SCALABLE FIDELITY COMPONENT-BASED SIMULATION SOFTWARE

Don Procuniar, L3 Communications Link Simulation & Training, Arlington, Texas
Paul E. McMahon, PEM Systems, Binghamton, New York

ABSTRACT

Scalable Fidelity Component-Based Simulation Software (SEFCOBSS) is an approach for taking existing legacy systems and adapting them to work with new, as well as, other legacy software systems. The SEFCOBSS method challenges a number of traditional notions of software development and maintenance.

Over the past 30 years the U.S Government has made a significant investment in varying fidelity real-time simulation systems. Until recently, it had been believed that many of these legacy systems were on the tail end of their life cycle and would need to be redesigned/rewritten to be effectively utilized. SEFCOBSS encompasses the principles, methods, architecture, and guidelines necessary to identify and effectively adapt legacy component software to meet a broad range of fidelity training needs.

Examples from the AVCATT-A project, where the SEFCOBSS method was first employed are included. AVCATT-A employed legacy high fidelity man-in-the-loop flight simulation software adapted for use in a lower fidelity collective training environment.

Key to the SEFCOBSS method is the ability to isolate legacy components from other legacy systems, as well as from newly designed software, while at the same time supporting effective communication among these systems. Fundamental principles of the supporting SEFCOBSS architecture are described.

The paper asserts that legacy systems can be cost effectively maintained, but to do so requires a well-defined and disciplined process that must include guidelines for legacy candidate selection and management. Included in the paper are guidelines for candidate selection, as well as guidelines for design modification, and software verification.

This paper tells you what you need to do to effectively leverage your legacy simulation assets when faced with new simulation requirements and/or changing fidelity requirements. This paper also dispels the traditional software myth that old legacy software is too expensive to use and maintain. References to previously published work that support SEFCOBSS are provided.

Author Biographies

Don Procuniar is the Director of Software Engineering for L3 Communications, Link Simulation & Training. Prior to his current position Don was the Software Lead Engineer for AVCATT-A Aviation Reconfigurable Manned Simulator Program. He has been employed at Link since 1980 holding numerous Software, Systems, Integration and Program Engineering positions over that twenty-year period. Donald has vast experience in large scale complex man-in-loop real time simulations including NASA Space Shuttle Mission Simulator, numerous F-16 training platforms, B-52 WST, C-130 ATS and AVCATT-A.

Paul E. McMahon, Principal, PEM Systems, provides technical and management leadership services to large and small engineering organizations. He has taught Software Engineering as an adjunct at Binghamton University, conducted workshops on Engineering Leadership, published over twenty articles, and a book on collaborative development entitled, “Virtual Project Management: Software Solutions for Today and the Future.” Paul held the position of Deputy Software Lead Engineer on the L3 Link AVCATT-A Project and led the AVCATT-A Architecture Team in support of the Project Software Lead Engineer.

INTRODUCTION

Scalable Fidelity Component-Based Simulation Software – or SEFCOBSS for short— is an approach for taking legacy systems and adapting them to work with new, as well as, other legacy systems. It is an approach to breath new life into perceived outdated simulation systems. Until recently, it had been believed that the cost to adapt and maintain legacy systems to meet new customer requirements would be prohibitive. Experiences working with SEFCOBSS indicate that to be cost-effective legacy systems must comply with the SEFCOBSS architecture criteria. Employing SEFCOBSS has also demonstrated that when key principles are followed-- including disciplined legacy product evaluations-- the life expectancy of many software simulation systems can be extended beyond what was once thought to be reasonable. AVCATT-A employed SEFCOBSS to create multiple unique helicopter simulators utilizing 1.5 million lines of code in a mix of new and reuse from over eight (8) different legacy systems. Today, many of the principles, methods and guidelines of SEFCOBSS are being integrated into the standard software development processes at L3 Link Simulation & Training.

WHAT IS SEFCOBSS AND WHY IS IT IMPORTANT?

SEFCOBSS is an architectural design philosophy, but it also encompasses a set of five (5) key steps to implement its architecture. The SEFCOBSS approach was first employed by L-3 Link within the Flight Simulation domain.

Many legacy Flight Simulation systems were built on full or near full fidelity models. Today, due to budget constraints and rapidly changing world conditions, there exists a greater demand for reconfigurable, focused, task-based training devices. In this global environment, different fidelity models are needed for a variety of different types of simulations. Instead of creating new simulation solutions for each of these types, SEFCOBSS provides the capability to utilize an existing known fidelity model to create a single reusable component, which meets a variety of the different fidelity needs.

The five (5) steps of SEFCOBSS include: Component Selection, Component Isolation & Architecture Porting, Environment Porting & Retesting, Design & Implementation of Fidelity Changes, and Software Verification. Each step is discussed in greater detail later in the paper.

The SEFCOBSS architecture can be used across a broad range of project types from Engineering Change Proposals to existing devices through brand new devices. It also allows a mix of both new and legacy software from multiple sources to run in a single environment. The philosophy of SEFCOBSS is to "Start Integrated, and Stay Integrated" through a spiral development process where each of the identified spirals focuses on a planned set of functionality inside carefully pre-screened software components. As used here the term component implies a standalone, isolated, and testable "chunk" of software.

While the work described in this paper is based on experiences encountered in the flight simulation domain, the need for scalable fidelity simulation software is not limited to this domain. As an example, currently there exists increasing interest in employing modeling and simulation in the acquisition of new weapons systems. One challenge to effectively utilizing simulation techniques in the acquisition process is the interoperability of legacy models developed under diverse architectures and environments [1]. Often, it is found that candidate legacy software models become less attractive when fidelity modification costs are considered. It is believed that the principles and methods of SEFCOBSS may hold a key to the interoperability cost and schedule challenge.

It is also worth noting that interest in varying fidelity simulation models is not limited to the DoD. As an example, today the pharmaceutical industry is utilizing simulation to model new drugs to help in creating those drugs [1]. Clearly, the demand for a systems solution to scalable fidelity simulation software, such as SEFCOBSS, has never been greater.

MEANING OF SCALABLE FIDELITY & SCOPE OF SEFCOBSS

Historically, simulations have been classified into the three areas of Live, Virtual, and Constructive. These classifications identify the degree of human and equipment realism. Live simulations refer to real people operating real systems. Virtual denotes real people operating simulated systems, and constructive refers to simulated people operating simulated systems. [1]

When we use the phrase "***scalable fidelity***" we mean the capability to vary the fidelity of individual simulation sub-systems within a controlled simulation environment.

Today SEFCOBSS is primarily concerned with controlling the fidelity of simulated sub-systems within a Virtual Simulation Environment interoperating with constructive simulations, as well as other virtual and live simulations. AVCATT-A, for example, encompasses both Virtual and Constructive elements, and the system interoperates with external virtual and constructive simulation systems such as Close Combat Tactical Trainer.

THE AVCATT-A SCALABLE FIDELITY CHALLENGE

US Army's Aviation Combined Arms Tactical Trainer—Aviation Reconfigurable Manned Simulator (AVCATT-A) is a networked system of systems providing combined arms training through six (6) reconfigurable manned helicopter simulators interoperating in a simulated battlefield environment. The system is housed in two trailers, and includes both a Battle Master Control and After Action Review Station. The six Manned Modules simulate pilot/copilot/gunner positions for UH-60A/L Blackhawk, CH-47D Chinook, OH-58D Kiowa Warrior, AH-64A Apache, AH-64D Apache Longbow, and the RAH-66 Comanche.

Prior to the AVCATT-A Request for Proposal, the Simulation, Training, and Instrumentation Command (STRICOM), conducted an extensive Fidelity Analysis to determine the device's requirements. Higher fidelity manned module simulation devices had previously been developed and deployed in support of pilot training needs. AVCATT-A was required to be a mixed fidelity set of devices deploying high fidelity "Shoot, Move and Communication" systems while providing low fidelity capabilities in the other helicopter systems. The challenge faced on AVCATT-A was to find the most effective way to leverage the existing higher fidelity legacy simulation system assets in meeting the US Army's combined arms training needs.

OVERVIEW OF PAPER & RELATION TO CURRENT INDUSTRY SOFTWARE INITIATIVES

In the following paragraphs the SEFCOBSS Architecture, along with its underlying principles, and methods, is discussed. Guidelines for legacy candidate selection, design modification, and software verification, are included.

It is also worth noting that a number of the principles and methods employed by SEFCOBSS exhibit similarities to what today is referred to as Agile Software Development, or "lightweight" methods [2, 3]. Agile Software Development is a methodology that has grown largely out of the needs of small software development organizations where "*process*" is desired, but only if it is *flexible* and *efficient*.

A flexible and efficient process has also been found to be critical to SEFCOBSS, particularly due to the constraints imposed by existing legacy models and associated processes. A comparison between SEFCOBSS and Agile Software Methods is provided.

Inherent within SEFCOBSS are key architecture compliancy rules, along with process flexibility. The flexibility of SEFCOBSS is discussed in the section on "Tailoring SEFCOBSS for New Applications."

STEP 1: COMPONENT SELECTION

Initial Component selection involves an analysis of the project requirements versus existing legacy component capabilities. This activity includes an assessment of the functionality, quality, and pedigree (history of use) of candidate components. This initial component screening is accomplished in preparation for component porting activities. Component evaluations should consider: ease of separation/containment; fidelity of model versus required fidelity; requirement match; language/operating environment of candidate software; and availability of specific test cases/procedures. Candidate legacy components undergo both a SEFCOBSS Architecture porting activity and an Environment porting activity. These two important porting steps have been separated to aid in managing defect injection.

Following are five (5) guidelines that can be used in establishing your own legacy assessment process:

- Include architecture & design criteria
- Include documentation & code assessment criteria
- Include test criteria
- Include configuration management criteria
- Include metrics

STEP 2: COMPONENT ISOLATION & SEFCOBSS ARCHITECTURE PORTING

Once a candidate component has been selected it is then isolated within its legacy environment and ported to the SEFCOBSS Architecture. In cases when the legacy environment is not available the architecture porting step is conducted in the chosen target environment. Oftentimes when dealing with legacy systems, engineers will want to redesign major sections of the system. It is not uncommon for good engineers to

desire to rewrite legacy code rather than reuse it. However, our experience indicates that this type of activity should be managed closely. Changing the legacy code introduces significant risk to cost and schedule due partly to the fact that the originator of the legacy code is rarely available. Past experience shows that very few legacy systems have remained in their original designed state, many modifications have been made without redesign, and the documentation that exists is frequently not in sufficient detail to describe all the functionality as it currently exists. These same legacy systems, however, have been deployed without error for many years, and our experience indicates that as long as they are not modified they can function error free in the new environment.

Background of Component-Based Software

One of the major challenges faced in establishing a cost-effective approach to adapting legacy systems is the simple fact that most of these systems were designed to work in a specific environment each with their own set of unique constraints. Examples of such constraints include operating system, software language, compiler, hardware platform, and the legacy system software architecture.

The phrase "***Component-Based***" refers to an approach to build large software systems by integrating previously developed software components. These systems can include a mix of both commercial and non-developmental items. Previously published literature indicates "the degree to which a component's internal structure is accessible suggests different approaches to adaptation." [4]

Three approaches have been identified:

- White box: This approach allows the component to be significantly redesigned/rewritten to operate with other components
- Grey box: This approach doesn't modify the actual source of the component, but utilizes an Application Programming Interface (API) provided by the component to essentially extend the components features
- Black Box: This approach uses the component as is with no API and no changes to the actual source code

Historically, experience has shown that white box approaches can result in serious maintenance problems as often the complexity and size of the planned changes are underestimated. One approach to minimize this risk is using what is referred to as "wrapper" software.

What Do We Mean by "Wrappers"?

"***Wrappers***" are specialized software elements that act as middleware between disparate legacy entities. "Wrappers" could be considered a form of API as identified in the "grey" box approach.

Why We Refer to SEFCOBSS as Component-Based

We refer to SEFCOBSS as a "Component-Based" software approach because it employs a set of well-defined *Wrapper* elements to isolate legacy components, while at the same time allowing these components to effectively communicate. See Figure 1.

The wrapper elements are discussed in greater detail in the following paragraphs. To help comprehend what SEFCOBSS Architecture Porting entails, a brief history of a flight simulation software architecture is provided.

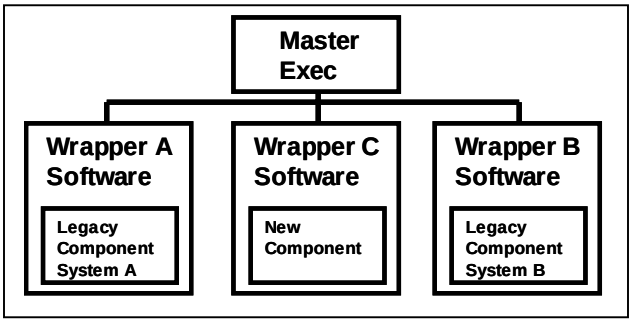

Figure 1 *Component-Based* Architecture Employing "*Wrapper*" Elements

History of L3 Link Flight Simulation Software Architecture

Since the early 70's it has been recognized that there exists a number of recurring software design patterns within the Flight Simulation product line. Examples of such patterns include:

- Methods to activate and communicate with real-time simulation modules
- Methods to communicate and implement simulated malfunctions
- Methods to modify and display simulation parameters.
- Methods to control and communicate modes and states such as freeze, reset, initialization, run, and playback.

Because these patterns frequently recurred, rules were developed, documented and provided to simulation engineers as guidance to simplify their engineering

simulation tasks as they developed simulation models in the past. The rules that were applied to implement the design patterns were simple, but, more importantly, they were leveraged again and again across many legacy flight simulators. The identified patterns, together with their associated implementation rules, embodied the software architecture for many of today's legacy simulation systems. This legacy simulation software architecture has been described in greater depth in previous published literature [5].

Fundamental Principle of the SEFCOBSS Architecture

Fundamental to the SEFCOBSS Architecture is the establishment of concise rules. For example, SEFCOBSS provides precise porting and fidelity scaling rules for legacy software. Examples of fidelity scaling rules include how to disable and enable simulated malfunctions, and how to disable and enable instructor station editable parameters. Fidelity scaling rules also include steps to enable and disable the processing of discrete inputs, and state-specific logic.

Many of the SEFCOBSS rules are complementary to the rules that were used by simulation engineers when originally designing today's legacy simulation systems back in the early 70's and 80's. It is important to note that without the structured architecture of the flight simulation product line, the fidelity scaling rules, which today are relied upon within SEFCOBSS, would have been much more difficult to develop.

SEFCOBSS Architecture

One of the key challenges faced with legacy systems is the potential high cost of change activity to meet new requirements. The SEFCOBSS method addresses this challenge by minimizing legacy software changes through the use of "Wrapper" elements. The "Wrapper" software includes three key elements:

- Component Executive
- Import Interface Agent
- Export Interface Agent

Fidelity Scaling Through Import/Export Interface Agent

A core principle of SEFCOBSS states that, "legacy system change activity should be minimized." By minimizing legacy changes, SEFCOBSS reduces the risk of cost and schedule overrun due to unanticipated system breakage. At the same time, minimizing change also increases our opportunities for future fidelity scaling. The Import and Export Interface Agents are key wrapper elements in support of this core principle.

Within the SEFCOBSS architecture, legacy software continues to operate internally just as it did within the legacy environment. This is accomplished through the use of a local data area structured identically to the legacy environment itself. This approach allows the legacy software to effectively run in a "black box" mode minimizing the chances of software breakage caused by new requirements.

It is important to note that within the SEFCOBSS Architecture, legacy components may continue to compute results that are not required within the new simulation environment. This is where the importers and exporters come into play. Importers are used to supply external interface data to the local data area where the legacy system acts upon it just as it did in the legacy environment. This technique may also be employed to disable logic that is not required in the new environment. Examples include disabling malfunctions, or discrete input (cockpit switches) processing logic. The Exporters move legacy system computed results out to a Shared Memory areas where the results can be accessed by other Importers.

Fidelity Scaling Through Component Executive

The Component Executive provides a second method to scale fidelity. The Component Executive acts as a middleware element between the MASTER Executive and the legacy system. This Wrapper element allows complete sub-systems or portions of sub-systems (modules) to be turned on or off. Another option provided by the Component Executive is the execution of sub-systems, or modules at lower rates. This wrapper element provides another method to scale fidelity while meeting our core principle of minimizing change to the legacy software itself.

Two (2) Key Attributes of Wrapper Software

The wrapper software supports two key attributes of the SEFCOBSS architecture. First, by minimizing the actual changes to the legacy software itself, we also minimize the need to re-test previously verified software. This attribute supports reduced cost and schedule.

Second, the wrapper software isolates legacy components. This key attribute allows legacy components that were originally developed using disparate methodologies and programming languages to coexist. The Exporters and Importers route data among

the components thereby providing consistent and reliable communication.

It is an outdated belief that software developed using different methodologies is at risk of not being able to communicate. This is only true, if you haven't established a sound architecture, such as SEFCOBSS, providing rules to ensure reliable communication. SEFCOBSS supports a hybrid of methodologies (object-oriented, structured analysis) and languages (C, C++, Fortran, Ada83, Ada95) because it was built specifically to do so.

STEP 3: ENVIRONMENT PORTING & RETESTING

Environment porting includes activities associated with moving the software to the target operating system, compiler and platform. This porting step has been separated from the SEFCOBSS Architecture Porting (development & testing of candidate component with SEFCOBSS wrapper elements) to manage potential error injection.

More Than Just a Technical Architecture

It is important to recognize that SEFCOBSS is far more than just a Technical Architecture. In order for SEFCOBSS to operate effectively a disciplined development process with precise rules must be followed. Those rules include the legacy candidate selection process. To appreciate this process within SEFCOBSS requires an understanding of how SEFCOBSS views code.

SEFCOBSS and Code

The SEFCOBSS Method has been referred to as a "Code-Focus" method, but this should not be misunderstood as coding before design, or coding before requirements.

Code is viewed within the SEFCOBSS method as a high valued legacy asset. As such, its value is crucial in establishing an objective and accurate assessment of the product. We have found that one of the best objective assessment methods of a candidate product is to execute its code, and measure the code's capability against the requirements. In support of this process, one of the early and crucial steps in the SEFCOBSS method is the porting of each legacy candidate to the chosen target environment.

Key Legacy Porting Rule

It is important to note that the SEFCOBSS method does not allow functional changes to the software as part of the porting activity. The only changes allowed are those architecture related modifications necessary to compile and execute the software within a SEFCOBSS-compliant architecture. Once the candidate component has been ported to both the SEFCOBSS Architecture and the target environment, a well-defined assessment process must be followed as discussed in Step 1.

The Criticality of the Development Environment and Target Architecture

To support a "code-focus" method such as SEFCOBSS, it is critical to get your development environment established and Target Architecture up and running as early as possible. This is necessary to support a code-executing legacy analysis process. It is also crucial that the Target Architecture include the actual chosen operating system, hardware platform, compilers and key tools that you plan to use in your delivered system.

Key to the SEFCOBSS method is getting your candidate software into your chosen environment to accurately assess required change activity. When this step is short-circuited, often we find that legacy products are selected under the false belief that the changes required to meet the new requirements will be small. Unfortunately, when this conclusion is not backed up by executing code within the chosen Target Environment it is frequently found to be erroneous.

STEP 4: DESIGN & IMPLEMENTATION OF CHANGES

Maintenance of Legacy Design Style

Once a legacy candidate component has been approved, design changes can proceed. We have found that the design process employed must be a disciplined one, but it also must be supportive of the special needs of legacy component software, and these needs often differ from traditional new development.

As an example, design, documentation and code guidelines should encourage the maintenance of existing legacy styles, rather than require legacy products to conform to the style employed by newly developed software. Rules for modifying legacy systems should be incorporated into your standard design process.

The Need for a Detailed Design Review

Frequently in a heavy legacy environment implementation changes will be small, but this doesn't mean you should waive the need for a detailed design review. Even if the legacy system meets 100% of the allocated requirements, it is important to conduct a detailed design review to verify key system requirements.

Following are five (5) sample checklist items indicating why a detailed design review is necessary in a heavy legacy environment:

- Documentation sufficient?
- System requirements met?
- Design complete?
- Test cases complete?
- Configuration management controls sufficient?

Keep in mind that even when legacy components appear to be perfect candidates to meet new requirements, oftentimes system level requirements and interfaces to other components that the system has not previously communicated with will require some level of design work.

Keeping with our core principle, the goal is to isolate as many, if not all, of this work to the wrapper elements. The primary focus of detailed design reviews for legacy systems may therefore become the wrapper software itself.

Manage Re-design Closely

In general, re-design should be discouraged because it is counter to the core principles and it effectively defeats the goal of maximizing the value of the legacy asset. The desire is to leverage all of the legacy systems code and all it's assets (i.e. test cases, testing, documentation).

When change to the original legacy component is required, the change should be made through the use of creating new software elements that provide that change without modifying the legacy component. This allows both the original and new capabilities to be tested and maintained. Any change that is not directly related to a customer requirement should be thoroughly analyzed. This topic is addressed further in the section on Verification.

STEP 5: SOFTWARE VERIFICATION

Once the design is complete, one can move on to the implementation and verification phase. It is important in the verification phase to maintain awareness of the fact that a goal of SEFCOBSS is to leverage more than just the code asset.

As an example, when the pedigree (history of use of product) of a legacy asset indicates that the component has been accepted and is being used successfully on one or more deployed simulation projects, it may be well worthwhile to leverage the test cases and test procedures for this product as well.

SEFCOBSS provides implementation rules that allow us to take advantage of previously tested and accepted legacy assets through a "black box" component level test philosophy. This approach reduces the cost associated with traditional new software low level testing.

COMPARISON BETWEEN THE SEFCOBSS METHOD AND AGILE SOFTWARE METHODS

Much has been written over the past few years on the subject of Agile Software Development. Agile methods, also referred to as "lightweight" methods, have recently appeared in industry literature primarily in response to the needs of small software development companies. Many of these small companies have traditionally operated in a "code and fix" development mode with very little formal processes and procedures.

Agile methods can be thought of as a compromise between the "code and fix", or "no process" approach, and the large company historical "too much process". [6] Key characteristics of Agile processes include:

- Code & Test Focus
- Continual Design Through Refactoring
- Pair Programming
- Continuous Planning and Integration
- Continuous Measurement

While the associated inefficiencies of "code & fix" are well known, many small companies also believe that the overhead costs of formal processes could not be tolerated within their cost constrained environments.

Clearly, due to communications needs, larger companies/projects need more formal processes. However, our experience with SEFCOBSS-- as discussed in the following paragraphs--may motivate some larger companies to take a closer look at what a Hybrid "Agile" Process could offer a large legacy-focused project within the context of a disciplined development environment.

CODE & TEST FOCUS

Similar to most Agile Methods, SEFCOBSS has a code and test focus, but the motivation for this focus within SEFCOBSS is different. Recall that SEFCOBSS was developed specifically to support scalable fidelity simulation software by leveraging existing legacy assets. Oftentimes, we find that legacy simulation software assets were originally developed to operate in environments quite different from current needs.

When it comes to evaluating the suitability of software assets, too often--in the past-- early evaluations have been found to be overly optimistic. This has been at least partially due to the lack of critical information necessary to make an accurate assessment. For example, analysis limited to documentation, or a desk-check of the code, can easily mislead when it comes to understanding the real value of a product.

Because the code already exists for legacy products, SEFCOBSS leverages this fact by placing high value on the evaluation of the executing code itself. Documentation is also evaluated, but the real value is based on what the product can actually do, rather than what it might be able to do in the future, or what its documentation claims it can do.

One of the concerns often expressed with this approach relates to the schedule time constraint. To address this concern, the SEFCOBSS method does not encourage spending a great deal of time studying the code. Rather the focus within the SEFCOBSS method is on getting the baseline legacy product running within the chosen target environment and testing it against its allocated requirements.

Before a single line of code is changed to add new functionality, the SEFCOBSS method requires that we know just what the product can and cannot do. This supports a more objective legacy evaluation process.

Because the baseline product is also controlled, the method also supports a more accurate measurement of change activity. For this reason, the SEFCOBSS method has been referred to as a "code and test" focus method, and a "continual measurement" method.

CONTINUAL DESIGN THROUGH REFACTORING

With respect to design, SEFCOBSS has both similarities and differences with Agile Methods. One of the well-publicized features of Agile Methods is Refactoring, or continual restructuring of the code to improve it. This thought admittedly scares many software managers in traditional large engineering organizations. It conjures up the notion that the code will never be quite "good enough" and the related fear of cost and schedule overrun.

Agile Methods in Small Organizations

In small organizations where Agile Methods are being deployed successfully, refactoring is usually implemented through direct programmer ownership of budget and schedule. For refactoring to succeed, programmers must be accountable for cost and schedule and self-manage any refactoring with respect to commitments.

SEFCOBSS View of Refactoring – "If it isn't broken, don't fix it"

SEFCOBSS takes a different perspective when it comes to improving the structure of the code and this perspective is derived from one of its core principles; that is the desire to leverage the maximum value from the legacy asset.

As previously discussed, SEFCOBSS views legacy assets as far more than just code. As an example, a significant legacy product value is found in the previous testing many of these products have undergone within their original environment, and often under the watchful eye of a customer.

Unfortunately, when you change the structure of the code you run the risk of invalidating previous testing. In this situation tests must be re-run to ensure functionality has not been compromised. This diminishes the core value of the legacy asset.

The SEFCOBSS approach to previously tested legacy software is a "black box" approach. SEFCOBSS uses "wrapper" techniques to transform the code into an executing system within the new environment, and then tests it at a "black-box" level to see what it can do. SEFCOBSS doesn't even recommend that time be spent understanding how the legacy component works "on-the-inside".

The SEFCOBSS counter principle to Refactoring is, "If it isn't broken, don't fix it." While admittedly there does exist some risk in this approach, it has proven to be a practical one in support of effective utilization of limited cost, schedule and personnel resources.

PAIR PROGRAMMING

Pair programming may be one of the most controversial topics found in Agile Methods. It is a

key principle of Extreme Programming (XP) [7], one of the better know Agile Methods. Pair programming requires two programmers, rather than one for each task. The rationale, as described by one programmer at a small company that uses XP is that "the dialogue" with the co-worker becomes invaluable in verifying the design and detecting defects early. [6]

Today, in many large organizations, formal peer reviews have been instituted to help detect defects early. While most believe that peer reviews have value, field reports of actual productivity gains resulting from peer reviews have been mixed [8].

Detecting Errors Early with the SEFCOBSS Method

While the SEFCOBSS method does not utilize pair programming, early defect detection is of paramount importance within the process. SEFCOBSS achieves this goal through a combination of early and continuous testing and mentoring and oversight provided by senior personnel. Formal peer reviews of code, test cases, and design artifacts are instituted with software leads empowered to make practical day-to-day decisions regarding who should attend which reviews, and how to most effectively apply the valued asset of senior oversight. For example, decisions on whether to send a senior engineer to a given peer review is made based on a number of factors including product pedigree (history of product), complexity of change, and track record of the assigned engineer. By employing such a criteria within SEFCOBSS, valuable senior talent is effectively utilized.

CONTINUOUS PLANNING AND INTEGRATION

When employing Agile Methods plans are usually of short duration (2-3 weeks), and re-planning is not discouraged. Historically, on many large projects, bulky software development plans have quickly become "shelf-ware". This is often caused by aggressive schedules that leave little time for maintaining verbose plans that can easily become out of date in a rapidly changing environment.

The SEFCOBSS Approach to Planning

The SEFCOBSS method is sensitive to today's cost and schedule constraints. In response the process encourages detailed plans of short duration (usually 1-3 months), although not as short as typical Agile Development plans. As an example, SEFCOBSS recommends that up front planning first focus on the development environment, and target architecture in support of legacy product porting and evaluation. This approach supports the baseline code-focus philosophy of SEFCOBSS. Once the baseline legacy products are ported and analyzed in accordance with the legacy candidate selection process, the next level of detailed planning can be put in place and managed more effectively. This approach also supports the development of more accurate schedules as greater understanding of the task exists.

CONTINUOUS MEASUREMENT

Continuous measurement in an Agile Development environment doesn't mean continually asking the programmer when they will be done. But it does mean gathering real project data that can provide real insight into project activities with the least interruption possible to the on-going effort.

Too often on large highly structured projects metrics plans are put in place, and data is collected, but the data simply doesn't get used in the day to day decision-making activities of the project. SEFCOBSS has inherent within its methods a standard set of metrics that are key to the on-going effort and effective day-to-day management. SEFCOBSS metrics include:

- Size
- Requirements
- Test
- Cost & Schedule
- Resources

Projects are encouraged to enhance this list with their own project specific metrics, but the core SEFCOBSS metrics are required and relate specifically to the status of the legacy products and the activities required to modify these products to meet new user needs.

Using Metrics to Manage

As the SEFCOBSS design process proceeds, engineers become increasingly familiar with their assigned software product. Initial estimates of lines of code are usually based on historical data and experience. By periodically updating the estimates, software lead engineers see trends early and initiate corrective action. As an example, a significant increase in the estimated new lines of a given sub-system may indicate greater complexity than originally thought. This could lead to the addition of experienced staff early thereby avoiding costly schedule impact late in the project.

SEFCOBSS AS AN AGILE METHOD FOR LARGE SCALE SIMULATION PROJECTS

As stated earlier, SEFCOBSS is more than a Technical Architecture. It is an approach for legacy systems that can be successful only when deployed together with its related processes and methods.

Some of the principles and methods within SEFCOBSS may appear to contradict traditional software engineering principles (i.e. focus on code early, testing before design). We believe this perception results from a misunderstanding of the fundamentals of good software engineering. A recent article by Mark Paulk, one of the coauthors of the Software Engineering Institute Capability Maturity Model (SEI CMM) indicates that Agile Methods are not counter to a disciplined process, but rather complement a disciplined process and the goals of the CMM [9].

Our experiences utilizing SEFCOBSS agree with this assessment and also go further to conclude that many of the Agile Methods, if used together with a sound technical architecture and related rules, such as what SEFCOBSS provides, can in fact work on large projects as well as small. This is due to the fact that an architecture such as SEFCOBSS supports the partitioning of a large development effort into smaller "chunks" where each can be managed through a "small team" philosophy. This subject was addressed in greater detail in an article published at the Software Technology Conference in 2001 by one of the co-authors of this paper [6].

TAILORING SEFCOBSS FOR NEW APPLICATIONS

While approaches to architecture and design require well-defined rules, a key to leveraging real legacy asset value is flexibility. Flexibility implies the need to be able to tailor one's approach to the specific requirements and constraints of selected legacy products and supporting organizations.

In this regard, we view SEFCOBSS as a framework [10], rather than as a final solution. The SEFCOBSS method, in effect, provides the starting point framework from which project unique tailoring is accomplished. As an example, we have found that an architecture criteria is essential and must be established early to succeed in a code-focused legacy asset environment. SEFCOBSS provides architecture criteria guidelines, but it doesn't dictate a single rigid criteria. SEFCOBSS Architectural guidelines are applied to each component supporting the right choice for that component. For example, the SEFCOBSS Architecture allows one component to be developed new (i.e. using Object Oriented methods), the next component to be reused without modification, and still others to be reused with modification following the legacy design style, all working together within one system.

SEFCOBSS Compliancy

While we do view SEFCOBSS as a framework, there also exists a "core" set of rules that must be met by each project to be considered SEFCOBSS compliant. For example, there are well-defined rules on how "wrapper" elements must be constructed to ensure components can be activated by the Master Executive and how they communicate with other components. These rules are crucial for baseline product porting and execution within the SEFCOBSS environment.

It is worth noting that the SEFCOBSS core does not dictate a specific documentation standard, but it does require core design artifacts and it does provide the guidance to tailor the process to meet your project's specific documentation needs.

OLD MYTHS DISPELLED

It is an outdated belief that legacy software will always be too expensive to maintain when used to meet new and modern requirements. It is an outdated belief that different methodologies and different languages can't coexist within a single cohesive and reliable architectural environment. It is an outdated belief that you must restructure poorly structured code if you want the product to be maintainable. It is an outdated belief that large projects cannot apply agile methods effectively.

Some of the methods of SEFCOBSS may appear non-traditional, but SEFCOBSS has proven to be a practical and effective method in a rapidly changing cost constrained world. What we have learned using SEFCOBSS on large efforts is not unlike what many small companies have recently learned on small projects using Agile Development Processes. While the SEFCOBSS method may appear non-traditional, it is not without discipline. SEFCOBSS supports the coexistence of discipline and agility through its Spiral-focused development process. In reality, SEFCOBSS represents the next step in disciplined process optimization for simulation projects.

CONCLUSIONS

Scalable Fidelity Simulation Software is in great demand today, but to achieve it requires more than just

software. SEFCOBSS encompasses the necessary architecture and rules to scale software, along with a well-defined and disciplined process. This process must start with product control, and understanding what the base product can and cannot do. This in turn leads to carefully managed and measured change to fit new customer needs.

When we think about Scalable Fidelity and Software often it brings to mind the end-product. That is, code running in the chosen target machine. However, to effectively leverage your legacy simulation assets when faced with new simulation requirements, or changing fidelity requirements, you must look beyond the code itself. Legacy code is a critical asset, not because it is expensive to code, but because of the value that underlies the code itself. That is, the requirements analysis, design, reviews, and testing that all went in to making the product what it is today.

To leverage the real value of your legacy assets you must first recognize where that value lies. Coding is not hard. Developing mature, proven code that produces a satisfied customer requires forethought and discipline across the full life cycle. By leveraging the full value across the complete product life cycle, the real payback of our legacy assets will be realized.

REFERENCES

1. Simulation Based Acquisition: A New Approach, Report of the Military Research Fellows, Defense Systems Management College, December 1998
2. Agile Software Development, Alistair Cockburn, Addison-Wesley, 2002
3. Adaptive Software Development, Jim Highsmith, Dorset Publishing House, 1999
4. Component-Based Software Development/COTS Integration, www.sei.cmu.edu
5. Pattern-Based Architecture: Bridging Software Reuse and Cost Management, Paul E. McMahon, March 1995, CROSSTALK
6. Integrating Systems & Software Engineering: What Can Large Organizations Learn from Small Start-Ups?, Paul E. McMahon, 2001 Software Technology Conference, www.stc-online.org
7. Extreme Programming Explained: Embrace Change, Kent Beck, Addison-Wesley, 2000
8. Software Project Management, Walker Royce, Addison-Wesley, 1998
9. Extreme Programming from a CMM Perspective, Mark Paulk, SEI, IEEE Software, Nov/Dec 2001
10. Virtual Project Management: Software Solutions for Today and the Future, Paul E. McMahon, CRC Press LLC, 2001