8-15-1997

# OOSD: A Framework ofAdvantages and Disadvantages

Richard Johnson
*University of Arkansas*, richardj@comp.uark.edu

# OOSD: A Framework of Advantages and Disadvantages

Richard Johnson
Dept. of CISQA, College of Business Administration, University of Arkansas,
Fayetteville, AR, 72701, 501 575-2695
richardj@comp.uark.edu

## Introduction

Information technology (IT) is widely recognized as critical for creating and maintaining competitive advantage. A major aspect of IT involves the development of information systems (Hodgkinson 1992). However, systems development has long been plagued by the so-called "software crisis," i.e., the inability of developers to keep pace with the increasing demands for both the quantity and quality of complex systems (Pressman 1992).

While systems development advanced from the traditional methods of the 1960s and 1970s to the structured and prototyping methods of the 1980s, the software crisis persists. The 1990s have witnessed the emergence of a relative newcomer to the software engineering scene, object-oriented (OO) systems development (OOSD). While perhaps not the proverbial "silver bullet" (Brooks 1987), OOSD is viewed by many as the best available opportunity to successfully address the current crisis (Booch 1994, Coleman et al. 1994, Jacobson et al. 1993, Rumbaugh et al. 1991).

The IS literature abounds with testimony extolling the virtues of object orientation. However, the somewhat complex relationships among both the advantages and disadvantages of OOSD are seldom explained in a concise fashion. The purpose of this paper is to examine both the pros and cons of OOSD found in the extensive OO literature and provide a framework for understanding the relationships among its advantages and disadvantages. The primary contribution of this study is an improved understanding of the evaluation of OOSD. The results should serve as a basis for both practitioners and theorists to improve OO implementation efforts and OO research, respectively.

## OOSD

OOSD consists of analysis, design, and implementation. Of course, the terms OO analysis (OOA), OO design (OOD), and OO programming (OOP) are most often used in the OO context. It is within these activities of OOA, OOD, and OOP that one finds the source of the advantages and disadvantages of OOSD relative to CSD methods. The advantages of OOSD are expected to ultimately translate into improved developer productivity and increased system quality (Weinberg et al. 1990).

## Advantages of OOSD

The evaluation of OOSD by systems developers can eventually lead to varying degrees of its acceptance or rejection as a viable methodology. Therefore, it is very important to not only identify advantages and disadvantages, but to understand their inner workings and relationships. The most prolific sources of these evaluations are based on expert opinion which is readily available in a seemingly unlimited number of articles and texts. A brief review of the advantages and disadvantages of OOSD as taken from these sources is now provided.

Easier Modeling Process. The process of creating the models of OOA and OOD is easier because many (if not most or all) of the objects in the models are abstractions of objects found in the real world (Khoshafian and Abnous 1995). Models developed using CSD are based on procedural decompositions that are more

difficult to understand and maintain (Eckert and Golder 1994). An easier modeling process should translate into improved productivity.

More Effective Modeling Products. Not only are the models of OOA and OOD easier to produce, the end products (the models themselves) are easier to understand and more accurate with respect to capturing user requirements (Coleman et al. 1994). More effective, accurate and understandable models should enhance quality.

Easier Transition from Analysis to Design to Implementation. Since the models of OOA and OOD, as well as the OO programming languages themselves, are all based on the same OO paradigm (object, class, and inheritance), developers can make the transition from analysis to design to implementation (programming) much more easily (Eliens 1995). An easier transition between phases of the development life cycle should improve productivity.

Improved Communication. Since the modeling products are more understandable by both users and development team members, necessary communication within and between these groups should be improved (Garceau et al. 1993). Improved communication should result in fewer misunderstandings and therefore increase quality.

Improved Modularity. The basic module of OOSD is the object. The object is simply an instance of the more generally defined class. Classes can be combined to form frameworks (groups of specific related classes) or design patterns (groups of more loosely defined classes). This type of more natural modularity is viewed as a major improvement over the procedural modules of CSD (Coleman et al. 1994). Since program code and system models exist in more well-defined and natural modules, verification of functionality becomes easier and the operation of systems becomes more reliable. Thus, both productivity and quality are enhanced.

Code Reuse. A consequence of the improved modularity (the "objectness") of OOSD is the enhanced capability to reuse objects (Tkach and Puttick 1994). The combination of data, behavior, and a well-defined user interface within the object makes it a prime candidate for a component to be used in the assembly of software products. Since objects contain program code, the reuse of objects results in the reuse of code. Of course, a well-organized means a cataloging objects is required to facilitate the reuse of objects. Code reuse has consequences of increased productivity (less rewriting of code) and increased quality (by using previously tested objects).

Model Reuse. Not only can code be reused, but the products of the analysis and design processes can be reused. Again, this benefit is a consequence of the more natural modularity of OOSD. Objects form classes and classes can be related in logical ways (frameworks and patterns) that can find application in a variety of similar situations (Wirfs-Brock and Johnson 1990, Gamma et al. 1995). Model reuse, as with code reuse, enhances both productivity and quality.

Improved Maintainability. A direct effect of code and model reuse is improved maintenance of systems (Chen and Chang 1994). Maintenance can be divided into three types: corrective, adaptive, and perfective (Pressman 1992). Corrective maintenance involves fixing the errors that surface during the development process. Because of virtually all of the advantages listed above, less corrective maintenance will be required both during and after the initial development process. Inheritance is also a major reason why maintenance is improved (Korson and McGregor 1990). Via inheritance, changes made at a higher level in the class hierarchy are automatically carried through to the lower levels of the hierarchy. This saves time and reduces the occurrence of errors.

As time passes, user requirements for a system change and adaptive maintenance will inevitably be required. Because of the more understandable modeling products and the improved modularity of OO systems, extending the system with enhancements of existing objects, frameworks, or patterns, or creating new entities, should be easier. The same arguments could be made in the case of perfective maintenance

where functionality is improved in the absence of a pressing need to meet new requirements. The OO system is simply easier to understand and, hence, to change (Coleman et al. 1994). Improved maintainability thus results in enhanced productivity.

## Disadvantages of OOSD

OOSD is not without its problems or its critics. The two most significant problems wit OOSD are (1) the apparent difficulty of learning both the OO paradigm and OO programming (Pancake 1995), and (2) the relative immaturity of OO methods and tools (Iivari 1995). Many authors estimate that adequately learning OO, even given that one is already experienced in CSD, requires anywhere from 3 to 18 months (Fayad et al. 1996). Learning a hybrid (as opposed to a pure) OO programming language such as C++ can also pose some difficulty for developers (Pancake 1995).

The relative immaturity of OOSD is evident in many respects. First, there are dozens of different OOA/D methods from which to choose, although there are movements to standardize on the Unified Modeling Language (Monarchi et al. 1995). Second, CASE tools to support some OOA/D methods may be nonexistent or limited (Pancake 1995). Third, OO DBMS are widely varied in their capabilities (Loomis 1995). This lack of maturity usually results in slower adoption of OOT by organizations (Hardgrave 1996).

## Discussion

The framework presented in Figure 1 is designed to explain some of the relationships among the various advantages and disadvantages discussed above While the framework may appear somewhat foreboding, it is intended to show the many complex interrelationships present among the characteristics of OOSD. Such a framework can be utilized to better understand those relationships and guide practitioners and researchers in their efforts to improve the acceptance of OOSD. (Note that all the linkages in the framework are positive unless denoted with a "-".)

There are obviously many relationships presented in Figure 1 that could generate even more lengthy discussion, and, preferably, more investigation of an empirical nature. Note that the OO paradigm (on the far left of the framework) eventually leads to some degree of acceptance (success) of OOSD (on the far right of the framework). This acceptance could be characteristic of individual developers as well as IS organizations on the whole. A critical question, however, is whether, over time, the positive influences of OOSD will outweigh its negative effects so that the net result would be both individual and organizational acceptance of OOSD. The time required for the effects of these advantages and disadvantages to be played out in the real world of software development may be crucial for the acceptance of OOSD.

The framework presented in Figure 1 is not intended to be exhaustive or iron-clad. There may be other more subtle relationships among these concepts that are not shown in the framework. There may also be differences of opinion on the precise nature of the linkages. Redundant concepts should be eliminated and new, independent concepts should be added. However, this type of approach should serve as a beginning for the analysis of the relationships among the salient concepts. Attempts should be made to more precisely define and operationalize such concepts presented in this type of framework. The existence and strength of the relationships among the concepts could then be tested empirically in order to yield refinements to the framework. A well-tested framework should serve practitioners in their OO implementation efforts and could eventually lead to the widespread acceptance of OOSD in all types of organizations.

*Complete references are available on request.*

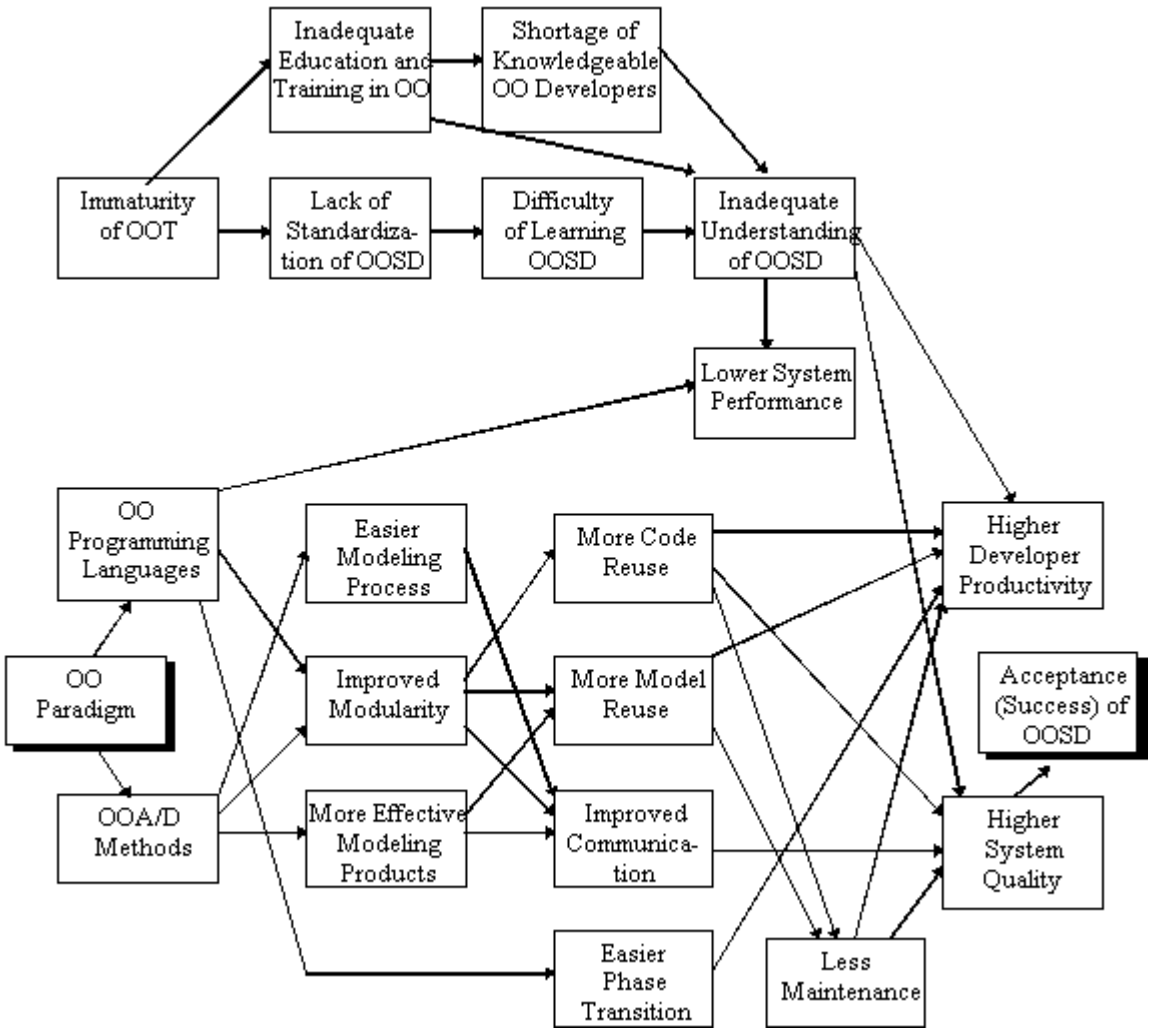


Figure 1. Framework of OOSD Advantages and Disadvantages