Introduction to Object-oriented design

Communications of the ACM 33.n9 (Sept 1990): pp38(2). (929 words)
Author(s): John D. McGregor and Tim Korson
Document Type: Magazine/Journal

Object-oriented, a buzzword of the late 1980s, has evolved into an accepted technology that has recognized benefits for the software development process. In its progression from a purely procedural approach, software development reached a data-driven--object-based--approach, and has grown beyond that to the object-oriented approach.
The impact of the object-oriented approach is not limited to the design portion of the software development life cycle--its effects are evident at every phase. One of the strengths of the object model is that it provides a unifying element that is common to every phase of the life cycle. This uniformity provides a smooth transition from one phase to the next.

The article by Henderson-Sellers and Edwards presents a revision of the traditional life cycle based on the object-oriented approach. It discusses the unique view of the design process and describes how it works: the process takes a specified problem and decomposes it. The resulting product forms the framework for a computer-based solution to the problem. Object-oriented techniques begin this decomposition process in the analysis phase and carry it on into the design phase. A modeling paradigm is used for the decomposition process: The top layer of an object-oriented system is a model of the real-life situation for which the software system is being created. The underlying layers provide the implementation of this model.

The "pieces" produced by object-oriented techniques are as unique as the design perspective. Its obvious similarities to and subtle differences from abstract data type (ADT) technology have led to much discussion of objects and classes in terms of ADTs. The unique coupling of data and behavior in the object-oriented components provides much more than a syntactic distinction from the usual ADT. Added to the modeling approach, it produces a recognizably different approach to systems development.

The term object-oriented is defined differently by different people. Many of professionals agree with the basics of Wegner's definition [1] that object-oriented includes three concepts: objects, classes, and class inheritance. Some would add a variety of other requirements including such concepts as polymorphism, dynamic binding, encapsulation, etc. The article by Korson and McGregor provides an overview of these concepts, leaving room for the reader to decide which to include and which to exclude; it also provides an overview of the basic concepts of object-oriented design.

The production of software in an increasingly competitive environment is making reuse a priority of software professionals. The popularity of the object-oriented technique is due, in part, to its support for reuse. Two important factors influence reuse: First, it is necessary to have a set of high-quality components that are worth reusing. Second, the components must be well-defined, easy to integrate, and efficient. Meyer's article presents some experiences in developing the classes for the Eiffel library; it also discusses characteristics of the library and the classes in the library. For components to be reused, the designer must have the means to locate a component which models an entity in the current problem. Not only must the component be located, but often necessary supporting pieces must be found as well. The article by Gibbs et al. provides information on the management of classes and the software components of the object-oriented paradigm.

Tom DeMarco, ir a recent interview, [2] declared parallel computing to be the emerging new paradigm. According to DeMarco object-oriented techniques will be an integral partner in this emergence. DeMarco observed that designing with objects preserves the natural parallelism in a problem. Agha addresses models for parallel objects, presenting an overview of the problem and focusing on the actor model as a possible solution. He presents examples of design issues when using the actor model. He also considers a basic reflective design architecture.

The importance of well-defined and well-managed abstractions in the software development process is discussed in the articles by Meyer and by Gibbs et al as they explore what has come to be called the software base, the set of software components from which future products will be built.

The unique components developed by object-oriented methods are characterized by an interface that is separate from the implementation of that behavior. The designer is free to concentrate on modeling the problem at hand either by developing specific classes or by locating and reusing existing classes that model some subset of the needed behavior. Meyer focuses on what he quotes McIlroy as terming a "software components subindustry," [3] presenting a case study of the development of the Eiffel libraries.

The final article in this special issue, by Henderson-Sellers and Edwards discusses modifications to the traditional life cycle supported by the object-oriented approach. The modified life cycle recognizes the iterative nature of the development process and incorporates that characteristic into its model.

Wirfs-Brock and Johnson present a sampling of current research into several aspects of object-oriented design. Their survey includes efforts to improve reusability through design technique and paradigm-specific tools. The works are representative of the broad spectrum of research activity currently under way.

We would like to thank the authors in this special issue for their hours of work both in developing their own articles and for the time spent evaluating and commenting upon the other articles.

(1) Wegner, P. Dimensions of object-based language design. In Proceeding of of Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA '87). (Orlando, Fla., Oct. 4-8) ACM, New York, pp. 168-182.

(2) Interview for the NEWSInterview column, IEEE Software, March 1988, 92-93.

(3) McIlroy, M.D.Mass-produced software components. In Software Engineering Concepts and Techniques (1968 NATO Conference on Software Engineering), J.M. Buxton, P. Naur, and B. Randell, Eds. Van Nostrand Reinhold, 1976, 88-98.