An Overview of Object-Oriented Systems

INTRODUCTION
Software development is dynamic and always undergoing major change. The methods we will use in the future no doubt will differ significantly from those currently in practice. We can anticipate which methods and tools are going to succeed, but we cannot predict the future. Factors other than just technical superiority will likely determine which concepts prevail.

Systems development refers to all activities that go into producing an information systems solution. Systems development activities consist of systems analysis, modeling, design, implementation, testing, and maintenance. A software development methodology is a series of processes that, if followed, can lead to the development of an application. The software processes describe how the work is to be carried out to achieve the original goal based on system requirements.

Furthermore, each process consists of a number of steps and rules that should be performed during development. The software development process will continue to exist as long as the development system is in operation.

TWO ORTHOGONAL VIEWS OF THE SOFTWARE
Object-oriented systems development methods differ from traditional development techniques in that the traditional techniques view software, as a collection of programs (or functions) and isolated data. What is a program? Nicklaus Wirth, the inventor of Pascal, sums it up eloquently in his book entitled, interestingly enough, Algorithms + Data Structures = Programs: "A software system is a set of mechanisms for performing certain action on certain data."

There are two different, yet complementary ways to view software construction: In traditional system we can focus primarily on the functions or primarily on the data. The heart of the distinction between traditional system development methodologies and newer object-oriented methodologies lies in their primary focus, where the traditional approach focuses on the functions of the system—What is it doing?—object-oriented systems development centers on the object, which combines data and functionality.

OBJECT-ORIENTED SYSTEMS DEVELOPMENT METHODOLOGY
Object oriented development offers a different model from the traditional software development approach, which is based on functions and procedures. In simplified terms, object-oriented systems development is a way to develop software by building self-contained modules or objects that can be easily replaced, modified, and reused. Furthermore, it encourages a view of the world as a system of cooperative and collaborating objects. In an object-oriented environment, software is a collection of discrete objects that encapsulate their data as well as the functionality to model real-world "objects." An object orientation yields important benefits to the practice of software construction. Each object has
attributes (data) and methods (functions). Objects are grouped into classes; in object-oriented terms, we discover and describe the classes involved in the problem domain.

In an object-oriented system, everything is an object and each object is responsible for itself. For example, every Windows application needs Windows objects that can open themselves on screen and either display something or accept input. A Windows object is responsible for things like opening, sizing, and closing itself.

The Object-Oriented environment emphasizes its cooperative philosophy by allocating tasks among the objects of the applications. In other words, rather than writing a lot of code to do all the things that have to be done, you tend to create a lot of helpers that take on an active role, a spirit, and that form a community whose interactions become the application. Instead of saying, "System, compute the payroll of this employee", you tell the employee object, "compute your payroll." This has a powerful effect on the way we approach software development.

**WHY AN OBJECT ORIENTATION?**

Object-oriented methods enable us to create sets of objects that work together synergistically to produce software that better model their problem domains than similar systems produced by traditional techniques. The systems are easier to adapt to changing requirements, easier to maintain, more robust, and promote greater design and code reuse. Object-oriented development allows us to create modules of functionality. Once objects are defined, it can be taken for granted that they will perform their desired functions and you can seal them off in your mind like black boxes. Here are some reasons why object orientation works:

- **Higher level of abstraction:** The top-down approach supports abstraction at the function level. The object-oriented approach supports abstraction at the object level. Since objects encapsulate both data (attributes) and functions (methods), they work at a higher level of abstraction. The development can proceed at the object level and ignore the rest of the system for as long as necessary. This makes designing, coding, testing, and maintaining the system much simpler.

- **Seamless transition among different phases of software development:** The traditional approach to software development requires different styles and methodologies for each step of the process. Moving from one phase to another requires a complex transition of perspective between models that almost can be in different worlds. This transition not only can slow the development process but also increases the size of the project and the chance for errors introduced in moving from one language to another. The object-oriented approach, on the other hand, essentially uses the same language to talk about analysis, design, programming, and database design. This seamless approach reduces the level of complexity and redundancy and makes for clearer, more robust system development.

- **Encouragement of good programming technique:** A class in an object-oriented system carefully delineates between its interface (specifications of what the class can do) and the implementation of that interface (how the class does what it does). The routines and
attributes within a class are held together tightly. In a properly designed system, the classes will be grouped into subsystems but remain independent; therefore, changing one class has no impact on other classes, and so, the impact is minimized. However, the object-oriented approach will promote perfect design or perfect code by raising the level of abstraction from the function level to the object level and by focusing on the real-world aspects of the system, the object-oriented method tends to promote clearer designs, which are easier to implement, and provides for better overall communication. Using object-oriented language is not strictly necessary to achieve the benefits of an object orientation. However, an object-oriented language such as C++, Smalltalk, or Java adds support for object-oriented design and makes it easier to produce more modular and reusable code via the concept of class and inheritance.

- **Promotion of reusability:** Objects are reusable because they are modeled directly out of a real—world problem domain. Each object stands by itself or within a small circle of peers (other objects). Within this framework, the class does not concern itself with the rest of the system or how it is going to be used within a particular system. This means that classes are designed generically, with reuse as a constant background goal. Furthermore, the object orientation adds inheritance, which is a powerful technique that allows classes to be built from each other, and therefore, only differences and enhancements between the classes need to be designed and coded. All the previous functionality remains and can be reused without change.

**OVERVIEW OF THE UNIFIED APPROACH**

The unified approach is a better understanding of object-oriented concepts and system development. The UA, based on methodologies by Booch, Rumbaugh, and Jacobson, tries to combine the best practices, processes, and guidelines along with the Object Management Group’s unified modeling language. The unified modeling language (UML) is a set of notations and conventions used to describe and model an application. Figure 1—depicts the essence of the unified approach. The heart of the UA is Jacobson’s use case. The use case represents a typical interaction between a user and a computer system to capture the user’s goals and needs. In its simplest usage, you capture a use case by talking to typical users and discussing the various ways they might want to use the system. The use cases are entered into all other activities of the UA.

The main advantage of an object-oriented system is that the class tree is dynamic and can grow. After your first few projects, you will accumulate a repository or class library of your own, one that performs the operations your applications most often require.

The UA uses a layered architecture to develop applications. Layered architecture is an approach to software development that allows us to create objects that represent tangible elements of the business independent of how they are represented to the user through an interface or physically stored in a database. The layered approach consists of view or user interface, business, and access layers. This approach reduces the interdependence of the user interface, database access, and business control; therefore, it allows for a more robust and flexible system.
In an object—oriented environment, software is a collection of discrete objects that encapsulate their data and the functionality to model real-world "objects." Once objects are defined, you can take it for granted that they will perform their desired functions and so seal them off in your mind like black boxes. Your attention as a programmer shifts to what they do rather than how they do it. The object—oriented life cycle encourages a view of the world as a system of cooperative and collaborating agents.
An object orientation produces systems that are easier to evolve, more flexible, more robust, and more reusable than a top—down structure approach. An object orientation

• Allows working at a higher level of abstraction.
• Provides a seamless transition among different phases of software development.
• Encourages good development practices.
• Promotes reusability.

The unified approach (UA) is the methodology for software development proposed and used and is based on the Booch, Rumbaugh, and Jacobson methodologie. The UA consists of the following concepts:

• Use-case driven development.
• Utilizing the unified modeling language for modeling.
• Object—oriented analysis (utilizing use cases and object modeling).
• Object—oriented design.
• Repositories of reusable classes and maximum reuse.
• The layered approach.
• Incremental development and prototyping.
• Continuous testing.