SYM-AM-18-078

PROCEEDINGS
OF THE
FIFTEENTH ANNUAL ACQUISITION RESEARCH SYMPOSIUM

THURSDAY SESSIONS
VOLUME II

Acquisition Research:
Creating Synergy for Informed Change

May 9–10, 2018

March 30, 2018

ACQUISITION RESEARCH PROGRAM
GRADUATE SCHOOL OF BUSINESS & PUBLIC POLICY
NAVAL POSTGRADUATE SCHOOL

Seven Tips to Support Rapid Product Deployment: Lessons Learned

Bruce Nagy—holds the position of Chief Architect for advanced development activities within NAWCWD. He leads forward thinking research efforts involving machine intelligence, simulation/regression techniques, software architecture, and battle engagement strategies. He specializes in proactive management using predictive metrics. Using predictive metrics, Nagy has shortened the development schedule for advanced R&D DoD and NATO-based projects. He has been noted to recover a Navy satellite program from excessive schedule delays. While as a Naval Officer, he specialized in troubleshooting critical path issues for high visibility programs. Nagy has degrees in mathematics, biology, and electrical engineering from The Citadel and Naval Postgraduate School. [bruce.nagy@navy.mil]

Abstract

Navy leadership has tasked its workforce to respond to an urgent need of identifying good ideas that fulfill immediate warfighter gaps in offensive and defensive capability. This paper describes how following seven tips gave a 3.5-person team the ability to develop a working prototype demonstrating the framework of a cutting-edge, counter Unmanned Aerial System (cUAS) technology, supporting a complete kill chain solution. When good ideas are accepted by Navy leadership for feasibility investigation, budgets are tight, teams are small, and expectations are high. This necessitates an environment that introduces a need for rapid product prototyping of a working system/framework that will inspire additional monies to support rapid deployment. Seven tips describe detailed lessons learned involving project management, resource management, system engineering, and architectural analysis, including the use of Open System Architecture (OSA). Specific best practice techniques applied to the rapidly prototyped cUAS technology example, including generic discussions of Program of Record technologies, are used as case studies emphasizing the benefits of each tip described. By considering the seven tips, the workforce is given guidance, examples, and food-for-thought as to how to meet Navy leadership's urgent need for rapid deployment of cutting-edge technology.

Background

"Months not years" is now the leadership's call for urgency in getting products out to the fleet in a shorter period of time. This call has become a mantra regarding the need for new ideas taking advantage of existing technology in creative ways.

"Easier said than done" is one phrase heard by frustrated acquisition professionals. For example, a reader's response as annotated in a blog of the AirTALKs' recipe (recited in the previous introduction) was, "what bureaucratic barriers were eliminated?"

The current "recipe for success" described by AIRTalks on August 15, 2017, to deliver products more rapidly to the fleet is as follows:

- Leadership set a *clear* and *urgent goal*
- Focused on *schedule* and *outcomes*; *tailored in* only what was critical
- *Empowered* the team to manage risk and make decisions; and
- *Eliminated* bureaucratic *barriers* to speed

Another initiative supported by the DoD's MD5 National Security Technology Accelerator focuses on teaching employees how to deliver a well-crafted elevator pitch and interview stakeholders in support of future adaptation.

The above points are valuable to heed and the related training is important, but the recipe misses a key ingredient: educating the workforce on how to deliver products more rapidly to the fleet. This paper does not attempt to reinvent the acquisition cycle, but to more effectively work with it to provide an approach to support rapid prototyping by using more effective Open Systems Architectures (OSAs) and standards supported by industry best practices associated with project management, system engineering, architectural analysis, and resource management. The lessons learned described in this document also include creating a constructive workforce environment—using a common-sense best practice, as will be described.

The need to begin the birth of a program by rapid prototyping that uses more effective OSAs and proper application of best practices ensures that the system of systems architecture will support a rapid product deployment. This document describes examples and related approaches associated with project management, system engineering, architectural analysis, and resource management to ensure the proper selection of OSA and application of best practices. Using the tips provided, a team can have a better chance to take a “good idea” and get it funded, proving out concepts through the use of rapid prototyping. Rapid prototyping, in the counter Unmanned Aerial System (cUAS) case provided as the primary example within this document, means delivering a working system of systems that demonstrates the concept and validates the selection of OSA that enables cost reduction, productivity improvement, and product reliability.

In today’s urgent need to maintain technical superiority within a theater of operations, rapidly deployable good ideas, using the proper OSAs through the use of best practices, are more likely to gain attention and get funded. Most ideas are supported with limited funding, therefore rapid prototyping, minimizing cost but emphasizing capability, becomes a necessity.

The Necessity to Rapidly Prototype a Good Idea

It is logical to any acquisition professional to be concerned about rapid deployment and its potentially adverse effects on the quality and reliability of product. Will the product suffer if pushed into the hands of the warfighter too soon? That is certainly one of the reasons why the acquisition cycle is so structured and rigorous. What emphasizes this concern of a poorly developed product from skipping steps is that warfighters need highly reliable, quality products because their lives might be in jeopardy and they are dependent on the technology they are using. In most commercial products, this level of rigor or dependability isn’t required. For example, if a smartphone breaks, the user is without a phone until he or she takes it to the store for a replacement. If a warfighter’s communication device stops working within a battle engagement, his lifeline for fire support may not be available, creating a potentially life-threatening situation.

The main challenge is that the acquisition turnaround time for a new technology consists of many approval cycles and gates, sometimes consuming a decade of reviews and meetings before the proposed product is seen by the fleet. Within this same period of time, the commercial market may have produced hundreds of components that might have affected the performance, cost, and quality of the product under development.

The obvious answer is to focus on OSA and use of industry best practices that make going through the acquisition cycle faster, but still allow for highly reliable, quality results. A popular analogy is to be able to create a system of systems architecture in a similar manner as putting together Legos, where the OSA associated with best practice interface standards represent the nubs on each Lego block. Therefore, in the simplest of explanations, this paper describes lessons learned in terms of using best practices based on specific, proven tips and sound engineering practices supporting the proper use of OSA.

The goal of the paper is to offer seven tips for consideration in helping acquisition professionals more easily create an effective Lego-based system of systems prototype products.

The tips are centered on rapid prototyping to ensure that the selected OSA through use of best practices is tested to be practical, economically feasible, and reliable in terms of meeting goals. Examples in this document will be provided in which the wrong OSA selection created increased acquisition issues. Specific solutions to this dilemma will be described in terms of best practice approaches.

If rapid prototyping is done properly using the “right” OSA and following effective best practices, then the benefit will more likely be a rapidly developed prototype, using minimal cost over a short period of time. With regard to the cUAS example used to support tip development, the technology was developed within two months using a team of 3.5 developers.

An additional benefit is that as new commercial technologies are created during the acquisition cycle, the prototyped architecture, following the tips offered in this document, will have proven to allow for a plug-and-play development environment. This will offer the product, under the acquisition cycle, greater opportunities to keep in pace with technology advancements, naturally reducing the acquisition cycle, increasing the reliability of the product, and significantly decreasing overall program cost.

Seven tips will be discussed in the form of lessons learned to share how to practically use OSA and industry best practices to answer Navy leadership’s “months not years” call for urgency.

Seven Tips

Tip 1: “A Picture Is Worth a Thousand Words!”—Use a Storyboard to Clarify the Problem and Solution

This tip supports the first step of gaining clarity regarding the problem and solution. To gain this clarity, four questions need to be answered:

1. What specific problem does the Navy need to solve?
2. What assumptions are being made about the problem domain, and do those assumptions still support Navy needs?
3. Is someone already solving this problem using the same assumptions? If so, was this group contacted and solutions compared?
4. With regard to the proposed solution, is a complete kill chain scenario described?

This first tip has to do with how the problem can be qualified to be a viable candidate for funding. It avoids spending time solving a problem that lacks interest from potential stakeholders willing to provide funds for the proposed solution. As stated, a cUAS project is offered as an example to illustrate the value this tip offers in clearly defining a problem and describing a complete solution.

A short background regarding cUAS: An effective cUAS solution is becoming an important goal for all armed services, especially because of recent events in the news. From a Navy perspective, the issue is potential threats to naval facilities from small, hard-to-spot drones.

Because of this Navy focus, the problem defined was limited to an attack from sea. The imagined attack involved a small boat, a small Unmanned Aerial Vehicle (UAV), and a sinister intent. The problem definition naturally eliminated many solutions developed by the Army or other branches. The Navy needs to be worried about sea attacks, more than other government agencies. In this project, the assumptions were that the UAV could be launched miles away from an ocean launch site. Depending on the distance, land-based sensors may have difficulty identifying small objects from long distances away. The small object would be a group 1 UAV with autonomous capability, designed as a fixed wing drone. The drone's performance would be typical of any fixed wing autonomous vehicle purchased on some popular website that could fly long distances without operator intervention.

Again, to adequately address the third question regarding this tip, it was important to investigate if there was already a solution to identify a group 1 UAV from far distances, launched from the sea platform, and then eliminate this threat, if needed. It seemed that a solution to support this water-based attack using the approach described would add to an arsenal of Navy capabilities. Although, the Army supported a detailed cUAS solution and because of this, a Technical Interchange Meeting was held to ensure the Navy cUAS solution related to its unique problem was not already solved.

Once it was determined that the solution was unique, where no other groups were providing similar solutions, the next step was defining a complete kill chain or mission scenario. The goal was not to just define a piece of the solution, but to support the entire kill chain scenario. The solution ranged from how the UAV was identified when approaching to its elimination.

To capture these ideas, Department of Defense Architecture Framework (DoDAF) Operational View-1 (OV-1) view was used in a unique way. Two OV-1s were combined to make up a storyboard. OV-1 views are commonly used. A two-frame storyboard, as shown in Figure 1, is a creative use of OV-1s. The point is that DoDAF views allow for creative license. In Figure 1, the cUAS example is described from supporting identification using video to a kamikaze defense solution.

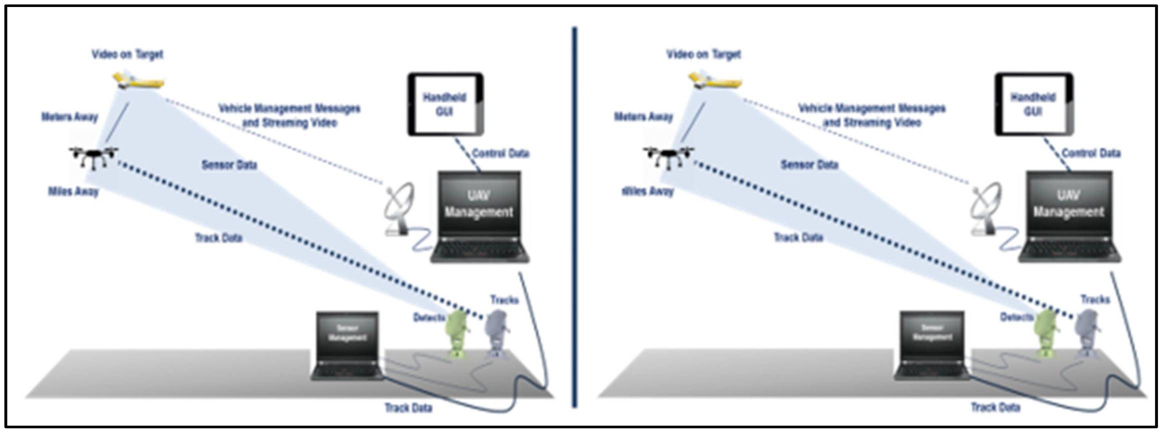


Figure 1. A Simplified Version of Two Related DoDAF OV-1s Used as a Two-Frame Storyboard

Figure 1 represents assumptions regarding both the problem and solution, which allowed stakeholders and the development team to understand what type of subsystem elements might be needed. "A picture is worth a thousand words," but it is still recommended that words provide the details behind each OV-1 views. With one sheet of paper, the views along with dialogue can now communicate both solution and assumptions to verify if both still support Navy needs. The goal is to use the OV-1 to ensure that there is a clear definition of the problem and a complete solution. “Complete” refers to including all parts of the kill chain scenario, from identification to elimination, as described in Figure 1.

Therefore, in following the approach described in this tip using OV-1s, all four questions were more easily answered and explained to others.

Tip 2: Apply Assembly Line Thinking to Make “Months Not Years” a Possibility

This tip illustrates how DoDAF views can greatly enhance the ability to meet the “months not years” goal without hindering productivity.

For review, an assembly line consists of a series of already prepared parts that are integrated over a set period of time, normally defined by the speed of the assembly line process. Using this analogy, consider platforms or subsystem elements to be the already prepared parts. The schedule defines the assembly line time period. Each of these parts are welded together via software interfaces or hardware constructs, like Ethernet cable. Note: Even if the assembly line’s purpose is to only develop one item, the assembly line process is still valid.

In the project mentioned above involving cUAS, the assembly line subsystem elements/platforms were as follows:

- Kinetic Integrated Low-Cost Software Integrated Tactical Combat Handheld (KILSWITCH), which has been used by the Marine Corps to support situational awareness and other combat-related goals.
- Navy Unmanned Common Control System (CCS) Science and Technology (S&T) version using the Office of Naval Research’s (ONR’s) Topside. Topside is a multi-dimensional ground control station. The combination of using CCS and Topside has been successfully demonstrated over the last three years with a variety of demonstrations, including Large Displacement Unmanned Undersea Vehicle (LDUUV), Autonomous Aerial Cargo/Utility System (AACUS), and Common Mission Command Center (CMCC) with the K-MAX Unmanned Aerial System (UAS).
- Maneuver Aviation Fires Integrated Application (MAFIA), Joint Multi-Platform Advanced Combat (JMAC), and MAFIA Association System (MAS), which is currently fielded as cUAS technology.
- Maneuver Aviation Fires Integrated Application (MFOCS), which is currently fielded.
- Standardized Payload Management Systems (SPMS), which has been demonstrated to manage a variety of weapons and payload systems on Navy UAVs and is currently being enhanced using an Unmanned Aerial Vehicle Control Segment (UCS) service interface.

The main interface, a common bridge, to connect these elements together was determined to be UCS, a Navy supported standard. For the subsystem elements selected, the bridges/interfaces to the UCS standard would therefore be

- Cursor on Target (COT) supporting KILSWITCH communication to UCS
- Tactical Counter-Unmanned Technologies (TCUT) supporting MFOCS communication to UCS

This common bridge ensured interoperability while keeping communication under a single bridge standard. The need for a common standard connecting all system elements is described in Tip 3.

To implement an assembly line paradigm, a series of sequence diagrams were chosen. The first diagram type started with operational sequence diagrams (DoDAF OV-6c; see Figure 2) and then moved to system sequence diagrams supporting a system view (SV). Figure 3 describes a DoDAF SV-10c which translated platforms in OV-6c to sub-elements within those platforms. Figure 2 describes the use of various types of radars to support various needs in identifying small targets. The Joint Integrated Fire Control System (JIFCS) was the technology developed within months using the tips described in this paper. Figure 2 describes how JIFCS needed to interface with sensor data, handheld devices, and a group 1 UAV to implement the desired cUAS solution. The goal of both Figures 2 and 3 is not to describe the JIFCS technology, but to show how assembly line thinking was applied to rapidly develop a new product.

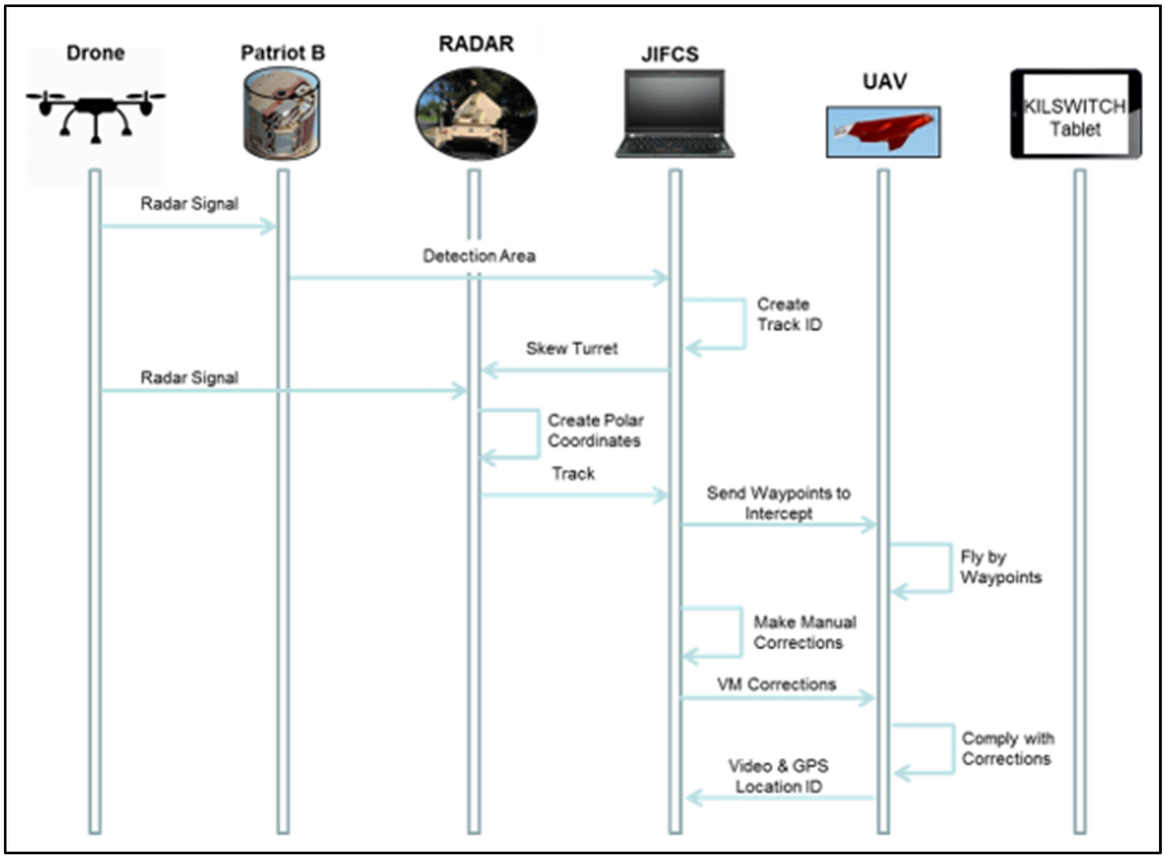


Figure 2. OV-6c View Regarding One Part of the cUAS Solution

The final suggestion related to Tip 2 is make sure the pieces to be assembled are mature enough to be used within the project's schedule. This pitfall was realized in a rocket development project, where one of the main elements relied on a rocket engine that was still in the experimental phase. The rocket engine needed several years to mature before it could be considered to be realistically used in a combat environment. In this case, because this element made the assembly line process take years, and not months, the project was rejected. Redesigns were eventually investigated, but the project's focus was lost. This type of rocket development project became a very low priority for funding and eventually was forgotten. Be careful: If an element takes several years to mature, it obviously should not be used as part of the assembly line process when attempting to support leadership's "months not years" call for urgency.

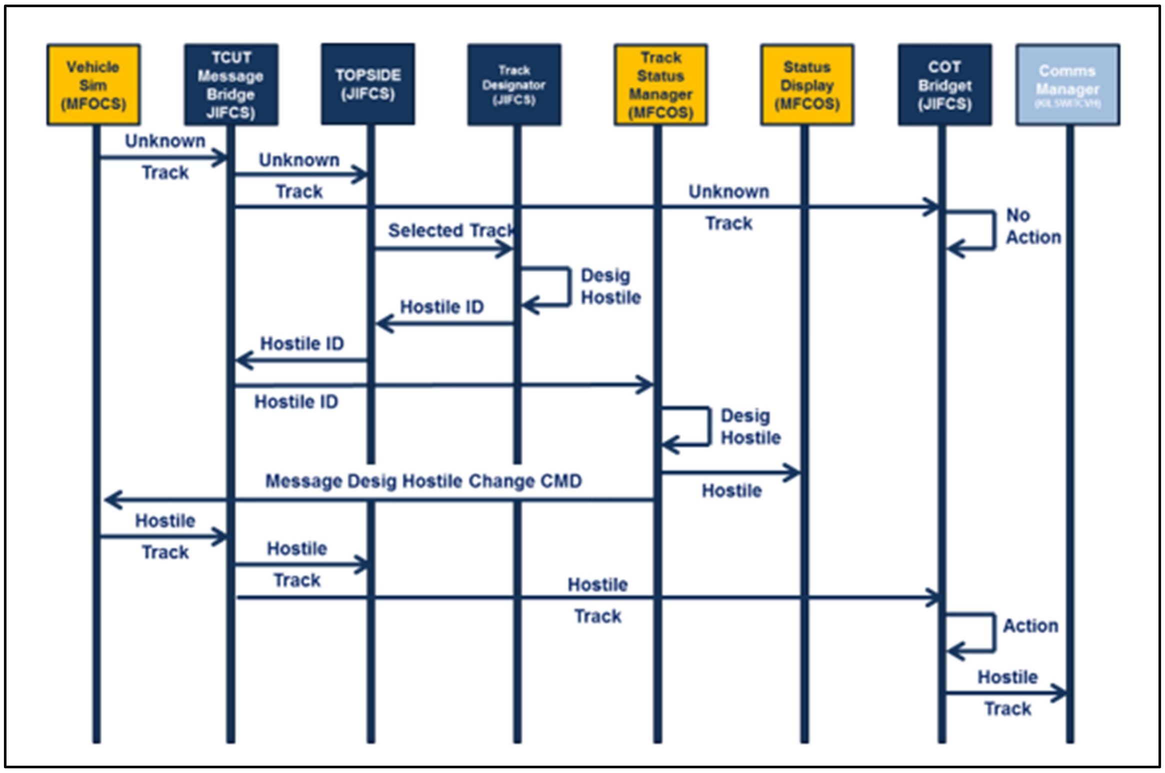


Figure 3. SV-10c View Regarding One Part of the cUAS Solution

Tip 3: "To Be or Not to Be?"—Analyze the Layers of Interfaces to Determine What Should or Should Not Be Used

Tip 3 provides a "rule of thumb" on how to determine whether it makes architectural engineering sense to proceed with regard to interfaces between subsystem elements.

In the cUAS example, there were subsystem elements that required the use of three to five protocols to exchange needed data. This caused an over-complication of the architecture and implementation, resulting in integration and performance issues. This came about because each of the original subsystems were developed using different interface standards. To create even more complications, some of the elements were developed in a Linux Operating System (OS) and others in Windows OS, causing additional enterprise gluing issues.

Although the protocol associated with each subsystem involved well-known and popular standards, the need to use many standards to support straightforward communication caused installation and performance issues.

As a tip, if a straightforward bridge cannot be used between standards of different subsystem elements, then consider reevaluating the subsystem elements. If there are no other choices, then consider new development instead of forcing a square peg to fit a round hole.

In the cUAS example, the UCS standard was determined to be able to bridge all subsystem elements. The other elements were assessed as to whether straightforward bridges could be produced, and the assessment came out positive. If it didn't, then a different architecture would have been investigated.

Figure 4 is an example of multiple protocols used through a thread of information in the cUAS project. The upper thread represents four translator services that would potentially be needed for communication between subsystem elements and a system that shouldn't be put together. The lower thread in Figure 4 represents an ability to architecturally reduce subsystem communication to one translator service, creating a good system. The architecture introduced an Air Force standard named Unmanned Command and Control Initiative (UCI); however, the ongoing challenge was to ensure UCI and UCS efficiently talked to each other.

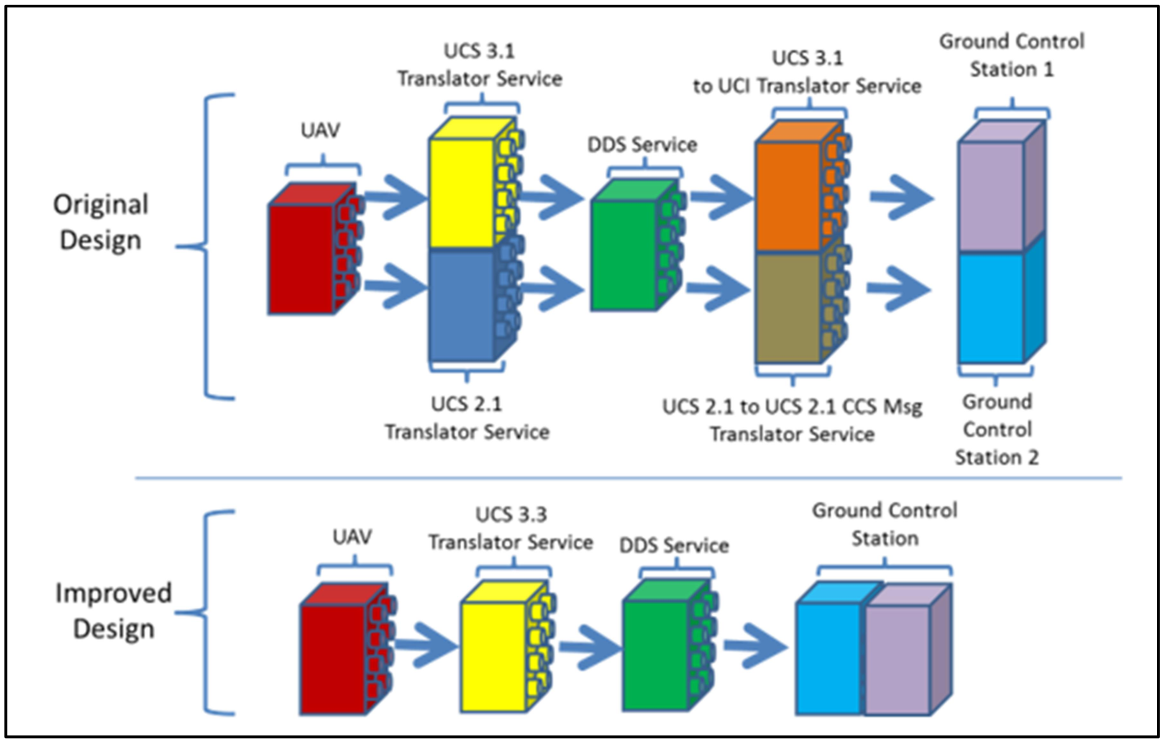

Original Design
UAV
UCS 3.1 Translator Service
UCS 2.1 Translator Service
DDS Service
UCS 3.1 to UCI Translator Service
UCS 2.1 to UCS 2.1 CCS Msg Translator Service
Ground Control Station 1
Ground Control Station 2
Improved Design
UAV
UCS 3.3 Translator Service
DDS Service
Ground Control Station

Figure 4. Translator Service Designs Used for Communication Between Subsystem Elements

Tip 4: Focus More on Integration, Less on New Development, to Create a "Months Not Years" Project Plan

Once it has been agreed to proceed with the project, Tip 4 provides suggestions on how to quickly create the first demo within the "months not years" timeframe. It provides examples of system engineering techniques that support rapid development.

In the case of the cUAS project, using Tip 4's technique showed that the demonstration could be done within four two-week sprints, which included architectural analysis, implementation, and test.

If the first three tips are already completed, then this tip will naturally follow and be easier to achieve. The Tip 4 technique suggests using the time it takes to do the integration/assembly of each primary element, including data usage, to determine the shortest timeline to release a demo. If the "right" primary elements aren't correctly identified from your assembly line, it's difficult to successfully use this tip. Primary elements are identified by understanding how the key platforms are used. Sometimes the best way to identify primary elements is to identify non-primary elements. Non-primary elements are subsystems that use primary elements as their hosted platforms.

In the cUAS example, the primary elements were the MFOCS emulator, JIFCS emulator, and KILSWITCH. Using the technique just described, the timeline for connecting the primary elements first involved the development of two bridges/interfaces:

1. From the COT message standard to the UCS message standard between KILSWITCH and JFICS
2. From the TCUT message standard to the UCS message standard between MFOCS and JIFCS

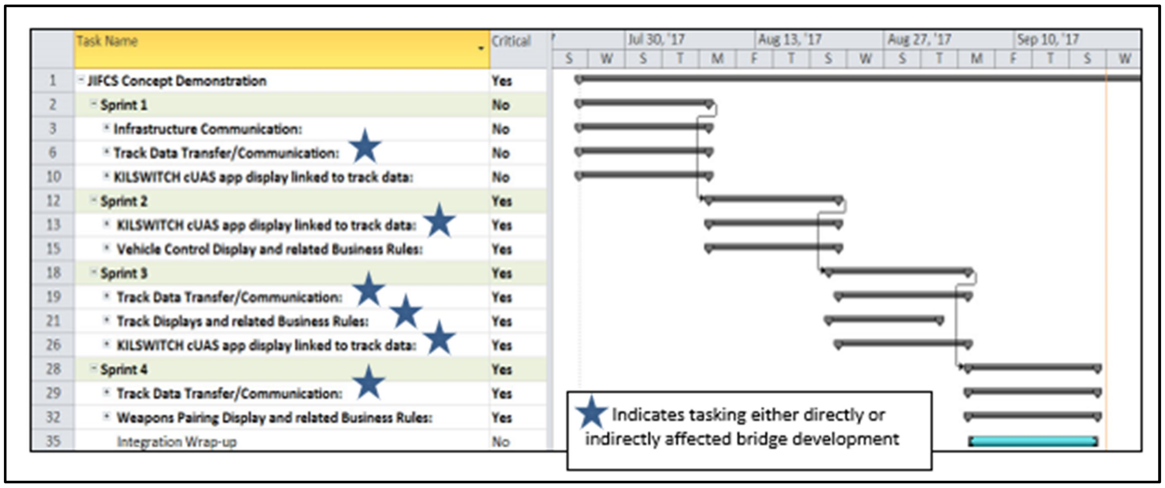
It should also be noted that the timeline also included the time it took to develop the software necessary to display the related data. Therefore, the completion of the demo was principally based on two time related factors:

1. The time it took to develop the bridges that connected the standards (and therefore connected these primary elements)
2. The time it took to display the related data on non-primary elements (e.g., for JIFCS, non-primary elements would be the Topside display)

Additional features were determined based on whether they could be done in parallel to bridge development or support bridge development, while keeping the main focus of the project on bridge development or data that used the messages translated by the bridge. Notice that the UCS message standard was the common connectivity between the primary elements. The reason for having a common standard for connectivity was described in the previous tip. If this technique is followed properly, there should be little to no lag time within the assembly line timeline associated with connecting these primary elements.

After identifying this primary element assembly line timeline, the next key question was what non-primary element features could be integrated without adding any time to the schedule? Understandably, sometimes it is necessary to require additional time, above and beyond the primary element timeline.

Figure 5, the cUAS project schedule, emphasizes that the timeline focused on bridge development or relate data management/display. This meant that most times, these tasks were on the critical path, meaning the smallest (to no) lag time between tasking. The purpose of putting emphasis on the primary element assembly line timeline means that the tasking is mainly focused on assembling primary and non-primary elements instead of creating technology to integrate.



	Task Name	Critical
1	JIFCS Concept Demonstration	Yes
2	Sprint 1	No
3	Infrastructure Communication:	No
6	Track Data Transfer/Communication: ★	No
10	KILSWITCH cUAS app display linked to track data:	No
12	Sprint 2	Yes
13	KILSWITCH cUAS app display linked to track data: ★	Yes
15	Vehicle Control Display and related Business Rules:	Yes
18	Sprint 3	Yes
19	Track Data Transfer/Communication: ★	Yes
21	Track Displays and related Business Rules: ★	Yes
26	KILSWITCH cUAS app display linked to track data: ★	Yes
28	Sprint 4	Yes
29	Track Data Transfer/Communication: ★	Yes
32	Weapons Pairing Display and related Business Rules:	Yes
35	Integration Wrap-up	No

★ Indicates tasking either directly or indirectly affected bridge development

Figure 5. Schedule to Ensure Bridge Work Is on Critical Path and Not Added Features

The tasking "Track Data Transfer/Communication" described in Figure 5 was the incremental development of the bridges. Most of the tasks in Figure 5 containing the words "track data" and "track displays" involved bridge development for primary elements to share data, data management, or data display.

Again, Tip 4's technique allowed the cUAS project team to maintain the goal of reducing the schedule as much as possible to support a rapid demonstration. This technique also supported a scheduling discipline when assessing how much added feature development the timeline could permit without taking the focus away from bridge development or related use of the messages translated.

In general, once a project's original concept is demonstrated, then the next phase of the project would be to add features to the existing framework, knowing the bridge development and message use were working properly. This technique allows the project to lay a foundation, like a solid chassis in a car assembly line. The only difference is that in this assembly line, only one product is being created. This technique also allows the project to identify the "ideal" shortest period of time. In following this technique, future development risk is reduced, and potential customers are given a better understanding of value.

Tip 5: Constantly Remember—It Takes a Village to Raise a … Product

During implementation of a project, Tip 5 can help ensure that the support network is adequately defined and that everyone remains "willingly helpful" during the development cycle. This tip also provides a suggestion on how to deal with folks who may not have time to be in a support network, but need to be available and willing in order for the project to succeed.

Although subtle, this tip should be followed, maybe before all other tips suggested in this paper: Be nice and help everyone as much as possible, because one day a challenging task may need a helping hand and the money to pay for it may not be available. With regard to the cUAS project being discussed, from video folks to weapons pairing experts, 15 to 20 minutes of conversational help about key problems became invaluable.

If individuals on a project team can sincerely promote another group's work or help out, even in small ways, great dividends are received. So, when the time comes that a project team member needs help, someone will probably show up, without a need for a charge object.

In the case of the cUAS project, a kind videographer took a few minutes to do some editing. Yet, those few minutes made a significant difference in the realism of the demo. Figure 6 is a snapshot of the video that took minutes to edit. Another example was a weapons pairing expert who provided a check and balance for related algorithms that were being developed regarding automated weapons pairing. These and other folks provided invaluable support for the success of the cUAS project.

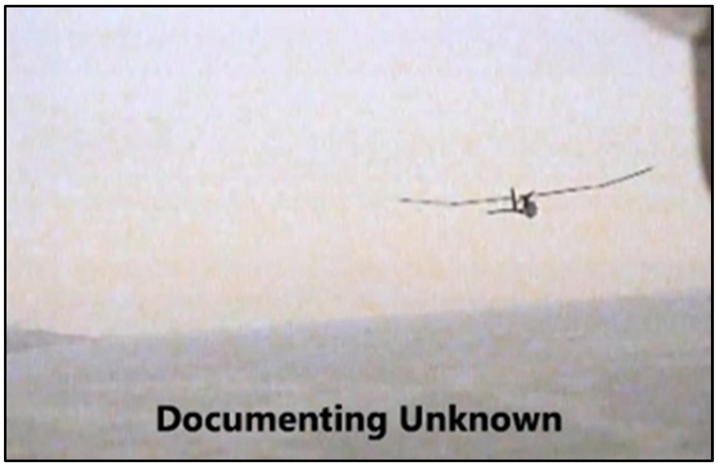


Figure 6. Snapshot of Picture Related to cUAS Project

It's the old saying, "what goes around, comes around." Tip 5: Help when possible, and one day the rewards will appear.

Tip 6: Consider the 80/20 Rule to Support New Talent Growth and Challenging Schedule Goals

There is a temptation to have only experienced people do the work when a short timeframe is involved. Tip 6 describes how to use a version of the 80/20 rule regarding developers and resource management.

In the cUAS example, the project team size was fairly small. Everyone had to pull their own weight. The team consisted of 80% (or four) seasoned developers, and allowed for 20% (one) highly energetic, highly motivated person to work on the project. This meant that the new developer would be allowed more scheduled time to accomplish the assigned tasks. It was important that the new developer was not put under the pressure of being on the critical path. Additionally, workarounds needed to be readily available. "Months not years" is a short-term philosophy, but developing people within the organization provides long-lasting results.

There's no substitute for experience, but there is no greater reward than giving a new developer a chance to shine. As stated, the cUAS development team consisted of five people, represented on the left side of Figure 7. There were senior people on and off of the project and "in the wings" ready to help all the developers, which goes back to following Tip 5: Be sincerely helpful to others and good things will become available.

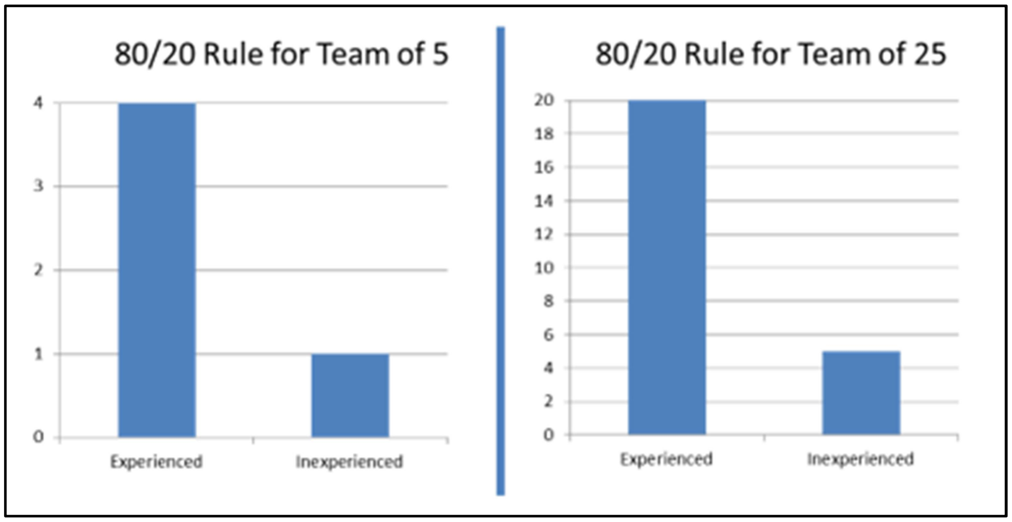


Figure 7. 80/20 Resource Mix to Support Professional Development

As an important note, in addition to providing a successful demonstration of a needed technology, the project also had the privilege to support the development of one who became a much more capable and seasoned software developer—a significant win for the organization.

Again to reemphasize per the cUAS project example, the development of junior personnel is still possible within a rapid development project in which timelines are tight.

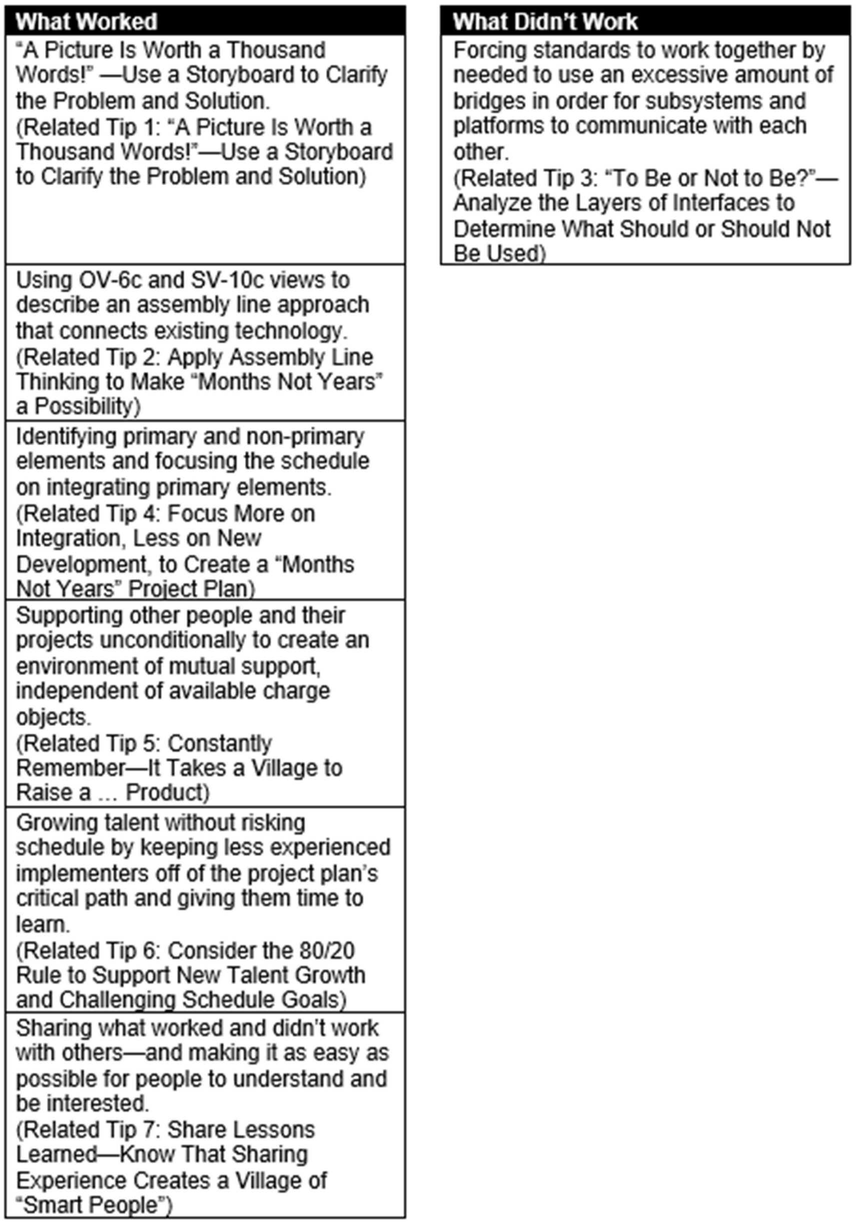
Tip 7: Share Lessons Learned—Know That Sharing Experience Creates a Village of "Smart People"

Along with the "months not years" call for urgency, there is another popular leadership phrase offered: "Take risks!" In some people's mind, that means potential failure. Tip 7 is about sharing knowledge so others learn how to overcome risks and succeed. In other words, it describes the best way to make "lemonade out of lemons." This paper was written to share lessons learned and hopefully help some other project team create an urgently needed product in a shorter period of time for the warfighter.

Lessons learned described via the previous tips came in two popular categories: (1) what worked and (2) what didn't work. Table 1 represents how the previous tips were categorized in terms of what worked and didn't work. During the previous description of each tip, greater detail was included as to why the suggestion worked or why it didn't work.

If one of the seven tips proves to support the ability of someone launching a good idea to support warfighter supremacy, then the cUAS project described has had a significant additional success, beyond cUAS. In other words, this tip suggests that success also includes sharing lessons learned for others to have successes.

Table 1. Lessons Learned: What Worked and What Didn't Work



What Worked	What Didn't Work
"A Picture Is Worth a Thousand Words!" —Use a Storyboard to Clarify the Problem and Solution. (Related Tip 1: "A Picture Is Worth a Thousand Words!"—Use a Storyboard to Clarify the Problem and Solution)	Forcing standards to work together by needed to use an excessive amount of bridges in order for subsystems and platforms to communicate with each other. (Related Tip 3: "To Be or Not to Be?"—Analyze the Layers of Interfaces to Determine What Should or Should Not Be Used)
Using OV-6c and SV-10c views to describe an assembly line approach that connects existing technology. (Related Tip 2: Apply Assembly Line Thinking to Make "Months Not Years" a Possibility)	
Identifying primary and non-primary elements and focusing the schedule on integrating primary elements. (Related Tip 4: Focus More on Integration, Less on New Development, to Create a "Months Not Years" Project Plan)	
Supporting other people and their projects unconditionally to create an environment of mutual support, independent of available charge objects. (Related Tip 5: Constantly Remember—It Takes a Village to Raise a … Product)	
Growing talent without risking schedule by keeping less experienced implementers off of the project plan's critical path and giving them time to learn. (Related Tip 6: Consider the 80/20 Rule to Support New Talent Growth and Challenging Schedule Goals)	
Sharing what worked and didn't work with others—and making it as easy as possible for people to understand and be interested. (Related Tip 7: Share Lessons Learned—Know That Sharing Experience Creates a Village of "Smart People")	

Conclusions

These are exciting times because the U.S. Navy is looking for solutions to warfighter needs, and the need to rapidly deploy a good idea is vital. Good ideas that can be rapidly deployed are more likely to receive funding. The goal of sharing these tips is to provide guidance to help navigate the challenges of assessing, designing, and planning a successful project using OSA and best practices. “Months not years” is a hard mantra to follow. But also consider the phrase, “Take risks!” With the myriad of technology currently developed, putting these elements together to create a new system of systems can be an exciting adventure. Yet, in an ironic way, the excitement is also associated with and sometimes driven by the risk. Prototyping through the use of the proper OSA by applying the appropriate best practices ensures greater rapid deployment success.

In the cUAS project, lessons learned showed the need to focus on sequence diagrams to help determine whether a demonstration was possible within months. The analysis showed how different elements could be integrated rather easily to fit within a months’ timeframe. It showed the need to get as many people involved (within the village) as possible. It doesn’t mean everyone needs to be on the payroll either. Fifteen minutes here and there from various experts regarding key areas associated with your project pays off in big dividends. These lessons learned also describe why a new programmer in the learning phase using the “right” resource management schema can add value, both short and long term, to an organization. Finally, whether a project is successful or not, sharing lessons learned is an important aspect to the rapid development process.

Consider the seven tips described when a good idea pops up and the assessment as to how to begin becomes an inviting next step. Table 2 describes a checklist based on the seven previous tips suggested to help determine if any good idea qualifies to be a “months not years” candidate. Answer the questions in this table using a product prototype.

Table 2. Checklist to Support "Months Not Years" Development Effort

Tip 1: "A Picture Is Worth a Thousand Words!"—Use a Storyboard to Clarify the Problem and Solution		
Tip 1	Was a specific problem defined that the Navy needs to have solved? (Ref: DoDAF Views, OV-1s to start in storyboard fashion)	Yes/No
Tip 1	Were assumptions made about the problem domain and did those assumptions still support Navy needs? (Ref: DoDAF Views, OV-1s to start in storyboard fashion)	Yes/No
Tip 1	Is someone already solving this problem using the same assumptions? If so, was this group contacted and solutions compared? (Ref: DoDAF Views, OV-1s to start in storyboard fashion)	Yes/No
Tip 1	With regard to the solution, is a complete kill chain scenario described? (Ref: DoDAF Views, OV-1 to start in storyboard fashion)	Yes/No
Tip 2: Apply Assembly Line Thinking to Make "Months Not Years" a Possibility		
Tip 2	Have the platforms been selected and their timing relationships been defined? (Ref: DoDAF Views, OV-6c and SV-10c in assembly line fashion)	Yes/No
Tip 2	Have the subsystem elements regarding the platforms been selected and their timing relationships been defined? (Ref: DoDAF Views, OV-6c and SV-10c in assembly line fashion)	Yes/No
Tip 3: "To Be or Not to Be?"—Analyze the Layers of Interfaces to Determine What Should or Should Not Be Used		
Tip 3	Are there straightforward bridge/interface connections between standards of different subsystem elements?	Yes/No
Tip 3	Is there one common communication standard used to connect the different subsystems elements and platforms?	Yes/No
Tip 4: Focus More on Integration, Less on New Development, to Create a "Months Not Years" Project Plan		
Tip 4	Is developing bridge work/interfaces between elements the main driver to the length of schedule?	Yes/No
Tip 4	Is the development of the bridge work/interfaces between elements most times on the critical path?	Yes/No
Tip 5: Constantly Remember—It Takes a Village to Raise a ... Product		
Tip 5	Is a formal (paid) support network, including experts in various fields related to the proposed solution, available to help?	Yes/No
Tip 5	Is an informal (not needing to be paid) support network, including experts in various fields related to the proposed solution, available to help?	Yes/No
Tip 6: Consider the 80/20 Rule to Support New Talent Growth and Challenging Schedule Goals		
Tip 6	Is around 80% of the development team experienced regarding the work that needs to be performed?	Yes/No
Tip 6	Are the experienced developers assigned work on the critical path?	Yes/No
Tip 6	Are the inexperienced developers not assigned work on the critical path?	Yes/No
Tip 6	Are the inexperienced developers highly motivated?	Yes/No
Tip 6	Are the inexperienced developers provided more time to complete tasks to account for their learning process?	Yes/No
Tip 7: Share Lessons Learned—Know That Sharing Experience Creates a Village of "Smart People		
Tip 7	Are you identifying what worked and why?	Yes/No
Tip 7	Are you identifying what didn't work and why?	Yes/No
Tip 7	Are you making suggestions to improve the process?	Yes/No

If the answers to the questions in the checklist in Table 2 are predominately "yes" based on a prototype, not just a paper exercise, then those good ideas are more likely to be a good candidate for rapid product deployment. And rapidly deployable good ideas are more likely to get funded. And if funded, those good ideas are more likely to be deployed in time, proven through your prototype, to ensure U.S. combat superiority is maintained, saving lives and eliminating enemy threats.

References

AirTALKS [Blog post]. (2017, August 15). Retrieved from https://myteam.navair.navy.mil/corpapps/NAVAIRComm/NAVAIRBlog

Dam, S. H. (2006). DoD architecture framework: *A guide to applying system engineering to develop integrated, executable architectures*. Marshall, VA: SPEC.

DoD. (2010). The DoDAF Architecture Framework Version 2.02. Retrieved September 1, 2017, from http://dodcio.defense.gov/Library/DoD-Architecture-Framework/

Grady, J. O. (2010). *System synthesis: Product and process design*. Boca Raton, FL: CRC Press.

Kerzner, H. (2013). *Project management: A systems approach to planning, scheduling, and controlling.* Hoboken, NJ: John Wiley & Son.

Langford, G. O. (2012). *Engineering systems integration, theory, metrics and methods*. Boca Raton, FL: CRC Press.

Maier, M. W., & Rechtin, E. (2009). *The art of systems architecting*. Boca Raton, FL: CRC Press.

Solomon, B. (n.d.). Crafting your “elevator pitch”; Building a tech transition plan.

Distribution Statement