Executive Summary

This case study is based on an enterprise-wide consulting project for a financial services firm in a major metropolitan area of the southwest United States. This case addresses the underlying principles (i.e., techniques and processes) and real-world practical application of object orientation (O-O). The objectives of this case study are to reinforce the student’s foundation in fundamental O-O concepts, to provide an in-depth example of the application of O-O analysis and design techniques and formalisms and to enable the student to transfer this knowledge to the student’s actual work.

The merit of the development process described here, Compass, is that it presents a repeatable process for delivery of client server architectures, object oriented systems, and distributed objects and components. It helps manage three interdependent variables common to most projects, i.e., deliverables, resources, and time, in a cost effective and efficient manner. The concept behind Compass is to integrate the best-proven solutions currently available, drawing upon several best-of-breed approaches used within the Information Services industry.

Background

EDS is implementing a mission-critical business system, which is the cornerstone for the customer’s strategic vision. For EDS/customer confidentiality reasons the company will not be named.

The system is being developed for a company that wants to grow its customer base by a focused targeting of niche markets. In order achieve this goal it was deemed necessary to expand the technological capabilities provided by the current mainframe based system. These capabilities would allow the business to reach potential customers in new ways such as the Internet / Intranet, kiosks, and “portable” business offices using laptop computers. The new system was also required to provide flexibility in areas such as new product creation.

The motivation for using the O-O paradigm for software development is illustrated by a closer look at two primary approaches to software development. These are the function-oriented paradigm, where functions are defined first and data is defined later, and the data-oriented paradigm, where data
are defined first and functions are defined later. Both these approaches have their drawbacks. The drawbacks of the function-oriented paradigm are as follows:

- The difficulty in maintaining the consistency between the data flow diagram (DFD) and the data dictionary.
- The difficulty in eliminating or minimizing the “ripple-effect”.
- The difficulty in software reuse due to tight coupling between the functions, and treating data and functions separately.
- The difficulty in following data flows in structured analysis and building hierarchies of tasks in structured design.

The drawbacks of the data-oriented paradigm are as follows:

- The difficulty in maintaining the consistency between the process model and the snapshot model.
- The difficulty in comprehending the effect of the interaction between the functions which are not explicitly modeled in the process or transaction specification language.
- The difficulty in software reuse since data and functions are treated separately.
- The difficulty in converting an analysis model into a good software design.

The object-oriented (O-O) approach attempts to give a balanced treatment of functions and data. An object is a state and a set of methods that explicitly embodies an abstraction characterized by the behavior of relevant requests and an entity that has the state and functionality. In the O-O context, an object is “an instance of a class” (www.cyberdyne-object-sys.com/oofaq). Thus the class is the fundamental building block of OO software. A class determines everything about an object (Elmasri and Navathe, 1993; Brown, 1997). A class defines a data type where a type consists of both a set of states and a set of operations which transition between those states. A class provides a set of (usually public) operations, and a set of (usually non-public) data attributes representing the abstract values associated with instances of the type. Object-oriented concepts such as inheritance, polymorphism, encapsulation, and data abstraction that lead to compatibility, flexibility, reuse, extensibility, and easier maintenance are the primary reasons for using the O-O paradigm in this case.

Distributed objects are extended objects that are not restricted to a single program and can exist as independent entities and be remotely accessible by other objects. Rather than producing monolithic applications, distributed object systems tend to consist of a number of small units or components. Components are the smallest, self-managing, and independent parts of a distributed-object model that work in heterogeneous environments (Lewandowski, 1998). The key benefits of distributed objects are that they allow the development of scaleable client/server systems by virtue of modularized software that features interchangeable parts and an option to add components for custom solutions in advanced architectures.

Setting the Stage

EDS and members of the existing information systems (IS) staff defined a process to gather defined project metrics for quantification of the results and a formal testing process in which deliverables must pass through a series of “quality-gates” (Q-Gate).

Compass, which was developed by EDS, is being used to provide a formal OO development process and OO methodology support. Like any of the popular O-O analysis and design approaches, Compass should not be seen as a definitive and comprehensive source of guidance for O-O software development. It should be treated as a set of pragmatic guidelines and instructional tools where the focus is on sensible use of formalism and process in laying the foundation for reliable O-O software development.

WHY IS ANOTHER PARADIGM NEEDED?

Current research indicates that practitioners are not necessarily using the methodologies as