SYSTEMS ENGINEERING AND ARCHITECTURE: LESSONS FROM THE F-22 TRAINER PROGRAM

Tony DalSasso
F-22 System Program Office
Wright-Patterson AFB, Ohio

George R. Rovny
Lockheed Fort Worth Company
Fort Worth, Texas

ABSTRACT

The successful implementation of a training simulation system requires that engineering constraints be communicated from the Systems Engineer to the designers in an unambiguous manner. This paper proposes that an architectural framework can be developed, providing the Systems Engineer with a tool to aid in this communication.

The paper documents the F-22 Pilot Training System team's observation that the term "architecture" has no universally-accepted definition. It chronicles the process used to resolve this problem, eliminating the confusion concerning both the terminology and the process of developing an architecture. It describes a hierarchy of definitions, allowing consensus to be reached among a group with widely varying experience levels, without creating a "least common denominator" definition. It explains the term by means of analogy - what "architecture" means to a builder, and how this maps into the trainer engineering context.

Emphasis is given to how an architecture needs to address hardware and software as a system. A preferred process for creating the architecture and managing the subsequent development of the product, using architecture as a systems engineering tool, is discussed. The paper describes the "litmus test" developed to determine whether an approach constitutes an architecture and describes the attributes of an architecture that allow its relative quality to be measured. It observes that most of what is touted as architecture doesn't pass the "litmus test," and why it does not.

Believing that the F-22 program is a microcosm of a trend throughout industry, the paper suggests that lessons learned by the F-22 can be effectively applied elsewhere. It discusses why this subject is so vital to a simulation development effort, and concludes with some thoughts on how a properly developed architecture can provide significant advantages to a system integrator.

BIOGRAPHY

Tony DalSasso is employed by the U.S. Air Force, Air Force Materiel Command, Aeronautical Systems Center, F-22 System Program Office, Wright-Patterson AFB, Ohio. He is presently serving as Lead Engineer for the F-22 Pilot Training System. He has previously served as Lead Engineer on the Standard Simulator Data Base Program (Project 2851), and participated in various training simulator acquisition programs, including the B-2 Aircrew Training Devices and Special Operations Forces Aircrew Training System, specializing in the areas of visual simulation and terrain data bases. He holds a BS in Computer Engineering from Lehigh University.

George Rovny is a Lead Engineer at the Lockheed Fort Worth Company, formerly the Fort Worth Division of General Dynamics. Mr. Rovny has been with Lockheed since 1984, holding various engineering positions in avionics integration and test, design of electronic support systems, simulation, and training systems. Mr. Rovny has been involved with the F-22 Training System since its Demonstration/Validation phase and currently has lead technical responsibility for the Lockheed contribution to the Pilot Training System. Mr. Rovny holds a Master's Degree in Electrical Engineering from the Pennsylvania State University.

SYSTEMS ENGINEERING AND ARCHITECTURE: LESSONS FROM THE F-22 TRAINER PROGRAM

Tony DalSasso
F-22 System Program Office
Wright-Patterson AFB, Ohio

George R. Rovny
Lockheed Fort Worth Company
Fort Worth, Texas

INTRODUCTION

As part of the Air Force's implementation of Integrated Weapon System Management, the F-22 Advanced Tactical Fighter program includes the development and deployment of *a complete training system for aircrew and maintenance* personnel. The development of the training system is proceeding concurrently with that of the aircraft, allowing for the early exchange of design information among the system of Integrated Product Teams (IPTs) developing the F-22. Early involvement allows training system requirements to be considered during the development of the air vehicle, supporting the incorporation of features in developmental *software which would facilitate the future reuse of this code in* the trainers; it also gives the Training System IPT detailed knowledge of the air vehicle design process.

Through the progress of the program, it has become apparent that the integration of reuse items from the aircraft will be difficult, if not impossible, in the absence of a defined hardware/software framework for the trainer. This is *particularly true of the Pilot Training System (PTS)*, which requires a significant amount of functionality beyond that found in the engineering labs. There are enough differences among the anticipated operational configurations of the aircraft, avionics engineering laboratories, and training system to cause concern that the initial concept of using the laboratory simulation architecture in the PTS will provide a substandard solution at excessive lifecycle cost.

With this philosophy, the F-22 program team set out to lay the groundwork for the establishment of a training system architecture. The first hurdle to be overcome was the lack of a common understanding of the terminology: "exactly what do you mean when you say 'architecture'?" This drove the team to develop a working definition, which turned out to be an iterative learning process, involving team members of varying disciplines. The definition of architecture, and its relationship to the systems engineering process, will form the basis of future work on the F-22 Training System.

ARCHITECTURE AND SYSTEMS ENGINEERING

What is Architecture?

Prior to relating the experiences of the F-22 program in developing a working definition of architecture, it is first necessary to understand the concept. Early on, it was discovered that considerable ambiguity surrounded the term. It was found that the definition of the word "architecture" seems to be quite controversial, probably not because it is an especially difficult concept to grasp, but more likely because *nearly everyone thinks they understand the term already.* Unfortunately, there is rarely consensus among individual interpretations; this leads to a breakdown in communication and creates controversy. The F-22 program solved this problem by establishing a common interpretation, as discussed herein.

General Definition

Webster's Ninth New Collegiate Dictionary includes five definitions for the word *Architecture*. The one which is most applicable in the systems engineering context is probably definition 2b, "a unifying or coherent form or structure." Before examining the engineering implications of this term, let us first understand the more familiar use of this definition, associated with the external appearance of buildings.

Construction Analogy

Everyone seems to understand the term "architecture" when applied in this context; so it is a good starting point for our discussion. Architecture is used to describe certain properties associated with a physical structure. Indeed, it is such a fundamental property that practitioners of building design are referred to as "architects"[1].

The architecture of a building might be described as how its individual components are arranged to give a characteristic appearance to the structure as a whole. In this sense, architecture is used to establish such unifying properties as proportion, materials, and ornamentation. Using just these key attributes, even the layman can differentiate among Greek, Colonial, Spanish, Gothic, Modern, and the various other architectural styles.

It is important to understand that architecture has little to do

[1] Calling this person an "architect" can actually lead to a misunderstanding of the term, insofar as very few "architects" actually develop an architecture. Under this definition, the building architect is a designer, applying architectural principles rather than developing them.

with functionality. Although certain architectural styles are often associated with buildings used in a particular way - a Gothic church, for example - it is not an inherent property of the architecture that makes this so; rather, the application of a given architecture to some construction problem is at the discretion of the building designer. The designer does not select an architecture based upon specific functional requirements, but rather on the basis of some unifying aesthetic theme which he desires the resulting structure to possess. Any building of sufficient size can be used as a church; nothing about its functionality requires that it be of Gothic design, and in fact the majority of churches are not built in that style. Conversely, structures of many different functions can share a common style. One could build a Gothic home, office, or factory, if desired.

Architecture vs. Design. *"Architecture" should not be* confused with "design" - they occur at different phases in the development of a structure. Architecture is effectively a constraint imposed upon the design process. It is a set of "ground rules" which guide the development of a design. Thus, the design of a particular building does not constitute the *development* of an architecture; it is, in fact, the *application* of an architecture. Architecture builds a repository of information for later use by individual designers.

Philosophically, one could think of the development of an architecture as a creative, "right-brain" activity, whereas the development of a design is an analytic, "left-brain" task. Architecture establishes the set of components and the rules for connecting them; design applies these materials and tools to create a functional entity. It is conceptually easy to see how the former is a creative process, while the latter is more or less mechanical. In general, it's easier to follow rules than it is to make them.

There are many benefits of having an architecture as a precursor to a design. For one, relative to methodology, *creativity is expensive. Risks are always greater when one* starts with a clean slate, rather than adhering to established techniques. Architecture allows the lessons of the past to be applied to the problem being investigated. Architecture allows a design problem to be solved but once.

Relationship to Engineering. But how does this relate to "architecture" as defined in the context of engineering? Consider the concept that architecture creates a set of components necessary to solve an entire class of problems, and that design applies these parts to create the solution to a specific problem. In construction, the facility designer takes architectural elements and arranges them to configure a functional structure. Engineering is equivalent, in that the designer starts with these fundamental "building blocks," and applies them to the problem space. In hardware, this equates to an engineer designing a board around a standard chipset, and interfacing to a standard bus. The preponderance of low-cost PC hardware attests to the validity of this process.

When one examines the design process, one can see that the design of a building is not significantly different than the engineering process used to build a trainer, or any other engineered product. The Systems Engineer (SE) - or system architect[2] - creates processes and templates; the design engineer, or implementer, uses them to create the product. The implementation is divorced from the creation of the rules which guide it. Just as the building designer is relieved of the responsibility of creating the system of proportions, materials, and ornamentation which define the style of his structure, the design engineer is free to concentrate on the functional aspects of his product, with the knowledge that the architecture, properly applied, will yield a unified result.

Building architecture does not unduly constrain the designer in functional terms - designing a church in the Gothic style *does not have any impact on its utility as a church*, for example. Similarly, the application of an engineering architecture does not limit the designer's control over the functional capabilities of the device. But it allows the designer to achieve a predictably unified result, while avoiding the difficulty associated with starting the design from scratch. In short, architecture makes the designer's job easier, while it assures the quality of the product.

Cost Management

If architecture were to be distilled down to its very essence, its sole *raison d'être* would be cost management. Architecture gives the SE the ability to divide the problem into a set of manageable sub-problems, the implementation of which can be delegated to individual design engineers. It gives him visibility into the development process, and control over it. When it becomes time to integrate the components into a *unified product, he can be confident that they will integrate in* the least amount of time. During the development phase of a system, all of these activities take time, which equates to dollars. Minimizing the time spent on any of these functions translates into savings to the program. Further, any "unknown" can impart a risk to the program, and the potential that a greater amount of time will be spent than originally anticipated. The architecture allows risk to be controlled, which results in the control of time and, in turn, control of the cost of the system.

The cost control aspects of architecture are not limited to the development phase, but can extend well into the lifecycle of the product. By enhancing the understandability of the design, the maintenance of the system can be performed by the fewest number of personnel; this will hold support costs down. With a consistent architecture, changes to the system will be easier to make, minimizing the amount of time spent in their analysis and implementation. Again, this will result in a cost saving over the lifecycle.

---

[2] In this case, the title of "architect" is applied in a manner consistent with our chosen definition.

Through its establishment of standards for developers, architecture can be a much more effective means of controlling the cost than less formal methods, such as handing off sets of requirements to developers who then proceed to design their subsystems independently. The latter case gives the SE little or no control over the quality of the individual subsystems, which will likely succumb to the creative whims of their developers. Architecture controls cost through the selective management of creativity. Designers at the lower levels are empowered to apply creativity only within their specific problem domains, and within the constraints imposed by the architecture and their specific standards.

What is Systems Engineering?

Systems Engineering is defined by the Defense Systems Management College [DSMC 90] as "the management *function which controls the total system development effort* for the purpose of achieving an optimum balance of all system elements. It is a process which transforms an operational need into a description of system parameters and integrates those parameters to optimize the overall system effectiveness."

Systems Engineering is a process by which a solution framework is created from some problem statement, the solution is communicated to designers, and the realization of the solution by designers is governed. The SE must understand the nature of both the Engineering/Manufacturing Development (EMD) phase and lifecycle constraints, the classes of platform technology available (including hardware, compilers, and toolsets), and must have a sense of the standards in use by the designers. The latter is important as the SE must strike a balance in the solution between the degree of specificity (level of constraint) and the degree of latitude afforded the designers. The balance will depend on the degree of confidence the SE has in the accepted level of standards in the design community.

Expanding on lifecycle constraints, in the world of Pilot Training devices, it is typical that the lifecycle will be characterized as being long and subject to numerous requirements changes, coming from changing user needs as well as changes to the air vehicle. Changes will always have to be accommodated very rapidly. The SE must devise solutions that accommodate frequent and rapid modification.

Process. The SE can ensure robust solutions if he bases the structure of the system on the structure of the problem. That is, he will define and bound a "problem space", and extract from it pertinent features, creating an abstract representation of the problem definition that will yield a set of patterns. The conceptual framework for the solution should be based on the structure of these patterns, which can be made to incorporate the desired attributes of the system. The design decisions must be conveyed to the developers; the SE uses the architecture as the primary vehicle to communicate the design rules. The SE monitors and guides the development process, through integration and test, enforcing the architecture and arbitrating conflicts as they arise. He will search for emerging problem areas and attempt to address them long before significant resources are expended.

There is a troubling tendency on a large program to embrace the prevailing standards and methodologies of the day, and the massive organizations that are created to enforce them, as the *de facto* Systems Engineering process. While programs are often required to utilize one methodology or another (as F-22 is), it is useful to recognize that these methodologies are often attempts to regain control over projects that have scaled beyond the limited utility of methods that worked for small systems, and that the methodologies represent more rigid, disciplined application of techniques that worked for small systems. The message here is simply that blind trust of methodologies to produce complex systems will *result in disappointing failure. Methodologies don't replace* Systems Engineering.

The Training System Development Challenge. When applied to the problems of Training Systems, the Systems Engineering process must deal with constant uncertainty and persistent information voids. During EMD, requirements are likely to remain unstable until late in development, as the mission and the air vehicle are in a constant state of flux. And the validation process at the end of EMD is not as straightforward for training devices as it is for many other products (especially in the case of F-22, where the contractor is to validate the creation of an effective "Training System", not a device that does x, y, and z). Training System development has been described using the "Q-Tip model," i.e. the longest part of the development process, in the middle, is fairly firm and easy to grasp, but there is fuzz at both ends.

These factors create a fascinating environment for the Training Device Systems Engineer. He faces a nebulous problem on the left, tempted to view the Instructional System Development (ISD) process as esoteric mysticism, while on the right he is constantly warned by the pragmatic designers that their primary metric used to divert the wrath of management is one called "requirements volatility".

At this point it is useful to recall the postulate observed earlier, that "architecture has little to do with functionality". Neither is its basic form affected much by fidelity requirements. So the seemingly impossible situation just described provides no excuse for the SE to despair. The SE must forge strong ties with the instructional analysts, to extract from them requirements that are meaningful in an engineering context, while alerting them to capabilities of the technology as well as the air vehicle that they may not have been privy to. He must be prepared to provide rapid analysis when asked by instructional analysts what the cost of various capabilities will be in various device types. Solid Systems Engineering can magnify the value of ISD.

The SE must forge a solution that easily accommodates

frequent change, minimizing both the cost and schedule impact of any change. He should provide to the designers a framework whose very structure embodies attributes amenable to change, isolating designers from the volatility inherent in Training Devices.

A Training Device SE thus bridges the gap between the instructional analysts and the developers, ensuring that function and fidelity are manifest within a robust, coherent, highly maintainable system by guiding the development process according to the architectural solution.

Role of Architecture

The architecture of a system is by far the most important tool available to a SE. With it he can predict and manage cost, comprehend system behavior, govern the creativity of developers, and communicate with the multiple entities who have a stake in the project.

Cost Management. Architectures have become increasingly important to SEs as projects have become more costly to develop and more costly to maintain. The SE will utilize the architecture to predict and manage the cost of a large project that requires a large staff. This implies that the architecture will express the structure of a system at a sufficiently fine level of granularity that the cost of individual pieces can be reasonably estimated. An argument exists that the smaller the partitions are, the less subjective the cost projection will be.

Comprehend System Behavior. It has been stated that the purpose of architecture is to give the SE intellectual control over a system. The SE will use the architecture to predict system behavior in a variety of scenarios, and will utilize this understanding to render decisions for developers as ambiguities are discovered. As system characteristics emerge during development that call into question the validity of assumptions made to develop the architecture, the SE will modify it as needed to capture that new understanding.

It has proven advantageous to understandability for the partitioning of a training simulation to exhibit a close mapping to the structure of the article being simulated for training. The SE will search for patterns in the air vehicle to serve as the basis for the partitioning of the training devices.

The architecture provides the basis for thought experiments, whereby the system can be stressed by numerous scenarios, including requirements changes, to see how it behaves. If necessary, the SE may code the architecture into some sort of analytical tool, plugging in known performance characteristics for some components, parametric estimates for others, and educated guesses for yet others. This provides a baseline for comprehension. The performance data is only as precise as the estimates, but at least the areas with data voids are illuminated and can be targeted for update as better information is found.

Risk Management. A SE who fully comprehends the characteristics of a system can pinpoint likely risk areas and take steps to mitigate the risk. It might be said that the ultimate success of any engineering development effort is directly related to the successful control of risk throughout its development. If the technical, cost, and schedule risk factors associated with each component of the system can be quantified, risk can be managed. Resources can be focused on those aspects of the development which harbor the greatest potential for creating problems later, and preventative measures can be taken to avert these problems. This analysis and decision-making process forms the essence of Systems Engineering.

Isolation of Creativity. It has been long understood that efficient development by large teams requires effective separation of labor. As idyllic as it sounds to have everyone on a team in on every decision, and to grow a culture of people equally good at everything, it is not conducive to competitive product development. A team should have specialists in the various fields that contribute to the product, and the SE must utilize the architecture to harness the diverse capabilities on the team. Just as in the case of the architect of a Gothic cathedral, the SE creates bounded "pools of creativity", within which a specialist is free to ply his trade to the best of his ability, without the fear of impinging negatively on someone else's area. The SE has made the decisions that affect the system in a global sense, and by so doing has empowered the specialist to develop focused solutions. As the cathedral architect has ensured that diverse pieces will come together to create the desired ambiance, so has the SE used the system architecture to guarantee that desired interfaces and overall style are attained.

Communication: One of the authors was asked by a manager long ago what the key attributes of a good SE would be. This particular author, who shall not be named but whose initials are not A. D., floundered about with discussions of requirements allocations and specmanship, etc. ad nauseam, until the manager put him out of his misery with the single word "communication". As time goes on, the manager's conclusion has been validated countless times. But a project cannot be left to the personality type of the Systems Engineer. The manager did not mean that a SE must be gregarious, vocal, nor even clear in his expression. The architecture holds the key to successful communication.

The SE can create simple tools to capture the important aspects of every facet of the architecture, and can require their use by the developers. Examples include component code templates, documentation templates, and guidebooks. If the tools are straightforward and their use contributes to productivity (especially if they handle some of the mundane aspects of design), they will be embraced by developers, part of the communication challenge will be solved, and the enforcement aspect of the SE's job will be easier.

Communication continues between the SEs and the developers throughout the development process. The architecture forms the guide for the SE to devise meaningful metrics that measure progress, indicate adherence to design rules, and illuminate emerging indications of flaws in the architecture.

Architecture also provides the best mode of communication with the maintainers of the system. A maintainer is much more likely to respond rapidly to change during the life cycle if he understands the behavior of the system, where the change must be implemented, and the ramifications of the new implementation on the entire system.

F-22 PROGRAM EXPERIENCE

By now, the reader should understand the fundamental concept of architecture, and its relationship to systems engineering. The following discusses how this concept is being used within the F-22 training system program.

Training System PAT

A Process Action Team (PAT) was formed to develop an overall strategy for the development of an architecture for the training system as a whole. One of the earliest challenges addressed by the PAT was the synthesis of a definition of the term "architecture." An effort was made to develop a standard definition for the term, facilitating communication among members of the PAT. Care was taken to avoid coming up with a "least common denominator" definition, with the argument that too loose a definition would be meaningless as a communication tool.

The architecture PAT consisted of members with widely varying backgrounds. Some members were managers, while others were engineers. Most of the group had maintenance trainer credentials, with fewer having a PTS perspective. Expertise varied from hardware to software to courseware. In the initial discussions, it was difficult to achieve a common level of understanding of the concept of architecture; each PAT member had his own interpretation, based upon his individual background. It became clear that a common definition of architecture would need to be found, otherwise the task of developing an architecture would be impossible.

A strawman definition was proposed by Lockheed-Fort Worth Company (LFWC.) LFWC suggested that the PAT use the working definition of "a framework that conveys engineering decisions regarding the partitioning of a system and establishes the coordination strategy that constrains how the partitions communicate, are controlled, are observed, and are synchronized." The Lockheed definition was the subject of much debate, insofar as not every member of the group was comfortable with the concepts it embodied. It was found to have weak areas, such as the fact that it was recognized as a good definition for software[3], but fell short for hardware, in that it did not address physical constraints or electrical characteristics. It was determined that the definition of "architecture" could not be limited to the software, but must apply to the system as a whole. Also, owing to their different backgrounds, each member had an individual "comfort zone," outside of which he could not concur with the definition. A number of members argued for a more general definition, whereas others favored a more explicit one. It appeared that it would be impossible to settle on common ground.

At that point, it was proposed that a set of options be identified, from which a single definition would be chosen. The approach taken was to generate a series of upwardly-compatible definitions, and step through them one by one until a common "comfort zone" could be found. The intent was to start with a minimalist definition which was acceptable to everyone, and add detail incrementally. When the detail began to fall outside the "comfort zone" of an individual member, an explanation of the added concept might be sufficient to expand that member's acceptability boundary. The following set of definitions was created and discussed:

1. "A philosophy about partitioning a system." An architecture can be considered to portray a view of how a system would be broken up to implement a specific philosophy. For example, a conscious intent to fit the software into a specific hardware configuration will force it to be partitioned in a certain manner, regardless of other factors.

2. "A *framework that depicts* the partitioning of a system." Rather than simply stating that the system will be partitioned to implement a certain philosophy, it shows how the system is actually partitioned to reflect this philosophy.

3. "A framework that depicts the partitioning of a system, *and describes the interfaces among partitions.*" This definition adds a level of detail to the preceding one, describing the connectivity among partitions.

4. "A framework that depicts the partitioning of a system, *and establishes* interface *relationships and coordination* among partitions *to a specified level.*" This introduces dynamic considerations, through the concept of coordination. This is an important aspect of architectures in general, and particularly so in real-time applications. In addition, this definition adds the idea that a level of detail constraint will be imposed prior to the partitioning.

5. "A framework that *conveys engineering decisions regarding* the partitioning of a system *in accordance with predefined system requirements*, and establishes interface relationships and a coordination *strategy* among partitions to a specified level." This establishes a relationship between

[3] This definition was, in fact, adapted from the definition of the structural model defined by the SEI's Structural Modeling Guidebook.

architecture and engineering; the responsiveness of the architecture to system requirements; and the idea of a coordination strategy. The thought here is that it is no longer sufficient to define just the coordination among the known partitions, but to establish an open-ended approach that can support future growth.

6. "A framework that conveys engineering decisions *regarding the logical and physical partitioning of a system in* accordance with predefined system requirements, which establishes *hardware and software* interface relationships among partitions to a specified level, and defines a coordination strategy *that constrains how the partitions communicate, are controlled, are observed, and are synchronized.*" This definition adds clarity, and the applicability of the architecture to the real-time domain begins to emerge.

7. "A framework that conveys engineering decisions regarding the logical and physical partitioning of a system in accordance with predefined system requirements, *and establishes and/or invokes standards for adherence by the designers of individual partitions;* it establishes hardware and software interface relationships among partitions to a specified level, and defines a coordination strategy that constrains how the partitions communicate, are controlled, are observed, and are synchronized." The last version adds *the concept of standards into the definition, an important* element in the establishment of consistency.

Selection of a Definition. Given the seven options above, and following considerable discussion among the members, the PAT ultimately adopted definition #5. It was found that this definition fell within the "comfort zone" of all members, proving that the communication which had taken place during the definition process had served to transmit each member's perspective to the others. An acceptable definition had been found which was not the "lowest common denominator."

PTS Architecture Design Team

ADT Architecture Definition. *Following the PAT's selection of* a definition, the PTS ADT addressed the architecture issue from its lower-level perspective. Unsurprisingly, the ADT chose a definition which was slightly more complex:

"A framework that conveys engineering decisions regarding the logical and physical partitioning of a system that meets system requirements *and constraints*, which establishes hardware and software interface relationships among partitions to a specified level, and defines a coordination strategy that constrains how the partitions communicate, are controlled, are observed, and are synchronized."

This variant of PAT definition #6 recognizes that there are *other constraints levied on a design which are not formally* called out as requirements, and may affect its partitioning.

Application of Definition. The PTS ADT went a step further than the PAT in it applying this definition. In the course of developing engineering concepts for the pilot trainers, the ADT identified a number of different approaches to building training devices. Recognizing the need for an architecture, it was determined that further investigation would need to focus on those approaches which represented true architectures. The ADT definition of architecture was thereby used to *identify valid architectural options from a list of trainer design* approaches, and cull the list for the more thorough investigations to follow.

High school chemistry students are familiar with the Litmus Test, wherein a chemically treated paper is dipped into an unknown liquid to determine whether it is an acid or a base. A chemical reaction will cause blue litmus paper to turn red in the presence of some solutions, allowing those solutions to be positively identified as acids; but the reaction will not occur - and hence the paper will remain blue - when immersed in "something else" (a base or a neutral solution.) In a like fashion, the ADT devised a "litmus test" to ascertain whether a proposed approach could be called an architecture. If the approach met the criteria - caused a reaction, as it were - it would be positively identified as an architecture. If it failed to meet the criteria, it would be labeled "something else."

The criteria against which the proposed approaches would be *evaluated were based on the ADT's adopted definition.* Essentially, to be evaluated by the ADT, an approach had to address system partitioning, establish inter-partition interfaces, and define a coordination and communication strategy, within the context of the established requirements and constraints. If the approach met all of these criteria, the "reaction" would occur, and the approach could be categorized as an architecture. If not, being deficient in some way, it would need to be either modified or dropped from further consideration.

Evaluation. The ADT generated a list of sixteen potential approaches to the development of pilot training devices, of which ten were identified as passing the "litmus test" for *qualification as architectures. (Some of the options were later* determined to be subsets of one another. The ten were subsequently collapsed into a set of six orthogonal options.) A few examples of these approaches follow:

An option which qualified as an architecture was the "Domain Architecture for Reuse in Training Systems (DARTS)" [Crispen 93]. Through its inheritance of partitioning and control concepts from the Air Vehicle Structural Model (AVSM)[4] and the Modular Simulator[5] program, DARTS was found to address all of the criteria for further consideration.

---

4 As described in the Structural Modeling Guidebook.

5 The Modular Simulator Program, A.K.A. ModSim or Have Module, was conducted in the late 1980's by the ASC Training System Program Office (ASC/YW) to develop a standard, high-level functional hardware partitioning scheme for training simulators.

Another alternative which passed the "litmus test" was the "Surrogate CIP." This approach attempts to minimize cost by substituting commercial computer hardware for the Common Integrated Processor avionics hardware in the training devices. Being based on the Avionics Integration Laboratory software coordination and partitioning approach, as well as commercial hardware standards, it was found to meet the architectural criteria.

An option which failed the test - i.e., did not qualify as an architecture - was "No Air Vehicle Hardware," an approach intended to build trainers without using any flightworthy avionics, cockpit controls, or display devices. While it might be considered an architecture from a hardware standpoint, this alternative failed the "litmus test" because it defined no scheme for partitioning or controlling software processes.

The remainder of the suggested approaches were assessed in a similar manner, allowing several approaches to be dropped from further consideration.

Relative Quality. Beyond just categorizing an approach as either an architecture or as "something else," a number of factors were used to assess the relative quality of an architecture. These factors[6] are as follows:

Scalability: Support for higher or lower fidelity devices.

Extensibility: Support for the addition of new functionality.

Distribution Across Processors: Allows the use of a multiprocessor hardware architecture, without requiring changes to the architecture.

Manage Complexity: Comprising a relatively low number of different object types; interactions among objects are simplified.

Contain Cost: Includes tools to measure and observe cost factors.

Manage Data Bases: Provides mechanisms to manage and control the amount of information needed to provide effective training.

Concurrency: Supports the rapid incorporation of air vehicle configuration changes into the training devices.

Support Changing User Requirements: Supports the rapid incorporation of changing user requirements into the trainers.

Simulate/Stimulate/Emulate Mix: Allows the deferment of decisions regarding whether specific components of the trainer will be simulated, stimulated, or emulated.

Manage Malfunctions: Supports the future addition of malfunctions, as well as all initial malfunction requirements.

Cost Considerations: Provides the means to measure EMD, Production, and Support costs.

In addition, a good architecture was defined as one which could accurately predict the behavior of the ultimate design; provide an understanding of the overall system; communicate implementation rules to designers; include mechanisms and tools to manage the development; provide quantifiable assets; and control complexity.

As an aid to understanding the nuances associated with each architectural option, a preliminary assessment of the six combined architectures was conducted with respect to these factors. Unsurprisingly, it was found that those architectures which were developed for the training system application were better suited to the F-22 PTS than those developed for the laboratory environment. In the examples above, DARTS fared better than Surrogate CIP for just this reason.

Lessons Learned

PAT Lessons. The lessons which can be taken from the PAT exercise to select a definition of the term "architecture" may be summarized as follows:

1. Since the concept of an architecture can be difficult to grasp, especially by non-technical personnel, a consensus definition needs to be agreed upon which falls within the "comfort zone" of each individual in the group.

2. Architecture can be defined using a series of discrete, additive concepts, to yield a hierarchy which can support the various "comfort zones" without reducing the definition to the lowest common denominator.

3. By presenting the group with a range of definitions, as opposed to a single one which embodies a considerable number of different concepts, each individual can find his own "comfort zone," and identify the specific factors which fall outside this boundary.

ADT Lessons. Additional lessons which were learned from the subsequent ADT activity might be encapsulated as:

1. The level at which an architecture will be applied will influence the consensus definition. As one gets closer to the implementers of the system, the more detailed the "comfort zone" becomes. The ADT "comfort zone" was accordingly lower in the hierarchy than the PAT definition.

2. It is not sufficient to simply state that an approach is or isn't an architecture, and proceed to implement it. All architectures are not equal; there are varying degrees of quality, even within the established set of architectures. To

[6] This list was derived from information provided to the ADT by Mr. Joe Batman of the Software Engineering Institute.

find the optimum solution for any problem domain, the available architectural options need to be weighed against the relative quality criteria for that domain.

Summary. Defining the architecture term was a significant step in the process of actually developing an architecture, because it established the objective for the activities to follow. This paper will now proceed to describe how this definition will be applied to create an architecture for the training system.

ARCHITECTURE IMPLEMENTATION

Creating the Architecture

The current approach of the Architecture Design Team is to lay out, top down, an architecture for what we expect to be the most functional and highest fidelity pilot training device. The architecture will be scalable to adapt to whatever suite of training media is recommended by the instructional analysts. The intent is to provide a common framework applicable not just across the range of pilot training devices, but for some number of maintenance training devices as well. This is one reason for starting with the most complex device in the training system; it is easier to scale down to other applications that don't require the full functionality or fidelity of the complex system, than to scale up to a complex device from the framework developed for a simpler device.

The ADT is considering a two-tier repeating pattern suggested by SEI. One tier comprises a series of components that represent different domains within the simulator device, and the other tier represents a "foundation" that manages the interactions between components, including communications and activation. Within any given component, the two-tier pattern may be repeated. The ADT, along with SEI, has developed a "first layer", which delineates the boundary lines between parts of the training device that have potentially different internal architectures. The problem represented by each component will be examined to determine whether any of the architectures are sufficiently similar to suggest common solutions. Each component will then be developed until its architecture reaches a level of granularity that provides the aforementioned benefits, e.g. cost predictability, comprehension, etc.

The basic method used to develop the architecture within any given domain will be to examine the problem domain it represents, bound and define the problem, and create abstractions of the problem definition in an iterative fashion until patterns emerge that can be captured in the solution.

The resultant solution will be analyzed for resource requirements, both in terms of real time dynamics and development costs. If possible, the architecture will be captured in a tool that will permit more exhaustive characterization of its behavior and platform requirements.

Using the Architecture

In the near term, the architecture will be utilized as a tool to assess various options under consideration by the instructional analysts, as a framework from which to assess the reusability of air vehicle developmental items that have been pursued by the Training System as "planned reuse" (as well as "opportunistic reuse" options), and as a basis from which to conduct a sound make/buy analysis.

Training Media Cost/Benefit Trades. The ISD process is approaching a phase where multiple device suite options will become apparent, and the analysts will require rapid analysis of the costs of various approaches as input to their trade studies. The SEs intend to utilize the architecture as a sort of design database, where the various media options represent different views into the database. This will permit a credible, rapid assessment of the viability and cost of options under consideration. This is one example of how the interaction that is possible between ISD and Systems Engineering can result in an optimal mix of training media for effective pilot and maintainer training.

Reuse Assessment and Recommendation. It is apparent that the reuse of a component from a foreign architecture will strain the target architecture in some manner. To some degree, the attributes that were built into the target architecture will be compromised by the introduction of an entity whose structure represents a different pattern set. Drawing again on the analogy of the architecture of buildings, [Alexander 77] states that in short, "no pattern is an isolated entity. Each pattern can exist in the world, only to the extent that it is supported by other patterns: the larger patterns in which it is embedded, the patterns of the same size that surround it, and the smaller patterns which are embedded in it." When this principle was realized by the F-22 PTS, an effort was made to anticipate the ultimate architecture of the training devices, and use its patterns to influence the structure of certain items targeted for potential reuse. Considerable success was achieved in the areas of Flight Dynamics Simulation, Air Data Sensor and Gyro & Accelerometer Simulations, Electronic Warfare (EW) RF Sensor Simulation, EW Countermeasure Controller Simulation, EW Missile Launch Detector Simulation, Comm/Nav/Ident (CNI) Radio, In-Flight Data Link, TACAN, ILS, and IFF Simulations. All of these simulations are being developed by air vehicle IPTs, and each IPT adopted a close variant of the AVSM (with a little encouragement, as well as SEI support, from the Training System IPT.) In any case, every reuse candidate must be analyzed with respect to the strain it places on the architecture (which impacts directly on life-cycle cost) versus the development cost of an alternative.

Make/Buy Analysis. As much as the architecture can serve as a tool for comprehension and communication to developers, it has utility to provide industry with a clear definition of the problem; every bit as clear as that available to the prime contractor, ideally. The F-22 training system is pursuing means by which industry can participate in the archi-

tectural development process with domain experts, in an attempt to build the best solution for the user that industry can create, and at the same time get industry familiar with the various problems being faced on the F-22 program. This will enhance the likelihood that a set of well-positioned contractors can be selected to build various parts of the F-22 training system. The architecture will also be used to provide insight into exactly what components, if any, are best built by the prime, while ensuring that pieces built by different companies will integrate well.

None of this discussion precludes a total buy option. Whether the device is built in-house or by a vendor, the problem should be well understood. The F-22 Training System is sensitive to the fact that our problems are of a sufficiently unique nature that vendors are not likely to desire involvement without a solid understanding of what they're getting into. The problem does not necessarily scale well from previous fighter simulation programs, so the existence of a descriptive architecture is a key to its success.

Managing the Development

Once the architecture is established, the SEs will create tools which will enhance their capacity to enforce the architecture. For example, component packages will be templated to ensure consistent treatment of interfaces and coherent style. Documentation standards will be published and perhaps templated to assist in the creation of a design database that will be used to analyze the ongoing development and define the executive layer. A package will be developed to permit rapid orientation of new developers to the context within which they will operate, and the specific requirements for the components they will be responsible for.

The SEs will monitor the development process using the standard documentation produced by the developers. The data being provided will be used to update the parameters that were assumed in the analysis of the architecture to validate expected system behavior. The iterative analysis will pinpoint emerging risk areas, which will be tended to quickly by reallocation of resources or by modification of the architecture. The SEs will resolve ambiguities and enforce adherence to the system design. This process will continue throughout integration and test.

SUMMARY

The role of an architecture within the Systems Engineering process is difficult to express, as few people agree on the definition of architecture and few companies have a strong Systems Engineering process.

The F-22 Training System wrestled with these issues and came to consensus on a range of complimentary definitions for the concept of architecture, and has considered how the architecture will be used by the SEs to confront the large problem ahead.

Applicability of F-22 Lessons. The F-22 Training System must provide training for an aircraft that is highly integrated, flexible in mission, smart enough to fuse data before the pilot sees it while offering the pilot pre-fused information on request, and reconfigurable in real time to adapt to failure conditions. Past "black box" simulation techniques will never capture the myriad possible modes. More recent trends to create strong mapping to air vehicle "boxes" won't work in many cases because we are not dealing with a federated system, and air vehicle software is not tied to any particular air vehicle processor; functions may move around.

What is occurring on F-22 is simply an example of systems scaling to levels of complexity that cannot be addressed with techniques that worked for smaller systems. Industry found that the last time there was a quantum leap in aircraft system complexity, the AVSM architecture was adequate to allow the aircrew training systems to keep pace. Now that another scaling leap is occurring with the F-22, it appears that the application of strong Systems Engineering throughout development, fully leveraging an architecture exhibiting the same principles as the AVSM, will prove capable of solving the F-22 Training System problems (and not just for aircrew training this time). It seems reasonable to expect that the process described can provide a generic paradigm with applicability to many domains facing similar problems of scaling complexity.

ACKNOWLEDGMENTS

The authors wish to acknowledge Mr. Joe Batman of the SEI, for his contributions toward the synthesis of an architecture for the F-22 PTS; Mr. Michael Rissman of the SEI and Mr. Bill Schelker of ASC, for their seminal efforts in the development of architectural constructs for training devices; and the members of the F-22 Training System Architecture PAT and the PTS Architecture Design Team, for their efforts leading to the findings described herein.

BIBLIOGRAPHY

Alexander, C., "A Pattern Language," Oxford University Press, 1977

Crispen, R.G., Freemon, B.W., King, K.C., and Tucker, W.V, "DARTS: A Domain Architecture for Reuse in Training Systems," 15th I/ITSEC, Nov '93.

Petroski, H., "Design Paradigms," Cambridge University Press, 1994.

"Structural Modeling Guidebook (Draft)," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, May '93.

"Systems Engineering Management Guide," Defense Systems Management College, Ft. Belvoir, VA, Jan '90.