8-16-1996

Functional Analysis and Object-Oriented Design- A Hybrid Methodology

Peretz Shoval
Information Systems Program, Department of Industrial Egineering & Management, Ben-Gurion University of the Negev, shoval@bgumail.bgu.ac.il

Fany Sadan
Information Systems Program, Department of Industrial Engineering & Management, Ben-Gurion University of the Negev

Functional Analysis and Object-Oriented Design - A Hybrid Methodology

Peretz Shovaland Fany Sadan
Information Systems Program, Dept. of Industrial Engineering & Management
Ben-Gurion University of the Negev, Beer-Sheva 84105 Israel
e-mail: shoval@bgumail.bgu.ac.il

Abstract

We propose a methodology for information systems analysis and design which is a hybrid of two main streams in software engineering, the functional (or process-oriented) approach and the object-oriented (OO) approach. System analysis, which aims at eliciting and defining user requirements, continues to be carried out in the functional approach, utilizing data flow diagrams (DFD). System design, which aims at designing the software, is carried out in the OO approach, yielding an object model that consists of an object schema and a behavior schema (i.e., methods and messages). The transition from the functional model (in the analysis stage) to the OO model (in the design stage) is enabled by the use of ADISSA methodology, which facilitates design of the object schema from DFD data stores, and design of the behavior schema from transactions, which by themselves are derived from the DFDs.

1. Functional Approaches

The conventional approach to information systems analysis and design is the functional approach. Common methodologies which support this approach are Structured System Analysis (SSA) and Structured System Design (SSD) [6, 7]. 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 which connect the above components. A data dictionary supplements the DFDs, defining the data elements which flow on the various data flows. When using SSD, DFDs are translated into a modular description of the system. The main tools which support this stage are Structure Charts (SC), which describe the division of the system to modules as well as the hierarchy of the different modules and their interfaces.

The main difficulty of the conventional 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 [3].

The ADISSA methodology [12] has been developed to solve the problem. It uses hierarchical DFDs during the analysis stage (similar to SSA methodology), but the design centers on transactions design. A transaction is a system process which 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 which are chained through data flows, and of data stores and external entities which are connected to those functions. The process logic of each transaction is defined by means of structured programming techniques.

In addition to transactions design, the 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 database 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.

2. The Object-Oriented Approach

The development of object-oriented 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. 1, 2, 3, 4, 5, 8, 9, 10, 11]. In the OO approach the world is composed of objects with attributes (defining its state) and behavior ("methods" or "services") 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. 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 objects structure rather than by defining its functions. They support this with 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 which 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). Some of the difficulties in the application of the OO approach are that there is no good way to identify the object classes and their methods (only rather general recommendations), and that it is difficult to describe processes which involve many objects.

3. The Hybrid Approach

We propose a hybrid methodology for the development of business-oriented information systems, which combines the functional approach and the OO approach. The functional approach is used at the analysis stage (where its advantage lies) to define users needs, and this is done by creation of a DFD model; the OO approach is used at the design stage (where its advantage lies) to define the structure and behavior of the system, and this is done by creation of an OO model.

We believe that since users express their information needs in a functional manner - not by means of the object structure and behavior - a more appropriate (natural) method to carry out the task is by functional analysis (functional abstraction). It is therefore worthwhile, on the one hand, that analysis be made through DFDs which express the functional requirements of the system. On the other hand, the design should be made through the OO approach to facilitate the transition of the design to OO programming, which proved to be a better approach to implement software.

The hybrid of the two approaches is made possible because it applies principles taken from the ADISSA methodology, especially transactions design. This enables the transition from the functional analysis DFDs to the OO model that consists of an object schema and behavior (that is: methods and messages).

The essential steps of the proposed methodology are as follows:

- Functional analysis:

System analysis is performed based on information needs of the users (functional requirements) and are expressed through hierarchical DFDs and a data dictionary.

- Transactions design:

The transactions of the system are identified and described in compliance with ADISSA methodology. For each transaction the following are identified: the event which causes its operation (trigger), the elementary functions, the order in which those functions are performed, the outputs and the inputs of the transaction, and the data being retrieved ("read") and updating ("write") the data stores. Various components of each transaction will later on be "distributed" among appropriate objects.

- Objects schema design:

The data stores which were identified during functional analysis and appear in the DFDs are translated into an object schema. Every data store is translated into one or more object classes. The data elements on the data flows which flow in and emanate from a data store (as defined in the data dictionary) enable to define the objects' attributes as well as their structural relationships.

- System behavior design:

The design of the system behavior is based on the transactions, which are "decomposed" into methods. Recall that a transaction performs various types of operations. Those operations which relate to objects (i.e., which retrieve or update data) are defined as methods. Each such operation is removed from the body of the transaction and attached to an appropriate object class. Hence, the methods that belong to each object class are derived from the transactions that access it. The "remains" of each transaction include data-free operations, the process logic of the transaction, and "calls" - actually messages which are sent to objects and fire the respective methods. The transactions can be triggered by users via the user-interface that is designed according to the ADISSA method (i.e., the menu-tree is derived from the hierarchy of DFDs). Finally, the transactions, the object classes and the user-interface can be implemented with an OO programming environment.

4. Summary

The advantages of the hybrid methodology are:

- Analysis and specification of user requirements is performed in functional terms, which is the natural way by which users express their information needs.
- Design of the object schema follows the analysis, after having identified the data stores and the data elements that flow in and out. This facilitates a systematic design of the object-class structure.
- Design of the behavior schema is based on the transactions of the system. This facilitates systematic definition of methods and messages in order to carry out the transactions.
- The methodology includes systematic design of the user interface - a menu tree - that enables access and trigggering of the system transactions.

- All products of the design stage - the object-class schema, the transactions and the user-interface - can easily be implemented with an OO programming environment.

Bibliography

[1] J. Rumbaugh et al., *Object-Oriented Modeling and Design*, Prentice Hall, Englewood Cliffs, N.J., 1991.

[2] J. Martin and J. Odell, *Object-Oriented Analysis & Design*, Prentice Hall, Englewood Cliffs, N.J., 1992.

[3] P. Coad and E. Yourdon, *Object-Oriented Analysis*, Prentice Hall, Englewood Cliffs, N.J., 1990.

[4] P. Coad and E. Yourdon, *Object-Oriented Design*, Prentice Hall, Englewood Cliffs, N.J., 1991.

[5] G. Booch, *Object-Oriented Design With Applications*, Benjamin/Cummings, 1991.

[6] E. Yourdon and L. Constantine, *Structured Design: Fundamentals of a Discipline of Computer Programming and Design*, Prentice Hall, N.Y., 1979.

[7] T. DeMarco, *Structured Analysis and System Specification*, Yourdon Press, N.Y., 1978.

[8] S. Shlaer and S. Mellor, *Object-Oriented Analysis: Modeling the World in Data*, Yourdon Press, Englewood Cliffs, N.J., 1988.

[9] S. Shlaer and S. Mellor, *Object Life Cycles: Modeling the World in States*, Yourdon Press, Englewood Cliffs, N.J., 1992.

[10] R.J. Wirfs-Brock et al., *Designing Object-Oriented Software*, Prentice Hall, Englewood Cliffs, N.J., 1990.

[11] /I. Jacobson, *Object-Oriented Software Engineering: A Use Case Driven Approach*, ACM Press, 1992.

[12] P. Shoval, *"ADISSA: Architectural Design of Information Systems based on Structured Analysis"*, Information System, Vol. 13 (2), 1988, pp. 193-210.