Global Systems Engineering at Rolls-Royce: Using Stories to Explore Ambiguous Standards and Emergent Architecture

Malvern J. Atherton, Shawn T. Collins,
Stephen A. Fisher
Control Systems Engineering
Rolls-Royce
PO Box 420, Indianapolis, IN 46206
malvern.j.atherton@rolls-royce.com,
shawn.collins@rolls-royce.com,
stephen.a.fisher@rolls-royce.com

Ralph W. Hill
Control Systems Engineering
Aero Engine Controls
Derby, UK
ralph.w.hill.JVAEC@rolls-royce.com

Abstract

This paper discusses Rolls-Royce' experience in implementing a standardized set of roles and control system requirements for its product development projects. It reviews the responsibilities of systems and software engineers, the task of system architecture and its implications on documentation structures. Industry guidelines allow for the definition of more detailed item-level requirements arising from the system architecture process. However, there is considerable ambiguity in the information transfers to the component life cycle processes and the distinction between these "item/ derived requirements" and "system requirements" in general. A review of projects shows that Rolls-Royce programs have developed different approaches to the system architecting process. This demonstrates the variety of interpretations for the industry standards such as DO-178B and ARP-4754. It also provides a broad set of potential resources for a globally managed development program. However, the ambiguity in the industry standards means that the system architecting process is not always straightforward (as evidenced by its varying Rolls-Royce embodiments). The paper concludes by developing stories and use cases that help to address differing cultural perspectives, and to replace existing local stories with a new set of stories that can be globally recognized and adopted.

Introduction

Rolls-Royce is executing a program to standardize its Systems and Software processes globally as part of an overall effort to define its engineering function roles, improve management of interaction between roles, and facilitate movement of engineering work between sites due to capacity limitations that are identified by the business opportunities. Rolls-Royce views standardization as an enabler to establishing common process for developing control systems and components, locating information within a documentation structure, and facilitating re-use of information across its various engineering sites. This paper discusses Rolls-Royce experience in implementing a standardized set of roles and control system requirements on its new engine projects.

Establishing processes that are globally effective requires that each project shares a common understanding of the stated documentation structure. It also requires a common understanding of the roles and responsibilities of engineers, and the implications this may have on organization structures. This challenge should be viewed as primarily a matter of

organizational change, which can be viewed through three lenses (Ancona, et al. 1999). The strategic design lens focuses on tasks, and considers the observable characteristics such as the organization structure, role definitions, and deliverables. The political lens focuses on stakeholders' interests, power and influence etc. The cultural lens focuses on history, and the stories that individuals use to shape perspectives and make judgments.

Many organizations fail to manage change successfully because they primarily focus on the strategic design lens, which is the most tangible. The political lens is considered only through the act of seeking agreement from stakeholders. It is often not analyzed to understand the underlying forces such as past investments in processes, tools and training, which cannot be easily removed. Finally, the cultural lens is frequently ignored due to its intangible nature, despite being one of the most powerful forces that affects commitment, judgment and perceived value of a proposed change (Ancona, et al. 1999).

The rest of the paper proceeds as follows. The paper begins by addressing the issue of global alignment through the strategic design lens based on role definitions in general industry standards. A review of Rolls-Royce programs shows different approaches to the system architecting process. The terms *systems engineering* and *software engineering* in this document are intended to be consistent with the usage in ISO 15288 (systems life cycle processes) (ISO 2002) and ISO 12207 (software life cycle processes) (ISO 2004). This demonstrates the variety of interpretations for the industry standards such as DO-178B (RTCA 1992) and ARP-4754 (SAE 1996). It also provides a broad set of potential resources for globally managed development programs. The ambiguity in the industry standards, however, means that the system architecting process is not always straightforward (as evidenced by its varying Rolls-Royce embodiments). The paper concludes by discussing hypothetical scenarios from Preliminary and Detailed Design activity to illustrate the task relationships discussed in this paper. Using scenarios to clarify assumptions about behaviors and responsibilities is one way to provide input from the political and cultural elements of organizational change to the strategic design lens that frequently governs organizational alignment efforts. It helps an organization address differing cultural perspectives and identify behavior patterns that can be globally recognized and adopted.

Systems and Software Role Boundaries

Role Definitions at Rolls-Royce

The Aircraft Engine Control System Engineer at Rolls-Royce is responsible to translate engine level constraints and airframer requirements into functional requirements and architectural design decisions for an engine control system. Their responsibilities include safety and reliability analysis, verification testing at the engine and flight-test level, and control system certification.

The Control System Software Designer is responsible to assess and understand the customer requirements on the software, produce software requirements which meet the customer needs, and produce software designs within the context of the software architecture that efficiently and effectively implement the software requirements.

The Control System Software Architect provides the main technical liaison between the software development team and the system and hardware platform developers. They provide the primary technical leadership on the software project, and are accountable for the overall architectural concept and design of the software system.

Both the systems engineer and the software designer must look at "customer requirements." According to Rolls-Royce control system development definitions, the systems engineer should "*analyze customer requirements and translate into functional requirements for the system.*" The software designer should "*assess and understand the customer requirements on the software and produce software requirements which meet the customer needs.*"

These objectives appear conflicting if we assume that they refer to the same customer. In practice, the customer of the engine control system should be the whole engine organization responsible for integrating the control system into the engine. In practice, the domain knowledge required for managing many of the airframe and inter-subsystem interfaces requires that control systems, software and hardware engineers will often need to work directly with the airframe customer or end user.

The system engineer's role is to translate customer requirements into functional requirements and the system architect's role is to allocate them to component level. The software designer's role does restrict their interest to only the "customer requirements on the software." This suggests that most inputs to the software requirements elicitation process will not be from the airframe customer or even whole engine integrator, but from the systems engineering process.

Roles in General Industry Standards

DO178B (RTCA 1992) is an industry standard that provides guidance for developing software in aircraft systems. In the DO178B descriptions for systems and software life cycles, external customer requirements are an input to the systems life cycle process. The systems life cycle process provides "system requirements allocated to software, including functional requirements, performance requirements, and safety-related requirements." The systems process is important here because much of the complex behavior emerges from interactions between physical components or between the pilot and physical components.

Aerospace engine control systems primarily lie towards the lower left of Figure 1. In this environment, software requirements become highly constrained by the preceding systems engineering process. However, in other industries, if most stakeholders and actors are human, with limited physical parts coupling, then systems engineering is less important. The overall architecting process can primarily be within the software engineering framework. As the actors shift towards physical parts, however, systems engineering is needed to manage the interfaces, allocations etc.

Expressed another way, in software systems with predominantly human actors and stakeholders, the software engineer can talk to the actors to elicit requirements, constraints and so on. By contrast, an engineer can't talk to a thermocouple, metering valve, or compressor to determine the constraints and requirements relevant to the software design.

The system life cycle process describes the architectural design process as generating an "*implementable set of system element descriptions that satisfy the requirements* " (RTCA 1992). Thus, the output of the system architecture is not necessarily decomposed requirements, but descriptions of components (referred to as *elements* in (ISO 2004)). The architectural design process generates only those *requirements* which arise from the system architecture definition itself (i.e. derived requirements).

ISO 12207 (ISO 2004) describes the software development process as potentially including system requirements analysis that a software developer "*shall perform or support as required by the contract.*" The handover from the system life cycle process to the software life

cycle process should be determined for each project as part of the *contract* between the organization managing the system life cycle process and the organization managing the software life cycle processes. Agreement to this relationship falls within the *agreement process* (ISO 2002), emphasizing the need to agree the boundary and relationship between the systems life cycle and the software life cycle process, rather than prescribing that boundary in the standard.

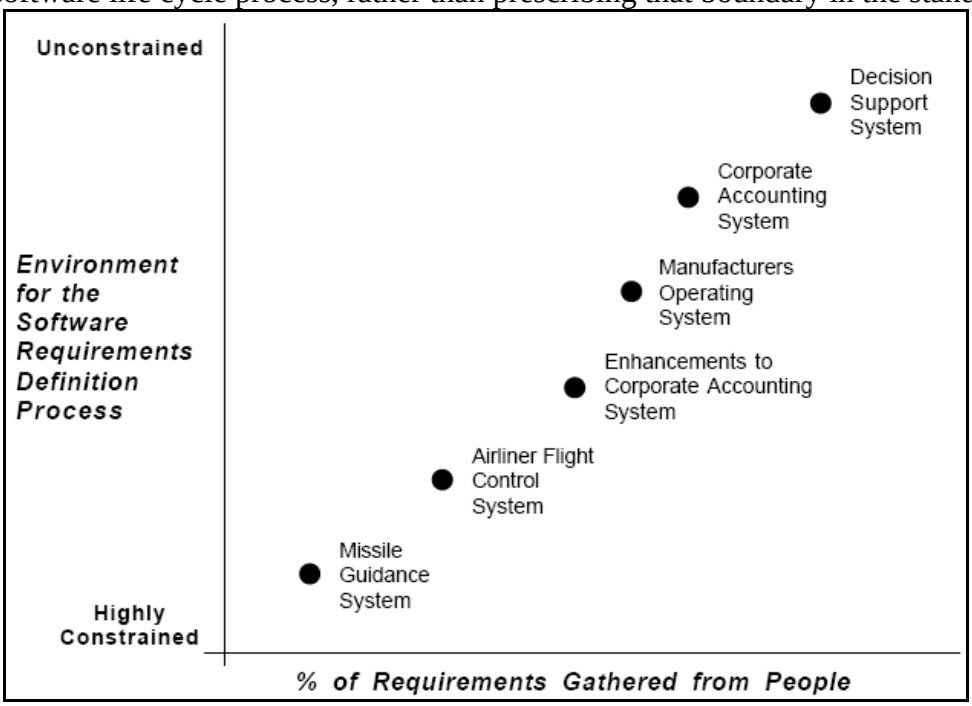

Figure 1 – Sources of Software Requirements (RTCA 1992)

Roles in Aerospace Standards

The discussion in the previous section does not imply that the systems engineering process should define the software requirements. Rather, it is intended to warn of the danger of using general software engineering principles in the aerospace environment. The guidance in DO178B and ARP4754 has been developed specifically for the aerospace industry, and hence should be compatible with the highly constrained software engineering environment depicted by the lower left area of Figure 1. These guidance documents state that the systems engineering life cycle process defines *system requirements*, and the software engineering life cycle process defines *software requirements*. This is consistent with the ISO/IEC standards.

It would be tempting to refer to DO178B and ARP4754 to determine the scope of system and software requirements. However, a weakness of both documents is that they do not fully account for the activities needed in the system architecture process. DO178B refers to the "system requirements allocated to software". It might be tempting to assume that the system requirements are from a flat structure in which some go to hardware and others go to software. DO178B does recognize the existence of system architecture by stating that "*Inputs to the software requirements process include the system requirements, the hardware interface and system architecture (if not included in the requirements) from the system life cycle process.*"

System Architecture Interpretation

The fact that both system requirements and system architecture are identified as inputs implies that some of the "zooming" required to create the system architecture (Crawley 2004) could be performed by the software life cycle process. That is, DO178B implies that software engineers can act in the realm of *systems engineers* by looking at the interaction of system components, assessing system properties (e.g. noise or accuracy) which could affect the software design. The guidance that "*if not included in the requirements"* implies that requirements arising from system architecture could be included in system requirements themselves. This removes the need to look at system architecture directly as an input to the software requirements development process. Hence, some degree of choice is available within the DO178B guidance. DO178B also shows that the software requirements flow back up to the system process for assessment by the safety assessment process. This is part of the software requirements validation process. This relationship is discussed further below.

One objective of the software engineering process, which includes software architecture, is to manage interactive complexity within software modules to ensure that unnecessary coupling is avoided, and beneficial cohesion between modules is obtained. This supports future developability and verifiability of the software. These are matters of software architecture that need to be managed within the software engineering process to ensure the product meets its safety requirements, and can be maintained and developed efficiently. If the system life cycle process overly constrains the software requirements, and hence the software architecture and development processes, then these objectives may not be achievable.

However, the systems engineering processes, which include control system architecting, involve allocation and decomposition to the component level, and managing the interactions between components. This includes documenting dependencies between component and functional boundaries. For complex hardware-in-the-loop systems, this objective often requires significant constraints to be imposed, which reduces the design freedom of the software process.

An alternative interpretation is to view the high level software requirements as being jointly owned by systems and software, in which each process advocates a different attribute of those requirements. In this view, requirements can be regarded both as the transfer of information from one organization to another, and as an agreement between them.

With lots of money and time, a similar arrangement could in theory be adopted between the system and software processes – that is two documents, one addressing functional behavior visible at the system level and the other addressing developability and verifiability at the software level – but without a decomposition/ allocation layer in between. However, this would fail to address the constraints which one process needs to impose on the other, and a sub-optimal solution could still emerge. A better approach would be one set of jointly-owned requirements, managed by the software engineering process (which ensures compliance to the objectives of DO178B), but in which the upstream system architecture process of decomposition and allocation to component level manages the functional behavior arising from those software requirements. The result would be that both processes constrain the design freedom of the other in a positive way. Only a solution which balances the potentially competing interests of functionality and modifiability (systems) and developability, verifiability, coupling and cohesion (software) would be selected.

Implications of System Architecture on Roles and Certification Compliance Processes

The role definitions in general industry standards discussed in Section 0 suggests that the standards do not adequately address the impacts that system architecture and dependencies across system architectural boundaries have on the software requirements development process. This highly complex relationship triggered the need to develop ARP 4754 (SAE 1996) to clarify the system life cycle processes which interface with the software life cycle. ARP 4754 acknowledges that this boundary is not obvious, and hence differing implementations across industry projects should not be unexpected. Given this ambiguity, we focus attention on the system life cycle processes and system architecture process to help clarify a suitable systems / software interface for Rolls-Royce.

Crawley (Crawley 2004) defines the role of the System Architect as defining system boundaries, goals, and functions, creating a system concept, allocating functionality, and defines interfaces. The system architecture process involves a *zooming process* and *decomposing form*. The system architecture should yield a more detailed functional requirement definition (i.e. process definition) as an output of the architecting process, rather than just a definition of and allocation to components and interfaces (i.e. physical decomposition). It is nearly impossible to zoom the process (i.e. define functionality) without a vision of the object concept (i.e. physical decomposition) to guide you. This means that decoupling the zooming of process from the system architecture process, as implied by DO178B, is not desirable. Furthermore, the task must be performed as part of the system architecture process, and hence within the systems life cycle.

ARP 4754 states that "Selection of the architecture establishes additional requirements necessary to implement that architecture. At each phase of the requirements identification and allocation process (i.e. system, item and hardware/software) both additional detail for existing requirements and new derived requirements are identified" (paragraph 5.1)

ARP 4754 describes the additional requirements arising from the architecture process as "derived requirements" that "*should be captured and treated in a manner consistent with other requirements applicable at that development phase*" (paragraph 5.3). This implies that they can be combined with higher level system requirements. It creates a somewhat messy or incomplete output from the system architecture process. It suggests that the high level system requirements are the preferred definition of desired architecture-independent behavior, and that derived requirements arising from architecture, are only needed to account for hardware dependencies.

However, it is argued here that the system architecture process should produce a clearly identifiable requirements layer which covers the complete set of decomposed and allocated physical and functional partitions in the system. Figure 3.2 of ARP 4754 shows a system development process which includes tasks for "Allocation of Aircraft Functions to Systems," "Development of System Architecture," and "Allocation of Item Requirements to Hardware & Software." This could be interpreted as implying that the system development process provides to the component development processes the fully partitioned and allocated requirements which could be used for component architectural development. This conflicts its own guidance on the role of derived requirements.

ARP 4754 attempts to provide clarification by emphasizing that derived requirements are often identified during the systems architecture process, and that they may change original higher level requirements. However, the guidelines create considerable room for interpretation or differing implementations depending on the needs of the project/ system.

In summary, the industry guidelines do seem to allow for the definition of more detailed item-level requirements arising from the system architecture process, but there is considerable ambiguity in the information transfers to the component life cycle processes and the distinction between these “item/ derived requirements” and “system requirements” in general.

DO178B lists the inputs to software from systems as including “*system requirements, the hardware interface and system architecture (if not included in the requirements).*” Figure 2 illustrates an interpretation of DO178B from a system architecture perspective. Because DO178B defines a software design process from which high level software requirements emerge, this implies an activity performed at the same level as the system architecture itself. ARP 4754 lists information flows from the system to the software life cycle processes. It includes “*Requirements allocated to software*” rather than *System Requirements*.

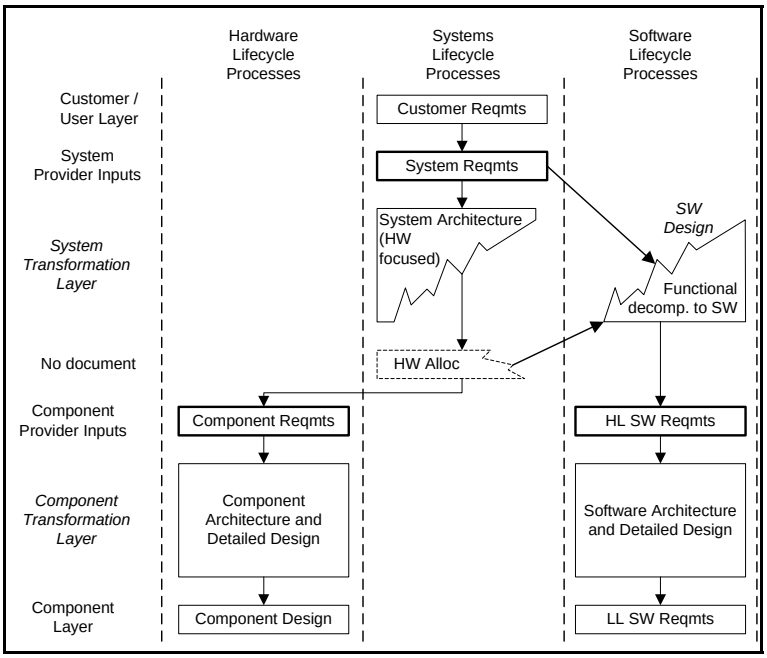

Figure 2 – System Architecture Perspective of the DO178B Process

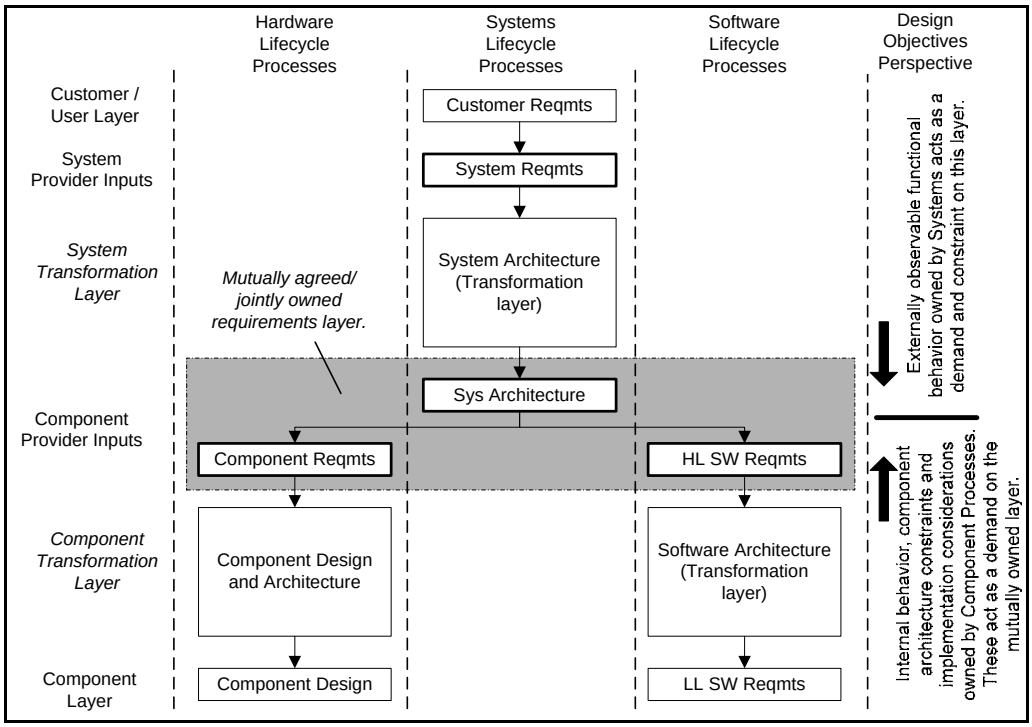

Figure 3 – System Architecture Based Proposed Flow-down

This description is consistent with our recommended structure (Figure 3), in which a system architecture definition emerges from the system architecture process, and component level requirements are developed by the appropriate component level process life-cycles in close cooperation with the systems architecture process. The structure in Figure 3 can also be compared to the documentation structure implied in Intent Specifications (Leveson 2000, Leveson and Weiss 2004, Navarro, et al. 2001).

Review of Rolls-Royce Experience

This section reviews Rolls-Royce Control Systems project documentation structures and process implementations and two geographically separated locations (SITEA and SITEB).

Organizational Boundaries – Functional versus Physical Decomposition

SITEA programs have not traditionally attempted to organize the systems or software teams around well defined functional boundaries in the way that the SITEB programs have. The early SITEB projects developed a systems design team divided according to functional boundaries of the control system. Figure 4 shows how requirements are allocated at the Control System Specification level according to functional boundaries.

The main advantage of this approach is workflow management. The engineer responsible for a function will be responsible for the technical content of all the deliverables which are affected by the function. For example, the requirements, system safety process (e.g. reviewing fault trees), responding to customer actions, addressing questions or feedback from component definition teams (software and hardware), supplier raised issues, system verification issues etc. This allows that engineer to be an expert on their function. One disadvantage is that the overall

systems perspective may get missed. However, overseer roles and interfacing roles are also developed, including the system architect, systems specifications coordinator, system design team leader, separate system safety team, separate system modeling team etc.

Risk Management Focus – Development Risk versus Certification Risk

The benefit of functionally oriented specifications is that they manage design risk well, where emphasis is on handling uncertainty in engine design and controls hardware integration. They do not manage certification risk as well, where emphasis should be on showing compliance to certification requirements and test coverage. In the SITEB model, system verification is performed against a mixture of sources and not purely from the requirements. This is not uncommon in complex system design. The approach naturally evolves on larger projects where the functional complexity increases to the point where no individual systems engineer is able to manage the functional and physical decomposition alone, and hence functions are partitioned out to different engineers. In the SITEB model, the need for integration at the design level is provided by a system architect, who acts alongside the system design engineers. The emphasis in this model is on managing design complexity and efficiently handling design change.

The SITEA model works well to develop evidence needed to support certification, including compliance to ARP 4754, DO178B and DO254 process objectives. System requirements have been developed with a greater emphasis on system testability rather than to support the design process at the next level down. On mature programs, lower level design freedom is necessarily simpler because it is heavily constrained within the existing architecture. Where complex changes are implemented, up front prototyping and simulation analysis is used to validate the design. This is often documented in an engineering report before the formal change control process is used to make the change in production. The formal change process is then able to focus on system verification. Although a good model for post Entry-into-Service modifications, it lacks the focus on design risk that is needed during development of a new control system.

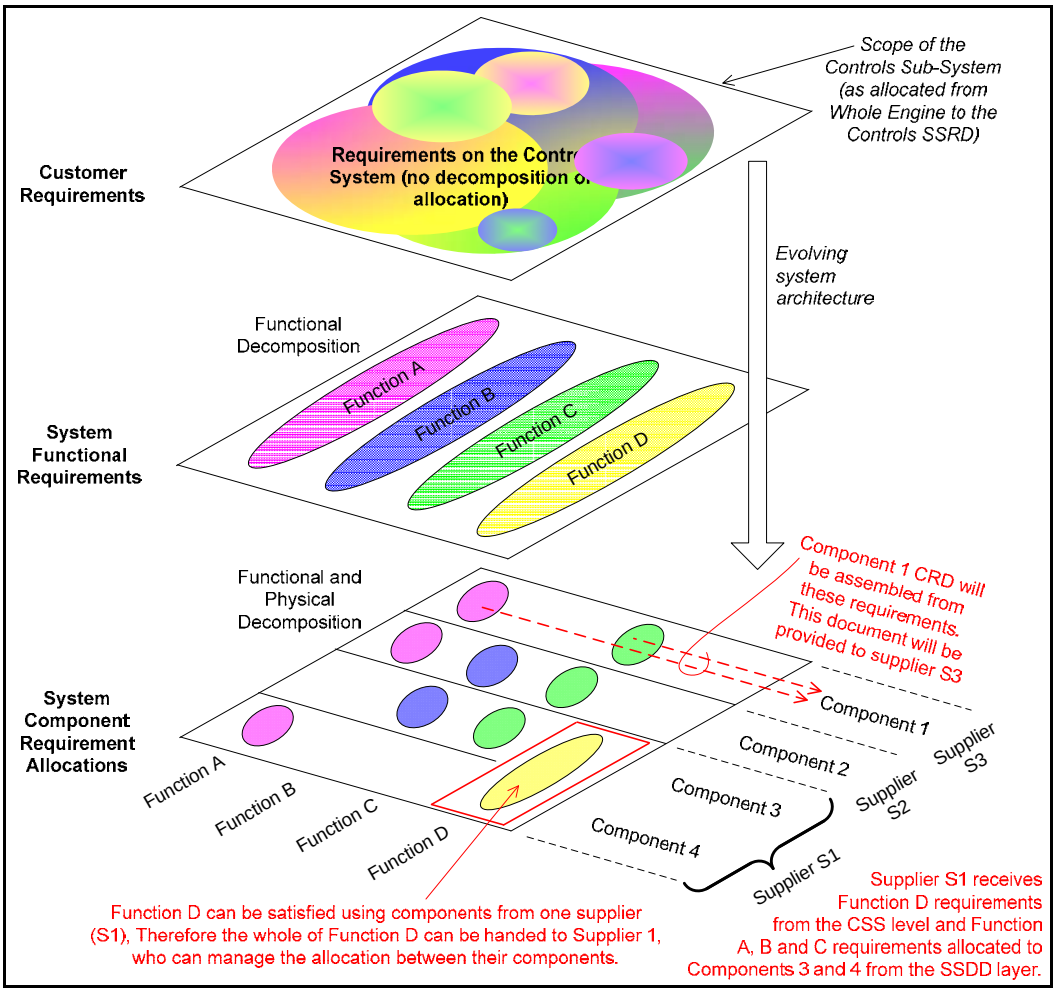

Figure 4 – Requirements Allocation According to Functional and Physical Decompositions

Requirements Documentation Structure for Globally Dispersed Teams

The preceding discussion demonstrates the variety of interpretations for the industry standards such as DO-178B and ARP-4954. Based on the preceding discussion, we propose that a requirements document structure for control system architecting in the context of globally dispersed engineering teams should at least:

- Enable compliance to certification process guidance standards (e.g. DO-178B, ARP 4754, ARP 4761) to be readily demonstrated. This must include partitioning between hardware and software to support DO-178B, rather than partitioning according to organizational boundaries.
- Support a design team decomposed into functions. This must include supporting functional partitioning within the requirements structure, developing a requirements data dictionary, and supporting black box testing.
- Allow for subsystems within the Controls Subsystem, such as the minor loop controls which can be allocated to a supplier so that the supplier manages the allocations between components they own, to meet the internal subsystem requirements allocated to that supplier.

- Support description documents and extracts of requirements documents to manage external interfaces. An example is an airframe interface control document, or environmental constraints applicable to multiple components.
- Support the different objectives of interfacing organizations. Several of these interfaces require joint ownership of requirements which flow from one to the other. Examples include other engine subsystems to Control Subsystems interfaces, Systems Engineering to Software Engineering interface, and Systems Engineering to Hardware Engineering interface.
- Account for best practices at each site, or at least provide a clear definition of how certain issues handled by a previous structure can be accommodated within the new structure.
- Flexibly adapt to hardware build standard changes of the engine, or new test results revealing a need for different control schedules or logic.

Using Stories to Communicate Systems Architecting Interactions

The ambiguity in the industry standards means that the system architecting process is not always straightforward (as evidenced by its varying Rolls-Royce embodiments). However, the variety of experience at Rolls-Royce provides a broad set of potential resources for globally managed development programs. This section uses several hypothetical scenarios to illustrate the task relationships discussed in this paper.

We use story-telling to elicit these scenarios, because stories shared in organizational settings can be useful tools for organizational and cultural change. Stories are of interest to those studying and/or problem solving within organizations because employees at all levels tell stories in one form or another for numerous reasons. They offer rich detail on a variety of topics including co-worker relationships and ways of getting the work done (Meerwarth, et al. 2006). As such, stories are a valuable method to identify the expected behavior patterns around which a particular documentation structure will be used. Our intent is to test role boundaries and review scope of responsibilities and show some organizational control and feedback loops. All of the cases are fictional. The case studies are grouped into Preliminary Design and Detailed Design. To illustrate the utility of these scenarios, we propose responses to the first few, while leaving the remaining responses for the reader to consider based on their context.

Stories from Preliminary Design

Scenario for Hardware Functions: The systems design team is busy working on trade studies and hasn't had time to work on the hardware allocations in the System Definition Document. To support early supplier engagement, the hardware teams have started working on component specifications ahead of released system requirements.

Question: Should the hardware engineers work ahead of the system requirements, and potentially create the missing system requirements?

Proposed response: As described in Figure 3, the system requirements could be viewed as an agreement layer. In the case where many hardware items can be developed using previously available designs, it does make sense for the hardware teams to populate the appropriate system requirements in advance of full top down definition.

However, this creates risk, which must be mitigated for requirements which depend on ill-defined interfaces. Provided the systems and hardware teams work closely together to manage this risk, this is an acceptable approach.

Scenario for Software Functions: The systems design team is busy working on trade studies and hasn't had time to work on the software allocations in the system requirements. To scope the software development effort, the software team has started working on the software requirements specification ahead of allocations in the system requirements.

Question: Should the software engineers work ahead of the system requirements, and potentially create the missing system requirements?

Proposed response: Software has more systems level interactive complexity than hardware, due to the fact that many internal software interactions have significant system observable effect. Therefore, more care should be taken over software advance working than hardware. However, reviewing existing software for reuse potential is possible based on high level functional requirements in the system level specifications. At the very least, a functional architecture diagram is needed to help scope the software effort ahead of the detailed system requirements.

Stories from Detailed Design

Scenario for Software Design: Late in a development program, available memory becomes critically low due to inefficient software implementation of some table lookups in the thrust management logic. Due to an old controller design, in which large lookup tables are only accessible from a given page at a time, an implementation decision was made to duplicate some tables in different pages to avoid having to switch pages every time a different lookup was needed. Although this avoided the coding overhead of page switching, it led to inefficient memory utilization.

To address the problem, a systems engineer reviewed the thrust management tables with the Performance group to see if there were any closely matched tables which could be combined or simplified with an acceptable loss of thrust scheduling accuracy. The Performance group would not accept any loss of scheduling accuracy. The systems engineer delved further into the problem, including the code implementation, and determined a possible code solution involving redefining an aircraft application index which had been used as a page select integer.

Question: Should the systems engineer have defined the software solution?

Proposed Response: When doing a design investigation, it is not necessarily important who does the initial work. The systems and software teams should have similar skills, but their responsibilities make them accountable for different tasks. Software engineers may be well suited to analyzing systems problems as part of the analysis of a software test anomaly for example, especially when the analysis reveals the software meets all the software requirements.

In this example case, if the system engineer found a possible software solution, this would need to be reviewed by the software team. If valid, the solution should be accepted as a way forward. One reason why solutions are often found by other teams is because constraints which one team assumes are unmovable may be flexible because they are under the control of the other organization. In this case, the software team may not have known that the aircraft application index could be redefined. This emphasizes the need for teams to develop skills which cross the boundaries of other teams. It also requires teams to be tolerant to recommendations coming from other teams which may appear to overlap their scope.

Scenario for Software Design: A system requirement states that a cockpit displayed parameter must be transmitted to the aircraft on ARINC, and filtered to reduce noise on the display. No time constant is defined in the system requirements, nor is any data available from

which to derive the time constant to be determined (e.g. the update rate of the cockpit gauge, any aircraft side filtering etc).

The software team implements a first order lag filter on the affected signal and assumes a time constant of 100 ms. In aircraft integration tests, it is observed that the display is too noisy, and the Airframer wants it reduced. The systems team analyses logged data to come up with a new time constant, which is provided to Software in a change request with no requirements changes at the system level.

Question: Should data definitions be provided between the Systems and Software teams without any requirements? How is this data managed?

Scenario for Hardware / Software Interaction: A system requirement exists to read the shaft speed signal above 5%. However, in rig tests, it is found that although the shaft speed is typically available by 5%, it remains noisy (intermittent dropouts to zero) up to 7%. This leads to some intermittent cross check faults due to the software trying to validate the signal from 5% and above.

The software team observes that the first software threshold which uses shaft is at 15%, and therefore implement a design change to only validate the speed signal above 10%.

Question: Should this design change have been made?

Scenario for Hardware / Software Interaction: Relay drive current levels are allocated to the controller hardware in the system requirements, based on an assumption about the hardware / software split. However, the controller supplier decides to use an FPGA design with OS software specifying the relay current level. The supplier OS software traces the current values to the controller hardware specification. This creates problems for showing test coverage and compliance to DO-178B. Testing the software requirements specification will not cover this code.

Question: Should the OS software trace to the controller hardware specification?

Scenario for Control Laws: Two schedules are defined for lightoff – one for starter assist and one for windmill starts. The data is specified with equivalent numbers in the first version of the performance requirements for engine control. These requirements include the logic diagrams, based on the Simulink model. A previous engine application had the same logic, but only one schedule. Because the new software specified the same data for each schedule, the old software was reused.

Question: Is it acceptable to reuse the previous software which only had one lightoff schedule implemented in software?

Conclusion

Establishing processes in a globally distributed organization requires Rolls-Royce controls engineers to share a common understanding of roles for systems, software, verification, and hardware engineers. They also must understand how the information shared among these roles is captured in a requirements hierarchy during a product development project. Our review of projects at Rolls-Royce programs demonstrates the variety of interpretations for the industry standards such as DO-178B and ARP-4754. This variety means that the system architecting process is not always straightforward (as evidenced by its varying Rolls-Royce embodiments).

The ISO standard emphasizes the need to agree the boundary and relationship between the systems life cycle and the software life cycle process, rather than prescribing that boundary in the standard. The variety of interpretations at Rolls-Royce shows architecting processes that have evolved to address the uncertainty associated with design risk (site A), and the compliance

needs associated with certification risk (site B). This suggests a broad set of potential resources for globally managed development programs. Consistent with the agreement process in the ISO standard, we suggest that the control system requirements structure allow two things. First, it should allow the system architecture definition to emerge. Second the component level requirements are developed by the appropriate component level process life-cycles in close cooperation with the systems architecture process.

We use the discussion of scenarios to identify various ways this emergent definition can occur. Our goal is not to prescribe a specific set of interactions. Rather it is to guide discussion on the interaction required for an effective team to address the challenges in these scenarios and others like them. Indeed, using scenarios to clarify assumptions about behaviors and responsibilities is one way to provide input from the political and cultural elements of organizational change to the strategic design lens that frequently governs organizational alignment efforts.

References

Ancona, D., Kochan, T., Scully, M., Van Maanen, J. and Westney, D. E. 1999. *Managing for the future: Organizational behavior and processes*. South-Western College Publishing, Boston.

Crawley, E., *System Architecture Lecture 14 (System Design and Management Program, MIT Engineering Systems Division)*. Massachussets Institute of Technology, 2004.

ISO. 2002. *ISO/IEC 15288:2002, Systems engineering — System life cycle processes*. ISO,

ISO. 2004. *ISO/IEC 12207, Amendment 2:2004, Information technology — Software life cycle processes*. ISO,

Leveson, N. G. 2000. *Intent Specifications: An Approach to Building Human-Centered Specifications*. IEEE Transactions on Software Engineering 26(1), Pp 15-35.

Leveson, N. G. and Weiss, K. A. 2004. *Making Embedded Software Reuse Practical and Safe*. ACM SIGSOFT Software Engineering Notes Archive 29(6 (November)), Pp 171-178.

Meerwarth, T., Briody, E., Trotter, R. and S., C. 2006. "Working with Stories: Tools for Organizational Change", Ethnographic Praxis in Industry Conference Proceedings.

Navarro, I., Leveson, N. G. and Lundqvist, K., *Reducing the Effects of Requirements Changes through System Design*. Massachusetts Institute of Technology, Software Engineering Research Laboratory Technical Report, 2001.

RTCA. 1992. *RTCA/DO-178B, Software Considerations in Airborne Systems and Equipment Certification.*

SAE. 1996. *SAE ARP 4754, Certification Considerations for Highly-Integrated or Complex Aircraft Systems*. SAE,

Author Information

Malvern J. Atherton holds a B. S. degree in Electrical and Mechanical Engineering from the University of Edinburgh, a Postgraduate Diploma in Control and Information Technology at the University of Manchester Institute of Science and Technology, and an M. S. degree in System Design and Management at the Massachusetts Institute of Technology. He has worked in the engine controls field at Rolls-Royce Aircraft Engines facilities in Derby, England, and Indianapolis, Indiana since 1990. His work at Rolls-Royce has included engine control laws, fault detection and accommodation designs, and systems engineering management for several projects. Mr. Atherton has led systems engineering design teams on several aero engine types including turbofans, turboprops and turboshafts.

Shawn T. Collins works in the Control Systems Engineering department at Rolls-Royce in Indianapolis. His professional work focuses on applying anthropological insight to engineering product development. His interests are in ethnographic methods, cognitive anthropology, product development methods, and team performance. Dr. Collins holds a B. S. degree in Mechanical Engineering from Purdue University and an M. S. degree in Mechanical Engineering from Renssalaer Polytechnic Institute. He holds M. A. and Ph.D. degrees in Anthropology in from the University of Connecticut. He is a member of INCOSE and the Society for Applied Anthropology.

Stephen A. Fisher works in the Control Systems Engineering department at Rolls-Royce in Indianapolis. He has been developing control systems for turboprop, turbofan and turboshaft engines for 26 years. He currently leads a team that is developing gas turbine engine controls for a range of future products. He is a member of the SAE Aircraft and System Development and Safety Assessment Committee where he serves as a section captain for the revision of ARP4754a, *Development of Civil Aircraft and Systems*. He holds a B. S. degree in Mechanical Engineering from Purdue University. He is a member of INCOSE.

Ralph W. Hill works in the Control Systems Engineering department at Aero Engine Controls in Derby, where he is the Chief Design Engineer for the Product Families and Preliminary Design group. In this role, he holds technical responsibility for the generation of processes and assets in support of a product line for Rolls-Royce aero-engine control systems. Mr. Hill joined Rolls-Royce in 1986. He has been involved, in different roles, with control system development on a number of aero-gas turbines, including the RTM322 and European Union technology demonstrator programmes. Mr. Hill contributed to this paper while working as the Chief Design Engineer for the Control Systems Engineering department at Rolls-Royce.