Implications of Megaprogramming for the Training Systems Community

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

ABSTRACT

STARS (Software Technology for Adaptable, Reliable Systems) is a long running ARPA project aimed at advancing the management, quality, adaptability, and reliability of DoD software intensive systems. Over the years, the STARS project has gradually focused on enabling a paradigm shift of DoD software practices to *megaprogramming*, or *software product line development*. The central megaprogramming concept is a process-driven, two-life-cycle approach to software development. One life-cycle spans the creation and enrichment of a family of related products, or *domain engineering.* The other life-cycle spans the construction and delivery of individual *instances* from the domain. This approach may provide substantial opportunity for *leveraged reuse*, that is, planned use of adapted software components in multiple products.

As part of the STARS project, The U.S. Navy and the ARPA are presently funding a megaprogramming demonstration project in the domain of Air Vehicle Training Systems (AVTS). The focus of this demonstration is the construction of a domain capable of supporting multiple instances of the Navy's training systems product family. This demonstration concludes in October of 1995. If megaprogramming proves useful in this domain, it promises dramatic increases in productivity along with corresponding reductions in the cost of building simulators.

Given the progress and investment in this technology thus far, it is appropriate to consider the changes in the training systems community that will occur if megaprogramming in the AVTS domain proves successful. Clearly, there will be significant changes in the nature of the training systems development, *both* in terms of technical practices and business practices. This paper introduces megaprogramming, discusses the challenges involved in adopting megaprogramming, and finally speculates as to the implications of change in the community if megaprogramming is adopted. The discussion is based on the experiences and lessons learned on the project.

BIOGRAPHICAL SKETCHES

David C. Gross is a software systems engineer with the Missiles & Space Division of the Boeing Defense & Space Group. He has worked in all life-cycle phases of simulation and training systems from requirements development through delivery. He is currently involved in applied research related to software reuse 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 Ada and C++. Mr. Gross is a doctoral student at the University of Central Florida. He can be reached at gross@eight-ball.hv.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 and a Master of Systems Engineering from the University of Alabama, in Huntsville. Mr. Stuckey is a doctoral student at the University of Central Florida. He can be reached as stuckey@eight-ball.hv.boeing.com

Randall R. Macala is a advanced computing systems technology manager with the Research & Engineering Division of the Boeing Defense & Space Group. He has worked all phases of the software systems lifecycle, including delivery and support. He is the assistant project engineer for the Navy/STARS Demonstration Project, which is applying megaprogramming in the Air Vehicle Training Systems domain. Mr. Macala holds a Bachelor of Science in Mathematics from the University of Washington and has completed advanced studies in applied physics at the University of Washington and in business administration at Kennesaw State College located in Marietta, GA. He can be reached at macala@plato.ds.boeing.com

Implications of Megaprogramming for the Training Systems Community

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

INTRODUCTION

STARS

In June 1983, the Department of Defense endorsed the DoD Software Initiative which had been proposed to offset growing manpower shortfalls and to improve capabilities for supporting software development and maintenance. The initiative included three components: the Ada Joint Program Office, the Software Engineering Institute, and the STARS project. Figure 1 illustrates the STARS approach which is based on megaprogramming. Megaprogramming, or *software product line development*, is characterized by a product line view of the reuse of software life-cycle assets (architecture, components, and processes) and a disciplined, process-driven approach to development and evolution of application systems and the product line. The principle benefit to be derived from software product line development is improved predictability and quality in software and system development.

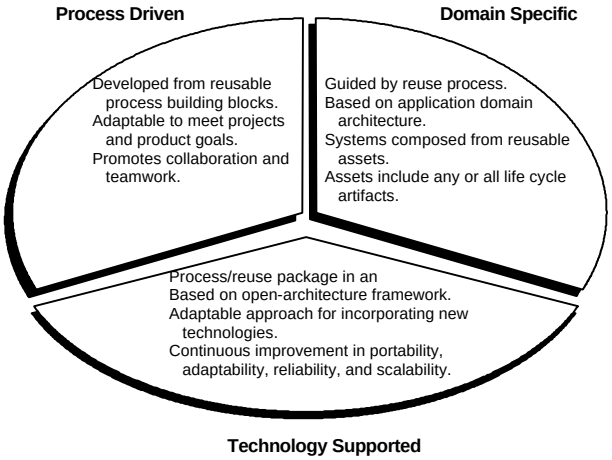

Figure 1: The STARS Approach

Process

Repeatable production of quality software begins with process. The Virginia Center of Excellence (VCOE) created a process called Reuse-Driven Software Process (RSP) for defining a domain, specifying the products and adaptation for instances for the domain, and implementing designs that leverage whole and adapted components for reuse in new systems. [VCOE93] RSP 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. RSP involves the definition, analysis, specification, and implementation of a domain which encompasses a viable product family. 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. RSP defines the work associated with creating the assets as domain engineering, and the effort in creating a specific product as application engineering. The central principle is that domain engineering creates the assets and processes by which application engineering can define and create the best fitting instance of the domain. Figure 2 illustrates the RSP process.

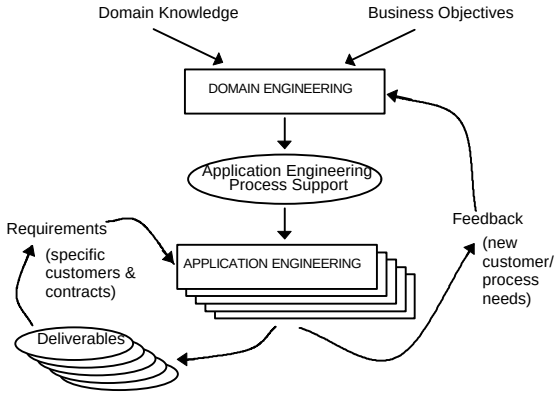

Figure 2: RSP Overview

The Demonstration Project

STARS is engaged in three demonstration projects (one per military service, each in cooperation with a prime contractor). The Navy/STARS demonstration project is in the field of simulation and modeling; selected because of its outstanding potential for systematic, long term product improvements. If their project is successful, the Navy intends to treat training simulators as a domain (product line) -- the Air Vehicle Training Systems (AVTS) domain. In addition to direct Navy involvement, the demonstration project contractors are The Boeing Company and Enzian Technology, Incorporated.

The Navy/STARS demonstration project has thus far defined the scope of the AVTS domain, focused thus far on as domain addressing the needs of trainers for T-series aircraft. These aircraft are used in the instruction of student pilots in the various aspects of naval aviation. The demonstration project will develop at least one instance from the domain, a T-34C Flight Instrument Trainer (FIT). The demonstration project has completed its pilot and preparatory efforts, and is in the midst of full scale development. The demonstration project is organized into four phases as illustrated in Figure 3. A functional T-34C FIT will be delivered using the domain in November of 1996.

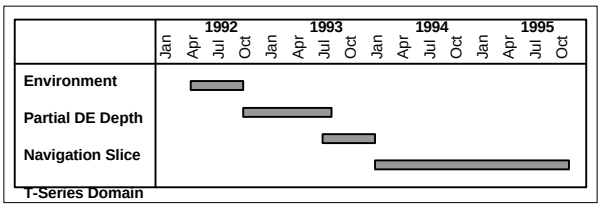

Figure 3: Demonstration Project Schedule

The demonstration project began in 1992 with the application of RSP to one aspect of the domain -- the Environment. This effort created a stand alone domain capable of supporting a few instances of the Environment. The next project phase involved application of the preliminary steps in RSP across the entire AVTS domain. This effort completed domain definitions, specifications, and preliminary design for the aspects most relevant to T-series aircraft. The third project phase finished the domain design and implementation for a slice of the Navigation subdomain, and demonstrated the slice in the context of the AVTS architecture. The current phase of the project is to use the domain engineering process in order to expand the AVTS domain to support the range of Flight Dynamics simulation for T-series aircraft, specifically a T-34C FIT instance. This phase will conclude with the demonstration of a functioning flight dynamics model for the T-34C FIT in October of 1995.

The latest milestone in this phase included an validation test of the Flight Dynamics subdomain with the existing T-34C FIT software. An application engineering session was exercised in order to produce an instance of the domain. That instance was "plugged" into the existing FORTRAN simulation, replacing the existing flight model. The validation was an unqualified success, with remarkable reductions in the resources required for integration and test of conventional software. The result reinforced the fact that the technical viability of this approach has been proven. The remaining challenge is the procurement and management of such an approach to building training simulators.

ADOPTION CHALLENGES

The innovative approach of domain engineering's product line focus frequently runs into business process barriers that are not prepared for, willing, or able to cope with a product line concept. The toughest of these is existing procurement models. A typical software development is part of a larger system procurement where the customer wants to buy a product that performs useful work. The customer lacks either the skills, the resources, or both to develop the product themselves so they elect to buy it. Since the customers are purchasing a complete product, the software is but a single component of the system. Customers do not care how the system is developed. They want a system that meets their functional and operational requirements within the schedule and budget specified. Regardless of its potential for reducing costs on future projects, customers have no interest in (and in the case of government procurements are legally prohibited from) paying engineering costs associated with the development of a product line domain which has a limited direct contribution to their specific product. Customers buy products, they do not buy product lines. So how do we sell a product line development?

Return-On-Investment

Software product line development will not be adopted unless it can demonstrate a positive return on investment. Therefore it is necessary to prove the investment will pay off in a specific and finite period of time. The Navy/STARS project has gone through many phases of redirection and goal setting. This case study is built upon the work commencing in January 1995, i.e., the development of a Flight Dynamics subdomain that will allow application engineering to specify a product instance that faithfully models the flight characteristics of an aircraft. The only product of this application engineering effort is source code ready for software integration with the rest of the simulation of the T-34C FIT which is being developed using traditional methods.

Megaprogramming Cost Estimate: We have estimated a cost of 46 labor months for development of a Flight Dynamics subdomain that supports the T-34C FIT, among other aircraft. This estimate is based on the metrics gathered by this team during the first of six planned iterations through the two life cycle model, and projections made by the Navy/STARS technical team for the remaining three

iterations. As of this writing, the estimate has proved accurate against the metrics that are now complete for four of the planned six iterations. The metric data is gathered on a weekly basis from files created by the team members using a X-windows tool built by the project team. Each primitive task in the work breakdown structure is assigned a metric category. As a person works the task they record their time spent in the appropriate metric category.

Since STARS is a research and development project we carry overhead burdens for tasks that would not normally be part of a true domain organization. Examples of such tasks are the contract deliverable experience report and software product line development demonstration as well as development of the Software Engineering Environment used to support the two life cycle development model. Therefore, these costs are not utilized in the foregoing estimate. We should also note that 90% of the foregoing estimate is domain engineering, with the remainder being application engineering.

Conventional Cost Estimate: We have estimated a cost of 35 labor months for conventional development of the T-34C FIT Flight Dynamics software. The conventional estimates were created using DoD`s validated COCOMO model. Note: neither the conventional or software product line development costs estimates consider maintenance costs.

Analysis: Of course, the objective is not to compare the costs of single system development, but to compare the costs over a family of systems. Suppose a series of three training system software developments, each within the context of the AVTS domain and roughly the size of the T-34C FIT Flight Dynamics.

The conventional development cost is easy to estimate. It is simply three times the estimated cost of one system. We should note that we assumed that the conventional software development organization was a mature organization, with substantial experience in this domain for the COCOMO estimate.

The software product line development estimate is more problematic. Ideally, the only repeated cost would the an additional application engineering effort, but this is unlikely to be the case. More probably, the domain will support some but not all of the needs of the next application, thereby requiring additional domain engineering work. In the best case, the total additional work would be about 20% of the initial investment split equally between domain engineering and application engineering. The 20% number derives from the sum of the application engineering effort and our experience in using a domain for another application. We project a middle case of 40% additional effort, and a worst case of 60% additional effort. Notice that the worst case assumes that we are unable to reuse more than half of the original domain engineering work on the next application. Figure 4 illustrates how these costs accumulate over a series of three trainers.

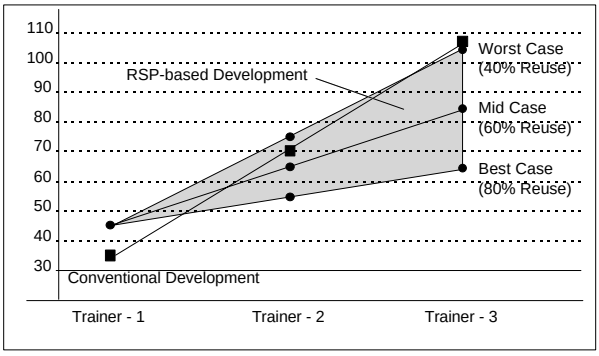

Figure 4: Cumulative Costs in Labor Months

Obviously, any analysis to predict cost to develop over a series is of simulator developments is just an estimate, with a probability or likelihood of correction associated with each estimate. Figure 5 shows the different costs estimates plotted against their probability of occurrence, for a total of three training systems. Inspection of this figure indicates that there less than a 15% chance that the cost of three training software systems developed via software product line development would exceed the cost of the same systems conventionally developed. This indicates that the cost of conventional development is likely to be greater than development via software product line development, for product families of three systems or more.

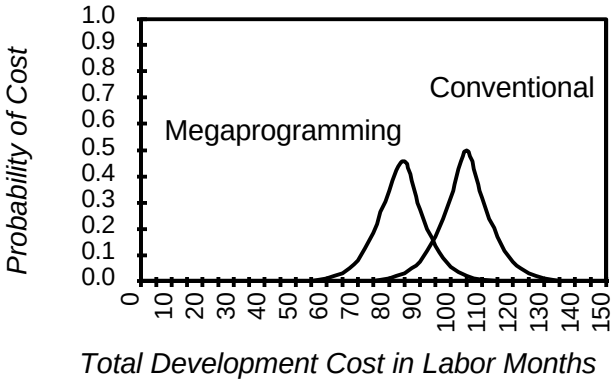

Figure 5: Probability of Cost for 3 Systems

Domain Ownership

Who decides on the evolution path of the domain, the level of adaptability supported, the provisions for interoperability with other software products, the

time and other resources invested in the domain? The *owner* of the domain. Obviously, there are only two candidates for ownership of the domain: the contracting agency or individual contractors. Ownership by individual contractors perhaps poses the lowest hurdle to adoption of software product line development, since seemingly all of the issues are internal to the organization. Certainly a contractor could simply decide to adopt software product line development for its internal business practices, dividing into domain and application engineering groups and proceeding according to the processes. In fact, most contractors already have an in-house business model for this approach in the separation of production from research and development. Domain engineering can be regarded as a particularly focused kind of applied research, whereas application engineering becomes a fairly straightforward production problem.

However, there are obstacles to ownership by individual contractors. For example, the domain engineering processes in their present form do not account for all of the products required by MIL-STD-498 or its predecessor DOD-STD-2167A. Therefore, contractors adopting software product line development and dealing with contracting agencies will have to accommodate these standards in one or more ways. The contractor may negotiate the format and content of some standard products. This approach would seem to be encouraged by MIL-STD-498, with its emphasis on defining and recording information without the emphasis on specific formats found in more traditional standards and data item descriptors [NEWBERRY]. Alternatively, the contractor could expand the domain generated products to account for the traditional documents, or require its application engineering group to create them for the generated instances to be delivered.

The biggest obstacle to adoption by individual contractors seems to be organizational inertia. There are few (if any) software development practices that are so persuasive that individual contractors will adopt them simply out of enlightened self-interest. Instead, most contractors have to be persuaded, by requirement or reward, to adopt a new approach. software product line development is no exception to this rule. Of course, the contracting agency could contractually influence contractors to adopt megaprogramming, but would that really constitute adoption by the contractors? A bargain at gunpoint is no bargain.

Even if individual contractors would adopt megaprogramming, this would probably not be in the best interest of the overall industry. After all, the objective driving the consideration of reuse intensive approaches is to reduce the cost of and improve the quality of trainers. Such a result would permit the contracting agency to continue expanding the role that simulators play in developing and maintaining a ready military force, as well as broadening the viable marketplace for simulation. Development of highly individualized and overlapping domains, maintained by separate contractors, and producing incompatible results is unlikely to achieve the objective. This approach would lead to the creation of proprietary, and therefore likely unusable, domains.

The preferred approach is ownership of the domain by contracting agency . Never the less, there are obstacles to contracting agency ownership. One obstacle may be the internal organization of the contracting agency or agencies. Just as there is almost never a single contractor on a training system development project, there is rarely a single government office involved either. For example, aircraft simulators for the U.S. Navy are built according to requirements developed at Naval Air Systems Command (NAVAIR), via contractual and engineering oversight at Naval Air Warfare Center Training Systems Division (NAWCTSD), for delivery to the aircrew training field sites ("school houses"). The Navy/STARS Demonstration Project is further complicated by contractual oversight being provided by Naval Information Systems Management Center (NISMC) and sponsorship by the Advanced Research Projects Agency (ARPA). So, there are three to five government offices, each with their own perspective on how the domain should evolve. Although this has been referred to with derision as "protecting rice bowls", in truth each of these government offices has an important and different role to play in developing a training system. While this organization can be rationalized, it is clear that it confuses the issue of who should own the domain.

A second problem with contracting agency ownership is the "color of money" problem, which is closely related to the organizational problem: Some contracting agencies simply do not receive the kinds of funds that could be utilized to develop or procure a domain. For example one might think that NAWCTSD, in its role of engineering oversight of simulator contractors would make a natural owner of the domain. This approach does not however take into account the fact that NAWCTSD actually receives a very small research budget, which most likely is the kind of funding best suited to performing domain engineering. NAWCTSD's budget is actually almost completely oriented toward procurement. The existing budget configurations actually indicate that the most likely scenario for ownership by con-

tracting agency is that NAVAIR would own the domain and NAWCTSD would own the application.

The Navy/STARS Demonstration Project has wrestled with the question of domain ownership throughout its existence. All of the models discussed above have been proposed and/or tried at some point. Our experience suggests that contracting agency ownership of the domain is the only path that will result in adoption of software product line development and the achievement of its goals of cost reduction and quality improvement.

Government/Contractor Roles

Ownership of the domain by the contracting agency does not necessarily mean that government personnel are actually performing the domain engineering function. It is likely that the contracting agency would utilize a number of contractors in support of domain and/or application engineering. Therefore, we should consider the different business models, and the resulting variety of roles for government and contractor personnel. There are three fundamentally different business models for software product line development, that divide a wide spectrum of subtly different models. The following discusses the advantages and disadvantages of each approach in turn.

First, the contracting agency could retain primary responsibility for performing domain engineering, while contracting out application engineering. There are several advantages to this approach. The contracting agency would develop a core engineering group that well understands the critical technology being utilized in its applications. Since domain engineering is best understood as a level of effort task, this approach would help stabilize the contracting agency's staffing levels. This approach would guarantee that the domain evolves and develops in precisely the manner the contracting agency foresees. There are two main disadvantages to this approach. The contracting agency may not wish to accumulate and retain the necessary subject matter expertise. This approach places the contracting agency in a much more pro-active role than has traditionally been the case, with a corresponding increase in responsibility for producing the trainer.

Second, the contracting agency could contract for domain and application engineering services. The advantages of this approach include: the ability to contract to different organizations as driven by the availability of subject matter expertise, a more traditional oversight role for the contracting agency, and increased competition. The disadvantages are: the increased risk of loss of control of the domain development path, an increased likelihood of the contracting agency failing to fundamentally understand the domain, and increased risk of loss due to the steep learning curve.

Finally, the contracting agency could participate on joint government/contractor development teams for domain and/or application engineering. If applied carefully, this approach can realize the advantages of the other approaches while avoiding their disadvantages. This has been the approach used on the Navy/STARS Demonstration Project after the Project Readiness Review. Our experience suggests that it is the most productive approach, particularly when the teams are collocated and dedicated full time to the project. We have seen very productive results when government personnel have worked in both domain verification/validation and in the iterative development of domain products. Substantially less productive results have occurred when government personnel (or for that matter contractor personnel) are part-time on the project. Use of part-time people very quickly begins to look like the second approach discussed, contracting domain or application engineering services, which we have found to be the least satisfying for the contracting agency's objectives. Joint government/contractor teams have been most critical in domain engineering; application engineering can easily be treated as a classic contracting agency oversight of contractor performance situation.

Identifying Suitable Domains

Successful application of software product line development requires a realistic assessment of the reuse potential within the domain of interest. The VCOE's Reuse Adoption Guidebook proposes a semi-formal method for assessing this potential. This model is designed to reduce the extensive commitment of resources needed for a detailed study of the domain.

The model has two parts - one to assess the potential of the domain to benefit from software product line development practices (the Domain Assessment Model) and the other which assesses the capability of an organizational entity to realize that potential (the Reuse Capability Model). The Reuse Capability Model will be discussed in the next section. Together these parts are used to develop an action plan by which an organization can develop the domain (and improve itself) to a point where it can successfully realize the benefits of software product line development.

The domain assessment process provides guidelines for identifying and quantifying factors that are known to limit or enable potential. The major attributes of a domain are grouped into the following analytic processes.

Forecasting the demand for domain assets: The potential for systems to be built in the domain is critical in determining whether the domain engineering investment should be made. Organization goals, current programs, past developments, technology trends, and budget forecasts are but a few of the indicators that can be used to estimate how many versions or models of a domain will be developed.

Understanding domain assets: The availability of domain expertise and existing software assets of high quality improve the probability that they can be reused, thus these factors can have a positive influence on domain potential. Also, from existing software assets and the staff responsible for developing the assets, an organization can derive much information to guide it in identifying the software functional areas that should be developed.

Assessing standardization in the domain: Standardization in a domain controls variation and improves the predictability of the context for use of an asset, thus increasing the reusability of an asset that conforms to a standard. A domain's potential improves with increasing levels of standardization. Levels of standardization range from none (ad hoc approaches to problem specification and solution) to full definition and recognition by standards organizations.

Assessing stability in the domain: Stability is an indicator of the rate of change of needs and technology in a domain. In stable domains, an organization is better able to predict future developer and end-user needs. Stability does not mean that there is no variation in the components in the domain, but rather that the variations are predictable.

Assessing the commonalties and variations expected in future applications: The nature of the domain's commonality and variability are the primary factors that lead to the feasibility of creating reusable assets that allow a variety of systems to be built. Without commonalties, reusable components cannot be built. But if components do not accommodate legitimate variations in the needs of end users, they may not have a large enough customer base to justify their development as reusable assets.

An example from the demonstration project is helpful in understanding the assessment. 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. The AVTS domain is partitioned into thirteen subdomains; eight are the focus of the demonstration: Flight Station (FS), Flight Controls (FC), Flight Dynamics(FD), Propulsion (PRO), Navigation/Communication(NAV), Instructor/Operator Station (IOS), Environment (ENV) and a support subdomain (TASS); and five that are deferred for later development: Weapons (WPN), Radar (RDR), Electronic Warfare (EW), Physical Cues (PHC), and Visual (VIS). Figure 6 illustrates the resulting domain assessment profile for the AVTS domain.

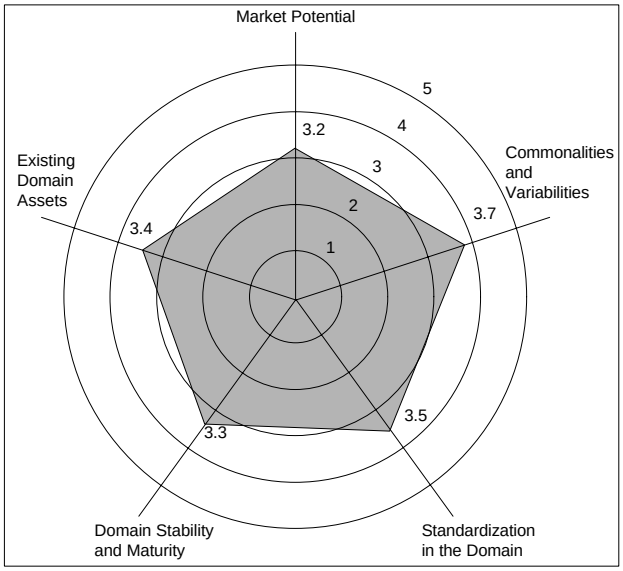

Figure 6: AVTS Domain Assessment Profile

A brief discussion of the interpretation of assessment scores is in order. The range of any single factor is from one (1) to five (5); the overall score for a domain therefore ranges from five(5) to twenty-five (25). Higher scores indicate that application of the Reuse-Driven Software Processes to that domain is more likely to produce satisfactory results for the organization than application to a lower scoring domain. The best use of the scores is for comparison between alternatives, rather than simple inspection of the raw assessment scores.

Figure 7 provides the basis for a simplified cluster analysis of the results. The values shown here are the sum of the scores in all factors for each subdomain. Although the sample size of eight subdomains is hardly sufficient to apply assumptions of normality, we provide the mean and standard deviation of the sample. In terms of number of standard deviations from the mean, the sample neatly divides

into two clusters: subdomains *less* than one standard deviation from the mean include FD, ENV, TASS, and FC; subdomains *more* than one standard deviation from the mean include IOS, NAV, PRO, and FS. We might characterize these as "high" and "low" potential groups respectively. A more subjective analysis based on inspection might suggest three groups: a "high" potential group (FD, ENV, and TASS); "medium" (FC and IOS); and "low" (NAV, PRO, and FS). This later grouping was more intuitive to the domain experts we consulted.

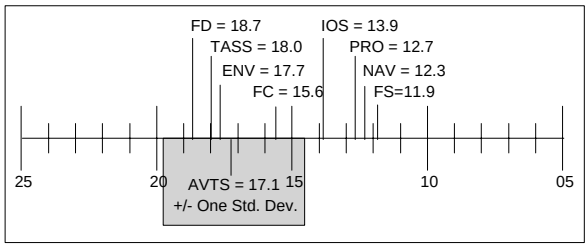

Figure 7: Cluster Analysis

The domain assessment provides at least three suggestions for selecting a set of specific subdomains for application of RSP. First, one might select the highest scoring subdomain (FD). Second, one might have select one from the highest group or cluster (FD, ENV, TASS, and perhaps FC). Finally, one might select one from each cluster to evaluate a range of possibilities (e.g., FD, FC, and FS). The selection should be driven by the needs and objectives of the project. In the case of the Navy/STARS Demonstration Project, we selected the highest scoring domain because of our resource constraints.

Building Adaptable Assets

Fundamentally, a domain engineer is one who enacts the domain engineering process. But on a deeper level, inspection of the process definition indicates that domain engineers assume three different roles while enacting it: (a) domain architect, (b) knowledge capturer, and (c) process provider. Each role requires a different perspective. Structuring the domain involves defining a set of strategies or ground-rules which define a context for work in the domain. These strategies express a long range vision for the domain, which supports the variety of different domain instances within the domain evolution plan. This role requires the domain engineer to apply systems and architectural expertise. Capturing domain knowledge involves the creation of domain components (source code, documentation, procedures, ...) that fit into the domain strategy. This role requires the domain engineer to apply substantial subject matter expertise while creating domain components, as well as supporting automated process support. This role requires the domain engineer to apply substantial process engineering expertise.

In all of these roles, the goal is creating domain components that easily adapt to the differing requirements of various domain instances. This goal shapes the form of each role of the domain engineer. The role of structuring the domain is not adhering blindly to traditional design strategies, but searching for the best strategy that supports variability. The role of capturing domain components is not optimizing individual components, but building components that are easily adapted. The role of providing access to the domain is not simply making CASE tools available, but integrating CASE into the domain engineering process. Neither is it simply providing a library of components, but mechanizing their selection, adaptation, and use.

ADOPTION IMPLICATIONS

Organizational Implications

There are two significant implications for organizations that want to adopt software product line development. First, the organization must establish that it is ready to adopt software product line development. Second, the organization must be prepared for the changes that will occur in the process of adoption. Perhaps an analogy from sports would illuminate these two implications. Considering this first, one role of the coach of a sports team is to decide when his team is ready to learn a new technique. One does not teach pro set offenses to a junior high school team -- they simply are not prepared to utilize the information! Considering the second, a coach must also realize that introducing a new technique will fundamentally change the way the players perceive, act, and react to the game. For example, when a team changes from a style of offense to another, the behavior of players and the quality of their performance will change.

How can we tell if our organization is ready for adopting software product line development? And if its not, what can we do to get there? The determining factor in the ability of an organization to realize the benefits of software product line development is its capability to accurately define, faithfully perform, and continuously improve appropriate reuse-based software development (and maintenance) processes. The Reuse Adoption Guidebook [SPC92] provides a technique for measuring an organization's readiness for advanced reuse techniques in its Reuse Capability Model (RCM). Analysis of the re-

sults of exercising the RCM can assist an organization in defining goals and developing strategies for overcoming identifiable weakness in the organization (opportunities for improvement) that prevent performing at the desired level.

As was the case with the Domain Assessment discussed earlier, the RCM is a self-assessment based on the organization's weighted level of agreement with a set of assertions. Each assertion states a reuse objective for the organization. For example, the first assertion is *Developers are aware of, can find, and have access to any relevant usable assets and external sources of assets identified by the organization*. The RCM provides sixty assertions, grouped into four major categories: application development, asset development, management, and process & technology. Figure 8 illustrates a fragment of the resulting organization reuse capability profile (note: the degree to which a square is filled in is the degree to which that objective is achieved).

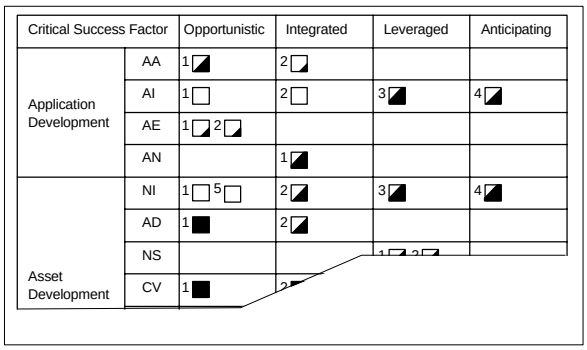

Figure 8: Reuse Capability Profile

Use of the model requires an organization to take a comprehensive view of its process for software development with respect to reuse. While we found the RCM not completely satisfying, it did help us determine which of the reuse capabilities needed to be improved in order for the project to achieve its goals for operational reuse.

Assuming that an organization is ready to adopt software product line development, what changes will occur as it does so? The first change will be the re-orientation toward an iterative approach away from a waterfall development cycle. This will require greatly increased discipline to iterate through a process to make incremental expansions to the domain's capability. Most organizations are geared to a single shot approach to building systems, but central to software product line development is the principle that the domain should be carefully and thoughtfully expanded, over a relatively long period of time, and (even stranger) over many individual projects (i.e., applications).

A second change will be the different nature of the technical staff. The organization will require a core technical staff, intimately familiar with the process and the domain to date, but not necessarily experts on the particularly domain expansion in progress. This core team would be retained for the life of the domain. In the case of simulation, the core should be individuals with a broad background in many different aspects of simulator software development, and a variety of engineering disciplines. The organization will want to augment the core with the best available subject matter experts relevant to the domain expansion in progress. The organization will frequently find that the subject matter experts will be drawn from a central technical staff (in large companies) or hired as consultants. These subject matter experts are not permanent members of the domain engineering team. The application engineering organization will continue to look like a classic simulator development team, with somewhat less emphasis than usual on software engineering since the domain engineering team really functions as a fully qualified supplier of software to the application engineering team.

Technical Implications

There are technical implications for a project when it chooses to utilize assets from domain engineering. The project creates an instance of the domain to fulfill certain requirements. These assets include documentation, designs, and software in the case of our demonstration. But along with the promise of leveraged reuse, there are certain restrictions that are placed on the project by the domain. A look at our demonstration can provide a couple of good examples; the implications of architecture and simulation strategies.

The central empowering concept in the domain is the insistence on a strong architecture. Only through such an architecture is leveraged reuse even possible, however, it does create specific limitations. The architecture for the Navy/STARS demonstration project is expressed in a structural model. The structural model is the framework through which components, attributes, and inter-relationships within the system are expressed. The structural model enforces a consistency in the software structure, thus aiding understanding.

An architecture, as we intend to use the term, consists of (a) a partitioning strategy and (b) a coordination strategy. The partitioning strategy leads to dividing the entire system into discrete, non-overlapping parts or components. The coordination

strategy leads to explicitly defined interfaces between those parts. These two strategies provide an engineering approach to bridging the gap between the system as a whole (as represented by its specification) and the design (the plan to build the product from primitive parts, such as computer instructions, metal struts, and switches). The reach of an architecture can extend from a single system (an architecture that solves a unique problem) to an entire family or product line of systems. In the latter case, once the partitioning methods and coordination rules are determined, multiple products can be generated using the same methods and rules.

The implication of the domain architecture is that all the assets exist as an expression of this architecture. Functionality is allocated according to this strategy as well as the interfacing scheme. A program that utilizes domain assets is required to adhere to this architectural approach. Thus if other parts of the software are produced by classical methods, it must follow the same standard architecture.

Another element of our technical approach that addresses the domain integration problem is our adoption of simulation strategies, which guide the design and implementation of simulation models. These strategies capture the design decisions, which permeate throughout the simulator. The strategies create a common vision in every domain engineer's mind on how ubiquitous simulation problems will be addressed. Furthermore, they tend to reduce the number of non-essential variations to which the domain would otherwise have to adapt. The simulation strategies identified so far include: notification of electrical power, notification of malfunctions, assessing ownship damage, tracking real versus computed values, and tracking commanded versus current values. These strategies are followed throughout the simulation software. The implication is the same as realized with the architecture; by utilizing the domain a project is accepting these strategies and agreeing to follow them throughout their product.

CONCLUSION

Software product line development holds much promise for reducing the cost and improving the quality of software development, but there are important implications for the training systems community if this is to occur.

First, the community must be prepared to plan in product cycles of approximately three systems, to realize a positive return on investment. While our experience suggests that the cost of developing the software for one training system via software product line development is very near the estimated cost for conventional development, there is less than a 15% chance that the cost of software for three training systems developed would be greater than the cost for conventional development.

Second, if software product line development is to have its maximum impact, the contracting agencies will have to step up to a greater involvement in product development. Our experience suggests that the best arrangement is ownership of the domain by the contracting agency. Furthermore, we believe that strong hands on involvement by government personnel in the domain engineering process is the most productive arrangement.

Finally, the adoption of software product line development will have important technical implications. Megaprogramming makes the existence of a strong, pervasive architecture for the product family critical for success. More subtly, the technical staff will have to articulate and enforce strategies throughout the system (such as malfunctions) -- strategies that are not directly enforceable by language features.

It has been said that a programming language that does not effect the way you look at a problem is not worth having. Certainly, as software product line development becomes more widespread in the community, is going to change the way we look at problems and opportunities. Our hope and expectation is that megaprogramming will make practical of system complexity and quality, here-to-fore out of reach.

REFERENCES

[NEWBERRY] "Changes from DOD-STD-2167A to MIL-STD-498", Major George A. Newberry, USAF, *Crosstalk -- The Journal of Defense Software Engineering*, April 1995.

[VCOE93] Software Productivity Consortium Services Corporation. 1993. *Reuse-Driven Software Process Guidebook.* SPC-92019-CMC. VCOE. Herndon, VA.

[VCOE92] Software Productivity Consortium Services Corporation. 1992. *Reuse Adoption Guidebook.* SPC-92018-CMC. VCOE. Herndon, VA.