

## Methodology and Technology Services



www.ies.aust.  
Web com

| [Home](#) | [Courses](#) | [Certification](#) | [Projects](#) | [Papers](#) | [EA Blog](#) | [Online Store](#) |  
[Contact Us](#) |

### A Visible Solution Paper

# Object-Orientation and Business-Driven Information Engineering

[Printable PDF Version](#)

By Robin Peel and Keith Harmon

Copyright © 1997, Visible Systems Corporation

---

## Contents

### ■ [Object-Orientation and Business-Driven Information Engineering](#)

- [Introduction](#)
- [Business-driven Information Engineering](#)
- [Object-Oriented Analysis and Design - Concepts and Terminology](#)
- [Benefits of Object-Orientation](#)
- [Implementing Object-Oriented Methodologies](#)
- [Re-engineering Legacy Systems](#)
- [Summary](#)
- [Bibliography - Sources and Further Reading](#)

[Home](#)  
[Courses](#)  
[Certification](#)  
[Projects](#)  
[Papers](#)  
[TEN Archive](#)  
[EA Blog](#)  
[Contact Us](#)  
[Search](#)  
[Links](#)  
[Online Store](#)

- [Comments on Object Oriented - Information Engineering Analysis](#)
- [Additional References](#)

---

## Introduction

Object oriented analysis, design, and system development techniques have become increasingly popular as a means of meeting the requirement for better tools to support complex systems. Information Engineering is a popular methodology for bridging the gap between business requirements and systems. These two sets of methodologies have similarities and complementary features that will be examined in detail.

Object-oriented programming has its roots in the simulation languages of the late 1960s, and has recently developed in response to a requirement for better tools to support the implementation of complex systems. The resulting programming languages (e.g. C++ and Smalltalk) permit or require the use of "objects", with the goal of producing complex applications which are more reliable, maintainable, and expandable than is possible or practicable with conventional programming techniques.

Object-oriented analysis and design methodologies have emerged to bridge the gap between business requirements and the physical implementation of a system. The techniques are focused upon the design of logical models and physical designs that will be implemented using object-oriented programming tools. They use a range of mapping, diagramming, modeling and documentation conventions and techniques. They model business requirements in terms of 'real world' objects and their interactions. See the Bibliography at the end of this paper for references to many of the common methodologies.

---

## Business-Driven Information Engineering

Information Engineering, as originally defined by Clive Finkelstein and James Martin in 1981, has evolved into two main variants:

- 'DP-driven Information Engineering', supported by James Martin, uses existing functions of an organization as a starting point; processes and data needed by the functions are then defined, and the business knowledge is captured, often through interviews with users.
- 'Business-driven Information Engineering', developed by Clive Finkelstein (and referred to as IE in this paper) captures strategic business requirements in terms of business statements which define the

organization's mission, goals, strategies, objectives and policies, and links these to a logical, technology-independent model of the data and processes required to achieve those statements. The resultant logical models then serve as a foundation from which physical systems can be designed and implemented.

Both business-driven IE and object-oriented analysis and design define business requirements using real-world terms. This narrows the gap between business experts and those who will implement and maintain the systems by generating logical models that are both unambiguous to developers and understandable to business experts. The focus of the two sets of methodologies is different (but complementary), as illustrated in the following table:

<b>Object-oriented analysis and design</b>	<b>Business-driven Information Engineering</b>
Object-oriented analysis and design techniques develop and link logical models to a physical system implementation.	The IE methodology provides a rigorous link between the logical data and process models and both the requirements of the business (as documented in a set of agreed business planning statements) and the physical system implementation.
An object-oriented implementation environment is assumed.	No assumptions about the proposed implementation environment are made during data and process modeling. Implementation can take place in either an object-oriented or other environment.
An object-oriented design carries the benefits of modularity and encapsulation from the analysis stage through to the completed application.	Data and process dependencies, which imply the boundaries of modules, are identified automatically and/or defined manually. Data access processes (DAPs) encapsulate the data upon which they act.

The use of objects implies a data-first design. A recurring theme in the object-oriented texts is that the data and their inter-relationships are much less vulnerable to changing requirements than the operations performed on the data.

Forward engineering (top-down) using business-driven IE is a data-first modeling methodology. Business changes bring about changes in procedures and processes more often than changes in the data upon which the business is based. A mature data model is a sound foundation from which to model changing business requirements.

IE also allows bottom-up modeling (for reverse engineering and systems migration and integration). The resulting data and process models can be implemented in either traditional (e.g. relational) or object-oriented environments. This will facilitate translation from one environment to another.

---

## Object-Oriented Analysis and Design - Concepts and Terminology

The fundamental concept of object-orientation is the "object," which combines data structure and behavior in a single component. The inner workings of an object may be hidden. Only its visible interfaces with the rest of the world need to be understood by any other processes (or objects) that interact with it. Objects with the same data structure and behavior are grouped into an object "class;" libraries of class definitions are an important asset, and are the basis for the reusability resulting from the object-oriented approach. This is in contrast to "conventional" programming, in which data structures and their behaviors are defined separately and are only loosely connected [Rumbaugh, 1991, p1].

There is an obvious mapping between the object classes of an object-oriented design and the entities in an IE data model. The implementation of the business-driven IE methodology in the CASE tool Visible Advantage™ uses the concept of "data access processes" (DAPs) to define the basic processes which can act upon a specific entity (documented as *Create*, *Read*, *Update* and *Delete* DAPs for each entity). Higher-level business processes (which are initiated in response to "business events") are documented in terms of the various DAPs which are required to interact with the data model (e.g. **Create EMPLOYEE**). Using the IE

methodology the definition of an entity, accompanied by its available DAPs, forms an encapsulated unit which provides the functionality of the OO object class definition. The IE definition of an entity also includes the linkages with the business planning statements.

[Booch, 1994] offers precise definitions of object-oriented analysis and design:

- *Object-oriented analysis is a method of analysis that examines requirements from the perspective of the classes and objects found in the vocabulary of the problem domain.*
- *Object-oriented design is a method of design encompassing the process of object-oriented decomposition and a notation for depicting both logical and physical as well as static and dynamic models of the systems under design.*

To support object-oriented analysis, design and programming methodologies, a nomenclature has developed to describe the key concepts. The following table lists and describes these concepts, and clarifies their relationship with the business-driven IE methodology and some of the features of Visible Advantage™:

	<b>OO concept</b>	<b>IE's relationship to OO concept</b>
<b>Class or Object class:</b>	<p>Objects with the same data structure and behavior (in the context of the particular application environment) are grouped into an object "class". Classes can be grouped in a hierarchical structure. Classes may have <i>attributes</i> (see below).</p> <p><i>Example: An object class of PERSON.</i></p>	<p>An entity in the data model with its associated Data Access Processes (DAPs). DAPs govern the behavior or operations on an entity.</p> <p>The IE definition of an entity additionally includes the linkages to the business planning statements.</p>

<p><b>Object instance:</b></p>	<p>A particular real-world occurrence of an object. Each instance of an object is assumed to be unique. OO assumes that all objects in an object class have a unique identity that separates and identifies them from all other object instances.</p> <p><i>Example: Robin Peel is an instance of the object class <b>PERSON</b>.</i></p>	<p>An occurrence of an entity or group of related entities.</p> <p>IE expressly identifies characteristics that identify one occurrence from another, in the form of a primary key. A system-generated primary key identifies entities that do not possess a natural primary key.</p>
<p><b>Attribute:</b></p>	<p>A data value which can be held by the objects in a class. Attributes may be assigned to different data types (e.g. integer).</p> <p><i>Example: An attribute of the class <b>PERSON</b> is last name (of type string).</i></p>	<p>An attribute within an entity. Attributes are assigned a domain, which define the valid data types. Domains offer additional flexibility by defining length, range and a specific valid set of values for one or more attributes.</p>
<p><b>Association or Relationship:</b></p>	<p>A conceptual relationship between instances of an object. Associations have <i>cardinality</i> including one-to-one, one-to-many, and many-to-many. Most object-oriented texts do not address the <i>nature</i> of an association (i.e. mandatory or optional), except in the definition of an object's behavior.</p> <p><i>Example: The object <b>CUSTOMER</b> is associated with the object <b>ORDER</b>. This will probably have a cardinality of one-to-many.</i></p>	<p>IE captures cardinality as <i>degree</i>.</p> <p>IE additionally captures:</p> <ul style="list-style-type: none"> <li>● Association <i>nature</i> (i.e. mandatory or optional).</li> <li>● <i>Delete rules</i> for referential integrity.</li> <li>● Whether the association is used in the</li> </ul>

		<p>identification of a child entity (i. e. cascaded as primary key or foreign key).</p>
<p><b>Abstraction or Generalization:</b></p>	<p>The essential characteristics of an object or class, ignoring irrelevant features, providing crisply defined conceptual boundaries. This maintains a focus upon identifying common characteristics among what may initially appear to be different objects. Abstraction enhances reusability and inheritance.</p> <p><i>Example: Objects <b>CUSTOMER</b> and <b>EMPLOYEE</b> may initially appear to be quite different objects, but applying the concept of abstraction will identify them as sub-classes of the object class <b>PERSON</b>.</i></p>	<p>The strategic, top-down analysis provides a high-level of abstraction as a starting point for modeling. Lower levels of abstraction are identified through tactical and operational modeling.</p> <p>Abstraction is really a modeling technique used to identify commonality between things in the real world. The level of abstraction appropriate to a particular model is subjective and based upon the specific context.</p>
<p><b>Inheritance:</b></p>	<p>The relationship between object classes, where one class shares the attributes and behavior of one or more other classes, based upon a hierarchical relationship. A <i>discriminator attribute</i> discriminates an instance of an object between the possible sub-classes.</p> <p><i>Example: If two sub-classes of <b>PERSON</b> were defined as <b>CUSTOMER</b> and <b>EMPLOYEE</b>, both would inherit all the structure</i></p>	<p>Secondary entities capture information that is unique to a particular type or category of the parent entity. Information that is common between a parent and all its secondary entities is retained within the parent and inherited by occurrences of the secondary entities.</p>

	<p><i>and behavior defined in <b>PERSON</b>, which in our example will include the attribute last name.</i></p>	<p>If a parent entity may simultaneously have more than one child entity, then this will be captured in a <i>role entity</i>.</p>
<p><b>Multiple inheritance:</b></p>	<p>An object class may have more than one parent.</p> <p><i>Example: The class <b>CUSTOMER</b> has been defined in our examples as a sub-class of <b>PERSON</b>, but it might also be a sub-class of <b>SALES LEAD</b>. Another sub-class of <b>SALES LEAD</b> might be <b>POTENTIAL CUSTOMER</b></i></p>	<p>A parent can have more than one group of secondary entities, and can be of more than one type within each group (captured in a role entity).</p> <p>Valid combinations of sub-types are documented in a <i>rule entity</i>. This allows the rules (and the behavior derived from them) to be changed by updating values in an occurrence of the rule entity.</p>
<p><b>Typing:</b></p>	<p>Quality control on the interchange of objects of different classes. Typing may be enforced to different degrees. Application environments may allow conversions between types.</p> <p><i>Example: Although the classes <b>CUSTOMER</b> and <b>EMPLOYEE</b> share a common super-class (<b>PERSON</b>), they are different. It might be invalid to perform a process to <b>Request current salary</b> upon an object of class <b>CUSTOMER</b>, although this process may be valid for objects</i></p>	<p>The object-oriented concept of <i>type</i> is essentially the same as the IE concept of a <i>type entity</i> or <i>category discriminator</i>.</p> <p>If an entity may simultaneously be of more than one type, or may have a history of type changes, then this will be captured in a role entity.</p>

	<p><i>of class <b>EMPLOYEE</b>.</i></p>	
<p><b>Encapsulation or Information hiding:</b></p>	<p>This hides the details of the implementation of an object. The result is that the interfaces of a class do not need to change if just the internal implementation of a class is changed. Encapsulation makes implementations more portable, reliable, stable and reusable.</p> <p><i>Example: The derivation of an employee's salary can be defined in many potential ways. A request to an object for an employee's salary can ignore the internal workings that the object class <b>EMPLOYEE</b> may use to calculate the appropriate salary. Any revisions to those mechanisms that may be made from time to time will not affect the process by which the salary is requested, or by which the result of the calculation is returned.</i></p>	<p>Data Access Processes (DAPs) represent processes that are encapsulated with the entity upon which they act. The inner workings of the DAPs are not visible at the business process level. The business process just provides a list of required information to a DAP.</p> <p>DAPs are the only processes which act directly upon entities within the data model.</p>
<p><b>Modularity:</b></p>	<p>Physical packaging of the classes and objects in the logical design into cohesive and loosely coupled modules.</p> <p><i>Example: The <b>PERSON</b> class, along with the sub-class <b>EMPLOYEE</b>, and other classes such as <b>JOB</b> and its sub-class <b>MANAGER</b>, might form the basis of a human resources management module.</i></p>	<p>Visible Advantage™ automatically identifies data <i>clusters</i> based upon data dependency, which can then be used to define the boundaries of implementation modules.</p> <p>It also automatically generates DAPs which invoke (re-use) each other to form process modules,</p>

		<p>which in turn may be re-used by other processes or modules.</p>
<p><b>Polymorphism:</b></p>	<p>Different object classes may behave differently in response to a common request or message. This enables instances of different object classes to respond appropriately to common operations upon them.</p> <p><i>Example: An operation of <b>Print</b> for an object of class <b>PICTURE</b> will behave differently than the same <b>Print</b> operation for an object of class <b>TEXT</b>, even though both these classes may have a common parent class of <b>FILE</b>.</i></p>	<p>A process may branch to one of several possible paths depending on the value of an individual attribute.</p> <p>Additionally, processes have a defined set of completion states, which can be used to trigger a range of responses in their calling processes.</p>
<p><b>State transition:</b></p>	<p>The <i>state</i> of an object instance is the status that the object instance is in. A <i>state transition</i> is the action of an object instance changing from one state to another.</p> <p>An event is some occurrence that may cause the state of a system to change [Booch, 1994].</p> <p><i>Example: When an order is shipped to a customer, the state of the attribute <b>order status</b> within the object <b>ORDER</b> will change from "In warehouse" to "Shipped."</i></p>	<p>The <i>state</i> of an entity is captured by a static entity related to that entity, or by a specific domain of values for a <i>state attribute</i>. Intersecting entities show the history of the state transitions for the entity.</p> <p><i>Rule entities</i> restrict the state value combinations to only those combinations that are pre-defined as valid. Any state transition would be triggered by a <i>business event</i> and implemented in a <i>business process</i>.</p>

<p><b>Concurrency:</b></p>	<p>It may be necessary to execute a set of processes either consecutively or simultaneously. If simultaneous processes are taking place, then multiple objects will be active concurrently.</p> <p>Synchronizing the activities of multiple active objects is difficult. Implementation environments to support truly concurrent applications are not widely available: it's difficult in C++, but it is supported in Ada (as <i>tasks</i>) and in Smalltalk (as the super-class <i>Process</i>).</p> <p><i>Example: The flight control system of an aircraft might include object classes such as <b>ENGINE</b> and <b>RUDDER</b>. To keep the aircraft flying safely, it must be possible to manipulate both these objects simultaneously and largely independently. Although concurrent, these example objects will interact on occasions (e.g. an engine failure might automatically trigger a change in the rudder position).</i></p>	<p>Visible Advantage™ process maps show which activities must be executed in a specific sequence and which can be executed in parallel (if this is supported in the chosen environment).</p> <p>If parallel processing is not supported, the parallel process steps imply that processing sequence is not important and can be optimized in the physical implementation for system performance.</p>
----------------------------	--	---

<p><b>Persistence:</b></p>	<p>A measure of the lifetime and location of the occurrence of an object, including both an object's state and its class. Encompasses both the role of variables in traditional application environments (such variables usually live just during the execution of a program or program module) and the domain of database technology (which enables information, such as the state and class of an object, to be retained between different executions of a program).</p> <p>Object-oriented DBMSs are in their infancy. Most practical implementation environments coerce existing relational DBMSs toward the needs of object-oriented application designs.</p> <p><i>Example: When an object of class <b>CUSTOMER</b> places an order (defined as an object of class <b>ORDER</b>), it will be necessary to keep a record of the each instance of <b>CUSTOMER</b> and <b>ORDER</b> until the order is filled, invoiced and paid for.</i></p>	<p>All data manipulated by processes must be a part of the data model. It is assumed that all elements of the data model will be stored and thus "persist".</p>
----------------------------	--	---

<p><b>Operations or Methods:</b></p>	<p>The direct manipulation of an object, categorized as:</p> <ul style="list-style-type: none"> <li>● <i>Constructor</i>: Create an object and/or initialize.</li> <li>● <i>Destructor</i>: Free the state of an object and/or destroy the object.</li> <li>● <i>Modifier</i>: Alter the state of the object.</li> <li>● <i>Selector</i>: Access and read the state of an object.</li> <li>● <i>Iterator</i>: Access all parts of an object in a well-defined order.</li> </ul>	<p>The specific DAPs created for an entity:</p> <ul style="list-style-type: none"> <li>● Equivalent to <i>Create</i> DAP.</li> <li>● Equivalent to <i>Delete</i> DAP.</li> <li>● Equivalent to <i>Update</i> DAP.</li> <li>● Equivalent to <i>Read</i> DAP.</li> <li>● Equivalent to a <i>Read</i> DAP that returns more than one occurrence for group or iterative processing.</li> </ul>
<p><b>Reusability &amp; Components:</b></p>	<p>A result of abstraction, encapsulation and modularity is that:</p> <ul style="list-style-type: none"> <li>● Object class definitions may be stored, catalogued and maintained in an <i>object library</i>.</li> <li>● Reusable code modules, compiled as binary files, are called <i>components</i>. Components exhibit some of the characteristics of 'objects', such as encapsulation, but do <b>not</b> necessarily exhibit other object features, such as inheritance or polymorphism.</li> </ul> <p><i>Example: Once the class CUSTOMER has been defined for</i></p>	<p>DAPs represent re-useable processes that pertain to a specific object.</p> <p>Business processes can capture reusable subsets of logic that are invoked by multiple other processes.</p> <p><i>Example: In the CUSTOMER example, the Add Customer process and CUSTOMER data structure(s) would be re-used in current and future implementations.</i></p>

*an order-entry and stock control system, it can be reused in the design of a marketing system. The code component that displays the input window for adding a new object of class CUSTOMER can also be reused.*

---

## Benefits of Object-Orientation

Object-orientation is perceived as providing many benefits which, at least at a conceptual level, are understood and seen as highly desirable by non-technical business experts. Some of these benefits are:

- Reusability.
- Modularity.
- Avoidance of duplication.
- Reliability.
- Flexibility for expansion.
- Portability.

Information Engineering provides all of the above benefits when used to generate logical data and process models, but also provides two important additional benefits:

- Documented links between logical data and process models and both the business requirements and the physical design.
- Technology independence at the logical design stage, linked to an optimized technology-dependent physical design.

---

## Implementing Object-Oriented Methodologies

The use of either an OO or IE methodology requires a major commitment of time and resources to the analysis and design stages of a project. To organizations focused upon traditional systems development techniques, this may appear as a high-risk approach. The benefits of a 'pure' OO approach will mainly accrue to the IS department within an organization (reusability, easier maintenance and extensibility) and may be difficult to sell to non-technical managers (such as the end-users of a proposed system). The benefits of an IE approach to the analysis stage of an OO project, which include formalizing the linkages between the

business requirements and the final implemented systems, are more clearly visible and desirable to non-technical managers.

Reusability is one of the strongest selling points of object-orientation. It is perceived as one of the rewards resulting from the extra time devoted to analysis and design. Object libraries can be built from objects designed during internal projects or purchased as 'off-the shelf' commercial products. But there are practical problems facing enterprises that establish and use such libraries:

- Reuse presents a cultural change in the way developers operate.
- Reuse enforces standards when new objects are created, which is likely to result in additional time and cost compared with generating a 'once-off' solution. The performance pressures placed upon those responsible for systems implementation usually focuses upon the timely and cost-efficient delivery of new or additional functionality to users, rather than offering a practical incentive to generate reusable objects.
- At the same time, reuse can be perceived by project managers as a loss of control, if most development is done by pulling pieces out of a reuse library.
- Locating the right object classes or code components in the available libraries is not easy. Organizations may need to establish a new group just to manage reusable objects and enforce OO standards. The operating cost of such a library is likely to be a vulnerable overhead to the IS department.
- There is a danger that reusable code components can become too large and generic to be effective. A balance needs to be maintained to avoid code components becoming generalized to the point at which their use becomes inefficient or ineffective.

---

## Re-engineering Legacy Systems

To reverse engineer legacy systems, with a goal of systems integration in an object-oriented environment, is difficult using just OO techniques, since the OO analysis and design methodologies require a means of defining object classes based upon 'real world' objects, rather than upon the old, possibly inappropriate data structures inherited from the legacy systems.

A combination of forward and reverse engineering using the business-driven IE methodology and Visible Advantage™ can resolve most of these problems, by combining the reverse-engineered logical data and process models of the legacy systems with an analysis of current business requirements. This enhanced analysis can form a solid basis for an object-oriented physical design.

---

## Summary

Object-orientation and business-driven IE should be regarded as symbiotic:

- IE can provide the leverage to gain the benefits of an object-oriented approach to systems design and implementation. The logical data and process models resulting from an IE approach to a business requirement are technology independent and reflect the business goals of an enterprise. They can be subsequently revised, amended and refined independent of the implementation environment of any physical systems. If an object-oriented application environment is chosen, the IE data and process models provide a firm base from which an object-oriented design can be developed.
- The concepts and terminology used to describe the features of object-oriented analysis and design are closely related to IE and the features of Visible Advantage™.
- The perceived benefits of an object-oriented approach are often quoted as being reusability, modularity, avoidance of duplication, reliability, and flexibility for expansion. IE also provides these benefits, but extends them to include technological independence and tightly defined links between logical models, business requirements and physical implementation designs.

---

## Bibliography - Sources and Further Reading

- *Object-Oriented Analysis and Design with Applications*, Grady Booch, 1994
- *Object-Oriented Modeling and Design*, Rumbaugh, Blaha, Premerlani, Eddy & Lorensen, 1991
- *Object-Oriented Software Engineering*, Jacobson, Christerson, Jonsson & Overgaard, 1992
- *Object-Oriented Information Engineering*, Stephen L. Montgomery, 1994
- *Object-Oriented Analysis*, Coad & Yourdon, 1991
- *Object-Oriented Design*, Coad & Yourdon, 1991
- *Object-Oriented Analysis and Design*, James Martin & James Odell, 1992
- *Distributed Object-Oriented Data-Systems Design*, Andleigh & Gretzinger, 1992
- *Information Engineering: Strategic Systems Development*, Clive Finkelstein, 1992
- *The C++ Programming Language*, Bjarne Stroustrup, 1991

Back to [Contents](#).

---

## Comments on Object Oriented - Information Engineering Analysis

This contains additional information about the Object Oriented - Information Engineering analysis. It consists of

- [\*Overview of the Main Object-Oriented Analysis and Design Methodologies\*](#)
- [\*Implementing Object-Oriented Methodologies\*](#)
- [\*Modeling Business Rules\*](#)
- [\*Developments in Visible Advantage version 7 to Support Object Orientation\*](#)

Back to [Contents](#).

---

### ***Overview of the main object-oriented analysis and design methodologies***

See the Bibliography on the previous page for references to supporting documentation for each methodology. Some further references are listed at the end of this Appendix.

### Booch

This is one of the original OO design methods, which is now mature and widely used. Originally, this method supported just OO design, but OO analysis has now (1994) been added in a style similar to OMT (see below). This methodology has a strong emphasis upon the definitions of classes and their instances (objects). The notation defined by Booch is regarded as very complete and the resulting class diagrams bear a strong resemblance to IE data models.

The book [Booch, 1994] is highly recommended as a readable, in-depth discussion of object-oriented concepts; it also has a good glossary of terms and an extensive bibliography.

## Rumbaugh - Object Modeling Technique (OMT)

OMT analysis is based upon three models:

- *Object model*, documenting entities, attributes and their relationships.
- *Dynamic model*, a state-transition diagram showing the reaction of objects in the system to events.
- *Functional model*, a data flow diagram (DFD), documenting the transformation of object values and the constraints upon these transformations. The use of the functional model DFD can be seen as a separation of data and process which is inconsistent with the focus of object-orientation, since processes (as single icons) are not clearly linked with the operations of a single object.

Now that Rumbaugh has joined Booch (at Rational), the boundary between these two methodologies is likely to become less clear. These methodologies are the ones favored by developers using C++ as an implementation language.

## Jacobson - Use-Case Analysis and Object-oriented Software Engineering (OOSE)

Use-case analysis is described by Jacobson [Jacobson, 1992] as:

- *... a particular form or pattern or exemplar of usage, a scenario that begins with some user of the systems initiating some transaction or sequence of interrelated events.*

This technique documents the scenarios essential to a system's operation, using storyboarding techniques which identify the objects (along with the responsibilities and interactions of the objects) needed to participate in each scenario. The initial, basic scenarios are later expanded to include exceptional cases. As well as helping identify objects in context, which helps ensure an appropriate level of abstraction, this technique can also identify scenarios which can serve as the basis of future system testing. This behavioral approach to the identification of objects is unusual in both object-oriented analysis and IE.

Analysis will refine the resultant objects into three types (interface, entity and control).

- *Interface objects*: Reflect the external, user's view of the problem domain.
- *Entity objects*: Reflect persistent behavior. These tend to be the major reusable objects.

- *Control objects*: Capture behavior specific to particular use cases. Avoiding mingling these objects with interface and entity objects helps ensure that control objects are reusable across multiple use cases.

Analysis models may be further refined and grouped into 'subsystems', which group functionally cohesive objects together. The design phase of this methodology includes a special model called the 'interaction diagram', which is especially useful for designing interactive software, and which reflects the roots of this methodology in the telecommunications industry.

## Coad/Yourdon

The main feature of this methodology is the unification of the different object-oriented models within a single conceptual structure on five layers: Subject; Class & Object; Structure; Attribute; and Service. Each layer of the conceptual model documents different types of things about the objects, which is different to the majority of methodologies which document process, data and data-flow in different diagrams with different notations.

## Martin/Odell - Object-Oriented Information Engineering (OOIE)

Uses a notation which is designed to ease the transition from Martin's IE methodology to OO analysis. OOIE adds new relationship types to the 'conventional' Martin IE methodology, which are believed to help in producing views of the model at higher levels of abstraction for presentation to business experts:

- *Composition*: Used to assist in the abstraction of a data model by encapsulating other object types. It is a "comprised of..." association. An aim of this association is to define business processes at a higher level of abstraction, where they appear more meaningful to business experts, whilst retaining the detail necessary to understand the implementation of lower-level processes (DAPs).
- *Generalization*: Equivalent to the conventional IE secondary/type associations, but extended to include typing on the basis of behavior. This is used to implement business rules within the data model.

OOIE also allows the inclusion of object *instances* on the data model, where these will add clarification. It also adds an information typing hierarchy, which can specify the precision, formatting, permitted ranges and possible operations

which are valid for a declared 'type'.

Additional notation is included to model the effect of 'events'. Events are classified as:

- *External request or time occurrence*: Similar to existing IE business events.
- *Process status event*: The status of a process (eg. process started) can itself act as a trigger to another event
- *Object state change event*: When the state of an object changes, an event is triggered.

## Stephen Montgomery

Although not a formal object-oriented methodology, Montgomery's book ("Object-Oriented Information Engineering") deserves a mention since, as is implicit in its title, it attempts to combine IE (both the Finkelstein and Martin flavors) with object-oriented analysis and design. Montgomery's approach is to use IE for business planning and analysis, but to then generate an object-oriented design from this base by mapping object classes onto IE entities, and encapsulating the processes which act upon an entity within the class definition. He generally refers to the Rumbaugh OMT and Martin/Odell methodologies as a basis for comparison with IE. The book is weak at discussing the boundaries between the IE and OO approaches.

Montgomery presents a discussion of *process normalization*. He recognizes that both IE and OO analysis can produce normalized data models, and recommends that, using the IE approach, the 'basic processes' which act upon entities be uniquely defined and encapsulated with the entity. This corresponds to the definition and use of DAPs within Visible Advantage™.

## Synthesized methodologies

The variety of object-oriented analysis methodologies available suggests that it may be possible to pick some of the best features from each, 'blending' them into a new synthesized methodology which contains the best and most appropriate features for a specific project or for a particular organization. Concurrently, as the available methodologies mature they are tending themselves to coalesce. Examples of how methodologies can be synthesized include:

- *Fusion*, developed by Hewlett-Packard, is a method that combines OMT [Rumbaugh, 1991], OOSE [Jacobson, 1992] and the Booch [Booch, 1994] methodologies.
- The recent update to the Booch methodology [Booch, 1994] has adopted some features from other methodologies, such as use-case analysis [Jacobson, 1992] and the object-state modeling notation shared with the dynamic modeling phase of OMT [Rumbaugh, 1991].
- An organization may choose the best and most relevant features from a group of different methodologies, and use the resultant blend as its in-house standard. Such a blend might include the combination of OO techniques with IE.

## ***Implementing Object-Oriented Methodologies***

The major difference between IE and OO is that the techniques of OO analysis and design have emerged to support the implementation of systems in OO programming languages, but IE can capture and refine the business requirements for systems independently of a particular implementation environment. Conceptually, the objects in an OO design can be easily related to 'real-world' things, but in practice the arcane terminology and the multiple layers of most OO analysis and design notations still leave a wide gap between non-technical business experts and the system designers and implementors.

<b>OO Penetration</b>	<b>1994</b>	<b>1992</b>
No plans for OO	32%	8%
Planning for OO	15%	16%
Exploring OO	31%	47%
Developing using OO	14%	20%
Production using OO	8%	8%

Source: Gartner Group (ADM Research Note T-899-1087)

One of the major advantages of the business-driven IE methodology is its tightly defined links between business planning statements and the data, process and design models. This bridges the gap between business experts and system designers. The IE methodology can be used to document business requirements in a form that can be subsequently implemented in an OO programming environment.

A survey performed this year by the Gartner Group identified little change since 1992 in the proportion of organizations using OO for systems development and production, and a decrease in the number of organizations exploring OO methods. It is suggested that part of the apparent lack of growth in enthusiasm for OO techniques is because organizations now entering the OO arena tend now to be more conservative and risk-averse, compared with the organizations embracing OO two years ago as a cutting-edge technology.

---

## ***Modeling Business Rules***

The different object-oriented analysis and design methodologies use a range of techniques to document and implement business rules, and as yet there is no clear standard. [Schultz, 1993] summarizes three methods by which business rules can be captured and implemented in an object-oriented analysis and design:

- Assign business rule information to existing objects within the logical model, in terms of additional behavior and state information. Although simple, this can complicate otherwise elegant objects, and decrease reusability.
- Assign the behavior and state information to the application. Reuse of the rule is made much more difficult using this technique.
- Create a new object which contains the behavior and state information necessary to implement the rule. Such an object may not have an intuitive connection with the real world of the problem domain, but is likely to be very reusable.

The final option above is similar to the IE approach to rule modeling, in which a rule entity is created which contains attributes which define the nature of rule. Control links from the rule entity identify the entities which control and controlled by the rule.

---

## ***Developments in Visible Advantage Version 7 to support Object-Orientation***

A major feature of version 7 of [Visible Advantage](#)<sup>™</sup> will be a restructuring of its design phase. The enhancements will change the present structure of data structures and data items to *design objects* and *relationships*. The types of design objects and relationships available will be modifiable in user-defined, extensible lists. The types of design objects would include data structures, menu objects, classes and windows. The new design objects will encapsulate associated *properties*; these properties can be defined as simple data items, methods, or other data objects. The potential relationship types might include the conventional data dependencies and new relationship types such as "inherited by...", "has a...", and "is a kind of...".

Diagrams and maps will be available in the design phase. Templates can be established to allow object hierarchies to be drawn, which would be equivalent to an OO map of classes and sub-classes.

These new design features will allow logical data and process models to be linked to object-oriented designs, conventional relational designs, or both. They will move Visible Advantage<sup>™</sup> towards being capable of generating detailed object-oriented or 'traditional' application design specifications. For example, if a new type of data object was defined, and linked to the logical business processes, all the elements of an application design would be available from the tool's design phase (data objects, DAPs, business processes, window specifications, menu structures and class hierarchies).

---

### **Additional References**

- *Business Rules within Object-Oriented*, Ron Schultz, **Database Newsletter**, Volume 21 # 2, March/April 1993
- *A Comparison of Object-Oriented Analysis and Design Methods*, Mike Gilpin, **Case World**, March 8-10, 1993

Back to [Contents](#).

---

### **More Information**

For more information concerning this *Visible Solution* please contact:

## North America

Visible Systems Corporation  
201 Spring Street Lexington MA 02421 USA  
Phone: +1-781-778-0200 · Fax +1-781-778-0208  
Web Site: <http://www.visible.com>  
Email: [mcesino@visible.com](mailto:mcesino@visible.com)

## Asia-Pacific

Clive Finkelstein, Managing Director  
Information Engineering Services Pty Ltd  
PO Box 246, Hillarys Perth WA 6923 Australia  
Phone: +61-8-9402-8300 Fax: +61-8-9402-8322  
Web Site: <http://www.ies.aust.com/>  
Email: [cfink@ies.aust.com](mailto:cfink@ies.aust.com)

| [Home](#) | [Courses](#) | [Certification](#) | [Projects](#) | [Papers](#) | [TEN Archive](#) | [EA Blog](#) |  
[Online Store](#) | [Contact Us](#) | [[Search](#) |

(c) Copyright 2004-2006 Information Engineering Services Pty Ltd. All Rights Reserved.