2015 NDIA GROUND VEHICLE SYSTEMS ENGINEERING AND TECHNOLOGY SYMPOSIUM
SYSTEMS ENGINEERING (SE) TECHNICAL SESSION
AUGUST 4-6, 2015 - NOVI, MICHIGAN

THE SYSTEMS ENGINEERING THREAD: FROM CDD TO TRANSITION

Gregor A. Ratajczak
VRS Project Manager
Vehicle Electronics and Architecture
TARDEC
Warren, MI

Keith MacFadyen
VRS Chief Engineer
DCS
Warren, MI

ABSTRACT

This presentation shows the process a team should use to initiate a design project based on the needs of the customer. The VRS project supports the future integration and development needs of four combat platforms (Abrams, AMPV, Bradley, and Stryker) and TARDEC's PM CVP. For this presentation, and to simplify the explanation, the TRADOC developed capability for Silent Watch is used to demonstrate the processes of analyzing Capability Description Documents (CDD), creating and deriving good requirements, allocating them to specific functions and activities, describing those activities to the lowest level, designing, building, and eventually testing.

INTRODUCTION

According to the International Council on Systems Engineering (INCOSE), "Systems Engineering is an interdisciplinary approach and means to enable the realization of successful systems. It focuses on defining customer needs and required functionality early in the development cycle, documenting requirements, then proceeding with design synthesis and system validation while considering the complete problem". "Systems Engineering integrates all the disciplines and specialty groups into a team effort forming a structured development process that proceeds from concept to production to operation." This definition, describing a structured and disciplined process, is a proactive approach clearly intended to begin as soon as possible. However, as the author has experienced both in industry and in the Government lifecycle, the earlier the "product" is in its conception the less likely structured processes will be used in its development.

As quality tools and methodologies develop, prove themselves through piloting, and establish themselves in industry and Government processes (often in that order), they find themselves beginning near the end of those processes. Or more simply, they start too late, being initially deployed in efforts where there might have been significant benefit in employing them earlier and more proactively. As a result, time and resources are often wasted to do rework or eliminate a defect through the identification of root cause. This implies problems exist, which further implies that enough work has been done to have problems. Reactive actions late in programs have less positive impact on cost, schedule, and performance than proactive prevention actions. In fact, proactive action often provides opportunity to create the most optimized performance at the overall best cost (when compared with the added cost of later failure) and in the least amount of time (when compared with the time wasted to do rework).

TARDEC, and in particular its Research & Technology Integration (RTI) group, has recently embarked on a Systems Engineering (SE) approach to ensure that its Combat Vehicle Prototyping (CVP) project culminates in the realization of a successful system. For this project multiple subsystems are being developed simultaneously by different entities within RTI, each of which contribute to the system as a whole. VEA is responsible for producing the data network and power management architecture for this vehicle, and along with the other subsystems will transition a product to meet the overall CVP project delivery date. VEA is trying to "build the bench" in regard to proactively using SE early in the development cycle. This presentation focusses on a real example of VEA using SE early, analyzing the needs of the customer, creating/deriving the requirements, synthesizing the design, and eventually testing and transitioning the product. VEA is supporting its SE application with System Modeling Language (SysML) and many of the graphical representations in the presentation are

excerpts from the SysML modeling tool. The presentation shows a complete thread of the process a team should use to initiate and complete a design project based on the needs of one or more customers.

The VEA Research SIL project is the means for VEA to deliver the next generation power and network architecture and design to meet the needs of CVP and the other combat platforms whose needs were part of the original problem set.

THE SYSTEMS ENGINEERING V MODEL

The overall approach used throughout the project is an adaptation of the SE "V". The V model has proven itself successful as a guide to creating a product intended to satisfy a defined set of needs. However, it is presented as a generic, high level process adaptable to many project types. For each project, team members should discuss, interpret, and eventually agree about what the major steps of the V model mean specific to their project. The very generic high level steps should be interpreted and given more project specific meaning. The end goals of the project are probably key words in some of this less generic interpretation. In a very preliminary way the beginning of the project schedule begins to take shape as some of the high level generic activities take on project language and a rudimentary translation takes place as shown in figure 1.

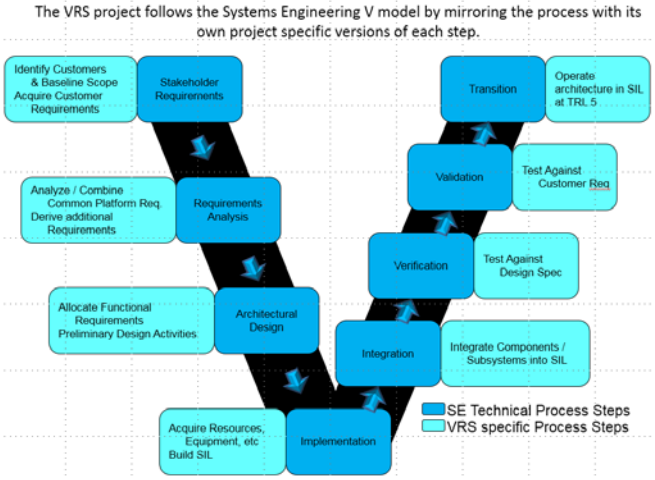

Figure 1: VRS specific interpretation of SE "V"

For the VRS project, a combined and common requirement set would be the first adaptation to the V. Requirements Analysis meant creating one set of requirements from many sets of requirements. Implementation meant acquiring the resources and building the laboratory that would house the development of the product and its later verification and validation. Integration meant integrating components and subsystems into a combat vehicle test shell. Finally Transition meant being ready to take the Test Readiness Level (TRL) from 5 to 6 by moving out of the test shell and into a vehicle demonstrator.

REQUIREMENTS

Requirements are the glue that bind the entire project together. Their intent is to inform the design and the architecture both of the needs and the constraints. A common approach to creating a requirement is as follows. Using original text from CDD, or other source document, identify language relative to a certain attribute. Transform that language into a singular, testable, shall statement. These are the attributes of well written requirements.

Some requirements may be deemed architecturally significant. The difference between architecture and design is distinct in that architecture guides design, and not the other way around. Architecture may be considered the set of rules to be followed while exercising design activities. The design should adhere to the architecture and the architecturally specific requirements while striving to satisfy the greater number of performance related requirements.

Traceability to the requirements as the product matures is critical. Although more requirements may need to be derived to satisfy a higher level source requirements, there should never be any activity or function that does not trace back to a requirement. In the end the goal is to satisfy only what was needed without scope creep or unjustified function or capability. A partial example of a well-organized requirements database appears in figure 2.

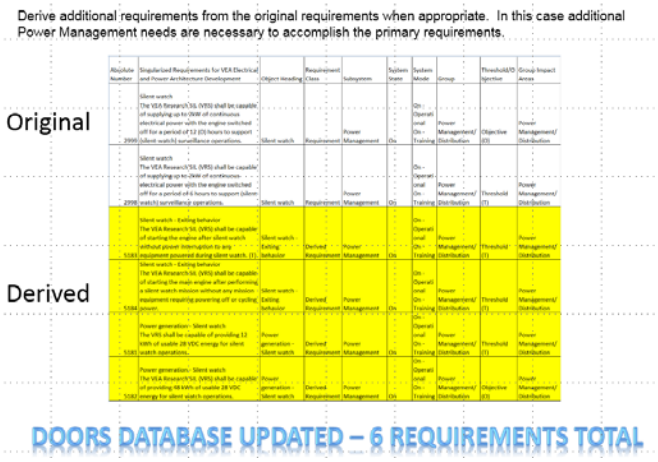

Figure 2: Source and derived requirements as part of a DOORS database.

ALLOCATION OF REQUIREMENTS TO FUNCTIONS

Requirements alone do not a vehicle (or architecture) make. Requirements are fulfilled by functions, and functions are parts of larger capabilities, which stem from WBS elements. Full traceability means that all the requirements eventually belong to at least one function. To

help establish this, a functional hierarchy may be created through the use of a WBS. After the major WBS elements are identified at level one and two, capabilities appear at level three, and functions at level four.

A Function is the associated action(s) that a particular part of a system will perform in order to provide a Capability and support satisfying the Capability's associated system requirements. Functions are decomposed and grouped by Capability.

Checks and balances: There should never be a requirement that has no assignment to a Function (the requirement would never be fulfilled) and there should never be a Function that has no associated requirements (if so then why have this function if there is no "need"?). Eventually all requirements, both source and derived, must be allocated to one or more Functions. An update to the traceability database, DOORS or other manageable tool, is appropriate after the allocation.

CREATING USE CASES

Whether done in parallel while creating the functional hierarchy and allocating requirements to functions, or before or after it, analyzing the capabilities and the requirements from a user perspective results in identifying "use cases". A use case is a graphical representation of the user's potential interaction with the system. As with many of the diagrams shown in this example, use case diagrams can cascade from one to many to show the different levels of use. Figure 3 illustrates the highest level of a use case, that which describes the broad use case "Operate Vehicle".

Similar to a WBS there are different levels of definition for use cases. When appropriate use cases should be brought down to the lowest level through decomposition. The highest level (for this ground vehicle example) is the use case called "1.0 Operate Vehicle", comprised of eight smaller (but still broad) use cases. Since we want to eventually fulfill the function "Provide Silent Watch Power", and thereby satisfy the requirements allocated to it, this function will have to have a place in this structure. A logical search through the secondary use cases under "Operate Vehicle" shows that "Provide Silent Watch Power" most likely belongs to the secondary use case "1.7 Protect Crew".

Continuing with traceability, all the secondary use cases are defined with their own diagrams which have the potential to identify even more defined use cases. Use cases should provide an eventual home for all the functions through the identification of activities (or scenarios).

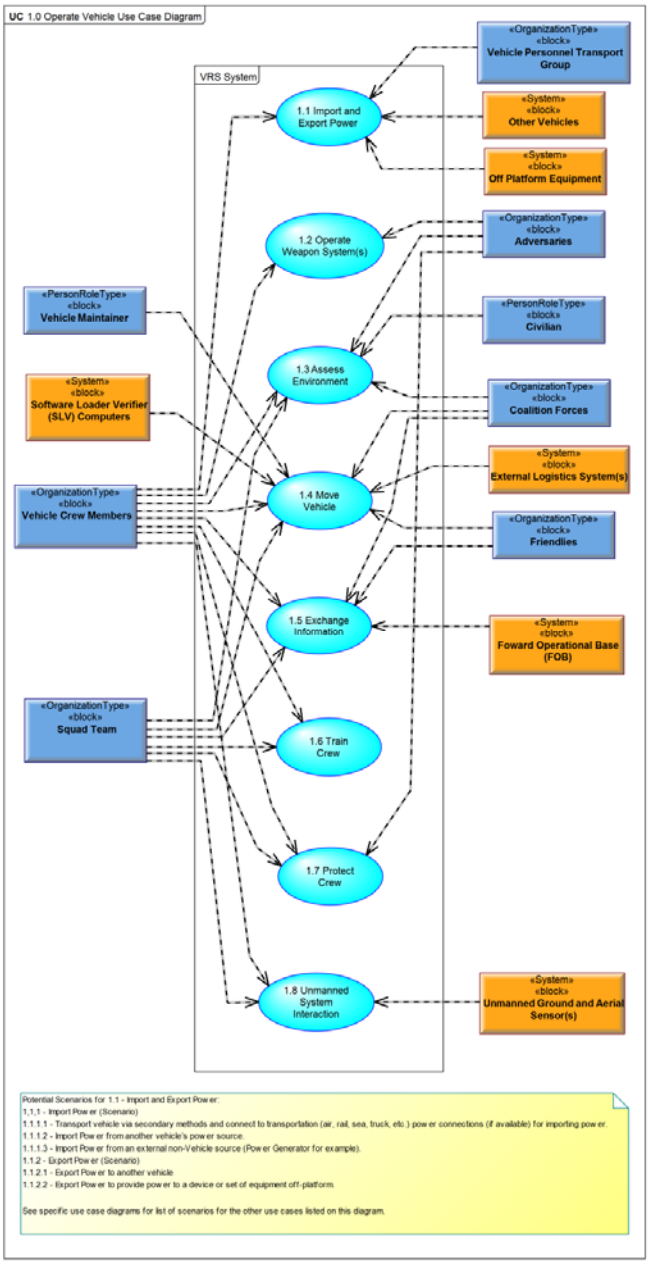

Figure 3: High level use case diagram

CREATING ACTIVITY DIAGRAMS

Eventually the lowest level use cases will result in the identification of activities (or scenarios). A logical transition from use case to activity is best identified from the operational viewpoint. In other words it is a determination of at what point it is appropriate to look into the "black box" that the user is "using" and define what is going on in there. For instance, the use case "1.7 Protect crew" is further defined by four more potential uses, one of which is "1.7.1 Avoid detection". To avoid detection, the signature of the

platform will need to be managed. Here the use case hierarchy has finally been decomposed to a level where the activities within the "black box" that manage platform signature can be defined. When use cases are broken down to the lowest level they result in the identification of many activities (also referred to as scenarios) which, from the user perspective, are the things going on in the "black box". "1.7.1.1 Manage signature of platform" is such an activity that is low level enough to be described using an activity diagram.

As with the use cases, activity diagrams start at a high level and may result in additional lower level diagrams that describe more detail. Continuing with the ever important aspect of traceability, every function must appear in at least one activity diagram.

An activity diagram uses swim lanes to show who is responsible for an action within an activity. Each lane is owned by an ACTOR (a term to describe roles played by human users, external hardware, or other subjects) or a SUBSYSTEM. Figure 4 shows the "Manage platform signature" activity diagram.

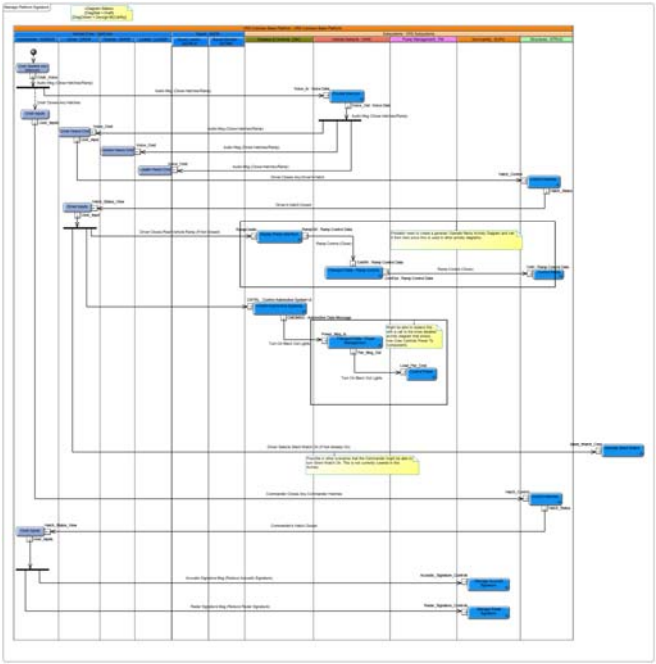

Figure 4: Activity diagram for "Manage platform signature"

Similar to the use case hierarchy, an activity diagram hierarchy most likely exists. On a subsequent, lower level activity diagram, the function identified earlier as "Provide Silent Watch Power" finally has a home.

Functions are allocated to subsystems, and those subsystems are responsible for making sure the design addresses the behavior and interfaces depicted in the diagrams and described in the requirements. As the design matures it becomes necessary to dive deeper into detail. Subsystems need to create subsystem activity diagrams to more fully explain the activities they are responsible for. In this example the Power Management subsystem would create a detailed activity diagram for Provide Silent Watch Power, an activity within their swim lane on the System level diagram. Subsystem activity diagrams look very similar to System level activity diagrams but the swim lanes now depict the components or the sub-assemblies.

Hierarchy continues to exist when appropriate. Subsystem activity diagrams describe the activity of components. When those activities can be broken down further another set of diagrams should be created. Component or sub-assembly level activity diagrams are justified when further explanation of the component activities is possible. Here swim lanes consist of hardware, firmware, or computer software considered Configuration Items (CI). If that component is hardware (HW) then it is an HWCI. If that HWCI is "smart", meaning that it has the ability to process data and act accordingly based on the use of software or code, it is considered to also have an associated Computer Software (CS) or CSCI. A single "smart" HWCI is likely to have multiple CSCIs. A CSCI may result in CSCI specific activity diagrams and the need for additional derivation of software requirements.

To summarize the evolution of activity diagrams: Functions enable capabilities and requirements through the execution of activities. Starting at the System level, the activities of subsystems are defined. Subsystem level activities are broken down into Component level activities performed by hardware and software. This level of definition is now approaching the level that can be used to implement software code.

CONFIGURATION ITEMS

HWCIs and CSCIs are described by different methods. HWCIs that do not have associated CSCIs are often described using CAD drawings, specifications, or other graphical representation methods. CSCI require further description of the logical activity with another activity diagram and it is not uncommon to derive software requirements at this time.

As stated earlier, smart HWCI will likely have multiple associated CSCI. In turn each CSCI is comprised of one or more Computer Software Components (CSC). A CSC is a grouping or aggregate of two or more Software Units (SU) or Computer Software Units (CSU). The SU is the smallest subdivision of a CSCI for the purposes of Engineering or Configuration Management. SUs are typically separately compiled pieces of code.

It is acceptable to have the same CSC appear in different CSCI and the same SU be reused in different CSC. It is also

acceptable to define CI as whatever is manageable from a deployment perspective. Configuration Management may suggest any combination of SU, CSC, SW, and HW to be a CI.

PHYSICAL DESIGN

In parallel to the creation of the activity diagrams, the physical design should take shape. Activity diagrams do not fully populate all the necessary information to describe the design, but they aid in supplying the subsystems and the actors and form the basis for discussion and evolution of the physical design. Physical designs are described by using block definition diagrams and internal block diagrams.

The BLOCK DEFINITION DIAGRAM describes the properties of each block and the relationships between them. The blocks themselves may represent subsystems or similar items that have a relationship to one another.

The INTERNAL BLOCK DIAGRAMS describe the interfaces and flows between the blocks. These interfaces are well defined in a good design and are the first description of how everything will get put together and work together. It is the first graphical representation of a tangible product.

Interface Control Documents (ICD) add further detail by describing the interface(s) between subsystems (or blocks), or the interface(s) to the system. ICD may get very detailed if appropriate by describing even lower level interfaces between the lowest physical elements.

VERIFICATION AND VALIDATION

Eventually Verification "that the product was built right", and Validation that "this is the right solution to the problem", is possible. If both are successful the product may finally transition. Testing in Verification and Validation begins with the lowest levels of logic and hardware functions, Verifying that the design fulfills system level requirements, and Validating that it meets customer needs.

The hierarchy of testing:

1. Development testing – provide data showing that logical paths and components perform as intended and that the functions and activities of the subsystems can be fulfilled. Development testing is often simple and specific with the intent that developers and designers can verify that they are "on the right path". Development testing provides confidence that the lowest level of activity is happening as intended. "Passing" and "failing" test steps provide direction in development testing.
2. Component and Subsystem testing – provide data showing that components address component level requirements and subsystems address subsystem level requirements. In both cases the intended behavior of the components and the subsystems relative to the activity diagrams is confirmed.
3. System testing – provide data showing that as a final roll up from HW, SW, components, and subsystems that in combination the system itself fulfills the intended functions and meets the original system level requirements identified at the beginning of the project.

For each level, from Component to System, Verification ("is it built right?") can take four typical forms:

1. Demonstration – visual observations are the primary means of Verification. Use when quantitative assurance is not required for Verification of the requirements.
2. Inspection – visual inspection of equipment and evaluation of drawings and other pertinent design data Verifies conformance.
3. Analysis – the use of analytical techniques or computer models to explain behavior or performance.
4. Test – an activity which provides data relative to function and operation in an environment whose conditions are completely controlled and traceable. Evaluation of the test data includes comparison against requirements. Test is done when other methods of Verification cannot deliver an acceptable level of confidence or if testing is shown to be the most cost effective method.

In regard to Validation ("is this the right solution?"):

- Validation happens (most often) at the end of development but can be used early to find issues if work products and relevant stakeholders are engaged.
- It is a process of establishing evidence that provides a high degree of assurance that a product, service, or system meets the needs of the customer.
- Can utilize methods similar to verification.

Although both may be performed this way, Validation is almost always performed by a "disinterested, independent third party". Third party Validation avoids possible bias and allows for the fairest evaluation of a product. Hence the use of the familiar acronym for Verification and Validation, I(ndependent) V&V, or IV&V.

It is possible that a product passes when verified but fails when validated. For instance if a product is built per the specifications but the specifications themselves fail to address the user's needs, the product is not validated. Verification and Validation are often (incorrectly) used interchangeably and can be difficult to differentiate from each other. It is best to always remember that the difference is Verification asks "Did we build it right? Did we build to specification?" and is measured against the requirements, while Validation asks "Did we build the right thing? Is this

the solution to the problem?" and is measured against the needs of the customer. In order to transition a product to the next TRL, or to field it, BOTH Verification and Validation must be successful.

SUMMARY

Success in design is dependent upon being able to identify what problem is to be solved through proper scoping. Understanding the needs leads to the creation of requirements and the beginning of the traceability line. By eventual allocation of requirements to functions, and those functions finding their way into activities via use cases, the lowest level activities identify configuration items which may be hardware or software, or even "smart" hardware. Low level activity descriptions aid in supplying the subsystems, components, sub-assemblies and actors (users) and form the basis for discussion and evolution of the physical design. Block definition diagrams and internal block diagrams allow for Interface Control Documents to add further detail by describing the interface(s) between subsystems (or blocks), or the interface(s) to the system. Beginning at the lowest level testing is used to both Verify and Validate the design, both against the requirements and the problem to be solved, respectively.

REFERENCES

[1] www.incose.org

[2] www.softwaretesting.com

[3] www.dau.mil