FOOM: Functional- and Object-Oriented Analysis & Design of Information Systems: An Integrated Methodology

PERETZ SHOVAL, Ben-Gurion University of the Negev, Israel
JUDITH KABELI, Ben-Gurion University of the Negev, Israel

We propose FOOM (Functional and Object-Oriented Methodology), an integrated methodology for information systems analysis and design, which combines two essential software-engineering paradigms: the functional/data approach (or process-oriented) and the object-oriented (OO) approach. System analysis phase, where user requirements are set and defined, includes functional analysis and data modeling activities. This phase produces a hierarchy of data flow diagrams (DFD) and an initial OO schema, which can be created directly or from an entity-relationship diagram (ERD). The design phase is performed according to the OO approach, producing a complete OO schema and a behavior schema. The seamless transition from analysis to design is enabled thanks to ADISSA methodology, which facilitates the design of the menus, forms and reports classes, and the system behavior schema, from the DFDs and the application transactions. The paper introduces the motivation for the combined approach, outlines the methodology, and presents an example that demonstrates it.

INTRODUCTION

Many paradigms for system analysis and design have been proposed over the years. Early approaches have advocated the functional approach. Common methodologies that support this approach are Structured System Analysis (SSA) and Structured System Design (SSD) (DeMarco, 1978; Yourdon & Constantine, 1979). SSA is based on the use of data flow diagrams (DFD), which define the functions to be performed by the system, the data stores within the system, the external entities, and the data flows that connect the above components. SSA and similar methodologies of that age emphasize the functional aspects of system analysis, neglecting somehow the structural aspect, namely that of modeling the data structure. Concurrently, Entity-Relationship (ER) became a popular model for designing the data structure, namely the database schema. SSD is based on the use of Structure Charts (SC), which describe the division of the system to program modules as well as the hierarchy of the different modules and their interfaces. Certain techniques have been proposed to create SCs from DFDs (see Yourdon & Constantine, 1979). The main difficulty of the above approach, that is, functional analysis followed by structured design, lies in the transition from DFDs to SCs. The translation is problematic because a DFD is a network structure, whereas a SC is a hierarchical structure. In spite of various guidelines and rules for conversion from one structure to the other, the problem has not been resolved by those methodologies (Coad & Yourdon, 1990).

Shoval (1988) developed the ADISSA methodology to solve the problem. It uses hierarchical DFDs during the analysis stage (similar to other functional analysis methodologies), but the design centers on transaction design. A transaction is a process that supports a user who performs a business function, and is triggered as a result of an event. The transactions of the system are derived from the DFDs: a transaction consists of elementary functions that are chained through data flows and of data-stores and external-entities that are connected to those functions. The process logic of each transaction is defined by means of structured programming techniques. In addition to transactions design, ADISSA methodology includes the stages of interface design, inputs/outputs design, and database design.

The interface design stage results in a menu-tree, which enables users to find and fire desired transactions. The menu-tree is derived from the hierarchy of DFDs in a semi-automatic fashion. The design of inputs and outputs of each transaction is based on data flows from external entities to functions and from functions to external entities. The data-
base schema is designed as based on the DFD data stores, resulting in a normalized relational schema. The data flows from functions to data stores and from data stores to functions serve as a basis for defining access-steps from each transaction to appropriate database relations. The products of the design stages can be easily prototyped or implemented using various programming environments.

The development of object-oriented (OO) programming languages gave rise to a new approach, which maintains that in order to develop information systems in such languages, it is recommended to perform object oriented analysis and design. Many OO methodologies were developed (e.g. Booch, 1991; Coad & Yourdon, 1990; Coad & Yourdon, 1991; Jacobson, 1992; Martin & Odell, 1992; Rumbaugh, Blaha, Premerlani, Eddy & Lorensen, 1991; Shlaer & Mellor, 1988; Shlaer & Mellor, 1992; Wirfs-Brock, 1990), and the area is still evolving. In the OO approach the world is composed of objects with attributes (defining its state) and behavior (“methods”) constituting the only way by which the data included in the object can be accessed. When using the OO approach, a model of the system is usually created at the analysis stage in terms of objects - an object schema (or OO schema). An OO schema consists of different object classes with various structural relationships between them (e.g. inheritance), and each object class having its attributes and behavior (functionality). During the design stage, implementation considerations are added and the result is a model, which is supposed to enable OO programming.

Many advocates of the OO approach claim (with no substantial proof) that it is more natural to begin the analysis of a system by defining its object structure rather than by defining its functions. They support this with the claim that the real world is not composed of functions but rather of objects. They claim that the OO approach simplifies the transitions in system development stages, enhances communication between users and developers, encourages code reuse and enables the creation of robust information systems that can be easily upgraded and maintained.

While there are no doubts about the advantages of the OO approach in programming, as it supports information hiding (encapsulation), software reuse and maintenance, there are doubts with respect to the effectiveness of the approach for analyzing business-oriented information systems (as opposed to real-time systems). OO methodologies tend to neglect the functionality aspect of system analysis, and do not show clearly and systematically how to integrate the application functions (transactions) with the object schema. On the other side, users usually express their information needs in terms of functions that the system ought to perform. Another difficulty with many OO methodologies is that they involve many types of diagrams and notations with no specific guidelines which to use and in what order. The multiplicity of diagram types in the OO approach has been a major motivation for developing the Unified Modeling Language (UML) (see, for example, Clee & Tepfenhart, 1991; Larman, 1998; UML-CD, 1998; UML-Rose, 1998). UML was developed in order to produce a standard (“unified”) modeling language. It consists of several types of diagrams with well-defined semantics and syntax, which enable presenting a system from different point of views. But UML does not offer a development methodology; it does not guide a developer in how to do or what development process to follow.

Nowadays, the convention is a separation between the system development approaches: either one follows a “pure” object-oriented, or a “pure” process-oriented approach. Our approach is to integrate the two paradigms. In our view, processes are as fundamental as objects, and the two complement each other. We propose a system analysis and design methodology that combines the two approaches and its result is an object model consisting of object and behavior schemas.

RESEARCH MOTIVATION AND GOALS

Information systems development is a multi-phase process, of which the analysis and design are of primary importance. Therefore it is vital to examine which methods are appropriate to perform each of these phases. If we take a look on the existing methods for performing each of these developing phases, we can conclude:

For the analysis phase there exist two major approaches: the functional/data approach and the OO approach, each one having advantages and disadvantages. On the one hand, those who advocate the OO approach claim that using data abstraction at the analysis phase, producing a model of reality by means of classes, is preferable to producing a functional model, because the real world consists of objects. However, as said, there is no research (that we know of) that proves that the OO approach is more effective compared to the functional/data approach in the development of business oriented information systems. OO methodologies tend to neglect the functionality aspect of system analysis, and do not show clearly how to integrate the systems functions, or transactions, with the object schema. Sometimes the impression is that all the functionality of the system is expressed by means of methods that are encapsulated within objects, not paying enough attention to functional requirements that cannot be met by simple methods. On the other hand, over many years we have learned and practiced to perform functional analysis with DFDs and there were no problems with them as a mean to express the functionality of the system; the problem was how to continue from them to the following phases of development.

In our opinion, since process and object are both fundamental building blocks of reality, the analysis phase must cover both the functional and the data aspects. The functional approach, using DFDs, is most suitable to describe the functionality of the system. ERD or OO schemas are most suitable to model the data structure. Since OO approach (as will be discussed later) is most appropriate to perform the design