable logic device programming, hardware processing functions, or equipment) to which functions are to be allocated

5.2.3.4.2 Allocate functions to components

This building process has to deal with each Viewpoint & associated design Rules, either by

·Grouping functions close to each other in the considered viewpoint (e.g. dealing with the same operational activity, having same hard real-time constraints, sharing complex interfaces…)

·Or segregating / separating functions that must not be grouped (e.g. functions of different criticality/certification levels, functions heavily consuming platform resources…)

·Or mixed.

5.2.3.4.3 Define interfaces and exchanges between behavioural components

based on allocated functions and deduced from functional exchanges between them, in accordance with logical components interfaces & exchanges;

functional exchanges should here be allocated to component exchanges.

5.2.3.4.4 Deduce scenario allocation to components

based on functions allocation & traceability links, create interaction scenarios between components and with external actors

5.2.3.4.5 Define Components Modes & States

based on system and actors modes and states, define the possible contribution of each component.

consider component own behaviour, and define its own modes and states if needed, based on functional analysis allocation to the component.

consider communications with other components or actors (incl. external systems), and define component dedicated modes and states describing the contribution of the component to the communication protocol.

verify consistency and coherency of all these modes and states, the condition of transitions between them (notably based on functional exchanges), and the related availability of architecture elements (functions, functional chains, component and sub-components etc.) in each of them.

5.2.3.4.6 Check & Ensure traceability/justification links with logical architecture

between physical architecture functions and logical architectured functions, functional chains, data and information, behavioural components and logical components, scenarios...

including checking consistency/coherency between them.

5.2.3.4.7 Define implementation components to deliver processing resources

Implementation components giving resources for behavioural components execution (e.g. processors, programmable logic devices such as FPGA but also middlewares and operating systems if needed) through allocation links

5.2.3.4.8 Implement behavioural components on implementation components

Implementation components giving resources for behavioural components execution (e.g. processors, programmable logic devices such as FPGA but also middlewares and operating systems if needed) through allocation links

5.2.3.4.9 Define physical links and ports between implementation components

based on allocated functions and behavioural components, and deduced from functional and behavioural exchanges between them all.

5.2.3.4.10 Allocate component exchanges to physical links

based on allocated behavioural components and implementation components, and deduced from behavioural exchanges;

behavioural exchanges should here be allocated to physical links.

5.2.3.4.11 Check each engineering/architecting decision against physical viewpoints

.check how much this architecture satisfies or infringes each viewpoint design rules, and expected non-functional properties

.analyse the impact of this architecture on allr key viewpoints (e.g. safety, performance), to detect discrepancies and "distortions" between viewpoints

.correct and iterate as needed.

5.2.3.4.12 Identify Architectural alternatives

consider and formalise different ways to implement expected behaviour, different allocation of a functional behaviour on behavioural components, different deployments of behavioural components on resource implementation components, including different non-functional properties and qualities of service.

5.2.3.4.13 Check against architecture drivers

verifying that architecture drivers are applied and that design rules are satisfied.

5.2.3.4.14 Perform finer-grain analyses (e.g. per viewpoint)

check in deeper details design hypotheses and choices, notably using specialised simulation and analyses models.

See also 'Perform Speciality or Discipline Analyses'

5.2.3.4.15 Update viewpoints models & figures

collect and synthetise results of fine-grained analyses and simulations to enrich and valuate the phsyical architecture model.

5.2.3.5 Select and finalise the Physical Reference Architecture

Starting from the candidate architectures identified previously, use the same approach of best compromise selection as in logical architecture (please refer to logical architecture 'Select best Compromise Architecture'), in order to finalise the reference architecture that will be designed and developped by sub-systems, software, hardware components engineering teams.

Drive an early verification & check

- Define finer-grained and more realistic behaviour models in order to check architecture compromise against finer analyses.
- Check the correctness of the physical architecture through simulation means, formal check… of these models, for each major viewpoint to be refined.
- Then update architectural viewpoints with results of these fine-grain analyses (e.g. estimations of resource consumption, fault propagation equations, more realistic real-time activation & behaviour rules, true hardware metrics…), and rerun a multi-viewpoint analysis to maintain an acceptable compromise/optimum solution.

Consolidate against Need & Finalise

Check final reference architecture against operational & System/SW functional/non-functional analyses.

Derive, allocate and define requirements for each of the newly defined components of the physical architecture.

Check and justify these requirements against physical architecture and former requirements.

Input:

- Logical architecture, architecture drivers
- Logical architecture Requirements
- Architectural patterns
- Detailed functional and non-functional behavior
- Candidate architectures

Output:

- Reference physical architecture
- Physical architecture Requirements

Target documents:

- System/Segment Design Document (SSDD)

Verification and Consistency checks:

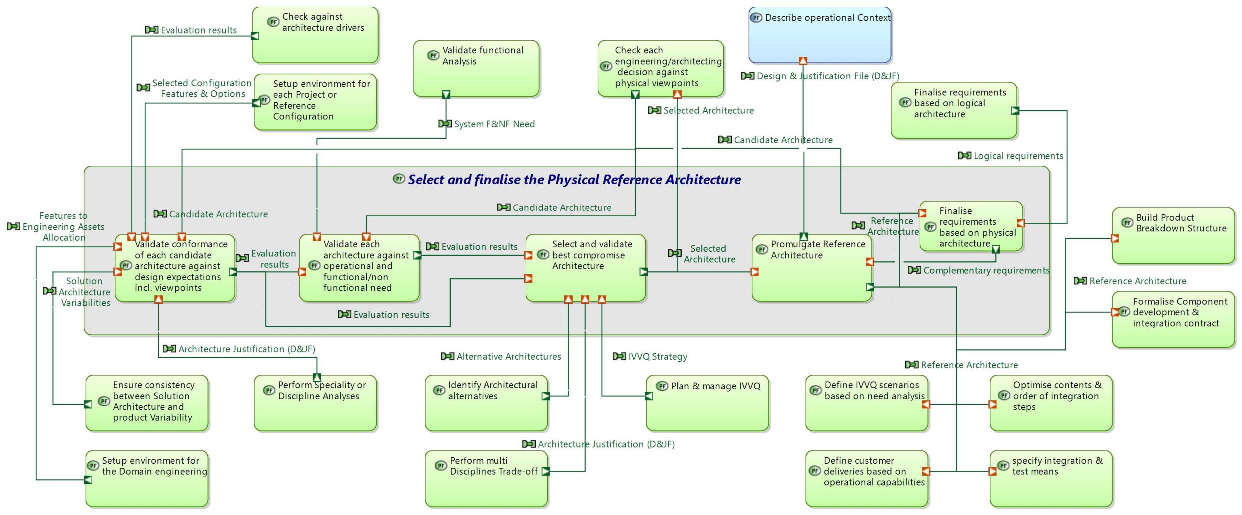
External consistency:

- Between Physical Architecture and Logical components and Logical interfaces, viewpoints…
- Between Physical Architecture components requirements and Logical Architecture requirements
- Between Operational & System/SW need analyses and Physical Architecture

Internal consistency:

- Between Physical components & interfaces and reusable Assets & Architectural patterns and their implementation in physical architecture
- Verify the reference physical architecture Description: coherent, complete, relevant: no contradiction, no gap, no inaccuracy.
- Between Physical Architecture components requirements & justifications and Reference physical architecture
- Verify the requirements Description: coherent, complete, relevant: no contradiction, no gap, no inaccuracy.

This figure describes the interactions of the considered task with other engineering activities.


Select and finalise the Physical Reference Architecture
Check against architecture drivers
Setup environment for each Project or Reference Configuration
Validate functional Analysis
Check each engineering/architecting decision against physical viewpoints
Describe operational Context
Finalise requirements based on logical architecture
Validate conformance of each candidate architecture against design expectations incl. viewpoints
Validate each architecture against operational and functional/non functional need
Select and validate best compromise Architecture
Promulgate Reference Architecture
Finalise requirements based on physical architecture
Build Product Breakdown Structure
Formalise Component development & integration contract
Ensure consistency between Solution Architecture and product Variability
Perform Speciality or Discipline Analyses
Identify Architectural alternatives
Plan & manage IVVQ
Define IVVQ scenarios based on need analysis
Optimise contents & order of integration steps
Setup environment for the Domain engineering
Perform multi-Disciplines Trade-off
Define customer deliveries based on operational capabilities
specify integration & test means
Evaluation results
Selected Configuration Features & Options
System F&NF Need
Selected Architecture
Candidate Architecture
Design & Justification File (D&JF)
Logical requirements
Features to Engineering Assets Allocation
Solution Architecture Variabilities
Reference Architecture
Complementary requirements
Architecture Justification (D&JF)
Alternative Architectures
IVVQ Strategy

5.2.3.5.1 Validate conformance of each candidate architecture against design expectations incl. viewpoints

A general approach (to be adapted to each domain) might be:

1.select most important viewpoint to structure the system (e.g. performance, or safety level…)

2.check how much this architecture satisfies or infringes the viewpoint design rules, and expected non-functional properties

3.analyse the impact of this architecture on other key viewpoints (e.g. safety, performance), to detect discrepancies and "distortions" between viewpoints

4.correct and iterate as needed.

5.2.3.5.2 Validate each architecture against operational and functional/non functional need

·how it supports operational activities,

·how it deals with functional behaviour (functional contents of each component, contribution to functional chains), interfaces, data flows & data models…,

·how it satisfies expected properties & constraints,

·how it implements/responds to operational scenarios & capabilities.

NOTE: also using traceability links and functional-to-architecture allocation

5.2.3.5.3 Select and validate best compromise Architecture

based on how each architecture fulfils need and main viewpoints analysis results

5.2.3.5.4 Promulgate Reference Architecture

finalise the definiiton and later use it as the unique reference for further engienering, development & IVVQ

5.2.3.5.5 Finalise requirements based on physical architecture

based on requirement - function links, and functions to component allocation, allocate requirements to components and check that architecture fulfils requirements.

Complement requirements as needed.

5.2.4 Analyse the solution

The goal is to early verify that the solution as designed meets all stakeholders expectations and requirements, industrial and project constraints (including cost, schedule, resources and more), notably functional and non-functional expectations.

A coarse-grained multi-viewpoint analysis is performed using the architecture core model, so as to discard irrelevant architecture alternatives as early as possible; this analysis shall be fast enough so as to be performed for each architecture design decision. It will evaluate how much the expectations of each viewpoint are satisfied according to design analysis rules for each alternative.

Then, in order to confirm the validity of architecture choices, for each major engineering and discipline or specialty viewpoint, a single-viewpoint, finer-grained dedicated analysis is performed, using more detailed representation, dedicated languages and models, specific analysis techniques and tools. The representation used for these analysis may be partly initialised from the architecture model contents, both for cost-effectiveness and coherency mastering.

If a single-viewpoint analysis shows inadequacy of the architecture with expectations, then a new multi-viewpoint analysis should be performed so as to find a better compromise.

See sub-activities.

5.2.4.1 Perform Speciality or Discipline Analyses

In order to confirm the validity of architecture choices, for each major engineering and discipline or specialty viewpoint, a single-viewpoint, finer-grained dedicated analysis is performed, using more detailed representation, dedicated languages and models, specific analysis techniques and tools. The representation used for these analysis may be partly initialised from the architecture model contents, both for cost-effectiveness and coherency mastering.

<Data elaboration description>

Designed Solution Architecture elements, e.g. **Components performing Functions**, are used to initialise **Simulation Models**, **Specialty Analysis Models**, **Architecture viewpoint Models**.

Simulation Scenarios, **Specialty Analysis Context** and **Viewpoint Analysis Context** are also initialised using **Design Scenarios**.

Then **Simulation Results**, **Specialty Analysis Results**, and **Viewpoint Analysis Results** are used to valuate **Designed Solution Architecture** elements.

<Data elaboration description end>

5.2.4.2 Perform multi-Disciplines Trade-off

A coarse-grained multi-viewpoint analysis is performed using the architecture core model, so as to discard irrelevant architecture alternatives as early as possible; this analysis shall be fast enough so as to be performed for each architecture design decision. It will evaluate how much the expectations of each viewpoint are satisfied according to design analysis rules for each alternative.

If a single-viewpoint analysis shows inadequacy of the architecture with expectations, then a new multi-viewpoint analysis should be performed so as to find a better compromise.

<Data elaboration description>

Operational Mission Analysis and **System Need Specification** elements are valuated or characterised with expected properties and constraints according to each viewpoint. Complementary **Viewpoint Elements** can be added (e.g. feared events) to express these expectations. **Designed Solution Architecture** elements can also be characterised similarly for each viewpoint.

Then, viewpoint-specific analysis are performed on the model, so as to check conformity to expectations. **Viewpoint Analysis Results** are added of fed with results from analyses, so as to compare each architecture alternative in terms of fulfilment of each viewpoint expectations.

<Data elaboration description end>

5.3 PREPARE AND PERFORM DEVELOPMENT AND IVVQ

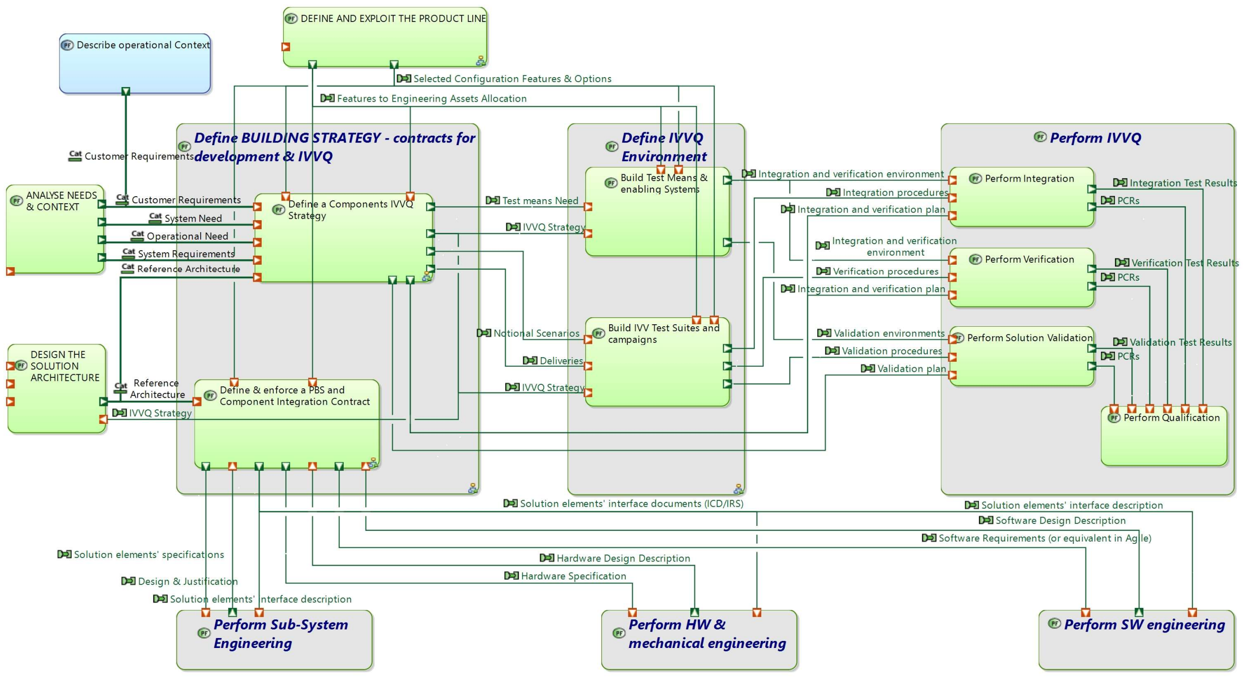
Define the requirements for each component of the solution to be purchased or built; define the strategy to integrate, test and verify the solution; then run design & development and integration, verification validation, qualification

See sub-tasks description

This figure provides an inner view of the task PREPARE AND PERFORM DEVELOPMENT AND IVVQ, in the context of Arcadia core perspectives.

For each of the first level tasks presented here, only links with other core perspectives are displayed.

For a complete view of links with other engineering tasks and activities, see the figure dedicated to this first level task.



DEFINE AND EXPLOIT THE PRODUCT LINE
Describe operational Context
Selected Configuration Features & Options
Features to Engineering Assets Allocation
Define BUILDING STRATEGY - contracts for development & IVVQ
Define IVVQ Environment
Perform IVVQ
Customer Requirements
ANALYSE NEEDS & CONTEXT
Customer Requirements
System Need
Operational Need
System Requirements
Reference Architecture
Define a Components IVVQ Strategy
Test means Need
IVVQ Strategy
Build Test Means & enabling Systems
Integration and verification environment
Integration procedures
Integration and verification plan
Perform Integration
Integration Test Results
PCRs
Integration and verification environment
Verification procedures
Integration and verification plan
Perform Verification
Verification Test Results
PCRs
Notional Scenarios
Deliveries
IVVQ Strategy
Build IVV Test Suites and campaigns
Validation environments
Validation procedures
Validation plan
Perform Solution Validation
Validation Test Results
PCRs
Perform Qualification
DESIGN THE SOLUTION ARCHITECTURE
Reference Architecture
IVVQ Strategy
Define & enforce a PBS and Component Integration Contract
Solution elements' interface documents (ICD/IRS)
Solution elements' interface description
Software Design Description
Software Requirements (or equivalent in Agile)
Solution elements' specifications
Hardware Design Description
Hardware Specification
Design & Justification
Solution elements' interface description
Perform Sub-System Engineering
Perform HW & mechanical engineering
Perform SW engineering

5.3.1 Define BUILDING STRATEGY - contracts for development & IVVQ

The fifth and last perspective is a contribution to EPBS (End-Product Breakdown Structure) building, taking benefits from the former architectural work, to enforce components requirements definition, and prepare a secured IVVQ (Integration Verification Validation Qualification).

All choices associated to the system/SW chosen architecture, and all hypothesis and constraints imposed to components and architecture to fit need and constraints, are summarized and checked here. IVV Strategy, including phasing/versioning, releases contents, integration trees, test means and enabling systems shall be defined based on the former perspectives models. Test campaigns are defined based on capabilities and scenarios.

Outputs from this activity are mainly "component Integration contracts" collecting all necessary expected properties for each constituent to be developed, on one side, IVV strategy and Test Campaigns/procedures on the other side.

Engineering goals

- Define contractual requirements for components and EPBS (End Product Breakdown Structure)
- Define an architectural frame & constraints to master components development & integration

- define an integration, verification, validation strategy defining contents of time-related releases/deliveries, their functional and structural contents, enabling systems and test means, and test campaigns to be run for each release

Tasks to be completed during this step

- Define a Components IVVQ Strategy
- Define & enforce a PBS and Component Integration Contract

Stop Criteria

This activity is achieved when an agreement with lower level stakeholders (incl. Lower level Engineering, suppliers) on EPBS and Integration contract has been obtained.

Contributors & Competencies

Major competencies required to complete this step are suggested below:

Required Competencies	**Major Expected Contribution**	**Possible Contributors (*)**
Systems engineering - Specialties		• Chief architect • Sub-contractors • Systems engineering manager • Software/hardware specialists • IVVQ manager • Program manager • Configuration manager • Others
Technical choices & TRL PBS	Selection of technologies & derisking Definition of PBS based	

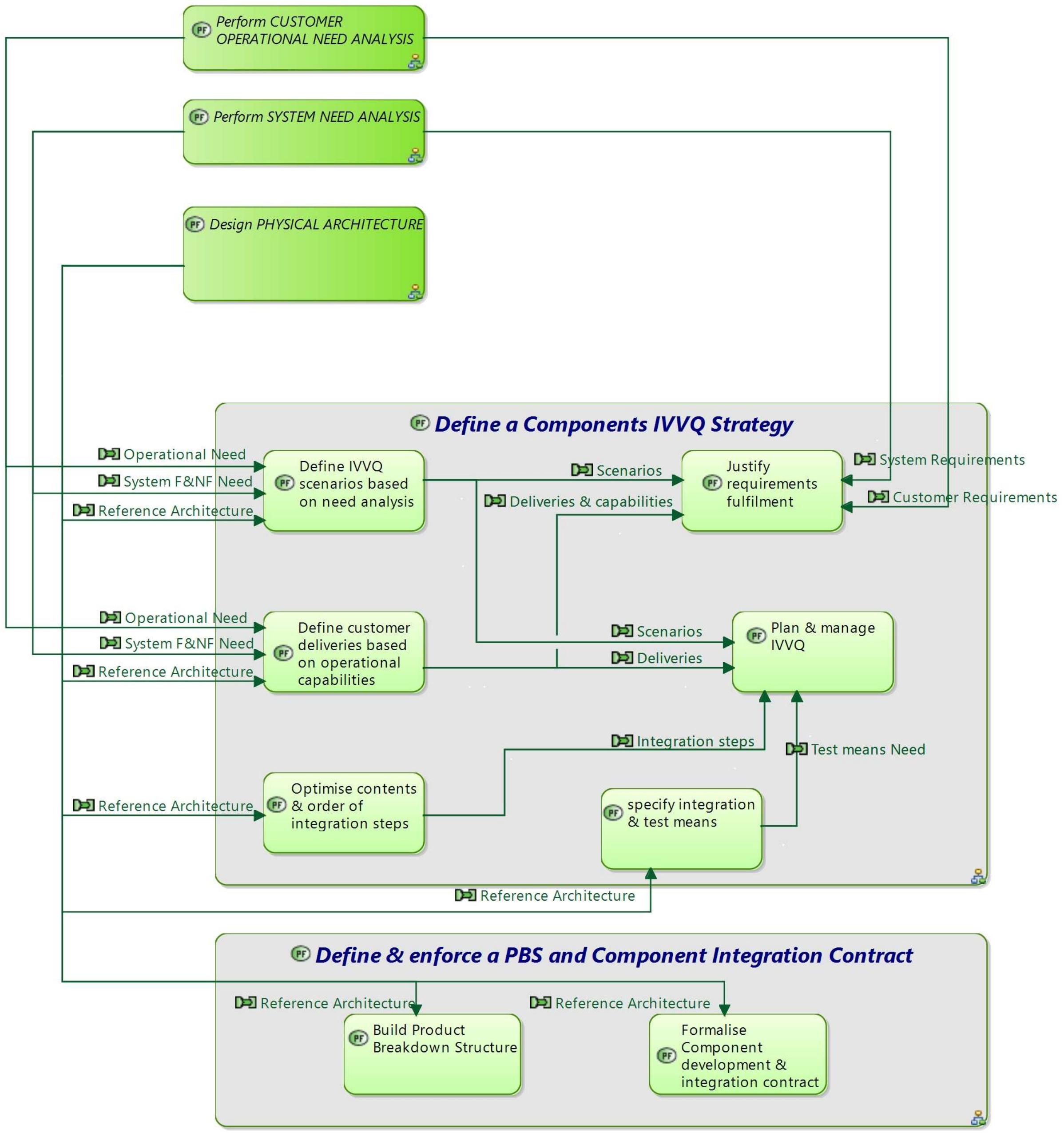
	on components
Configuration Management	Adequacy of PBS & configurations
Systems engineering / IVVQ	
Test & Trials Strategy Plan	Definition of integration constraints
Integration means definition	Definition of Component test means

(*) Depending on each organisation, competencies may be allocated to different actors; the following contributors list is therefore just an example to be adapted to each organisation:

This figure provides an inner view of the task Define BUILDING STRATEGY - contracts for development & IVVQ, in the context of Arcadia core perspectives.

For each of the first level tasks presented here, only links with other core perspectives are displayed.

For a complete view of links with other engineering tasks and activities, see the figure dedicated to this first level task.


Perform CUSTOMER OPERATIONAL NEED ANALYSIS
Perform SYSTEM NEED ANALYSIS
Design PHYSICAL ARCHITECTURE
Define a Components IVVQ Strategy
Operational Need
System F&NF Need
Reference Architecture
Define IVVQ scenarios based on need analysis
Scenarios
Deliveries & capabilities
Justify requirements fulfilment
System Requirements
Customer Requirements
Operational Need
System F&NF Need
Reference Architecture
Define customer deliveries based on operational capabilities
Scenarios
Deliveries
Plan & manage IVVQ
Integration steps
Test means Need
Reference Architecture
Optimise contents & order of integration steps
specify integration & test means
Reference Architecture
Define & enforce a PBS and Component Integration Contract
Reference Architecture
Reference Architecture
Build Product Breakdown Structure
Formalise Component development & integration contract

5.3.1.1 Define a Components IVVQ Strategy

Defining an IVVQ strategy is out of scope for this document; nevertheless, the approach presented here may contribute to this strategy elaboration in the following way:

- Define IVVQ scenarios for components, based on physical architecture scenarios and functional chains realising Operational Need capabilities, scenarios and operational processes.

- Define the contents of partial product deliveries based first on required capabilities, then on the contribution of requirements to operational tasks & goals (from all former perspectives, requirements analysis & check against need).
- Define the contents of integration steps (increments) based on cross- viewpoints impact analysis (other components, communications & interfaces, shared resources, critical paths, functional chains…): e.g. list other components from which a component depends, in order to include them in the same integration step .
- Test both requirements and operational need at once, using traceability between requirements, operational/functional analysis and architecture. The path should be:

1. relate tests to scenarios and functional chains from which they are specified
2. relate each tests step to model elements involved in it (functional chains, scenarios, components, interfaces, etc.)
3. for each test successfully passed, check model elements involved
4. when all tests related to a model elements are passed, consider this element as verified
5. when all model elements and tests linked to a textual [user] requirement are verified, consider that the requirement is verified

Note that the method allows a fine-grained integration plan definition, thanks to viewpoints impact analysis:

In former steps (logical and physical architecture design), creating viewpoints dedicated to IVVQ may help in defining components outlines and integration steps:

- identifying integration dependencies, functions or components to be simulated in early steps of IVVQ, and integration paths transverse to components (e.g. integrating in one stage all functions contributing to a sensor management, among all components)
- if needed, outlining components in order to favour their integration at once
- or identifying "integration paths" transverse to components, and using viewpoints analysis to identify necessary contents of each integration path thanks to dependencies modelling.

See **<Data elaboration description>** below

Input:

- Physical reference architecture and requirements.
- Operational and system/SW need analyses

Output:

- Components IVVQ strategy
- Target document:
- IVVQ plan
- System Integration and tests plan, including specification of test campaigns and tests contents

Verification and Consistency checks:

External consistency:

- Between Components IVVQ strategy and Operational needs, requirements, reference architecture, architectural viewpoints.

Internal consistency:

- See internal consistency specified at previous steps

<Data elaboration description>

Defining the **IVVQ Strategy** starts with defining the expected **IVV Workflow** :
for each of planned **IVV Steps** of IVVQ, the workflow describes its contents in one or more **IVV Releases** to be verified and possibly delivered.

This contents is mainly defined by the **Designed Solution Capabilities** that describe what the solution should deliver to the users.
These Solution capabilities consist of **Design Scenarios** and **Design Functional Chains**, that describe some typical use cases or user stories, that should be achievable with the release.

Scenarios and functional chains involve **Design Functions** that also contribute to defining the release. All are related to **Textual Requirements** that they contribute to verifiy.

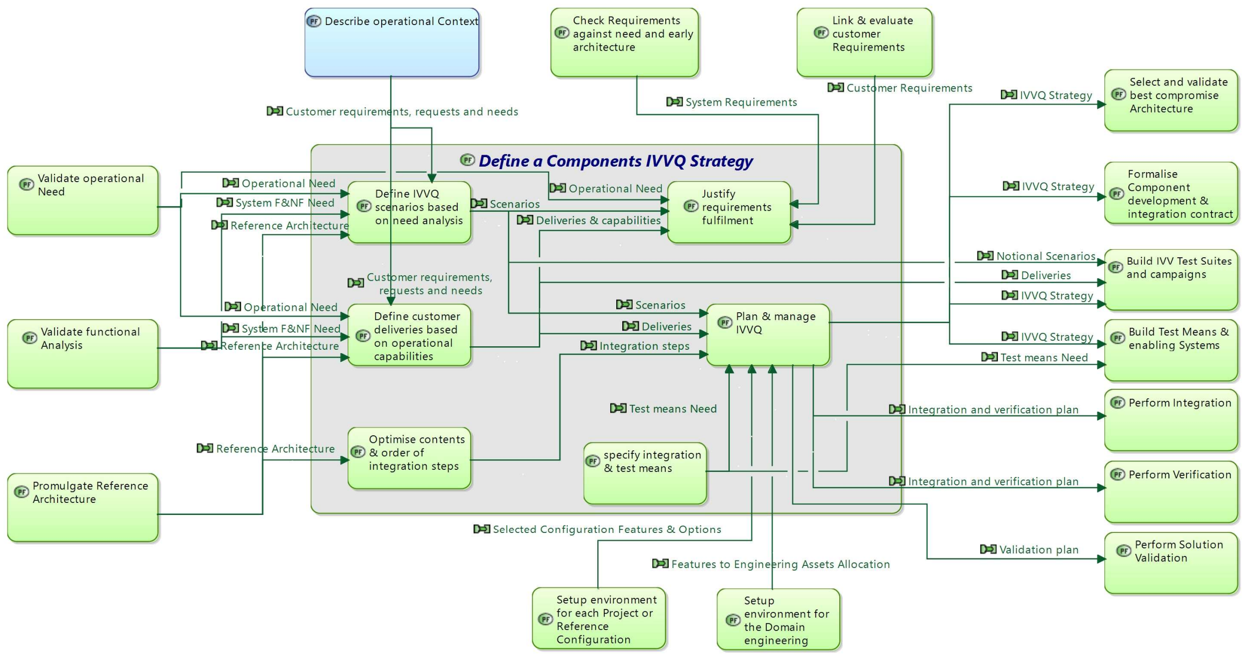
Note that these **Designed Solution Capabilities** should be traceable towards user-oriented **Specified Capabilities** and **Users Missions & Capabilities**, that should have been prioritized according to value Analysis. This value analysis should drive the definition of IVVQ order and priorities.

The **IVVQ Strategy** also includes the **Assembly Order Tree**, defining the logical order of building the **System Parts Assembly**, based on the **PBS**, Product Breakdown Structure of the architecture, and its **Configuration Items**.

The **IVV Order Tree** complements the **Assembly Order Tree** by adding, for each of the **IVV Steps**, the **IVV Configurations** to be set. These IVV Configurations include the **System Parts Assembly** required for the step (from the **Assembly Order Tree**), and the **Enabling/Test Means Definition** necessary to perform the expected tests on the components to be assembled and tested.

<Data elaboration description end>

This figure describes the interactions of the considered task with other engineering activities.


Describe operational Context
Check Requirements against need and early architecture
Link & evaluate customer Requirements
Customer requirements, requests and needs
System Requirements
Customer Requirements
IVVQ Strategy
Select and validate best compromise Architecture
Define a Components IVVQ Strategy
Validate operational Need
Operational Need
System F&NF Need
Reference Architecture
Define IVVQ scenarios based on need analysis
Scenarios
Operational Need
Deliveries & capabilities
Justify requirements fulfilment
IVVQ Strategy
Formalise Component development & integration contract
Notional Scenarios
Deliveries
IVVQ Strategy
Build IVV Test Suites and campaigns
Customer requirements, requests and needs
Operational Need
System F&NF Need
Reference Architecture
Define customer deliveries based on operational capabilities
Scenarios
Deliveries
Integration steps
Plan & manage IVVQ
IVVQ Strategy
Test means Need
Build Test Means & enabling Systems
Validate functional Analysis
Test means Need
Integration and verification plan
Perform Integration
Reference Architecture
Optimise contents & order of integration steps
specify integration & test means
Integration and verification plan
Perform Verification
Promulgate Reference Architecture
Selected Configuration Features & Options
Features to Engineering Assets Allocation
Validation plan
Perform Solution Validation
Setup environment for each Project or Reference Configuration
Setup environment for the Domain engineering

5.3.1.1.1 Define IVVQ scenarios based on need analysis

using operational & system need scenarios, and associated physical scenarios linked to them by traceability.

For partial integration, allocate these scenarios to relevant components

5.3.1.1.2 Define customer deliveries based on operational capabilities

based on the contribution of requirements to operational tasks & goals (from requirements analysis & check against need).

using traceability links between need and architecture

5.3.1.1.3 Optimise contents & order of integration steps

based on cross- viewpoints impact analysis: e.g. list dependencies of a component in order to include them in the same integration step (communications & interfaces, shared resources, critical paths, functional chains…).

5.3.1.1.4 Justify requirements fulfilment

both requirements and operational need at once, using traceability between requirements, operational/functional analysis and architecture

5.3.1.1.5 Plan & manage IVVQ

Check consistency of all elements contributing to the IVVQ strategy.

During IVV, manage impacts of components maturity level and deliveries delays, using the architecture model to identify consequences on test plans, necessary non-regression testing, etc.

5.3.1.1.6 specify integration & test means

based on analysis of dependencies between components, and specifying test means thanks to corresponding component description (interfaces, behaviour thanks to internal functions & scenarios...)

5.3.1.2 Define & enforce a PBS and Component Integration Contract

At this step, Configuration Items (CI: either component software CSCI or hardware HWCI) contents are to be defined in order to build a Product Breakdown Structure (PBS) , e.g.

- By grouping various former components in a bigger CI easier to manage,
- Or by federating various similar components in a single implementation CI that will be instantiated multiple times at deployment.

The component integration contract should therefore be based on:

- Operational & System/SW Views allocation (scenarios, system functions, properties)
- Resulting Functional, Interface, Performance Requirements
- Common system/SW-wide expected behaviour thanks to architectural Patterns & Framework Standards compliance
- Non-functional Requirements as defined and checked in each main architectural viewpoint
 (e.g. incl. Critical paths, resource Consumption, ability to reset/restart, redundancy, safety…).

It should be built, negociated/validated with component suppliers.

This integration contract should mention (same as above)

- Expected services, functions
- Interfaces with other components and outside
- Contribution to the system/SW-wide information model (or global interchanged data)
- Dynamic behaviour expectations
- Requested other services, components… to be used by it
- Expected operational performances
- Non functional performances, expected Quality of Service
- Internal modes & states expected management
- Contribution to system management (surveillance, start-up/shutdown, redundancy issues…)
- For a software component: allocated – allowed - resources (CPU, memory, communication bandwidth, real-time tasking & priorities…)
- interface with framework, middleware, hardware technical services (e.g. communication API, hardware drivers…)
- For a hardware component: environment constraints (temperature, vibration, ambient atmosphere, mechanical constraints…), allocated – allowed - resources (power consumption & dissipation, cooling…)

Define means to early validate or check the respect of components contract:

Supplement Requirements for Technical functions, services (middleware, Framework, hardware cores, execution Platform…) to ensure that check, and secure system behaviour if one component contract is not fulfilled.

See **<Data elaboration description>** below

Input:

- Physical architecture;
- Requirements;
- Operational and system/SW need analyses

Output:

- Product Breakdown Structure (PBS)
- Components Integration contracts including Configuration Items (CI) definition

Target documents:

- Contract definition documents (e.g. preliminary sub-systems SSS, software SRSs), EPBS

Verification and Consistency checks:

External consistency:

- Between Physical components (including interfaces) and Configuration Items
- Between EPBS requirements and Physical Architecture requirements
- Between Operational & system/SW need analyses & physical architecture, and component integration contracts

Internal consistency:

- Between EPBS requirements and Configuration Items

- Verify the EPBS & component integration contracts Description: coherent, complete, relevant: no contradiction, no gap, no inaccuracy.

<Data elaboration description>

Subsystem (or component) need definition is extracted from **Designed Solution Architecture** : functional contents from allocated functions, behaviour from scenarios and functional chains in which it is involved, allocated modes and states; external interfaces and data model, including exchanges, physical links etc.

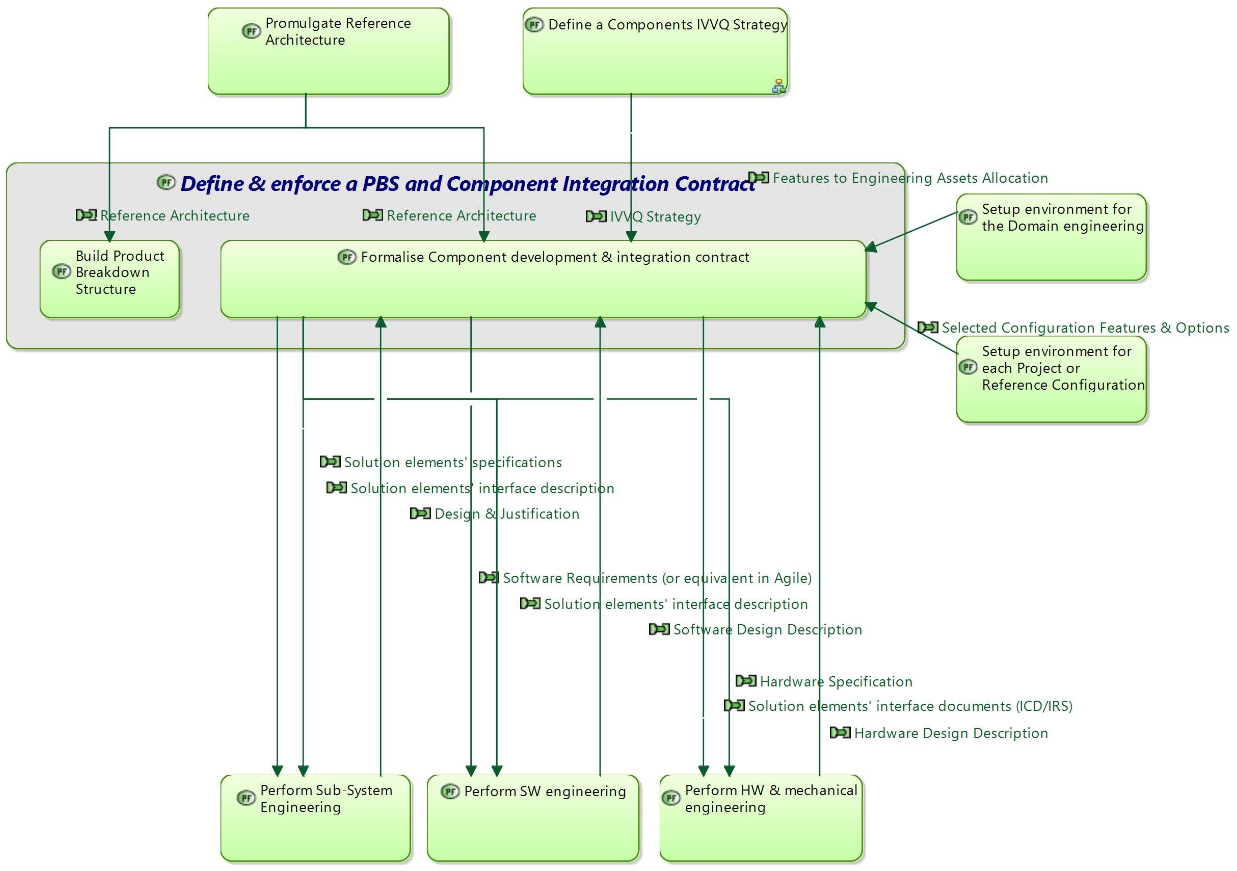
Subsystem expected validation test procedures and Test Casess are defined mainly based on system physical architecture scenarios and functional chains.

Scenarios allocated to each subsystem should be defined in physical architecture, for those tests that are to be used as subsystem validation scenarios, from the system engineering point of view.

Same for functional chains.

<Data elaboration description end>

This figure describes the interactions of the considered task with other engineering activities.


Promulgate Reference Architecture
Define a Components IVVQ Strategy
Define & enforce a PBS and Component Integration Contract
Features to Engineering Assets Allocation
Reference Architecture
Reference Architecture
IVVQ Strategy
Setup environment for the Domain engineering
Build Product Breakdown Structure
Formalise Component development & integration contract
Selected Configuration Features & Options
Setup environment for each Project or Reference Configuration
Solution elements' specifications
Solution elements' interface description
Design & Justification
Software Requirements (or equivalent in Agile)
Solution elements' interface description
Software Design Description
Hardware Specification
Solution elements' interface documents (ICD/IRS)
Hardware Design Description
Perform Sub-System Engineering
Perform SW engineering
Perform HW & mechanical engineering

5.3.1.2.1 Formalise Component development & integration contract

The component integration contract is to be defined for each component of physical architecture to be developped; it is built by using components description in physical architecture, and should include:

·Operational & System/SW Views allocation (scenarios, system functions, properties)

·Resulting Functional, Interface, Performance Requirements

·Non-functional Requirements as defined and checked in each main architectural viewpoint

(e.g. incl. Critical paths, resource Consumption, ability to reset/restart, redundancy, safety…).

It should be built, negociated/validated with component suppliers.

This integration contract should mention (same as above)

·Expected services, functions

·Interfaces with other components and outside

·Contribution to the system/SW-wide information model (or global interchanged data)

·Dynamic behaviour expectations

·Requested other services, components… to be used by it

·Expected operational performances

·Non functional performances, expected Quality of Service

·Internal modes & states expected management

·Contribution to system management (surveillance, start-up/shutdown, redundancy issues…)

5.3.1.2.2 Build Product Breakdown Structure

a tree of Configuration Items (CI: either component software CSCI or hardware HWCI), built by

·grouping various former components in a bigger CI easier to manage,

·Or federating various similar components in a single implementation CI that will be instantiated multiple times at deployment.

Main contents of the PBS are physical components (behavioural and implementation), and physical links.

5.3.2 Perform Sub-System Engineering

Out of scope, but part of engineering activities may take benefit from applying Arcadia.

5.3.3 Perform HW & mechanical engineering

Out of scope, but part of engineering activities may take benefit from applying Arcadia.

5.3.4 Perform SW engineering

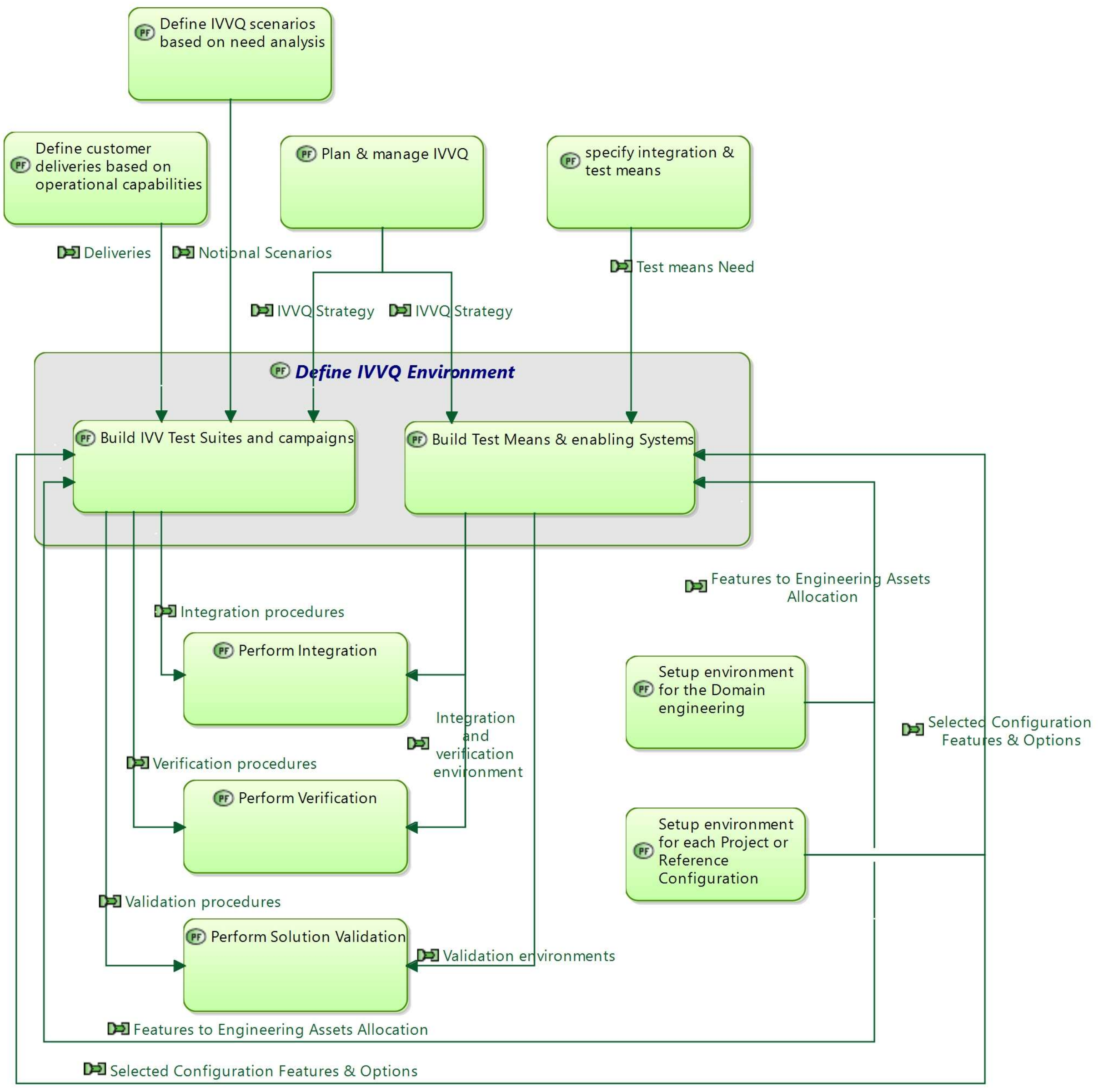
Out of scope, but part of engineering activities may take benefit from applying Arcadia.

5.3.5 Define IVVQ Environment

Based on IVV strategy information originated from model, design and develop test environment and etst campaigns in details.

See sub-tasks description

This figure describes the interactions of the considered task with other engineering activities.


Define IVVQ scenarios based on need analysis
Define customer deliveries based on operational capabilities
Plan & manage IVVQ
specify integration & test means
Deliveries
Notional Scenarios
Test means Need
IVVQ Strategy
IVVQ Strategy
Define IVVQ Environment
Build IVV Test Suites and campaigns
Build Test Means & enabling Systems
Features to Engineering Assets Allocation
Integration procedures
Perform Integration
Setup environment for the Domain engineering
Integration and verification environment
Selected Configuration Features & Options
Verification procedures
Perform Verification
Setup environment for each Project or Reference Configuration
Validation procedures
Perform Solution Validation
Validation environments
Features to Engineering Assets Allocation
Selected Configuration Features & Options

5.3.5.1 Build IVV Test Suites and campaigns

<Data elaboration description>

For each of the **IVV Steps** of the **IVVQ Strategy**, a set of **Test Campaigns** is defined in order to achieve its objectives (as mentioned in **Designed Solution Capabilities**).

Each of the **Test Campaigns** is composed of one or more **Test Suites**, each being in turn decomposed in **Test Cases**, consisting of elementary **Test Steps**.
Each of them is built based on the **Design Functions** and **Components Exchanges** between system **Components performing Functions** and **External systems/actors** (users, operators, other systems).

The interfaces are defined based on **Design Functional exchanges**, **Components Exchanges**, **Design Exchange contents**, etc.

At Integration stage, the definition of each of the **Test Suites** and **Test Cases** is mainly based on physical architecture **Design Scenarios** and **Design Functional Chains**.

At Verification stage, the definition of each of the **Test Suites** and **Test Cases** is driven by System need **Specified Scenarios** and **Spec. Functional Chains**.
Each of the system need **Specified Scenarios** is to be transformed into one (or more) system solution **Design Scenarios**; same for **Spec. Functional Chains** and **Design Functional Chains**. This is aided by traceability links between Need and Solution.

At Validation stage, the definition of each **Test Suites** and **Test Cases** is driven by Operational Need **Users Missions & Capabilities**, **Operational Scenarios**, and **Operational Processes**, along with System need **Specified Scenarios** and **Spec. Functional Chains**.
Each **Operational Scenarios** or **Specified Scenarios** is to be transformed into one (or more) system solution **Design Scenarios**; same for **Spec. Functional Chains** and **Design Functional Chains**. This is aided by traceability links between Need and Solution.

<Data elaboration description end>

5.3.5.2 Build Test Means & enabling Systems

<Data elaboration description>

Enabling/Test Means Definition is performed so as to specify their requested behaviour, along with their interfaces with system parts under test.
For each of the **IVV Steps**, the corresponding **Enabling/Test Means Definition** is performed, so as to version them, and to include them in the **IVV Configurations** of the **IVV Steps**.

In order to test some **Components performing Functions** in the **IVVQ Strategy**, the functional part of the system physical architecture which interacts with them is outlined.
This outline consists in **Design Functions**, **Design Functional Chains** parts and **Design Scenarios** to be implemented by Test Means for the current **IVV Releases**, based on the **Test Campaigns** to be performed.

Definition of interfaces between test means and system parts is based on this functional analysis, and on components interfaces definition : **Design Functional exchanges** and **Components Exchanges**, **Design Exchange contents**, **Physical Component Links** etc.

<Data elaboration description end>

5.3.6 Perform IVVQ

Out of scope, but investigation and bug-fixing activities may take benefit from exploiting Arcadia architecture models.

5.3.6.1 Perform Integration

<Data elaboration description>

Main outputs are **Test Results** and **Problem Reports** for each of the **Test Suites**, **Test Cases** and **Test Steps** previously defined.

Problem Reports can apply to most elements of solution description, mainly **Designed Solution Architecture** elements.

<Data elaboration description end>

5.3.6.2 Perform Verification

<Data elaboration description>

Similar to 'Perform Integration'. Main outputs are **Test Results** and **Problem Reports** for each of the **Test Suites**, **Test Cases** and **Test Steps** previously defined.

Problem Reports can apply to most elements of solution description, mainly **Designed Solution Architecture** elements, notably **Components performing Functions**.

Conformity Matrix will use the links between **Test Results**, **Test Cases** etc., and customer/User **Textual Requirements**, possibly through **Designed Solution Architecture** elements.

<Data elaboration description end>

5.3.6.3 Perform Solution Validation

<Data elaboration description>

Similar to 'Perform Integration' and 'Perform verification'. Main outputs are **Test Results** and **Problem Reports** for each of the **Test Suites**, **Test Case** and **Test Steps** previously defined.

Problem Report can apply to most elements of solution description, mainly **Designed Solution Architecture** elements, notably **Components performing Functions**.

Conformity Matrix will use the links between **Test Results**, **Test Cases** etc., and customer/User **Textual Requirements**, possibly through **Designed Solution Architecture** elements.

<Data elaboration description end>

5.3.6.4 Perform Qualification

Out of scope.

5.4 DEFINE AND EXPLOIT THE PRODUCT LINE

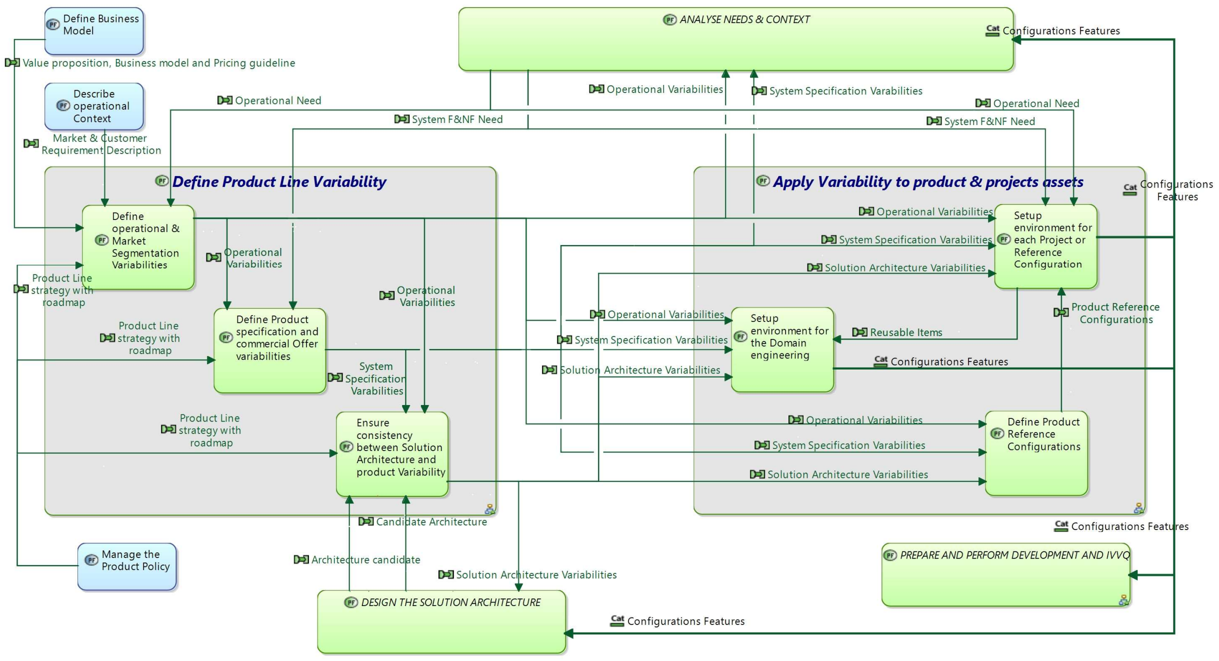
Identify the common and optional features that can ease and reduce cost of each product building while offering a relevant portfolio to customers; apply to engineering assets

See sub-tasks description

This figure provides an inner view of the task DEFINE AND EXPLOIT THE PRODUCT LINE, in the context of Arcadia core perspectives.

For each of the first level tasks presented here, only links with other core perspectives are displayed.

For a complete view of links with other engineering tasks and activities, see the figure dedicated to this first level task.


Define Business Model
Value proposition, Business model and Pricing guideline
Describe operational Context
Market & Customer Requirement Description
ANALYSE NEEDS & CONTEXT
Configurations Features
Operational Need
System F&NF Need
Operational Variabilities
System Specification Varabilities
Define Product Line Variability
Apply Variability to product & projects assets
Define operational & Market Segmentation Variabilities
Define Product specification and commercial Offer variabilities
Ensure consistency between Solution Architecture and product Variability
Product Line strategy with roadmap
Setup environment for each Project or Reference Configuration
Setup environment for the Domain engineering
Define Product Reference Configurations
Solution Architecture Variabilities
Reusable Items
Product Reference Configurations
Candidate Architecture
Architecture candidate
Manage the Product Policy
DESIGN THE SOLUTION ARCHITECTURE
PREPARE AND PERFORM DEVELOPMENT AND IVVQ

5.4.1 Define Product Line Variability

Segment users operational need analysis based on market segmentation; create a first set of operational variabilities (e.g. core generic missions, capabilities, contexts, and options).

Contribute to shape the offer and commercial options portfolio by defining variabilities on specification features expected from the solution, in accordance with former operational segmentation.

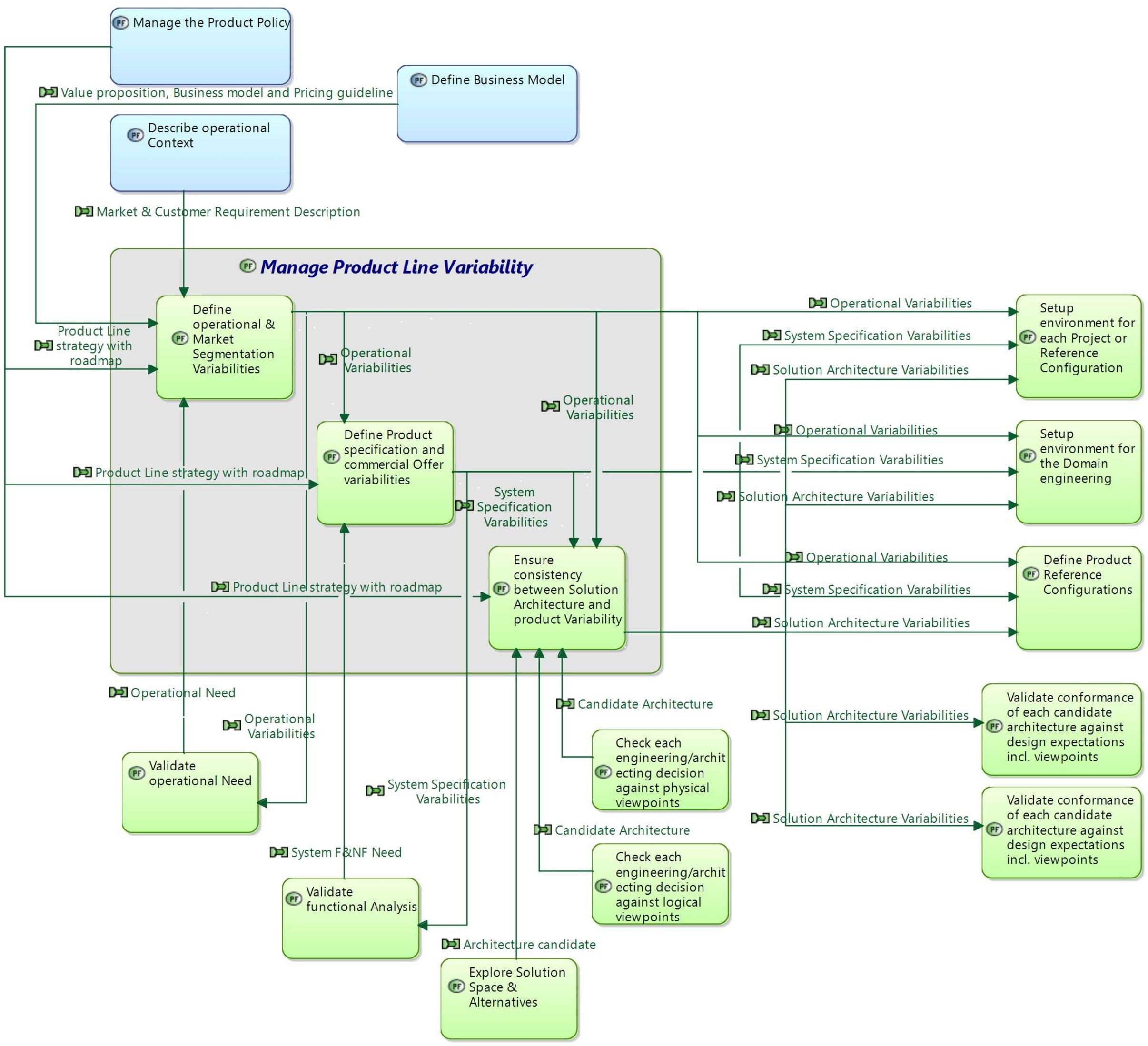
Analyse consequences of operational and specification variabilities on solution definition, incl. architecture;
seek for simplification of variabilities thanks to existing relations in architecture (e.g. select a feature based on a capability, rather than on related functions and components) ;
check for consistency between architecture and variabilities, adapt architecture if needed, revisit variabilities accordingly.

Formalise the variabilities at each perspective level, in an Engineering Feature Model describing:

• Commonality & variability: Mandatory Features, Options, Configurable Options, Alternatives,

• Variability Constraints.

This figure describes the interactions of the considered task with other engineering activities.


Manage the Product Policy
Define Business Model
Value proposition, Business model and Pricing guideline
Describe operational Context
Market & Customer Requirement Description
Manage Product Line Variability
Define operational & Market Segmentation Variabilities
Product Line strategy with roadmap
Operational Variabilities
Define Product specification and commercial Offer variabilities
Product Line strategy with roadmap
System Specification Varabilities
Ensure consistency between Solution Architecture and product Variability
Operational Variabilities
System Specification Varabilities
Solution Architecture Variabilities
Setup environment for each Project or Reference Configuration
Setup environment for the Domain engineering
Define Product Reference Configurations
Operational Need
Validate operational Need
System Specification Varabilities
Validate functional Analysis
System F&NF Need
Candidate Architecture
Check each engineering/architecting decision against physical viewpoints
Check each engineering/architecting decision against logical viewpoints
Architecture candidate
Explore Solution Space & Alternatives
Validate conformance of each candidate architecture against design expectations incl. viewpoints

5.4.1.1 Define operational & Market Segmentation Variabilities

Segment users operational need analysis based on market segmentation; create a first set of operational variabilities (e.g. core generic missions, capabilities, contexts, and options).

<Data elaboration description>

Starting from Product Line strategy with roadmap, Value proposition, Business model and Pricing guideline, the market segmentation is formalised by defining **Operational Segmentation Variability** by means of **Features**.

Most of these **Features** apply to different **Users Missions & Capabilities** that may be needed or optional for different users and markets. Selecting choices in a feature gives optional access to corresponding **Operational Processes** and **Operational Scenarios**, **Operational Activities**, etc.

<Data elaboration description end>

5.4.1.2 Define Product specification and commercial Offer variabilities

Contribute to shape the offer and commercial options portfolio by defining variabilities on specification features expected from the solution, in accordance with former operational segmentation.

<Data elaboration description>

In continuity with market segmentation, the major options and contents of the portfolio are formalised by defining **Portfolio Specification Variability** by means of **Features**. The main starting point for this is **Operational Segmentation Variability**, along with Product Line strategy with roadmap, Value proposition, Business model and Pricing guideline.

Most of the portfolio variability **Features** apply to different **Specified Capabilities** expected from the system for different users and markets. These **Features** should be initialised in accordance with those applied to **Users Missions & Capabilities** identified in **Operational Segmentation Variability**.

Selecting options or choices in a feature gives optional access to corresponding **Spec. Functional Chains** and **Specified Scenarios**, **Specified Functions**, or interactions with **External systems/actors**, etc. Their identification should also be initialised starting from **Operational Scenarios**, **Operational Processes**, etc., defined in **Operational Segmentation Variability** **Features**.

<Data elaboration description end>

5.4.1.3 Ensure consistency between Solution Architecture and product Variability

Analyse consequences of operational and specification variabilities on solution definition, incl. functional contents, behavior, interfaces, architecture components, simulation & test means, enabling systems, Test Campaignss, etc.;
seek for simplification of variabilities thanks to existing relations in architecture (e.g. select a feature based on a capability, rather than on related functions and components) ;
check for consistency between architecture and variabilities, adapt architecture if needed, revisit variabilities accordingly.

<Data elaboration description>

The definition of a design-oriented and design-driven **Solution Variability** starts from the need-related variabilities defined in **Portfolio Specification Variability**.

For each of the **Portfolio Specification Variability** **Features**, its "footprint" on the **Designed Solution Architecture** is determined using **System Need Specification** as an intermediate. This is done using links between **Features** and **Specified Capabilities**, then links between **Specified Capabilities** and **Designed Solution Capabilities**, between **Spec. Functional Chains** and **Design Functional Chains**, between **Specified Functions** and **Design Functions**, etc.

Then, **Features** constituting **Solution Variability** are created, some being derived from **Portfolio Specification Variability** ones, other being added for design considerations. These **Features** apply notably to **Designed Solution Capabilities** and related **Design Functions**, **Design Functional Chains**, **Design Scenarios**, and also on **Components performing Functions**, using links with **Design Functions**.

They are also likely to be "propagated" towards **Physical Hosting Components** according to their implementation of **Components performing Functions**.

Reference Configurations are also initialised to shape contents of the solution for each standard product offer. Their initial outline is based on former elements, defining **Variability Choices** among **Features** of the **Solution Variability** .

A verification of coherency should be done during all this process : for example, are the **Components performing Functions** designed so as to separate optional **Design Functions** according to **Features** definition ?.

Are all combinations of features possible due to architecture constraints ? Can several features be simplified into one because of architecture dependencies or constraints ?.

This may lead to modifying or simplifying **Features** of the **Solution Variability**.

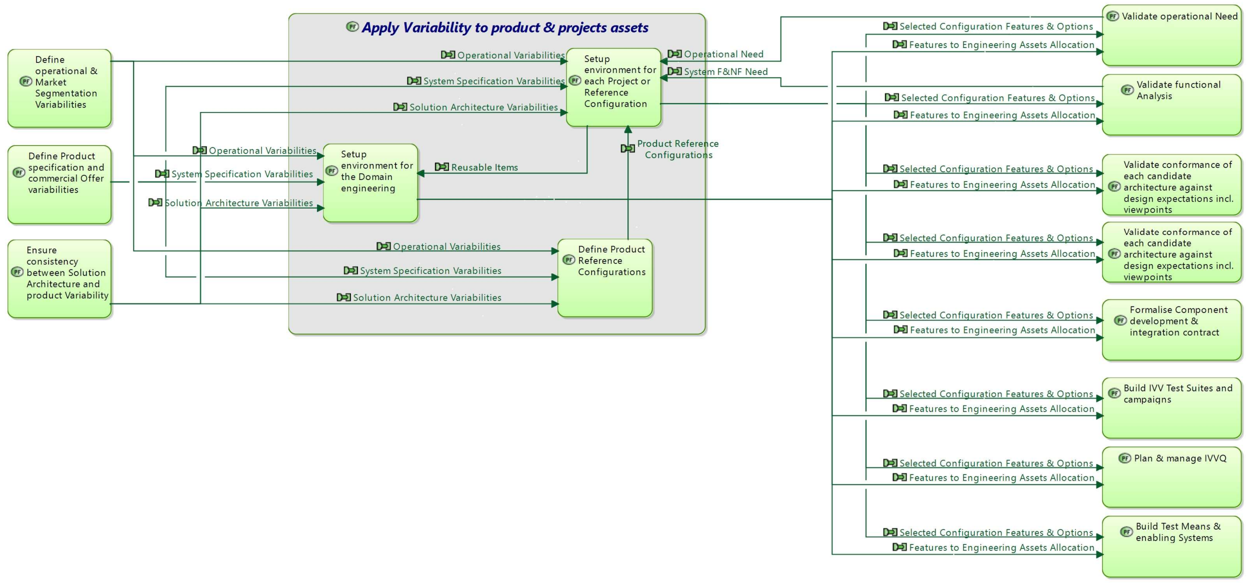
<Data elaboration description end>

5.4.2 Apply Variability to product & projects assets

For each variability identified in the product, analyse all engineering artefacts and assets, to determine if and how each of them is affected. Apply this approach to the product itself as a whole, then for each project according to selected variabilities.

In order to simplify the work for each project, define reference configurations that will select default sets of variabilities, according to the portfolio.

This figure describes the interactions of the considered task with other engineering activities.


Apply Variability to product & projects assets
Define operational & Market Segmentation Variabilities
Define Product specification and commercial Offer variabilities
Ensure consistency between Solution Architecture and product Variability
Setup environment for each Project or Reference Configuration
Setup environment for the Domain engineering
Define Product Reference Configurations
Operational Variabilities
System Specification Varabilities
Solution Architecture Variabilities
Reusable Items
Product Reference Configurations
Operational Need
System F&NF Need
Selected Configuration Features & Options
Features to Engineering Assets Allocation
Validate operational Need
Validate functional Analysis
Validate conformance of each candidate architecture against design expectations incl. viewpoints
Formalise Component development & integration contract
Build IVV Test Suites and campaigns
Plan & manage IVVQ
Build Test Means & enabling Systems

5.4.2.1 Setup environment for the Domain engineering

<Data elaboration description>

Features constituting **Solution Variability** are applied notably to **Designed Solution Capabilities** and related **Design Functions**, **Design Functional Chains**, **Design Scenarios**, and also on **Components performing Functions**, using links with **Design Functions**.

They are also likely to be "propagated" towards **Physical Hosting Components** according to their implementation of **Components performing Functions**.

Beyond the design and architecture description, most engineering data should be considered in this process, such as **Textual Requirements**, **Simulation Model Elements** and **Simulation Scenarios**, **Specialty Model elements** and **Specialty Analysis Context**, **Architecture viewpoint Models** **Viewpoint Analysis Context** and **Viewpoint Elements**, **Test Suites**, **IVV Releases**, **IVV Configurations**, **Enabling systems description Elements**, etc.

<Data elaboration description end>

5.4.2.2 Setup environment for each Project or Reference Configuration

<Data elaboration description>

The **Project Configuration** is built by performing **Variability Choices** on each of the **Features** present in the **Solution Variability**. As much as possible, this **Project Configuration** should be reusing **Reference Configurations**.

This results in selecting the engineering data needed for the project as referenced by the **Features** , among **Textual Requirements**, **Designed Solution Architecture**, **Simulation Models**, **Specialty Analysis Models**, **Test Suites**, **Enabling/Test Means Definition**, etc.

check for consistency and correctness;

capitalise reusable assets to be included in the product line.

<Data elaboration description end>

5.4.2.3 Define Product Reference Configurations

Define standard reference configurations to support the portfolio, according to market segmentation and commercial product line strategy.

Check consistency of each reference configuration with regards to engineering assets such as definition, architecture, etc.

<Data elaboration description>

Reference Configurations are defined, to shape contents of the solution for each standard product offer.They define **Variability Choices** among **Features** of the **Solution Variability**.

A verification of coherency should be done during all this process, regarding consistency, coherency, feasibility. This verification should apply on most engineering data addressed by the configuration, such as : **Textual Requirements**, **Designed Solution Architecture**, **Simulation Models**, **Specialty Analysis Models**, **Test Suites**, **Enabling/Test Means Definition**, etc.

<Data elaboration description end>

6