ARCADIA REFERENCE: ENGINEERING CAPABILITIES

Arcadia capabilities to address key Engineering Concerns

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 lists some major engineering stakes, goals or concerns, and the way Arcadia suggests to address them, under the form of capabilities and processes taking benefit from Arcadia activities to reach these goals.

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.

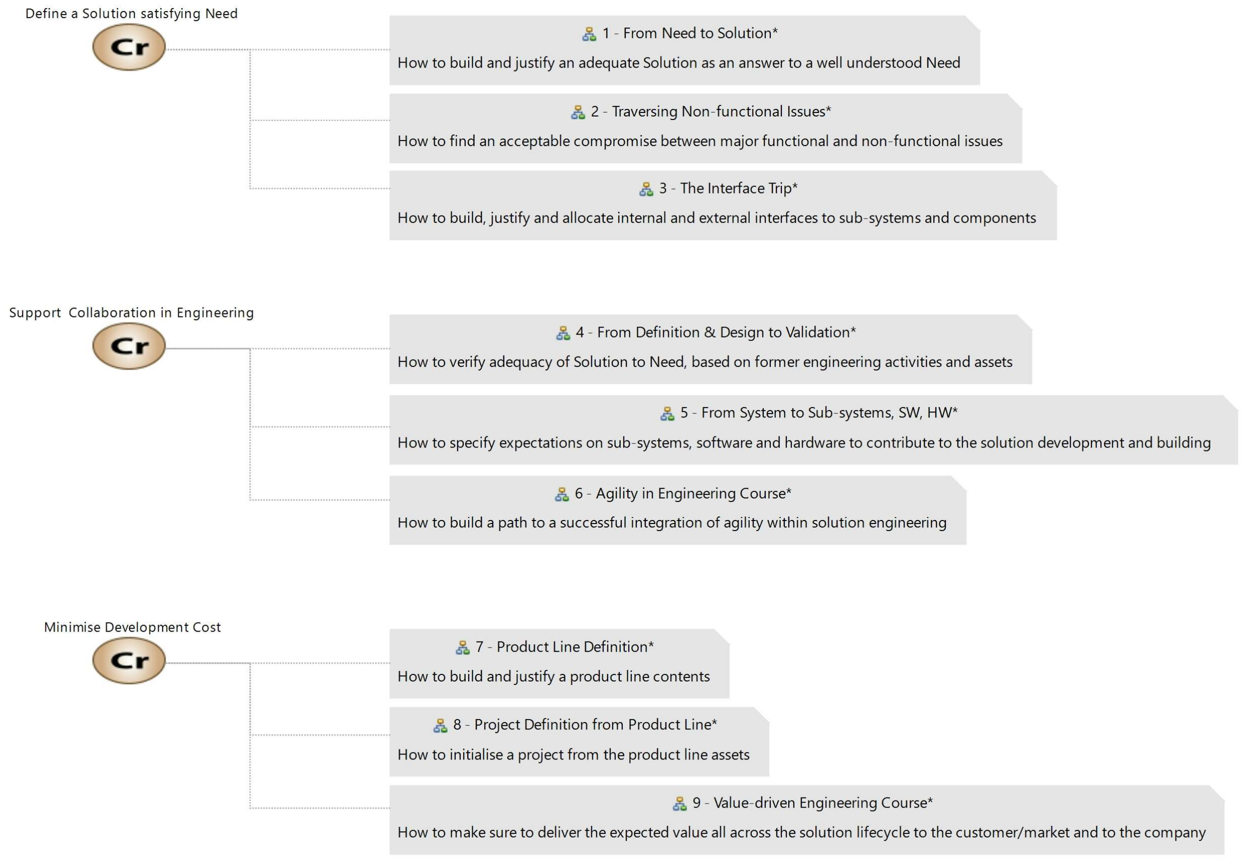
Summary of reference Documents Contents		Book	Landscape	User Guide	Reference - Activities	Reference - DataModel	Reference - Capabilities	Language - MetaModel	Q&A
History	Why was the method created and tooled? For which purpose? With which benefits?	✓							
Philosophy	What are its objectives and expected 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 Main key Stakes and Capabilities at a glance

This figure lists major engineering challenges (“How to…”) and required capabilities to address them (‘Cr’ on the left).

Arcadia suggests core workflows to address them, based on Arcadia perspectives and activities described in companion document ‘Arcadia Reference – Activities’. These workflows will be described in the rest of this document.



Define a Solution satisfying Need

1 - From Need to Solution*
How to build and justify an adequate Solution as an answer to a well understood Need

2 - Traversing Non-functional Issues*
How to find an acceptable compromise between major functional and non-functional issues

3 - The Interface Trip*
How to build, justify and allocate internal and external interfaces to sub-systems and components

Support Collaboration in Engineering

4 - From Definition & Design to Validation*
How to verify adequacy of Solution to Need, based on former engineering activities and assets

5 - From System to Sub-systems, SW, HW*
How to specify expectations on sub-systems, software and hardware to contribute to the solution development and building

6 - Agility in Engineering Course*
How to build a path to a successful integration of agility within solution engineering

Minimise Development Cost

7 - Product Line Definition*
How to build and justify a product line contents

8 - Project Definition from Product Line*
How to initialise a project from the product line assets

9 - Value-driven Engineering Course*
How to make sure to deliver the expected value all across the solution lifecycle to the customer/market and to the company

4 How Arcadia suggests to address these challenges

For each Arcadia capability, the workflows (or notional processes) suggested to address the related challenges are described below.

These workflows orchestrate Arcadia activities as described in document 'Arcadia reference – Activities'. The reader is invited to refer to the description of each activity prior to considering these workflows.

For each activity involved in a workflow, the specifics of its contribution to the workflow are described in the workflow context here. They complement the description of the activity in the 'Arcadia reference – Activities' document.

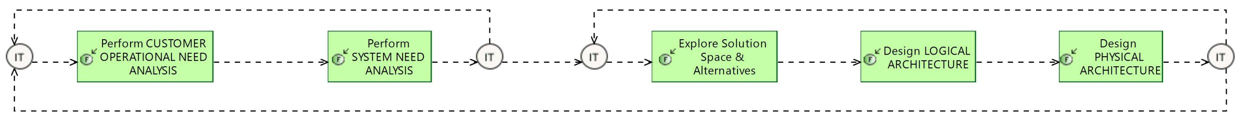
4.1 Define a Solution satisfying Need

The workflows supporting 'Define a Solution satisfying Need' are described below.

4.1.1 From Need to Solution

Challenge: How to build and justify an adequate Solution as an answer to a well understood Need.

- Clearly understand the end users need and the context of use of the system.
- Check requirements understanding and adequacy with them.
- Build a solution and check that it is compliant with these needs.


IT
Perform CUSTOMER OPERATIONAL NEED ANALYSIS
Perform SYSTEM NEED ANALYSIS
IT
IT
Explore Solution Space & Alternatives
Design LOGICAL ARCHITECTURE
Design PHYSICAL ARCHITECTURE
IT

4.1.1.1 Contribution of activity "Perform CUSTOMER OPERATIONAL NEED ANALYSIS "

Principles: Besides and before Requirement Management, drive an Operational Need Analysis, describing final user expectations & major conditions of operations.

Focus on analysing the customer needs and goals, expected missions & activities, far beyond System requirements. This is expected to ensure good adequacy of System definition with regards to its real operational use – and prepare Qualification conditions.

Outputs consist mainly in an 'operational architecture' describing and structuring this need, in terms of actors/users, their operational capabilities and activities, operational use scenarios giving dimensioning parameters, operational constraints including safety, security, system life cycle….

See also 'Explore Solution Space & Alternatives'

4.1.1.2 Contribution of activity "Perform SYSTEM NEED ANALYSIS"

Principles: Formalise Customer Requirements using a functional analysis approach to check their coherency, consistency and completeness, for sake of robustness and feasibility; take into account the former Operational Analysis, to ensure Usability.

Focus now on the system itself, in order to define how it can satisfy the former operational need, along with its expected behaviour and qualities: system functions to be supported & related exchanges, non functional constraints (safety, security…), performances allocated to system boundary, role sharing and interactions between system and operators…

Check also for feasibility (including cost, schedule and technology readiness) of customer requirements, and if necessary give means to renegotiate their contents. This may require, a first early system architecture, from system functional need; then requirements are confronted to this architecture in order to evaluate their cost and consistency.

Outputs mainly consist of system functional Need description, interoperability and interaction with the users and external systems (functions, exchanges plus non-functional constraints), and system requirements.

See also 'Explore Solution Space & Alternatives'

4.1.1.3 Contribution of activity "Explore Solution Space & Alternatives"

also refers to 'Understand Context and Needs' and 'Analyse Needs and formalise Requirements' :

First of all, we need to build with stakeholders (both internal and external) a shared vision of the goal, that is to say, a good understanding of what is at stake, where we would like to go, i.e. a rough shaping of the solution (whether it is a set of products or services, or a mix) where all major stakeholders are convinced that they reach a pretty good deal.

To state this balanced target, in others word, a engineered value proposition, we have to rely on a certain amount of knowledge and to deal with uncertainty. Some of this knowledge is already built and should be reused accordingly; some come from other sources (inputs from others Chorus processes), but some need to be built within the orientation in order to comprehend the various contexts (business context, technological context, operational context).

This requires distinguishing what really matters, the key topics and what really hurts, the critical topics, in order to focus our effort (usually the orientation activity uses resources sparingly) on important points of interest, postponing less important aspects until downstream activities.

At this level, in terms of solution, only the main concepts and principles, forming the (high level, abstract) architecture should be addressed. Obviously, there is no one unique set of concepts or principles fitting the targeted solution, but a deliberate choice among a wider and multiple set. Thus the ideal reached architecture should not only be described, but also justified.

Since orientation should also be seen as the top most upstream activity for all engineering activities, it should give some guidelines, and provide some strategic plan (part of e.g., the Engineering Plan, the IVVQ Plan…), to support further engineering activities.

4.1.1.4 Contribution of activity “Design LOGICAL ARCHITECTURE”

Principles: Structure system & build a notional (aka "logical") Architecture, by searching for best Compromise between [non-functional] Constraints & Viewpoints:
 consider each Viewpoint dealing with issues such as Functional Consistency, Interfaces, Performances, Real Time, Safety, Security, Integration, Reuse…

Identify the system parts (hereafter called *components*), their contents, relationships and properties, excluding implementation or technical/technological issues. This constitutes the system logical architecture.

In order for this breakdown in components to be stable in further steps, all major [non-functional] constraints (safety, security, performance, IVV…"Viewpoints") are taken into account so as to find the best compromise between them.

Outputs consist of the selected logical architecture: components & interfaces definition, including formalisation of all viewpoints and the way they are taken into account in the components design.

Since the architecture has to be validated against Need, links with requirements and operational functional chains and scenarios are also produced.

4.1.1.5 Contribution of activity “Design PHYSICAL ARCHITECTURE”

Principles: Ease and secure development & IVVQ through a finalised (aka "physical") Architecture dealing with viewpoints, technical & development issues, favouring separation of concerns, efficient and safe components interaction through components integration contracts (e.g. layered architecture, generic behaviour & interaction patterns, component model).

It defines the "final" architecture of the system at this level of engineering, ready to develop (by lower engineering levels). Therefore, it introduces rationalisation, architectural patterns, new technical services and components, according to implementation, technical & technological constraints & choices (at this level of engineering).

Note that a'Viewpoints-driven' architecture justification method is used for physical architecture definition.

Outputs consist of the selected physical architecture: components to be produced, including functional contents, behaviour, interfaces, complementary requirements, formalisation of all viewpoints and the way they are taken into account in the components design. Links with user and system need requirements and operational functional chains / scenarios are also produced.

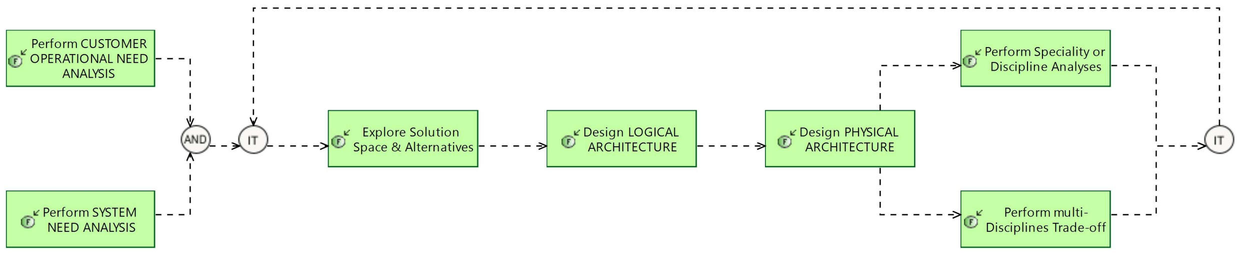
4.1.2 Traversing Non-functional Issues

Challenge: How to find an acceptable compromise between major functional and non-functional issues (1).

- Analyse and characterise expectations for each viewpoint in the need descriptions
- when designing solution architecture, submit each alternative and design decision to analysis of consequences for each viewpoint, in order to reach the best compromise between all
- once acceptable architecture orientations are reached, perform fine grain analysis and simulations for each viewpoint and expectation, in order to confirm compromise choices

(1) Examples of non-functional issues are: safety, [cyber]security, reliability, dependability, maintainability, durability, sustainability, integrability, testability, size weight and power, etc... and of course performances and product line.

The way each of these issues is considered and adressed in engineering, notably by related disciplines and specialities, is called a "viewpoint" on engineering.


Perform CUSTOMER OPERATIONAL NEED ANALYSIS
Perform SYSTEM NEED ANALYSIS
AND
IT
Explore Solution Space & Alternatives
Design LOGICAL ARCHITECTURE
Design PHYSICAL ARCHITECTURE
Perform Speciality or Discipline Analyses
Perform multi-Disciplines Trade-off
IT

4.1.2.1 Contribution of activity “Perform CUSTOMER OPERATIONAL NEED ANALYSIS ”

Operational Analysis is the place to analyse, for each non-functional viewpoint, its context, constraints, opportunities, critical use cases, regulations, etc.

For example, for safety viewpoint, feared events should be identified along with related operational scenarios and operational consequences to evaluate their criticality;
 Security viewpoint will focus on primary assets to be protected, threats and threat sources, feared events, all related to operational processes and scenarios.

Beside this analysis work, operational simulation can help illustrate operational processes and scenarios captured in the operational analysis; it can be fed from these, and in turn can valuate or complement the analysis itself with simulation results.

4.1.2.2 Contribution of activity “Perform SYSTEM NEED ANALYSIS”

Beyond customer requirements and regulations refering to each viewpoint (safety insurance level, reliability figures, cyber-security and confidentiality level, etc.), non-formalised constraints will be allocated to the elements constituting the system need functional analysis : e.g. level of criticity of a function, functional chains likely to produce a feared event, required reconfiguration functions for dependability, rainy days scenarios when facing a threat, etc.

Along with this system need analysis formalisation, simulations may complement the need understanding and description, the expected system behaviour when facing undesired events, etc.

4.1.2.3 Contribution of activity “Explore Solution Space & Alternatives”

Each major non-functional viewpoint will shape definition of solution alternatives, for example by suggesting state-of-the-art architectural patterns to cope with non-functional constraints. It will also contribute to evaluation and choice criteria, to early eliminate alternatives that will not fulfill related constraints identified in need analysis.

Functional or behavioural simulation means are useful to detect flaws and inadequacies in candidate alternatives, as early as possible.

4.1.2.4 Contribution of activity “Design LOGICAL ARCHITECTURE”

The (almost) final compromise between former analyses in 'Explore Solution Space & Alternatives', 'Perform multi-Disciplines Trade-off' and 'Perform Speciality or Discipline Analyses' is obtained in an iterative manner, each detection of issue or flaw leading to reconsidering the global compromise.

When all analyses converge towards one proven satisfactory design, justification files can be formalised, and the architecture design be finalised.

4.1.2.5 Contribution of activity “Design PHYSICAL ARCHITECTURE”

Architecture design is finalised only when the former analyses have given expected results verifying that the obtained compromise is satisfactory.

4.1.2.6 Contribution of activity “Perform multi-Disciplines Trade-off”

When the number of alternatives has been reduced by 'Explore Solution Space & Alternatives' activity, the remaining ones are evaluated in a more precise manner, against most important fucntional and non-functional viewpoints.

Each engineering or architecture decision is to be submitted to multi-viewpoint analysis, so as to check that satisfyong one of them does not impede or degrade others (e.g. grouping functions on one execution node to optimise performance, could lead to introducing new common mode failure in the safety domain).

In order to be efficient, this analysis requires a short loop timing, so as to apply it at each major definition and design choice or decision. This means that the level of detail of architecture description and of analyses will stay at a relatively coarse grain. More in-depth analyses will be considered later in 'Perform Speciality or Discipline Analyses'.

This approach mainly uses multi-purpose architecture analysis techniques and tools, in which constraints and golden rules of each discipline and speciality are checked accordingly.

4.1.2.7 Contribution of activity “Perform Speciality or Discipline Analyses”

Once the candidate solution architectures are reduced to just one or a few, 'Perform multi-Disciplines Trade-off' ensures that a coarse grain, multi-viewpoint analysis, definition has reached a good compromise between all major viewpoints. But this coarse-grain analysis must be confirmed and deepened for each discipline and speciality, applying state-of-the-art approaches.

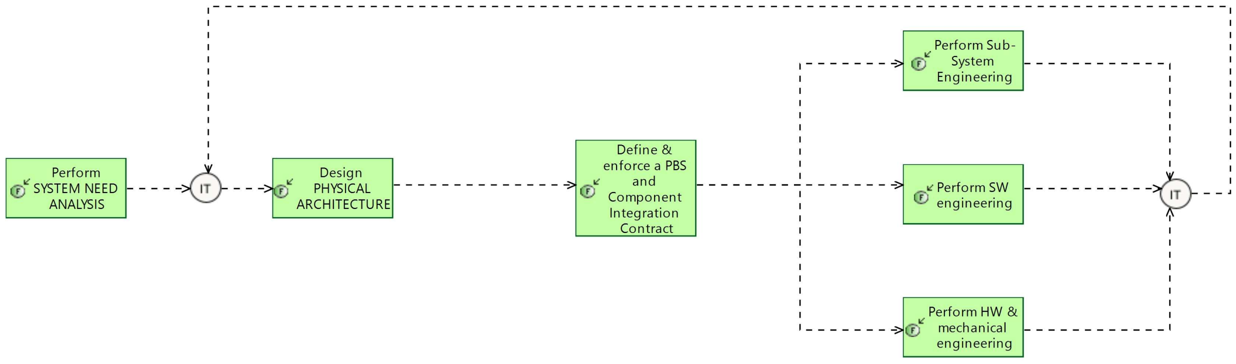
This is done in each domain, with fine-grain architecture and design descriptions, focused on one single viewpoint mainly; it uses dedicated speciality engineering methods and tools, along with simulation environments and models. These analyses and tools should be initialised by the common designed architecture description, sio as to ensure coherency and completeness of analyses.

Note that this activity can also be run at any time when questions or issues arise regarding disciplines and specialities, including in early analysis if needed.

4.1.3 The Interface Trip

Challenge: How to build, justify and allocate internal and external interfaces to sub-systems and components.

- Check consistency of imposed external interfaces description, and system need/solution definition
- Build and justify internal interfaces definition upon functional analysis, exchanges, functional chains etc.
- Enforce compliance of sub-systems/SW/HW parts with these interface through automatic generation of their requirements and interfaces definitions


Perform SYSTEM NEED ANALYSIS
IT
Design PHYSICAL ARCHITECTURE
Define & enforce a PBS and Component Integration Contract
Perform Sub-System Engineering
Perform SW engineering
Perform HW & mechanical engineering
IT

4.1.3.1 Contribution of activity "Perform SYSTEM NEED ANALYSIS"

From the Interface Maangement point of view, Need Analysis mainly focuses on checking compatibility of external interfaces between the system of interest and other systems or actors.

For each external system whose interface is imposed, check that all functions, functional chains and scenarios allocated to the system have corresponding elements in the external system interface definition : exchanged elements and data, time-related protocols if any, performance or non-functional constraints, etc. Update function, functional chains and scenarios definition if needed.

From system users / operators point of view, similarly capture their expectations in terms of nature of expected interactions with the system, state-of-the-art practices, preferred modalities, expected information delivery, eetc.

4.1.3.2 Contribution of activity “Design PHYSICAL ARCHITECTURE”

Internal interfaces definition (and external interfaces not imposed by the customer, including user interfaces) is based on functional analysis of the solution architecture.

The solution functional analysis describes its designed behaviour in terms of functions and their interactions/functional exchanges, functional chains and scenarios defining the way these functions and exchanges are involved in given situations, capabilities etc. Elements exchanged are also defined in terms of structure, contents, data.

The architecture building process allocates functions to system components, and to external systems/actors/users. Therefore, the nature and contents of interfaces between solution components (internal interfaces) and external systems/users (external interfaces) are fully defined by the functional exchanges at boundaries of each component, along with their contents and dynamic behaviour from functional chains and scenarios.

This conceptual, functional interface definition can then be optimised from an implementation and design point of view (e.g. by grouping exchanges, structuring exchange contents, adding technology-related representations, etc.). this results in precisely defining the interface contents to be developped and respected for each component. This implementation definition is fully explained and justified by the former functional description to which it is linked.

The physical support of these communication and interface elements is also to be defined in terms of physical links that will carry them, and to which they will be allocated.

4.1.3.3 Contribution of activity “Define & enforce a PBS and Component Integration Contract”

The definition of internal and external interfaces made in the 'Design Solution Architecture' activity is a major part of the contract towards sub-systems, software and hardware engineering. It should be elaborated in a collaborative manner between all engineering teams concerned, so as to integrate constraints of each team. Once approved, interfaces definition is the single and common reference for all stakeholders.

4.1.3.4 Contribution of activity “Perform Sub-System Engineering”

External interfaces of each sub-system should not be modified by the sub-system engineering team alone. If modifications are required, they should be collaborative, under responsibility of the system-level engineering team, along with other sub-systems, software, hardware engineering. Once a new reference interface definition is produced and validated by all stakeholders, it will be promoted and be imposed to all.

4.1.3.5 Contribution of activity "Perform SW engineering"

External interfaces of each software component or sub-system should not be modified by the software engineering team alone. If modifications are required, they should be collaborative, under responsibility of the system-level engineering team, along with other sub-systems, software, hardware engineering. Once a new reference interface definition is produced and validated by all stakeholders, it will be promoted and be imposed to all.

4.1.3.6 Contribution of activity "Perform HW & mechanical engineering"

External interfaces of each hardware or mechanical component or sub-system should not be modified by the component engineering team alone. If modifications are required, they should be collaborative, under responsibility of the system-level engineering team, along with other sub-systems, software, hardware engineering. Once a new reference interface definition is produced and validated by all stakeholders, it will be promoted and be imposed to all.

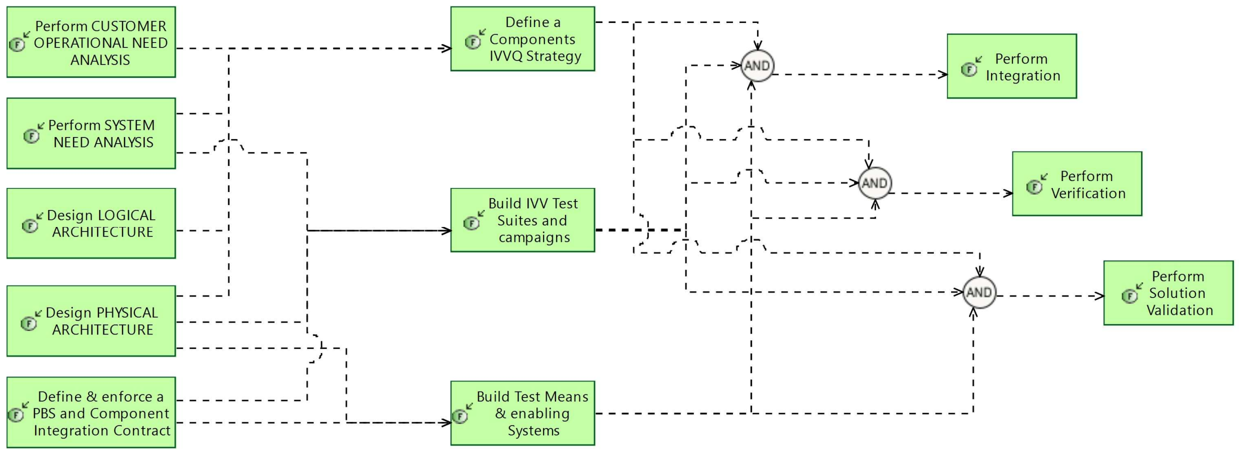
4.2 Support Collaboration in Engineering

The workflows supporting 'Support Collaboration in Engineering' are described below.

4.2.1 From Definition & Design to Validation

Challenge: How to verify adequacy of Solution to Need, based on former engineering activities and assets.

- Build the IVV strategy based on operational and system need analysis and customer value analysis to deliver appropriate capabilities to customer.
- Define test campaigns and test cases according to these capabilities, based on functional chains and scenarios describing these capabilities.
- Use links between need and solution descriptions to justify the strategy, and compliance with user requirements.


Perform CUSTOMER OPERATIONAL NEED ANALYSIS
Perform SYSTEM NEED ANALYSIS
Design LOGICAL ARCHITECTURE
Design PHYSICAL ARCHITECTURE
Define & enforce a PBS and Component Integration Contract
Define a Components IVVQ Strategy
Build IVV Test Suites and campaigns
Build Test Means & enabling Systems
AND
AND
AND
Perform Integration
Perform Verification
Perform Solution Validation

4.2.1.1 Contribution of activity “Perform CUSTOMER OPERATIONAL NEED ANALYSIS ”

The IVV strategy will order deliveries according to the customer operational Capability acquisition roadmap described in Operational Analysis.

Operational Analysis will also shape Validation Test campaigns, notably using operational scenarios and mission description.

4.2.1.2 Contribution of activity “Perform SYSTEM NEED ANALYSIS”

The IVV strategy will order deliveries according to the customer value and priorities for each System Need capability and most important functions expected from the system.

System Need Analysis will also shape verification and Validation Test campaigns, notably using System Need scenarios and System Usage as agreed with the customer.

4.2.1.3 Contribution of activity “Design LOGICAL ARCHITECTURE”

Definition of the solution architecture should take into account IVVQ constraints, such as ease of progressive building and verification, observability of the behaviour, etc.

Architecture-related conscerns such as performance verification, complexity and technical risk, might also impact the IVV Strategy.

4.2.1.4 Contribution of activity “Design PHYSICAL ARCHITECTURE”

The detailled design of the solution architecture feeds IVV configurations definition. Achitectural, functional and technical dependencies may influence and constrain IVV strategy and roadmap.

Functional chains and scenarios defined in the solution architecture are the basis for integration and verification test campains and test cases. Those of the functional chains and scenarios that are based on operational and system need analyses will be the reference and starting point of verification and validation tests as well.

Test means definition is based on the architecture description, in terms of functional contents, interfaces, use scenarios, as described in the solution architecture.

4.2.1.5 Contribution of activity “Define & enforce a PBS and Component Integration Contract”

The definition of subsystems technical contract will include definition of tests campaigns requested from subsystems, according to those architecture functional chains and scenarios that are allocated to them.

Tests means specification is driven by the architecture description : interfaces to implement, components functional contents and behaviour, fu ntional chains and scenarios to enable, etc.

4.2.1.6 Contribution of activity “Define a Components IVVQ Strategy”

IVV Strategy is built on User and Solution Capabilities roadmap, under value analysis-originated criteria. Functional chains and scenarios describing these capabilities bridge them with functional contents and components required at each IVV step, to build the expected release configurations.

4.2.1.7 Contribution of activity “Build IVV Test Suites and campaigns”

IVV Tests are built based on users operational missions and capabilities descriptions by means of operational processes and scenarios. These are transposed to System Need analysis by describing expected contribution of the system to them, resulting in need-level functional chains and scenarios.

When solution architecture is defined, these functional chains and scenarios are transformed to solution-level ones, dealing with designed components functional behaviour and interfaces.

Tests cases are built from these functional chains and scenarios.

Note that Simulation and Specialities scenarios and models can also contribute to tests definition and solution verification.

Validation preparation tasks also take benefit from need analysis and solution definition. Using an approach similar to 'Define IVV Test Suites and campaigns'.

Validation use cases are mainly based on operational and system need analysis functional chains and scenarios.

4.2.1.8 Contribution of activity “Build Test Means & enabling Systems”

Tests means specification is driven by the architecture description : interfaces to implement, components functional contents and behaviour, fu ntional chains and scenarios to enable, etc.

Note that Simulation and Specialities scenarios and models can also contribute to tests means and enabling systems definition and solution verification.

4.2.1.9 Contribution of activity “Perform Integration”

Integration will be easier thanks to not only relying on requirements only, but also and mostly on architecture : better mastering of functional behaviour thanks to functional chains and scenarios, better default and problem analysis in identifying components flaws, etc.

4.2.1.10 Contribution of activity “Perform Verification”

Similar to Integration, verifying that user-oriented functional chains and scenarios are fulfilled.

4.2.1.11 Contribution of activity “Perform Solution Validation”

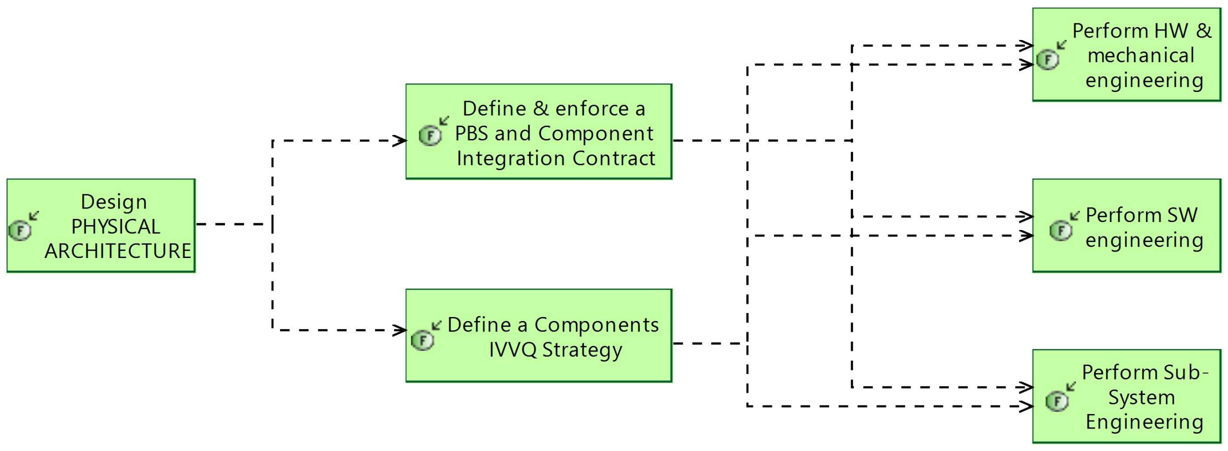
similar to verification.

4.2.2 From System to Sub-systems, SW, HW

Challenge: How to specify expectations on sub-systems, software (SW) and hardware (HW) to contribute to the solution development and building.

- Build the solution architecture in co-engineering between system, sub-systems/SW/HW teams, and specialities/disciplines.

- Once a consensus is built, and its hypotheses checked by all stakeholders, extract the specification of each sub-system/SW/HW part from the solution architecture
- Define the contribution of each part and team to IVV strategy and test cases the same way, and add IVV expectations to the parts specification


Design PHYSICAL ARCHITECTURE
Define & enforce a PBS and Component Integration Contract
Define a Components IVVQ Strategy
Perform HW & mechanical engineering
Perform SW engineering
Perform Sub-System Engineering

4.2.2.1 Contribution of activity “Design PHYSICAL ARCHITECTURE”

The solution architecture design results in the definition and description of components, their functional contents and behaviour, their interfaces, allocated functional chains and scenarios, resources, etc. plus textual requirements describing or complementing them.

This is the main input to defining the need of each sub-system, SW or HW item, as a (set of) component(s).

4.2.2.2 Contribution of activity “Define & enforce a PBS and Component Integration Contract”

As mentioned in 'Design Solution Architecture',the specification contract of each sub-system, software or hardware item is mostly extracted from the system architecture : description of components, their functional contents and behaviour, their interfaces, allocated functional chains and scenarios, resources, etc. plus textual requirements describing or complementing them.

This 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.

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 summarised and checked here.

Outputs from this step are mainly "component Integration contract" collecting all necessary expected properties for each component to be developed, and building & IVVQ Strategy.

4.2.2.3 Contribution of activity "Define a Components IVVQ Strategy"

The definition of the system-level IVV Strategy usually includes delegating part of verifications to sub-systems, when appropriate and confinable into one (possibly more) of them. The specification of delegated tests and verifications uses mainly allocation of all or parts of system-level functional chains and scenarios to sub-systems.

4.2.2.4 Contribution of activity "Perform HW & mechanical engineering"

Hardware and mechanical engineering initial tasks (such as need analysis, general architecture, integration and verification policy, etc.) may be similar to the system-level one, properly adapted, tuned and taylored.

4.2.2.5 Contribution of activity "Perform SW engineering"

Software engineering structuring tasks (such as need analysis, general architecture, integration and verification policy, etc.) may be similar to the system-level one, properly adapted, tuned and taylored.

Specific software-related issues such as agile practices can also benefit from the system approach. See 'Agility in engineering course'.

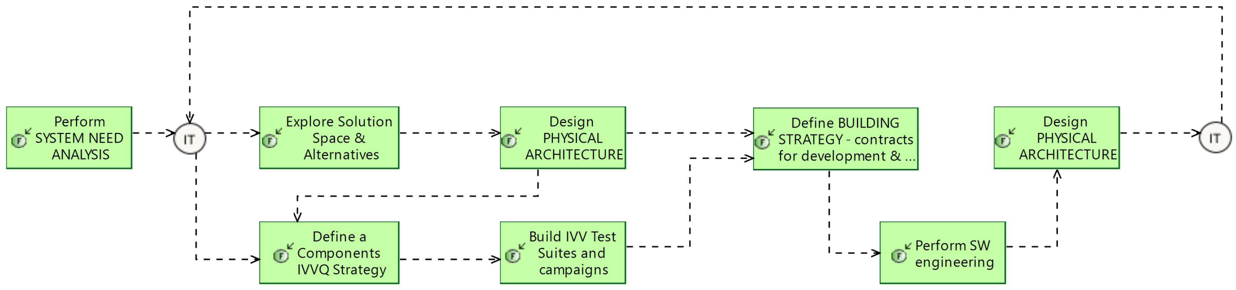
4.2.2.6 Contribution of activity "Perform Sub-System Engineering"

Sub-system engineering structuring tasks (such as need analysis, general architecture, integration and verification policy, etc.) may be similar to the system-level one, properly adapted, tuned and taylored.

4.2.3 Agility in Engineering Course

Challenge: How to build a path to a successful integration of agility within solution engineering.

- Define contents of agile increments based on valued capabilities
- Analyse impact of each one and preserve an (evolutive) vision of final solution architecture
- Align increments contents with IVV strategy and scheduling
- Maintain both incremental and final visions aligned


Perform SYSTEM NEED ANALYSIS
IT
Explore Solution Space & Alternatives
Design PHYSICAL ARCHITECTURE
Define BUILDING STRATEGY - contracts for development & ...
Design PHYSICAL ARCHITECTURE
IT
Define a Components IVVQ Strategy
Build IVV Test Suites and campaigns
Perform SW engineering

4.2.3.1 Contribution of activity "Perform SYSTEM NEED ANALYSIS"

Capabilities or Functional Chains describing customers/users expectations in System Need Analysis are a good support to both capture their expected value, and drive agile increments contents in a value-driven approach. Each of them should be valued, and development ordered according to this expected value (e.g. highest Business Value, highest Criticality first).

4.2.3.2 Contribution of activity "Explore Solution Space & Alternatives"

Build the solution in an incremental way based on value creation, using end-to-end Functional Chains and Scenarii. Develop the necessary and sufficient engineering artefacts in an iterative & incremental way.

4.2.3.3 Contribution of activity "Design PHYSICAL ARCHITECTURE"

Maintain a vision of the architecture and product at completion, and confront each iteration to it.

Update the vision according to evolution of engineering and development.

4.2.3.4 Contribution of activity “Define a Components IVVQ Strategy”

IVV Strategy will use the definition of increments and sprints based on functional chains and scenarios, and sequence IVV activities accordingly.

Conversely, IVV constraints or opportunities may influence definition of increments.

4.2.3.5 Contribution of activity “Build IVV Test Suites and campaigns”

Test Campaigns and Test Cases are based on Capabilities, functional chains and scenarios feeding agile software development, hence securing coherency of IVV with system architecture and software.

4.2.3.6 Contribution of activity “Define BUILDING STRATEGY - contracts for development & IVVQ”

Define increments at solution engineering level, based on Capabilities and Value Analysis.

Keep coherency with IVV Strategy.

Drive software agile backlog accordingly; feed EPICs and User Stories based on Capabilities, and their descriptive functional chains and scenarios, appropriately cut into increment-size chunks if needed.

4.2.3.7 Contribution of activity “Perform SW engineering”

Define Agile Release content and Epics from the Functional Chains, being defined and refined in co-engineering. EPIC/User Stories used for « value » implementation scheduling. (TCE)

Maintain a vision of software architecture, including the goal at completion, and update them at each sprint.

4.2.3.8 Contribution of activity “Design PHYSICAL ARCHITECTURE”

Maintain a global "goal at completion" view of the solution, and ensure adequacy and validity of architecture at each increment. Adapt view at completion according to backlog and previous deliveries contents and results.

4.3 Minimise Development Cost

The workflows supporting ‘Minimise Development Cost’ are described below.

4.3.1 Product Line Definition

Challenge: How to build and justify a product line contents.

- Integrated approach between market/business analysis, architecture definition and variability engineering
- Can largely rely on architectural method such as Arcadia

Harmonisation of market & business analyses with architecture & design:

- ‘Operational stakeholders Need Analysis’ is the place to start collaboration
- ‘Need vs Solution’ articulation in architecture & method secures coherency

Verification process for variabilities & options:

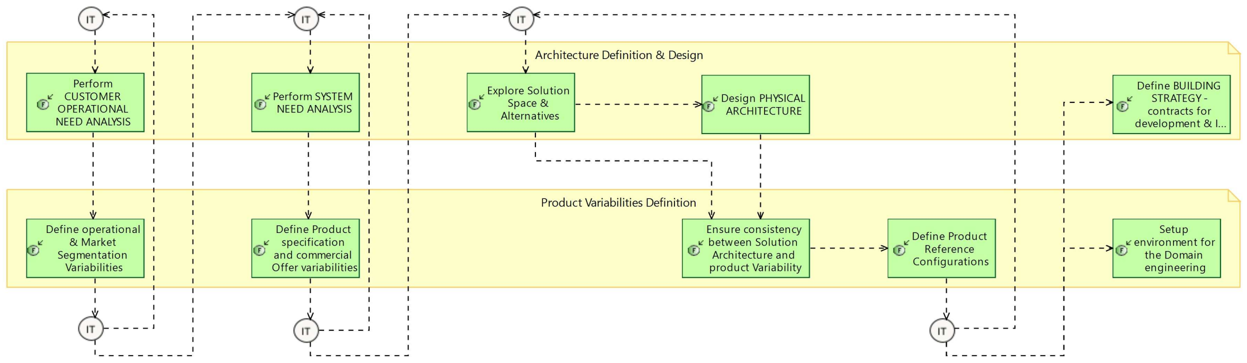
- Confront to customer need and solution architecture, value analysis…
- to manage complexity of variability (Feature model) and simplify/secure it in coherence with architecture description

Benefits of integrated approach:

- because Feature Model alone is not enough guiding to choose among options
- Architecture analysis simplifies some variability sources,
- helps in defining and reducing dependencies between variants,
- and secures adequacy between variability and architecture,
- eases and encourages using high level variability(e.g. on op. capabilities),
- and achitecture model helps and justifies option selection and combinations

How to define and check valid architecture configurations?

- By derivating an architecture model per configuration, and analysing it for consistency, completeness and adequacy to needs & constraints



Architecture Definition & Design
Perform CUSTOMER OPERATIONAL NEED ANALYSIS
Perform SYSTEM NEED ANALYSIS
Explore Solution Space & Alternatives
Design PHYSICAL ARCHITECTURE
Define BUILDING STRATEGY - contracts for development & I...
Product Variabilities Definition
Define operational & Market Segmentation Variabilities
Define Product specification and commercial Offer variabilities
Ensure consistency between Solution Architecture and product Variability
Define Product Reference Configurations
Setup environment for the Domain engineering
IT

4.3.1.1 Contribution of activity "Perform CUSTOMER OPERATIONAL NEED ANALYSIS "

During the operational need analysis activity, a market segmentation is elaborated based on classifying and characterizing the stakeholders potentially involved in system use and operation. They are captured as Operational Entities (organizations, systems, etc.) or Operational Actors (human users for example). For each market segment, the expectations of the stakeholders are expressed in terms of Operational Capabilities, Missions, Scenarios, Operational Processes and related Operational Activities.

Value characterization for each market segment applies on former elements. It captures end-user value, criticality, and conditions of combination/exclusion, including expected quality of service and non-functional aspects.

4.3.1.2 Contribution of activity "Define operational & Market Segmentation Variabilities"

In parallel with the Operational Analysis, customer profiles are established to describe each customer segment, collaboratively with marketing and sales teams, according to enterprise product policy. For each segment and each relevant stakeholder, jobs, pains and gains are described: Jobs are what customers are trying to get done in their work and in their lives; Pains are anything that annoys the customers before, during and after trying to get a job done; Gains are the outcomes and benefits customers would like.

The concepts of the Operational Analysis constitute a great support as jobs can naturally be related to Operational Activities and pains/gains can consist in characterizations of these Activities.

Operational Analysis and customer profiling favor thorough analyses and understanding of the stakeholders activities and expectations, from which user-level features and options emerge.

This emerging variability tree is captured in a high-level Feature Model that describes firstly orientations and alternatives. Variability analysis may in turn shape the content and structuring of the Operational Capabilities and Activities.

Checks for global consistency of options & alternatives and user profiles should be performed here (e.g. seeking for Activities needed to ensure a capability and missing in a feature, or scenarios in an alternative mentioning activities of another incompatible one, or operational process broken by an optional activity…)

4.3.1.3 Contribution of activity “Perform SYSTEM NEED ANALYSIS”

The System Need Analysis activity is about identifying and describing the system expected capabilities. Capabilities rigorously capture the ability of the system to render services contributing to the realization of one or several operational capabilities. These capabilities are exemplified with functional chains, scenarios and functions that not only describe how the system is expected to behave, but also help specify non-functional expectations.

Following realization links from Operational Analysis, a first orientation of the variability analysis is possible thanks to this functional analysis, taking into account the former market segmentation and users expectations.

4.3.1.4 Contribution of activity “Define Product specification and commercial Offer variabilities”

Value and variability analysis

The value analysis approach is similar to Market Analysis and Segmentation. In response to the stakeholder pains and gains, services, gain creators and pain relievers are identified. Services (typically captured as capabilities and functions) describe what the product will offer and what will help stakeholders complete their jobs or reach their objectives. Gain creators emphasize how certain product functionality will help users/customer be more efficient (in terms of time, quality, or effort for example). Pain Relievers will emphasize how certain product functionality will contribute to help users/customer address the difficulties they face.

Value analysis strengthens the system need analysis. It grounds the product definition on solid foundations and when combined to the stakeholder profiles, it justifies the creation of a new set of system-level variability features and options that are are captured in a refined version of the Feature Model.

This integrated, model-supported approach benefits and simplifies variability analysis in at least three ways:

- The alignment between system capabilities and the structure of the Feature Model brings both organizational (structuring of engineering responsibilities and activities) and technical advantages (easier consistency checking, easier impact analysis).
- The modeling effort favors the identification of commonalities, with elements of the functional analysis (capabilities, functional chains, scenarios, and functions) that are transverse to all segments and markets
- The dependencies between elements of the functional analysis influence and even simplifies features. For example, defining one single feature or option on a functional chain instead of several ones on the functions involved in the functional chain is much simpler. In addition, dependencies between features can be deduced from dependencies between elements of the functional analysis.

Commercial Portfolio Definition

Based on these functional and value analyses, the Commercial Offer Portfolio can be defined and structured (such as a car range with different equipment and finishing levels and option packs). For this purpose, standard configurations can be built from analysis and characterization of the assets above, each configuration being a selection of appropriate features options or alternatives. These standard configurations constitute the portfolio to be proposed to customers.
These standard configurations are key in order to simplify the definition of the solution for a given customer: they guide the customer choices according to market segments and towards company preferred capabilities and assets, hence maximizing reuse; they reduce the number of architectures to be evaluated and validated both at definition time and at Integration Verification and validation (IVV) time, etc.

The same kind of coherency checks between System Need Analysis and Commercial Portfolio Definition has to be performed, as it is supposed to be done at operational level. Coherency with market analysis should also be checked thanks to links with operational analysis.

4.3.1.5 Contribution of activity “Explore Solution Space & Alternatives”

The model-based design of the solution architecture follows the same patterns than the ones of Operational and System Need Analyses. Capabilities are exemplified with scenarios and functional chains that describe how the system works and not only what is expected from it. They help specify the exact contribution of each system constituent. The functional behavior of the solution must not only realize the functional analyses defined in Operational and System Analyses, it must also reflect and implement their variabilities: for example, if a functional chain of the system need analysis is associated to a variability feature, its corresponding functional chains in solution architecture must also be associated to it.

From a structural point of view, product variability may significantly constrain architecture design: for example, functions that have different variability conditions should be implemented by separate components; similarly, alternative behaviors should yet preserve similar common interfaces, and common core behavior should be implemented by dedicated core components.

New variabilities are likely to appear, based on limitations or opportunities in solution, technology, or context, leading to a solution feature model. Links between architecture and feature models are built accordingly.

4.3.1.6 Contribution of activity “Design PHYSICAL ARCHITECTURE”

Architecture Variability constraints may deeply influence architecture design (e.g. separating functions or components that have different variability conditions).

Architecture can simplify and reduce the number of alternatives and options, notably due to architectural consistency or dependencies for example :

- Because architecture constraints may lead to group different variabilities, that can be then considered as a whole
- Because some options are not independent due to architecture constraints
- Because one model element may group several options, etc.

4.3.1.7 Contribution of activity “Ensure consistency between Solution Architecture and product Variability”

Consistency checking

Here again, checking coherency and consistency between architecture, feature model and configurations contents is key, notably:

- Consistency of components breakdown with variability
- Consistency of function to component allocation with variability and options
- Identification of dependencies or incompatibilities between features due to architectural concerns
- Validity of features and configurations in preserving components dependencies, scenarios & functional chains consistency…
- Specific work on non-functional properties, quality of service, etc.

Variability Implementation and Mastering

Once architecture and feature models are consistent with each other, the features and configurations definition should be applied to most engineering assets and beyond : requirements, architecture definitions and models, simulations models and scenarios, specialties and disciplines specific assets and models, test means and enabling systems, test campaigns, test cases, and beyond product breakdown structure, development and production means, support means, tooling, etc.

4.3.1.8 Contribution of activity “Define Product Reference Configurations”

Solution Variabilities Orientation

An analysis of the solution architecture can simplify and reduce the number of variability features, notably due to architectural consistency or dependencies. For example, architecture constraints may lead to group different variabilities, that can be then considered as a whole: there is no need to treat them separetely if the same components and functions are required for all of them.

Design standard configurations are also defined to implement and refine the former market profiles and portfolio standard configurations, including solution-specific variabilities, and dealing with former architecture-originated constraints and simplification opportunities. Product standard configurations are expected to cover and fit most users needs, by appropriate contents according to users, system need and architecture. So each final, solution-level standard configuration should notably:

- satisfy a set of users belonging to a segment identified in Operational Analysis
- be compliant with commercial offer orientations as defined in portfolio and at System Need Analysis level
- be coherent with designed architecture, for feasibility and efficiency reasons.

4.3.1.9 Contribution of activity “Define BUILDING STRATEGY - contracts for development & IVVQ”

Building Strategy

Development contract (Specification) of solution components and building blocks must integrate the product variabilities and configurations policy: the fact that for the system engineering, a component should be optional, does not necessarily impact the engineering of this component; but a component may have to adapt to other components variability (e.g. dealing with optional components missing in some configurations), and it may have to deal with internal functional or non-functional variability, as requested by system engineering.

Integration, verification, validation procedures should test variability points and check major standard configurations as such.

Building Blocks Variability Management

The solution architecture might rely on certain building blocks that can have their own product variabilities and policies. Variability at both levels must be compatible and harmonized accordingly. The result of this necessarily collaborative work should be part of the development contract for each building block.

4.3.1.10 Contribution of activity "Setup environment for the Domain engineering"

Once architecture and feature models are consistent with each other, the features and configurations definition should be applied to most engineering assets and beyond : requirements, architecture definitions and models, simulations models and scenarios, specialties and disciplines specific assets and models, test means and enabling systems, test campaigns, test cases, and beyond product breakdown structure, development and production means, support means, tooling, etc.

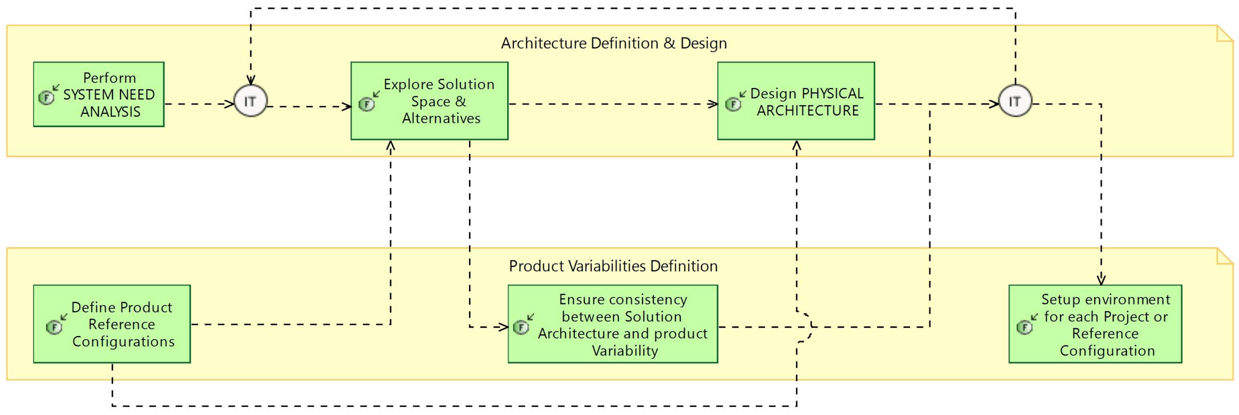
4.3.2 Project Definition in PLE

Challenge: How to initialise a project from the product line assets

Project should confront its own Need Analysis to the one driving Product Policy and PL Engineering, and orient the customer towards existing Product capabilities.

The solution should be based on tuning existing product reference configurations, and if not possible, on reusing/selecting existing product assets.

The project could contribute to enriching product assets baseline, and adapt or complement product reference configurations.


Architecture Definition & Design
Perform SYSTEM NEED ANALYSIS
IT
Explore Solution Space & Alternatives
Design PHYSICAL ARCHITECTURE
IT
Product Variabilities Definition
Define Product Reference Configurations
Ensure consistency between Solution Architecture and product Variability
Setup environment for each Project or Reference Configuration

4.3.2.1 Contribution of activity “Perform SYSTEM NEED ANALYSIS”

When building offer for customer, try to propose one of existing product reference configurations.

If not possible, try to build a configuration only based on selecting options in product variability tree.

4.3.2.2 Contribution of activity “Define Product Reference Configurations”

Reuse existing product reference configurations as is, as much as possible.

Adapt reference configurations according to new needs and segmentation possibly brought by the project.

4.3.2.3 Contribution of activity “Explore Solution Space & Alternatives”

Reuse product definition, reference configurations and reusable engineering assets to speed up alternatives exploration and orient towards existing assets.

4.3.2.4 Contribution of activity “Ensure consistency between Solution Architecture and product Variability”

verify that existing product variability fits most project and customer needs.

Check notably that reusing existing assets as is do not hinder performance and non-functional expectations on project solution.

Check coherency and completeness of architecture when merging product reusable assets and project-specific ones.

4.3.2.5 Contribution of activity “Design PHYSICAL ARCHITECTURE”

Consider promoting some project-specific assets as product reusable assets.

4.3.2.6 Contribution of activity “Setup environment for each Project or Reference Configuration”

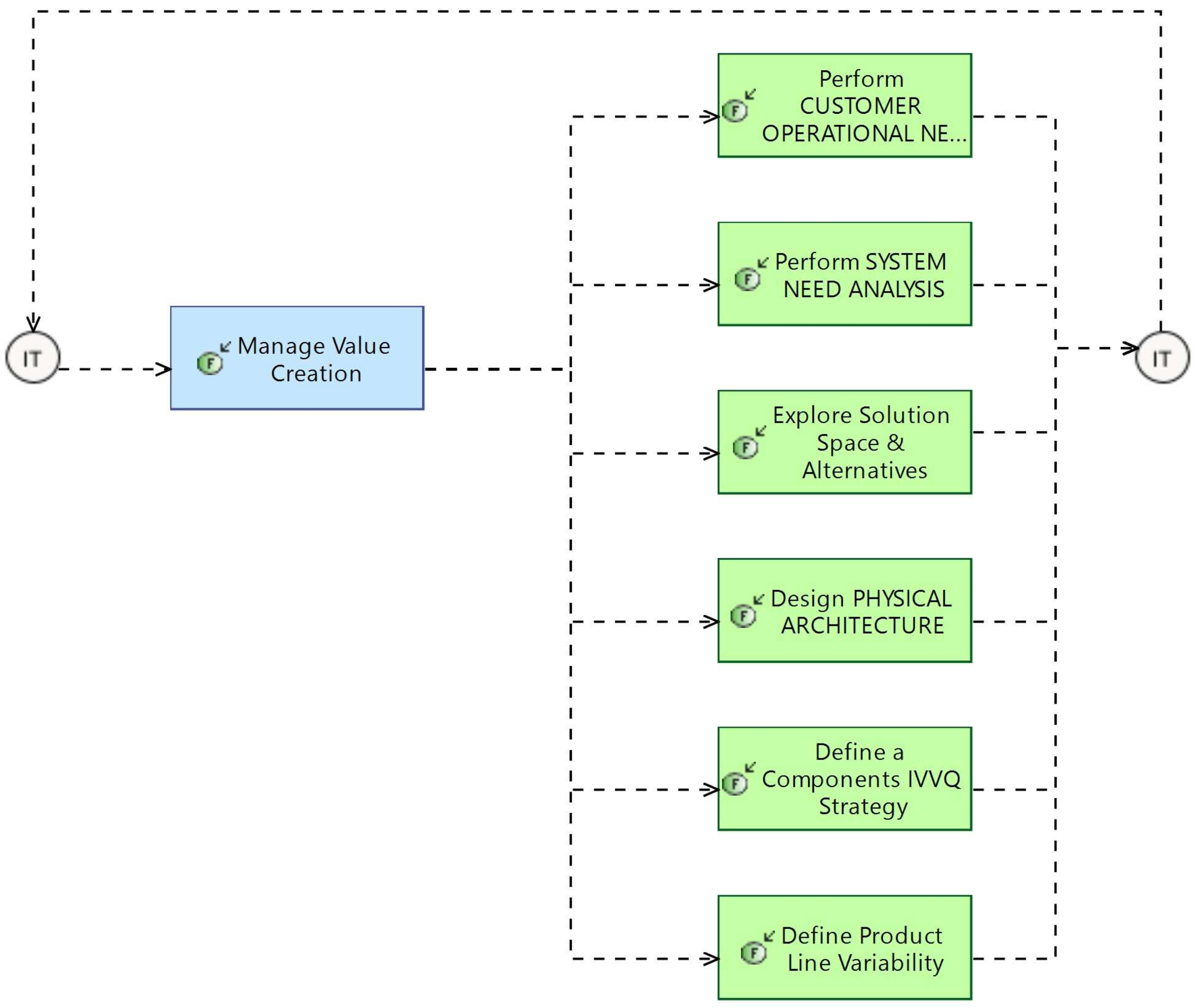
Select the product reference configuration closest to the project need, if any. Otherwise, create a project-dedicated configuration from product variability and reusable assets, complementing with project-specific assets.

Based on this selection, initialise the project engineering assets from existing ones : requirements, architecture definitions and models, simulations models and scenarios, specialities and disciplines specific assets and models, test means and enabling systems, test campaigns, test cases, and beyond product breakdown structure, development and production means, support means, tooling, etc.

4.3.3 Value-driven Engineering Course

Challenge: How to make sure to deliver the expected value all across the solution lifecycle to the customer/market and to the company.

- Evaluate value and priorities of main engineering assets for stakeholders : need, solution, schedule and deliveries, and more
- Organise engineering, development and deliveries according to most valuable and prioritary assets production


IT
Manage Value Creation
Perform CUSTOMER OPERATIONAL NE...
Perform SYSTEM NEED ANALYSIS
Explore Solution Space & Alternatives
Design PHYSICAL ARCHITECTURE
Define a Components IVVQ Strategy
Define Product Line Variability
IT

4.3.3.1 Contribution of activity “Manage Value Creation”

ObjectiveTo organise and plan engineering activities with the objective of creating value for our customers/market and the company.

- Value for customers:
  - Added value for their mission execution, at an affordable price and on time
  - Best user experience during the whole product lifecycle
  - Deliver solutions that users can trust in all circumstances
- Value for the company:

- Deliver the expected level of profitability
- Create reusable assets in consistency with product policy
- Continuously capitalize and share knowledge

To make sure the engineering activities create the expected value and take appropriate resulting actions when deviates from the plans.
To make created value visible to all stakeholders.

4.3.3.2 Contribution of activity “Perform CUSTOMER OPERATIONAL NEED ANALYSIS ”

Examples of valuable assets:

from the customer and end-users point of view :

- Users missions and required capabilities (both in terms of operational importance, of urgency)
- Operational processes and scenarios

from the supplier company point of view :

- Differentiators Vs competition: Capabilities that competitors deliver or not, new capabilities that the system could contribute to.

4.3.3.3 Contribution of activity “Perform SYSTEM NEED ANALYSIS”

Examples of valuable assets:

from the customer and end-users point of view

- Most important requirements, system expected functions and capabilities (both in terms of operational importance, of urgency)
- System use cases, functionzl chzins and scenarios

from the supplier company point of view :

- Costly or risky elements
- impact on the product policy and product line
- industrial constraints (production, sub-contracting, reuse...)

4.3.3.4 Contribution of activity “Explore Solution Space & Alternatives”

Include former value analysis as major criteria for alternatives exploration.

4.3.3.5 Contribution of activity “Design PHYSICAL ARCHITECTURE”

Check solution against value for customer and company.

Evaluate valuable assets in the solution definition and architecture, in order to optimise risk management, design & development planning, and possibly user stories and sprints in agile practices.

Note: also apply to Logical Architecture design

4.3.3.6 Contribution of activity “Define a Components IVVQ Strategy”

Order IVV steps according to customer and company value analysis.

4.3.3.7 Contribution of activity “Define Product Line Variability”

Decide of each variant according to customer and company value analysis.

Check each product reference configuration against them.

5