The Quarks of Object-Oriented Development

A two-construct taxonomy is used to define the essential elements of object orientation through analysis of existing literature.

By Deborah J. Armstrong

Even though object-oriented development was introduced in the late 1960s (beginning with the Simula programming language), OO development has not yet lived up to its promises. A major stumbling block to reaping the promised benefits is learning the OO approach (see [6]). One reason that learning OO is so difficult may be that we do not yet thoroughly understand the fundamental concepts that define the OO approach. When reviewing the body of work on OO development, most authors simply suggest a set of concepts that characterize OO, and move on with their research or discussion. Thus, they are either taking for granted that the concepts are known or implicitly acknowledging that a universal set of concepts does not exist.

Several authors, asserting there is no clear definition of the essence of OO, have called for the development of a consensus [9]. While a few have tried to develop such a consensus [4], to date a thorough review of the literature and identification of the fundamental concepts of the OO approach[1] has been lacking. The goal of this article is twofold: to identify and describe the fundamental concepts, or quarks,[2] of object-oriented development, and identify how these concepts fit together into a coherent scheme.

[1]This article focuses on the fundamental concepts that define the OO development approach and not a specific OO methodology such as the Rational Unified Process, or application such as OO programming.
[2]A quark is a fundamental particle that represents the smallest known unit of matter. These particles are the basic building blocks for everything in the universe.

Understanding what concepts characterize OO is of paramount importance to both practitioners in the midst of transitioning to the OO approach and researchers studying the transition to OO development. How can we hope to achieve the productivity gains promised by the OO development approach, effectively transition software developers, or conduct meaningful research toward these goals, when we have yet to identify and understand the basic phenomena?

Sample and Method

To address this question material related to OO development published from 1966–2005 was reviewed using the keyword search 'object-oriented development'. A wide variety of sources (journals, trade magazines, books, and conference proceedings), viewpoints (computer science, information systems) and emphases (programming, methodologies, modeling, and databases) were reviewed for the sampling frame.[3] The analysis consisted of reviewing each source document for the identification of specific concepts as *the* OO concepts. For example, Morris, Speier, and Hoffer [6] list abstraction, attribute, class, encapsulation, inheritance, message passing, method, object, polymorphism, and relationships as central OO concepts; whereas Rosson and Alpert [9] list abstraction, class, encapsulation, information hiding, inheritance, instance, message passing, method, object, object model, and polymorphism as the OO concepts. Of the 239 sources reviewed, 88 asserted that a specific set of concepts characterize the OO approach. Those 88 sources are used as the data set in this article. For this study, both the concepts directly listed and implied (concepts used in the explanation of other concepts) from the 88 papers were included as potential fundamental concepts.

Once a paper was selected for inclusion in the data set, the concepts from the 88 sources were recorded. There were 39 concepts mentioned as those comprising the OO approach (such as abstraction and dynamic binding). Of the 39 concepts, eight were identified by the majority of the sources: inheritance, object, class, encapsulation, method, message passing, polymorphism, and abstraction. A discussion and definition of the concepts in the order of their frequency of listing by the sources follows (see Table 1 for the numerical data). Interestingly, many of the remaining concepts with lower levels of agreement (such as instance) are either similar to, or can be subsumed under, the more agreed-on concepts (for example, object).

Concept	Count	Percentage
Inheritance	71	81%
Object	69	78%
Class	62	71%
Encapsulation	55	63%
Method	50	57%
Message Passing	49	56%
Polymorphism	47	53%
Abstraction	45	51%
Instantiation	31	35%
Attribute	29	33%
Information Hiding	28	32%
Dynamic Binding	13	15%
Relationship	12	14%
Interaction	10	12%
Class Hierarchy	9	10%
Abstract Data Type	7	8%
Object-Identity Independence	6	7%
Collaboration	5	6%
Aggregation	4	5%
Association	4	5%
Object Model	4	5%
Reuse	3	3%
Cohesion	2	2%
Coupling	2	2%
Graphical	2	2%
Persistence	2	2%
Composition	1	1%
Concurrency	1	1%
Dynamic Model	1	1%
Extensibility	1	1%
Framework	1	1%
Genericity	1	1%
Identifying Objects	1	1%
Modularization	1	1%
Naturalness	1	1%
Safe Referencing	1	1%
Typing	1	1%
Virtual Procedures	1	1%
Visibility	1	1%

Table 1. Concept count.

What are the OO Quarks?

Inheritance. The concept of inheritance was introduced in 1967 in the Simula programming language and further developed in the Smalltalk language [3]. Inheritance is frequently cited as a new concept introduced by OO [7], and it has been suggested that inheritance is the only unique contribution of the OO approach [4].

Inheritance has been defined as a mechanism by which object implementations can be organized to share descriptions [11]. Another conceptualization of inheritance is as a relation between classes that allows for the definition and implementation of one class to be based on that of other existing classes [10]. Inheritance has also been explained (in association with the class hierarchy concept) as a mechanism by which lower levels of the hierarchy contain more specific instances of the abstract concepts at the top of the hierarchy [6]. Based on these definitions, inheritance is: *a mechanism that allows the data and behavior of one class to be included in or used as the basis for another class*.

Object. The concept of an object was also introduced in the Simula programming language as a new concept. The object is both a data carrier and executes

[3]While extensive efforts were made to gain complete coverage of the OO approach, it is not claimed that the sources reviewed are exhaustive.

actions [8]. An object has been defined simply as something that has state, behavior, and identity [1], and as an identifiable item, either real or abstract, with a well-defined role in the problem domain [6]. By far the most common reference to an object is as an instance of a class [1]. Based on these definitions, an object is: *an individual, identifiable item, either real or abstract, which contains data about itself and descriptions of its manipulations of the data*.

Rosson and Alpert [9]	Henderson-Sellers [4]
1. Communicating Objects a. Object, Message Passing, Method	**1. Encapsulation, Information Hiding**
2. Abstraction a. Information Hiding, Encapsulation, Data Abstraction, Polymorphism	**2. Abstraction, Class, Object**
3. Shared Behavior a. Inheritance, Class, Instance	**3. Inheritance, Polymorphism**
4. Problem Oriented Design a. Object Modeling	

Table 2. Comparison of OO taxonomies.

Class. Also introduced in the Simula programming language, a class is a set of objects described by the same declaration [8, 9] and is the basic element of OO modeling. Some have conceptualized a class as an encapsulation of data and procedures, which can be instantiated in a number of objects [3]. Others have defined a class as a set of objects that share a common structure and common behavior [1].

A class does several things: at runtime it provides a description of how objects behave in response to messages; during development it provides an interface for the programmer to interact with the definition of objects; in a running system it is a source of new objects [8]. Based on these definitions, a class is: *a description of the organization and actions shared by one or more similar objects.*

Construct	Concept	Definition
Structure	Abstraction	Creating classes to simplify aspects of reality using distinctions inherent to the problem.
	Class	A description of the organization and actions shared by one or more similar objects.
	Encapsulation	Designing classes and objects to restrict access to the data and behavior by defining a limited set of messages that an object can receive.
	Inheritance	The data and behavior of one class is included in or used as the basis for another class.
	Object	An individual, identifiable item, either real or abstract, which contains data about itself and the descriptions of its manipulations of the data.
Behavior	Message Passing	An object sends data to another object or asks another object to invoke a method.
	Method	A way to access, set, or manipulate an object's information.
	Polymorphism	Different classes may respond to the same message and each implement it appropriately.

Table 3. OO taxonomy.

Encapsulation. Encapsulation is one of the most debated concepts in OO development. The concept of encapsulation has been said to exist prior to the introduction of OO [7] while others assert it was a new concept introduced in the Simula programming language [1]. Still others assert that encapsulation is a new term for the already existing information-hiding concept [4, 12].

There are three primary conceptualizations of encapsulation in the literature. The first conceptualization of encapsulation is as a process used to package data with the functions that act on the data [11]. The second, most common conceptualization of encapsulation, is that encapsulation hides the details of the object's implementation so that clients access the object only via its defined external interface [1, 11]. The third conceptualization includes both of the previous definitions and can be summarized as: information about an object, how that information is processed, kept strictly together, and separated from everything else [6]. Bringing these conceptualizations together, encapsulation is: *a technique for designing classes and objects that restricts access to the data and behavior by defining a limited set of messages that an object of that class can receive.*

Method. The concept of a procedure as a unit of software has existed within software development for a long time [8]. The concept of a method (also called a procedure or operation) as tied to/or inseparable from an object emerged from the Smalltalk programming language [1]. Methods typically involve accessing, setting, or manipulating the object's data [8]. In most cases, a discussion of methods is intertwined with the concept of messages. A method is the fundamental element of an object program. Typically, a method sends messages to other objects that invoke that object's methods [9]. Based on these definitions, a method is: *a way to access, set or manipulate an object's information.*

Message Passing. Some authors state that a message is merely a procedure call from one function to another [5]. Others assert that function calls are used to apply some standard process to data whereas messages invoke a specific object activity [8]. The distinction could be summed up as a procedure call takes an action and message passing makes a request.

Within the group of authors that see message passing as a distinct concept, there seem to be two fairly consistent emphases in the definition of message passing. The first emphasis is on the invocation of a method. This group sees message passing as a signal from one object to another that requests the receiving

The simplified OO taxonomy presented here may reinforce OO thinking by helping learners see how the concepts come together into the OO approach via the structure and behavior constructs.

object to carry out one of its methods [2]. The second group looks at message passing as objects making requests for actions *and* passing information to each other [6, 8]. Bringing together these definitions, message passing is: *the process by which an object sends data to another object or asks the other object to invoke a method.*

Polymorphism. The polymorphism concept was used in software development prior to the introduction of OO. The most basic conceptualization of polymorphism appears to be the ability to hide different implementations behind a common interface [12]. Some have conceptualized polymorphism as the ability of different objects to respond to the same message and invoke different responses [10]. Others have thought of polymorphism as the ability of different classes to contain different methods of the same name, which appear to behave the same way in a given context; yet different objects can respond to the same message with their own behavior [5, 6]. The literature appears to inconsistently apply the concept of polymorphism with some likening polymorphism to late binding or dynamic binding [2]. Bringing together these conceptualizations, polymorphism is defined as: *the ability of different classes to respond to the same message and each implement the method appropriately*.

Abstraction. The earliest application of structural abstraction[4] to programming languages was in the 1950s with symbolic assemblers. Data abstraction is possible in classical development, but it is enforced in the OO approach [9]. Many authors define abstraction in a generic sense as a mechanism that allows us to represent a complex reality in terms of a simplified model so that irrelevant details can be suppressed in order to enhance understanding [4, 5, 12]. Others have conceptualized abstraction as the act of removing certain distinctions between objects so that we can see commonalities [6]. Based on these definitions, abstraction is: *the act of creating classes to simplify aspects of reality using distinctions inherent to the problem*.

OO Taxonomy. Recall the dual purposes of this article: to identify the fundamental concepts, or quarks, of object-oriented development, and determine how these concepts fit together into a coherent scheme. What has been presented are the eight OO concepts that a quorum of sources cited as concepts that characterize the OO approach. These concepts have been discussed and defined within the context of the literature reviewed. Now that the fundamental

[4]Structural abstraction encompasses both generalization and aggregation.

concepts have been identified we can address how the concepts may fit together to create the OO approach.

In addition to a lack of consensus on the fundamental concepts, the software development field lacks an understanding of how OO concepts can be classified to characterize the OO approach. There seems to be an absence in the literature of an OO taxonomy with the exception of two studies. In the first study, Rosson and Alpert [9] present a taxonomy in which they list four constructs of OO: communicating objects (object, message passing, method), abstraction (information hiding, encapsulation, data abstraction, polymorphism), shared behavior (inheritance, class, instance), and problem-oriented design (object modeling). In the second study, Henderson-Sellers [4] presents a taxonomy using the idea of the object-oriented triangle. Each construct is represented by a corner of the triangle (which unfortunately he does not name). The first corner of the triangle includes the concepts of encapsulation and information hiding. The second corner includes the concepts of abstraction, class, and object. The third corner includes the concepts of inheritance and polymorphism.

Looking at the two taxonomies (see Table 2) there seems to be a large overlap in the concepts included, but little overlap between the two classifications of the concepts. While the concepts identified in these taxonomies substantiate the OO quarks identified in this work, they also identify the need for a simplified conceptualization of the OO approach.

In an attempt to reconcile the taxonomies presented in Table 2 with the information found in this study, a new taxonomy is proposed composed of the eight fundamental OO concepts placed into two constructs labeled *structure* and *behavior*. This taxonomy uses an OO perspective to classify the quarks of OO development. The first construct, *structure*, includes the abstraction, class, encapsulation, inheritance, and object concepts. These concepts are focused on the relationship between classes and objects and the mechanisms that support the class/object structure. In essence a *class* is an *abstraction* of an *object*. The class/object *encapsulates* data and behavior and *inheritance* allows the encapsulated data and behavior of one class to be based on an existing class. The second construct, *behavior*, includes the message passing, method, and polymorphism concepts. This construct is loosely based on the communicating objects construct found in the Rosson and Alpert [9] taxonomy and is focused on object actions (behavior) within the system. *Message passing* is the process in which an object sends information to another object or asks the other object to invoke a *method*. *Polymorphism* enacts behavior by allowing different objects to respond to the same message and implement the appropriate *method* for that object. Table 3 presents the taxonomy discussed.

So how do these two constructs fit together? Within the OO approach, behavior (the actions and reactions of the system) is a way of manipulating structure (objects and/or processes within a system and their relations). However, behavior must also support the actions of the system. So this taxonomy emphasizes the interconnected nature of OO development. Compared to the two previous taxonomies, this taxonomy is simpler (only two constructs) and uses an OO perspective (structure and behavior) to frame the concepts. The two constructs are consistent with the OO approach, and simplify the process of organizing the OO quarks into a coherent conceptual scheme. Therefore, rather than complicating the learning process with a complex conceptual scheme, this taxonomy will both simplify the learning process and reinforce the OO approach.

Discussion and Implications

This study indicates the first step toward mastering the OO approach by presenting the eight fundamental OO concepts (quarks) consistently identified by a heterogeneous sample of sources from the OO literature. In addition to identifying and defining the fundamental OO concepts, this study has presented a taxonomy for the eight OO quarks by organizing them into two constructs labeled structure and behavior. This is the first taxonomy that has used an OO perspective to classify the quarks of OO development.

Why has there been no consensus around the fundamental concepts that define the OO approach? Perhaps differences in perspectives (conceptual versus implementation), emphasis in life cycle (analysis, design, implementation), and orientation (computer science versus information systems) provide insurmountable obstacles. If there has been no consensus to date, why is a definitive set of OO quarks and organizing taxonomy so important? Select any five OO books off your shelf and you'll get five different conceptualizations of the fundamental OO concepts. This is very confusing, especially for a developer learning the OO approach. Recall that one stumbling block to obtaining the benefits of OO is learning the OO approach [6]. One way to decrease the confusion and ease the learning process is to establish a standard set of OO quarks. Developing a consensus around the fundamental concepts that define the OO approach will aid developer learning by providing a common language and knowledge base.

By putting these findings into practice, an imme-

diate impact can be felt in both academia and the software development industry. Both universities and organizations can use the concepts demonstrated in this study to design learning materials and techniques aimed at clarifying these fundamental OO concepts for the learner. The simplified OO taxonomy presented here may reinforce OO thinking by helping learners see how the concepts come together into the OO approach via the structure and behavior constructs. This may aid training programs and the retraining of developers moving to the object-oriented approach. In addition, an established set of fundamental OO concepts within a taxonomy may enhance the maturity of the OO development discipline through standardization, and increase the portability of developers across organizations and environments. C

References

1. Booch, G. *Object Oriented Analysis and Design with Applications*. Benjamin/Cummings, Redwood City, CA, 1994.
2. Byard, C. Object-oriented technology a must for complex systems. *Computer Technology Review 10*, 14 (1990), 15–20.
3. Dershem, H.L. and Jipping, M.J. *Programming Languages: Structures and Models*. PWS Publishing Company, Boston, MA, 1995.
4. Henderson-Sellers, B. *A Book of Object-Oriented Knowledge*. Prentice-Hall, Englewood Cliffs, NJ, 1992.
5. Ledgard, H. *The Little Book of Object-Oriented Programming*. Prentice-Hall, Upper Saddle River, NJ, 1996.
6. Morris, M.G., Speier, C., and Hoffer, J.A. An examination of procedural and object-oriented systems analysis methods: Does prior experience help or hinder performance?" *Decision Sciences 30*, 1 (Winter 1999), 107–136.
7. Page-Jones, M. and Weiss, S. Synthesis: An object-oriented analysis and design method. *American Programmer 2*, 7–8 (1989), 64–67.
8. Robson, D. Object-oriented software systems. *Byte 6*, 8 (Aug. 1981), 74–86.
9. Rosson, M. and Alpert, S.R. The cognitive consequences of object-oriented design. *Human Computer Interaction 5*, 4 (1990), 345–379.
10. Stefik, M. and Bobrow, D.G. Object-oriented programming: Themes and variations. *The AI Magazine 6*, 4 (Winter 1986), 40–62.
11. Wirfs-Brock, R.J. and Johnson, R.E. Surveying current research in object-oriented design. *Commun. ACM 33*, 9 (Sept. 1990), 104–124.
12. Yourdon, E., Whitehead, K., Thomman, J., Oppel, K. and Nevermann, P. *Mainstream Objects: An Analysis and Design Approach for Business*. Yourdon Press, Upper Saddle River, NJ, 1995.

Deborah J. Armstrong (darmstrong@walton.uark.edu) is an assistant professor in the Information Systems Department in the Sam M. Walton College of Business at the University of Arkansas.