IDEA GROUP PUBLISHING
701 E. Chocolate Avenue, Hershey PA 17033-1117, USA ITB7406
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com

## Chapter IV

# FOOM–Functional and Object-Oriented Methodology for Analysis and Design of Information Systems[1]

Peretz Shoval and Judith Kabeli
Ben-Gurion University, Israel



FOOM is an integrated methodology for analysis and design of information systems, which combines the two essential software-engineering paradigms: the functional- (or process-) oriented approach and the object-oriented (OO) approach. In FOOM, system analysis includes both functional and data modeling activities, thereby producing both a functional model and a data model. These activities can be performed either by starting with functional analysis and continuing with data modeling, or vice versa. FOOM products of the analysis phase include: a) a hierarchy of OO-DFDs (object-oriented data flow diagrams), and b) an initial object schema, which can be created directly from the user requirements specification or from an entity-relationship diagram (ERD) that is mapped to that object schema. System design is performed according to the OO approach. The products of the design phase include: a) a complete object schema, consisting of the classes and their relationships, attributes, and method interfaces; b) object classes for the menus, forms and reports; and c) a behavior schema, which consists of detailed descriptions of the methods and the application transactions, expressed in pseudo-code and message diagrams. The seamless transition from analysis to design is attributed to ADISSA methodology, which facilitates the design of the menus, forms and reports classes, and the system behavior schema, from DFDs and the application transactions.

This chapter appears in the book, Advanced Topics in Database Research by Keng Siau.

# INTRODUCTION

## The Functional Approach and ADISSA Methodology

Many paradigms for system analysis and design have been proposed over the years. Early approaches have advocated the functional (or process) 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 (usually user-entities, but sometimes also other types, e.g., time-entities), and the data flows among the above components. Early SSA and similar methodologies emphasized the functional aspects of system analysis, neglecting somehow the structural aspect of data model. This was remedied by enhancing those methodologies with a data model, usually the entity-relationship (ER) model (Chen, 1976), that is used to create a diagram of the data model, according to which the database schema of the application is designed.

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 an approach where functional analysis is 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, 1991) developed the ADISSA methodology that solved this problem. It uses hierarchical DFDs during the analysis stage (similar to other functional analysis methodologies), but the design centers on *transactions design*. A transaction is a process that supports a user who performs a business function, and is triggered as a result of an event. Transactions will eventually become the application programs. The transactions are identified and derived from the DFDs. A transaction consists of elementary functions (namely, functions that are not decomposed into sub-functions) that are chained through data flows, and of data stores and external entities that are connected to those functions. Hence, a transaction consists of at least one elementary function and one external entity, which serve as its trigger. The process logic of each transaction is defined by means of structured programming techniques. Based on the hierarchical DFDs and the transactions, ADISSA methodology provides well-defined procedures to design the user-system interface (a menu-tree), the inputs and outputs (forms and reports), the database schema, and detailed transactions descriptions, which are eventually translated into application programs.

The sub-stage of interface design results in a menu-tree, that enables users to find and to fire desired transactions. The menu-tree is derived from the hierarchy of DFDs in a semi-algorithmic fashion, based on functions that are connected to user-entities. Briefly, a general function that is connected to a user-entity produces a menu, at some level, while an elementary function that is connected to a user-entity produces a menu item within the menu created from its super-function. Obviously, the hierarchy of menus is equivalent to the hierarchy of DFDs. Certain rules that take into account human-engineering factors enable us to modify the initial menu-tree (which is generated algorithmically). (More details can be found in the ADISSA references, and in Shoval, 1990.)