Benefits of Object-Orientation

Software systems that incorporate object-oriented design techniques realize several benefits:

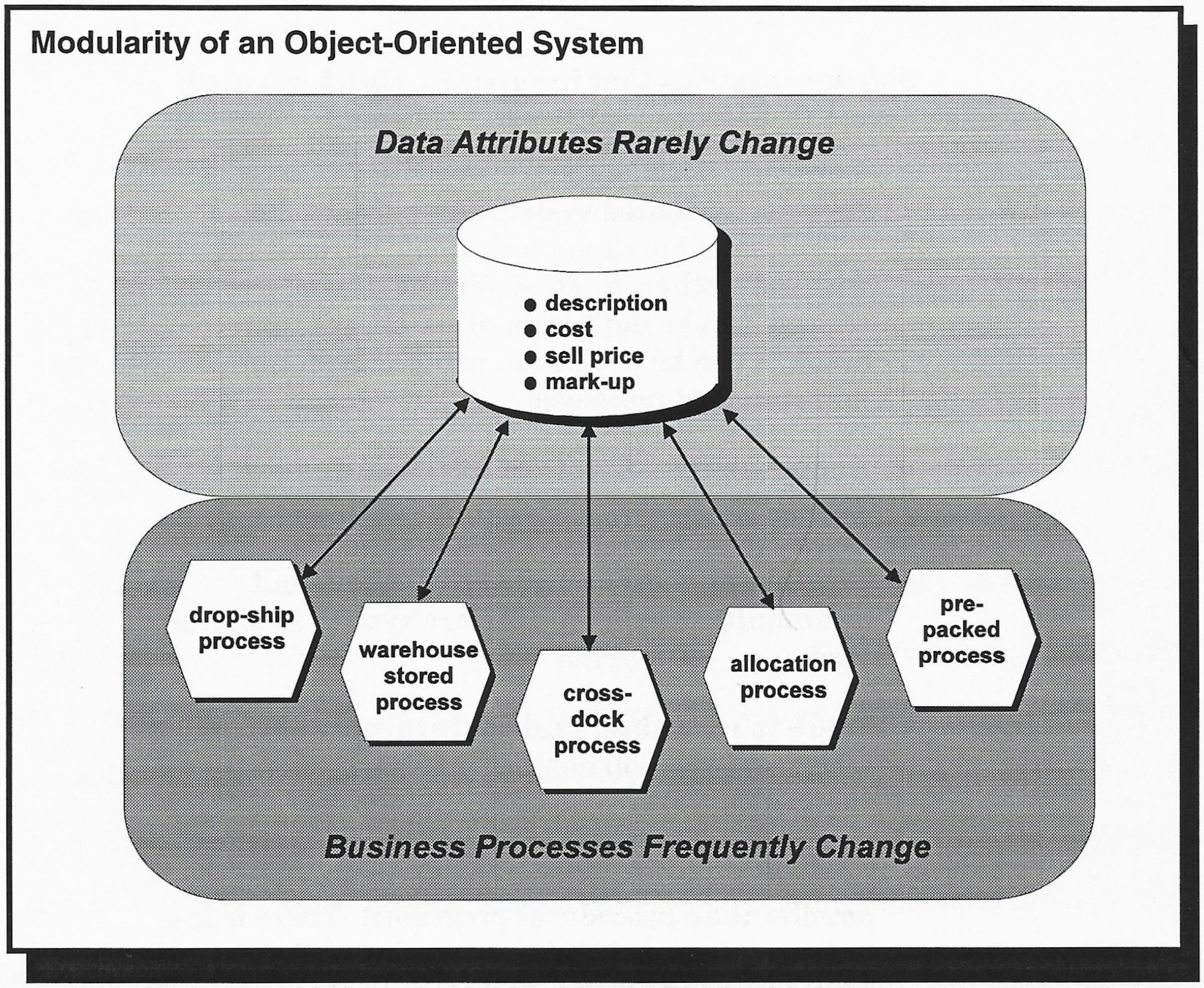
- **Complex systems are simplified**
  An object-oriented system can be viewed as a collection of building blocks (objects) that interact with each other, as shown in the previous diagram. Objects can be built from other objects, and complete systems can be developed using proven components and standard messages.

- **Accurate models of reality are created**
  An object-oriented system models the way that reality is understood by people. Analysts, programmers, and users can all think and communicate about a software system in similar terms.

- **Code is reusable, and maintainable**
  Object-orientation enables objects to be independently designed, coded, and tested. Since objects can be reused without having to code additional programs, programs tend to be much smaller than procedural programs. There is less code to debug and maintain, and modifications that would normally be applied to multiple programs can often be made to a single object.

- **Programs are modular**
  Retail business processes change more frequently than the actual data requirements. Therefore, data attributes should be maintained separately from the business logic and business logic should be modular so that a change made to one processes does not affect another process. Object-orientation services both of these objectives, as shown in the following diagram. In this example, the data attributes for a Stock Keeping Unit (SKU) are distinctly separate from business processes. Also, the replenishment processes are coded separately from each other.

Benefits of Object-Orientation



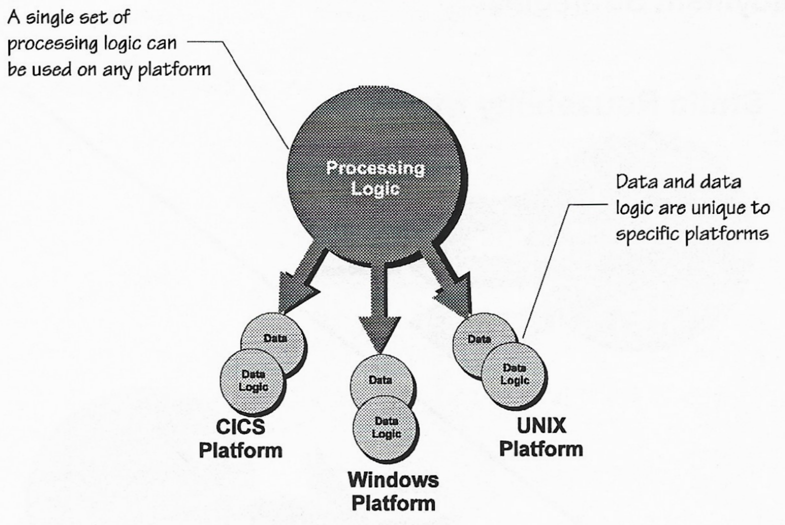
Benefits of Object-Orientation

- **Programming productivity increases**
  The software industry is currently experiencing a demand for more software at a lower production cost. Object-orientation, because of its simplicity and reusability, enables systems to be designed and implemented in less time, using less code.

- **Programs are portable**
  Object-oriented design techniques are ideal for planning and organizing client/server applications because a significant amount of code can be written without regard to the physical computing platform. Platform-independent processing logic can be located anywhere on the network, and platform-dependent data logic can be easily moved to the same platform as the data.

  The following diagram shows how platform-specific data and data logic can be kept separate from the processing logic.

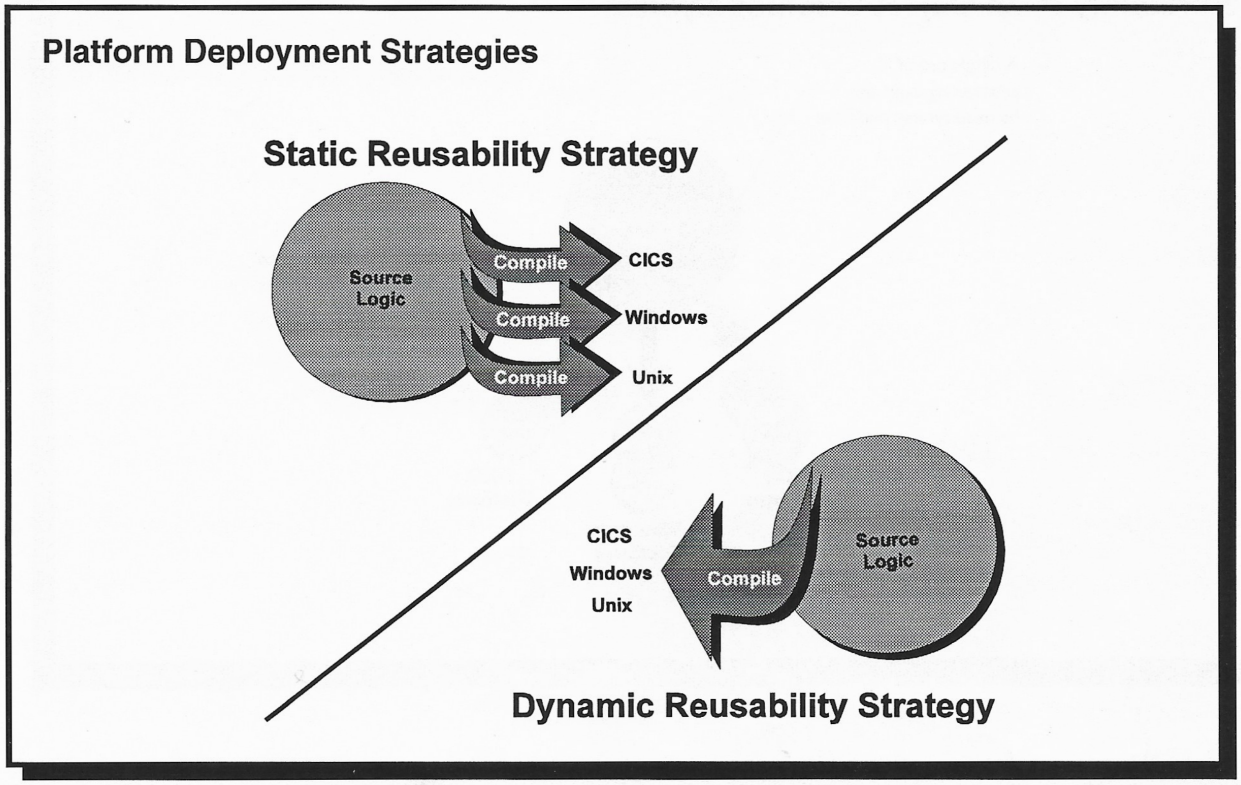
Portability of an Object-Oriented System


A single set of processing logic can be used on any platform
Processing Logic
Data and data logic are unique to specific platforms
Data
Data Logic
CICS Platform
Data
Data Logic
Windows Platform
Data
Data Logic
UNIX Platform

Benefits of Object-Orientation

- **Source code can be easily deployed**
  Object-oriented design techniques facilitate client/server computing by enabling source code to be deployed to various platforms. Two source code deployment methods can be used:
  - **Dynamic reusability**
    Source code is moved between platforms without being recompiled. This method is dependent upon a number of factors, including the use of an object-oriented programming language, such as Object COBOL.
  - **Static reusability**
    A deployment strategy in which a single set of source code is recompiled for execution on various platforms.

The following diagram illustrates the concepts of static and dynamic reusability.


Platform Deployment Strategies
Static Reusability Strategy
Source Logic
Compile
Compile
Compile
CICS
Windows
Unix
CICS
Windows
Unix
Compile
Source Logic
Dynamic Reusability Strategy

Appendix D Glossary

A

Attributes
The properties of data elements within a system.

B

Business item
A single, discrete item within a business software system, such as a purchase order or an invoice.

C

CICS Data Link Interface
The interface between the GENESIS Data Access Objects and data that resides in files.

Class
Defines the data structures and routines for a group of similar objects. A GENESIS class is a program shell that contains parts of working storage and procedure division code. This information is used as a template by each instance (object) of the class.
See also: object

Client/server
Any software system, regardless of platform, where one application asks for service (client) and another application performs the requested service (server).

Common System programs
Core programs and objects that are used by all application programs.

Configuration section
The portion of a COBOL II program that provides information such as the type of computer on which the program will be compiled and the type of computer on which it will run.

Conversational Program
A type of program or system that interacts with a user, alternately accepting input and then responding to the input quickly enough for the user to maintain his or her train of thought.
Contrast with: Pseudo-Conversational Program

Copybook
Standard, reusable code that is copied into an object when the object is compiled. In GENESIS, copybooks are used to define discrete components such as paragraphs of code, file layouts, or to simulate messages.
See also: Messages

D

Data Division
The portion of a COBOL II program that declares all variables and data structures. The Data Division consists of the File Section, the Working-Storage Section, and the Linkage Section.
See also: File Section, Working Storage Section, Linkage Section

Data Entity
An item of data that can be treated as a unit, often as a member of a particular category or type.

E

Encapsulation (Information Hiding)
A protective encasement that hides the implementation details of an object, making its data accessible only by operations that are also encased within the object. This technique simplifies maintenance, defines ownership of data, and ensures platform-independence and data integrity.

Environment Division
The portion of a COBOL II program that describes hardware the program needs and relates that information to the files used by the program. The Environment Division consists of the Configuration Section and the Input-Output Section. For GENESIS, the Environment Division is only used by Common Systems programs.
See also: Configuration Section, Input-Output Section

F

File Section
The portion of a COBOL II program that defines file formats and characteristics. For GENESIS, this section is only used by Common Systems batch programs, and it only contains the companion copybooks for the sequential files defined in the Environment Division.

I

Identification Division
The portion of a COBOL II program that specifies information about the program, such as the program and object class identification, program description, and a record of all changes made to the program.

Inheritance
Enables the features of an object class to be physically available to, or reusable by, its subclasses as though they were features of the subclass. Class-to-object inheritance allows instances of a class (objects) to inherit default values from the class. Object-to-object inheritance transfers the state of one object to another.