ARCADIA USER GUIDE

Arcadia Principles and Contents Overview

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 high level view of engineering activities defined and supported by Arcadia, their relations, and engineering information that they build or exploit.

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

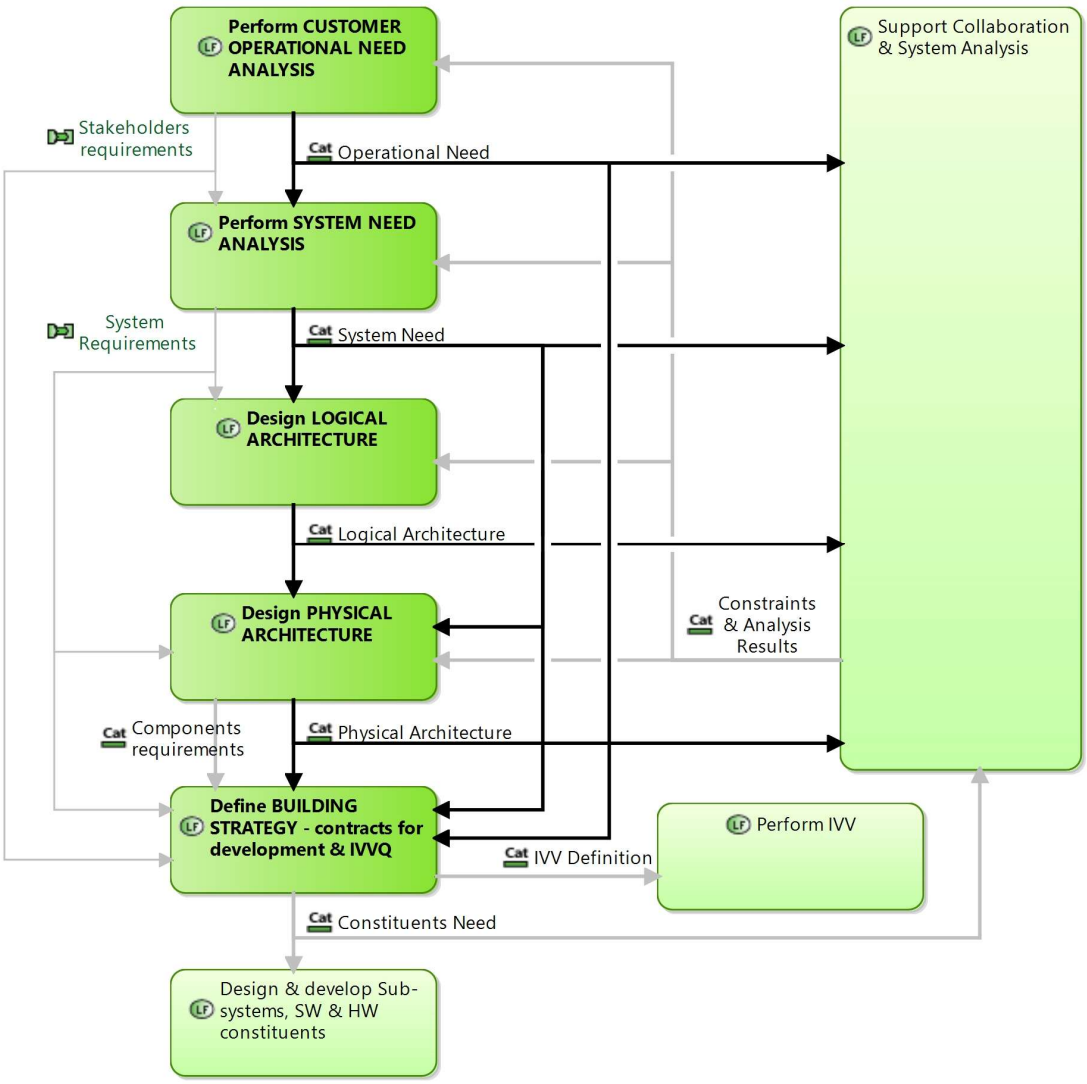
3 What is the Approach promoted by Arcadia?

Arcadia defines core engineering activities to analyse need and define solution architecture:

- OA: Operational Analysis;
- SA: System need Analysis;
- LA: Logical Architecture Design;
- PA: Physical ArchitectureDesign;
- BS: Building Strategy Definition

These core activities formalize their results in an architecture description and analysis model; each core activity builds a part of the model, called a perspective.

The figure below presents the Arcadia core activities (on the left), and other activities involving engineering stakeholders in a collaboration based on exploitation of Arcadia perspectives contents.



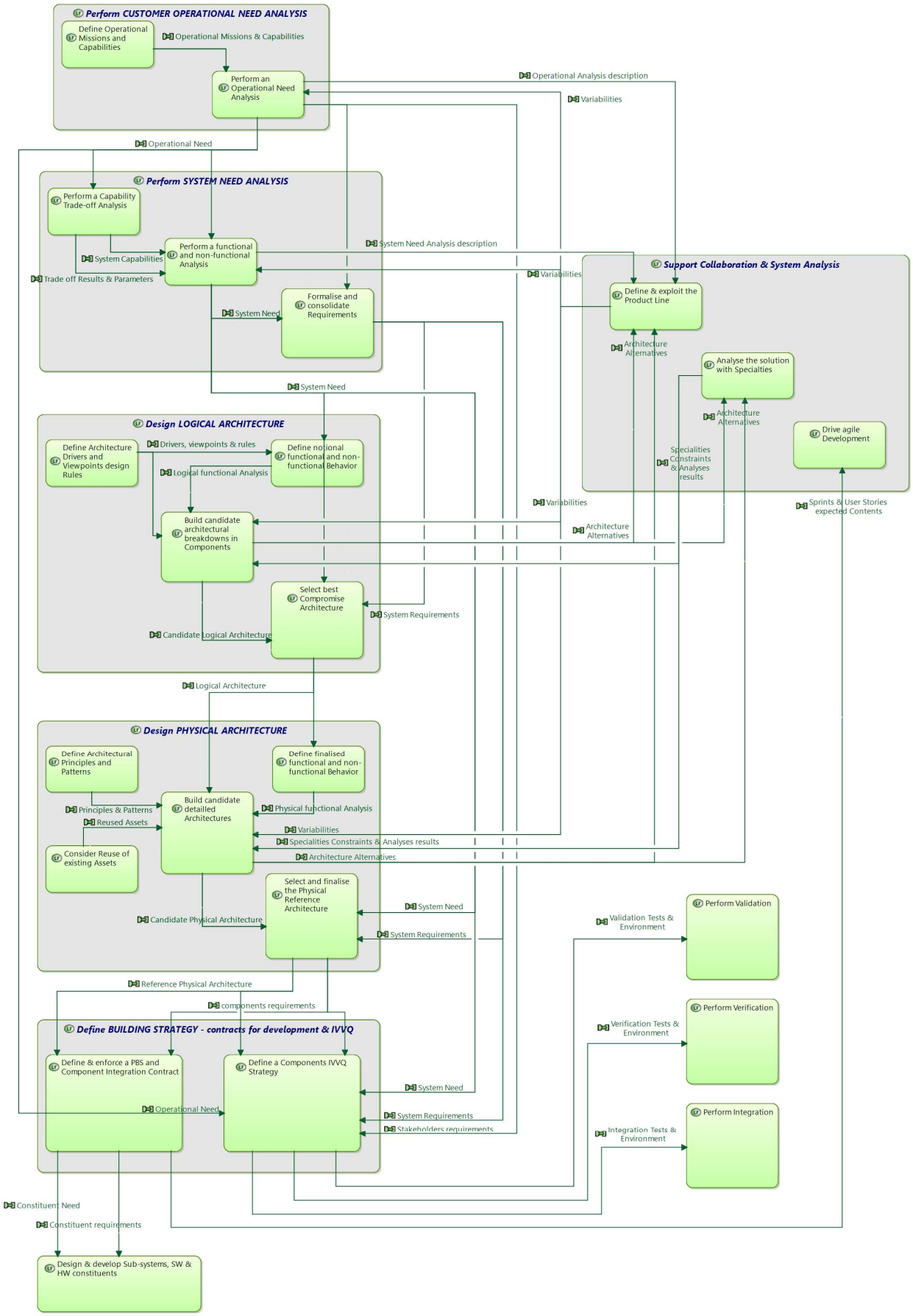
4 Which Engineering Tasks does Arcadia define and support?

This figure gives a view of major engineering tasks taking benefit from Arcadia, including the data that each core perspective provides in Arcadia models.

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,
- ...



Each activity is described below.

4.1 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.).

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:

Sub activities are described below.

4.1.1 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.

4.1.2 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.

4.2 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).

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:

Sub activities are described below.

4.2.1 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.

4.2.2 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.

4.2.3 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.

4.3 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.

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:

Sub activities are described below.

4.3.1 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

4.3.2 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.

4.3.3 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.

4.3.4 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.

4.4 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.

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	Architectural design &	

(logical, physical)	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	Validity of component breakdown
Systems engineering / IVVQ	
Test & Trials Strategy Plan	Definition of integration constraints
Integration means definition	Realism of integration 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:

Sub activities are described below.

4.4.1 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

4.4.2 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.

4.4.3 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

4.4.4 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.

4.4.5 Select and finalise the Physical Reference Architecture

Starting from the candidate architectures identified previously, use the same approach of best compromise selection as in logicial 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.

4.5 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	Selection of technologies	

	& derisking
PBS	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:

Sub activities are described below.

4.5.1 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.

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.

4.5.2 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.

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

4.6 Support Collaboration & System Analysis

This set of activities contributes to building, verifying and shaping the solution development, in a collaborative work between stakeholders. Please refer to sub-activities for more information.

Sub activities are described below.

4.6.1 Define & exploit the Product Line

Based on elaboration and analysis of each perspective model, Arcadia provides support to:

- segment market & customers expectations thanks to operational need analysis,

- formalise portfolio, including options offered to customers, thanks to system need analysis,
- justify and check product variability Vs need and solution architecture definition;
- allocate variability to model elements for each need and solution perspective;
- analyse consequences of variability on architecture (e.g. optional capabilities, functional chains etc.), and check for compatibility;
- drive variability definition, justification and checks based on architecture analysis;
- derive projects architecture and assets based on selected options, generating project models;
- secure reuse of existing components by multi-model confrontation.

4.6.2 Analyse the solution with Specialties

for each of the specialties (safety, security, RAMT etc.) likely to influence solution architecture, Arcadia perspectives allow to:

- Identify non-functional constraints (e.g. feared events, reliability expectations) on operational and system need;
- analyze solution logical and physical architectures based on specialty golden rules;
- feed specialty detailed analyses and tools using logical/physical architecture description;
- perform a multi-viewpoint approach assessing each candidate architecture against each specialty viewpoint, to find best architectural compromise in short loop, using architecture model analysis.

Notably regarding Supervision Engineering, Arcadia allows to:

- define operational and system modes and states, situations faced by the system;
- specify associated expected behaviour through functions, functional chains and scenarios configurations and expectations (e.g. reliability, resilience…), in each of the perspectives models, both at need and solution levels;
- define and allocate modes and states to components in logical/physical Architectures;
- analyze system behavior in critical situations and consequences of failures, startup and shutdown phases, reconfigurations, etc., by confronting situations superposing modes & states, associated functional behaviour, and availability status of components.

4.6.3 Drive agile Development

Arcadia drives agile development, using the solution architecture perspectives models:

- define contents of each increment, EPICs, User Stories, based on selected capabilités, functional chains and scenarios described in physical architecture, and versioned for each sprint;
- maintain a view of final target architecture view in physical architecture, and update it according to sprints results & evaluations

4.7 Design & develop Sub-systems, SW & HW constituents

Not described here.

Note: Arcadia could be applicable for need analysis and architecture definition, depending on the domaine and context.

4.8 Perform IVV

No Arcadia-related specificity. Refer to standard engineering practices.

Sub activities are described below.

4.8.1 Perform Integration

No Arcadia-related specificity. Refer to standard engineering practices.

4.8.2 Perform Verification

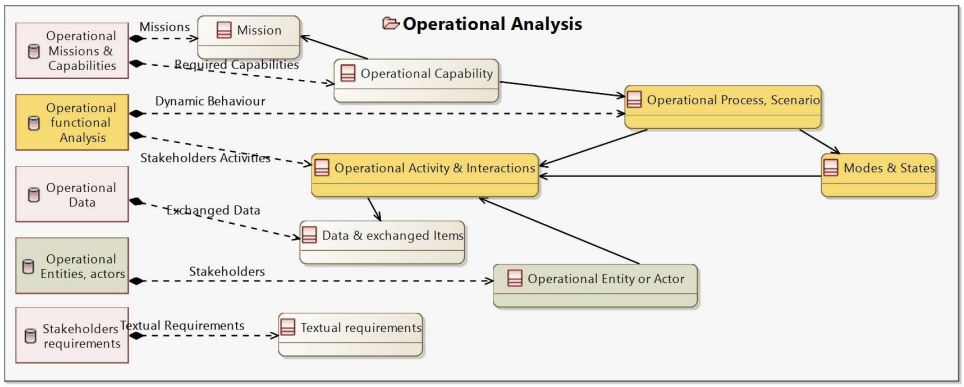
No Arcadia-related specificity. Refer to standard engineering practices.

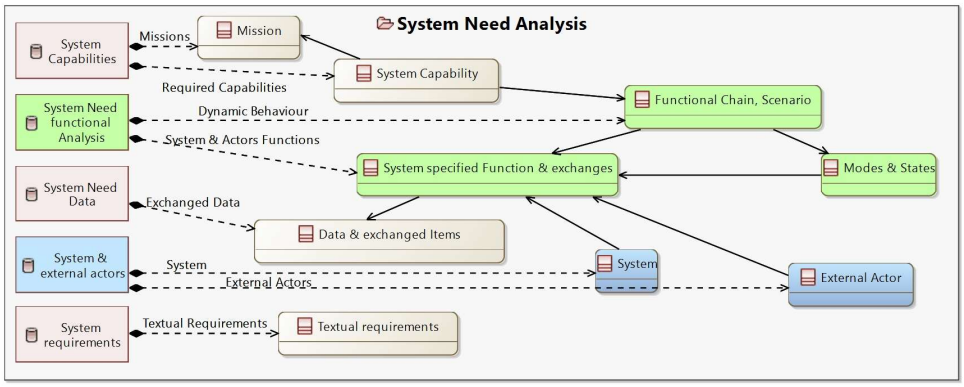
4.8.3 Perform Validation

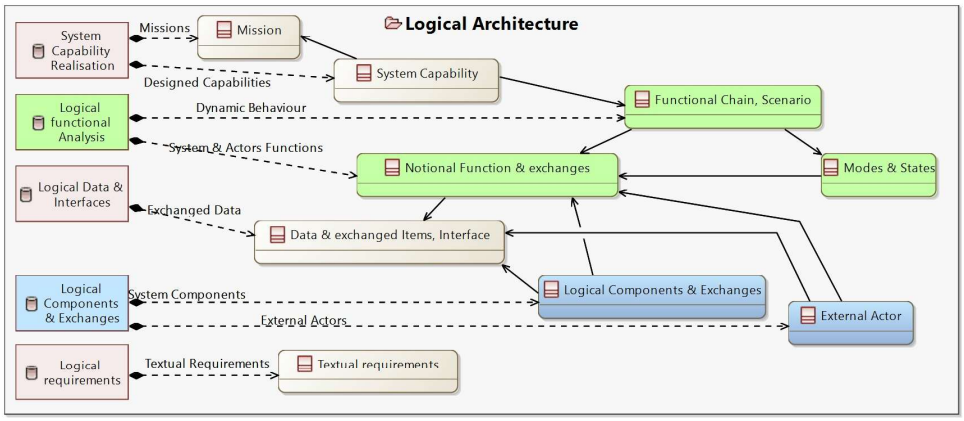
No Arcadia-related specificity. Refer to standard engineering practices.

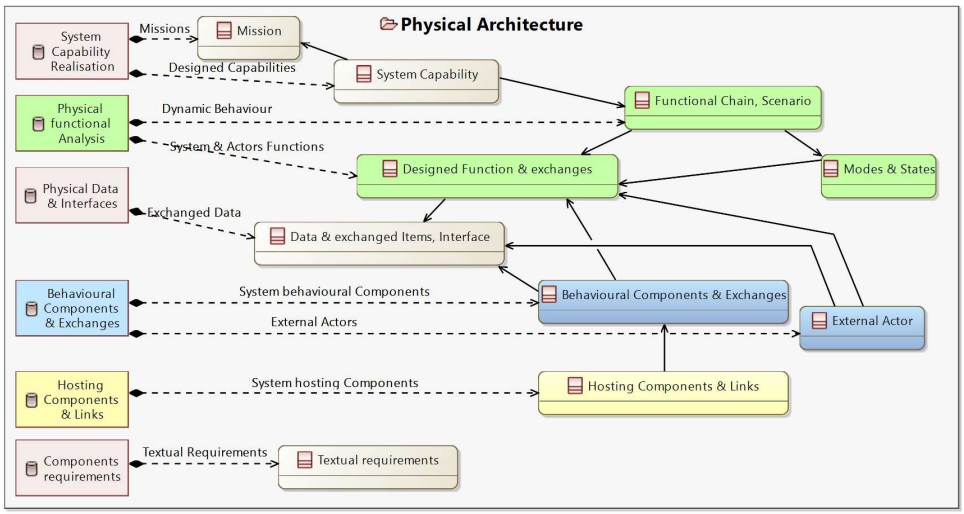
5 What are the major Concepts used by Arcadia?

This figure describes main concepts and links used in the engineering assets (models) defined and shared by each of the core Arcadia perspectives.







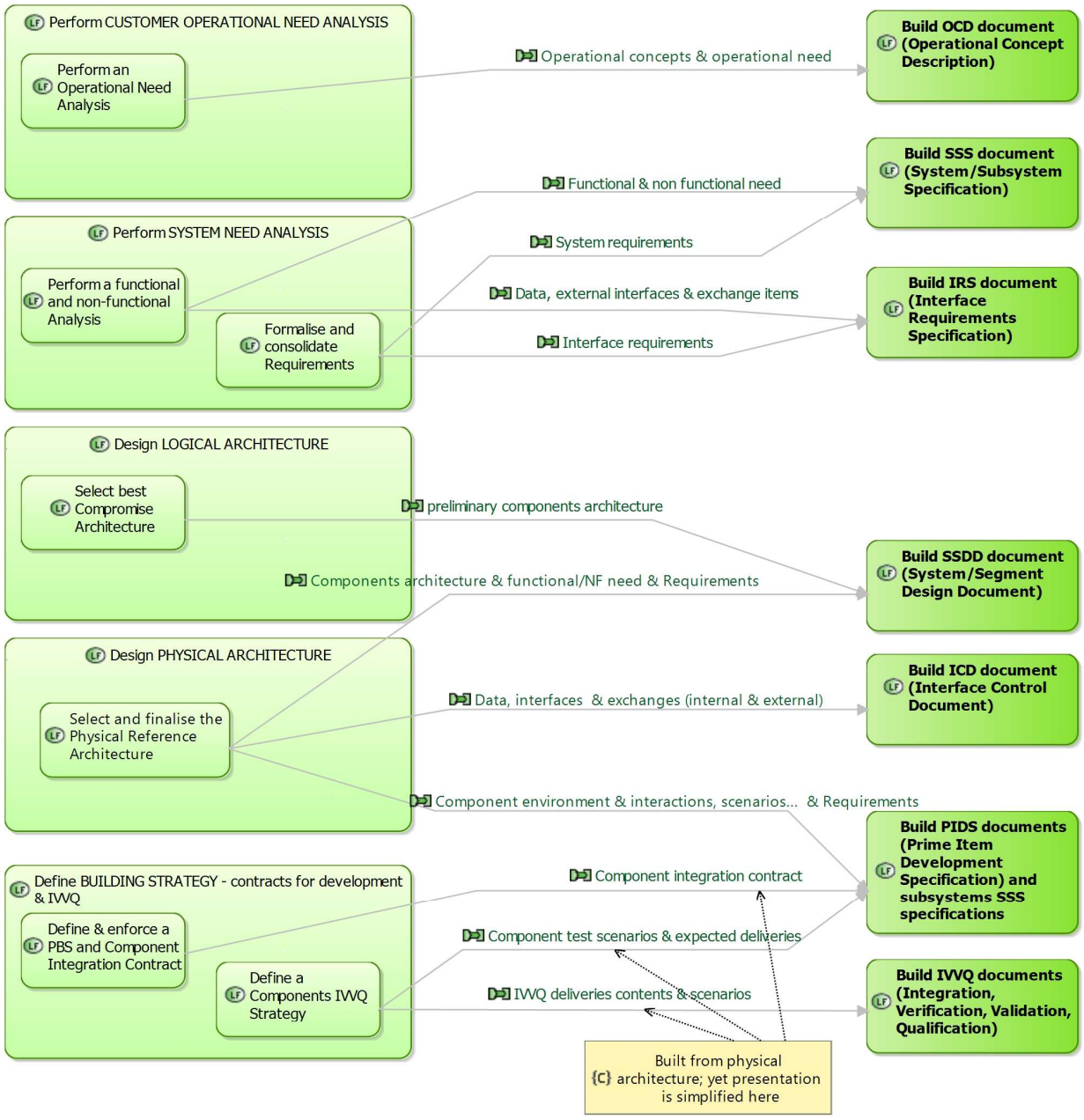


6 Documentation generation

This figure illustrates the use of data elaborated in core Arcadia perspectives, to populate main documents required by engineering standards.

More details on these data are accessible through the interactions (arrows) between tasks.

Note: simplified.



7