OOPSLA 1993

Addendum to the Proceedings

Washington, DC—26 September–1 October, 1993

Experience Report

Training Realtime Simulation Developers in Object-Oriented Methods with Ada

Report by:

Gary J. Cernosek
Fastrak Training, Inc.

Introduction

In December 1991, Fastrak Training Inc. was awarded the Object-Oriented Development and Ada Training (OODAT) Contract let by CAE-Link Flight Simulation Co. in Houston, Texas. Fastrak began administering this comprehensive training program in January 1992. I was assigned at that time as the lead instructor for the program, assuming responsibilities for developing and delivering courses in object-oriented methods and Ada programming. This report describes my experiences with the OODAT program, including its technical foundation, overall results, lessons learned, and future direction. My goal is to have readers appreciate what it takes for an organization of several hundred engineers to transition into an object-oriented environment.

Project Description

CAE-Link is a prime contractor to NASA's Johnson Space Center and is responsible for developing mission trainers for the Space Shuttle and Space Station programs. The Space Station systems are being developed using Object-Oriented (OO) Analysis and Design methods with an Ada implementation platform for both realtime and non-realtime applications. The Station trainer will be a ground-based distributed computer system that allows NASA to prepare crew personnel for their missions and activities. The effort will ultimately produce an estimated 2 million source lines of Ada code.

Training Curriculum

The original OODAT contract required Fastrak to provide a curriculum of four specific courses:

- Object-Oriented Requirements Analysis (OORA) (4 days)
- Object-Oriented Design (OOD) (4 days)
- Ada Syntax and Semantics (7 days)
- Advanced Ada (5 days)

New requirements emerged for several forms of secondary training:

- Special training was required for NASA customers and the various user organizations. A 2-day course was given that covered OO concepts and how to read OO documentation.
- For those that had to review actual software designs, a half-day Ada packaging and design course was administered.
- To support groups facing a very tight schedule, a special 1-day Preliminary OOD course helped the groups reach their Preliminary Design Review.
- A fifth day of OOD training was added to provide engineers involved in realtime applications to learn how objects fit into their project-specific realtime software architecture.
- After assessing the background of the various students, significant modifications were made to the Advanced Ada course to address basic data structures and reusable components engineering.

Integrated with the formal training program was a more loosely structured mentoring function. Fastrak instructors worked very closely with students in a post-classroom capacity in an effort to help guide their way through critical areas of development. Mentors provided key experience in computer science and basic data structures, realtime applications, software reuse, and general software engineering. A great deal of OO mentoring resources were dedicated to acquiring feedback from students who were applying the methods in an effort to continuously improve the training materials.

Foundations in Object Technology

For a training effort as comprehensive as this, we needed to first establish a sound foundation in object-oriented resources. Starting with the human resources, CAE-Link assigned David Weller, William Wessale, and later, George Heyworth as my technical contacts for the training program. These individuals were very well versed in object technology and/or Ada, and thus provided a clear vision into what kind of training program CAE-Link needed.

My personal background coming into the training program included over eight years at McDonnell Douglas in Houston, most of which were dedicated to seeing the company help NASA transition to object technology and Ada. During this time, I was heavily involved with the University of Houston at Clear Lake where I ultimately wrote a Master's thesis on the need for an object-oriented approach to requirements analysis (May 1988). My first exposure to formal OO methods followed shortly with the early works of Sally Shlaer and Stephen Mellor on OO Analysis.

By the time the CAE-Link training program commenced, several new texts had appeared and were in large circulation. Our OO courses were ultimately based primarily on the following works:

- Grady Booch, *Object-Oriented Design with Applications* (1991)
- James Rumbaugh, et al, *Object-Oriented Modeling and Design* (1991)
- Colin Atkinson, *Object-Oriented Reuse, Concurrency, and Distribution—An Ada-Based Approach* (1991)
- Ed Seidewitz and Mike Stark, *Principles of Object-Oriented Software Development with Ada* (1992)

Booch brought forth a legacy with Ada and the recognition as a leading OO methodologist. His text proved to be the most sound basis for OO concepts and was distributed to all students.

While Booch is strong in OO design (proper), many feel that his method does not adequately address the analysis phase. Rumbaugh, et al, provided a very appropriate front-end complement to Booch's work.

The Atkinson text had its greatest influence in the details of representing OO designs specifically in Ada. Atkinson was a key player in the development of the DRAGOON language, which is an Ada-based OOPL. Since DRAGOON source code translates to an Ada intermediate representation, his representation techniques proved to be very helpful in dealing with Ada's limited support for OOP.

The work of Seidewitz and Stark formed the initial basis for our material. Their effort represented a leading technique in applying OO development specifically within an Ada environment.

Resulting Methodology

The courseware used at CAE-Link evolved significantly from that used for the initial classes. A strategic decision was made to tailor the previously identified methods, resulting in the following:

- The OORA technique is very graphically oriented, language-independent, and logical in its nature of modeling the problem domain.
- The OOD technique is very textually oriented, language specific, and is subjected to project-specific architectural constraints.

OORA is segmented into Static Analysis, Dynamic Analysis, Object Classification, and Class Specification. Static Analysis and Object Classification are very consistent with the Object Modeling Technique (OMT) defined by Rumbaugh, et al. Static analysis takes full advantage of OMT's treatment of associations and aggregation as they apply to requirements analysis.

The key mechanisms of Dynamic Analysis are Event and Effects Tables, Object-Message Diagrams and Harel Statecharts. Event tables help to analyze the abstract behavior of a system (or some component thereof) from a stimulus-response point-of-view. Each event-effect pair may be analyzed using an Object-Message Diagram, which provides a means for illustrating the object-message "chaining" that results when a given event occurs. Lastly, Harel Statecharts provide an excellent improvement over standard state transition diagrams for modeling state-dependent behavior.

The notation adopted by CAE-Link is unique and was made so to meet the constraints of the specific CASE tool imposed by NASA. However, the notation is analogous with that of the authors previously stated. (In fact, Fastrak later defined a formal mapping of the CAE-Link notation into the equivalent OMT constructs so that the basic OORA course could be taught in a public forum.)

The OOD course focuses on mapping the language-independent OORA models into canonical forms of Ada packages. Variations exist between realtime and non-realtime forms, but these variances are managed in the private part or the body of each class's package. OOD addresses alternative mappings that might be needed, as well as the limitations of implementing OO designs into an "object-based" language such as Ada. Since software reuse is a major initiative for many NASA projects, the course has specific exercises that force students to adapt their existing work into meeting new requirements.

Magnitude of the Training

The following table summarizes the amount of total training delivered for this contract:

Course	No. Deliveries	Tot. No. Students
OORA	14	255
OOD	11	142
Ada Syntax	15	211
Advanced Ada	14	158
	TOTAL	766
OO Concepts	5	92
Ada Design	5	108
Preliminary OOD	3	22
	TOTAL	222
	GRAND TOTAL	**988**

Course Format

All training courses employed a balance of lecture and student application. The lectures focused on presenting the concepts, terminology, and notations, using examples and exercises to illustrate small-scale solutions. Daily workshops were also used, placing students into teams as to better simulate real project dynamics. Team leads were often assigned to make presentations representing their group's incremental solutions to larger-sized problems.

Measuring the Effectiveness of the Training

Upon completion of a course, each student was required to formally evaluate the course. The quantitative results were averaged and summarized in a standardized course evaluation summary. This report was distributed to key CAE-Link and Fastrak personnel so that delivery results and quality improvement could be continuously monitored. A database was maintained that kept track of all courses delivered, student attendance, and evaluation grades. Trends were easily identified by graphically depicting the grades as a function of course and time. As would be expected, the quality and effectiveness rose steadily for each of the four courses offered in the curriculum.

Something rather unique to CAE-Link's training program is that proficiency measurement was explicitly called out for in the training contract. Fastrak was required to administer both objective and a subjective examinations at the end of each course delivered. Those students taking the course for "credit" had to demonstrate basic proficiency in the course's subject matter. Such measures greatly assisted in verifying that students met the prerequisites for advanced courses. The students' grades and the tests themselves were kept confidential with the instructors and were used only for the purpose of ensuring student proficiency.

Lessons Learned

After 20 months of administering the OODAT program for CAE-Link, we come to the following conclusions:

- OORA can be used to generate native requirements specifications ("to-be" models), or to model existing systems ("as-is" models), making it applicable to both forward and reverse engineering.
- Inheritance should be deferred to an "advanced" phase of the OORA training. There is enough to learn in basic object and dynamic modeling to consume the first three days of training. Inheritance, polymorphism, and other more OOD/OOP concepts are best presented on Day 4.
- We found it advantageous to explicitly separate the OORA and OOD courses on the basis of where and when to introduce language dependency and software architecture; OORA should produce logical models that are independent of these concerns.
- A uniform process and notation should be used for both analysis and design, and should be independent of implementation language and realtime vs. non-realtime requirements.
- We found that a simple compositional approach worked best for simulating inheritance in Ada. This required a lot of manual respecification in the subclasses, and thus constrained designers to temper their inheritance hierarchies to 3-4 levels. But the approach proved to be more efficient and maintainable than using variant records, and more modular than using nested or "flattened" packages. The compositional approach was also deemed to be more migratable to how inheritance is implemented in the upcoming revision to the Ada language, currently known as Ada 9X.[1]
- Although standard Ada does not support dynamic polymorphism very directly, its rich forms of overloading and generics provided some reasonable alternatives.[2]
- Portions of a training system need global visibility to all data in a system. This required special tools that circumvent the data protection inherently provided by encapsulation techniques. These tools were necessary so as to prevent generalized source-level access to hidden data.
- A standardized software architectural definition is required, and must be defined and put into place prior to developing the software design. This is especially true for realtime systems.
- Realtime performance requirements can be met while preserving most object-oriented principles. The resulting software quality is significantly better than could be achieved using more traditional approaches to simulation design.
- The OO development process proved to be the least mature, and thus, the most difficult aspect of the training program to teach.
- Students familiar with user-defined types in strongly-typed languages learn the object paradigm easier than those without such exposure. We feel that this was due to the analogy between the object/class relationship found in OOA&D and the variable/type relationship found in modern languages.
- Students with traditional engineering backgrounds will generally require more introductory training in basic computer science due to their lack of exposure to it in their formal education.
- It helps to have instructors be well-versed in several OO methods and programming languages, for they then stand a better chance to demonstrate an objective and credible posture. It helps even more to have customer training liaisons that are knowledgeable in the technical nature of the training program (as was the case here).

Status and Future Directions

The OODAT program is basically completed. Continued training will be subject to CAE-Link's future growth and attrition. Outside of the CAE-Link effort, Fastrak has found that a combination OMT/Booch method serves very well for general training purposes. The notation used in Fastrak's public OOA course is based on OMT and has proven to be very effective for introducing basic object modeling. While not as semantically expressive as that of Booch, the OMT notation is adequate for analysis purposes, and is certainly easier to work with

---

[1] Although derived types in standard Ada 83 provide "half" of inheritence (i.e., the operations), we found no way to effectively use the feature. However, this deficiency is well remedied in Ada 9X.

[2] This deficiency is also addressed in Ada 9X.

in lieu of having CASE tool support. However, when getting into the "heavier" design concerns, more and more of Booch's method comes into its own. This is especially seen with respect to detailed design and code generation concerns. And even with the contributions of OMT and Booch, portions of the overall training can still be augmented by selected aspects of Seidewitz, Wirfs-Brock, and Atkinson.

At the time of this writing (November 1993), Booch's latest edition of his OOD book ("Booch'94") is finding its way into the mainstream. Booch, along with HP's *Fusion* method, are seen as attempts to integrate the "best of all worlds." The *Use Cases* of Jacobson and *Object Behavior Analysis* from Rubin and Goldberg also stand to make some contribution to the OO methods community.

In general, however, we see little room for completely new OO methods. As the concepts and terminology become more stable, a "yet another notation" syndrome will begin to settle into the minds of many (if it hasn't already). Thus, a maturing of existing methods is more likely to result rather than a proliferation of new methods.

On the CASE front, some really good OO tools are (finally) appearing. The ones that offer multiple methodology support and end-user configurability—the "meta-CASE" tools—are the most promising. And the whole area of "visual programming" could very well revolutionize what is meant by "programming." But even with all this, I feel that the next generation of CASE technology must better manage the dependency between object models and their corresponding source code, specifically in synchronizing changes made to either level of abstraction.

As for a future with Ada 9X, there will certainly be areas of improvement for all existing Ada-based OOD courses. Ada 9X will include, among other things, direct support of single inheritance and dynamic polymorphism—the two additional elements of programming required to make Ada a "true" OOPL. These enhancements will simplify the representation of inheritance, and thus reduce the amount of code needed to implement OO designs.[3]

But just as C++ is said to provide for a "better C," Ada 9X goes beyond simply providing direct support for OOP. The CAE-Link realtime software architecture currently employs "work-arounds" that will be much better implemented in the revised language. Thus, Ada 9X will bring forth a whole new realm of opportunity to CAE-Link, to the Ada community, and to the realtime and OO communities in general.

Contact Information:

Gary J. Cernosek
1303 New Cedars Dr.
Houston, TX 77062
(713) 280-8508
Internet: *cernosek@source.asset.com*

---

[3] For those so inclined, you may contact the author for a paper presented to the Tri-Ada'93 conference that presents a technique for representing object models in Ada 9X notation.