A PRACTICAL ANALYSIS ON THE COMPLEXITY OF AIRCRAFT SYSTEMS ARCHITECTURE AND REQUIREMENTS DEVELOPMENT

Felipe Magno da Silva Turetta, EMBRAER

Keywords: *Systems Engineering, Complexity, Validation and Verification, Requirements Engineering*

Abstract

This paper analyses the "Development of system architecture and allocation of systems requirements to items" steps of the SAE ARP 4754, discusses the engineering reality and challenges associated. After a theoretical discussion three vectors of research are defined to apply practice theory concepts: Knowledge mapping, requirements management and testing strategies.

1 Introduction

Developing modern aircraft systems is a challenging task and as stated by Johnson [1] 'coping with the resulting design complexity while maintaining time to market and profitability is the latest challenge to hit the engineering industry.'. New appearing technologies allow pushing the boundaries creating more efficient products but the drawback is an ever increasing complexity of the on board systems. Such complexity creates the need for engineering practices to avoid design errors and the SAE document ARP 4754 consolidates guidelines with that goal. Still the need exists to improve safety margins and reduce design cycles. This paper proposes to analyze the steps "Development of system architecture and allocation of systems requirements to items", shown in blue Fig. 1 System Development Process [2], using practice theory concepts. Those steps were chosen because as stated by Leveson [3] "Almost all software-related accidents can be traced back to flaws in the requirements specifications and not to coding errors…", and from the system standpoint, this is where the number of requirements and their complex relations begins to grow rapidly. This indicates that these are crucial steps to search for improvements in order to avoid future accidents.

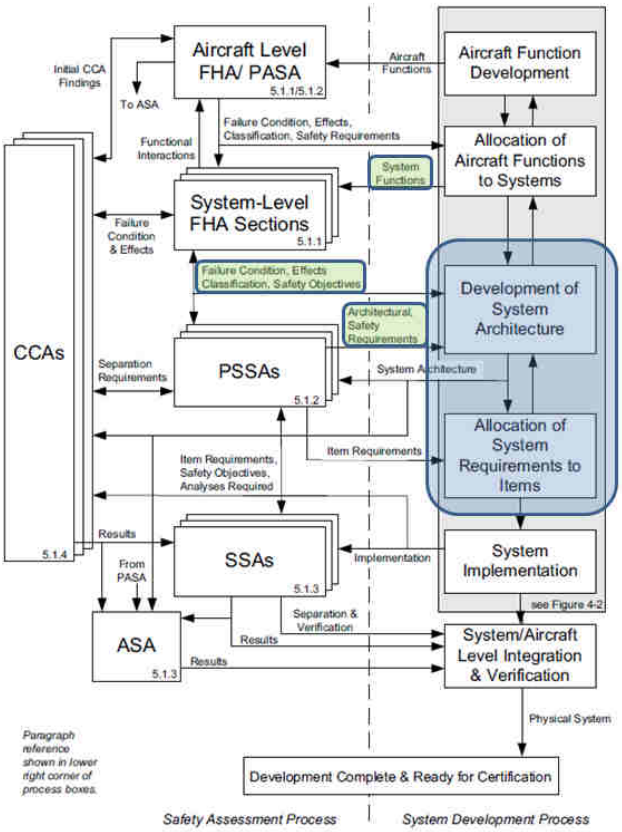

Fig. 1 System Development Process [2]

The development of system architecture is initiated with the inputs marked in green on Fig. 1 System Development Process [2]; the functions the system is supposed to perform and safety goals for each function. Then each proposed architecture will generate specific failure conditions and architectural safety requirements, as well as derived requirements. Later, each system requirement will be allocated to one or more system components and safety

process will generate additional requirements for those components that may in turn, prompt changes in the previously defined architecture or requirements. From [2] "In practice, system architecture development and the allocation of requirements are tightly-coupled, iterative processes. With each iteration cycle, the identification and understanding of the requirements increases and the allocation of the system-level requirements to hardware or software items becomes clearer". Here it already becomes clear that the engineers do not have all the knowledge about the system when the design process is started. “Designing and assessing architecturally complex computer systems is a classic but still open challenge” [6] and the goal of this paper is to identify how to increase the knowledge of the engineers earlier in the design process to avoid excessive re-work and, most importantly, not leaving residual unknown emergent behaviors in the system.
In chapters 2 and 3 a theoretical discussion is made on complexity, communication and knowledge in a development process. Chapter 4 defines the three vectors to apply practice theory concepts and chapter 5 presents the final conclusions.

2 Complexity and Emergent behaviors

There are several definitions for complexity in the literature and one of the most used is that complexity is related to “degree of difficulty in predicting the properties of a system if the properties of the system's parts are given” [12]. For engineers a practical definition that we can include is that a “system is complex when it becomes impossible to test all the combinations of different system inputs”. Fig. 2 Sample Engine/Bleed Air/Anti Ice systems Architecture shows a simplified architecture diagram of a hypothetic Aircraft Engine/Bleed Air/Wing Anti Ice System that would be considered complex by anyone not familiar with those technologies. Still it could be considered “quite simple” for an experienced aerospace systems engineer.
But that same engineer even considering the design “simple”, would recognize that it is almost certain that he and a team of other experienced engineers would make mistakes when designing such a system, or that some system behaviors would only be captured later in the development cycle, during the testing phase. This statement serves as an illustration of one recurrent definition of complexity, which is that complex systems are those that present emergent behaviors [1, 4]. It is a convenient definition because it removes a portion of the subjectivity of the evaluation, since even an experienced engineer that comprehends the system very well, will admit this property in “his” system.

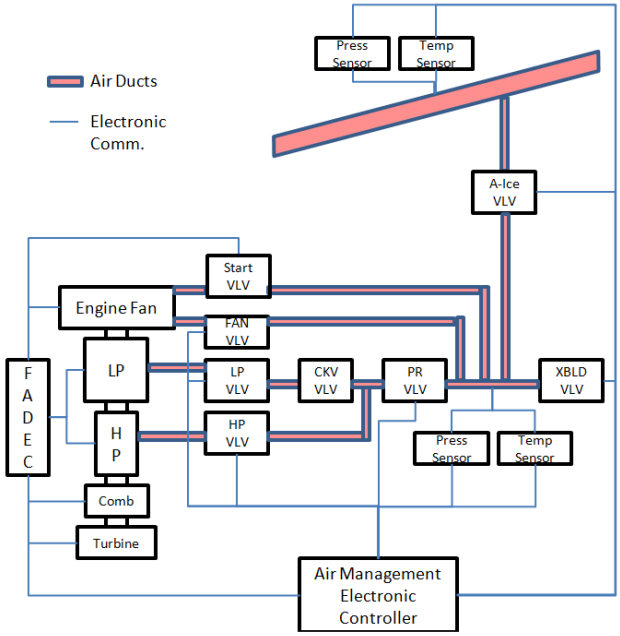

Fig. 2 Sample Engine/Bleed Air/Anti Ice systems Architecture

Emergent behaviors are those that stem from a system, in an unexpected manner, from the interaction of its multiple components between themselves and the environment. Some state that those properties cannot be identified through functional decomposition [1] and this idea challenges various engineering methods such as FMEAs and FTAs, heavily based on this kind of decomposition. This is also a problem for most traditional requirements engineering practices, since natural language text requirements do not interact with each other. Still this specific problem could theoretically be solved using model based techniques that can be simulated.
Bedeau [7] differentiates between strong and weak emergences, in where the weak events could be predicted using powerful modeling

techniques and the strong events would be those so complex that they are really "out of our grasp", like the emergence of life from inanimate matter. Strong events are characterized by what he calls 'downwards causation', meaning that the low level behavior is constrained by the higher level in what an engineer would understand as a constant 'closed loop feedback' dependence.

Johnson [1] also states that some authors deny this concept of emergence, saying that those behaviors have nothing special and are just manifestations of our knowledge gaps. This view may indeed be correct, but it is of no practical use, since in reality we do have knowledge gaps and even when we close them, as technology progresses new gaps are created and it is impossible to keep up because the process of innovation involves learning. Thus in practice, it does not matter if the emergent behavior is really undetectable or if it is "only a problem with our current knowledge", it is an issue that must be dealt with.

Although not explicitly stated in the documents, one of the main goals of the processes documented on SAE ARP 4754 and 4761 [2, 5] is exactly to prevent unwanted emergent behaviors to remain hidden in the systems. From ARP 4754: "Aircraft/System integration is the task of ensuring all the aircraft systems operate correctly individually and together as installed on the aircraft. This provides the means ***to show that intersystem requirements, taken as a group, have been satisfied.*** It also provides an opportunity to ***discover and eliminate undesired unintended functions***." (emphasis added). Those "unintended functions" are exactly emergent behaviors, as they were not included "by design", but emerged from the complexity of the systems integration. One of the most difficult tasks in systems integration is to find and evaluate those unintended functions, since there is no place to start looking for them. Requirement testing is not effective to capture those behaviors since no requirement exists to implement them. Usually a combination of engineering analysis and use cases based testing is done to search for them, but there is no accepted test coverage criteria to guarantee all those functions have been exposed and eliminated (or proven to be harmless). This is fertile ground for new research and is exactly the target of this initial research. From this point on in the paper, the terms "Emergent Behavior" and "Unintended Function" are used as synonyms.

3 Potential Source of Unintended Functions: Communication and lack of knowledge

To be able to uncover those functions it is necessary to know (or at least have a general idea) of the reasons why those behaviors ended up in the system. Although it is impossible to perfectly point out the reasons, it is possible to learn from previous experiences and target at least those sources. In this section and overview of a traditional development will be briefly shown, and then concepts related to communication and tacit knowledge are discussed. Those concepts will later be used to propose methods of minimizing potential unintended functions.

3.1 Development process

A traditional development process can be seen on Fig. 3 Traditional development with NL requirements. Simply put it consists in a first cycle where the requirements are elicited from the stakeholders and a requirement base is created and validated. As we will be explained later, this validation is limited since it is usually based on natural language requirements, and communication issues exist. For this reason it will be called "weak validation".

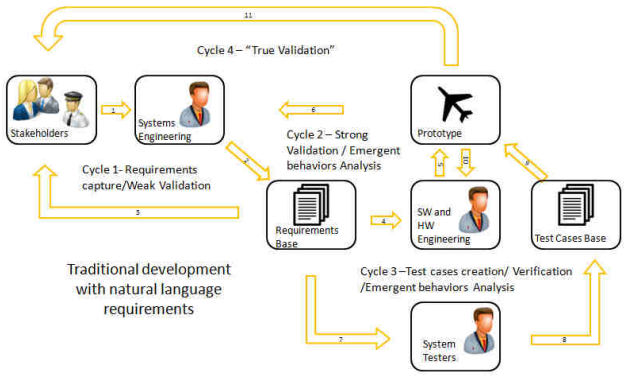

Fig. 3 Traditional development with NL requirements

The second cycle consists of software and hardware engineers building the first prototypes, based on the written requirements. In this cycle several issues start to surface like completeness of the requirement base (lack of information for implementation), consistency (conflicting requirements) ambiguous requirements and other. This allows for a better validation, since there is a "reification" of the written requirements and many communication issues surface. At this point the prototypes are usually far from the final configuration and final customer involvement is little in traditional processes and although agile methodologies advocate for customer involvement in all phases, even under those methods it may not be practical to call the costumer to see an electronic box controlling a prototype valve controlling the pressure of a manifold, when the real interface with him, (the air conditioning in this example) is not really there yet.

The third cycle consists of the test experts building a test base and exercising the prototype with those tests. This is where the implementation is verified to meet the requirements, and another communication barrier exists here, since the test engineers will have to interpret the requirements in order to design the tests.

Finally the cycle 4 is where the "true validation" occurs, since the prototype is near the final configuration and both the costumers and regulatory agencies specialists are largely involved. At this point a greater understanding of the requirements and system behavior is achieved due to extensive use of the prototypes and clarification of the requirements. Of course, problems found at this point in the development are much more expensive to solve than the ones that were found during cycle 1, where no material had been bought, no prototype was constructed and so on.

3.2 Communication and Sub-System Perspectives

As it became clear in the previous section, development of a complex system is about integrating knowledge, thus its success is highly dependent on good communication. And the more complex the system is, more different engineering specialties will be required, causing communication issues to not be restricted only between costumer/designer, but largely between designers of different parts of the whole system.

It is not possible for a single person to have the knowledge from all the integrated system. Thus considering each person involved in the process has mastery over one portion of the system (which is still not complete mastery), they will all have a different perspective from the "complete system".

Figures 4 and 5 show possible perspectives of the system from Fig. 2, as viewed by different specialists. Another possible perspective would be the one from an engine system engineer, where the whole Air Management System would be a black box (Anti-Ice plus Bleed). Also even inside the same specialty different views may arise due to different backgrounds (historical bodies) or development tasks assignment.

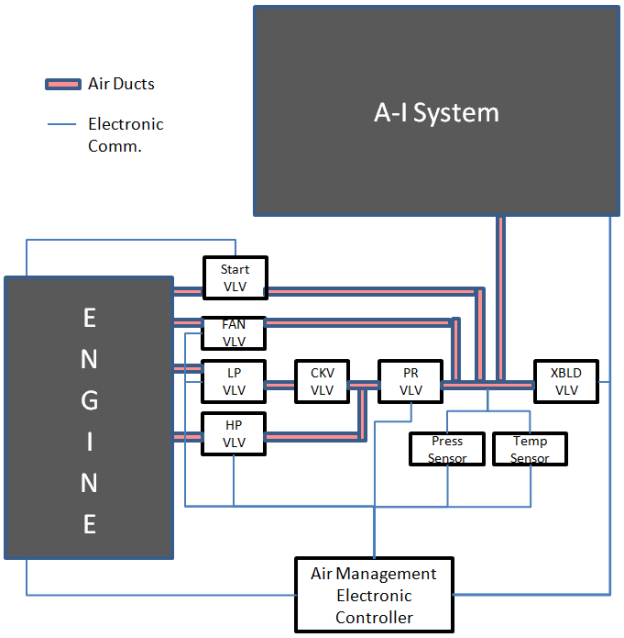

Fig. 4 Sub-System perspective from the Bleed System engineer

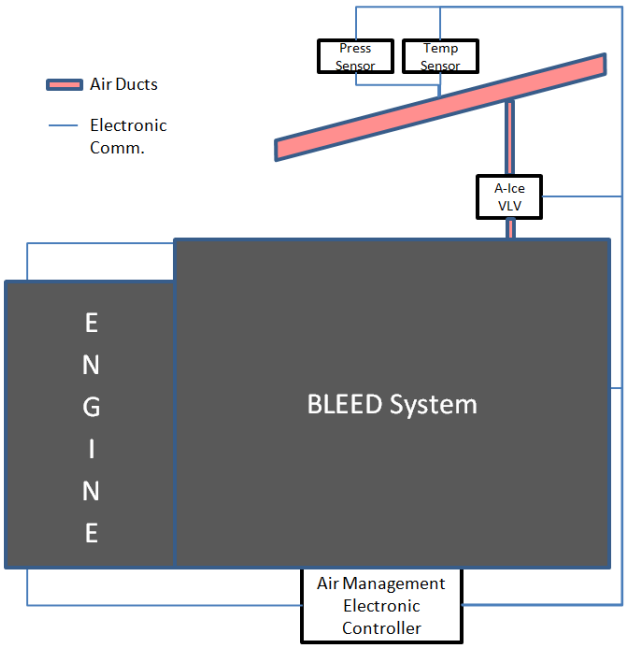

Fig. 5 Sub-System perspective from the Anti Ice System engineer

The System engineering Book of Knowledge present several types of complexity, and one of them is called structural complexity, with is related to the number of components of a system. This is directly related to the concept we are discussing here. In order to further illustrate the concept, for a system with N components, the total number of possible sub-system perspectives is given by equation (1). Where it is possible to combine N in groups of 'g' components, with g varying from 2 to N-1 (because g=1 means only one component, thus not a sub-system, and g=N means the whole system).

$$TN_{SSP} = \sum_{g=2}^{g=N-1} \frac{N!}{g!\,(N-g)!} \qquad (1)$$

Equation (1) shows that for the simple system from Fig. 2 Sample Engine/Bleed Air/Anti Ice systems Architecture which has 19 components, more than half a million potential perspectives are possible (and real aerospace systems usually have hundreds of components). Although not all those views are "plausible", the equation (plotted in Fig. 6 Growth of possible sub-system perspectives with increasing number of system components) serves as a powerful illustration of that growing complexity, and how even engineers from the same sub-system could easily have different system perspectives by missing just a single component. It also serves as an illustration of how quickly it becomes impossible to test all possible combinations of system malfunctions (thus classifying the system as complex by some definitions).

Those different perspectives can have no direct impact on how the person understands the system´s operation under normal conditions, but might have an enormous impact in specific abnormal conditions, that will appear to the team as an emergent behavior during testing or even real world operation.

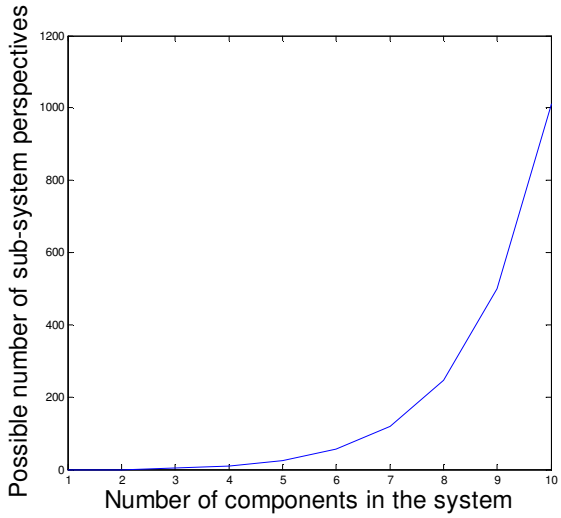

Fig. 6 Growth of possible sub-system perspectives with increasing number of system components

Those different perspectives may create issues during the development process that will emerge later during testing or even during operation. Below are two examples of design errors that could occur due to this lack of knowledge.

3.2.1 Sample design error 1 – Potential incorrect assembly

Note from the figures that both the BLEED system and the A-I system have Pressure and Temperature Sensors that are connected to the Air Management Electronic Controller. There is a possibility that the components are physically identical and thus have similar contactors that could theoretically be incorrectly connected in the controller. In this case, even if the system has "out of range" monitoring for the sensors, it

would not be able to detect the incorrect assembly unless a power up test is executed and the associated valves are exited (because without pneumatic pressure both sensors would be indicating ambient pressure). Thus the hazard being described here is an "Incorrect assembly of the pressure/temperature sensors". In this case, the issue could be resolved easily by including a power up test that opens each valve, and checks the pressure and temperature of its associated sensors. An incorrect assembly would be detected because of the lack of pressure/temperature rise once the valve is open. But the only way to include this hazard in the FHA is for an engineer to have an insight of this potential interaction of the components. And in order to have this insight, the system perspective of that engineer would need to include both the Bleed and A-I systems. The insights are limited by the engineer´s perspective. Unfortunately we know that due to the huge complexity, it is not possible for a single person to have the complete perspective, then the question is; ***how do we identify the relevant components from one system to be made explicit for the others?*** This dilemma is discussed by Ribeiro in [8] when he comments that in knowledge transfer between an expert and a novice, "the novice does not know what to ask and the expert does not know what to teach".

3.3.2 Sample design error 2 – Dynamic coupling of pressure controls laws: Bleed and Anti-Ice with states from FADEC

The pressure in the wing anti ice ducts is controlled by modulating the wing anti ice valve (A-Ice VLV in the figures), that uses air from the bleed manifold. The bleed manifold pressure is controlled by bleed pressure valve (PR VLV in the figures), that extracts air from the high or low pressure compressors from the engine, depending on bleed clients (typically anti ice, pressurization and air conditioning buy may include others) demand and engine current trust rating (takeoff, climb cruise or descend) and thrust lever angle.

The paragraph above illustrates that the two control laws for each valve are tightly coupled and depend on various external factors. Thus it is a challenge by itself to define under which conditions the laws should be tested considering normal and abnormal conditions. Which failures outside those systems may affect the dynamics of the system? Certainly the engine has controls and failures of its own that are also coupled to that dynamic.

It is possible for example that the engine specialists use the exhaust gas temperatures in some control law that influences the compressors pressure, but for the bleed engineer that characteristic is transparent (since he uses in his control law, the direct pressures in the compressors). Is the failure of the exhaust gas temperature sensor a relevant failure for the bleed control? It may not be relevant under certain conditions, for example with anti ice on but air conditioning off. This is an example of how it can quickly become impossible to test all possible system inputs. Then how should the tests be designed?

3.4 The types of tacit knowledge

In the introduction of [8] Ribeiro starts with a quote from Polanyi: "we know more than we can possibly say". He is referring to a type of knowledge that cannot be transferred through explicit instructions. This type of knowledge is described as tacit knowledge, as opposed to an "explicit" knowledge that could be transferred more easily.

One concept that helps understanding this kind of knowledge is "the regression of the rules" described by Wittgenstein in [9]. He states that the rules do not contain the rules for their own application, suggesting that in order to be able to adequately follow a strict rule, previous knowledge is always required and the only way to acquire that previous knowledge is by socializing in what he calls a "form of life". In our engineering world, each engineering branch could be understood as a different form of life (but the branches being far more detailed than just "mechanical", "electrical" or "industrial") and this generates the Sub-system perspectives described earlier. In our example it is clear that an anti ice system engineer would not be able to design an aircraft engine even with the most

complete books and procedures from the engine specialists. What is considered "obvious" inside one form of life (and thus omitted from procedures) is clearly not obvious to the other. Ribeiro distinguishes three different types of tacit knowledge: somatic, contingency and collective.

Somatic tacit knowledge refers closely to physical skills like riding a bicycle (that could never be really taught in the classroom for example) and needs to be experienced with its own body in order to be learned. This is not the point of interest of this research.

Contingency tacit knowledge are usually captured in the "out of the normal" situations. Since normal procedures address only a limited amount of scenarios, deviations from those scenarios may be intuitive to deal for someone experienced at that specialty, but very difficult for others. An example is the design of a overtemperature monitor that could be inside the controller software of the Anti Ice System of Fig. 2 Sample Engine/Bleed Air/Anti Ice systems Architecture. The novice engineer would simply program (in pseudo code):

if *temperature* > *100 degC*
    overtemp=true
Elseif *temperature* <= *100 degC*
    overtemp =false

But an experienced engineer with this kind of application would quickly recognize the need to treat intermittent transmissions issues, and would include the need for a "persistence" of for example 5 seconds, and would program the following:

if *temperature* > 100 degC
    if *persist* > 5
        overtemp=true
    Elseif
        persist=*persist*+1
Elseif *temperature* <= 100 degC
    overtemp=false
    persist=0

This kind of knowledge is, in principle transferable, although it is impractical due to time constraints and the difficulty the experts will have in delimiting it. Lessons learned databases, mentoring and other methods are used to deal with that contingency tacit knowledge.

Collective tacit knowledge is the one that allows the individual to stop Wittgenstein´s regress of rules, or in other words to interpret and correctly apply a set of rules from a specific form of life. Developing this kind of knowledge means joining a form of life; in our case, acquiring the skills of that engineering branch being able to "see" that sub-system perspective. The next section discusses collective tacit knowledge in more detail, as it is the trickier to deal with.

3.4.1 Collective tacit knowledge

Detailing the idea that collective tacit knowledge allows to stop the regress of rules, Ribeiro describes three types of judgment that only experts from a specific form of life are able to do; similarity / differences, relevance / irrelevance and risk / opportunity.

They are usually intertwined in engineering, and can be closely related to design trade-offs or safety analysis. Back to the monitor example in the previous section; is the temperature of 100 degrees C adequate for this situation? Imagining that the concern is damage to the wing structure (aluminum with a fusion point of 660 deg C), the limit seems far too conservative (with only this information we could judge that 100 or 150 is the "same"). But is fatigue a problem? There are any other system components that might be damaged installed nearby? Even with the temperature being far lower than the fusion point, might this temperature affect the material properties in the long term? What is the precision of the temperature sensor? In addition to the temperature, is the persistence time adequate? What if the system stays around 100 deg C for a long time never staying above that value for more than 5 seconds?

Some of those questions would never be asked by an experienced engineer, because they are irrelevant, other could never be asked by anyone other than an experienced engineer, because it would be impossible to have that insight. Some would come only from someone with a specific sub-system perspective. This illustrates the need not only of several engineering branches knowledge, but also of a necessary **mix of experiences** (The fatigue issue for example is

far from a systems engineer expertise, on the structures field).

4 Practice theory and epistemology for dealing with design complexity: Three research vectors

With the discussion done in chapters 2, 3 and 4 and epistemology we will define three main research vectors.

Epistemology is the branch of philosophy that studies knowledge itself. Ratcliff discusses in [10] potential applications of epistemology in systems engineering and even the Systems Engineering Book of Knowledge mentions that it is "one of the foundations of integrative systems science". After the discussion done here in section 2 on complexity and emergent behaviors and how this may be seen by some authors as a "lack of knowledge", the connection between epistemology and systems engineering should be quite clear.

The three vectors will be briefly addressed here, but not exhausted. Further studies shall be made to stress the subjects. They are: knowledge mapping, requirements management and testing strategies.

4.1 Knowledge Mapping

In [8] Ribeiro studies the startup process of a complex industrial plant, having to deal with a considerable amount of new employees with no experience on the plant processes and a limited number of experienced personnel. To manage the formation of the teams for each part of the plant he proposes a process called "Tacit knowledge management and similarity levels mapping". A similar process could be used in the formation of aircraft systems development teams, or at least, to guide the need of participants in design review meetings.

The idea consists (in a simplified manner), to consider in addition to the time of experience each specific has, the "level of similarity" of the task he is to perform. For example, if an engineer has 10 years of experience with engine systems, but most of those years have been spent working on turboprop engines, he has a considerable experience on engine systems, but if he is working on an aircraft with turbofan engines, his similarity level would be only medium. If he is working on aircraft with a scram jet engine, the similarity level would be low. This means that he has limited tacit knowledge on those projects, and is not as fluent in making the judgments described in section 3.4.1. This may have a considerable impact on the decision making capabilities of an engineering team, especially in very multidisciplinary ones. Imagine that a team of ten engineers is making a decision about the design of the system in Fig. 2 Sample Engine/Bleed Air/Anti Ice systems Architecture There are three engineers for each sub-system (with their different sub-system- perspectives) and a integration systems engineer. Each one of them has more than 15 years of experience in aviation, some in more than one system. It is expected that they would feel very comfortable in making design decisions since there is a lot of experience and knowledge gathered. What may be hidden is the fact that, for example, no one in that room has ever worked with a specific technology contained in one of the systems. It would be normal to assume that "somebody here knows about ***that***". The problem may be even hidden in some subsystem that is not depicted in the architecture under study, for example the electrical system that feeds energy to all the components in all those systems.

The previous paragraph illustrates the need for an active knowledge management during system design. The experience and similarities levels mapping proposed by Ribeiro in [8] might provide a good starting point to develop that kind of management procedures.

4.2 Requirements Management

Ratcliff discusses in [10] how different cultures (or even engineering disciplines) may have different definitions for certain terms resulting in different or conflicting requirements, or even (potentially the worst case) one requirement being understood completely differently by two different stakeholders. He mentions the United States DoDAF document [11] that maps different stakeholders viewpoints that should necessarily be considered when designing a system architecture. Still he points out that simply having iterative discussions and

"translations" of the requirements between the different viewpoints, may instead of solving the issue, exacerbate it to a point where no viewpoint is actually represented and traceability of the requirements may be lost. (Those viewpoints may be compared to the concept of "sub system perspectives" presented here, but in reality it is a much broader concept, considering the complete lifecycle of that system or activity, and the sub system perspective aims to map only the understanding of that specific system architecture.)
At the end of his work he proposes a step by step process to try to avoid those pitfalls, and one of the steps is actually to model the requirements into petri nets and simulate them. This appears to be a very efficient solution and we discuss it further in the following section, model based design.

4.2.2 Model based design

Although model based design is not a discipline from practice theory, many of its foundations match very well with intents identified during the discussion of tacit knowledge. The idea of having executable models as requirements instead of natural language text removes communication barriers as the model behavior can be directly observed. Text requirements do not interact and thus present no possibility for emergent behaviors to appear.
Ratcliff asks in [10], "How can one know, or even can one know, if the requirement statements purportedly regarding the same phenomena, as stated in two separate viewpoints, actually refer to the same concept?". This issue can be solved by modeling the requirements, because the phenomena can be "seen" through the behavior of the model.
Fig. 7 Model Based Development shows a sample development cycle with MBD, and it becomes clear in comparison with the process from Fig. 3 Traditional development with NL requirements a stronger validation is possible is cycle 1 due to the possibility of exercising the model with the stakeholders. Also weak emergent behaviors might be detected on cycle 1, even those that could not be anticipated by the costumer/stakeholders. Also the task of creating the test cases and even constructing the first prototypes is much easier since the testers and other engineers are allowed an early "socialization" with the modeled system behavior.

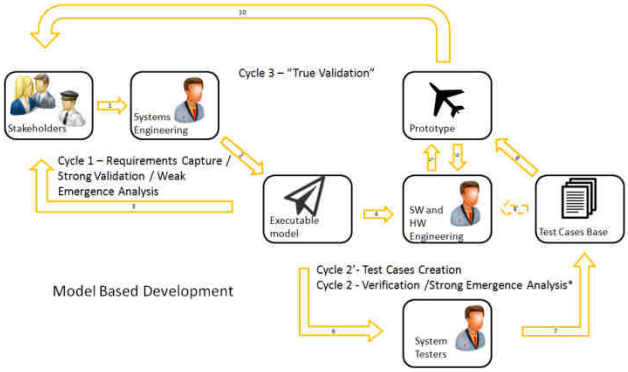

Fig. 7 Model Based Development

4.3 Testing Strategies: An "epistemological loop" in the search for unintended functions.

Ratcliff illustrates typical epistemological questions with "how do we know this?", "is this fully know; can it be fully known?". This brings us back to the point of emerging behaviors and that fact that apparently, ***complex systems cannot be fully known,*** at least not until later in its lifecycle, where it has been largely exited in different situation. This brings up the question: If the system is not fully known, how do we test it? This appears to be an "epistemological loop", since it is impossible to design a test to search for something you not know what it is. This also strongly questions the validity of requirement based testing in order to search for unintended functions. If a function is unintended, there is no requirement to implement it, thus it may be impossible to find it with tests designed from requirements. Since complexity theory explains that emergent behaviors cannot be uncovered through functional decomposition, it appears that it may be indeed impossible to uncover all unintended functions hidden in a complex system.
With this rationale it appears that the best way to look for unintended functions is to exercises the system in the largest possible combination of use cases scenarios and analyze the system response. Note that those tests will need to come directly from the use cases, without taking into account low level requirements. If those low

level requirements are considered they may "contaminate" the test procedures with operational sequences that were already considered during the design.
It still can be argued that it is impossible to uncover all unintended functions until a full factorial test is performed, and this is potentially true for complex systems, but the goal here is to uncover at least the unusual but plausible situations. Thus we will introduce a new concept of test coverage criteria the Operational coverage criteria. It will be divided into 'theoretical' and 'practical' operational coverage. A system is said to have 100% theoretical operational coverage when all possible operational scenarios have been exercised. This is of course impossible, which is why we need the "practical" operational coverage concept, which we will define as:
'A system is said to have 100% practical operational coverage when a full factorial design of the use case scenarios has been performed, but not of the input variables.'
Even the practical definition may generate an impractical number of tests, but it will at least provide some insight and measurement on how well the operational scenarios have been explored (and thus the likelihood that new unintended functions may appear).
The concept is included here for the aviation community evaluation (as this is a preliminary research article), but it will need to be detailed in a later article.

5 Conclusions

This paper is but the first step in a marathon to apply more practice theory into the design of complex systems, but it has been able to show that there are fertile grounds for applying practice theory into systems engineering. Further works include defining a procedures for Tacit knowledge and similarity levels mapping for engineering teams (from section 4.1), detailing the concept of operational coverage and how the design and manage the tests to attain it and evaluation the state of the art requirements capture and model based design techniques with practice theory to search for improvements. Studies on top of real cases are mandatory in the next phases of this research.

References

[1] Johnson, Christopher W. What are emergent properties and how do they affect the engineering of complex systems? *Reliability Engineering and System Safety* 91 pp 1475–1481 (2006) .

[2] Society of Automotive Engineers**.** *ARP 4754 Guidelines for Development of Civil Aircraft and Systems*. REV. A, SAE, 2010.

[3] Leveson, N. G. *Systemic Factors In Software-Related Spacecraft Accidents*. Conference: AIAA Space 2001 Conference and Exposition • August 2001.

[4] Dekker, S. Ciliers, P. and Hofmeyr, JH. The complexity of failure: Implications of complexity theory for safety investigations. *Safety Science,* Vol. 49, pp 939–945, 2011.

[5] Society of Automotive Engineers**.** *ARP4761 Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment.* Original, SAE, 1996.

[6] Iacono, M. Gribaudo, M. and Pop, F. Modeling and evaluation of highly complex computer systems architectures. *Journal of Computational Science. Vol.* 22. pp 126–130, 2017.

[7] Bedau M. Weak emergence. *Philosophical perspectives: mind,causation, and world.* Vol 11 pp 375–399, 1997.

[8] Ribeiro, R. Tacit knowledge management. *Phenomenology and the cognitive sciences*. 2012.

[9] Wittgenstein, L. Philosophical investigations. *Oxford: Blackwell* (1976 [1953]).

[10] Ratcliff, R. Applying epistemology to system engineering: an illustration. *Procedia Computer Science* 16, pp 393 – 402, 2013.

[11] DoD. (2010, August) DoD Architecture Framework Version 2.02. [Online]. http://dodcio.defense.gov/Library/DoD-Architecture-Framework/.

[12] BKCASE. Guide to the Systems Engineering Body of Knowledge (SEBoK) v1.8.

8 Contact Author Email Address

If you desire to contact the authors please mailto:felipeturetta@gmail.com

Copyright Statement