MTR 04B0000036

MITRE TECHNICAL REPORT

Architecture Development Lessons-Learned:

A Three-Year Retrospective

Jan 2004

Carlos Troche
Gerald F. Eiden Jr.
Frederick C. Potts

Sponsor:	Col Bruce R. Sturk	**Contract No.:**	FA8721-04-C-0001
Dept. No.:	D440	**Project No.:**	03046970CX

MITRE

Corporate Headquarters
McLean, Virginia

"But what ... is it good for?"

– Engineer at the Advanced Computing Systems Division of IBM, 1968, commenting on the MICROCHIP.

Architecture provides vision, roadmap and control mechanisms – the basis for strategic planning in any enterprise.

– John J. Colligan, Project Leader, rendering the AFC2ISRC/CXCQ Architecture Mission

Abstract

This is a retrospective synthesis of three years of experience developing detailed architectural views in compliance with the Command, Control, Communications, Computers Intelligence, Surveillance, and Reconnaissance (C4ISR) Architecture Framework, the DoD guidance that implemented the statutory requirements of the Clinger-Cohen Act of 1997. It represents the collective judgment of nine professionals, all of whom had Air Force operational, and/or, system development; and, experience, or both; were formally trained in architecture development; and, have been dedicated to the development of these architecture views almost exclusively. After more than three years of developing architectures, all these lessons-learned point to one basic conclusion: architectures are developed to be used. Thus it is incumbent upon the architecture developer to work with the user to create something that has practical and immediate application to that user's needs. Everything else – development process, tools, methods, etc. – should be subordinated to this utility. We take the point of view in this report of an action or mid-level staff officer who has just been tasked with developing an architecture. We try to provide a minimum set of "rules of the road" lessons learned to assist in architecture development its tools. Two appendices detail our experiences with SA (Popkin Software's System Architect) and our analysis of the impacts of the new security policy imposed by the Office of Management and Budget on architecture information.

Keywords: architecture, lessons learned, architecture tools.

Table of Contents

1 Executive Summary

This is a retrospective synthesis of three years of experience developing detailed architectural views in compliance with the Command, Control, Communications, Computers Intelligence, Surveillance, and Reconnaissance (C4ISR) Architecture Framework, the DoD guidance that implemented the statutory requirements of the Clinger-Cohen Act of 1997. It represents the collective judgment of nine professionals, all of whom had Air Force operational, and/or, system development; experience, or both; were formally trained in architecture development; and, have been dedicated to the development of several of these architecture views almost exclusively. After more than three years of developing architectures, all these lessons-learned point to one basic conclusion: architectures are developed to be used. Thus it is incumbent upon the architecture developer to work with the user to create something that has practical and immediate application to that user's needs. Everything else – development process, tools, methods, etc. – should be subordinated to this utility.

This report is neither exhaustive nor prescriptive. It represents but one set of experiences that may or may not be applicable in other settings. We mention organizations deliberately to maintain the proper perspective and context on the experiences related.

We take the point of view in this report of an action or mid-level staff officer who has just been tasked with developing an architecture. We try to provide a minimum set of "rules of the road" lessons learned to assist in architecture development and its tools. Two appendices detail our experiences with SA (Popkin Software's System Architect) and our analysis of the impacts of the new security policy imposed by the Office of Management and Budget on architecture information.

- Lessons dealing with the architecture development process focus on the most critical preparatory aspects, because, it is in this stage where – in our experience – the future success or failure of the architecture is sealed.

    - Our overarching conclusion is that an architecture is complicated and resource-intensive enough that it should be managed as if it were a program, with a clear purpose and, scope and clear lines of responsibility. Because the Air Force, by definition, made architecture development a collaborative process (see AFI 33-124), we further strongly recommend documenting the degree of collaboration and interaction among all players in an architecture, and appointing a Chief Architect vested with directive authority to maintain accountability of the partners.

    - Across our three-year journey we have also found that people with certain experience or backgrounds make for better candidate architects than others. Professionals with diverse operational and long-term planning experience, and a clear predilection for jobs that deal with

    complexity and seeking solutions among competing alternatives clearly make the best candidates to learn the architecture trade. As with any collaboration, these skills cannot be effectively employed unless there is a strong policy framework that protects the integrity of what they develop, both from accidental errors and from unauthorized disclosures. Thus we recommend establishing strong configuration and security management policies that are enforced by everyone, but particularly by a single person charged with their daily management within the architecture development organization. We go as far as recommending that architecture development not start until these policies and procedures are clearly established, especially in light of the new Office of Management and Budget policy that requires protecting architecture data and tools as "mission critical" systems (see the detailed discussion of the policy and its impacts in Appendix B).

  - The complexity of developing architectures is managed through formal methods e.g., structured analysis, object-oriented, that often bring their own language and notation. Hence, architecture proponents suggest that senior-level sponsors of architectures should be strongly encouraged to focus more on the results of applying the architecture and less on the details within it. In our experience, the investment in architectures is such that sponsors cannot be dissuaded from "looking under the hood," often being repelled by what they see or becoming blind advocates of the one method they learned. We strongly recommend actively engaging the sponsors throughout the development, placing these development details in the proper framework of utility, and engaging other potential users of the architecture early on to serve as spokespersons for the architecture.

  - In the C4ISR community, integration is an unquestioned goal of most efforts. Thus, architecture sponsors should count on an effort to "integrate" their architecture with other projects. Because "integration" is in the "eye of the beholder," we strongly recommend taking the time early on to define the degree of integration with the same zeal used to define a purpose, scope, context, and scenario for the architecture.

- Lessons dealing with architecture development tools focus on our experiences with a variety of tools ranging from Microsoft Office, netViz, and Macromedia Flash to formal modeling tools like Popkin Software's System Architect (SA). Overall, we believe that any candidate tool should have a proven record of being able to support, secure and clearly present an architecture similar to the one being contemplated. Unfortunately, the likelihood of meeting such a standard today with a single tool is very low. Therefore, we are convinced that only using a variety of tools will provide the best results.

- Standard office automation tools (e.g., Microsoft Office) store and present large amounts of information quickly and efficiently. Because these are also pervasive throughout the using communities, office automation tools enable quick penetration of the architecture into daily use. Unfortunately, standard office automation programs lack the controls of formal methods and modeling languages and are unable to display dependencies and parallelisms among activities, processes, and supporting technologies – relationships that are essential to analyze the behavior of the system the architecture depicts.
- Modeling tools (e.g., System Architect, Rational Rose) implement formal modeling methods used across industry. They are generally well suited to illustrating dependencies and parallelisms, but often their ability to handle large data stores with multiple relationships is limited. We suspect we have run into these limitations with System Architect because the tool's behavior became increasingly erratic as the data and relationships increased.
- None of the formal modeling methods fully meets the requirements of the C4ISR Framework, and most must often be "extended" when used. What is an allowable "extension" is, again, in the "eye of the beholder." Hence, one should employ these tools with the understanding that a bias will be needed to "extend" them to meet the requirements of the Framework and, that this bias needs to be decided early on in the development process, should be applied uniformly, and will be the subject of disagreements with those outside the team. In some cases, the tools are "extended" to meet the C4ISR Framework requirements. In our experience with System Architect, this extension (the C4ISR "Overlay") was based on erroneous interpretations of the Framework that required extensive workarounds. The XI-supported extension attempted to corrected System Architect's problems but created some of its own. In addition to the background section, Appendix A details our experience with System Architect and its XI-supported extension.
- None of the modeling tools we evaluated implemented configuration or security management features that adequately protected the data collected within it. For most vendors, these features were considered "external" to the tool, a situation that almost guarantees the loss of or unauthorized disclosure of the information. Even the nominal "check in/check out" feature in System Architect that permits the selective, read-only extraction of part of the data can be easily thwarted to enable changes to the extracted data.

Overall, we strongly recommend thorough testing or review of customer experiences before designating any tool a "preferred" one. On the specific case of the HQ USAF/XI preferred tool – SA and its XI-developed extension –, we recommend developing a new strategy for using SA in its native form without any modification, and using SA within the context of a suite of tools and for a scale appropriate to its capabilities.

2 Architecture Development Process

Background

The Clinger-Cohen Act (CCA) of 1997 required the development and implementation of a series of management tools – among which were architectures – to guide the investment in information technologies (IT) within the Federal bureaucracy. Provisions of the CCA accounted for and impacted automated information and embedded systems. The Office of Management and Budget's (OMB's) A-130 circular and later Office of the Secretary of Defense policy further reaffirmed this mandate and provided additional guidance for its execution. Ultimately the DoD translated the CCA and OMB guidance into the C4ISR Framework, a standard that defines the minimum ways of depicting an architecture to be fully described by three "views": operational, system, and technical.

The Air Force, in the meantime, established the Air Force Command, Control, and Intelligence, Surveillance, and Reconnaissance Center (AFC2ISRC) and charged it with – among other missions – developing and maintaining ***"...near-term and long-term operational architectures for AF/XO...integrate[ing] MAJCOM/FOA inputs in design and management of C2ISR system-of-systems architecture in conjunction with the acquisition community..."*** AFI 33-124 institutionalized this role by assigning the development of the operational "views" to the AFC2ISRC for the C2 & ISR mission areas. Some of the staff elements consolidated into the AFC2ISRC had been modeling operational processes for at least three years prior to CCA, and they brought that experience and those results with them into the AFC2ISRC. The AFC2ISRC organized a dedicated architecture team and published its first major architecture in 2000. The team varied some, but it generally consisted of the same nine professionals – multi-disciplined, with varying operational, engineering, and other technical backgrounds – attempting to describe Air Force roles in C2 & ISR within and outside of a major theater war context, in the near, mid, and far terms. Generally all released products described Air Force operations within the theater context; the Air Force's role in theater/reachback support was developed but never officially released. In 2002, control of the AFC2ISRC was transferred from the Air Combat Command (ACC) to a new organization in the Headquarters USAF staff, the Deputy Chief of Staff for Warfighter Integration (HQ USAF/XI). HQ USAF/XI has continued to ask for the same type of products that were delivered before to the AFC2ISRC leadership though they have established new rules governing the tools used to develop them.

Lesson #P-1: Understand what you are getting into before proceeding.

An architecture is a model of a complex system, a detailed description of the systems' components and the relationships among those components developed to evaluate the systems behavior and performance. It is not a set of C4ISR Framework products. Because an architecture is detailed, it takes significant amount of time (weeks, months), resources (persons, tools), and commitment (yours and your leadership). Disabuse yourself of the notion that this is a brief, quick and easy project. At best, this is your legacy to those who will come after you. Your professional reward (and that of your sponsors as well) will be the stewardship of this investment as it grows from concept to reality. If you are the (un) lucky one "stuck" with an architecture tasker, time spent understanding what an architecture is could save you orders of magnitude of time and aggravation trying to organize and conduct an architecture development effort or convincing your leadership that one isn't needed.

Metrics & Mitigation: If you are tasked to build an architecture, call on experienced architects and seek their advice on the purpose, scope, and commitment required for your "tasker." Evaluate your boss' understanding of architectures and commitment to them, as measured by his or her attention span when you mention architectures. Ensure that the leader understands the commitment architecture development represents, including the fact that anything that temporarily prevents you from being totally dedicated to the architecture (e.g., routine staff taskers) will likely impose a penalty on the architecture development of twice the amount of time to recover for every day spent off task. Make sure both you and your leadership understand that architectures must be managed as a **program**, with clear **leadership** and single-mindedness of purpose. Architectures represent such a significant investment that any hint of a "box checking" motivation or an unwillingness to carve out time to understand and actively guide the development of the effort should raise a cautionary flag against attempting the development. If this initial research reveals major mismatches between what you are being asked to do, your resources, and, especially, your leadership's commitment, attempt to delay or "kill" the tasker.

Lesson #P-2: Architectures must have a specific purpose (at least one) and should not be developed until there is at least one agreed-upon, purpose (and "because the CCA says we need one" is not a purpose!).

Just as there is no such thing as a "generic" weapons system, there is little sense in a "generic" architecture. Weapons systems are built for specific purposes – interdiction, counter air, airlift, command and control, reconnaissance, counter space, et al. Equally, an architecture must have at least one purpose that guides its scope, development, and use. In the case of multiple sponsors, different views are likely as to what that purpose should be. Sometimes these differing views cannot be reconciled in a way that results in a clear unambiguous statement to guide the development of the architecture. In such cases, the standard reaction is to continue negotiating until agreement is reached. Generally this results in a purpose statement so generic and ambiguous that although it satisfies every sponsor's (different) interpretation of the purpose, it provides little guidance to the architecture developers. This approach guarantees that the resulting architecture will not satisfy anyone's individual expectations.

Metrics & Mitigation: Spend some time discussing with your leaders the purpose of the architecture. Frame it in very practical terms: At the end of what could be a considerable period, what would they expect to find or get from the architecture? Encourage a very parochial, selfish approach so you can get their actual views and desires. In the end, you may find you have incompatible objectives, a clear indicator that you need more than one architecture. This may turn out to be a preferred option, because each architecture would be more specific and thus more relevant to the sponsor's needs. Consider using "canned" purpose statements to "prime the pump" in your discussions, "The purpose of the architecture is to..."

- Visualize a problem and its impacts.
- Understand the impacts of specific investment decisions.
- Understand where and how performance may be best improved.
- Optimize investment decisions by evaluating its impacts across the larger context where the investment takes place.
- Understand the impacts of procedural changes and new technologies.

Lesson #P-3: Manage an architecture as a development program with a clear, unambiguous Chief Architect, and a detailed *written* charter for the architecture that is approved by your leadership.

An architecture requires such a single-mindedness of purpose and dedication that it must be managed as a program with clear lines of authority and responsibility vested in a Chief Architect. By the way, the old saying of "where there's more than one in charge, no one is in charge" applies in architecture development as in other projects. This administrative structure, essential for architecture development, needs to be clearly documented and agreed-upon by the sponsors of the architecture.

Metrics & Mitigation: Just as you would in any situation where your personal fortunes are at stake, zealously document the agreements with your leadership. Ensure that the document clearly shows who the Chief Architect will be, and avoid the bureaucratic euphemisms that tend to dilute authority and diffuse the directive and advocacy responsibilities of that position. Exploit the flexibility inherent in the Framework's All Views (AV)-1 product to document all taskings and obtain approval. Ensure that major sponsors physically sign this "charter;" there's hardly a better way of reminding them of their accountability in this process. As you conduct progress reviews, lead in with the signed "charter," as the guiding light for your development.

Lesson #P-4: A stated purpose must be bound by a statement of scope, point of view, and a scenario as context for the architecture.

Scoping and committing to a point of view adds reality to the purpose of the architecture. It also serves to ensure that the architecture sponsor understands the effort and commitment. A scope statement should be developed in both the positive and the negative cases. That is, there should be little doubt as to what will be **included** and **excluded** by the boundaries of the architecture. A point of view helps further hone the scope by determining what areas within the scope will be emphasized. Both scope and point of view should be discussed in terms understandable to all participants. Choosing a scenario helps scope the scale of the architecture development effort. This scale, in turn, is one of the key drivers of the level of effort and investment required to complete the architecture. For instance, an architecture that describes all air operations in a theater war scenario is of a larger scale and will take a larger effort than one focused on search and rescue of a downed airman within a particular sector of enemy territory. Discussions that devolve into arguments over Framework products e.g., AV-1, OV-1, should be quickly steered back to the substance of the subject that created the need for the architecture to begin with.

Metrics & Mitigation: Test the commitment to a scope by discussing those areas that would be "out of scope" by the selected scope statement and point of view within the agreed-upon scenario. The more disagreements the discussion engenders, the more concerned you should be that the architecture and its purpose are not understood. The degree of disagreement is an early indication of the architecture's potential to fail to meet expectations. If the discussion begins to center on Framework products, you will need to quickly refocus it on the content of the prospective architecture. If necessary, be zealous and overbearing! Like the "mission creep" that impacts many operations, scope tends to "creep" in an architecture, especially as those involved at the beginning quickly forget the limitations imposed on the scope to make the architecture realizable and begin to get creative.

Lesson #P-5: Architecture development requires collaboration. Ensure your collaborators are equally committed to the same effort as your organization.

The Air Force decided in AFI 33-124 to make architecture development a collaborative, decentralized effort among, – for example, – operators (AFC2ISRC) and system developers (ESC and AFCA). By definition, then, any architecture developed in the Air Force will require collaboration across very different organizations, each with its own agenda and priorities. So, in addition to having to deal with the challenges of developing the product, someone tasked to develop an architecture must also succeed in convincing other organizations to support their project at the level of effort required to complete it on time. In this day of netted staffs, advocating and communicating are not a problem, but what continues to be critical challenges are the leadership and accountability required to maintain an equal level of commitment across organizations.

Metrics & Mitigation: Test your partners' level of organizational commitment early and often. Any wavering, – for example, an inability to obtain their leadership commitment to the priority, purpose, scope, scenario, and viewpoint of the architecture – should be a warning signal. In such cases, limit the scope of the collaboration to that which can be realistically achieved. If commitment and support evaporate altogether, ensure that you inform your sponsors **early** in the process that your efforts will be hampered by the absence of collaboration.

Lesson #P-6: How to develop architectures can be learned; however, certain backgrounds facilitate learning.

At its core, architecture development demands analytic *and* engineering skill sets. This does not mean, however, that only professional systems analysts and engineers can be good architects, but that the further away the experiences of a candidate architect are from the outlooks of analysts and engineers, the steeper (longer) the learning curve will be. Generally, a candidate whose background demonstrates a clear preference and ability to deal with complexity and an ability to wrestle it into practical, actionable results within a specified period of time, is a good candidate to become an architect.

Metrics & Mitigation: If you are empowered to do so, evaluate the background of candidate architects and consider the following experience or backgrounds (in addition to domain expertise) as indicators of good future potential to easily train as architects:

- Prior "Architects" (C4ISR Framework-experienced)
- Systems Engineering
- Software Engineering
- Computer Scientists (especially with embedded, real-time, development experience)
- Mechanical/Control Engineering
- Systems Analysis
- Doctrine & Procedure development
- Standardization & Evaluation
- Long-term planners
- Project Management (Defense Systems Management College-certified)
- Training Management

- Implementation Management (installation, cutover, and operations management)
- Electrical/Electronic Engineering

Seek to balance your team skills, if possible. Avoid overspecialization. Additionally, any candidate who cannot articulate his or her understanding of analysis, architectures, or immediately retorts with the "Keep-It-Simple-Stupid" principle during an interview should not be favorably considered because their bias is to reject complexity outright and thus lose the power of architectures to address the sponsor's purpose.

Lesson #P-7: Besides domain expertise, the most important skill to have in an architecture development team is Configuration Management (CM). Without CM you run the risk of losing the architecture work, having it misquoted and misused. In some cases, this can have legal consequences. Do *not* start developing an architecture without agreed-upon CM policies.

Assuming that a prospective architecture development will require the talents of more than one person, it is fair to also assume that you will need a governance structure (e.g., process, procedure, policies) to prevent the work of one architect from being corrupted, overwritten, or otherwise misused by another architect or a party outside your team. In turn, this governance structure needs to be flexible enough to allow the architecture to be exposed, vetted, and ultimately used by those you authorize, all without corrupting it or using the information out of the intended context. The collection of policies and procedures that governs the management of all this data is generally known as CM.[1] The sponsors as well as the team must adopt and adhere to CM early in the architecture development effort. Significant resistance is common: CM often sounds Byzantine; however, you should insist on the protections it offers and have someone designated to be your primary Configuration Manager (CMgr).

Metrics & Mitigation: How do you know if you have the right person for CM? You are on the right track the candidate CMgr is able to propose policies and procedures – e.g., he or she describes how to ensure you that the right data and products are maintained and disseminated – *without* degenerating into a discussion of automated tools for configuration management and how they would be employed. If your team members have no clue what CM is, seek assistance. Above all, do *not* ignore this critical area; you are risking, at worse, the entire viability of the architecture; at best, a lot of "redo" you will not have time to undertake. No CM policy, little understanding of the subject, or a

[1] CM is a discipline applying administrative and technical direction and surveillance to: identifying, documenting the physical, functional, and performance characteristics of items; baselining, controlling changes to, providing status on, and conducting audits on those characteristics. References: Incervic, E. L., *Configuration Management: A Software Acquisition Guidebook*, TRW Software Series, November 1978; and, *Capability Maturity Model Integration*, Version 1.1, Software Engineering Institute, December 2001.

penchant for describing it in terms of automated tools should raise a warning flag concerning the team's ability to handle the scale and protection of the architecture adequately.

Lesson #P-8: Because architectures take a long time to develop, count on the architecture sponsors' losing interest and, enthusiasm, or radically after changing their outlook on what they want within the first 3-4 weeks after starting the development. Prepare to continuously "promote" your architecture and to highlight its importance to the mission.

Attention span is a precious commodity; so much so that medical science recognizes that attention deficits are serious conditions requiring treatment not only in the young but in adults as well (see adultadd.com). This, coupled with the nature of staff work where issues are never fully resolved but repeatedly reengaged, increases the number of issues a leader has to deal with, shortening the amount of time any one issue is considered in full. The shorter this period becomes, the more difficult it becomes to recall it in detail in the future, especially for complicated issues like architectures. A week in the architecture sponsor's schedule is often a long time; a month, an eternity. So, you should expect your architecture sponsor(s) to quickly lose interest or feel completely lost as to the motivation for the enormous investment in something like an architecture that takes dedicated work and a long time to bear fruit. Many respond to this simple limitation of human nature by attempting to flood the sponsor with information about the architecture and its progress. In today's netted environment, this is relatively easy, but it just adds more e-mail to a growing stack of unread or belatedly read missives in the sponsor's mailbox. Readers learn to quickly scan at subject lines and relegate those e-mails that require the most pondering (like architectures) to the very last, "if-I-get-to-it," pile. In one stark example, the architecture sponsor reached into his desk, pulled out a jar of aspirin and took two before he would consider discussing the subject.

Metrics & Mitigation: Understand that it is inevitable for the sponsor's interest to wane in long, complex projects, often regressing through three stages: zealous interest and advocacy of the need for an architecture; sober realization of the complexity and enormity of the task; and, "grudging" acceptance and disillusionment with the entire idea. Your mitigation strategy should seek to keep your sponsor's interest somewhere between the first and second stages, and away from the last one, by always providing "fresh" outlooks on the utility and application of what your team is doing with what is being developed, even before the final product is ready for release.

The suggested mitigation strategy requires two components: a communications plan and the opportunity to promote the use of the architecture as it is being developed (i.e., a partial architecture) by its prospective user(s) so they can become, in turn, the architecture's endorsers. This two-pronged approach is analogous to that used by weapons systems under development, where sometimes the horizon to operations is a decade or more away.

- The communications plan is nothing more than a means of organizing how you are going to publicize the architecture, its progress, and its benefits to your sponsors and ultimate users succinctly without being overbearing. A wry observer once commented that a new program initially does not need great accomplishments, just a logo, a coffee mug, a T-shirt, a web page, and a few promotional trinkets to give it a distinct identity and promise, all while teasing the users' interests. Similarly, your communications plan should create an identity for the architecture while attempting to entice interest in its promised benefits. One of many components of your communication strategy could be an advanced schedule of periodic meetings to brief the status of the architecture development and focus attention on the task.

- The second element is a little harder to manage but of potentially greater benefit to the architecture and to its utility. The sponsor provides executive supervision over the development of the architecture, but the actual architecture users are often others. Empowering your team to interact with these users early and often during the architecture development could create a pool of goodwill and open–mindedness that would enable greater acceptance of and curiosity about the architecture once delivered, especially if the interaction results in adjustments and tailoring of the end-product to those users' needs. Using this strategy, those users interacting positively with the architecture development team become spokespersons for the use of the architecture in their operation. Instead of boring the sponsor with constant architecture information from the development team, these users would rekindle his or her interest in the subject more frequently by addressing it from the users' individual perspectives. However, realizing the benefits of this interaction requires the sponsor's initial recognition and acceptance that the architecture and its views are only useful if they are applied; that they can only be applied if they are tailored to the needs of the user; and, that this tailoring must be led by someone who is interested in applying the architecture and who thoroughly understands it and the user's needs (i.e., the architect, not the user). It also requires careful scheduling and resource management so that architecture development effort is not delayed or derailed under the influence of a particularly capricious user.

Lesson #P-9: Develop and enforce a set of security policies and procedures to govern the control and release of architecture information.

The OMB directed all Chief Information Officers in January 2003 to treat architecture information as mission critical to be secured in accordance with the standards of the Federal Information Security Management Act (FISMA). The OMB policy and FISMA require the establishment of a risk-based security management program to protect every mission-critical system and application (e.g., architectures by OMB's designation) and an annual evaluation of the information security management program protecting these mission-critical systems. FISMA further requires that the security status of mission-critical systems and information be reported to Congress periodically. To date, detailed guidelines for implementing this new policy and the FISMA requirements on architectures have *not* been issued, although we have verified *that the policy mandate has not been reduced, removed, or otherwise modified.* The impact of this mandate on architecture development is detailed in **Appendix B** to this report.

Metrics & Mitigation: While we await further guidance, be sure that you keep written records of who is issued architecture data, when and for what purpose. Issue the data in formats that can't be easily altered, unless you have willfully authorized the recipient to reuse and alter the data. Ensure that your data is clearly labeled as to its availability for public release (i.e., use For Official Use Only) in the case of unclassified information, and ensure that you have approved methods and systems for secure information processing in the case of classified information. Periodically audit your records and determine the disposition of the internal and external distribution (e.g., whether the information still exists, whether the recipient still possesses the information, and what its time/date of destruction is).

Lesson #P-10: Architecture sponsors should only be interested in the results of applying the architecture, not in how it was developed or what the Framework products look like. If your sponsors, however, do want to view the architectures in detail, prepare to educate them on the design methodology and the modeling language used. More important, be ready to defend your choices.

Architectures are developed using a design methodology, often instantiated by a modeling language. There are several design methodologies and modeling languages, all stemming from individual sectors of industry or academia. The two most common ones used in developing C4ISR Framework products are Structured Analysis Design Technique (SADT) and Object-Oriented Analysis and Design (OOAD). SADT has a long history in major projects like the space missions. SADT models are often documented with modeling languages like IDEF0.[2] OOAD is more modern and stems from software development. It is

[2] For those unfamiliar with the Integration Definition for Function Modeling (IDEF0) language, Federal Information Processing Standard 183, it is a set of rules that define how functions and their interrelationships may be depicted. Two key IDEF0 principles are inheritance and node or actor indifference. You start IDEF0 models at the very top with a "context" diagram, which is a single statement of the function you will decompose and assign it inputs, outputs, controls (guidance), and mechanisms (systems). Under inheritance, functions in IDEF0 are broken down into their component subfunctions. Functions decomposed into subfunctions bequeath unto their children all inputs, outputs, controls and mechanisms (ICOMs) that affect the parent; and, vice versa, all subfunction inputs, outputs, controls, and mechanisms are inherited "upwards" into the parent. Depending on the complexity of the model, this can be a very large number of ICOMs, rendering the diagrams at the top unreadable. Artifacts within the

often documented with modeling languages like the Unified Modeling Language (UML). A good rule-of-thumb is that OOAD/UML is far superior to other methods for the design of software-intensive systems. A whole cottage industry has developed around the use and advocacy of SADT/IDEF0 versus OOAD/UML, especially as it pertains to the C4ISR Framework products. So, it is likely that your sponsors have heard about "use cases," "objects," "functions," "inputs," "outputs," and so forth and feel comfortable enough with this deceptively simple vocabulary to demand that OOAD/UML or SADT/IDEF0 be the norm for your architecture. Furthermore, it is very likely that they have reached this conclusion after one presentation from an advocate of either "religious" camp.

Metrics & Mitigation: You will need to select a design methodology and modeling language to develop your architecture. Obtain the advise of seasoned architects when making your selection. Ultimately, however, both OOAD and SADT will produce the same results, but they will require entirely different skill sets to understand them. It won't be obvious to any but a trained eye that an OOAD and SADT architecture refer to exactly the "same" reality.[3] Thus you need to obtain a detailed understanding of each methodology and be ready to defend your choice, ensuring that you frame your argument within the purpose, scope, point of view, and scenario of your architecture. This is probably the only way you can prevent your development from being engulfed in what is ultimately an endless discussion of nothing more substantive than individual preferences.[4]

Lesson #P-11: Lead your architecture development with the same zeal and practices you would use for a major program.

You have zealously developed and obtained the approval of your architecture's purpose and scope; carefully selected and trained your development team members (if possible); and, you are ready to begin development, but realize that everyone on your team has a slightly different idea of the architecture and its bounds.

modeling language like tunneling enable the modeler to limit the number of ICOMs that inherit "upward" without losing fidelity. The IDEF0 methodology closely resembles the outlining methods we all learned in secondary education as a way to understand the components of a major concept and their relationships. Node or actor indifference refers to the assumption made at the very beginning of an IDEF0 model that the human, machine, or process interacting with the functions in the model is either a single entity or it is otherwise irrelevant to the context of the model because the model is focused on *what* is being done, independently of *who* is doing it.

[3] The argument among the adherents of both techniques often becomes so intense that it overshadows the real purpose of their use: enhancing combat capability. Beware, then, of critics who often base their comments, not on the substance of what you are depicting, but on the technique you used.

[4] As of this writing, OOAD and SADT are the two big "camps" promoting their "truth as the only one." But, stay tuned; Aspect-Oriented Programming (AOP, some say a modality of object-oriented programming) is making its debut, so you shouldn't be surprised if, by the time you read this, AOP's design technique has joined this fray as the technique *du jour* for C4ISR architectures.

Metrics & Mitigation: Just as we recommend having a strategy for communicating to outside organizations (see lesson #**P-8**), we strongly recommend having an equally organized approach to internal communications. The objective of internal communications is to ensure that all team members understands what is expected of them, in exactly the same form and level of detail as you have tasked them. There's probably no better way of making this happen than by *jointly* developing a detailed *internal schedule* for the development of the architecture, a *step-by-step plan* to build the architecture and everyone's role in that process. This is both time consuming and exasperatingly laborious because you are dealing with how humans (mis)understand one another. Remember that this is not just making sure everyone "gets the word," but rather guaranteeing that everyone internalizes the purpose and spirit of the task. If part of your team is not collocated with you, it will be that much harder as you deal with the logistics of transportation and scheduling available time. His process has no shortcuts. You should count on *repeating* it often during the first 30 – 45 days of development until you can be reasonably assured that enough of a "mind meld" has occurred among your team members that everyone is building the same architecture.

Lesson #P-12: Get ready to "integrate" your architecture.

You will not develop your architecture in a vacuum. More than likely, you will be asked to be "integrate" with other architectures. And, as we mentioned before (lesson #**P-2**), your architecture is bound to have a unique purpose, scope, point of view, context, and scenario, all likely to be different than those of the architecture you are being asked to integrate with. Also, "integration" seems to be one of those terms of art that is left deliberately vague so it can be used indiscriminately without attribution. So, it is absolutely essential that you pin down what your sponsors mean when they ask you to "integrate" your architecture and that you plan for, and implement it early in your development.

Metrics & Mitigation: Ensure that your sponsors understand that architectures are not "self-integrating," if nothing else because you must ensure a degree of commonality that is likely not to exist between any two architectures.[5] At best, integration is a matter of degree: the higher the degree of integration, the higher the degree of commonality of purpose, scope, point of view, context, and scenario between the integrating architectures. The converse is also true. We recommend spending some of the time allocated to defining the purpose, scope, point of view, context and scenario of the architecture in defining the degree of integration you are willing to invest in. You may be asked to integrate your architecture with one that is so radically different that it will be nearly impossible to show except in the most contrived way. In such cases, when the commonality is not there, you may suggest a mapping document that shows how your architecture's context and other elements "map" to, match, the others (where they, in fact, do). On the other hand, yours may

[5] Some architecture experts firmly believe architectures can *never* be integrated, just compared.

simply be a detailed version of a portion of a higher-level architecture. In which case, your "integration" would simply be an extension of the higher-level architecture into a more detailed area. In both cases, mapping or extension, you should plan for the time and logistics of making this happen because it will involve your entire team and likely those individuals who developed the architecture you are integrating into. Count on integration taking long negotiating sessions that should be tried out early in the development so that the teams get used to working together.

3 Architecture Development Tools

This section summarizes our collective experiences with tools for architecture development. **Do not skip reading the Background and the two Appendices** because you will miss the entire context of the lessons learned that have been derived from our experiences.

- **The Background** addresses our experiences with tools since we started developing architectures as a dedicated team. Most of this information however, is focused on our latest experience with a formal modeling tool called Systems Architect (SA).
- **Appendix A** provides a detailed chronology of the efforts made to deal with SA's limitations within our context and will give you an appreciation of the level of difficulty of the challenge and the importance of CM.
- **Appendix B** details the impact of security policies on the future selection and use of architecture development tools and the features you should keep in mind when evaluating prospective tools.

Background

Winter 2000 – Fall 2002

We used a variety of tools to develop operational views. Until late 2002, the tools used were Microsoft Access, Excel, Word, PowerPoint (the MS-Office Professional suite), and netViz. Architecture data was captured in an MS Access database, and netViz provided a method of diagramming architecture products that were connected to the database. Supplemental tools were used to enable a more user-friendly presentation of the architecture, e.g., Microsoft FrontPage, Adobe Photoshop, Premier, and Acrobat. Macromedia Flash was also used to enhance, by way of multimedia presentation, animation, and interactive menus, the navigation to MS Office, Adobe Acrobat, and Hyper Text Markup Language (HTML) products delivered on compact disc (CD) for distribution to the user community. The use of MS-Office as a core application made the architecture universally accessible since MS-Office was pervasive throughout the users' community.

We held strongly to this universality of application as a design principle because it was extremely successful in cheaply introducing large organizations to the concepts of architectures and convincing at least that initial generation that an architecture was a detailed description predominantly of data, not just a bunch of "cartoons." Because a premium was placed on accessibility and usability, the product design was deliberately loosely coupled with architecture development activities e.g., information gathering, association, and analysis was the first phase, product release creation the second. Large quantities of data were collected, some of it as "placeholders" for future development, others for publication and use in the next release. This loose-coupling enabled great flexibility because at the time that a product was scheduled for release, a determination was made with the primary sponsor as to which information would be in the release; however, the database was not purged of important leads or other information we had not had the time to fully develop. Within this initial phase, we were also able to maintain a degree of loose-coupling that allowed parallel development using the Replication & Synchronization features in MS Access. Unfortunately, as the data grew in quantity and complexity, we ran into some data integrity problems with these features, e.g., data flagged to be overwritten wasn't, while data flagged to remain was changed.

Early in the development effort, rendering Architecture data for distribution was accomplished via HTML output from netViz, made "user friendly" by adding a navigation “front-end” in HTML that linked to integrated architecture products. netViz is an affordable tool solution that combines a powerful graphics capability with the ability to create “data enriched” content by connecting the graphical objects to a data source. However, we found significant limitations with netViz in the display of descriptive text and problems with refreshing data as changed in the source MS Access database. Additionally, the hierarchical structure of netViz HTML output was not intuitive, requiring users to maneuver up and down menu levels and multiple screens. We addressed some of these limitations by distributing a "viewer" tool that provided the netViz interface for those without the software. However, distributing a "viewer" tool shifted part of the workload (installation and use) to the user, thus limiting some the ease of use and accessibility to the information. In light of these limitations and changes imposed on the users, we abandoned netViz, finding it not suitable to our increasingly detailed and complex data. Our data remained in MS Access enabling other options in how it could be rendered for distribution to the user community.

By late November 2002, we changed our release design to a simpler structure of menus linking to product reports. So, instead of requiring the user to navigate across several screens to view information, basic menus provided quick links to database reports provided in Adobe Acrobat or MS Office formats for viewing or printing. Our navigation *front-end* was created using Macromedia Flash. This tool provided the capability to create a multimedia-enhanced navigation interface that did not depend on the HTML conversion solutions or limited presentation capabilities we encountered using netViz. The results were significant: the product worked better, the multimedia *front-end* helped "shore-up" the natural limitations of flat data and, overall, the CD product was better received. The combination of Macromedia Flash, MS Access, and Adobe Acrobat provided the most usable and user-friendly architecture product we had developed so far.

Still in all, the overall limitation of this approach specifically the use of MS Office generic tools as the core of the architecture was that it did not enable the display of activities and information exchanges in a way that highlighted their interdependencies or showed which activities occurred in series and which others in parallel. Being able to show dependencies and parallelism becomes crucial if the architecture is to fulfill its potential as an analytical tool, enabling the user to evaluate options, “what if” scenarios, etc. Also, the tools, by their sheer flexibility, lacked the rule framework to ensure that the resulting models not only depicted the operational reality, but also depicted it in a fairly universally recognizable way that would allow their reuse. Although we yet had to be tasked to apply the architectures, in any formal way, we believed that the limitations of our approach would have ultimately eroded the credibility of the overall product if there was a call for the formal, repeatable, and comparable application of the architecture.

Fall 2002 – Present

After the Air Staff's Deputy Chief of Staff for Warfighter Integration (HQ USAF/XI) began its operations in the fall of 2002, it established a series of architecture "integration" councils to - among other tasks - bring the different air and space operational communities to adopt architectures and govern their use. At the top of this mission-focused governance structure, XI and the Air Force CIO established an executive level council to coordinate the operation of the lower level councils. This executive level council established Popkin Software's System Architect, with its C4ISR Framework Overlay front end, as the "preferred" tool for architecture development.

Systems Architect (SA) is a formal modeling tool, automating the creation of models and associated information objects in a variety of modeling languages. To capture the growing C4ISR Framework market, Popkin developed a front-end application, the C4ISR Framework Overlay, that enables the user to use the underlying methodologies in the SA tool with additional Overlay-specific routines to build the Framework products (both operational and systems views). The formality embedded in the modeling language rules, the use of automated rule checkers, plus the addition of Overlay-specific automated routines held the great promise of quality, reusable products that showed parallelism and dependencies, while realizing high productivity in architecture production. For instance, the user would develop a comprehensive activity model (OV-5), and the Overlay would automatically generate the node connectivity description (OV-2) and information exchange matrix (OV-3) with relatively little user intervention. Although all data was stored in dBase III compatible formats, the user would not fully appreciate the relationship among the data objects outside of the diagrams. The information in SA definitions and diagrams could be output to hard copy and, in some cases, to fairly-universal comma-separated-value files or to hyper text markup language (HTML) for distribution and viewing with a web browser.

SA stored all data used in the models as "definitions" and related all definitions through a drawing interface. From the perspective of the operational views, the relevant data objects were activities; nodes; the activities' inputs, outputs, controls, and mechanisms (ICOM); information exchanges and, their attributes, processes, and rules. Nodes performed activities, and activities depended on one another by either generating or consuming information "exchanges" generated by other activities, described in the drawings as ICOM arrows between activity boxes. The designers of the SA Overlay, however, seem to have misinterpreted the Framework requirement for information exchanges. Under the Framework, information elements (e.g., Air Tasking Order, mission reports) are exchanged among activities depending on scenario-specific attributes. So, for example the same report may be exchanged under two separate scenarios, one routine, and the other emergency, each representing a different exchange requirement. What makes the exchange different in this case is not the information element (the report) but the conditions or attributes under which it is exchanged. By contrast, SA associated each information element to its attributes, "hardwiring" scenario and other situational attributes into each element. By extension, this meant that you could no longer have, for instance, one ATO broadcasted out because the conditions under which that ATO was sent out (the attributes) had to be exactly the same for all receivers; otherwise, the exchange would not be valid.

HQ USAF/XI tasked the AFC2ISRC, ESC, and AFCA with the development of the Command and Control (C2) Constellation Architecture operational, systems, and technical views, respectively, in three timeframes, all using the new tool. Later, we found that the tool had not been tested in projects of the scale we were dealing with, and, effectively, we had been tasked to develop a large-scale architecture while testing a new tool, all within a tight schedule.

The remainder of this section and our lessons learned are based strictly on the use of SA and it's Overlay (both terms used interchangeably throughout) to develop the operational views, specifically those requiring data and the depiction of the data in drawings.[6]

The tool (version 8.18, fully patched and updated) and a separate license manager were installed in every architect's desktop system. Instead of competing for licenses from a network server, this enabled off-network development. Generally we found this to be the best option because the tool's network performance and responsiveness were measured sometimes in tens of minutes (vice seconds).[7] As with past tools, CM was key, so very strict procedures were established that enabled maximum autonomy for each architect while protecting the integrity of the data.[8]

[6] Graphical products like the OV-1 and SV-1 can be developed in SA. However, SA is not a drawing or presentation tool so it allows for these products to be developed in another application (PowerPoint, Adobe Photoshop, et al.) and pasted into SA. Generally this was a routine, non-controversial interface.

[7] We were also aware of the difficulties that the AOC Weapons System Architecture Team had run into when attempting to manage SA information over a network.

[8] We found that SA provided very limited configuration management capabilities. A much-touted "check-in/check-out" tool that enabled selective extractions of information for off-net development, SA performed erratically and caused major tool crashes. We abandoned its use and also verified that it had been a major trouble spot for other teams.

The Constellation Architecture was based on the Monitor, Assess, Plan, and Execute paradigm. Accordingly, one architect each was charged with developing the model for each "branch" (M, A, P, and E) in a separate SA model, or encyclopedia, to allow maximum flexibility. Other architects assisted these branch architects with the contents of each branch and in reviewing the overall consistency of the resulting integrated model.

Each branch architect was responsible for modeling the activity sequences according to the agreed-upon context and rules, including inputs, outputs, and controls; for following all IDEF0 rules (the modeling language used in the tool for OV-5); and, for providing the CMgr with all the inputs, outputs, and controls to be shown at the very top of the model. The inputs, outputs, and controls (mechanisms were not modeled because they were to be designed by ESC), represented information elements (IE) on a one-for-one basis. ICOMs and IEs were reused throughout the model, reflecting the operational reality that most exchanges of information among activities are a finite set that vary mostly in their source and specific values at the time of the exchange. For instance, a number of activity sequences would produce mission reports (MISREPs). It would not have been faithful to the operational reality and the ultimate users of the architecture if the model made an artificial distinction between the MISREP filed by a sensing mission vice a bombing mission. When you do that, the number of ICOMs and IEs balloons artificially and you are unable to illustrate that common information is being exchanged by multiple activities. To cope with the tool's apparent design flaw of tying scenario specifics to the information elements, we decided not to complete the attribute-related information with each element.

At the call of the CMgr, modeling of the branch activity sequence diagrams (the "branch" OV-5s) would cease, and the branch models would be merged and integrated at the two top levels. Once this was completed, the architects would match previously defined operational nodes to the specific activities as the executors of those actions. Because the OV-5 was focused on what was being done, not who did it, it was quite possible that more than one actor performed a sequence. Just as it is reasonable for an operational commander to have more than one asset to interdict a target, it is equally reasonable to expect that a model of that operational reality depict more than one actor for each activity sequence where there's more than one asset that can perform it. Further, maintaining fidelity to the operational reality ensured a common mission context that enabled the architecture user to evaluate areas of duplication, commonality, or collaboration among those actors.

The plan was that once the node assignments had been completed for an activity sequence diagram at a particular timeframe, the branch architects would proceed to develop the next timeframe's activity sequence diagram, while the CMgr would concentrate on developing the node connectivity (OV-2) and information exchange matrix (OV-3) for that timeframe using the tool's automated routines. Here, unfortunately, the plan did not survive its contact with the vagaries of the tool. **Refer to Appendix A for a detailed chronology of the challenges encountered**, which we will summarize below.

Assigning nodes to activity sequences is a feature of the Overlay in SA but it is otherwise not found in IDEF0 (the tool's modeling language for OV-5s) because the language focuses on functions, not on who performs them. The Overlay matches nodes to activities as a way of creating the information exchanges between nodes, exchanges that are assigned to needlines for the node connectivity diagrams (OV-2) and the information exchange matrix (OV-3).

The Overlay's assignment of nodes was fairly basic: you assigned the node to the activity and the tool assumed that all ICOMs impacting that activity sequence were used by every single operational node/actor (CAOC, ASOC, SPACE AOC, et al)[9] chosen. For instance, if an activity had three inputs, two outputs, and two nodes/actors performing it, the tool created a total of 12 unique combinations of inputs, outputs, and nodes (3X2X2). The operational reality, however, is more nuanced. Although multiple nodes may perform the same sequence of activities, not all nodes will use all individual ICOMs to perform those activities. In short, the totality of all ICOMs in the activity sequence represented the superset of that used by all activities and associated nodes, not necessarily what is used by a particular node-activity combination within it. If the outputs of these 12 combinations were then the inputs to an additional 12 combinations of another activity, the Overlay would produce 144 (12X12) information exchange combinations, each of which had to be manually reviewed and validated. Further, if those 144 combinations were activities with parent activities that in turn had a node assigned as well, the node-activity-ICOM combinations resulting from ICOMs being "inherited" upward to the parent would be added to the total number of node-activity combinations at the parent, once again increasing the number of different combinations that had to be reviewed and validated manually. In short, the Overlay did not seem to have an automated feature to assign each node-activity to their relevant ICOMs as you developed them; and it did not regulate the applicability of IDEF0 rules (e.g., inheritance) to those features not supported by the language (e.g., node assignment and its consequences in the presence of IDEF0 inheritance).

We also observed two problematic tool behaviors when generating the information exchange requirements in those cases where the same information was broadcasted from one activity to multiple receiving activities; and, where the same ICOM/IE was reused throughout the model. In the case of broadcasted information (e.g., ATO, ACO, STO),[10] the tool would build an OV-3 that listed the correct generating activity, node, and IE, but repeated the receiving activities irrespective of the different receiving nodes. In the case of reused IEs, the behavior was more erratic. The tool (SA & its Overlay) would

[9] CAOC=Combined Air Operations Center, ASOC=Air Support Operations Center, SPACE AOC= Air Force Space Command Aerospace Operations Center
[10] ATO=Air Tasking Order, ACO=Airspace Control Order, STO=Space Tasking Order

create information exchanges that were never shown or documented in the drawings, often quintuplicating the number of exchanges. It would also create needline relationships in the OV-2 that were not supported by the OV-5. Finally, it would also document all the exchanges in the OV-3 without regard to the specific sending and receiving activity pair, repeating every single sending and receiving activity that exchanged a particular element, regardless of sending or receiving node.

This is by no means a complete catalog of the tool or its limitations, just the behaviors we were able to discover and work-around. **Appendix A** details our efforts to work out solutions with Popkin and with the designer of the Overlay. In general, we found that the Overlay treated information exchange requirements (IER) not as a unique combination of sending node and activity, receiving node and activity, and information element and attributes; but as a set of combinations built "on the fly" by the tool through the association of all activities inputting or outputting common information elements, with the nodes added as associated data to the activities, not as potential determinants of the uniqueness of the IER.

As the Constellation model grew, the sheer size and complexity of the model combined with the tool's shortcomings nearly became showstoppers. This was largely because the output of the tool could not be trusted, and there was so much data to cull through and validate that all the productivity gains promised by the tool were lost in the manual workarounds. What's more, the problems we noted were widespread in other architectures published using this tool. We also noted that the tool responded erratically when trying to develop an OV-5 Node Tree diagram that approaches its maximum drawing area. The tool displays no indication of drawing size and allows the user to work and save large diagrams. However, if somehow this maximum size has been exceeded, the tool truncates the "node tree," only displaying the top-level, context activity, and there is no way of recovering the remaining activity tree. Reports to Popkin revealed this to be a known bug for which the only solution is to purchase the upgraded version of SA.

XI responded to some of these problems by developing an "extension" (add-on) that attempted to overcome the limitations of the Overlay by establishing a new information object called IEX that instantiated the correct interpretation of information exchange requirements, i.e., a unique combination of sending node, sending activity, receiving node, receiving activity, information element, and situational attributes (criticality, timeliness, etc.), creating an unambiguous OV-3 that nevertheless had to be validated because, – as said before, – not all nodes assigned to an activity used all ICOMs. The extension had the additional advantage of a set of reports that facilitated validating the information. It also produced the information in a format that could be imported into a database and analyzed with the ease of MS Access.

The extension could not make up for the limitations of the original tool. Though the extension tried to create unique exchanges, several of the exchanges reported were not supported by the OV-5 (i.e., the tool got "creative") largely because the extension could not address the internal workings of SA and its Overlay where the relationships were built. We also found the extension created or discovered additional limitations.

First, the extension seems to have "broken" the relationship between certain operational and system views, limiting the tool's utility for integrated development of the architectures. One of the touted advantages of using SA and its Overlay was that the systems views could be built and associated with the operational view objects within the same tool, ensuring integration and data integrity. The extension, however, did away with the Overlay's incorrect design of the IE, creating instead a new object called an IEX. IEXs did not have an equivalent in the system view portions within the tool. This meant that system information like that in the SV-6 that needed to be associated with an IE to show its operational relevance no longer had a linking object in the operational views.

Second, the extension may have discovered or added another inherent limitation in terms of the overall amount of data that any SA/Overlay entity can store. The underlying database in SA and, by implication, its Overlay does not seem to store all information objects in a way that will facilitate their minute recall. Each individual node is a separate record, but each activity performed by that node is not. Instead, the activities are shown as a stack of textual values assigned *en masse* to a particular node, unable to be individually queried. Each operational activity may be individually queried in its own table, but not the activity-node combination. In the case of activity values, each activity is listed separately, but all nodes performing it are textual values, literally stacked within the same field. The same situation is repeated with needlines, IEXs, and so forth. The use of this technique of stacking associated values as text strings instead of as separate records seems to run against an overall limit on the number of characters that can be stored in any one text field (about 32KB). Once you try to exceed this limit, the tool is unable to handle any more information for that particular definition. With IEXs being the unique combination of five values (sending node and activity, receiving node and activity, and information element), the total number of IEXs that could be supported without exceeding this 32KB character limit hovers around 400. Although that number seems large, for a model as large as the Constellation, this was modest at best and left hardly any room for growth. Without all IEXs in the model, the OV-2 cannot be fully developed and the OV-3 can only be developed and maintained manually. With a model the size of the Constellation (information exchange requirements numbering in the thousands), these are neither efficient nor effective options.

The extension also exhibited some erratic behaviors in the pairing of nodes to activities. After nodes had been assigned to activities, the tool reported a different set of activities per node than nodes per activities (instead of one being the mirror of the other). To date, this behavior remains unexplained. Originally we thought we had been able to work around this problem by only allowing one individual to assign all nodes to activities and manually verifying the results afterwards. Once again, however, the workaround further drained productivity from the overall development effort, eroding what little schedule was available and, trading off the quality of the product for the schedule. Recently, however, we have discovered that even when little or no work has been done recently on the activity-to-node assignments, the tool somehow loses activity-to-node pairings, apparently randomly.

To address some of these behaviors and limitations, the extension developer recommended limiting node assignments to the "leaf node" within the OV-5 as a way of reducing the number of information exchange requirements that had to be validated and published in the OV-2 and OV-3. "Leaf node" is an IDEF0 artifact that refers to an activity that cannot be decomposed any further. However, "leaf node" assignments, although indeed reducing the number of information exchange requirements documented in OV-2s and OV-3s, lose all semantic context because an activity that requires no further decomposition ("leaf node") is not generally descriptive enough to make the information exchange requirement intelligible. For instance, if a parent activity is made of three subactivities (the "leaf node" activities) that are not decomposed further, it is reasonable to expect that each subactivity's content will only address one component of the parent activity. Standing alone, each of these components would seem like disconnected threads, missing a context. The OV-3 reports exchanges between "stand-alone" activities; the OV-2 summarizes these exchanges into needlines. Hence, if the activity is but a component of a larger one, a node assigned to that activity under the "leaf node" principle would create information exchange requirements that, - in the aggregate, - would make little operational or semantic sense. Assigning operational nodes only to "leaf node" activities is analogous to solving a sensor's processing limitations by deliberately curtailing its field of view and range. In the presence of limited input, the sensor's processor is able to cope, the processing problem is mitigated, but so is the military utility of the sensor itself. Similarly, with the architecture, the "leaf node" assignment does limit the number of invalid information exchange requirements, but in the process limits the utility of the architecture itself.

Lesson #T-1: Research tools' reputation with projects of your size and complexity before spending funds or schedule on them.

No *single* tool is either perfect or fully able to comply with the C4ISR Framework largely because many of the architecture products in the Framework cut across modeling language lines and scopes. Modeling also tends to be limited in scope and scale, so it will not be easy to find strong analogs to any of the defense domains.

Metrics & Mitigation: Ensure that you are empowered to engage the tool vendors directly. Ask for references that you can engage. Shy away from vendors who push more their consulting services than their tool. Also, shy away from vendors who are unwilling to share valued references. If you have the time and the information, seek references in the defense environment. Absent a defense or aerospace customer, seek customer references in the supply and financial industries (large-scale exemplars) as both tend to have the diversity, scale, and sensitivity requirements that mimic the defense domain. Take the time to interview those customers. Think broadly! You may have to apply what you learn by analogy to your architecture's purpose and scope. **Do not think in terms of one tool for all your needs – none exists.** Consider, instead a **suite of tools** that will enable you to complete the project quickly and cost-effectively.

Lesson #T-2: Choose a bias in your modeling and stick with it.

As mentioned before, no single modeling language will satisfy the requirements of the C4ISR Framework. In fact, SA directs the C4ISR architecture developer to use "all methodologies" (see tool browser). You are, in fact, extending the language rules to serve the particular purpose of creating a Framework product. Once you leave the bounds of the modeling language rules, there are few rules to serve as a common reference; hence, you need to make some of your own.

Metrics & Mitigation: We strongly recommend that your first rule should be to set a bias in your modeling and stick to it, similar to the "bias" you create when you choose a certain a scope. You will run into situations where the "rules" of the modeling language or methodology conflict with what you are trying to model (see the previous "leaf node" discussion). In such cases, you will need to decide what is more important, those modeling rules or the scope of your architecture. Choose carefully and consistently because, unfortunately, you will find the architecture judged not only on its operational fidelity and value, but also on your adherence to the modeling language and methodology, especially by those who do not have any inkling of the operational environment you are modeling but must nevertheless express an opinion on the quality of the model. In the case of the C2 Constellation, we chose to maintain close fidelity to the operational "reality" we were trying to model, and to do so often in spite of the limits of the modeling language. In short, we were willing to trade the endorsement of modeling "experts" for that of the operator who may ultimately be the architecture user.

Lesson #T-3: Get ready for the "experts" and their "assistance."

Once you begin developing the architecture, you will be offered more "advice" and "assistance" than the President in picking a Cabinet. With "advice" and "assistance" comes influence, which may add to or erode from your architecture's purpose and scope. Deciding whether the assistance is useful takes time, time your schedule may not permit. The situation is very similar to the quandary of software reuse: advocates believe it actually saves time to use someone else's developed and tested code; detractors, on the other hand, assert that the time saved in coding is more than outweighed by the time spent finding out if the "borrowed" code can be reused.

Metrics & Mitigation: Be ready and receptive to advice and assistance; however, ask for the **help** to be matched by a willingness and ability to **work** to implement the advice. Ensure that your communications plan promoting the architecture highlights your openness and willingness to listen to expert advice, and your offer of a partnership with those experts to work together on – not merely pontificate about – the architecture and its development. Beware of those who try to reinterpret (change) your architecture's purpose and scope to fit their advice.

Lesson #T-4: Apply strong configuration management.

Just as strong CM is necessary to maintain the integrity of the data making the architecture, CM is also essential to properly applying any tools used to develop the architecture. CM is necessary to ensure that all architects maintain the same version of the tool, to exchange any data in the same and controlled manner (to avoid overwrites) and to maintain a minute watch on the schedule and workload.

Metrics & Mitigation: See lesson #**P-7**. Also, ensure that the person you designate to be your CMgr:

- Is formally trained on CM.
- Obtains any advanced training on the tool you are using before anyone else in your team so he or she can guide the team in the most efficient use of the tool.
- Becomes the tester of any new version or patches on the tool prior to anyone else in the team.
- Maintains an active exchange and, – if possible, – personal contact with the tool's vendor, its developers, and other project CMgrs, contacts that may help the architecture team avoid the pitfalls others discover in their projects.

Lesson #T-5: Few, if any, architecture development tools consider or contain security features. Seek tools and vendors that implement security. If the vendor does not directly implement security as a part of the tool, determine the compatibility of third-party security tools, such as encryption engines and access control tools. Barring that, develop a set of security practices for tool use that protect the resulting data.

Out of our contacts with tool vendors an interesting pattern emerged: – none of the tools had any security features and the toolmakers considered security an external, environmental requirement independent of their tool. We believe this is bound to change in the future as the business sector continues to realize the importance of protecting its data from unauthorized access if nothing else from a business intelligence perspective. The national security community cannot wait, especially because of OMB policy and the statutory requirements of the

FISMA. In the meantime, however, we have found SA is not compatible with third-party access control tools, and is also incompatible with *any* third-party encryption engines. We also suspect that the version of SA we used (V8.18) will not allow data to be encrypted in-residence and is not compatible with "encrypt-decrypt on the fly" schemes that would help shore up any access weaknesses.

Metrics & Mitigation: As an integral part of the architecture project planning, develop a set of project-specific security policies and practices that include tool use and protection of the resulting data. For instance, control access to the architecture information by knowing who is seeking access and deliberately granting access to the information. Ensure that all those granted access understand the security rules you have established for data development, release, copying, and so on. Other policies should control distribution, control, and audit of the information. (See **Appendix B**)

Lesson #T-6: Generally, architecture tool outputs have very poor presentation quality and lack the information to place the depictions within a larger context understandable by the uninitiated. Therefore, plan on having more than one tool to *develop* and *present* your architecture. Also, plan on supplementing the architecture outputs with a "front-end" presentation that introduces the architecture.

Ideally sponsors and users should not be concerned with the internals of the architecture, its detailed views and data. Their focus should be on application of the architecture and its results within the architecture's purpose. Unfortunately architecture development represents such an investment that no sponsor is willing to limit itself to results and not "look under the hood" (see lesson #**P-10**). Generally, when a sponsor is exposed to the details of architecting and its outputs –what some call "sausage making"– they are repelled by the need to learn a new language and symbology, all while evaluating the substance of what is being modeled. Hence, the quandary: sponsors wanting to look and understand the details, but refusing to be bothered with learning the new language that enables that understanding.

Metrics & Mitigation: Plan on having a variety of tools for architecture development – at the very least two – one to develop, the other to present the architecture. Also, develop a strategy to bridge the conflicting demands of the sponsor, i.e., wanting to look into the architecture and immediately understanding its contents without learning the details of the language. Consider developing a front-end program to unify the presentation of your architecture models. This front-end program supplements terse models with textual explanations and builds an understandable context that enables the user or sponsor to gain an appreciation for what the models depict and its complexity. The front end could also be useful to place the architecture within the larger context of other efforts to enhance combat capabilities as well as a means of introducing the architecture with a presentation from the sponsor. Regardless of the strategy chosen, however, count on the fact that the innards of an architecture will look unintelligible to your sponsor unless you do something extra to place them in context. And, often, that "extra" requires a separate set of tools because the architecture development tools are extremely poor at presenting the information.

Appendices:

A. Chronology of Problems with SA

B. Protection of Enterprise Architecture Tools and Dat