IMPACTS OF OBJECT-ORIENTED TECHNOLOGIES: SEVEN YEARS OF SEL STUDIES

Mike Stark

SOFTWARE ENGINEERING BRANCH
Code 552
Goddard Space Flight Center
Greenbelt, Maryland 20771
(301) 286-5048

ABSTRACT

This paper examines the premise that object-oriented technology (OOT) is the most significant technology ever examined by the Software Engineering Laboratory. The evolution of the use of OOT in the Software Engineering Laboratory (SEL) "Experience Factory" is described in terms of the SEL's original expectations, focusing on how successive generations of projects have used OOT. General conclusions are drawn on how the usage of the technology has evolved in this environment.

INTRODUCTION

The Software Engineering Laboratory (SEL) has examined many technologies, some of which have major effects on how software is developed in the SEL production environment, where ground-support software is produced for the Flight Dynamics Division (FDD) at Goddard Spaceflight Center (GSFC). One technology, Object-Oriented Technology (OOT), has attracted special notice in recent years, causing Frank McGarry, head of Goddard's Software Engineering Branch, to remark a year ago that "Object-Oriented Technology may be the most influential method studied by the SEL to date" (Reference 1).

The SEL, sponsored by the National Aeronautics and Space Administration/ Goddard Space Flight Center (NASA/ GSFC), has three primary organizational members: the Software Engineering Branch of NASA/GSFC, the Department of Computer Science of the University of Maryland, and the Systems Development Operation of Computer Sciences Corporation. It was created in 1976 to investigate the effectiveness of software engineering technologies applied to the development of applications software. As it seeks to understand the software development process in the GSFC environment, the SEL measures the effects of various methodologies, tools, and models against a baseline derived from current development practices.

In the SEL production environment, the language usage is approximately 70 percent FORTRAN, 15 percent Ada, and 15 percent C. This is in contrast to the almost 100-percent FORTRAN environment in 1985. Projects typically last between two and four years, and they range in size from 100,000 to 300,000 source lines of code (SLOC). A typical project consists of between 20 percent and 30 percent code reused from previous projects.

THE EXPECTATIONS AND REALITY OF OOT

The development of highly reusable software is one of the promises of OOT. The initial expectations for OOT were that this increased reuse would yield benefits in the cost and the reliability of software products. In addition, it was expected that OOT would be more intuitive than the structured development traditionally used in this environment, making the development process more efficient. Therefore, the SEL expected that, in addition to the reuse benefits, the cost of developing new code would also decrease.

The specific measures applied to assess the effect of OOT include cost in hours per thousand source lines of code (KSLOC), reliability by measuring errors per KSLOC, and the duration of the project in months. To date, OOT has been applied on eleven projects in the SEL. These projects can be grouped into three families of completed projects and an ongoing effort to develop generalized flight dynamics application software.

The completed projects (Figure 1) include three early Ada simulators built between 1985 and 1988, as well as three FORTRAN ground-support systems developed from the Multimission Three-Axis Attitude Support System (MTASS) and four telemetry simulators developed from multimission simulator code, all of which multimission applications were developed between 1988 and 1991.

During the seven years the SEL has been experimenting with OOT, developers have gained more understanding of which object-oriented concepts are most applicable in the FDD environment. The most important part of the evolution is the application of object-oriented concepts to a greater portion of the development life cycle over time. The knowledge gained during the development of these three families of systems is being applied in the development of generalized flight dynamics applications.

Despite its later appearance chronologically, the MTASS family of systems (Figure 2) should be examined first because it represents a modest infusion of OOT.

MTASS started with a ground-support system that was developed as a common system for two

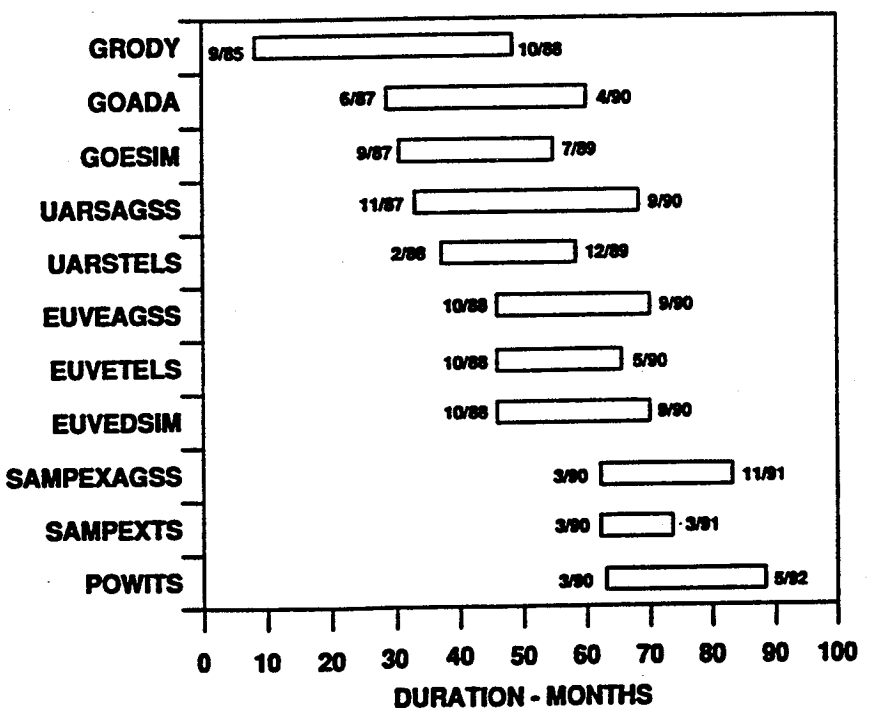


Figure 1. Projects Using Object-Oriented Technology

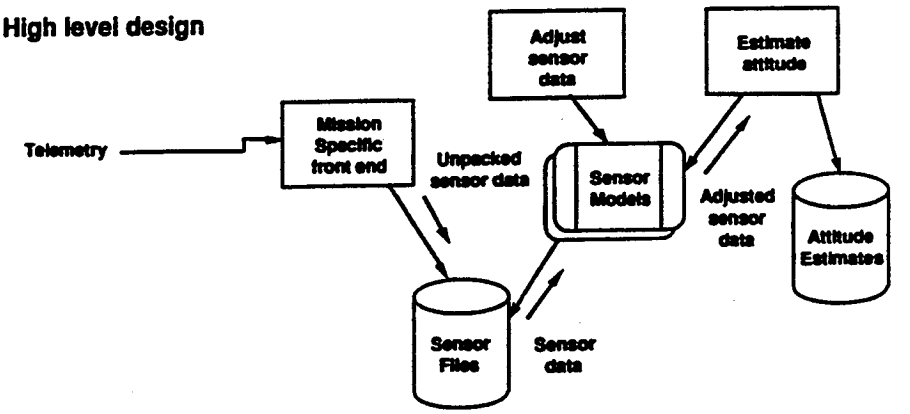


Figure 2. MTASS Design

different satellites, the Upper Atmosphere Research Satellite (UARS) and the Extreme UltraViolet Explorer (EUVE) satellite. It was then reused for the Solar, Anomalous, and Magnetosphere Particle Explorer (SAMPEX).

All ground-support systems read in telemetry and produce attitude (spacecraft orientation) estimates. The difference is that, where previous systems had stored all sensor data in one file specifically designed for the mission, MTASS developed separate interface routines and file formats for each kind of sensor. Only a mission-specific front-end telemetry processor had to be developed for new missions.

This basic grouping of data and operations on the data is the most important object-oriented concept in the FDD environment. This change alone enabled an increase in code reuse from the baseline 20 percent to 30 percent to around 75 percent or 80 percent.

It should be emphasized that the use of OOT on these projects was modest. The implementation language is FORTRAN, and the standard structured design notation was used to document the system. The object-orientation of the sensor model design was recognized during coding, rather than consciously planned during design. Nonetheless, this one simple concept has had tremendous benefit in developing ground-support software faster and at a lower cost.

The earliest purposeful use of object-orientation in the SEL environment was associated with the introduction of Ada in 1985. The first Ada project, the Gamma Ray Observatory (GRO) Dynamics Simulator in Ada (GRODY), was developed as an experiment in parallel with an operational FORTRAN simulator. Previous Ada experiments (Reference 2) had produced designs and code that looked like Ada versions of FORTRAN systems. To avoid this, the GRODY team was trained in a variety of design methods, including Booch's Object-Oriented Design (OOD) method (Reference 3), stepwise refinement, and process abstraction. In addition, one of the team members had an academic background in OOD.

OOD emerged as a clear favorite, but in early 1985 Booch's method was not mature enough to support large production projects. Stark and Seidewitz developed the General Object-Oriented Design (GOOD) method during the GRODY project to meet these needs (Reference 4). Its first application was on the Geostationary Operational Environmental Satellite (GOES) Dynamics Simulator in Ada (GOADA), a project started in 1987.

The goal of the early Ada simulation projects was to learn the appropriate use of the Ada language, with a view towards increasing software reuse. Other goals were considered less important. The GRODY team, for example, was specifically instructed not to worry about the real-time requirement being imposed on the FORTRAN simulator, and in fact GOADA was able to achieve higher than usual reuse from GRODY code. However, the lack of attention to performance led to systems with disappointing performance.

The SEL responded to this issue by studying the performance of the GOADA simulator in detail to determine if the performance problems were caused by the Ada language, the OOD concept, or by the GOADA design itself. The studies estimated the effect of various improvements on the execution speed of a simulation. The key improvements were as follows:

- Removing repeated inversions of the same matrix from the integrator's derivative function
- Modifying the storage of arrays of variant records to arrays of pointers to variant records; this reduced the amount of memory used to store program input parameters
- Changing package state from dynamically allocated parameters to static variables
- Optimizing utilities for three-dimensional linear algebra to use constrained types for vectors and matrices*
- Removing a string conversion from the main simulation loop
- Replacing a schedule queue with hard-coded fixed time step-simulation loop
- Compiling debug code conditionally by using dead-code elimination in "if" blocks

None of these changes fundamentally altered the object-oriented nature of the design. Figure 3 shows that making all these changes to the full simulator would improve performance to the levels attained by similar FORTRAN simulators.

The next generation of projects is a multimission telemetry simulation architecture, built around Ada generic packages. Figure 4 shows how two sensor models use a generic sensor package for common functions such as writing reports and simulated data files.

Here, each sensor has its own specific modeling procedure that is used to instantiate the generic. In addition, these model procedures are built around other generics that provide common functionality such as modeling sensor failures or digitizing simulated sensor data. One of the interesting consequences of the extensive use of generics is that the system size decreased. The previous generation of Ada telemetry simulator contained 92 KSLOC, but this multimission simulator contains only 69 KSLOC.

This architecture was the first simulator designed to facilitate reuse from mission to mission. Unlike the MTASS system, this simulator does not need a mission-specific subsystem to handle telemetry; the telemetry formats can be set by run-time parameters. When this strategy is used appropriately, the reuse levels approach 90 percent verbatim code reuse, with the remaining part undergoing minor modifications.

While this 90-percent reuse level has helped reduce software costs and shorten development schedules, it has only done so on a limited class of systems. When the telemetry simulator was reused for a new class of systems (spin-stabilized spacecraft), the system complexity increased, reuse decreased, and run-time performance suffered. MTASS had a similar problem when it was applied to a spacecraft that didn't have a sensor on which the original MTASS design depended. In addition to variations between spacecraft, simulators and ground systems contain many common models. However, the current practice is to create separate systems from separate specifications. The way to account for variations between satellites and to exploit commonality between software systems is to perform domain analysis, rather than attempting to generalize the specification of a single satellite's simulator and ground-support system.

In the FDD, this domain analysis is being done as part of a generalized system development initiative. The attempt to develop generalized software to support multiple flight dynamics applications was based on the experiences of the projects described above. The multimission simulators demonstrated the feasibility of generic architectures, and it had been demonstrated that applying the object-oriented concepts of abstraction and encapsulation was sufficient to increase reuse dramatically. Finally, the existing designs were highly reusable, but had severe limitations in the areas of adaptability and run-time efficiency.

* Most vectors and matrices in flight dynamics are three-dimensional. Using this constraint allowed hand optimization of all the operations.

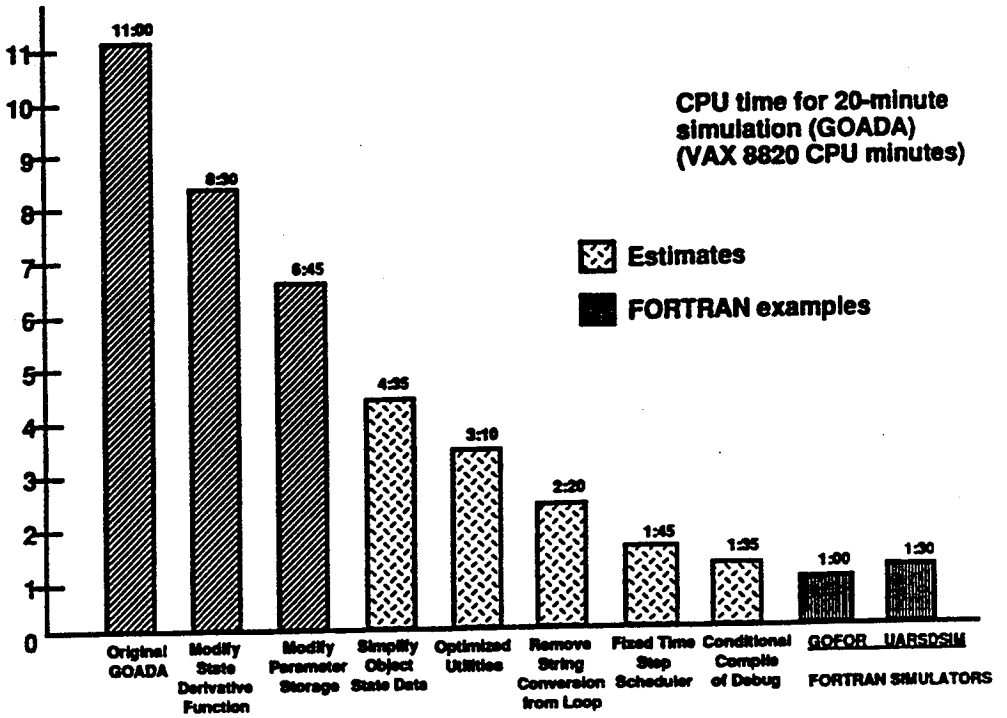


Figure 3. Impact of Performance Goals

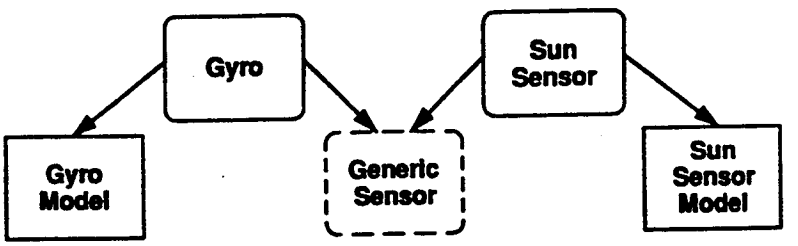


Figure 4. Multimission Telemetry Simulator Design

The key concepts selected for generalized system development in the FDD are to perform object-oriented domain analysis, and to have a standard implementation approach for the generalized models. Figure 5 shows a typical diagram from the generalized specifications.

The boxes are generalized superclasses with their subclasses listed inside; Gyro, Sun Sensor, and Star Camera, for example, are subclasses of Sensor. The arrows between categories represent dependencies between classes. For example, estimators depend on Sensor for measurements and Dynamics for state propagation. These dependencies are matched in the implementation with Ada generic formal parameters. The classes themselves are implemented as abstract data types in Ada packages. With this generalized development effort, object-oriented domain analysis and standard implementation, as well as other features of the object-oriented paradigm, are now being applied to the entire software life cycle.

With the successive generations of object-oriented development efforts defined, the next step is to examine how the SEL's approach has changed between 1985 and 1992. The approach has evolved in what concepts are used, when they are used in the life cycle, and how they are taught.

The concepts of data abstraction and encapsulation, used from the beginning, have themselves enabled the high reuse observed on the MTASS system; even the second Ada simulator attained higher reuse than is typical for similar FORTRAN simulators. The multimission telemetry simulator introduced the idea of inheritance by taking a general model for sensors and tailoring this model for each type of sensor. It also introduced the idea of parameterizing dependencies with Ada generic formal parameters. The generalized application work added the use of abstract data types, where previous systems had implemented objects as state machines. The generalized systems also have a superclass/subclass hierarchy limited to superclasses (called "Categories") and one level of subclasses for each superclass.

Dynamic binding is coded using Ada case statements, not an object-oriented programming language feature. Having support for object-oriented programming in Ada would remove the need to write this code, but the cost reduction from using data abstraction is much greater.

The other notable change is in how OOT affected the development process. In the MTASS system, it had minimal impact, as the design approach was structured, with the object orientation being recognized during coding. Both generations of simulators used object-oriented design and object-based coding based on Ada packages; the generalized system project added an object-oriented approach to defining specifications. It is anticipated that having an object-oriented view throughout the life cycle will make the use of the technology easier by removing the need to recast functional specifications into an object-oriented design.

The SEL provided training in Ada and design techniques for the early Ada simulator experiments, but the not the later multimission simulators. The MTASS FORTRAN system involved no training in

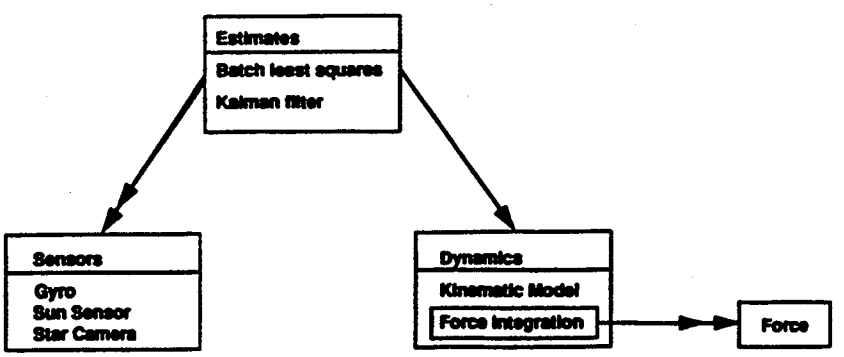


Figure 5. Generalized System Specifications

OOT, as the project did not set out to use a new language or design technology. The subjective experience of the SEL has been that the application of OOT was not so intuitive as expected, as functional decomposition has been successfully applied for over 15 years. The SEL, recognizing that transition to a new technology must factor in time to learn the new way of thinking, is creating a new training program that captures the lessons learned on previous projects and describes the overall object-oriented software development process as well as specific language and design concepts.

The use of OOT also affected software reuse, which in turn affected how software specifications are written. FORTRAN ground-support systems have always relied on libraries of software components, and MTASS continued that tradition. The early Ada simulators also developed utility packages that played a role analogous to FORTRAN reuse libraries. Both MTASS and the early Ada simulators pioneered idea of reusable subsystems; the simulators contain a reusable collection of orbit packages and MTASS contains major capabilities (such as attitude estimation) that are implemented as a reusable collection of subroutines. The multimission simulators added the idea of generic, tailorable models; instead of reusing single subroutines or entire subsystems, Ada generics were used to design generic packages to implement reusable objects from common templates. The generalized system work added the concept of parameterizing the dependencies between objects, easing the configuration of multiple systems that include different models from the same general categories.

The evolving reuse concepts affected the specifications for missions. The early simulator work focused on specifying simulators to support single missions. When reusable subsystems were first applied on Ada simulators, specifications were written for multiple missions. However, separate specifications were written for simulation and ground-support systems, and the focus was on two missions supported simultaneously, rather than on a generalized domain analysis. The generalized systems project identified common elements among application areas by means of domain analysis and wrote specifications that accounted for variations between missions by parameterizing the dependencies between classes. The current approach, in contrast, calls for both reconfigurable specifications and a standard, reusable system architecture.

The goal of bringing new technology into the SEL is to measurably improve the software development process. Figure 6 shows the project characteristics of the three multimission simulator projects.

The project labeled UARSTELS was developed to be reused for future simulators, and the projects labeled EUVETELS and SAMPEXTS represent the first two projects to reuse this architecture. Costs were reduced by a factor of 3, change and error rates were reduced by a factor of 10, and project cycle time was cut roughly in half. However, we have already shown that when an attempt was made to reuse this architecture for a different class of projects there were difficulties adapting the code, and run-time performance was unsatisfactory.

The generalized system effort is attempting to gain the benefits shown for this single family of projects over a wider variety of flight dynamics applications. This will allow the FDD to support more missions simultaneously, and will free resources to concentrate on improving existing capabilities or defining new ones.

THE ANSWER

This paper addresses the question, "Is Object-Oriented Technology, then, truly the most influential method studied by the SEL to date?" The conclusion of the SEL is that OOT does promote reuse, sometimes even neglecting other important issues like run-time efficiency. When coupled with domain analysis, OOT enables high reuse across a range of applications in a given environment. While the reuse expectations were met, the use of OOT was not so intuitive as expected, partly because the technique was new to an organization with a mature structured development process. The other factor affecting the ease of transition is the inherent and growing complexity of flight-dynamics problems; OOT may be a better process but, in addition to software techniques, skilled designers are still needed to solve difficult problems.

Still, few (if any) of the other technologies studied here have effects so widespread or so profound as

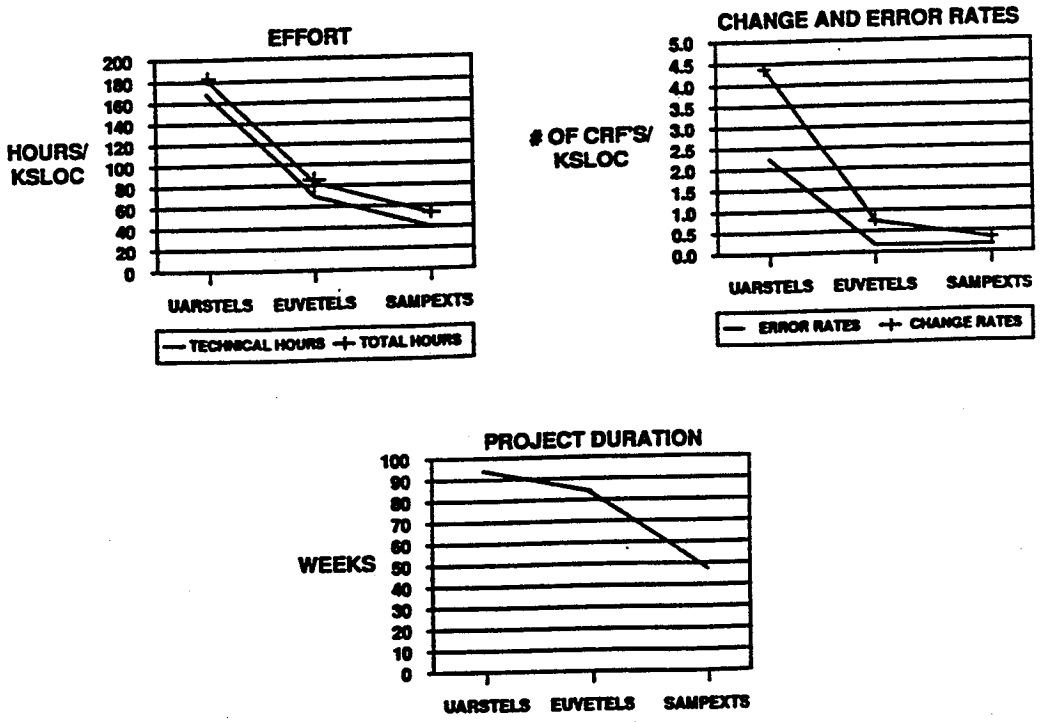


Figure 6. Project Characteristics, Multimission Simulators

OOT. In fact, OOT is the first technology that covers the entire development life cycle in the FDD. It is an entirely new problem-solving paradigm, not simply a new way of performing familiar tasks in a traditional life cycle. It has been demonstrated to expand the reusability and reconfigurability of software, with resultant improvements in productivity and development cycle time. In this sense, OOT is arguably the most influential technology studied by the SEL.

REFERENCES

1. McGarry, Frank E., and Waligora, Sharon, "Recent Experiments in the SEL," *Proceedings of the Sixteenth Annual Software Engineering Workshop*, Greenbelt, MD, December 1991, pp. 77–85.
2. Basili, Victor R., and Katz, Elizabeth E., "Software Development in Ada," *Proceedings of the Ninth Annual Software Engineering Workshop*, Greenbelt, MD, November 1984, pp. 65–85.
3. Booch, Grady, *Software Engineering With Ada* (First Edition), Benjamin/Cummings, Menlo Park, CA, 1983.
4. Seidewitz, E., and Stark, M., *General Object-Oriented Software Development*, SEL-86-002, August 1986.

IMPACTS OF OBJECT-ORIENTED TECHNOLOGIES:

SEVEN YEARS OF SEL STUDIES

Mike Stark
December 2, 1992

"OBJECT-ORIENTED TECHNOLOGY MAY BE THE MOST INFLUENTIAL METHOD STUDIED BY THE SEL TO DATE"

Frank McGarry at the 16th Annual Software Engineering Workshop

December 4, 1991

AGENDA

- Background
- Evolution of Object-Oriented Technology
- Observations and Recommendations
- Conclusions

OBJECT-ORIENTED TECHNOLOGY EXPECTATIONS

Common Software Engineering Measures

Measures	Expectations
Software Process Measures	Shift of effort towards design phase
Cost of new line of code	More efficient development
Overall cost of software projects	Lower project development costs
Reliability (Errors per KSLOC)	Lower error rates during development
Maintainability	Lower maintenance costs

Studies compare new technology (OOD) to well-measured baseline (FORTRAN structured design)

Reasons to expect improvements

High reuse potential	Substantial increase in reuse
Improved software development process	Object concept more intuitive

Must apply specific measures to assess new technology against current practice

SEL PRODUCTION ENVIRONMENT

SOFTWARE CHARACTERISTICS
- SCIENTIFIC (FLIGHT DYNAMICS)
- GROUND BASED (NON-EMBEDDED)
- INTERACTIVE

LANGUAGES
- 75% FORTRAN
- 15% ADA
- 10% OTHER (C, PASCAL, LISP, ...)

PROJECT CHARACTERISTICS

	TYPICAL
• DURATION (MONTHS)	24-40
• EFFORT (STAFF YEARS)	30-45
• SIZE (KSLOC)	100-300
• STAFF (FTE)	5-15
• REUSE	20-30%

*SEVERAL PROJECTS EXCEED 200 STAFF YEARS, SEVERAL LESS THAN 3 STAFF YEARS

*HOMOGENEOUS CLASS OF SOFTWARE

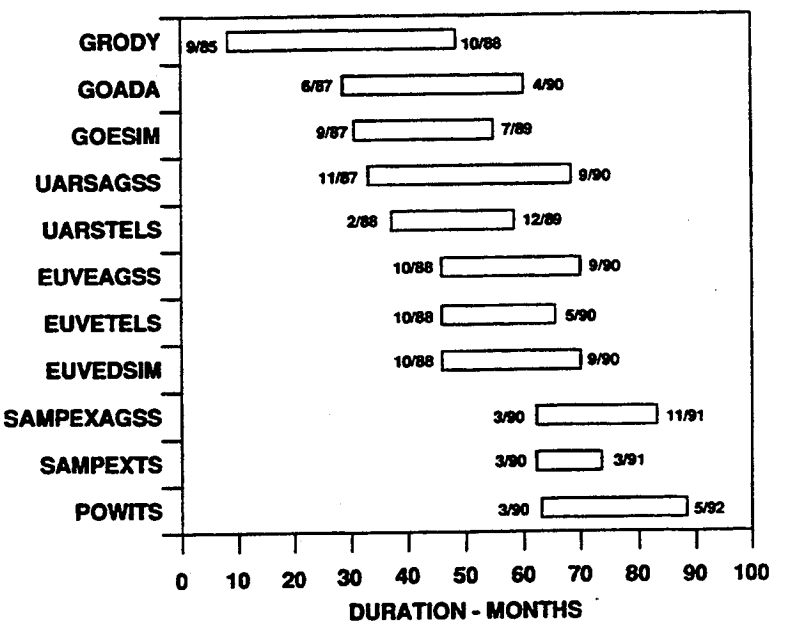
*CONSISTENT SUPPORT ENVIRONMENT

*CONTROLLED PROCESS

*EVERY PROJECT ANALYZED WITHIN EXPERIENCE FACTORY

10006934G

PROJECTS USING OBJECT-ORIENTED TECHNOLOGY



10006934G

EVOLUTION OF OBJECT-ORIENTED TECHNOLOGY

Evolution

- Use of concepts
- Life cycle coverage
- Tools and Training

Systems Studied

3 Early Ada simulators (1985-1988)

3 Ground Systems built from Multimission Three-Axis Attitude Support System (MTASS) (1988-1991)

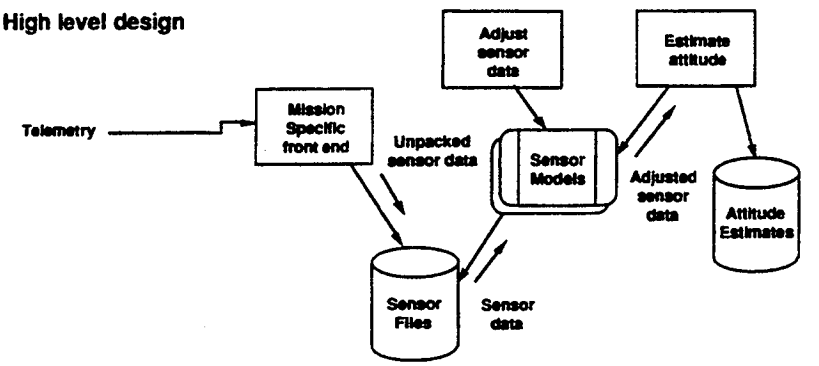
4 Telemetry simulators built from multimission simulator code (1988-1991)

Generalized System Development (1991-)

10006934G

MTASS CHARACTERISTICS

High level design



General Characteristics

Concept development: New view of sensor data

Language: FORTRAN

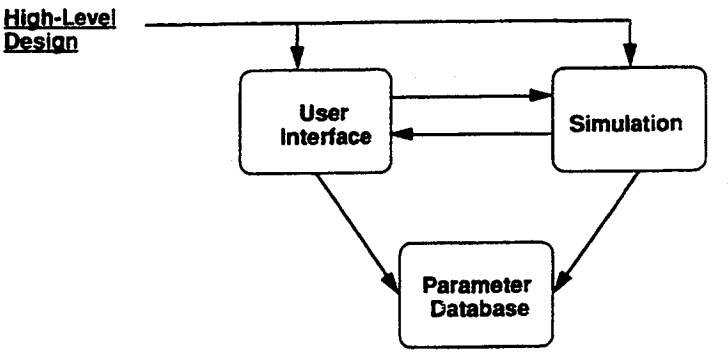
Requires new code for mission unique telemetry processing

Object-orientation recognized during coding

10006934G

Object-Oriented Technology

EARLY SIMULATOR CHARACTERISTICS



General Characteristics

Language: Ada

Training: 1st project (GRO) : Ada language, design techniques (Booch, Cherry)
2nd project (GOES) : Ada, GOOD, DEC tools

Concepts Developed: General Object-Oriented Development (GOOD)

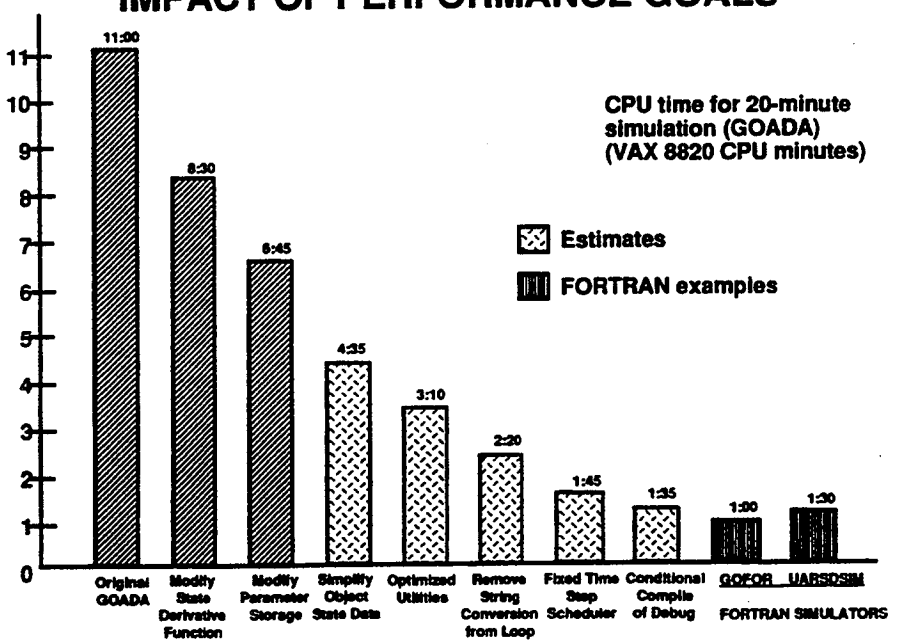
Early Ada simulators used object-oriented design

SEL Software Engineering Laboratory

10006934G

Object-Oriented Technology

IMPACT OF PERFORMANCE GOALS

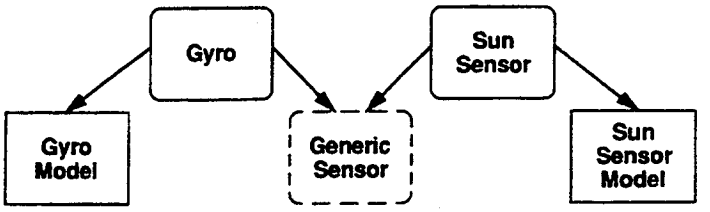


No trade-off between object-orientation and performance

SEL Software Engineering Laboratory

10006934G

MULTIMISSION TELEMETRY SIMULATOR CHARACTERISTICS



General Characteristics

Language: Ada

Concept Development: Use of Ada generics
Tailoring of generalized hardware package

Telemetry formats set by run-time parameters

3/4 of the code needed by early telemetry simulators

First reusable architecture in SEL

DOMAIN ANALYSIS

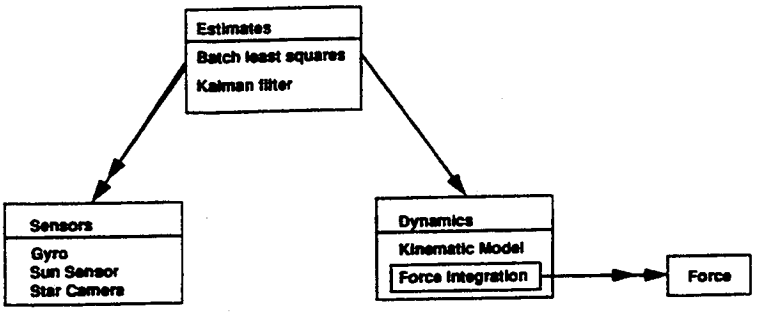
- Single-system designs don't account for variations between missions.

Examples

1. MTASS didn't consider spacecraft without gyros.
   SAMPEX ground system needed to create pseudo-gyro data to feed into reusable code - instead of estimating directly from sensor data.
2. Generic telemetry simulator didn't consider spinning spacecraft.
   Reuse of three-axis design for spinning spacecraft led to
   - higher complexity
   - slower execution
   - lower reuse
3. Commonality between simulation and estimation is ignored (duplicated code)

Domain Analysis is needed to account for variations between systems.

GENERALIZED SYSTEM DEVELOPMENT



General Characteristics

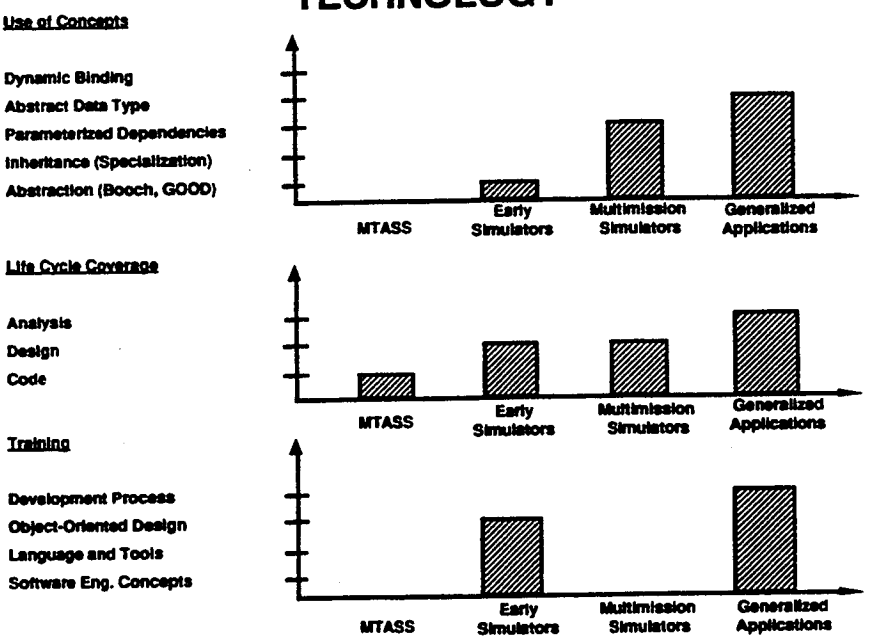
Concept Development: Configurable object-oriented domain analysis
Standard design approach for classes specified

Language: Ada

Training: SEL Training Courses (Ada, OOD, development process and tools, flight dynamics)
Generalized system concepts
Ada efficiency

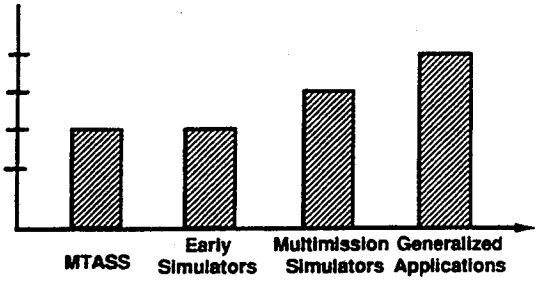
Object-oriented technology is extended to specifications

EVOLUTION OF OBJECT ORIENTED TECHNOLOGY

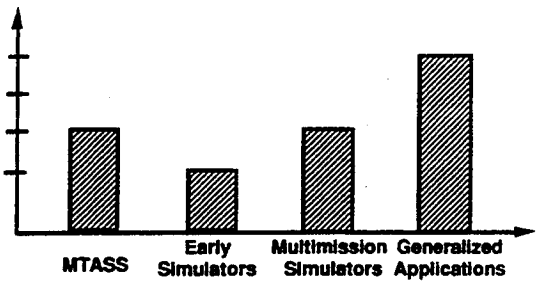


IMPACT OF EVOLVING TECHNOLOGY

<u>Reuse Concepts</u>
- Parameterized Dependencies
- Generic Models
- Reusable Subsystems
- Component Libraries

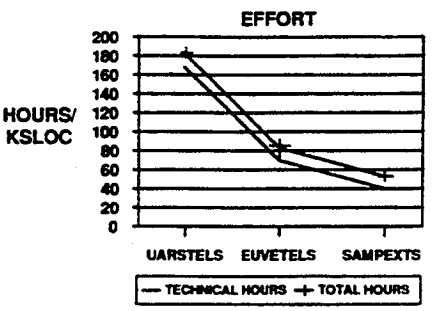


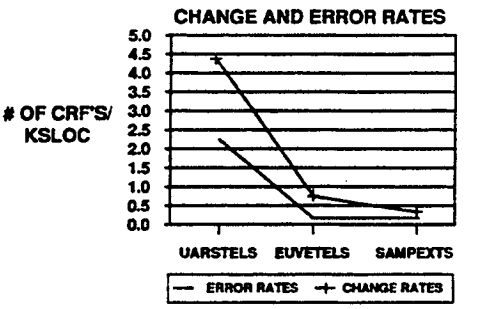
<u>Specifications</u>
- Configurable Specifications
- Multiple Missions and Applications
- Multimission Application
- Single Mission and Application

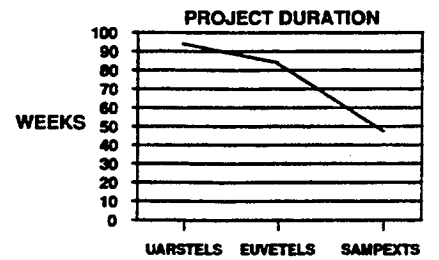


Object-oriented technology has evolved to reconfigurable specifications and reusable architectures

PROJECT CHARACTERISTICS - MULTIMISSION SIMULATORS







<u>Benefits</u>
- Decreased cost
- Shortened cycle time
- Increased reliability

C-2

OBSERVATIONS AND RECOMMENDATIONS

Observations

- Domain analysis is necessary to gain full reuse benefits
- OO technology promotes reuse over other considerations
- Transition to new technology takes time and effort
- OO technology is not a silver bullet (skilled designers are still needed)
- OO is first technology to cover full lifecycle in FDD

Recommendations

- Tailor usage as needed for environment (FDD didn't use OOP language)
- Start with smaller pilot projects
- Don't reuse an architecture until it meets other design goals

10006934G

CONCLUSIONS

Object-oriented technology is the most influential technology studied by the SEL because it

- Is a new problem-solving paradigm
- Is applied to all life-cycle phases
- Expands reusability and reconfigurability of software
- Improves productivity and cycle time

10006934G

Session 2: Process Measurement

Rose Pajerski, NASA/Goddard, Discussant

John O. Jenkins, City University, London

Anthony J. Verducci, Jr., AT&T Bell Laboratories, USA

Charles B. Daniels, Paramax Space Systems Operation