Evolving the Reuse Process at the Flight Dynamics Division (FDD) Goddard Space Flight Center

S. Condon,[1] C. Seaman,[2] V. Basili,[2] S. Kraft,[3] J. Kontio,[2] Y. Kim[2]

Abstract

This paper presents the interim results from the Software Engineering Laboratory's (SEL) Reuse Study. The team conducting this study has, over the past few months, been studying the Generalized Support Software (GSS) domain asset library and architecture, and the various processes associated with it. In particular, we have characterized the process used to configure GSS-based attitude ground support systems (AGSS) to support satellite missions at NASA's Goddard Space Flight Center. To do this, we built detailed models of the tasks involved, the people who perform these tasks, and the interdependencies and information flows among these people. These models were based on information gleaned from numerous interviews with people involved in this process at various levels. We also analyzed effort data in order to determine the cost savings in moving from actual development of AGSSs to support each mission (which was necessary before GSS was available) to configuring AGSS software from the domain asset library.

While characterizing the GSS process, we became aware of several interesting factors which affect the successful continued use of GSS. Many of these issues fall under the subject of evolving technologies, which were not available at the inception of GSS, but are now. Some of these technologies could be incorporated into the GSS process, thus making the whole asset library more usable. Other technologies are being considered as an alternative to the GSS process altogether. In this paper, we outline some of issues we will be considering in our continued study of GSS and the impact of evolving technologies.

1. Introduction

Since 1985 the Software Engineering Laboratory (SEL) has been evolving methods of software reuse through a series of studies, experiments, pilot projects, and full-fledged development projects at the Flight Dynamics Division (FDD) of NASA's Goddard Space Flight Center (GSFC). The SEL adopted Ada83 for these experiments and projects at a time when C++ was still relatively unknown. From this Ada work, the SEL determined that object-oriented (O-O) technology was providing the best reuse benefits within the FDD.

Around 1989-90 the Ada/O-O experience merged with an FDD-wide initiative to develop a "configurable" flight dynamics attitude support system. The result evolved into the Generalized Support Software (GSS) Domain Engineering Process. By means of this process, the FDD has shifted from developing applications to configuring applications out of generalized, reusable assets. The term "assets" encompasses design specifications, code components, tools, and standards. To date, eight applications, supporting two NASA satellite missions, have been configured from the GSS asset library and delivered to acceptance testing.

A SEL Reuse Study team was tasked to analyze the GSS process, determine the cost and quality of the resulting systems, document and evaluate its strengths and weaknesses, and propose modifications to it. This paper presents the preliminary results of this SEL study.

The paper examines several relevant cost issues. It compares the cost of investment in the GSS asset library to the investment in previous FDD reuse libraries. It compares the deployment costs (design, configuration and testing) of GSS-based applications to the development

[1] Computer Sciences Corporation, Lanham-Seabrook, Maryland

[2] Computer Sciences Dept., University of Maryland, College Park, Maryland

[3] Goddard Space Flight Center, Greenbelt, Maryland

costs of previous FDD applications and contrasts the resulting cost savings with the investment cost in the GSS asset library. The paper also demonstrates that the GSS process has resulted in a significant decrease in the time required to field a new application.

In addition to analyzing software metrics such as effort and cycle time, the reuse study team interviewed numerous domain analysts, mission analysts, component engineers, application configurers, and application testers who have been involved in the GSS process. The study team adopted Yu's Actor-Dependency (AD) formalism to model the dependence of various GSS process actors on other actors and resources. In order to further understand more complex actors in this process, the team applied Yu's Agent-Role-Position (ARP) formalism to make explicit the many different roles one actor may play in the process. (Reference 1)

2. History of FDD Reuse

2.1 Environment of the FDD & SEL

Over the past decade, the FDD of GSFC has usually consisted of about 100 civil servants supported by 300-400 CSC and subcontractor personnel. (In the last two years, NASA-wide reductions in the workforce have reduced these numbers somewhat.) Of these personnel, about 40% are software developers or testers. Another 40% are operations personnel or FDD analysts. The analysts are the experts in orbital mechanics, mathematics, or other technical disciplines who write the software requirements for FDD applications.

The mission of the FDD is to build, deploy, and maintain space ground systems for NASA science missions, with emphasis on earth orbiting satellites. Flight dynamics applications are essentially scientific data processing systems: some are institutional (i.e., they support multiple missions) and others are mission-specific (i.e., a new one needs to be built for each new spacecraft). Each FDD application supports some aspect of spacecraft flight dynamics via one of three domains: (1) attitude determination,[4] (2) mission and maneuver planning, or (3) orbit and navigation. This paper focuses on the evolution of software reuse within the attitude determination domain of the FDD.

The SEL is a virtual organization which consists of civil servants from the software development group of the FDD, CSC contractors supporting them, and representatives from the Computer Science Department of the University of Maryland at College Park. The SEL has been in existence for over 20 years, during which time it has guided, studied, documented, and nurtured software experimentation within the FDD. (Reference 2)

2.2 History of S/W Reuse at the FDD & SEL Prior to GSS

During the last dozen years, the SEL and the FDD have focused in particular on how to increase software reuse levels, with the expectation that this would reduce cost and cycle time. At the beginning of this experimentation, the FDD was developing software applications in a FORTRAN mainframe environment, achieving a modest level of reuse of very low level utilities. Through a series of studies, experiments, pilot projects, and full-fledged development projects, the SEL and FDD began evolving methods of software reuse. Efforts were focused in the attitude determination domain, whose class of mission-specific applications would benefit most from increases in software reuse.

The SEL learned a great deal about using O-O and Ada generics for one particular type of application, a simulation test tool whose development was transferred from the IBM mainframe to an Ada-friendly platform, the DEC VAX. From these experiments and mission projects, the SEL determined that the use of object-oriented principles, rather than the Ada language itself, was providing the primary reuse benefits within the FDD. (Reference 3)

The bulk of the FDD's mission-specific applications, the AGSSs, however, continued to be developed in FORTRAN on the IBM mainframe. The SEL was unable to transfer its Ada practices to the mainframe because adequate Ada tools for the mainframe environment were lacking. In lieu of this, the FDD applied some domain engineering

[4] "Attitude" means the spatial orientation of a spacecraft

concepts to create two FORTRAN reuse libraries for developing AGSSs. One library was developed to support AGSSs for non-spinning satellites, and the other for spinning satellites. The majority of satellites supported by the FDD, traditionally, are non-spinning. The FDD had some success with the FORTRAN reuse libraries, but the results were not truly "generalized" and the libraries grew with each new mission and became cumbersome to maintain. Nonetheless, these were all valuable experiences on which the FDD was able to build.

2.3 Motivation, Goals and Definition of GSS

Concurrent with the SEL-sponsored experiments in O-O, was a division-wide FDD initiative to examine the possibility of generalizing all flight dynamics software so that in future all applications would be configured rather than developed. The members of this team wrestled with what it means to "configure" an application, as opposed to "develop" an application, and came to the conclusion that it was only possible if an FDD reuse library were built around objects. This decision made the O-O experiments all the more important. Around 1989-90 the Ada/O-O experience and the search for "configurable" flight dynamics software applications merged and evolved into what was to become the Generalized Support Software (GSS) Domain Engineering Process.

The GSS process relies upon the GSS Asset Library, a library of generalized, configurable application components developed by the FDD with an object-oriented domain engineering approach. GSS specifications adhere to a standardized approach for specifying object-oriented classes. This standardization allows the use of standard rules for the implementation of each class, including a generic detailed design for each class and a system architecture that allows classes to be configured into a program that communicates with the FDD's User Interface and Executive (UIX). By means of the GSS process, the FDD has shifted from developing applications to configuring applications out of generalized, reusable assets. The term "assets" encompasses design specifications, code components, tools, and standards.

In 1992 the design of the GSS asset library got into full swing, followed in early 1993 by coding of the assets, which were implemented in the Ada83 language and resided on a DEC Alpha workstation. In February 1995 work began in earnest on configuring the first application from this asset library. To date, eight applications, supporting two NASA satellite missions, have been configured from the GSS asset library and delivered to acceptance testing. These applications run on HP or Sun workstations.

2.4 GSS as an Experience Factory

In order to carry out process improvements within the FDD, the SEL functions as an experience factory in relation to the project

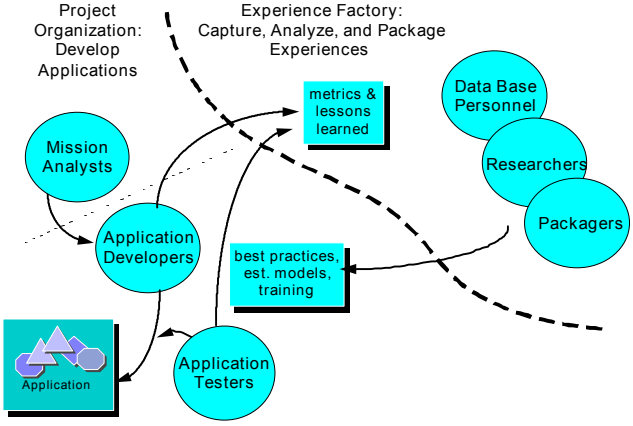


Figure 1: Traditional SEL Experience Factory

organization. The project organization consists of FDD mission analysts, application developers, and application testers. The mission analysts are the FDD personnel whose training and experience in orbital mechanics and mathematics qualifies them to write the requirements for FDD applications. As the project organization goes about its business of developing applications, the experience factory collects metrics and lessons learned from them. The experience factory staff stores these data in a database, analyzes the data, suggests and conducts additional experiments, and finally packages these distilled project organization experiences into recommended best practices, estimation models, and software development training courses, which spread these process

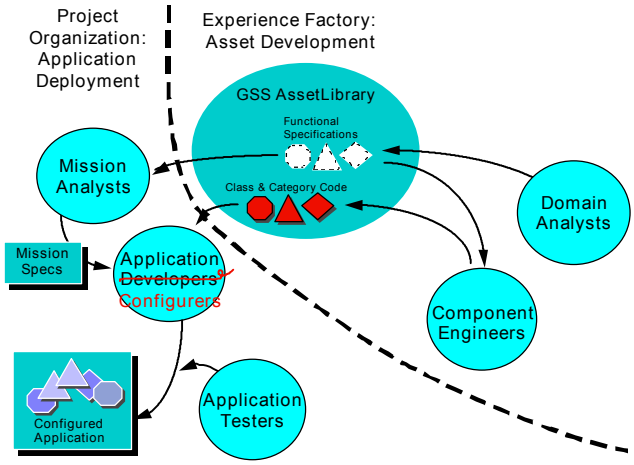


Figure 2. GSS Component Development and Application Deployment Process

improvements throughout the FDD project organization. Figure 1 depicts this traditional relationship between the project organization and the experience factory. A heavy dashed line separates the two groups. The light dotted line separating the mission analysts from the software developers on the project organization side reflects the fact that traditionally the SEL has not collected metrics from mission analysts in the FDD.

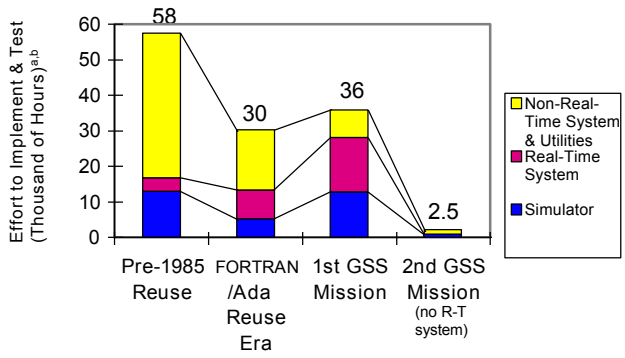
With the development of the GSS Asset Library, the boundaries and scope of the experience factory appear to have expanded. New personnel, formerly part of the project organization, are now fulfilling experience-factory-type roles. Instead of supplying only process improvements to the FDD project organization, however, these people are also supplying product improvements to the FDD in the form of generalized library assets.

Figure 2 depicts this new dimension to the experience factory concept at the FDD. A few former mission analysts have become domain analysts. They have designed the GSS architecture and written the GSS functional specifications for the library assets. At the same time several applications developers have become component engineers and have coded the classes and categories defined by the GSS functional specs. With these assets developed, the project organization then follows a streamlined process for application deployment. Under the new deployment process, a mission analyst must write the GSS mission specification that stipulates which GSS classes & categories are required for the application, which of the many parameters associated with these assets are necessary for this application, and what values need to be assigned to these parameters. This mission specification is passed to an application configurer—application developers are no longer needed—and the configurer then instantiates the specified objects from the generalized classes in the asset library and links them to form the desired application. The application testers then test the application and turn it over to operations.

3. Characterization of the GSS Application Deployment Process

A SEL Reuse Study team was tasked to analyze the GSS configuration process, determine the cost and quality of the resulting application systems, document and evaluate the strengths and weaknesses of the process, and propose improvements to it. In this section, we describe the preliminary results of this study of the GSS configuration, or application deployment, process, which is used to define, configure, and test an attitude support software application. Below, we describe the methods we used to gather and analyze this process information. In the sections which follow, we first characterize the configuration process quantitatively with respect to its cost, schedule, and the errors in the resulting applications. We then present the process graphically and analyze its inner workings.

To model the GSS configuration process, the team began by studying documentation and holding informal discussions with managers, task leaders, and a few key technical personnel. At the same time we began to analyze SEL data on effort, estimates, schedules, and software changes related to the GSS asset library and to the software applications that were configured from it. As this metrics data analysis was proceeding, we conducted numerous detailed, structured interviews with people playing a variety of roles related to GSS in order to obtain information of sufficient detail to model the configuration process.



[a] TP costs removed from application costs for first 2 eras; TPs unecessary in GSS era.
[b] Library maintenance costs included in 2nd era; GSS mission costs include total of 10 Khr of GSS overhead (library maintenance, etc.)

Figure 3: Reduced Deployment Costs Due to GSS Process

applications was more than it had been in the FORTRAN/Ada reuse era. When the time came to configure the non-real-time portion of the AGSS and the utilities, the asset library and configuration process had stabilized. As a result, this cost only a fraction of the typical cost from the previous era. With the second GSS-supported mission, we see even more dramatic savings. The simulator and the non-real-time system plus utilities each cost on the order of 10% of their cost from the FORTRAN/Ada reuse era. No real-time system was required for this application.

3.1 Analysis of Metrics Data

3.1.1 GSS Costs

There are two relevant costs to consider when evaluating the GSS project. One is the cost associated with configuring applications from GSS components. Figure 3 compares the cost of deploying GSS-based applications to costs in the previous two eras, and demonstrates that GSS-based applications can be deployed for as low as 10% of the cost required during the FORTRAN/Ada reuse era.

Prior to 1985 it cost 58,000 hours to develop and test the attitude support applications for a typical FDD mission. Later, when the FDD was using Ada reuse libraries to develop simulators and FORTRAN reuse libraries to develop AGSSs, this cost dropped to 30,000 hours per mission. In both eras the development of the non-real-time system and the utilities required the most effort.

When it came time to support the first mission with the GSS library, the simulator was configured first, and the real-time portion of the AGSS was configured second. In each case, the GSS asset library was still undergoing redesign and growth. The configurers were also evolving the configuration process. Consequently, the cost of deploying these first two

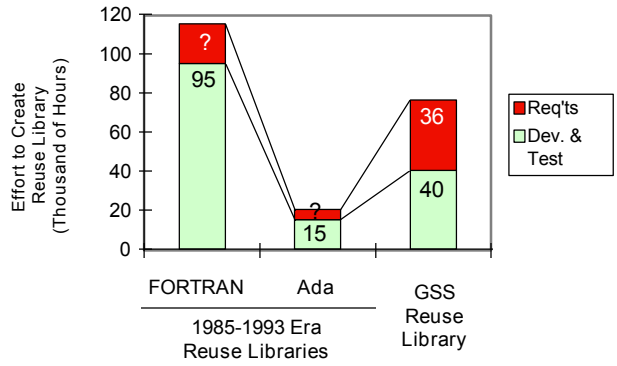
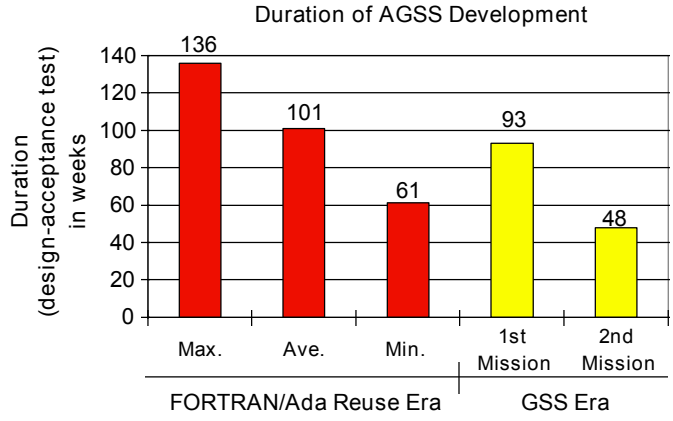


Figure 4: Library Investment Costs in Two Eras



Note: GSS era estimates assume project completions by 1/30/97

Figure 5: GSS Reduces Deployment Cycle Time

The other important cost to remember is the initial cost of building the GSS library itself. These costs are shown in Figure 4 alongside the costs to develop and test the FORTRAN and Ada reuse libraries from the previous era. For the GSS asset library we know that the domain analysts spent 36,000 hours defining the requirements and the logical design in the GSS functional specifications. The component engineers spent 40,000 hours creating the physical design and implementing, inspecting, and unit testing the generalized Ada83 classes and categories. We know the effort required to develop and test the FORTRAN and Ada reuse libraries, but we do not know the hours spent on requirements, since traditionally the SEL does not collect metrics from FDD mission analysts. Even so, we can see that the GSS library was developed for less than the combined cost of developing the FORTRAN and Ada reuse libraries, which it replaced.

Figures 3 and 4 further demonstrate that if the FDD continues to deploy GSS-based applications for 10% of the cost of the preceding era, the FDD will recoup its entire library investment cost of 76,000 hours by the fourth GSS supported mission.

3.1.2 Application Deployment Cycle Time

The GSS process has resulted not only in a great reduction in the cost of deploying an application, but also in a significant reduction in the cycle time required to deploy an application. Figure 5 reveals that the time to field an AGSS during the FORTRAN reuse era ranged from 61 to 136 weeks, with an average of 101 weeks. The time required to design, configure, and test the applications for the first GSS-supported mission was a little less than the average for the preceding era. The second project, however, was completed in less than half of the average cycle time for the FORTRAN/Ada era. In fact, it took less time than any project in the previous era. It seems likely that project duration can be further reduced with this reuse process.

3.2 Process Diagrams

After gaining an initial understanding of the GSS environment and how it is used, the team developed a detailed interview guide and conducted structured interviews with most of the designers, developers, configurers, and testers involved in the GSS processes. Once a sufficient body of information had been collected, we began to organize it by modeling the relevant processes, in particular the GSS configuration process.

We chose to use Yu's Actor-Dependency (AD) model to portray the interactions, roles, and dependencies between the actors in the GSS processes. Figure 6 is an AD model reflecting the same level of detail as depicted in Figure 2. The AD diagram reflects how each team depends on other teams. The types of dependencies are

- resource dependencies (depicted by a rectangle), which indicate that the depender relies on some artifact, document, or information from the dependee;
- task dependencies (depicted by a hexagon), which indicate that the depender relies on the dependee to complete some defined set of steps. The dependee may or may not be aware of the goals of this task;
- goal dependencies (depicted by an oval), which indicate that the depender relies on the dependee to achieve some well-defined goal. The depender has a great deal of freedom to determine how to reach that goal; and
- soft goal dependencies (depicted by a distorted oval, i.e., a "peanut" shape), which indicate that the depender relies on the dependee to achieve some goal which is not well-defined, i.e. the depender and dependee may not agree on, and must negotiate, exactly how the goal is to be satisfied.

The following AD diagrams focus more on the GSS application configuration process and show the relevant roles and dependencies at a lower level of detail.

Figure 7 expands the complex social actors of Figure 6 into their substructure of agents, roles, and positions. Agents are actual, physical people and groups of people that actors represent. Roles indicate what parts of the process an actor is involved in. Positions are the organizational titles and jobs that an actor holds. Positions generally “cover” one or more roles, while roles are “played” by an agent, who also “fills” one or more positions. In Figure 7,

only some of the relevant dependencies are shown and (for the most part) are not identified by type in order to simplify the diagram.

Figure 8 shows, at a high level, the sequences of tasks that must be completed in order to configure a GSS application, and the inputs and outputs of those tasks. Tasks are represented as ovals and artifacts (inputs and outputs) as rectangles. Many of the tasks refer to task dependencies in Figure 6.

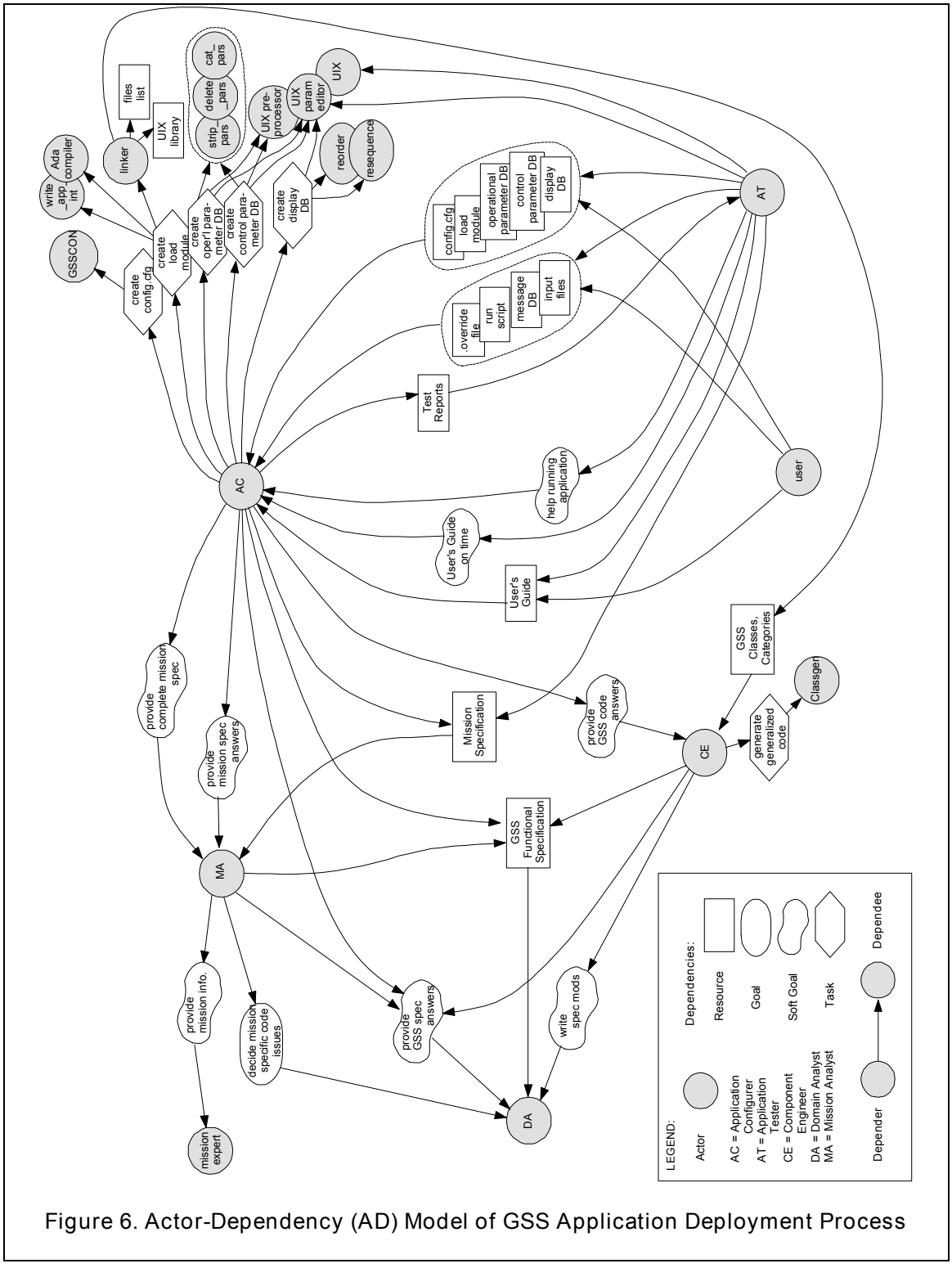


Figure 6. Actor-Dependency (AD) Model of GSS Application Deployment Process

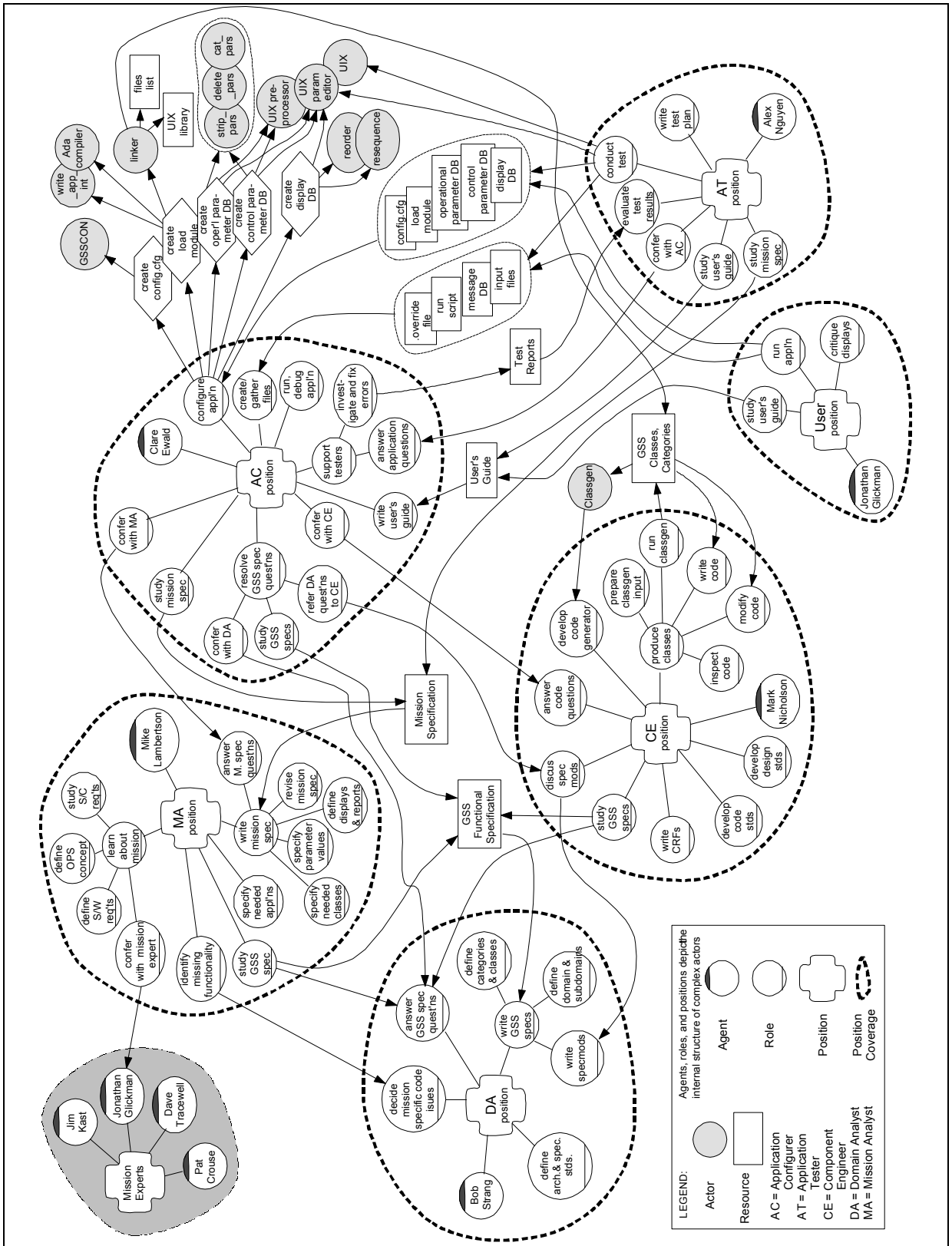


Figure 7: Agent-Role-Position (ARP) Model for GSS Application Deployment Process

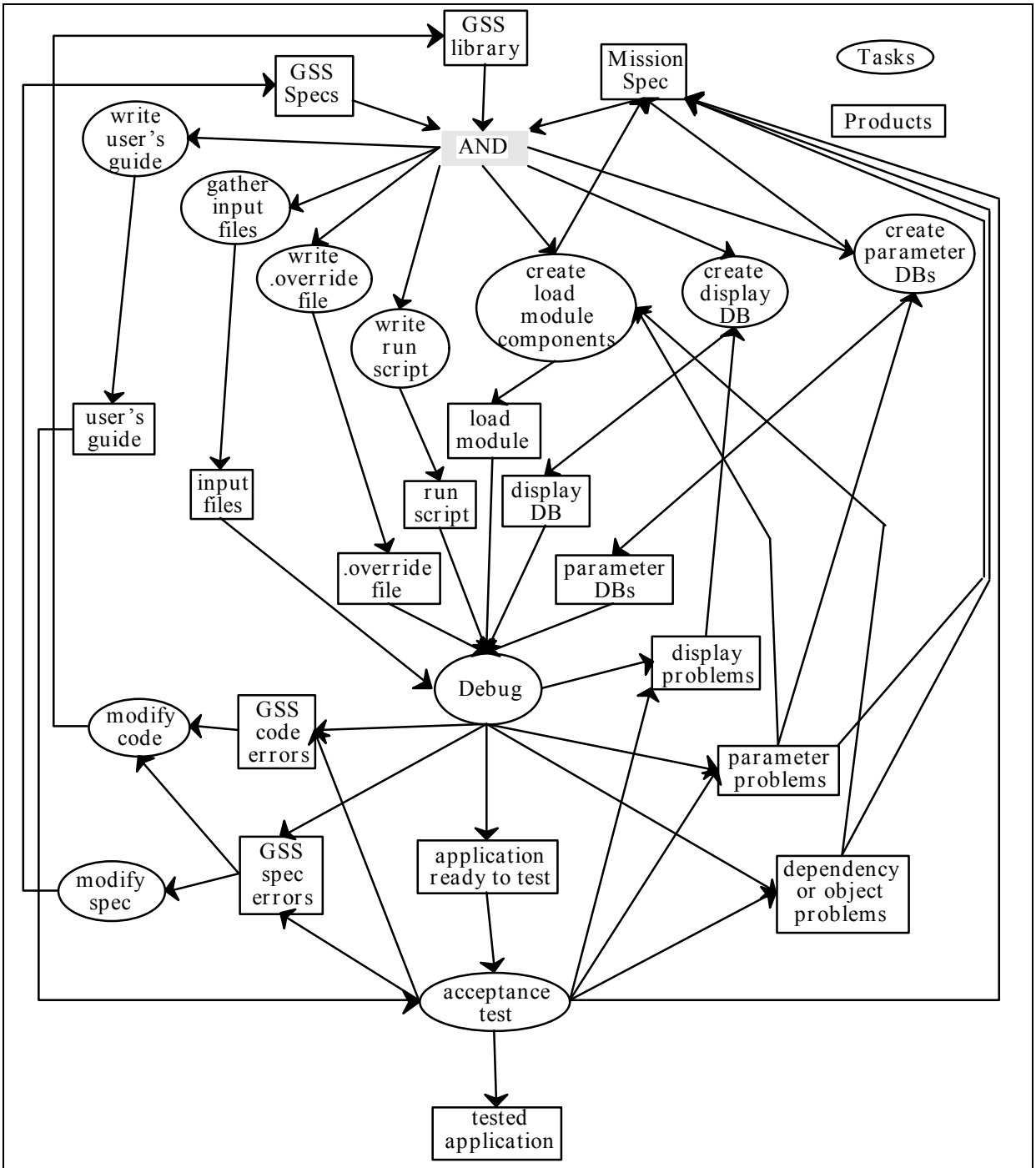


Figure 8: GSS Configuration Tasks

4. Recommendations for Improvements to the GSS Configuration Process

As is often the case, organizational and technical details which were overlooked at the project's inception have come back in various forms to threaten the full success of GSS. Despite dramatic reductions in application deployment cost and cycle time, the GSS process has not won the full support of all groups within the FDD. Although FDD management mandated that software developers and analysts would jointly design the GSS process, the resulting process is today viewed by many as the child of the software developers, with less than full partnership from the analysts.

But this is more than merely a perception. The current GSS process provides a good tool that allows traditional software developers to quickly configure flight dynamics software applications. At the same time, however, the current GSS process contains hurdles for mission analysts, whom FDD management would like to see making more direct use of the GSS. This is because the GSS process and the GSS documentation are inherently more understandable to the GSS developers and configurers than to the majority of FDD mission analysts. As discussed later, the writing of the initial mission specification in particular is a task logically performed by mission analysts, but at this time it requires a very technical level of understanding of GSS. This level of understanding is very difficult, and not necessarily appropriate, for analysts to achieve. As a result of this, relatively few FDD analysts are currently involved in the GSS process.

As a result of our in-depth characterization of the GSS configuration process, we discovered several opportunities for improvement. Some of these were synthesized from the comments of several interviewees, while others came directly from GSS developers, configurers, and testers. Most relate to the problem described above (of the barriers to use by analysts), but also would improve the GSS process in other ways as well.

4.1 Storing application requirements

Several problems were cited that might be ameliorated by storing the information contained in the mission specification in database form. First of all, it would facilitate the reuse of requirements, which is common from one application to another. Instead of manually editing reused parts lists, display files, parameter files, etc., database operations could be used to modify these elements in the database to help ensure consistency and avoid errors.

Secondly, it has been stated as a goal of GSS that eventually mission analysts should be able to configure attitude software with little or no intervention from GSS developers. There are several barriers to achieving this goal, one of which is that the writing of the mission specification seems to require very specialized skills. This is more than a user interface problem, but using a database format rather than a textual one may help.

Designing and maintaining a database for mission and application requirements would not be a simple task. It would require the borrowing or hiring of a specialist in database design, and a careful analysis of the needs that the database is meant to satisfy. Because of some of the points discussed above, a database system with an adequate user interface is especially important. Also, it would be helpful to be able to integrate this database with other databases used in the environment, e.g. databases used to store new component information.

4.2 Automatic generation of configuration inputs

Another advantage of storing mission-specific information in a database is that it would facilitate the automatic generation of some of the inputs to the GSS configuration. Generating these files at present is tedious and time-consuming. Writing the parts list in particular has been described as a translation of the mission specification from one notation to another. Such a translation could be automated if the mission specification were stored electronically. Even better, the tools which process the parts list could be rewritten so that they access the database directly. As mentioned later, such a database could also facilitate the automatic generation of some parts of the user's guide. Also, it is conceivable that a database of application requirements could also be used to

automatically generate the artifacts needed as input to UIX (the user interface facility), including the display files, the parameter files, and the message files.

4.3 Support for learning GSS

As mentioned earlier, the specialized skills required for writing mission specifications seem to be a barrier to making GSS usable by mission analysts. Making the mission spec database-based rather than a textual document may help somewhat. However, it does not solve the root problem, which is that writing the mission specification involves choosing the proper configuration of GSS components for a particular mission. This requires a level of understanding of the GSS architecture that, up until now, mission analysts have been unable or unwilling to attain. This problem has both organizational and technical aspects. Analysts were not involved enough in the development of GSS to give them any sense of ownership. Thus, they are not highly motivated to take the time necessary to learn to use GSS. Motivation is further inhibited because, up until now, one particular analyst has been willing to take on the task of writing mission specifications for all missions using GSS-based software. From a technical point of view, the current documentation on GSS (the GSS functional specifications) are written by and for software developers, not mission analysts. Their size and technicality are daunting, to say the least, and their organization is closely tied to the organization of the software, which is not necessarily the most logical from a user's point of view.

Thus, if GSS is to achieve the goal of being fully usable by mission analysts, a serious effort must be made to support learning. There is a growing area of research and development in software engineering in object-oriented frameworks; for example, the SEL is studying learning and reading techniques for frameworks (Reference 4). GSS fits the definition of an O-O framework, which is a domain-specific repository of software classes which fit into a cohesive architecture designed specifically for the domain. To the best of our knowledge, GSS is the only O-O framework specific to the flight dynamics domain. However, much of what has been learned about how to support the learning of frameworks in other domains could be applicable here. A number of strategies have been used: cookbooks of application templates and variations, example applications, documented class hierarchies, etc. One approach may be to develop a scenario-driven overlay for the GSS functional specifications which helps organize the specifications according to user scenarios. Many of these techniques could be useful in helping mission analysts understand GSS sufficiently to begin producing their own applications.

Designing learning support materials for GSS would involve some experimentation to determine which strategies are most helpful for mission analysts. This would require some investment of time and resources, and a serious commitment to finding an appropriate solution for the FDD domain and organization. It is also crucial that the support materials are designed for the most part by mission analysts, not software developers. The involvement of members of the analyst branch of FDD is necessary to ensure that the materials, and GSS, will be used in the future.

4.4 User's Guide

User's guides are required to be delivered to the acceptance testers with the application, but they are usually not completed until well after that point. Testers usually do not have them available in time to help with testing at all. Instead, they rely for the most part on the mission specification. However, the testers did not seem to see this as a big problem. The configurers, on the other hand, were not highly motivated to write user's guides and it was treated as a necessary but low-priority chore. A suggested improvement, then, is first to determine what information is really useful in the user's guide (for both testers and eventual users), then to investigate the possibility of automatically generating parts of the user's guide from the mission specification (this might be facilitated by the database suggested earlier), and finally, if necessary, assign a qualified technical writer to take on the writing of user's guides, as a task apart from configuration of the application.

5. New Directions for Reuse Study

Having characterized the GSS process, the Reuse Study Team will concentrate in the coming months on putting this process into perspective, particularly with respect to its changing technical and organizational context. First of all, a number of technological advances have taken place in software engineering since the inception of GSS. These advances may be relevant to how GSS is used in the future. Furthermore, some developments in the marketplace have produced alternative approaches to reuse. Some of these may be appropriately used instead of GSS in some cases. The focus of the Reuse Study Team in the near term will be to study which of these emerging technologies could best be incorporated into GSS and how, and under what conditions GSS could be supplanted with technology that is now available elsewhere. We hope to evolve guidelines to be used by FDD mission teams in choosing how best to produce their software applications. In the sections below we outline some of the issues on which we will concentrate.

5.1 Evolving Technologies

Over the years that the GSS has been evolving, many technologies have been evolving in the marketplace. Some of these technologies require a second look to see how they compare to the GSS process today. It may be that the GSS process could benefit from incorporating some of these technologies.

5.1.1 Object Orientation

The GSS assets have been built from an object-oriented perspective since its inception. In many ways, the development of GSS was ahead of its time, in that tools and techniques for developing object-oriented systems were not available when the GSS team needed them. For example, the only object-oriented programming languages that were available at the inception of GSS were Ada83 and Smalltalk. Now, other languages are available, such as C++ and Ada95, along with supporting tools. We will consider whether or not GSS suffered from not having these languages and tools available, and if any of the currently available languages and tools might be useful in the future maintenance of GSS. The software engineering field also knows more now about such topics as object-oriented design, testing, and maintenance. New advances need to be examined to determine their applicability to GSS.

5.1.2 Graphical User Interfaces

A User Interface and Executive (UIX) was developed by a separate group of FDD developers, in parallel with GSS, to provide GUI capability for GSS-based applications. It was decided to develop the GUI capability in-house because, at that time, no appropriate GUI packages were available in the commercial market. That is no longer the case, so it is appropriate to compare UIX to what is currently available commercially, off-the-shelf (COTS). It may be cost-effective to replace UIX with a more user-friendly and robust GUI capability developed elsewhere.

5.1.3 Other COTS Products

To support the GSS process, a number of tools have been developed in-house, such as code generators and editors. Most of these were developed in an ad hoc (as needed, as time permitted) manner. As the sophistication and quality of currently available COTS products has risen, we will investigate whether some could be used to support the GSS process. Some COTS products may even be appropriate to replace the GSS process in some cases, as discussed below.

5.2 Alternative Reuse Processes

For several years, the FDD has been slowly developing more and more software on UNIX workstations and weaning itself from its traditional reliance on the IBM mainframe. In the 1990s the FDD began to develop some of its attitude support software for execution on UNIX workstations rather than on the IBM mainframe computer. For example, the AGSSs supporting the three most recent operational satellites (SOHO, SWAS, and XTE) ran partly on the IBM mainframe and partly on the UNIX workstations. Since the FORTRAN reuse libraries resided only on the mainframe, the subsystems based on the workstations had to be written essentially from scratch. The GSS

strategic reuse library was designed entirely for UNIX workstations, and would have been useful for these subsystems, but it was not yet available.

The movement from the mainframe to workstations received a big impetus near the end of fiscal year 1995, when FDD management mandated that all software would be removed from the IBM mainframe computers by the end of fiscal year 1996. Consequently, much of the institutional and mission-specific FORTRAN code on the IBM mainframes needed to be ported to workstations in a hurry.

It was initially decided that the mainframe portions of the three most recent operational AGSSs would be re-implemented on the workstations by configuring them from the GSS library. In order to continue supporting the older legacy missions, however, an alternative method was sought. Since these AGSSs were built primarily from the FORTRAN reuse libraries and ran entirely on the mainframe, it was decided to port these libraries to the workstations.

The FORTRAN reuse library used for supporting non-spinning satellites was rehosted by two mission analysts with considerable support from some COTS products. FORTRAN subroutines were edited using word processors in order to conform to language restrictions of the COTS products. The analysts followed some process shortcuts and made liberal use of certain language features provided by the COTS products. During this rehost, the library specifications were not rigorously followed and were not updated to reflect the rehosted version of the library. Another FORTRAN reuse library, used to support spinning satellites, was rehosted by software developers, using the same COTS products. However, they closely followed the library specifications and made little attempt to take advantage of language features unique to the COTS products.

The analysts who rehosted the first library enjoyed using the COTS product and demonstrated that the rehost could be done cheaply and quickly. They found that they had a lot of control over the process and were able, because of their position, and/or the features of the COTS products, to rapidly make changes to the library during the rehost. As a result of their favorable experience, the rehosted libraries, together with their COTS umbrella, are now viewed as an alternative process for supporting new FDD missions as well as legacy missions.

In addition to these COTS products used for rehosting attitude determination systems, there are additional COTS products that can meet various other parts of typical FDD mission requirements. Some of these products are already being reviewed and adopted to support mission/maneuver planning and orbit/navigation requirements for upcoming FDD missions.

The Reuse Study Team has been charged with studying the processes associated with the maintenance and reuse of GSS, as well as those that utilize the rehosted FORTRAN reuse libraries in the development of mission support software. Our work thus far has resulted in a detailed understanding of the GSS configuration process, described in the previous sections. As well, we have come to some understanding of the questions around which to focus this comparison. These questions represent some points of disagreement between COTS and GSS proponents, some concerns raised by developers and users of both approaches, and our own analysis of interview data. These questions are presented in the sections below.

5.2.1 User Interface

GSS uses a unified user interface called UIX for all applications. UIX was developed in-house, in parallel with GSS. This has caused some problems in the testing of GSS, when errors turn out to be UIX errors, not errors in the GSS code. The use of UIX also requires the handling and formatting of a number of large files (parameters, displays, messages) in configuring an application, which can be tedious and error-prone.

Many COTS products provide their own GUI capability, which is used to create a user interface for each application. This interface is not necessarily consistent.

How important is a unified user interface? How difficult would it be to unify all the COTS-based user interfaces?

5.2.2 Is Object-Oriented Technology Superior?

The rehosted libraries are written in a procedural language associated with the COTS products used to support the rehost, in some cases from scratch and in others converted from FORTRAN code using a text editor. GSS applications are mostly Ada83 with a small amount of C code in some cases. Thus, the GSS library is based on O-O concepts, whereas the rehosted libraries, and their related applications, are not. Prior to GSS, the SEL determined that the use of Ada and O-O concepts in the FDD resulted in smaller systems to perform more functionality, while the FORTRAN reuse libraries continued to grow in size.

Since they are based on FORTRAN, will the rehosted reuse libraries continue to have the same disadvantages (in particular, code growth) as did the original FORTRAN libraries? If so, this makes the FORTRAN libraries a less attractive choice compared to O-O Ada reuse libraries. Or is there some attribute of the COTS products or the rehosting process which mitigates these disadvantages?

5.2.3 Software Engineering Practices

The design of the rehosted libraries relies heavily on the use of Global COMMON data. The software elements of the resulting applications are very tightly coupled to these data structures. Also, as mentioned earlier, one of the rehosted libraries has a code structure which mirrors the original FORTRAN structure very closely. Some developers also expressed concern that the rehosting efforts did not follow standard software engineering practices, such as inspections. On the other hand, it could be argued that rehosting does not warrant such a high process overhead because it is based on software that has been in operation for a long time.

GSS, on the other hand, was developed in accordance with more modern O-O concepts and practices. A rigorous software engineering process was followed, including design and code inspections and rigorous testing.

Does the use of O-O concepts and software engineering practices really make a difference in this case? Or does the fact that the rehosted software is based on such a time-tested library make up for its deficiencies in this area?

5.2.4 Maintenance

Both FDD COTS users and GSS proponents stress the advantages of their respective approaches for maintenance. The systems based on the rehosted libraries are argued to be easily and quickly modified by someone who is familiar with the domain, but not necessarily with software development. That is, an analyst does not have to rely on a software developer to make every change required. Using a GSS-based application, on the other hand, requires a delay whenever a change is requested, often until the next release of the GSS library. Thus using the MATLAB-based rehosted libraries provides users much quicker turnaround time on modifications of the application than does using GSS.

GSS proponents argue, on the other hand, that any system will degrade over time if it is allowed to be changed unsystematically by users. Also, the structure of GSS was designed to facilitate change without adding complexity or large amounts of new code.

Is it more important for the user to have quick turnaround on requested changes, or to manage the evolving structure of the software? Is there a reasonable compromise between the two? Do the COTS-based applications become more difficult to maintain the larger the application is? Does the design of GSS really ensure that it will not degrade over time?

Are developers and analysts using different time scales (i.e., "quick" is 1 hr. for an analyst, but 1 day for a developer?)? Are developers and analysts looking at different scopes of the modification process (i.e., a developer looks at how quick it is to change the code, whereas an analyst looks at how long he has to wait to get the revised)?

5.2.5 Performance

The applications based on the rehosted libraries are interpreted, not compiled. In some cases the source code was automatically converted to C, then compiled. This compilation step improves processing speed by a factor of two, but still remains slower than traditional FDD

applications. How much slower are the COTS-based applications than GSS-based applications, and is this difference noticeable or important to users?

5.2.6 Reliability

The AGSSs based on the rehosted libraries rely heavily on the intrinsic capabilities of the underlying COTS software for performing a number of mathematical manipulations. Care must be given to separate out errors in the COTS software from errors in the custom developed portions of the code. GSS components, on the other hand, have exhibited very low defect levels in acceptance testing. No applications of either approach, however, have been operational for long enough to assess field reliability.

What assurances do we have of the reliability of COTS products? How can it be assessed?

5.2.7 Portability

The applications based on the rehosted libraries are all designed to be part of a single system using the GUI provided by the COTS product used in the rehost. This makes porting the components relatively easy for any target platform which supports that product. On the other hand, there were some difficulties recently in porting one of the GSS-based AGSSs from the HP to the Sun workstations because UIX (the user interface which GSS uses) had not previously been ported to the Sun.

How important a criteria is portability? Can UIX and GSS be made more portable in the future?

5.2.8 Documentation

During the porting of one of the FORTRAN libraries, the original FORTRAN code structure was followed very closely. Thus, the original specifications for the FORTRAN software are still valid for the rehosted version. However, none of the advanced features of the COTS products were used which would have allowed a more efficient restructuring of the code. These features were used heavily in the porting of the other FORTRAN reuse library. As a consequence, the code is more compact than it was, but the original software specifications are no longer valid and no new specifications have been written. The analysts who were responsible for porting the libraries believe that, to a certain extent, a separate specifications document becomes less necessary because in the programming language used (associated with the underlying COTS products), the equations are written exactly as they would be written in the specification.

The design of the GSS system is documented in the GSS functional specifications, but these are 1600 pages long and, as mentioned earlier, are a real barrier to understanding the system for its eventual intended users, mission analysts. However, they seem to provide all relevant information necessary for maintaining the GSS components, and are written from a software developer's point of view.

Is either type of documentation sufficient for operation and maintenance purposes? Is the COTS-based code really self-documenting enough for maintainers to correctly make modifications? Can users of GSS components and applications be taught to use the GSS specifications effectively?

6. Conclusions

This paper presents the interim results from the SEL's Reuse Study. The team conducting this study has, over the past few months, been studying the GSS domain asset library and architecture, and the various processes associated with it. In particular, we have characterized the process used to configure GSS-based attitude ground support systems to support FDD missions. To do this, we built detailed models of the tasks involved, the people who perform these tasks, and the interdependencies and information flows between these people. These models were based on information gleaned from numerous interviews with people involved in this process at various levels. We also analyzed effort data in order to determine the cost savings in moving from actual development of AGSSs to support each mission (which was necessary before GSS was available) to configuring AGSS software from the domain library.

While characterizing the GSS process, we also became aware of several interesting factors which affect the successful continued use of GSS. Many of these issues fall under the

subject of the evolving technologies, which were not available at the inception of GSS, but are now. Some of these technologies could be incorporated into the GSS process, thus making the whole asset library more usable. Other technologies are being considered as an alternative to the GSS process altogether. In this paper, we outline some of issues we will be considering in our continued study of GSS and the impact of evolving technologies.

7. References

1. Yu, E., "An Organizational Modeling Framework for Multi-Perspective Information System Design," [Conference currently unknown], 1993(?).

2. McGarry, F., R. Pajerski, G. Page, S. Waligora, V. Basili, M. Zelkowitz, An Overview of the Software Engineering Laboratory, Software Engineering Laboratory, SEL-94-005, December 1994

3. Waligora, S., J. Bailey, M. Stark, Impact of Ada and Object-Oriented Design in the Flight Dynamics Division at Goddard Space Flight Center, Software Engineering Laboratory, SEL-95-001, March 1995

4. Basili, V., G. Caleiera, F. Lanubile, F. Shull, "Studies on Reading Techniques," Proceedings of the Twenty-First Annual Software Engineering Workshop, Greenbelt, MD, December 1996

8. Other Sources

Boland, D., L. Cisney, S. Godfrey, S. Green, T. Gwynn, J. Langston, Upper Atmosphere Research Satellite (UARS) Attitude Ground Support System (AGSS) Software Development History, Flight Dynamics Division/GSFC, FDD/552-90/092, November 1990

Briand, L., W. L. Melo, C. Seaman, V. Basili, "Characterizing and Assessing a Large-Scale Software Maintenance Organization," ICSE'95, Seattle, WA, 1995.

Brown, C., R. Coon, J. Langston, D. Spiegel, T. Wood, Internatinal Solar Terrestrial Physics (ISTP) Program/Global Geospace Science (GGS) Project, WIND and POLAR Spacecraft Flight Dynamics Support System (FDSS) Software Development History, Flight Dynamics Division/GSFC, 552-FDD-93/008R0UD0, March 1993

Condon, S., M. Regardie, M. Stark, S. Waligora, Cost and Schedule Estimation Study Report, Software Engineering Laboratory, SEL-93-002, November 1993

Coon, R., J. Golder, S. Green, J. O'Neill, Internatinal Solar Terrestrial Physics (ISTP)/Collaborative SolarTerrestrial Research (COSTR) Initiative, Solar and Heliospheric Observatory (SOHO) Mission Attitude Ground Support System (AGSS) Software Development History, Flight Dynamics Division/GSFC, 552-FDD-95/026R0UD0, November 1995

FDD analysts, developers, and testers, interviews with

FDD/GSFC, MTASS FDSS Overview, Revision 1, Update 1, October 1995

Green, D., T. Gwynn, G. Moschoglou, M. Regardie, L. Lindrose, A. Calder, S. Valett, X-Ray Timing Explorer (XTE) Submillimeter Wave Astronomy Satellite (SWAS) Utilities Software Development History, Flight Dynamics Division/GSFC, 552-FDD-96/007R0UD0, October 1996

Gwynn, T., M. Mills, M. Regardie, T. Rogers, Submillimeter Wave Astronomy Satellite (SWAS)/X-Ray Timing Explorer (XTE) Attitude Ground Support System (AGSS) Software Development History, Flight Dynamics Division/GSFC, 552-FDD-95/024R0UD0, September 1995

Kulp, D., P. Myers, M. Regardie, Total Ozone Mapping Spectrometer-Earth Probe (TOMS-EP) Attitude Ground Support System (AGSS) Software Development History, Flight Dynamics Division/GSFC, 552-FDD-94/031R0UD0, September 1994

MathWorks Web Site, http://www.mathworks.com/ and http://www.mathworks.com/matlab.html

NASA/GSFC Software Engineering Laboratory (SEL), The Generalized Support Software (GSS): A Description of Its Current Software Development Process, February 1996

Software Engineering Laboratory: data from its database

Spiegel, D., J. Doland, Fast Auroral Snapshot Explorer (FAST) Attitude Ground Support System (AGSS) Software Developement History, Flight Dynamics Division/GSFC, 552-FDD-94/040R0UD0, September 1994