9th AIAA/ISSMO Symposium on Multidiscipinary Analysis and Optimization
4-6 September 2002, Atlanta, Georgia

AIAA 2002-5442

MANAGING MDO SOFTWARE DEVELOPMENT PROJECTS

J. C. Townsend* and A. O. Salas†
NASA Langley Research Center, Hampton, VA 23681-2199

Abstract

Over the past decade, the NASA Langley Research Center developed a series of "grand challenge" applications demonstrating the use of parallel and distributed computation and multidisciplinary design optimization. All but the last of these applications were focused on the high-speed civil transport vehicle; the final application focused on reusable launch vehicles. Teams of discipline experts developed these multidisciplinary applications by integrating legacy engineering analysis codes. As teams became larger and the application development became more complex with increasing levels of fidelity and numbers of disciplines, the need for applying software engineering practices became evident. This paper briefly introduces the application projects and then describes the approaches taken in project management and software engineering for each project; lessons learned are highlighted.

Introduction

Over the past decade, the Computational Aero-Sciences (CAS) Team at NASA Langley Research Center (LaRC), under the High Performance Computing and Communications Program (HPCCP), managed the development of grand challenge applications demonstrating the use of parallel and distributed computation and multidisciplinary design optimization (MDO).[1] Teams of discipline experts developed these multidisciplinary applications by integrating legacy engineering analysis codes. As the application development became more complex with increasing levels of fidelity and numbers of disciplines, the need for applying software engineering practices became evident.

All of the CAS project applications at LaRC, except for the most recent one, focused on the high-speed civil transport (HSCT) vehicle (Fig. 1). The early HSCT applications (HSCT2.1 and HSCT3.5) and their accompanying software infrastructure were referred to as the Framework for Interdisciplinary Design Optimization (FIDO) project. The last and most complex HSCT project was referred to as HSCT4.0-CJOpt. The application was called HSCT4.0, and its significantly different software infrastructure was called the CORBA-Java Optimization (CJOpt) framework. The final CAS grand challenge application project focused on analyzing reusable launch vehicles (Fig. 2 is an example); the name of this project was Environment for Launch Vehicle Integrated Synthesis (ELVIS). The ELVIS project used a commercial framework to provide a software infrastructure. In this paper, the term "framework" means a hardware and software architecture that enables integration, execution, and communication among diverse disciplinary processes.

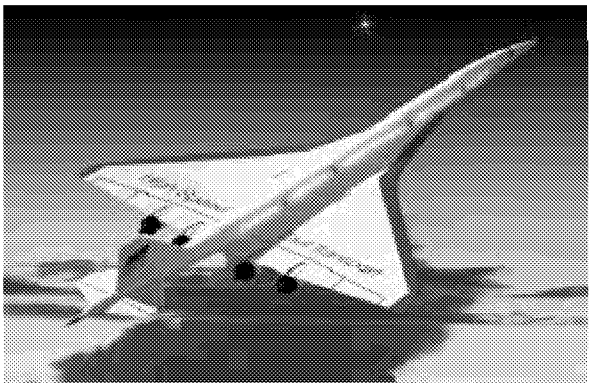

Fig. 1 – Conceptual high-speed civil transport.

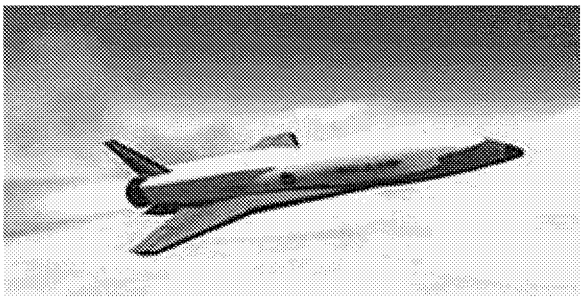

Fig. 2 – Conceptual reusable launch vehicle configuration.

The following sections provide brief introductions to the three projects mentioned above. The main sections of the paper describe the approaches to project management and software engineering for each project and highlight lessons learned. Because software management was not introduced until halfway through the program, no software improvement metrics were col-

* Member, Multidisciplinary Optimization Branch; Associate Fellow AIAA
† Member, Multidisciplinary Optimization Branch

lected; therefore, this paper gives only subjective conclusions. The Multidisciplinary Optimization Branch at LaRC was a major contributor to the CAS projects. The authors are members of that branch, and one or both of them were on each of the project teams.

FIDO Overview

The HPCCP CAS Team began the FIDO project in early 1992. The goal was to distribute an MDO application across a network of heterogeneous computers including UNIX workstations, parallel computers, and vector computers. The team members' emphasis was on understanding how to build a system to meet this goal. Conceptual design of an HSCT was chosen as the application for FIDO because of the general aerospace interest in HSCT research at that time.

Development of the FIDO system began as a proof of concept by performing relatively simple analyses on a group of workstations. This first version used fast, low-fidelity discipline codes (equivalent plate structural analysis, linearized aerodynamic analysis, propulsion table lookup, and a simple range equation for performance fuel weight estimation), a geometry given by a set of points, a small number of design variables (on the order of 10), a simple objective function (minimize gross aircraft weight), and two simple constraints (specified range and payload).

After the FIDO concept demonstration, initial development was mostly devoted to the framework infrastructure, particularly the communications library. These developments culminated in 1994 as the HSCT2.1 application,[2,3] which included static aeroelastic analysis using linear analyses. With a working framework available, development continued with the addition of nonlinear analyses for aerodynamics (marching supersonic Euler code) and for structures (parallelized finite-element analysis). The resulting HSCT3.5 application was completed in 1996.

HSCT4.0-CJOpt Overview

Changes in CAS Team and MDO Branch objectives in 1997 brought a decreased emphasis on framework development and an increased emphasis on more realistic models and higher fidelity analysis codes. The HSCT4.0 application objective was to demonstrate simultaneous multidisciplinary shape and sizing optimization of a complete aerospace vehicle configuration by using high-fidelity finite-element structural analysis, high-fidelity computational fluid dynamics (CFD) aerodynamic analysis, full mission-cycle performance evaluation, and an actual proposed HSCT geometry. The optimization problem definition was to minimize aircraft gross takeoff weight using 271 design variables and approximately 32,000 constraints (primarily structural). An important goal of HSCT4.0 was to support upcoming configuration decisions within the NASA High Speed Research (HSR) program.

An integration approach that used more standardized technologies than FIDO was desired. Because none of the commercial frameworks available in 1997 could handle the size and complexity of HSCT4.0, the CJOpt framework was developed in-house using industry standard middleware. Although the emphasis of the project was on the realism of the application, switching to new technologies required some focus on framework issues. The analysis part of the design loop was implemented with CJOpt and was completed in 1999.[4,5,6] However, before optimization could be implemented, the project was cancelled in early 2000 because the HSR program was terminated.

ELVIS Overview

The Environment for Launch Vehicle Integrated Synthesis (ELVIS) project was initiated in 2000 with the goal of using high performance computing to implement a system performing conceptual to early preliminary design of a reusable launch vehicle (RLV). An integrated RLV application was defined with processes from the Vehicle Analysis and Aerothermodynamics Branches at LaRC.

The three major elements of the ELVIS system were an integrated, low-fidelity system analysis (trajectory, weights, aerodynamics, and geometry), an advanced structural weights analysis (combined finite-element load-path analysis and theoretical weight estimation), and an aerothermal database buildup (inviscid flow and boundary-layer heating analyses) and integrated thermal protection system (TPS) sizing analysis (point-wise thermal analysis coupled with a multilayer sizing procedure). Minor elements included an in-orbit radiation analysis and code parallelizations.

The ELVIS project used a commercial integration framework, so framework development was no longer an issue. Initial capabilities were demonstrated in all elements. Cancellation of the HPCCP led to termination of the ELVIS project in early 2002. Currently, a NASA technical memorandum is being prepared that will describe the ELVIS project in detail.

Paper Organization

The remainder of the paper discusses some of the issues that arose during the FIDO, HSCT4.0-CJOpt, and ELVIS projects introduced above. The paper is organized into sections on project management, requirements management, software configuration management, software design, and testing. Each section discusses the projects' experiences in these areas and

highlights issues. The paper ends with concluding remarks on each project.

Project Management

The goal of software project management is to deliver a successful product while planning, tracking, and controlling the project's objectives, resources, and risks. In this section, the following critical project management topics will be discussed as experienced in each of the three CAS Team projects: software life cycle application, project scope, software teams (roles and responsibilities), and customer involvement. A good Software Project Management Plan (SPMP) should address these topics.

FIDO Project Management

FIDO started with a small project team of civil servants and contractor employees working closely together. The contractor employee who first envisioned the FIDO architecture was the chief designer, and a civil servant was the project manager. Each team member acted as a discipline expert responsible for integrating one or more of the disciplines represented in the project. Thus, the roles and responsibilities of the team members were initially well-defined.

Based on general guidelines from the HPCCP CAS Team[1] and a few milestones to be met, the team adopted a development strategy that began with a feasibility demonstration and followed with a series of increasingly complex systems that would ultimately meet the CAS goals. As the team member with the most solid computer science background, the chief designer introduced some software engineering practices to the team (such as coding standards and software configuration management), along with a standard directory structure across disciplines, a configuration file approach, and a standard design for the legacy code drivers. However, the team members did not follow his guidance consistently. There was no written SPMP, no clear choice of software life cycle or phases, nor any development of software requirements.

After choosing the overall architecture, process to be modeled, and interfaces, the team proceeded directly to coding without a detailed design or design reviews. The project documentation was sketchy at best. Although several attempts were made to obtain customers who could give direction and focus to the FIDO project, these efforts were unsuccessful.

HSCT4-CJOpt Project Management

The HSCT4.0-CJOpt team had approximately ten civil servants from three LaRC branches and four full-time contractor employees. The civil servants provided the discipline and multidisciplinary expertise under a technical leader, and the contractor staff supported the code integration under a contract technical monitor. Approximately one full-time person supported both requirements and configuration management; one person supported testing half time.

In the early stages of the software development, an initial draft of an SPMP was written, addressing issues such as project requirements, personnel, management practices, and software engineering. Although the software engineering portion was incomplete, the plan was expected to evolve. The plan identified incremental development as the best candidate life cycle, but the SPMP was not maintained throughout the project.

Because of its initial link to the HSR program, the HSCT4.0-CJOpt project had an aggressive software development schedule. In addition to the integrated analysis, the requirements included the ability to perform various single discipline analyses and optimizations. To meet the schedule, the project scope was narrowed and some responsibilities were shifted. The team decided to have the contractors begin the system design and implementation while the civil servants continued to define the multidisciplinary application requirements. This overlap in life cycle phases is acceptable in an iterative life cycle. However, coordination between the civil servants (whose time was stretched too thinly across multiple projects) and the contractors (most of whom were supporting the project exclusively) was not maintained. Consequently, some requirements and codes were not completely developed when delivered to the contractors.

Some additional factors made it difficult for the team to attain project goals. The team transitioned through three technical leaders with different management styles. Also, although a project goal was to support the HSR program in analyzing HSCT configurations, close interaction between the CAS and HSR organizations was not maintained. Thus, the HSCT4.0-CJOpt project suffered from the lack of customer focus; team members were left to serve as their own customer but adequate time to perform this role was not allocated.

ELVIS Project Management

The ELVIS team had approximately eight civil servants from four LaRC branches and seven contractor employees; two members supported requirements and configuration management full time. Among the lessons learned on HSCT4.0 and presented to the ELVIS team was the recommendation to adopt an evolutionary software development life cycle. Although the team agreed with this recommendation, no formal process for software development was defined. The team defined a

long-term vision and scope for the project and a list of desirable ELVIS system features. However, it did not prioritize these features nor use them to develop a clear set of requirements. The SPMP contained no details of how the software would be managed.

Three subteams were established to address the major project elements: LittleMAC to demonstrate multidisciplinary process integration by using a limited geometry code and fast, low-fidelity analysis codes; Advanced Structures to automate and extend a manual sequence of high-fidelity RLV structural analyses; and Aerothermal-TPS to automate a manual sequence of high-fidelity analyses for aerothermal database construction and thermal protection system (TPS) design. Each subteam acted virtually independently within its scope; the only interteam coordination was participation in weekly ELVIS meetings. Although the subteam products were to be integrated after a year or so to produce an initial ELVIS system, the ELVIS team never planned how this integration would take place.

ELVIS had the advantages of a well-defined customer for each of the subteams and no external program demands affecting the project milestones. Both the Vehicle Analysis Branch and the Aerothermodynamics Branch desired to have automated integrated procedures for their conceptual design and analysis work. The principal advocates for automation became leaders or active participants in the three teams. This active customer participation helped focus the teams on producing useful systems.

Project Management Summary

Because of the evolving nature of the vision and requirements for a research project, it is impossible to completely define detailed software requirements during its early stages. Thus an evolutionary development life cycle approach is particularly suited to research projects because it manages an incremental series of software builds, each of which begins with a requirements analysis phase that can utilize the results of previous builds. Project scope should be set consistent with the available resources (time, money, and people) and should be revisited whenever resources change. Also, sufficient resources should be allocated to software engineering activities.

Team leaders are presented with the challenge of coordinating team members from organizations with different cultures and mixes of civil servants and contractors. Other issues include the need for team leader authority in a matrix organization, for clear assignments to team members, and for team member time commitment and priority (especially when members are involved in multiple projects). Experience with a number of projects has shown that having an active customer who is a user of the product being developed focuses project goals; a development team that is its own customer may suffer some loss of focus as a result.

Requirements Management

Requirements management involves determining the requirements, organizing and documenting the requirements, and controlling changes to the requirements.[7] It is a critical area of software engineering that must be addressed for the success of software projects. Requirements management is a challenge for any project and can be even more challenging in a research environment, where requirements for a typical project evolve during the lifetime of the project. Issues present, but not necessarily addressed, during the CAS projects included determining the scope of the project, determining different levels of requirements, documenting known requirements, managing change to the documented requirements, defining requirement attributes, and defining traceability links. This section describes how requirements management was applied in each of the three CAS projects.

FIDO Requirements Management

For FIDO, the project manager and chief designer defined the requirements. The team did not emphasize distinguishing between requirements development, design, and implementation phases of the project. In the end, the team produced no requirements document as such, although the high-level requirements appeared in process diagrams, advocacy materials (e.g., presentations to management), and research papers. The demonstration problem implemented in FIDO was based on an application from the earlier Pathfinder project.[8] The FIDO application was documented at a high level showing the sequence of codes and key data generated.

Even though requirements were not clearly distinguished from other phases, requirements issues were brought to the attention of the team at weekly meetings, discussed, and a team consensus reached. However, because the only documentation was individual notes kept by team members, these requirements sometimes had to be revisited and clarified several times. As the complexity grew in the series of applications implemented in FIDO, the team began to realize the need to develop and document the requirements further before proceeding to implementation.

The FIDO vision evolved as the project progressed. A significant project goal was to develop a "plug and play" capability that allowed module exchange based on different computer codes for the same discipline. This goal was found to be much more complex than expected; the differences in the code input and output files made it very difficult to develop a

common interface. Therefore, this requirement was never adequately addressed.

HSCT4.0-CJOpt Requirements Management

In the early stages of the project, the team consulted with the LaRC HSR Office and with representatives from major aerospace companies to understand the minimum requirements for an HSCT analysis and optimization system that would address their needs. The majority of the requirements defined by the HSCT4.0 team addressed the integrated multidisciplinary computational analysis and optimization process. These requirements were captured by defining the various computational functions in a hierarchical series of diagrams. The diagrams showed the order and relationships of the computational functions along with the data required and generated by the system. This material and supporting text were assembled into a Software Requirements Specification (SRS).[6]

At most 20 percent of the process was taken from a previous project (LCAP[9]); thus, defining the HSCT4.0 application was a substantial part of the work done in the project. The SRS evolved over the project lifetime. Initial draft definitions of the integrated analysis were developed before coding began. However, some of the subprocess definitions changed over time. To manage the requirement changes, several reviews were held; major releases of the document were placed in configuration management. The low priority placed on documentation delayed updates to the SRS with changes identified during the requirements reviews; thus, several portions of the system remained undefined for long periods of time. Because the SRS was the most comprehensive description of the HSCT4.0 computational process, it was published after a thorough review towards the latter stages of the project.

To address traceability, several tables were created to trace the HSCT4.0 functional requirements to the HSCT4.0-CJOpt design elements; the design elements were traced to configuration items.

ELVIS Requirements Management

The ELVIS team spent considerable time defining a long-term vision statement for the project, a set of desirable system features, and a mission analysis, which included three elements with processes that were for the most part known. Subsequently, the team broke into three subteams to automate these processes. Several Unix scripts, which existed to support low-fidelity system analysis work at LaRC, were connected for the ELVIS project and provided the complete LittleMAC subprocess implementation. The Advanced Structures subsystem was based on a structural analysis process that, although previously conducted in various forms manually, had been neither documented nor automated. Although all of the pieces of the Aerothermal-TPS process existed prior to ELVIS, they were not tied together, they had never been executed together as a complete process, and several codes were targeted for replacement.

Initially, various approaches were suggested for documenting requirements; however, the team never settled on one approach. Instead, each subteam devised its own method for documenting its computational processes. Process information was captured in varying ways, including flowcharts, use cases, spreadsheets, and meeting minutes. The requirements were provided by each subteam's discipline expert and were discussed at the weekly subteam meetings. Change management was not an issue for the subteams because the process requirements were stable and the subteams were small (4 to 5 members). But, even with stable requirements, it was noted by the subteams how time-consuming it was to keep documentation up to date; thus, activities were documented at varying degrees of detail across the teams. Although the expected interactions between the three subsystems were known at a high level, the requirements for interfaces between the subsystems were never specified. No reviews of the subteam requirements were held at the project level.

Requirements Management Summary

Good communication among team members is a necessity in software projects; requirements management involves communicating goals to the team. Communication became especially important in the multidisciplinary CAS projects because of the different areas of expertise across the team members.

Project management support and team resources are needed to ensure that the project requirements are clearly documented and are accessible to all team members. The project's approach for requirements management and analysis should be defined as early as possible so that the requirements can be written consistently. Handling changing requirements should be a high priority so that all team members are aware of the changes. In each CAS project, it was important that the team knew the process that was being developed; the sequence of computations and data dependencies needed to be understood.

To minimize the work involved, requirements management tools should be employed when possible. Their use would have assisted the CAS projects in managing changes to the documents and defining requirements attributes and traceability. However, the use of new tools requires allocating project resources so that the tools can be applied in the project and experience can be gained.

Software Configuration Management

The goal of software configuration management (SCM) is to increase the reliability and quality of software. SCM defines a set of methods and tools for identifying and controlling software during its development and use. The essential elements of an SCM system are identifying configuration items, establishing baselines, and controlling changes to the baselines. Using informal, manual SCM approaches may suffice for small projects; however, as the size of the team and the software system increases, version control and formal processes become necessary. This section describes how SCM was applied in each of the three CAS projects.

FIDO SCM

Manual SCM practices were used in the early stages of the FIDO development. All the files that were to be configuration managed were placed in a simple directory structure under one main directory name. The configuration item list was the list of subdirectories. One team member acted as configuration manager, who made baselines by changing the directory permissions to "read only" and starting the next baseline as a copy of the baseline with a new name. Each subdirectory was the responsibility of a single team member, who informed the configuration manager when a code was ready to become part of a baseline. Although no formal change procedure was defined, this manual SCM process worked reasonably well while the team was small. However, when the team grew beyond a few members and the system became more complex, problems arose in tracking and controlling software changes. Team members made their own copies of directories and neglected to follow the established naming conventions. As a result, confusion eventually arose about which were the "official" copies of files and directories. These problems pointed to the need for a more sophisticated SCM system that allowed the developers to keep track of multiple versions and multiple system baselines.

One of the SCM problems encountered in FIDO involved maintaining consistency between the original legacy code and the corresponding code that became part of the FIDO system. The development team neglected to baseline the original legacy codes before changing them to subroutine library form for integration into FIDO. Consequently, when the owner of the legacy code produced a new version, it was difficult to merge the changes into the corresponding FIDO version. Eventually, the two code versions became too different and changes from the code owners were no longer accepted.

HSCT4.0-CJOpt SCM

The HSCT4.0-CJOpt project worked to establish an SCM system that included both version control and change management.[10] The TRUEchange software tool marketed by TRUE Software, Inc. (currently owned by McCabe & Associates) was selected for version control.* The Metrics Database software developed by Computer Sciences Corporation was selected for change management.

The TRUEchange projects and repositories were defined based on the organization of the HSCT4.0-CJOpt software; the SCM personnel needed close interaction with the CJOpt developers to define a repository file structure that efficiently supported both TRUEchange tool use and system builds. The Metrics Database software was customized and a detailed SCM Plan was written. Although the plan called for multiple reference areas that supported developer, test, and user system baselines, only one reference area supporting testing activities was created because of complications in the system makefiles. The test-system build was delayed by the priority of other project activities, primarily the preparation of technical papers. The system build required several months to complete, because various problems were encountered with the files that had been loaded.

Once the build was completed, the SCM personnel decided to begin utilizing the Metrics Database (to gain experience with it) rather than improving TRUEchange file layout or reference area issues. The Metrics Database system consisted of web-based software change request (SCR) forms, software trouble report (STR) forms, and promotion notification forms (PNF). The Metrics Database also produced reports that summarized essential information from the SCRs, STRs, and PNFs collected. These forms were found to be helpful in controlling changes to existing project baselines.

Because the time required to adequately support SCM was underestimated and the participation and support of managers and team members for SCM was variable, the goal of consistent SCM use throughout the HSCT4.0 project was not achieved. However, the project clearly accrued the benefits from using SCM; baselines were created and controlled using the SCM tools and processes, which ensured the integrity of the HSCT4.0 system.

---

* The use of trademarks and names of manufacturers in this report is for accurate reporting and does not constitute an official endorsement, either expressed or implied, of such products or manufacturers by the National Aeronautics and Space Administration.

ELVIS SCM

By the time the ELVIS project began, the team members did not question the need for software configuration management. The ELVIS project chose Rational Software's ClearCase and ClearQuest products for version control and change management software. An SCM manager was appointed and ClearCase administrative training was obtained. In addition, several of the project members received training in the general usage of ClearCase and ClearQuest. A ClearCase directory structure was defined for the ELVIS system, and an SCM Plan was initiated. After the project terminated due to cancellation of HPCCP, a subset of the ELVIS software and documents was loaded into the SCM system, anticipating a possible resurrection of ELVIS.

While the ClearCase SCM tool was being set up, each ELVIS subteam addressed configuration management independently. The Aerothermal-TPS team managed codes using a system supported by the Aerothermodynamics Branch. The LittleMAC team began managing files with the manual techniques described previously. However, due to the number of LittleMAC versions, this method quickly became insufficient. The Advanced Structures team did not address SCM during the ELVIS project time frame.

The modification of original legacy codes was less of an issue on ELVIS because a commercial framework was used for integration. However, no plans were made to bring the analysis source codes into the ELVIS SCM system, ostensibly because all of them were maintained under other SCM systems outside of the ELVIS project. This decision could present a problem if consistency issues of the nature described in the FIDO section arise in the future. Also, when codes need to be recompiled, this approach assumes that the correct code versions have been recorded and that the codes can be accessed from the other systems.

SCM Summary

Legacy code configuration management remains an issue. Often the codes are modified for integration purposes while the original stand-alone codes continue to evolve. Merging changes to the original code into the integrated system can be difficult. Baselining the original code in the project's SCM system can help with this problem because the version control tools can be used to merge changes between the different code versions. Although the tool can perform related functions, good communication between the developers associated with a particular code is still required.

Project managers must support the use of configuration management. Successful SCM needs to be done consistently and requires participation from all project members. Project management must define how the SCM will be addressed when both contractor and non-contractor personnel are involved in the development. The use of a software tool does not guarantee that everyone will follow good SCM practices. The team using the tool must understand the concept of configuration management and consistently follow processes. A project with little or no experience with configuration management requires SCM personnel with sufficient time to dedicate to this area. They need to monitor usage and push the processes; they also have to see that the team receives sufficient training on the use of SCM.

Software Design

To meet the CAS high performance computing requirements, each project's architecture addressed parallelism and distribution across heterogeneous platforms, data management, and legacy code integration. Frameworks can provide this architectural infrastructure. Among its advantages, a framework supports the implementation and execution of applications and provides a set of services commonly needed in MDO applications.[11] It supports the integration of various processes, allowing the designer to concentrate more on the application and less on the programming details. In addition, a framework can provide a common working environment, which can increase the productivity of multidisciplinary projects.

Both the FIDO and HSCT4.0-CJOpt projects included the development of the framework as part of the software development. The ELVIS project employed a commercial framework to integrate analysis codes and to execute them in a distributed system. The characteristics of a framework influence the design and implementation of an MDO application. Therefore, this section discusses the development of in-house frameworks and the use of commercial frameworks in each CAS project.

FIDO Software Design

The FIDO architecture was based on a modular design. FIDO application modules were created for discipline drivers, interdiscipline file translation, and optimization. In addition, a set of service modules provided execution start-up, execution flow control, data management, monitoring, user interface, and initialization of data. Each of the legacy discipline source codes was converted to subroutine library form and wrapped in a driver module. The modules resided on the most appropriate types of computer and were connected through a message-passing library.

Most message passing went through the data manager; minimal data was passed directly between mod-

ules for synchronization. The data manager read configuration files and the initial data to set up the application. These configuration files included input and output data definitions and other information, such as the machines on which the modules executed. To improve communication efficiency, the data were grouped into packages defined in the configuration files.

Other FIDO features included the Spy module and the graphical user interface, which displayed a hierarchical flow diagram of process modules colored to depict their status. The Spy module, which could be started in multiple instances during execution, allowed local and remote users to see text or graphics of values associated with problem definition, cycle status, and available data; to see cycle history for selected data; and to see timing information. In addition to the data available through Spy, the FIDO design allowed any module to be run under a debugger while the others ran normally. This capability for debugging individual modules was particularly helpful when tracking down communications problems.

HSCT4.0-CJOpt Software Design

Changes in both the MDO Branch and the CAS Team areas of focus led to more emphasis on realism of application and a de-emphasis on framework development. Framework evaluations taking place at this time determined that no commercial product would be able to support the expected complexity of HSCT4.0 and the HPCCP requirements for distribution and parallelism. Because of the complexities of the FIDO system, the team decided to replace it with a simpler in-house framework based on current industry-standard technologies. The key architectural decisions for the new framework, called CJOpt, were to use current distributed object middleware technologies (Java Remote Method Invocation (RMI) and CORBA (Common Object Request Broker Architecture) compliant software), a current object-oriented language (Java), and a commercial relational database.[12,13] The approach to integrating legacy codes within this architecture was prototyped with two smaller systems (the HSCT2.1 application and a nonlinear aerodynamics optimization[14]) prior to designing and implementing HSCT4.0.

The HSCT4.0 application was designed as a hierarchy of top-level "parent" objects, generally corresponding to primary discipline processes, and lower level "child" objects performing support services. These objects were distributed over a network of Sun and Silicon Graphics computers. For use in objects, most analysis codes were converted from their standalone program form to a function form, primarily so they could return error codes to the HSCT4.0 system. FORTRAN analysis function calls and system calls to invoke executable codes were made from C wrappers.

Java was chosen to define the objects for the following reasons: it supports distributed applications, its multithreading capability provides a convenient method for implementing parallelism within the application, it supports access to relational databases, and it allows integration with codes written in languages other than Java. The use of Java RMI and CORBA with the Inter-ORB Protocol allowed the application objects to be invoked remotely without considering the details of the underlying communication constructs and protocols. With this architecture, the client only needs to know the interface definition of a remote object to communicate with it. The database held key data used in the HSCT4.0 analysis and also information about the numerous data files, including host computer and file path. Using this information, the HSCT4.0 Java objects moved files from one machine to another with the UNIX remote copy command.

The contractors provided flowcharts documenting the design to the team at various points during the project. However, a design document and a users' manual for internal use were not completed until the latter stages of the project.

ELVIS Software Design

The three ELVIS subteams were formed early in the project. Each of these subteams proceeded independently to develop its own useful product, which would eventually become a part of the ELVIS application. The ELVIS project chose to use a commercial integration framework, ModelCenter and Analysis Server from Phoenix Integration. Also, based on HSCT4.0-CJOpt experience, an ELVIS multiuser runtime environment was defined. Other than these two decisions, the subteams did not coordinate on the design of the framework components or the interfaces between the subsystems, which posed a risk for the eventual ELVIS integration effort. Each subteam held weekly meetings to review its design and implementation. The subteams took different approaches to design documentation.

The LittleMAC team used an experimental prototype based on its scripted process to prove that ModelCenter could integrate the legacy computer codes. The prototype and the more robust system that followed were designed using ModelCenter's direct variable linkage. Later, two other strategies for implementation were developed by using ModelCenter script components for process flow control. All three methods gave the same computed results; the subteam documented the advantages and disadvantages of each and recommended an approach. One unresolved issue was whether to use an external database (as was done in HSCT4.0) to hold variable values and file names rather than to have them all contained within ModelCenter. A

partial design was captured after implementation by using several Unified Modeling Language diagrams.

The Advanced Structures team captured their software design in flowcharts and pseudo-code diagrams from which the ModelCenter components were identified. These components corresponded to a number of basic physical components (nose, tank, payload bay, etc.) that could be assembled in various ways into an RLV. Each component used a coarse finite element model for a structural load-path analysis supplemented by a HyperSizer analysis to obtain the equivalent of a high-fidelity stress analysis on a fully detailed finite-element model.

The ELVIS Aerothermal-TPS team process performed an aerothermodynamics analysis and generated a database, which was then used to design a thermal protection system. The database generation used both coarse-grain and fine-grain parallel computing techniques. The ModelCenter implementation used wrappers to run scripts spawning the executions of the process codes. In this case, the wrapped script implementation was needed because ModelCenter did not directly support parallel execution of codes. Design issues, including directory structure details, were captured in charts and spreadsheets.

Software Design Summary

One goal of the CAS projects was to develop easier ways to assemble applications. The HSCT4.0-CJOpt project moved in this direction by replacing the FIDO architecture with standard distributed-object middleware and languages that supported networking. The use of the Phoenix framework on ELVIS further advanced this goal by providing framework tools that essentially glue the application pieces together. Whether or not a framework is used, the team needs to address wrapping and integration issues early in the project. On each of the CAS projects, prototyping has been helpful in sorting out architectural issues.

All of the CAS projects would have benefited from earlier and more complete design documentation. Documentation is especially important for communication when different groups are responsible for different areas of the software development. Having documentation for review would also help ensure consistency of design across the system.

Testing

In the early stages of the software development life cycle, a software project should begin to develop verification and validation plans. These plans evolve during the course of the development. Software verification determines if the software correctly implements the software requirements. Multiple levels of tests should be conducted, including unit, integration, system, and acceptance tests. Software validation determines if the software fulfills the customer's requirements. Because in research the answers are not known ahead of time, validation is difficult and likely to be mostly empirical, e.g., made by comparing results with other computations or with experimental data. The following subsections discuss the testing experiences of each CAS project.

FIDO Testing

FIDO began with a stronger focus on demonstrating the parallel, distributed computation capability than on the multidisciplinary application realism. Therefore, testing the distributed application was not emphasized. Tests were neither planned nor documented.

Although the initial demonstration application was based on the Pathfinder project,[8] no test cases were conveniently available to the project for conducting comparisons across the two systems. Tests were conducted to verify that the automated version ran correctly. The analysis code integration and the control flow implementations were checked for correctness. Care was taken to verify that the data were passed correctly within each discipline driver and between the disciplines and the data manager. To address validation, the engineers used their expert judgment to determine if the applications results looked reasonable.

HSCT4.0/CJOpt Testing

Increased focus on application realism and anticipated interaction with the LaRC HSR Program Office led to plans for formally testing the HSCT4.0 application. As was noted earlier in the Requirements Management section, defining the multidisciplinary process was a large part of the project's effort. Having no previously existing system test cases available for validation and verification complicated the testing process. LCAP results could be used to verify computational trends only for a portion of the process.

The civil servants were to be responsible for validating the multidisciplinary analyses and optimization runs. This validation involved defining the system analysis and optimization test cases and analyzing the results from the system studies. Two system test cases were defined to support verification and validation. However, these test cases were not defined until most of the integration had taken place. In addition, the civil servants provided unit tests for some of the analysis codes to the contractors. The late test planning and the lack of defined tests for some portions of the HSCT4.0 process made it more difficult to adequately test the CJOpt implementation.

To compensate for insufficient testing of the integrated system, the civil servants performed independent unit and integration tests outside of the CJOpt framework using the original unmodified HSCT4.0 codes. It was later discovered that this testing process was not performed thoroughly enough. The testing needed to be monitored more closely and may have needed more precise definition. Part of the problem was due to a necessary cultural change; researchers normally accustomed to dealing with their own codes appeared reluctant to perform integration tests involving codes they did not own. In addition, to adequately perform these independent integration tests would have required communicating with other project personnel and understanding the corresponding CJOpt implementation.

The testing difficulties that arose called attention to the consequences of not having planned the tests early in the project. In particular, having a carefully considered test plan would have improved the efficiency of the overall integration activity, saving time in completion of the HSCT4.0 analysis process.

ELVIS Testing

As was noted in the Requirements Management section, the ELVIS subsystem computational processes were for the most part defined prior to ELVIS. Previously computed results were available for supporting subsystem implementation verification within the commercial framework. None of the subsystem projects explicitly documented test plans or test results, and there were no documented ELVIS system test plans.

The LittleMAC script implementation provided a convenient way to verify that the ModelCenter implementation of the process was done correctly. These original scripts had never been validated completely; engineering judgment was used to determine if the results were reasonable. After the LittleMAC process was implemented in ModelCenter, several optimization runs and parametric studies were conducted to assess reasonable problem formulation.

Verification plans for the Advanced Structures subsystem included comparing the implementations with past work with the structural process. However, during the design of the ELVIS structures subsystem, some modifications were made to the manual process. Therefore, the comparisons were to focus on matching trends rather than exact results. The structures process was never validated; to do so, the computed weights would need to be calibrated with a real vehicle.

Although sample results existed for the various codes and for some portions of the Aerothermal-TPS process, no existing test case exercised the entire process. A set of unit tests was put together for verification testing on the analysis codes. The subteam created a simplified test case that exercised the entire process and could be used as a demonstration case. The team planned to extend a more complicated test case, which existed prior to ELVIS, to cover the entire process. As was the case with the other subsystems, validation of the computational process consisted of applying engineering judgment when analyzing results.

Testing Summary

The need for verification and validation rose as the complexity of the CAS applications increased. Verification is more easily conducted when previous implementations of the process exist. If the process has to be defined as part of the project's work, verification becomes more complicated. If time permits, supporting scripts can be created to assist in verifying the integrated system. But, in general, it is not practical to implement the system in multiple ways just to support verification within a project.

Contractor personnel have performed the code integration for the CAS projects. The project management needs to decide who will do which types of testing. If the contractors are to perform the bulk of the testing, they need to provide a documented plan of how this testing will be accomplished. However, if the civil servants are doing integration and system testing, the project should adopt a life cycle such that the civil servants receive planned, incremental builds and thus can perform continuous testing throughout the software life cycle. Regardless of who performs the testing, teams should develop verification and validation test plans early in the project; these plans should cover all types of tests to be performed and should be updated regularly.

Conclusions

The general conclusion from experience with the CAS projects is that large research projects involving large-scale computer implementation are also software development projects and need to be managed as such. This paper reviews the critical areas of software management and describes the approach taken on each CAS project; each software topic section concludes with a summary subsection that highlights key issues and lessons learned. Although the project teams were inexperienced in software management, progress was made in this direction after surviving the initial culture shock. Each of the projects described was very different, and each had its own problems and successes.

The FIDO project was the first of the CAS projects and it emphasized developing an integration system that would support maximizing computation distribution and parallelization. Although the steps

taken to manage the project as a software project were largely *ad hoc* and the system was viewed as unnecessarily complex, FIDO successfully demonstrated distributed MDO and was visionary in some of the features that were prototyped or implemented.

The HSCT4.0-CJOpt project defined a complex multidisciplinary analysis and implemented it by using modern distributed computing technologies. Although the project could have benefited from better communication and coordination between civil servant and contractor team members and the presence of a real customer, the complex analysis was successfully implemented in a relatively short period of time. The project also introduced some fundamental software management practices, specifically requirements analysis and configuration management, to a research project.

Although the ELVIS project could have used stronger coordination among subteams, each subteam had a key member who was also a direct customer; each one provided requirements and actively participated in developing the subteam's software. The focus they provided to the project and their satisfaction with the progress contributed to creating a positive team environment. Also, the project was not pushed by unrealistic milestones, so the work could take place at a reasonable pace.

References

[1] Holst, T. L., Salas, M. D., and Claus, R. W., "The NASA Computational Aerosciences Program – Toward TeraFLOPS Computing," *30th Aerospace Sciences Meeting and Exhibit*, Reno, NV, AIAA-92-0558, Jan. 1992.

[2] Townsend, J. C., Weston, R. P., and Eidson, T. M., "A Programming Environment for Distributed Complex Computing – An Overview of the Framework for Interdisciplinary Design Optimization (FIDO) Project," NASA TM-109058, Dec. 1993.

[3] Weston, R. P., Townsend, J. C., Eidson, T. M., and Gates, R. L., "A Distributed Computing Environment for Multidisciplinary Design," *5th AIAA/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, Panama City Beach, FL, AIAA-94-4372-CP, Sept. 1994, pp. 1091–1097.

[4] Walsh, J. L., Townsend, J. C., Salas, A. O., Samareh, J. A., Mukhopadhyay, V., and Barthelemy, J.-F., "Multidisciplinary High Fidelity Analysis and Optimization of Aerospace Vehicles – Part 1: Formulation," *38th AIAA Aerospace Sciences Meeting*, Reno, NV, AIAA-2000-0418, Jan. 2000.

[5] Walsh, J. L., Weston, R. P., Samareh, J. A., Mason, B. H., Green, L. L., and Biedron, R. T., "Multidisciplinary High Fidelity Analysis and Optimization of Aerospace Vehicles – Part 2: Preliminary Results," *38th AIAA Aerospace Sciences Meeting*, Reno, NV, AIAA-2000-0419, Jan. 2000.

[6] Salas, A. O., Walsh, J. L., Mason, B. H., Weston, R. P., Townsend, J. C., Samareh, J. A., and Green, L. L., "HSCT4.0 Application Software Requirements Specification," NASA TM-2001-210867, May 2001.

[7] Leffingwell, D., and Widrig, D., *Managing Software Requirements – A Unified Approach*. Upper Saddle River, NJ, Addison-Wesley, 2000.

[8] Barthelemy, J.-F. M., Wrenn, G. A., Dovi, A. R., Coen, P. G., and Hall, L. E., "Supersonic Transport Wing Minimum Weight Design Integrating Aerodynamics and Structures," *Journal of Aircraft*, Vol. 31, No. 2, 1994, pp. 330–338.

[9] Walsh, J. L., Dunn, H. J., Stroud, W. J., Barthelemy, J.-F., Martin, C. J., and Bennett, R. M., "Aeroelastic Sizing for High-Speed Research (HSR) Longitudinal Control Alternatives Project (LCAP)," NASA TP to be published 2002.

[10] Townsend, J. C., Salas, A. O., and Schuler, M. P., "Configuration Management of an Optimization Application in a Research Environment," NASA TM-1999-209335, June 1999.

[11] Salas, A. O., and Townsend, J. C., "Framework Requirements for MDO Application Development," *7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, St. Louis, MO, AIAA-98-4740, Sept. 1998.

[12] Sistla, R., Dovi, A. R., Su, P., and Shanmugasundaram, R., "Aircraft Design Problem Implementation Under the Common Object Request Broker Architecture," *40th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference and Exhibit*, St. Louis, MO, 1999, pp. 1296–1305B.

[13] Sistla, R., Dovi, A. R., and Su, P., "A Distributed, Heterogeneous Computing Environment for Multidisciplinary Design & Analysis of Aerospace Vehicles," *5th National Symposium on Large-Scale Analysis, Design and Intelligent Synthesis Environments*, Oct 12–15, 1999, Williamsburg, VA.

[14] Biedron, R. T., Samareh, J. A., and Green, L. L., "Parallel Computation of Sensitivity Derivatives with Application to Aerodynamic Optimization of a Wing," *1998 Computer Aerosciences Workshop*, NASA CP-20857, Jan. 1999, pp. 219–224.