PANEL: Structured Analysis and Object Oriented Analysis

Dennis de Champeaux, *Hewlett-Packard Laboratories, Moderator*
Larry Constantine, *Independent Consultant*
Ivar Jacobson, *Objective Systems SF AB*
Stephen Mellor, *Project Technology, Inc*
Paul Ward, *Software Development Concepts*
Edward Yourdon, *American Programmer*

The object-oriented paradigm still faces an open challenge: Delivering huge software systems routinely and cost effectively. To quote Ed Yourdon: "A system composed of 100,000 lines of C++ is not to be sneezed at, but we don't have that much trouble developing 100,000 lines of COBOL today. The real test of OOP will come when systems of 1 to 10 million lines of code are developed."

Although the object-oriented community has an opening flirtation with exploratory programming and rapid prototyping by exploiting reuse via inheritance, there is for now, in my opinion, no hope that huge systems can be developed without giving due attention to what a target system is supposed to do. Which should produce an (electronic) (graphical) (Pseudo-formal) document, the requirements, that a customer can initially sign off. We believe as well that for huge systems a programming language independent design activity, that bridges the requirements and the actual programming effort, is mandatory. It goes without saying that we do not suggest that these activities constitute a waterfall sequence.

Consequently, the object-oriented community needs to address the question whether well established analysis techniques, like Structured Analysis, Jackson's JSD, etc. can be reused for object-oriented system development or whether a dedicated object- oriented analysis (and design) method is called for.

The panel members have been asked to consider the following of question: What is the relationship between Structured Analysis (SA) and Object Oriented Analysis (OOA)?

More specifically:

- Can SA be used effectively to produce the requirements for a system that will be designed and implemented in an OO fashion?
- If not, is it possible to adjust SA, what needs to be added? If SA cannot be used at all, what is the key obstacle?
- In case SA and OOA have different applicability ranges, how do we circumscribe — positively and negatively - these ranges? Any overlap?

We appreciate that the organizing committee of this conference has selected this crucial topic.

Larry Constantine, Independent Consultant

In coanventional structured methods, the model of a problem or application and the model of the software that solves the problem are quite distinct and are typically represented in completely different notations. Using object-oriented organization, it is possible to design software that models the structure of an application more closely. At least in principle the models developed in OO analysis and in OO design can be expressed in the same or equivalent notations based on common principles.

Conventional structured analysis can be and has been used successfully as the "front end" to object-oriented design and programming, but the models produced may provide only somewhat limited leverage toward developing the object-oriented design model. Experience suggests that approaches based on entity- relationship models and their extensions are better for this purpose than are data flow models. As larger applications are developed, the need for specifically OO analysis models and mthods increases.

Iterative, exploratory, or prototyping approaches further blur the boundaries between OOA and OOD. Nevertheless, although more tightly coupled than SA and SD, OOA and OOD are not identical activities. From a managerial standpoint, it may even be desirable to artificially increase the differences in order to enhance the controllability of OO software development.

Ivan Jacobson, Objective Systems SF AP

Two of the most used paradigms for software development are the function and data approach and the object-oriented approach.

The function and data approach models a system in terms of two concepts. Without trying to be precise, functions are active and have behavior whereas data are passive containers of information that can be manipulated by the funtions. The function and data approach models abstractly the behavior of a traditional computer system with a program and a data store. Most traditional software engineering methods such as SA, JSD, SREM (RDD), SADT are function and data methods.

In the object-oriented approach (and here I also include object- based techniques) a system is modeled in terms of objects only. These objects offer services (behavior) to the surrounding world and they contain information. Real-world entities are directly mapped onto objects in the model world. This is in contrast to the function and data approach whee the real-world entities are mapped onto two structures: functions and data.

Both approaches have been in practical use many years, say 25 years. The function and data approach has more users particularly for data processing systems. The object-oriented approach has primarily been used for large complex applications such as telecommunication systems. It seems today that within five or ten years the object-oriented approach will dominate, for both technical applications and information systems.

My position can briefly be summarized by:

1) The object-oriented approach should be introduced as early as possible in the lifecycle of a system. Thus, one should use an object-oriented technique for enterprise modeling, use it for systems analysis, for systems design and for programming. A shift of paradigm from one development phase to another is very complex. It requires training in two different

paradigms. The developers must manually translate from one set of modeling concepts to another.

2) Given that one has an analysis model based on a function and data approach, there is a way to manually tranform this model into an object-oriented model. This way should be faster than starting from the original requirements specification. We recommend that the translation is done by those anlaysts that did the earlier model.

3) Some of the diagramming techniques used in SA can also be used in OA. State transition graphs are useful to model some object types. Data flow diagrams can be used at a low level to model complex objects.

Stephen Mellor, Project Technology Inc.

The relationship between Structured Analysis and Object-Oriented Analysis depends primarily on the definition of the terms. The "structured" in SA can be defined to mean "organized" and "systematic" both in terms of the approach to the analysis effort, and the framework for the resulting documentation. Functional decomposition (traditionally identified with SA), the event-response approach (McMenamin and Palmer, Ward/Mellor,) and OOA can all, to a greater or lesser degree, be identified as members of the structured analysis family.

Functional decomposition and event-response cannot be used effectively to construct systems in an OO fashion because the analysis proceeds by examining processes in the former, and by considering events in the latter. In both approaches, data is organized only to support processing. On the other hand, abstract data types are the basis of OOA: first the analyst finds conceptual entities, then describes their behavior over time, and finally derives the processing.

However, the other analysis approaches are less effective than OOA. Functional decomposition fundamentally fails to address the semantics of a problem. The event-response approach was defined precisely to address this issue, but solve it by taking a phenomenolgical view of systems more suited to specification (what other user sees) than analysis (what the world really is). Consequently, we should be asking if there are any obstacles to using OOA in traditional systems design, rather than the inverse.

The applicability range of an analysis technique can be divided into types of problems, and types of designs to which the technique applies. All systems have semantics — even highly functional systems — so the semantically driven approach, of OOA applies.

Design is the choice of the rules for the orgainization of data, control and algorithym in a system, independent of the analysis and the programming language. No analysis approach should incorporate concepts of a pre-chosen design approach, even an object-oriented design. This applies especially to inheritance which exists only to a limited extent in the real world.

- precisely defines the semantics of a problem in terms of the conceptual entities;

- is based on abstraction from the real world; and

- permits the designer to choose the most appropriate inheritance structure.

OOA meets these requirements in full.

Paul Ward, Software Development Concepts

Structured Analysis (SA) is an eclectic approach to systems development that has evolved to accommodate a variety of notations and model building heuristics. with the additon of some object-oriented modeling guidelines, SA can be used effectively to express a specification in terms of classes of objects with inheritance properties. Such a specification can then be translated in a straightforward way to an object-oriented design.

The modeling notations of SA, originally confined largely to the data flow diagram, have evolved to include the entity- relationship diagram and its extensions, and various graphic and tabular state machine representations. This group of notations is capable of expressing various views of an objec-oriented specification model. A two-level data flow model can show interacting objects, and the operations and encapsulated data (instance variables) for each object. An entity-relationship model can show the structure and associations of the classes of objects in a problem domain. A state diagram or table can describe the life-cycle of an object.

The model partitioning guidelines of SA, originally confined (for" logical" models) to functional process partitioning and ad- hoc data partitioning, have evolved to include as alternatives event-response process partitioning and object-based data partitioning. If responses are decomposed into components that access single data objects, an event-response model can be partitioned in terms of interacting objects. If the entity- relationship model identifies superclass objects, corresponding superclass process (operations) can be embedded in subclasses in the data flow model.

The overall top-down structure characteristic of SA data flow models was originally closely tied to a functional decomposition strategy for model development. A more recent approach accepts a top-down organization of the final mode, preceded by a more "moddle-out" model building strategy. This permits object identification as a preliminary strategy, followed by a top-down packaging based on composite objects.

Since many current CASE products implement the various SA notqtions, these products can be used, with the guidelines mentioned above, to create object-oriented specifications models.

Edward Yourdon, American Programmer

We have been asked to consider several questions about the relationship between structured analysis and object-oriented analysis. My position on these questions is as follows:

1. Can SA be used effectively to produce the requirements for a system that will be designed and implemented in an OO fashion? The key word in this question, I believe, is "effectively." From this perspective, the answer is "no." The tranition from structured analysis to structured design is difficult enough; the tranition from SA to OOD/OOP is even more difficult, because the modeling notation is so different, and because the emphasis is so different in SA (i.e., functions) than it is in OOD/OOP (objects). Brute-force methods can be used to move from SA to OOD, but it is not a seamless integration.

2. If not, is it possible to adjust SA, what needs to be added? If SA cannot be used at all, what is the key obstacle? Some people have claimed a successful marriage between SA and OOA by using the "event-partitioning" approach described by McMenamin and Palmer in Essential Systems Analysis to "objectify" a requirements model initially created with data flow diagrams; this would them make it possible to go from SA to OOA. But there are three

problems with "classical" SA that will render it, I believe, inadequate for a world that want to implement systems in an OO fashion. These problems are: (1) separate disjoint notations for process (DFDs), data (ERDs), and time-dependent behavior (STDs), one of which (usually the DFD) usually dominates and eliminates the others in real world projects; (2) thus subtly influences software engineers to approach each systems development project as a "design from first principles," rather than the OO paradigm of "design by exception." (3) The SA notation provides no help in modeling the human interface, which is becoming more and more important in the world of GUI environments.

3. In case SA and OOA have different ranges, how do we circumscribe - possitively and negatively - these ranges? Any overlaps? I regard this more as a cultural issue than a technical issue. Even if OOA is more applicable than SA, from a technical perspective, many DP organizations have too much inertia to switch. New paradigms are generally accepted only when the old paradigm fails to solve new problems with which the organization is faced. Thus, I feel the OOA will be more politically acceptable in environments where SA has failed on large, visible projects; and also on projects where reusability and graphical user interfaces are seen as key issues from the onset.