30th Annual INCOSE international symposium
Cape Town, South Africa
July 18 - 23, 2020

Models as enablers of agility in complex systems engineering

Juan Navas
Thales Corporate Engineering
juan.navas@thalesgroup.com

Stéphane Bonnet
Thales Avionics Technical Directorate
stephane.bonnet@thalesgroup.com

Jean-Luc Voirin
Thales Airborne Systems
Thales Technical Directorate
jean-luc.voirin@fr.thalesgroup.com

Guillaume Journaux
Thales Airborne Systems
guillaume.journaux@fr.thalesgroup.com

Abstract. Complex systems engineering programs not only deal with the inherent complexity of the systems they develop, they also face shorter time-to-market, increasing changes in environments and usages, and more sophisticated industrial schemes. The ability to adapt to new circumstances, or agility, becomes mandatory. In this paper we present how Model-Based Systems Engineering (MBSE) approaches can be enablers of the implementation of agility in complex systems engineering programs. Known to provide additional engineering rigor and quality, MBSE also brings key concepts favoring agility and co-engineering.

Introduction

Agility, defined as the ability to adapt to new circumstances, is intrinsic to systems engineering. The systems approach highlights the interactions between the system parts and between the system and the entities external to it, in order to better understand, analyze and develop solutions that satisfy the expectations and the constraints of the stakeholders. Such an approach is well suited to address the cases in which these expectations and constraints evolve in time, as the elements of the system are not considered as single entities, but as parts of a whole which environment and context of usage may evolve in time.

Nevertheless, the way systems engineering has been traditionally implemented in organizations developing complex systems, struggle to address situations in which expectations and constraints change at a very fast pace. As the pressure for developing new products and services even faster and cheaper increases, agility becomes mandatory for organizations developing such systems[1].

In more recent years, the agile approaches originated in software engineering have been extended and tailored to complex systems engineering, claiming similar outcomes on speed of development and on efficiency of the use of resources. Their applicability is not straightforward though, as software engineering, contrary to systems engineering, almost exclusively involves virtual objects that can be evolved rapidly. Also, these approaches introduce new concepts and practices which are not always consistent with well-proven systems engineering ones. Finally, complex systems engineering

[1] As an example, the number of Internet of Things (IoT) connected devices installed base worldwide will increase up to 75 to 100 billion in 2025, depending on the estimations. This is 3 to 4 times the numbers in 2019. The possibilities of offering services by orchestrating these objects are countless, and the competition to win market shares will be rude.

faces non-functional constraints, such as safety and cybersecurity, which are poorly supported by current agile practices. On the other hand, agile approaches have also highlighted some values and principles that are in the very nature of the systems approach and that had been muted by traditional systems engineering approaches[2].

Model-Based Systems Engineering (MBSE) has gain popularity in complex systems engineering in the last ten years. MBSE provides a certain level of formalism and rigor at performing systems engineering. It introduces a set of concepts, analysis perspectives and views that are understood by all engineering stakeholders, enabling reaching a common comprehension of what the system is and of the contribution of its parts to the common goals. Finally, MBSE advocates a single source of truth, a repository on which all engineering artefacts and their relations can be inspected.

This paper presents how MBSE can be an enabler for a successful implementation of agility in the context of complex systems engineering organizations. The first section provides the necessary background – a presentation of the minimum set of concepts from MBSE and Agility that are required for the following chapters. The second section describes the concepts of a methodological approach for implementing an incremental evolution of the system. The third and last section illustrates this methodology with an example of product development performed by three engineering teams and explicitly shows how models are exploited. The conclusion of this paper summarizes the contributions and provides perspectives.

Background

Model-Based Systems Engineering

Model-based systems engineering is the formalized application of modelling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases. [INCOSE 2014]. In addition to providing an increased rigor in these engineering activities, one essential objective of a model-centric approach is to provide a single source of truth that can be shared with all stakeholders.

Arcadia is a model-based method devoted to systems, software and hardware architecture engineering [VOIRIN 2017]. It describes the detailed reasoning to understand and capture the Customer need, define and share the product architecture among all engineering stakeholders, early validate its design and justify it. Arcadia can be applied to complex systems, equipment, software or hardware architecture definition, especially those dealing with strong constraints to be reconciled such as cost, performance, safety, security, reuse, consumption, weigh, etc. It is intended to be embraced by most stakeholders in system/product/software/hardware definition as their common engineering reference. It has been applied in a large variety of contexts over the last ten years.

The Arcadia method intensively relies on functional analysis, which is a very popular technique among systems engineers. Arcadia enforces an approach structured on different engineering perspectives establishing a clear separation between system context and need modeling (operational need analysis and system need analysis), also called here *need perspectives*, and solution modeling (logical and physical architectures) also called here *solution perspectives*, in accordance with the [IEEE 1220] standard and covering parts of [ISO/IEC/IEEE 15288].

As the lack of properly tailored tools has proven to be a major obstacle to the implementation of MBSE in industrial organizations [Bonnet 2015], Arcadia is recommended to be implemented using

[2] One example: the agile value “Customer collaboration over contract negotiation”, when “Customer” is replaced by “stakeholders”

the open-source modelling workbench Capella, whose diagrams are inspired from SysML and that has proven suitable for systems engineers with diverse backgrounds and skills [Capella 2019].

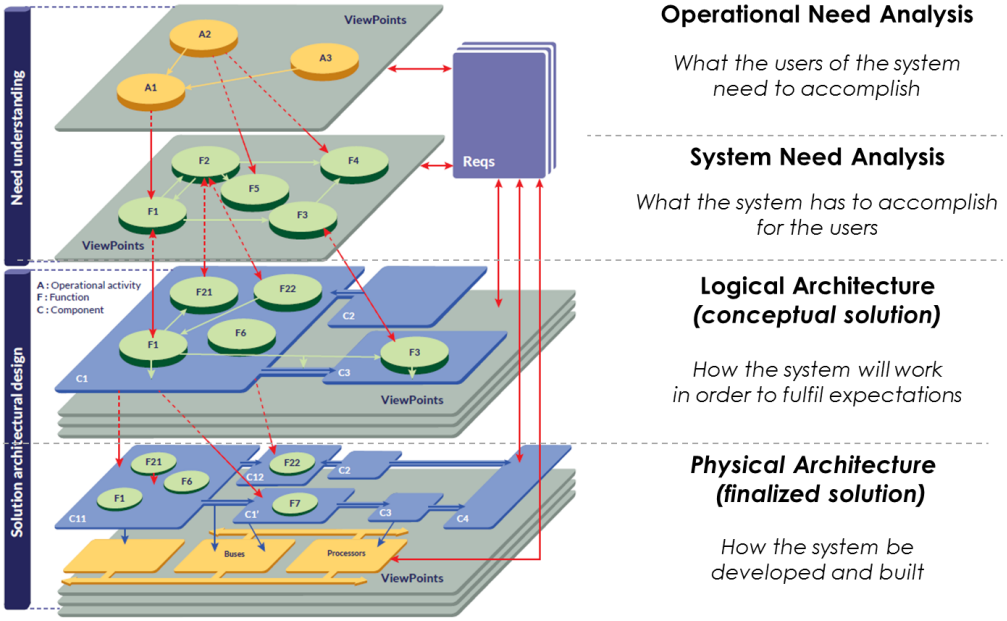

Figure 1: Arcadia engineering perspectives

In this paper, we only exploit the subset of the Arcadia concepts that are key for implementing agility. For the sake of clarity, they are defined here. An *Actor* is an external entity (user, operator or other system) interacting directly with the System under study. A *Capability* is the ability of the System to provide a service that supports the achievement of high-level operational goals; a Capability is described by *Scenarios* and *Functional Chains*, all illustrating possible usage contexts of the Capability. Both Functional Chains and Scenarios involve *Functions*, which are actions, services or operations fulfilled by an Actor or by the System, or by one of the Components of the System. Scenarios are specific in which they can describe dynamic, time-related interactions, whereas Functional Chains don't.

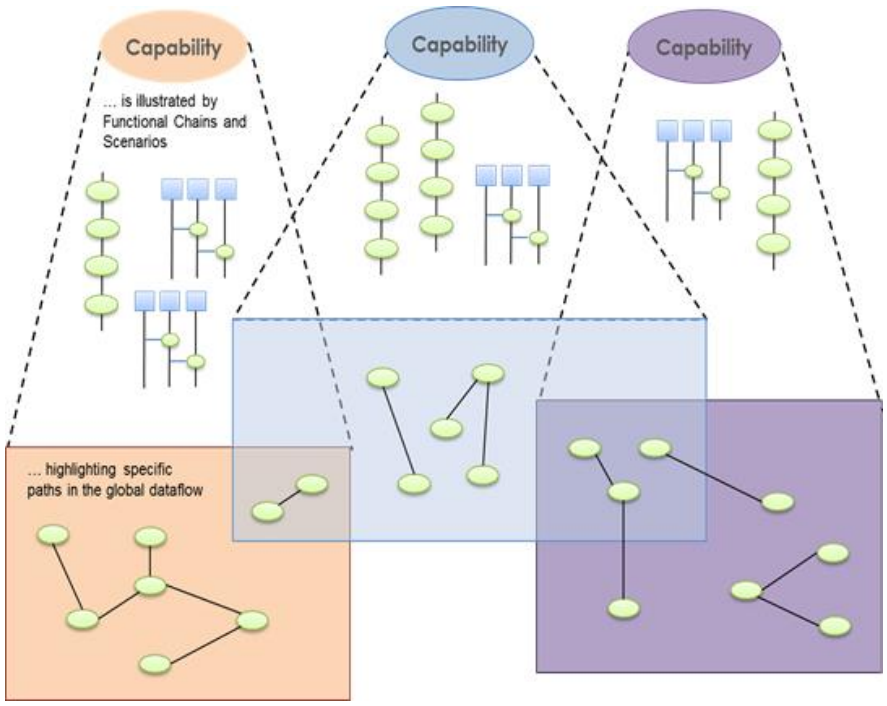

Figure 2 - Capabilities, Functional Chains and Scenarios, and Functions

Figure 2 summarizes the relationship between Capabilities, Functional Chains and Scenarios, and Functions. The level of abstraction of these concepts decreases from top to bottom. Note that a given Function can contribute to several Capabilities, as it can be involved in several Functional Chains and Scenarios that describe different Capabilities.

Agility in Systems Engineering

In this paper, agility in systems engineering refers to an engineering effort in which teams are able to adapt to new circumstances (e.g. changes on or new stakeholders needs, technological innovations, findings in the development process ...) while meeting the Customer expectations in terms of schedule, quality and cost.

As shown in [DOVE, 2018], the ability to achieve this agility shall consider not only the systems engineering process that is put in place (which includes the activities, practices, ceremonies, ... and the tools being used), but also the characteristics of the system under development, and the properties of the enterprise(s) that carry on the development effort. Indeed, flexible system architectures greatly contribute to the ability to make evolve this architecture in a faster way in order to satisfy new needs, and an efficient research & development service can support the engineering processes so to integrate new technologies to their products and go faster to the market.

Regarding the engineering process that is put in place when implementing an agile approach, a number of frameworks have emerged in past years. They are devoted to the development of different kinds of systems (software-only, software intensive systems ...), scaled to different number and sizes of engineering teams. They define agile principles to be followed, concepts, planning patterns, activities, tasks, ceremonies, work products, team roles, and interactions between these roles, among others. A non-exhaustive list includes Scrum [Scrum, 2019], SAFe [SAFe, 2019], LeSS [LeSS, 2019]. This paper does not directly refer to these frameworks but to a minimal set of concepts that can be tailored to implement them.

Agility can be achieved through increments and iterations. An *increment* is a subset of the system (or of the systems engineering artefacts) that is delivered at the end of a time box. An increment is the addition of value to the system stakeholders built on top of an existing baseline. This value can be knowledge, risk reduction, new features, enhanced performance, etc. Increments may be divided into finer increments. *Iterations* are usually fixed-length time boxes in which engineering teams create value for the stakeholders by producing an increment. Iterations can also be divided into shorter ones. A *backlog* is an area where are hold the upcoming needs to be fulfilled by the system so that it provides value to the stakeholders.

Methodological approach

Incremental evolution of the system

A system life cycle is the series of stages through which the system passes. [ISO 15288] states that a system progresses through its life cycle as the result of actions, performed and managed by people in organizations, using processes for execution of these actions. [ISO 24748] defines six generic life-cycle stages with their purpose: Concept - define the problem space, characterize the solution space, identify and refine stakeholders needs, explore ideas and technologies, explore concepts and propose viable solutions; Development - define and refine system requirements, create solution description, implement, integrate, verify and validate the system; Production - produce systems, inspect and verify; Utilization - operate system; Support - provide sustained system Capability; Retirement - store, archive or dispose the system.[ISO 15288] does not define a unique life cycle approach, i.e. a way the system shall evolve through the life-cycle stages. Instead, it presents several sequential, and incremental and iterative development methods that can be used to plan the systems engineering processes. The tailoring of these life-cycle stages is out of the scope of this paper.

What [ISO 15288] does define is the existence of *gates*, which may consist of decisions, control reviews, milestones or system releases. These gates are key events that represent an expected progress in the systems engineering effort. Gates are hence used to assess the quality, costs and delays of the systems engineering effort, to synchronize teams work, and to feed the evaluation of the risk (if any) and opportunities of pursuing next systems engineering activities. A gate represents a point on time on which value is provided to the stakeholders; this value can take different forms: a consolidated conceptual design, a set of studies required for certification purposes, a simulation of the system, and the deployment of a subset of system Capabilities, among others.

Between two gates, the system or parts of it may be in different life cycle stages. The Figure 3 below illustrates a systems engineering effort driving the system through different gates and through different life cycle stages. At the beginning of the engineering effort, the system is at a conceptual stage; one of the gates marks the first release of the system for operation, this release will provide some Capabilities, where others remain under development stage and are released at the following gates.

A life-cycle stage may require one or several systems engineering processes to be performed in order to be able to pass to another stage. This is illustrated by the size of the circles representing the execution of systems engineering processes (only some of the involved technical processes are shown here). A typical case is the development stage, where a set of Capabilities of the system may be already verified and validated, while others are being defined, designed and may imply changes in the architecture.

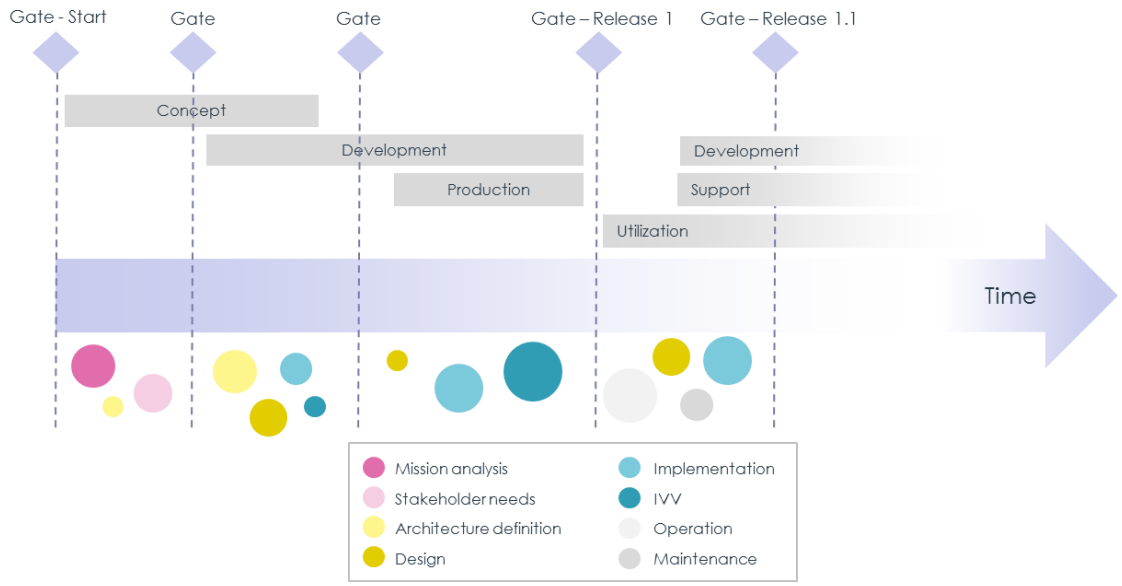

Figure 3 - Life-cycle stages and systems engineering technical processes

In this paper we will assume that a Master Schedule exists and that it provides a preliminary definition of the gates and of the foreseen evolution of the system life cycle stages that will lead to the satisfaction of the stakeholders' expectations. This assumption allows us to consider that the systems engineering effort performed between two gates leads to a *macro increment* of the system, and that these macro-level increments are possibly themselves divided into finer increments to address the required agility and reactiveness to changes. This paper is focuses on what happens during these finer increments.

Incremental and iterative execution of systems engineering activities

Three phases are executed to increment the system between two gates:

- Warm-up - collaborative definition of the detailed scope, goals an schedule of the increment and of the necessary resources, ensuring that the conditions required to perform Run iterations

without danger are met (e.g. are the expectations of the iteration well defined, are the system Capabilities sufficiently well-defined, are all enabling systems available)

- Run - iterative effort punctuated by iteration reviews
- Evaluation - evaluating how the engineering was performed, assessing that the expected outcomes are achieved and that conditions for entering and existing the next gate are met

As an illustration, the Figure 4 below refers to the production of a macro increment between two gates, in which certain system Capabilities are developed by performing systems engineering activities. The diameter of the circles representing the activities indicates the effort devoted to each activity. During the *Warm-up* mostly architecture and design activities are performed in order to define the scope of the subsequent iterations, define their number and frequency, and secure their execution. *Run* phase complete the architecture and design work and progressively implements and integrate, verify and validate the Capabilities (and deploys the updates of those already operational). The *Evaluation* phase assesses the validated scope and updates the architecture and design definitions.

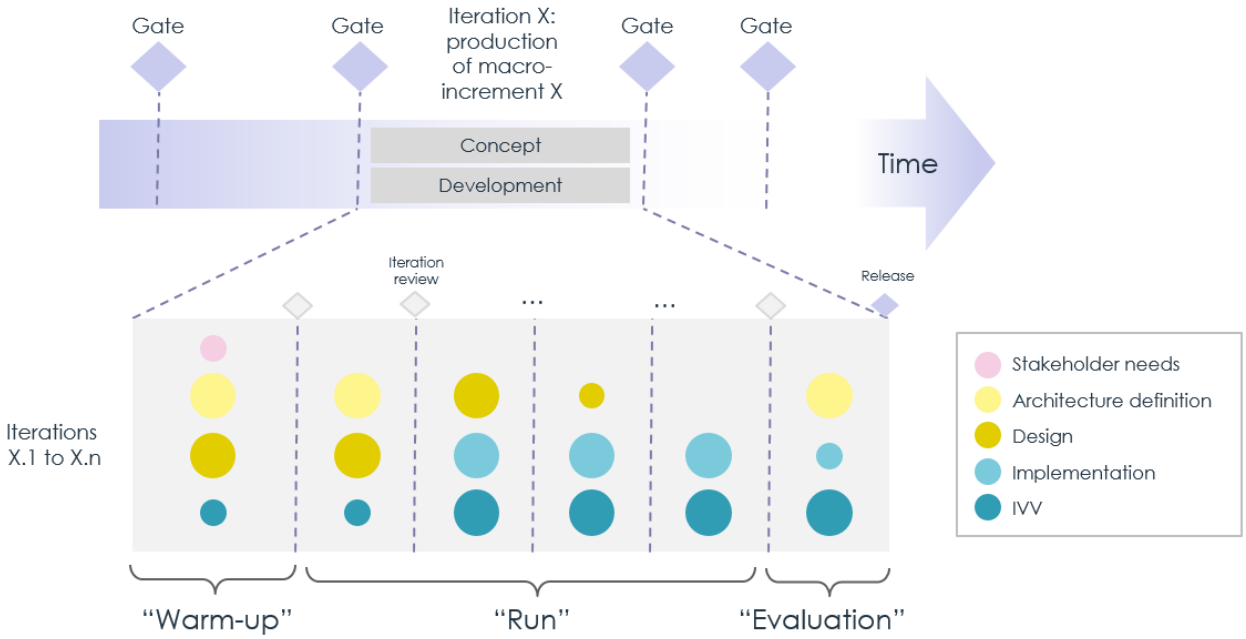

Figure 4 - Phases of the production of a macro increment

Note that the length of the iterations is not necessarily fixed. In particular, the systems engineering effort of *warm-up* and *evaluation* tends to evolve according to the evolution of the system life cycle. When the system is in a Concept stage, the effort required to achieve a well-defined definition of the system Capabilities may consume most if not all of the resources available for the increment. Similarly, when the system is in a Development stage, close to Utilization, most of the activities will be related to the validation of the system.

Also note that this pattern can be applied at different engineering levels. At system-level, iterations will address the engineering tasks aiming at developing the system; at the subsystems-level they will address the engineering tasks of the subsystems; and at the software-level they will address software development following well-known agile approaches. What will differ is the kind of engineering activities that are performed, the effort that is devoted to each of them, and the length of the iteration: indeed, whereas at system and subsystem level iterations may last around 3 months, agile software development iterations may last around 3 weeks.

Iterative and incremental evolution of engineering artefacts

The engineering activities performed during these iterations lead to the emergence and evolution of a high number of engineering artefacts. In this paper we focus on a subset of the artefacts that are particularly useful in organizing the engineering effort and that can be considered as references for all engineering teams - systems, software, hardware, IVV... We focus here on Capabilities, Functional Chains and scenarios. Other kinds of artefacts, such as interfaces, are out of the scope of this paper. The following chapters will describe how these artefacts evolve through the engineering increments.

Warm-up. The main objective here is to identify and select the scope of work of the increment and of each one of the subsequent iterations, and to define the schedule for performing this work. This scope of work is defined so that it meets the objectives of the next gate, providing the expected value to the stakeholders. The backlog strongly evolves during this iteration.

At most life cycle stages, this can be done through by selecting the Capabilities that will be developed, validated, maintained or retired, along with their related Functional Chains and Scenarios. This selection takes into account the system concerns (e.g. the expectations of stakeholders, the value the Capabilities will provide, the previously engineered Capabilities), but also the concerns related to the organization performing the systems engineering effort (e.g. the strategy of the organization, the available resources, the distribution of engineering skills).

Selected Capabilities can also be partially included in the scope of work of the increment. In this case, what is included is a subset of the Functional Chains and Scenarios that describe what are the usages contexts of the Capability that will be considered. These Functional Chains and Scenarios are described by the Functions (System but also Actors functions) and the Functional Exchanges that are involved in them, by the chronology of the exchanges and by the pre and post conditions to be satisfied.

Note that at very early stages, when system Capabilities have not yet been elucidated and there is no architectural view of the system, this iteration includes engineering tasks driving to their definition. This may include performing an Operational Analysis in Arcadia to assess the expectations of the stakeholders; defining the major Capabilities of the system; defining an intentional architecture of the system, i.e. the architectural principles in which further architectural definition work will be based. It also includes all other tasks aiming at defining the way engineering teams will work together. In further Run and Evaluation phases, updates or complements of these assets (operational analysis, capabilities, architecture, etc.) should be considered as well.

Run. This phase is made of a set of iterations aiming at implementing the Capabilities mentioned before, which includes activities such as (non-exhaustive list) the detailed definition of the functions and exchanges involved in the Capabilities; the development of the system and subsystems' architecture; the development of the software and hardware implementing expected behavior; and the verification and validation of what will be delivered at the end of the increment.

The *run* iterations tend to have the same length, but their number may vary according to many factors, both system-related (e.g. life-cycle phase, complexity of the Capabilities included in the scope) and organization-related (e.g. system or software engineering levels, available resources, human resources policies).

At the systems level, the Capabilities, Functional Chains and Scenarios in the scope of work are refined and textual requirements are associated to them. During this process, it may be useful to refine and/or partition these Functional Chains and Scenarios in smaller chunks, in order to incrementally transfer the specifications to subsystems, software and hardware engineering teams, for further design, implementation and verification. The chunks are defined to provide value to either the actors of

the system, the other external stakeholders of the system, or the internal stakeholders in the organization. They can also be defined following the organization of engineering teams, their expertise or the technologies they use.

During the run iterations, verification activities are performed to ensure the proper quality of the outcomes of the iteration. Then, at the end of each system-level iteration, a set of model elements and other engineering artefacts associated to them can be transferred to either lower engineering levels teams (subsystems, software, hardware) or to other engineering teams (e.g. Verification & Validation, specialty engineering), becoming inputs for subsequent iterations. This is illustrated in Figure 5. The increment data packages contain both needs and solution perspectives model elements, and in particular need perspectives Capabilities. This not only facilitates subsequent integration tasks, but also contributes to reach a shared vision of the objectives that the whole engineering effort is intended for.

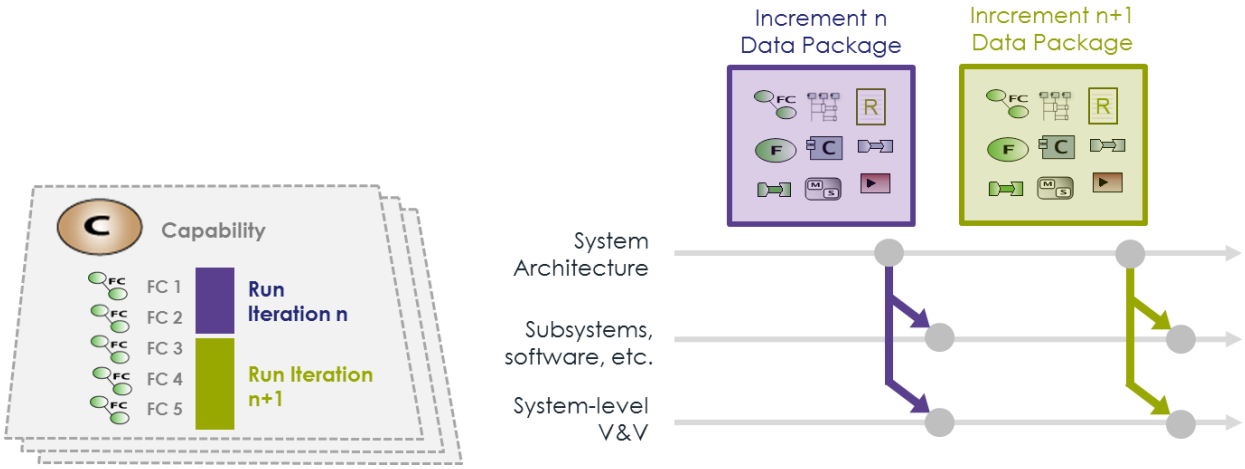

Figure 5 - Increment packages dispatched to other agile teams at the end of iterations

Evaluation. This phase is a single iteration aiming at both evaluating the system increment and the engineering effort that has been performed in the previous iterations.

Regarding the first objective, note that the major part of the integration, verification and validation effort is performed incrementally during the *run* iterations, and the *evaluation* focuses on ensuring that the whole can be released. The nature of the tasks performed here strongly varies following the life cycle of the system. During Concept stage, it may include multi-viewpoints analysis (safety, security, performance, reliability, testability etc.), the preparation of a review of experts or the execution of simulations, whereas during Development it may include the approval by the Customer or the packaging of software and hardware releases.

Regarding the second objective, the engineering teams members review the engineering practices, identify what went good and wrong, elucidate ways to improve the way they perform their engineering effort, including the dependencies with regards to external and internal stakeholders in their organization.

Immersion in the drone-based product development program

The purpose of this chapter is to illustrate the concepts presented so far with a virtual example inspired from actual Thales engineering practices, involving simple product development and three synchronized engineering teams.

The product. The “Pythagoras” organization is developing and selling lightweight drone-based products for different markets: agriculture, aircraft exterior inspection, and public security enforcement. In addition to the drones themselves, these products embed mission control and data analysis

software. These drone-based products feature manual and automated piloting, data acquisition using a wide range of technologies, live data processing, data recording, live and post mission data analysis. Figure 6 is a simplified view of the drone-based product context. In light blue, the actors around the system of interest. In green, the high-level functions allocated to the system

The engineering teams. The development of the drone-based product mobilizes a significant amount of Pythagoras resources. In this case study, we put the spotlight on three of them:

- The systems team is strongly involved in the needs capture. Architects are in charge of formalizing the needs and designing the overall solution. There are accountable for the proper integration of system constituents.
- The Integration Verification and Validation (IVV) team at system level is in charge of producing the test procedures, organizing the validation campaigns, specifying the test means, and running the actual tests.
- The "control" software team is made of embedded software engineers, specialized in motion control laws.

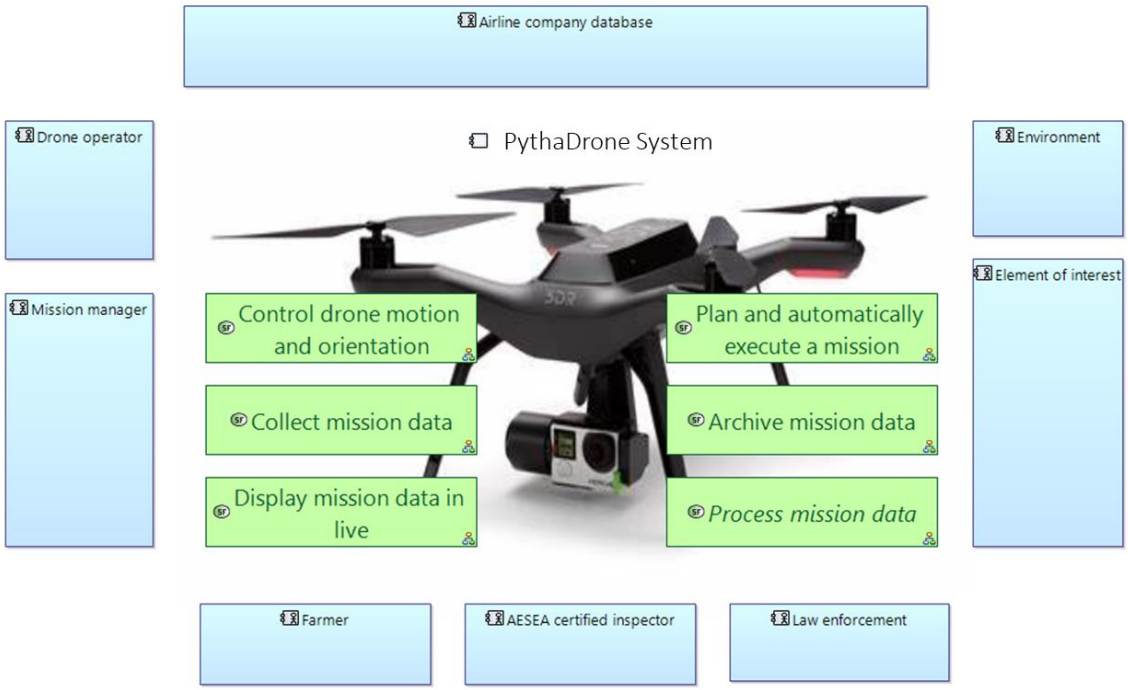

Figure 6 – The drone-product context and high-level functions

The phase in the global engineering workflow. The illustration below takes place in the context of the production of one given macro increment of the system between two Gates.. This development involves several iterations producing finer increments.

Iteration 0 - Warm-up

The objectives of the warm-up iteration are multiple: set up of the engineering environment, definition of engineering practices, provision of a first vision of the system to be developed.

Engineering environment and practices. In particular, when the product/system is in the Concept stage or when the Development is starting, it is critical to work on the engineering practices that will be implemented. Examples of questions to cover include:

- Engineering data model: what are the relationships between textual requirements and model elements, how are contractual documents produced, how are the V&V artefacts related to textual requirements and models, etc.
- Engineering tooled environment: which tools are used, how they are configured, etc.
- Articulation between engineering teams: what is a contract made of, what are the outputs/inputs of each team, what is the development pace (length of iterations at each engineering level).
- Organization and exploitation of models: What is the purpose of each model view? How will they be structured? Are there existing building blocks to assemble?

Orientation workshops can be organized to collectively provide answers to all these questions. This set-up stage is critical for the success of a project, and the help of coaches (Agile, MBSE, etc.) is highly recommended.

Figure 7 shows one possible organization. Each Capability of the system (cf. below) is assigned a Capability leader, who is accountable for the associated Functional Chains. The Capability leader coordinates the co-engineering with IVV practitioners and software teams on their Capability iterations after iterations. The results of these workshops as well as the model-based engineering strategy are intended to be formalized in the Systems Engineering Management Plan. In the context of the case study, the length of the iterations is three months.

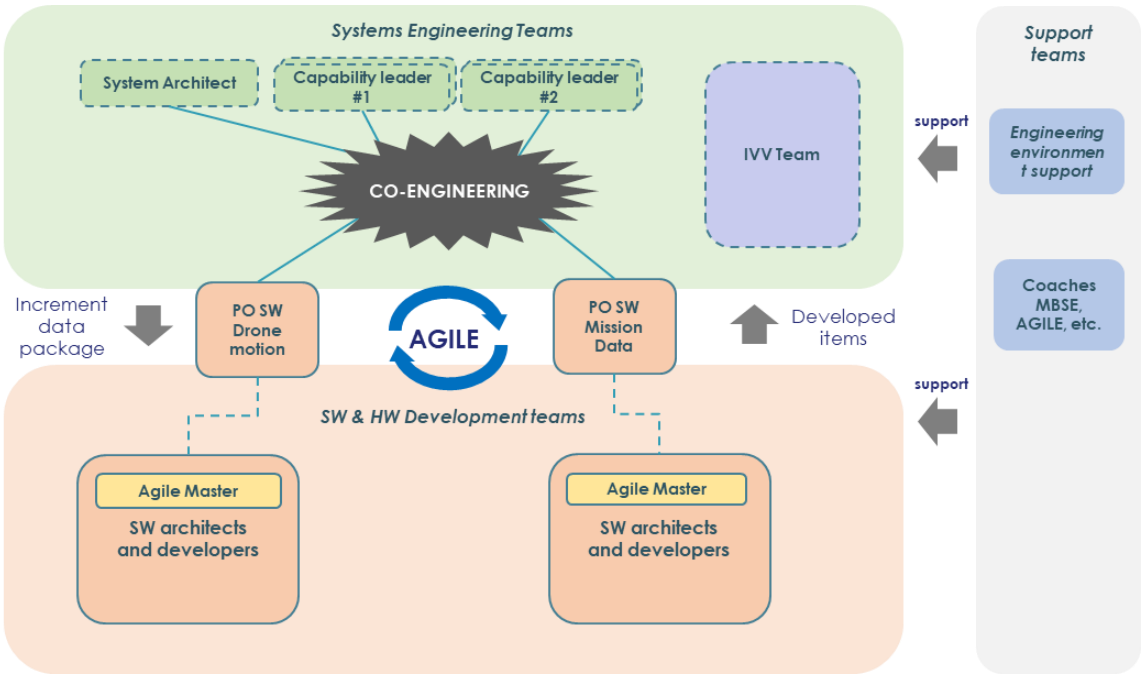

Figure 7 – OBS, Organizational Breakdown Structure

Vision of the product/system to develop. Figure 8 illustrates the main Capabilities of the product, covering all phases of a mission, from preparation to post mission data analysis. Note that a certain level of variability can be captured by these Capabilities, typically to support product line engineering [XXXXX 2019].

Each Capability is described by a certain amount of usage contexts, captured in Functional Chains and/or Scenarios. The Capabilities and Functional Chains are dispatched in different increments that will be further specified and developed in the corresponding iterations. Value analysis drives the definition of the increments. Figure 9 is an extract of the increment definition showing, for example, a Capability spread across two increments.

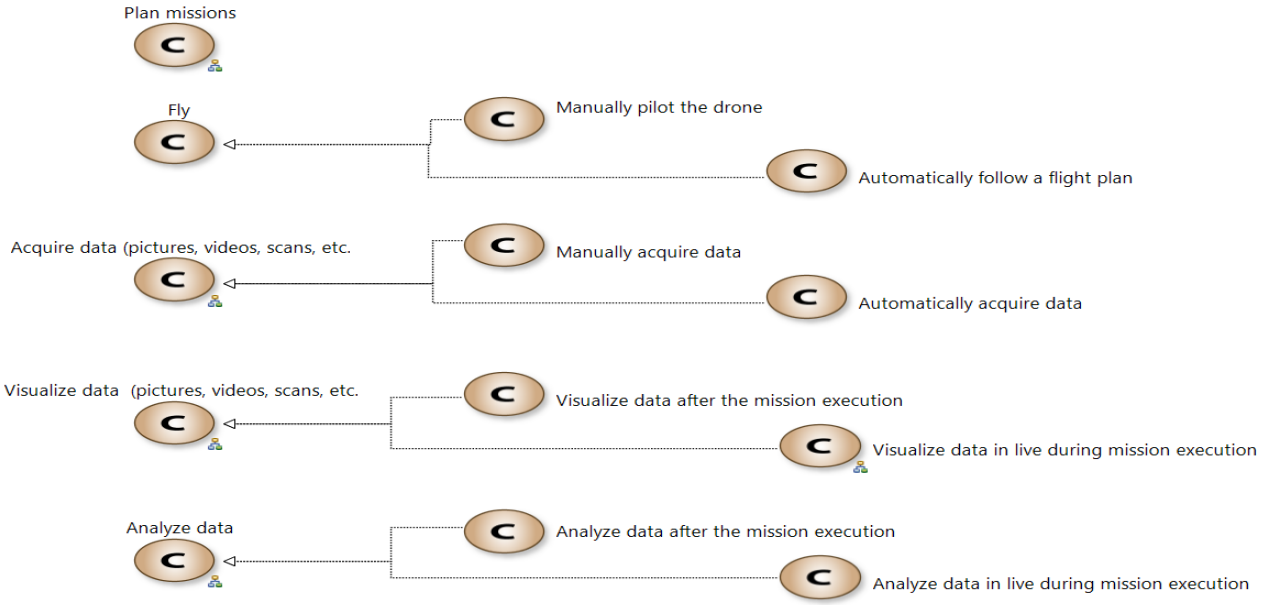

Figure 8: Extract of the Capability tree

	ITERATION 1
	Acquire data (pictures, videos, scan...
	Acquire multi-spectral image
	Acquire thermal image
	Acquire 3D image
	Manually acquire data
	Manually trigger thermal image acquisition
	Manually trigger multi-spectral image
	Manually trigger 3D image acquisition
	ITERATION 2
	Acquire data (pictures, videos, scan...
	Acquire HD video of moving element
	Acquire HD video
	Acquire HD image
	Automatically follow a flight plan
	Automatically follow a moving target
	Visualize mission progress status

Figure 9: Extract of the repartition of Functional Chains in different iterations

Run iteration n

Focus on systems team. Several Functional Chains are part of the iteration n: Manually pilot the drone, Manually trigger and record HD image, Manually trigger and record multi-spectral image, Measure and display mission metrics in live, etc.

The following paragraphs describe the specification and design efforts for the Functional Chain “Manually control drone motion and orientation”. The dataflow diagram of Figure 10 illustrates the manual control of the drone. Non-highlighted functions and functional exchanges exist from previous iterations. For example, the functions related to the automated piloting of the drone with a predefined flight plan are already available, as the provision of flight data for display on the control device. The novelty of this Functional Chain is to enable manual piloting. Using a command device, the operator will send motion orders that will ultimately dictate the drone propulsion and orientation.

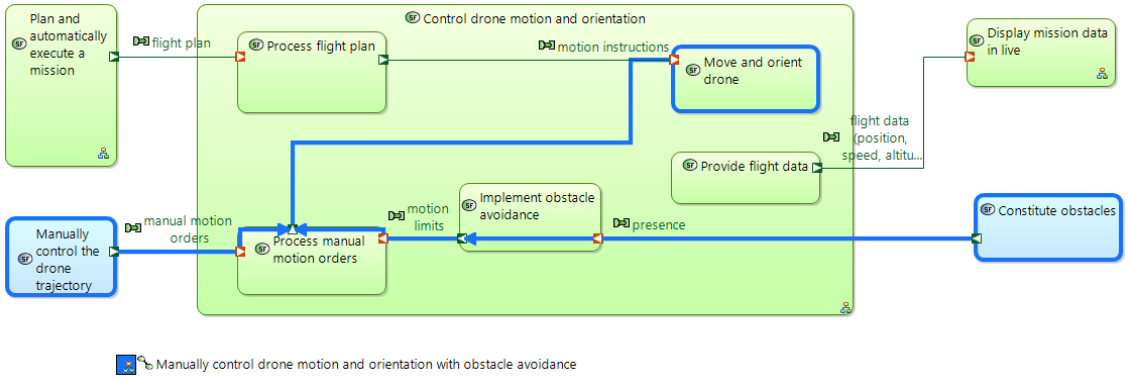

Figure 10: Functional Chain example (need perspective)

Accompanying the creation of Functions and Functional Exchanges that describe the Functional Chains, systems engineers will also create textual, need-perspective requirements imposing constraints on Functions. Textual requirements and model requirements complete each other, as presented in [BONNET, 2019].

The need-perspective Functional Chain is analyzed and taken into account in the solution model. Figure 11 shows that three smaller solution-perspective Functional Chains actually contribute to the realization of the need-perspective one. As the automated piloting of the drone was already addressed in previous iterations, it is assumed that the Functional Chain “Control drone motion and orientation” has already been developed and tested.

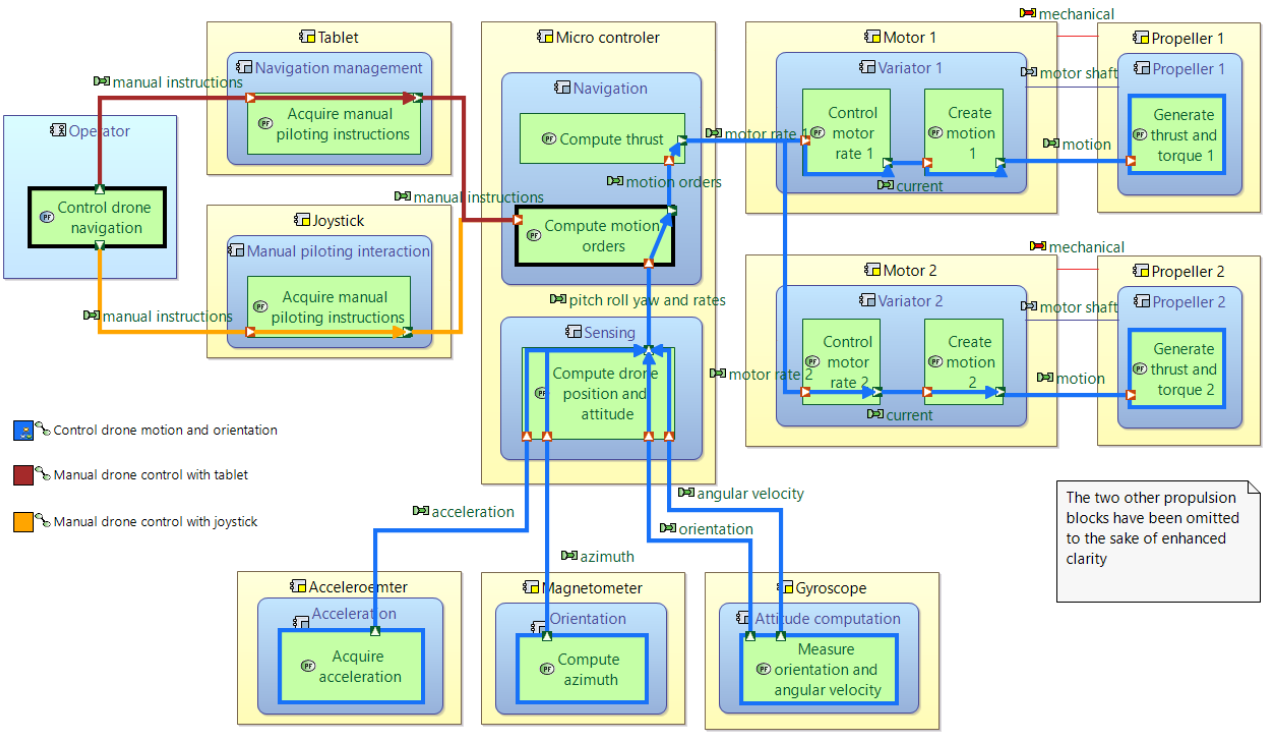

Figure 11 – Example of Functional Chain (solution-perspective)

The novelty of this iteration is to provide two ways of manually transmitting motion instructions, and to introduce alternative operating modes for the drone system (automated or manual control). Figure 12 illustrates the system mode machine. Note that here, there is no possibility to switch from automated to manual. Note here that systems engineers are likely to create new textual, solution-perspective requirements allocated to the Functions they create. For example, textual requirements can typically specify how the inputs and outputs of the Functions relate to each other, specify expectations on user interfaces, specify applicable regulations, or specify non-functional constraints.

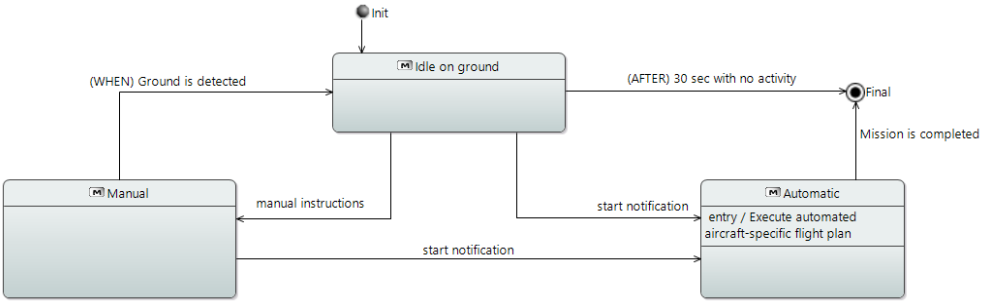

Figure 12 – System mode machine (solution-perspective)

The final increment data package is a set of consistent artefacts including models and textual requirements. Queries and automated validation allow the checking of the quality of the produced engineering artefacts. This consistency and completeness enforcement step is crucial to ensure a proper integration later on.

Focus on integration, verification and validation team. In this iteration, IVV practitioners work closely with the Capability leaders in order to translate each need-perspective Functional Chain in a corresponding system-level test procedure. Each test procedure "tells the story" of its corresponding Functional Chain, the test steps roughly matching the steps of the Functional Chain.

Figure 13 describes the traceability scheme between textual requirements, model requirements, and test procedures. As textual requirements are typically linked to Functions and Functional Exchanges, the traceability between test procedures and textual requirements can be computed [BONNET, 2019].

In addition to writing the test procedures, the IVV team also performs other activities such as preparing the test data, specifying or developing the test means, in order to be able to run these validation campaigns in the following iteration.

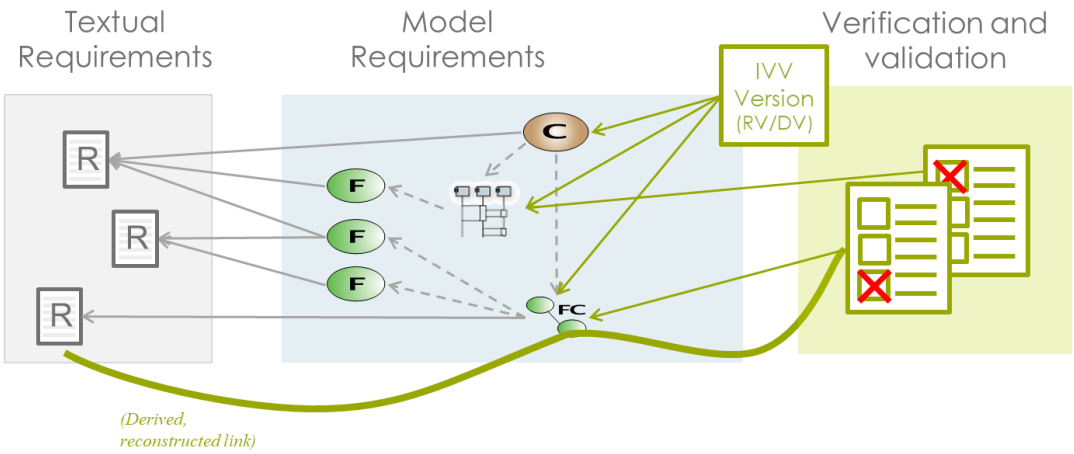

Figure 13 – Relationship between textual requirements, model requirements and test procedures

Focus on the control software team. Representative members of the software team participate to regular reviews of the increment data package currently build by the systems team. Their role is to anticipate the feasibility and to make sure the system-level vision of the solution is compatible with the current software architecture. Participation of software architects in the co-engineering effort at system level is key for the developers to "accept" the models they will receive from the systems engineers.

Run iteration n+1

Focus on the control software team. The development team receives the increment data package, and creates one chunk of work – EPIC for each Functional Chain. During this iteration, the team applies the Agile precepts, refining the received EPICs in user stories that are developed in successive sprints. For example, as iterations here are 3-month long, the software team chooses to have four sprints of three weeks. The content of the user stories is defined so that value (working software) is delivered after each software iteration – sprint.

In the current example, the following could happen:

- Sprint 1 is mostly about initializing the graphical user interface and giving the possibility to switch between manual and automated modes. The development of the mode supervision function is done in parallel (see Figure 12).
- Sprint 2 is focused on translating the instructions provided by the joystick firmware in motion instructions compatible with the solution in place (previously developed for automated piloting). Graphical user interface for manual piloting from the tablet is developed too, even though it is not plugged yet. At the end of this sprint, the drone can be piloted with the joystick.
- Sprint 3 is about plugging the actual drone motion to the piloting graphical interface on the tablet.
- Sprint 4 focuses on switchover corner cases and graphical user interface finalization.

Experience shows that software teams appreciate receiving inputs that are more rigorous. In particular, sharing Functional Chains improves the synchronization and consistency between system design, software development and IVV teams. From the point of view of the software team, Functional Chains both help better understand how the implementation of a small piece of interface or of a small feature fits in the system-wide picture and help split EPICs in more meaningful user stories.

Focus on integration, verification and validation team. The pace of the IVV team is aligned with the one of the software development team. In the current example, the IVV team integrates the components delivered by the software team every third week and runs the test procedures written during the previous iteration.

The added-value of the model-based approach becomes even greater here. When a problem is encountered on a test step, finding the corresponding Function or Functional Exchange in the model is straightforward. Using automated impact analysis, the investigation on the related data is also immediate. It is straightforward to locate the possible cause of a problem. This analysis of the model can have different outputs. If the model (need and corresponding solution) is correct, a defect is created on the faulty component. If the model is actually faulty, then a defect is created on the model itself, and an evolution request is created for the involved component (Figure 14).

Run iteration n+y

Some of the system increments might be aligned with gates related to Customer visibility milestones. Here, we consider that the result of the iteration n+1 was released to the Customer. While the Customer validates the manual piloting both from the joystick and from the tablet, he requests two evolutions: obstacle avoidance should be added, and a simple move of the joystick should trigger a mode change, interrupting the automated mode to enable the manual mode.

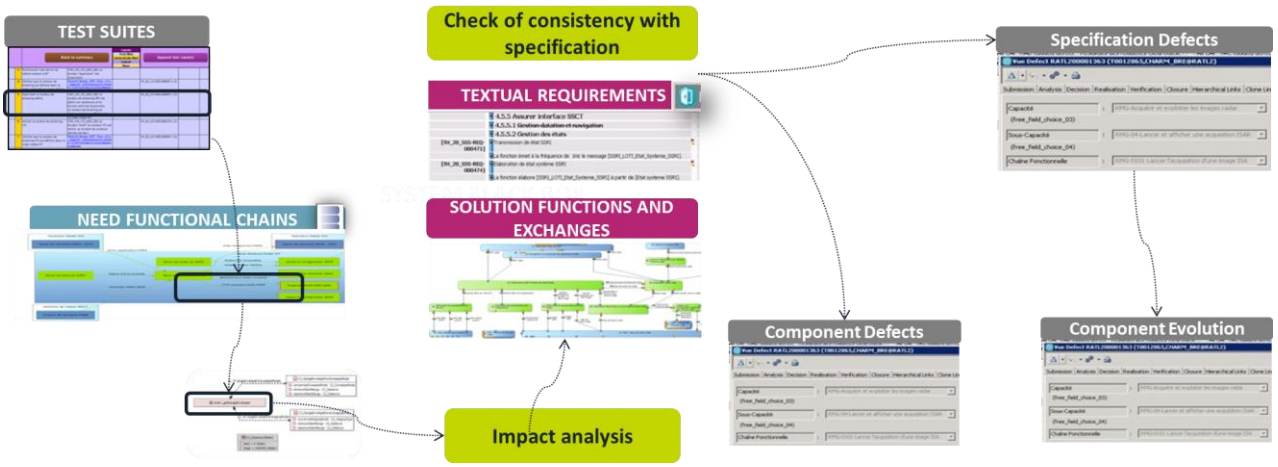

Figure 14 – Investigation of failed tests

The two evolution requests are easily captured in the need-perspective model, as shown in Figure 15. Co-engineering between the systems and the software teams will help build the corresponding solution model. Models are precious here, as they help perform more accurate impact analysis and better understand all the ramifications of a change request. Queries and any other form of data extraction from the model are means to precisely compute the consequences of any change or new need.

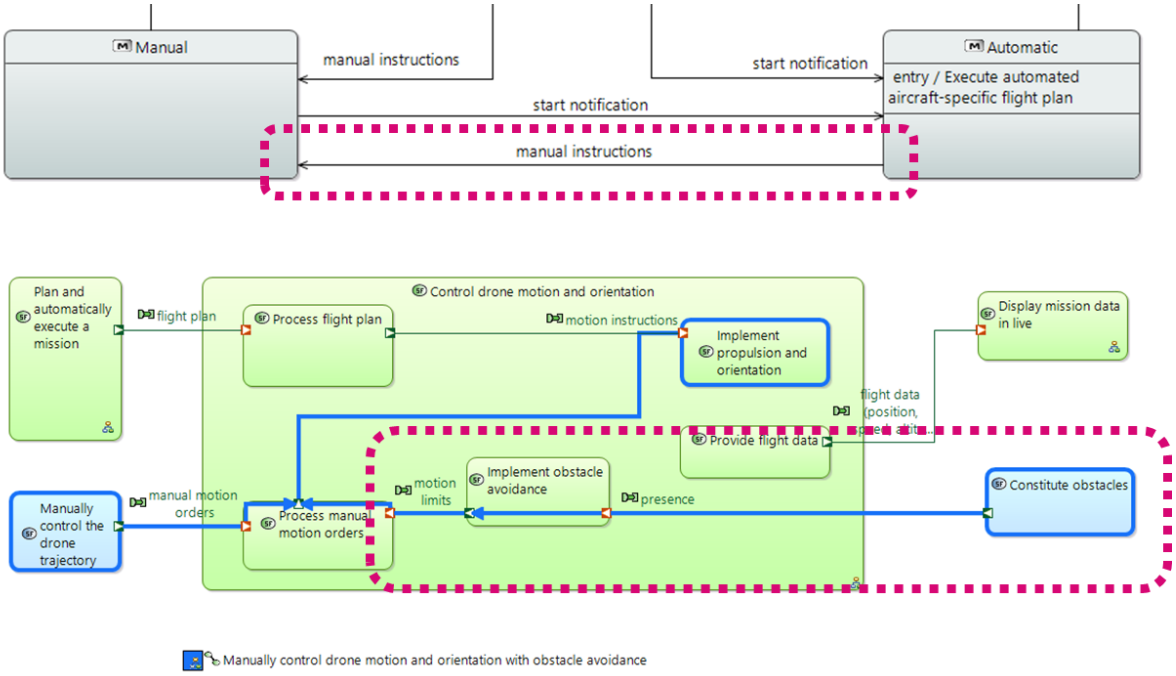

Figure 15: Two evolution requests captured in the need model

Final evaluation iteration and beyond

The process ends when the Capabilities in Figure 8 are all released and accepted by the Customer. The life-cycle of the system transitions from Development to full Operation by the Customer. Pythagoras organization may also be an active actor of the Evolutionary Maintenance life-cycle stage of the system: in this case, a “warm-up” iteration may be performed to prepare the engineering teams for this new phase.

Conclusion and Perspectives

In this paper we presented how the use of models is an enabler for a successful implementation of agility in the context of complex systems engineering. The methodological approach was illustrated with a case study inspired by current engineering practices in our organization. Agility with user stories for example and MBSE with Functional Chains share the common objective of making operational need the primary citizen of engineering practices.

The benefits of models at deploying agility are manifold, in this paper we focused on some of them. First, the concepts used in MBSE and especially those with a sufficiently high level of abstraction such as Capability and Functional Chain, are useful at *organizing and synchronizing* the development and validation effort of multiple engineering teams. Second, these concepts and the use of models as a single source of truth are also useful for *improving communication* between these engineering teams and *giving a meaning* to the engineering effort performed by subsystems, software and hardware teams. Third, the rigor and formalism introduced by the use of models enable *faster impact analysis* and hence a faster reaction to evolutions in expectations and constraints, i.e. the agility of systems.

An effective and high-performance implementation of agility in systems engineering imposes new requirements on engineering tools. First, the fast access, consistency and the integration of data produced by different sources, and particularly those issued from technical and technical management process (e.g. integrate Capability increments and releases management with configuration management and business strategies): this not only demands an effort on the integration of tools, but also on organizations' processes and mindsets. Second, agility requires the development and maintenance of multiple architectural alternatives until one of them is selected to be further developed to fulfil the expectations of the stakeholders: this requires developing means to early-evaluate architectural alternatives and to efficiently reuse existing building blocks. Last, agility requires the continuous integration and validation at the system level to ensure that global optimums are reached, instead of by-increment local optimums: this requires the development of means to simulate and automatically check system-level expected properties at a very fast pace.

References

Dove, Schnidel 2018. Agile Systems Engineering Life Cycle Model for Mixed Discipline Engineering. 28th Annual INCOSE International Symposium 2018.

Scrum 2019, https://www.scrum.org/

SAFe 2019, https://www.scaledagileframework.com/

LeSS 2019, https://less.works/

INCOSE, 2014, 'A World in Motion – Systems Engineering Vision 2015'.

ISO/IEC/IEEE 15288, 2015, 'Systems and software engineering – System life cycle processes'.

IEEE 1220, 2005, IEEE Standard for Application and Management of the SE Process

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

Capella, 2019, Capella Polarsys Website: https://www.polarsys.org/capella/

Bonnet S, Voirin J-L, Normand V , Exertier D, 2015, 'Implementing the MBSE Cultural Change: Organization, Coaching and Lessons Learned', 25th INCOSE International Symposium

Navas, J, 2019, 'How MBSE practices support the effective implementation of a Product Line Engineering approach', 29th Annual INCOSE International Symposium, 2019.

Bonnet S, Voirin J-L, Navas J., 2019, 'Augmenting requirements with models to improve the articulation between system engineering levels and optimize V&V practices', 29th Annual INCOSE International Symposium, 2019.

ISO/IEC TR 24748-1, 2010, 'Systems and software engineering — Life cycle management — Part 1: Guide for life cycle management'

Biography

Juan Navas is a Systems Architect with 10-years' experience on performing and implementing Systems Engineering practices in multiple organizations. He is in charge of the Thales Corporate MBSE coaching team and dedicates most of his time to training and other consulting activities worldwide, for Thales and other organizations. He accompanies systems engineering managers and systems architects implement MBSE approaches on agile operational projects, helping them define their engineering schemes, objectives, and guidelines. He holds a PhD on embedded software engineering (Brest, France), a MSc Degree on control and computer science from MINES ParisTech (Paris, France) and Electronics and Electrical Engineering Degrees from Universidad de Los Andes (Bogota, Colombia).

Stéphane Bonnet is Systems Design Authority for the Avionics Global Business Unit of Thales. He oversees system architectures for the five business lines and leads the system-level R&T. He holds a PhD in software engineering and has led for numerous years the development of Capella, an open source modeling workbench for systems, hardware and software architectural design. He has been an active contributor to the Arcadia method. For several years, he has helped engineering managers and systems architects implement the MBSE cultural change, with a range of activities spanning from strategic engineering transformation planning to project-dedicated assistance to modeling objectives definition and monitoring.

Jean-Luc Voirin is Director, Engineering and Modeling, in Thales Defense Missions Systems business unit and Technical Directorate. He holds a MSc & Engineering Degree from ENST Bretagne, France. His fields of interests include architecture, computing and hardware design, algorithmic and software design on real-time image synthesis systems. He has been an architect of real-time and near real-time computing and mission systems on civil and mission aircraft and fighters. He is the principal author of the Arcadia method and an active contributor to the definition of methods and tools. He is involved in coaching activities across all Thales business units, in particular on flagship and critical projects.

Guillaume Journaux is a Systems Engineering Manager in Thales Defense Missions Systems business unit. He is also a MBSE coach with an extensive experience on Arcadia-based modeling approach. He has an important software background and is interested in fighting against silos between System and software teams thanks to MBSE approach. He is now working on a very complex project on which he aims at coupling MBSE and AGILE methods.