The Heritage of the Air Vehicle Training Systems Domain

David C. Gross and Lynn D. Stuckey, Jr.
Boeing Defense and Space Group

ABSTRACT

One of the Holy Grails of software development has been *reusability*. Everyone is frustrated with continually reinventing the wheel; everyone knows that reuse would dramatically cut costs; and no one has shown an effective reuse paradigm. The trend has been to develop reuse paradigms without regard to past successful projects. Historically, successes with reuse have been accidental -- based on personnel, not on process. Now a new paradigm has emerged that includes a focus on past investments in forming a reuse process. This initiative is DoD's push toward the *megaprogramming* paradigm. Megaprogramming divides system development into two lifecycles, the first focusing on the problem of leveraging assets through a family of related products, and the second focusing on the problem of delivering a single product. The process for the first lifecycle is domain engineering.

Domain engineering is not easy. It revolves around all kinds of questions that simulation software engineers are not used to asking such as: (a) Is this a viable domain?, (b) Is there an acceptably standard partition of the domain?, (c) Is this domain definable?, (d) What granularity is best for domain work products?, and so forth. Yet, if the DoD is going to successfully transition its approach for the development of software intensive systems to the megaprogramming paradigm, software development organizations are going to have to be empowered to meet these challenges.

The U.S. Navy and the Advanced Research Projects Administration are presently funding a megaprogramming demonstration project in the domain of Air Vehicle Training Systems. How has this project come to grips with the technical challenges of domain engineering? Mostly by leveraging the investments of previous research and development projects in this domain such as the Ada Simulation Validation Program (ASVP), the HAVE Module (Mod Sim) Project, the Software Engineering Institute's Structural Model Initiative, the Manned Flight Simulator (MFS), and a series of planned pilot efforts. This paper discusses the advantages and disadvantages on leveraging previous investments into new domain engineering efforts. Its discussion captures valuable lessons about the transition of existing organizational assets into the megaprogramming paradigm.

ABOUT THE AUTHORS

David C. Gross is a software systems engineer with the Missiles & Space Division of the Boeing Defense & Space Group. He has worked in all lifecycle phases of simulation and training systems from requirements development through delivery. He is currently involved in applied research related to megaprogramming in the domain of training simulator systems. Mr. Gross holds a Bachelor of Science in Computer Science/Engineering from Auburn University and a Master of Operations Research at the University of Alabama at Huntsville. His thesis compares the utility of high level languages such as C++ and Ada for simulation. Mr. Gross is a doctoral student at the University of Central Florida, and can be reached at gross@plato.ds.boeing.com

Lynn D. Stuckey, Jr. is a software systems engineer with the Missiles & Space Division of the Boeing Defense & Space Group. He has been responsible for software design, code, test, and integration on several Boeing simulation projects. He is currently involved in research and development activities dealing with software reuse in the domain of air vehicle training systems. Mr. Stuckey holds a Bachelor of Science degree in Electrical Engineering from the University of Alabama, in Huntsville and holds a Master of Systems Engineering from the University of Alabama, in Huntsville. His thesis presents a systems engineering approach to software development. Mr. Stuckey is a doctoral student at the University of Central Florida, and can be reached at stuckey@calif.enzian.com

The Heritage of the Air Vehicle Training Systems Domain

David C. Gross and Lynn D. Stuckey, Jr.
Boeing Defense and Space Group

INTRODUCTION

"The whole of science is nothing more than a refinement of everyday thinking."
Albert Einstein, 1936

Research efforts in software, especially simulation software, have been slow to come to grips with the concept of evolution versus revolution. There are few if any silver bullets in software development, but the industry continues to re-invent the wheel in search of new productivity and quality. These leaps forward are more likely the result of understanding present efforts and then building upon their strengths. If Einstein is correct, then a critical aspect of any project attempting to advance the state of practice must be an understanding of what constitutes "everyday thinking" in that practice. How can we refine something without first understanding it enough to communicate with it? This challenge is further complicated by the diverse nature of technology today -- individual problem domains for completely legitimate reasons adopt approaches which directly conflict with the solutions adopted in other problem domains. Consider for example the Ada language debate -- much of the heat in language discussions arises because the participants fail to acknowledge that different problems have different solutions. Finally, complications arise because the state of the *practice* so badly lags the state of the *art* -- indeed the state of the practice is hardly a tightly constrained range of behavior!

The STARS demonstration of megaprogramming in the Air Vehicle Training Systems (AVTS) domain is certainly trying to advance the state of practice. The STARS approach is a three-pronged attempt to improve the reliability and adaptability of complex software systems. The first prong is *automation* -- the realization that demand for quality software has outstripped our resources to supply it. The second prong is *process* -- the recognition that repeated improvement is only possible when the method of construction is defined and followed. The process in the context of STARS is megaprogramming -- which separates the creation of systems into two distinct lifecycles: domain engineering and application engineering. Domain engineering aims at creating assets for use within a product family; application engineering aims at delivering specific instances of that family. The final prong, and our focus in this paper, is *domain-specific* -- the acknowledgment of the issues raised above.

So in which problem domain are we working? The AVTS domain is a family of air vehicle training devices that provides the simulation, stimulation, and/or emulation of all the components and systems for a real-time air vehicle simulation. [STUCKEY] This domain encompasses the systems necessary to provide training devices that a trainee uses to become familiar with the configuration and/or flight characteristics of an application air vehicle, gain proficiency in executing normal procedures, recognizing malfunctions/abnormal indications and executing the corresponding standard/emergency procedures, and executing mission procedures. The devices, or *instances*, of this domain are some proper subset of the domain. This domain includes the system diagnostic and test requirements for the applicable air vehicle devices based on individual segment requirements.

But this work is not occurring in a vacuum, as illustrated in Figure 1. Indeed, where it not for the rich

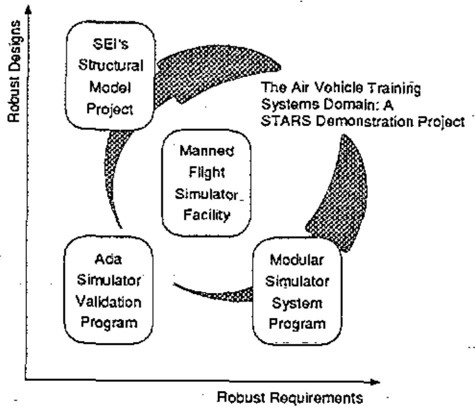
Robust Designs
SEI's Structural Model Project
The Air Vehicle Training Systems Domain: A STARS Demonstration Project
Manned Flight Simulator Facility
Ada Simulator Validation Program
Modular Simulator System Program
Robust Requirements

Figure 1: Projects Leveraged in the AVTS Domain

resources developed by earlier research in the field, this project could not be attempted -- the problem domain would simply not be sufficiently mature to make the attempt with constrained resources. This paper discusses some of the research in the domain that makes this project possible, and the challenges in incorporating their results

HERITAGE PROGRAMS

Ada Simulation Validation Program (ASVP)

Synopsis.

The ASVP was an Air Force contracted research and development program. The program spanned some 24 months from 1985 to 1987 with extensions that followed. It was part of the continued effort by the DoD to have contractors demonstrate the validity and application of the Ada programming language in a variety of environments.

The task for this project was to re-develop in Ada, a substantial portion of the application software for the existing E-3A Flight Crew Trainer (FCT), and to answer the following questions: Does Ada work on simulators? Is Ada better or worse than FORTRAN for simulators? What are the better methods for designing simulator software in Ada? What software development and support tools are necessary in the development lifecycle?

ASVP was a 24 month program consisting of some 23,000 man hours. Boeing was teamed with SAIC and Encore. The E-3A simulator re-developed was fielded at Tinker Air Force Base. The software was re-developed and tested in Huntsville Alabama, while hardware/software integration, system test, and demonstration occurred at Tinker. The work at Tinker was accomplished on second and third shifts. This schedule permitted E-3A crews to continue training with no disruptions to their training mission.[ASVP-FR]

Contribution.

ASVP was an overwhelming success. It received the first YW Systems Quality Award. The system passed 87% of the acceptance test procedures on the first pass. The FCT was evaluated by Air Force pilots with over 4000 flying hours in an E-3A and by instructors who trained pilots daily. The program also became the basis for future Air Force simulator work. It provided guidelines for the implementation of structural models, coding standards, and an object-oriented design methodology for simulators.

Modular Simulator System Program (Mod Sim)

Synopsis.

The Mod Sim program was a tri-service supported development program. The primary goals of the modular simulator design were to shorten simulator development schedules, reduce simulator development costs and improve simulator supportability. The program was organized into three phases. Phase I surveyed the industry as to the desirability and feasibility of introducing a generic modular simulator concept. Phase II, Modular Simulator Design Concept Development, produced a conceptual modular simulator architecture with a focus on aircrew simulator functional analysis and intermodule communication architecture/design. The contractors developed a conceptual modular simulator design for this effort. Phase III consisted of design, demonstration and validation of the modular simulator concept. To foster industry participation and "buy in" to the Mod Sim design, Boeing was required to subcontract the design and development of 50 to 75 percent of the segments. The Phase III subcontractors were Rediffusion Simulation Limited (RSL), Science Applications International Corporation (SAIC), AAI, and Intermetrics. To gain further industry participation, regular Interface Standards Working Group (ISWG) meetings were held. At these meetings both industry and government simulation experts were allowed to participate in the review of the modular simulator design and subsequent demonstration.

Phase III was divided into two parts. Part 1 accomplished four major tasks:

a. System Partitioning. This task involved the analysis of simulations for a large number of fixed and rotary wing training devices. This data, along with other raw data and the conceptual partitioning from Phase II were used to create a Functional Dictionary that contained an allocation of all functions and the interface requirements between functions. The Functional Dictionary and segment partitioning were refined through an iterative process using an Artificial Intelligence tool. This resulted in segments that had generic intersegment interfaces, were loosely coupled, and focused on a specific area of simulation expertise.

b. Communication Architecture. This task involved the specification of a hardware and software communication architecture that would allow the segments to communicate effectively.

c. System Performance Model. In order to efficiently select a communication architecture a System Performance Model was constructed to emulate the various design alternatives. Fourteen data buses and seven protocols were analyzed.

d. Specifications. To promote the standardization of the Mod Sim architecture, a thirteen volume generic System/Segment Specification (DOD-STD-2167A) was prepared. The system level specification defines the communication architecture and requirements common to all segments. The segment level specifications define the unique requirements applicable to the segment.

During Part 2 of Phase III, Boeing and the subcontractors demonstrated the Part 1 design. The demonstration was accomplished using a government provided F-16 crew station and existing F-16 simulator source code. The government furnished products were adapted to the modular simulator partitioning and communicated using the modular simulator communication architecture.

At the completion of the Phase III, several follow-on tasks were contracted. These consisted of adding an interface to the Mod Sim architecture to support multiple simulator/team training (e.g.; Distributed Interactive Simulation), adding tailoring instructions to the generic specifications to ease adaptation to specific applications, and the creation of Mod Sim guidance documentation. This documentation included an engineering design guide, a management guide and an executive report that provides an overview of the Mod Sim approach. The Mod Sim architecture is shown in Figure 2. The architecture consists of 12 distinct segments that communicate via a Virtual Network (VNET). [MSS-MGT]

Contribution.

There are several distinct advantages to using the modular simulator design and design concepts in developing training devices. These advantages include:

a. Systems Engineering. The Mod Sim design provides a wealth of generic systems engineering products that are reusable for any application. This reduces front end development cost and schedule and mitigates risk throughout the project.

b. Subcontracting. One of the primary requirements for the Mod Sim architecture was the capability to independently specify, develop, and test individual segments as stand-alone products. This enhances the ability to subcontract development of segments by providing well-defined interfaces that reflect a straight-forward allocation of simulator functions along traditional subsystem

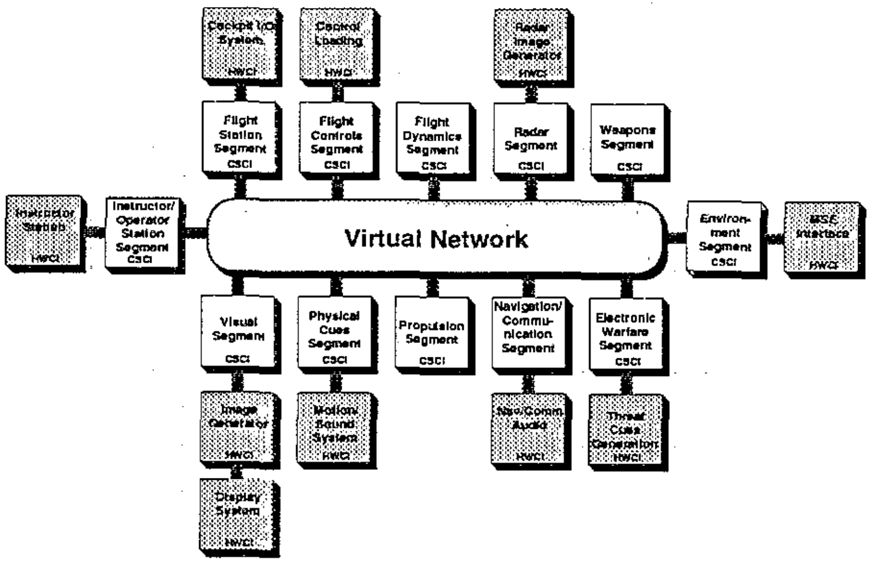

Figure 2: Mod Sim Architecture

product boundaries, to make best of advantage of individual organization's strengths.

c. Integration. Use of the Mod Sim architecture and strong Ada design principles significantly reduces integration time. This has been proven in a series of demonstration projects.

d. Reusability. The Mod Sim architecture *promotes and enables reuse among families* of training devices and applications. Experience has shown that architecture is the key to successful higher order reuse.

e. Design Flexibility. The Mod Sim architecture allows latitude in design to support low cost and high cost devices. The Mod Sim architecture does not place any requirements on the internal design of the segments.

f. Parallel Development and Stand-alone Testing. Mod Sim segments can be developed and tested in parallel due to the well defined segment requirements and intersegment interfaces. This can significantly shorten the overall system development schedule and reduces integration risk by eliminating common interface problems early in the development and testing phases.

SEI's Structural Model

Synopsis.

The contract work to define a structural model and the concept of structural modeling has been a collaborative effort between the United States Air Force, it contractors, and the SEI. A structural model is an application framework for flight simulators *and the structural modeling process is the* means by which this framework is engineered into a complete system. The recognition of the technical risks associated with building complex systems such as flight simulators was the catalyst that drove the development of the structural modeling method. The broad objective behind structural modeling was to take a complex problem domain and abstract it to a coarse enough level to make it manageable, modifiable, and able to be communicated to a diverse user and developer community. Structural modeling experience has been gained in a number of recent simulator acquisitions, including the B-2 Weapon Systems Trainer, the C-17 Aircrew Training System, and the Special Operations Forces Aircrew Training System. In addition, the Real-Time Simulators Project at the SEI is currently drafting a guidebook describing in detail structural modeling as it applies to the development of an air vehicle within a flight simulator, specifically addressing the case study of the T-39A flight simulator. Figure 3 illustrates the Air Vehicle Structural Model. [ABOWD]

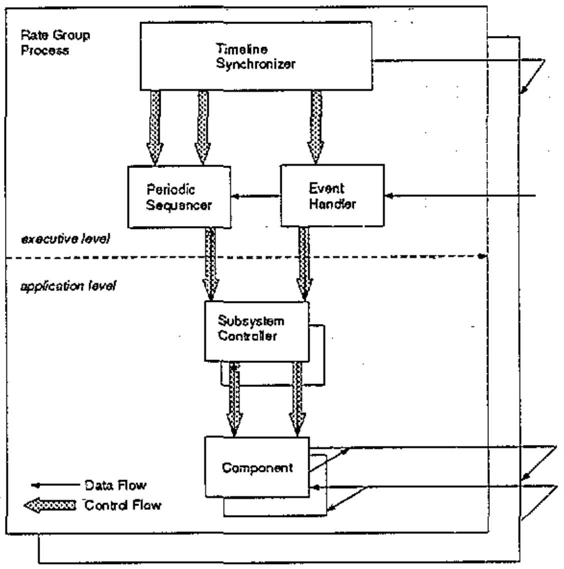

Figure 3: SEI's Air Vehicle Structural Model

Contribution.

Specific benefits realized in flight simulator software development from the structural model described are:

a. Increased separation of the coordination model from partitioning strategy,

b. Easier integration of independently developed software components, and

c. Stronger classification of systemic and (particular) mission requirements.

The Structural Model project determined that considerable design reuse is feasible; whereas code reuse is restricted to the level of the component. The project concluded that modeling mission requirements is more volatile than modeling the other requirements, and it is appropriate that they are no longer the main drivers of the software design.

Manned Flight Simulator (MFS)

Synopsis.

The U.S. Navy, in response to every increasingly risky and costly flight test of new aircraft, decided to create Manned Flight Simulator(MFS). The goal of MFS was to provide to the Navy a low cost high fidelity simulation capability which would allow for growth. The Navy created a laboratory facility and funded the development of a highly modular engineering software architecture. The architecture was created and called Controls and Simulation Test Loop Environment(CASTLE). CASTLE is a highly modular simulation system which affords the simulation engineer the ability to create models *with the knowledge that his equations of motions*, environment, hardware and visual environment interface are all predefined. The pre-definition and reusable modules allow the simulation engineer to concentrate on the functionalities of his math model. The manned flight simulator has a long history of opportunistic reuse of such airframe models as, F-14, F/A-18A, T-45A, AV-8B, AH-1W, V-22, UH-60. All of these airframe models were obtained from outside sources and rehosted to the CASTLE system. Upon rehost, the NAVY was able to apply *advanced engineering and analysis tools to in*crease the fidelity of these models. This opportunistic reuse has saved the Navy money and created a simulator environment that works in close conjunction with the flight test personnel. [PRYOR]

Contribution.

MFS has evolved techniques for developing simulation models that are abstracted from the underlying hardware. It is clear that many simulation models are completely independent of hardware, for example, the atmosphere model. However, it is more difficult to isolate other models to a minimum of dependencies. MFS has established that this paradigm shift can be made, and real benefits for reuse flow from it.

Other Relevant Projects

There are several other DoD initiatives that involve the standardization of simulators and training systems. Examples include Distributed Interactive Simulation, Project 2851 Database Standardization, the Simulator Data Integrity Program, and the Universal Threat System for Simulators. AVTS does not constrain the use of these standards. In fact, some characteristics of the architecture were designed to accommodate or enhance compatibility with these standards, in so far as publicly available materials permit.

LEVERAGING LEGACIES INTO DOMAIN ENGINEERING

It is important to understand our perspective on why the use of legacy assets is important. To this end, a description of our process, level of reuse, and utilization of legacy assets is appropriate.

Megaprogramming

Over the years, the STARS project has focused on enabling a paradigm shift of DoD software practices to megaprogramming. The central megaprogramming concept is a process-driven, two lifecycle approach to software development. One lifecycle spans the creation and enrichment of a family of related product, or *domain* (see Figure 4). The other lifecycle spans the construction and delivery of individual products to customers, or instances of the domain. The STARS project uses the Virginia Center of Excellence's (VCOE) process called Synthesis for detailing the separate lifecycles. Synthesis defines the effort associated with creating the assets as *domain* engineering, and the effort in creating a specific product as *application* engineering. With a legacy asset perspective, domain engineering is the important lifecycle.

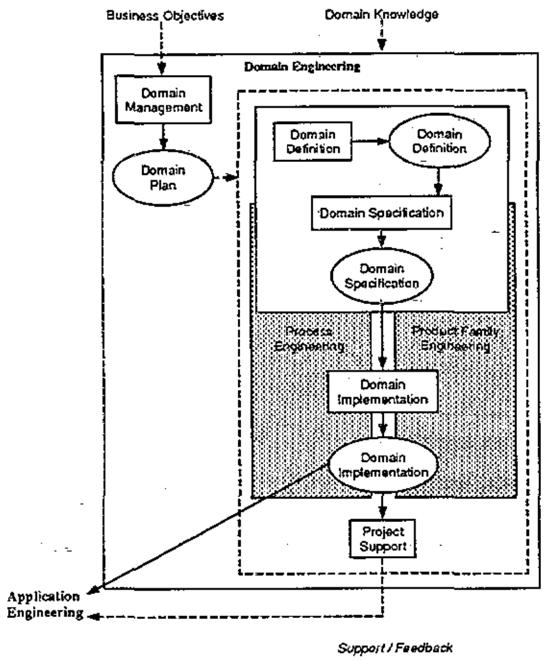

Figure 4: Domain Engineering

Level of Reuse

Synthesis is a process developed for *leveraged* reuse. Leveraged reuse is one of five approaches to reusing software: (a) ad hoc, (b) opportunistic, (c) integrated, (d) leveraged, and (e) anticipated. Leveraged reuse assumes that a given product is actually a member of a product family, the members of which share some degree of commonality. Synthesis involves the definition, analysis, specification, and implementation of a domain which encompasses a *viable* product family -- a family which shares sufficient commonality (and predictable variability) to justify an investment in the domain. Individual products are developed as instances of the domain, which reuse common elements of the domain and adapt variable elements *using a defined, repeatable process.*

Given our focus on leveraged reuse, legacy assets become increasingly important. They are the basis for the structure, evolution, and *product fragments* held within the domain. Without prior research in the field, the AVTS domain would be non-viable.

Application of Legacy Products

Figure 5 describes the usefulness of legacy assets within the domain engineering cycle of the Synthesis process. It is evident that these assets have a major role in most activities. In domain management they form the basis for viability assessment, evolution planning, and risk analysis. In domain definition the assets are used to bound the domain, validate terms, and define assumptions. In domain specification they are used in requirements analysis and product design. In domain verification they provide historical context to evaluate correctness. In product implementation the assets are used for algorithmic development as well as entire module utilization. In domain validation they are used as a measure of effectiveness. Finally, the assets are utilized as an historical perspective on domain delivery.

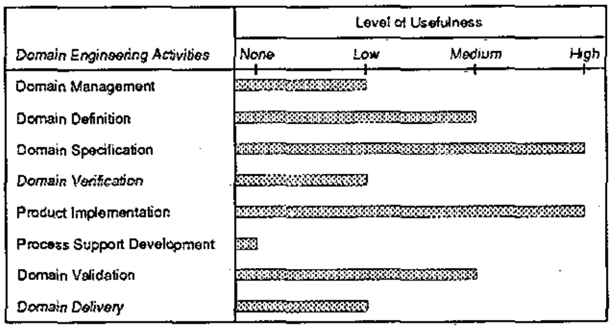

Figure 5: Usefulness of Legacy Assets

LEVERAGING PROGRAMS

The attempt to incorporate new technologies, such as megaprogramming, into the state of the practice is fraught with peril. Figure 6 illustrates the process (which by itself is a example of building upon prior research -- this is a chart from ASVP with Megaprogramming substituted in for Ada). The existence of useful prior research presents a paradox: prior research created the critical mass to make the attempt, but using such research can open a Pandora's box of problems. The following discussion outlines the pitfalls and heuristics for such use.

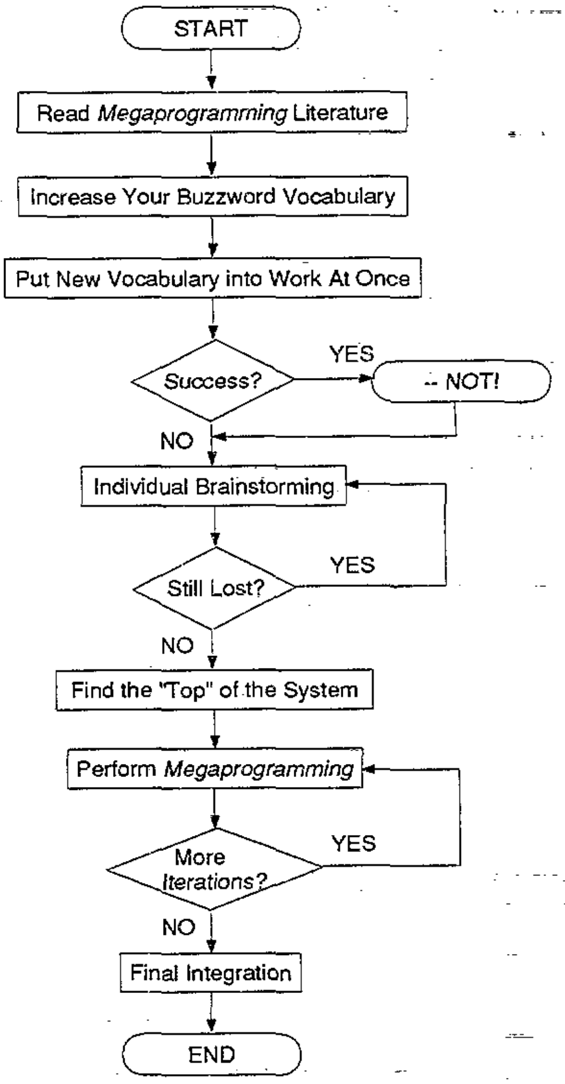

Figure 6: Adopting Megaprogramming

Pitfalls

Our experience has uncovered some of the pitfalls of building upon research legacies.

❑ Making the Leap of Faith

Every demonstration project is organized around some central "great concept." The great concept for the STARS demonstration project in the AVTS Domain is megaprogramming For the purposes of the project itself, it is (surprisingly) not so important whether or not it *actually is* great -- the purpose of the demonstration project is to evaluate that very premise. In fact, the participants on the project must be able to maintain a *suspension of disbelief* about the worthiness of the concept in order to make their very best effort at making it work. This is analogous to the difference between "pure" science and "pure" engineering. Failed science experiments are not surprising and even expected -- but failed engineering projects are failures for which (generally) someone is held accountable. Science (and engineering research) is a venture into the *unknown* -- engineering is the application of *known* mathematical and scientific principles to practical ends. Therefore, the productive participant must maintain this mindset -- but this attitude alone is not enough! The productive participant must also be prepare to re-evaluate the most fundamental principles in light of the great concept. He/she must be able to conceive and implement a paradigm shift -- discarding long held and favored practices in so led by the great concept. This requirement of suspending disbelief and questioning principles is a trap which captures many fine engineers. Time, and time again, we have seen perfectly good engineers who were unable to make this leap of faith.

❑ Quoting Scripture

The great concept generally arrives in a holy book developed in some other context. Sometimes, it arrives in a well-defined specification e.g., MIL-STD-1815A in the case of ASVP). In the case of the STARS Demonstration in the AVTS Domain, the great concept was defined by the Reuse-Driven Software Processes, developed by the VCOE. The guidebook (along with companion material and help from VCOE) has proved extraordinarily useful in proceeding with the demonstration project. The pitfall is that while the guidebook is a useful first approximation , project participants are tempted to argue from it as revealed truth. Rather than arguing the merits of a particular position, we frequently found ourselves saying "but, the guidebook says..." Relevant and understandable examples prove much more powerful than quoting the guidebook -- its use is in provoking thought.

❑ Ivy Tower Meets the Gridiron

Just because the concept is expressed does not mean it is fully fleshed out. Often, the painful details are "left as an exercise for the student". Organizations frequently respond to such situations with lip service -- how many times has empowerment become a euphemism for no responsibility. How many projects have claimed adherence to some hot technology (e.g., object-oriented) not because they understood it and believed in it, but because it sounded so good.

❑ Not Invented Here Syndrome

If we are to make use of any legacy projects, there must be kernel of truth in them that project participants are able to utilize. *Of course, just because* a research project has occurred does not mean it is prima facia useful -- but the more frequent difficulty is that some people have Not-Invented-Here syndrome. While skepticism is a natural and healthy attitude for an engineer to bring to the latest "revolutionary solution", refusal to consider the possibility of some advance is the death knell for a research effort. If the project can not persuade participants to overcome this attitude, then the legacy *can not be used* no matter how significant their potential application.

❑ Leveled Experience Base

One dilemma that arises in a research project is that the various participants come with the baggage of their own experiences. Each individual sees this project in terms of the ones he or she has worked most often. The AVTS domain includes some natural examples: consider the level of complexity differences between a part-task instrument trainer and a full capability weapon system trainer. Some of our project's participant expect a software system size on the order of 15,000 lines of code -- and at the other extreme are those that expect any "real" trainer must have at least a 1,000,000 lines of code. Obviously, your expectations about size impact your approach to the problems of developing software in the domain. Once again, just as specific trainers are different, so are specific individuals -- the project must be able use and apply this diverse experience base.

❑ The Vision Thing

The empowering premise of a demonstration project is the promise it brings. For example, the Mod Sim project held out the promise of avoiding repeating high cost analysis, and building in the expectation of a diverse contractor base. However, the glitter of the promise is different in every participant's eye. The project *must* be able to create and communicate a common vision for the future. Failure to do so has two major disastrous outcomes. First, while participants are still ignorant that their version of the vision is not the "right" version, they will waste resources working hard in conflicting directions. Second, things get worse when they find out their vision is not the "right" version. Most people will feel a sense of betrayal. This can happen at any level -- engineers or management or project sponsors.

Heuristics

Our experience supports the assertion of a set of heuristics from building upon research legacies. Application of these heuristics will conserve the scarce resources of a research project.

❑ Accept Overlapping Legacies

Our first heuristic is that when various legacies overlap in the approach to some aspect of the domain, the project can productively accept the overlap as "proven" in the context of the domain. The easiest example is Ada -- three of this domain's four research legacies adopted Ada, so the AVTS domain accepts Ada as its language of choice. Few project resources were invested in consideration of the language choice issue.

❑ Seek Synergies Between Legacies

Next, we suggest that when the research legacies seem to complement each other, there are synergies that the project can take advantage of while minimizing the required resource investment. For example, all of the AVTS research legacies pointed at a controlled, hierarchial, architecture with controlled communication -- hence AVTS's adoption of the Domain Architecture for Reuse in Training Systems (DARTS). DARTS is *not* the architecture of any of the research legacies but derives from the best aspects of all of them.

❑ Architecture as the Glue

A heuristic that derives from our participation on all of these research projects is that architecture is the glue that holds the project together -- at least for domains involving complex software systems. If *the project can achieve consensus on an architectural* approach which reflects the needs of that particular project, the architecture serves as the critical context within which participants interpret research developments. On a aggressive project, it is easy for participants to be overwhelmed and left out of the loop -- architecture is an important part of avoiding this disaster.

❑ Controlling Concept Introduction

This heuristic relates to self-constraint -- it is not so important that a given project adopt every idea in the marketplace as it is that the project understand them in its own context. Any research project wants to be at the cutting edge in every aspect of its domain -- if for nothing else to protect its own credibility. But the volume of potentially useful ideas is so great that a project can easily gorge upon the feast and choke to death. Visionaries on the project tend to seek to integrate everything, without regard to how tenuous its relationship to this project. Since the resources of demonstration projects are even more limited than for delivery orders in this age of declining budgets, a project can not be successful by trying to do everything. Everyone must "play by the rules", that is, focus on the objectives of *this* project. Healthy projects will carefully observe the rapid river of research, and *decide* what to fish out rather than swallowing it all.

❑ "Prophets" Vs. "Priests"

This heuristic points out that while the current project is only possible in the light of what has gone before, the current project is fundamentally *different*. The participants on a research project have a mini-lifecycle all their own. When a project is young, the participants are like prophets, challenging the existing order and predicting the future. But as a project matures, its participants become more like priests, protecting and defending the faith against the barbarians at the gate. This is particularly dangerous when a project attempts to leverage a legacy by bringing participants of earlier projects into the new research. Such individuals may have trouble adjusting their mindset to the new challenge. Consider for example one difference between the Mod Sim challenge and the AVTS challenge. Mod Sim's requirement was to build tailorable products; AVTS's is to build adaptable products (and processes). This seemingly trivial shift of nomenclature has extraordinary impact throughout the project. Consider for example the difference between defining an interface between components that can be tailored (i.e., rewritten) by some end user, and an interface that must automatically adapt to an end-user's specific requirements. It is not hard to imagine someone "stuck" in the Mod Sim mindset and unable to correctly address the AVTS challenge.

CONCLUSION

"If I have seen further it is by standing on the shoulders of Giants."

Issac Newton, 1675

Newton was referring to the tremendous advances made by mathematicians, physicists, and astronomers such as Da Vinci, Copernicus, Galileo, and Kepler. At its best, science is the most constructive of human endeavors, with each development building carefully upon the discoveries and inventions of earlier generations. But two things have increased the difficulty of scientific advances in our times. First, no longer do we build upon the shoulders of individual giants, we must build upon the accomplishments of teams of giants. While this is a direct outcome of the complexity of the challenges we are addressing, the product of teams are typically much more amorphous and difficult to quantify -- hence the aphorism, "a camel is a horse designed by a committee." When confusion about the use of an individual's product arises, one consults the individual for a (generally) consistent explanation. But teams are notorious for lack of shared visions, so explanations from different team members frequently conflict.

The second difficulty arises from the size of the technical marketplace. When Newton developed his calculus, he faced competition from perhaps two serious contenders, in a scientific community of certainly less than 10,000 members. In contrast, Capers Jones reports that there are some 1,818,500 programmers working in the United States alone, each one with no doubt strongly held opinions about the "right" language, methodology, tools, etc. Two orders of magnitude more people working on a problem that must be three orders of magnitude less pervasive. Consider the debates in our community about the suitability of Ada -- but Jones says that the leading language in use today is still *COBOL*. Ada fails to make the top five and is lumped in with "all other languages"! How can any purported advance make headway in this tower of Babel? [JONES]

We can advance the state of the practice, not by re-inventing the wheel, but by carefully building upon work which has gone before. Software has been called by many others the most complex endeavor attempted by mankind -- the likelihood of a single individual uniquely making a significant discovery is negligible. The complexity of the problem domain demands teams of world class experts building upon the work of other such teams.

REFERENCES

[ABOWD] Abowd, Gregory D., et. al., *Structural Modeling: An Application Framework and Development Process for Flight Simulators*, CMU/SEI-93-TR-14. August 1993. Software Engineering Institute, Pittsburgh, PA.

[ASVP-FR] *Ada Simulation Validation Program Final Report.* 9 September 1988. The Boeing Company, Huntsville, AL.

[JONES] Jones, Capers. *Gaps in the Object-Oriented Paradigm*, IEEE Computer. June 1994. IEEE Computer Society.

[MSS-MGT] *Modular Simulator System Management Guide*, D495-10439-1. 13 September 1993. The Boeing Company, Huntsville, AL.

[PRYOR] Pryor, Greg. *Personal Communications.* June 1994. Former lead aerodynamics engineer of the MFS AH-1W Aircrew Procedures Trainer.

[STUCKEY] Stuckey, Lynn, et. al., *Technical Expectations of a Full Scale Domain Engineering Demonstration Project.* November 1994. The 16th Interservice/Industry Training Systems and Education Conference.