Appendix A.

1 From Structured (Analysis and Design) to Object-Oriented Analysis and Design

Introduction

The so-called "structured approach" to **S**ystems **A**nalysis and **D**esign (S.A.D.[1]) became the most common approach by the early 1980's. Two different techniques were used by adherents to S.A.D., i.e. Data Flow Diagrams (DFD's) to represent the "process" being modelled, and Entity-Relationship diagrams (E-R diagrams) to represent the "data" being modelled. Once a suitably complete description in terms of DFD's and E-R diagrams was developed, the resulting diagrams (which are intended to be independent of any hardware or software considerations) which collectively represent the "analysis model" are 'transformed'[2] into a "design model" which was usually represented in terms of the structures supported by a relational database system.

By the mid 1990's it became clear, even to the most die-hard S.A.D. adherent, that the approach was inflexible, could not support successive adaptive refinements across the "anaysis" and "design" stages, and could not directly support the notion of (component) reuse.

Whereas S.A.D. places its emphasis (initially at least) on "processes" rather than "data", it became evident that object-oriented programming languages were now widely used, and hence, that some equal measure of importance should be given to "data" if systems were to be usefully analysed and/or designed with the intention of implementing them using object-oriented programming languages.

UML

The **U**nified **M**odelling **L**anguage (UML) provides an example of an attempt to shift the emphasis away from "processes" also (in the first instance), and to incorporate considerations of "data" and processes in a way that is analogous to the notions of operations over objects with (or on) attributes in the OO programming language paradigm.

Thus, UML "incorporates" the notions of classes, attributes, methods, inheritance, etc as drawn from the OO programming language paradigm and permits these notions to be used as the basis of developing descriptions in a graphical notation of software systems prior to their construction in (hopefully) an OO programming language.

Developing UML descriptions can be characterised by the "bottom-up" nature of the kind of thinking involved, i.e. we identify "objects" (i.e. class instances) and then, with a suitably large set of such instances that have common structure and/or behaviour, we hypothesise classes which abstract over the mutual differences between our related objects.

The typical text on UML contains the typical example of its application, e.g. some form of "shop" (either a typical hypothetical shop, or in more recent texts an "on-line" shop) and proceeds by identifying the obvious (?) "actors" (or some other equivalent abstraction) in the "system", e.g. the "customer", the "salesperson" or the "warehouse", etc.

Next, so-called "use cases" are identified each of which is intended to represent some aspect of "functionality" of some value to each actor. These use-cases are typically derived by considering the "tasks" of an "actor" using such searching questions as:-

"What are the actor's responsibilities towards the system and expectations from the system?"

So-called "functional requirements" (as opposed to so-called "non-functional requirements") are then used as a means of associating individual "requirements" with given "actors" in a given "use-case", e.g.

1.1 Req. No. 1

> The customer uses the manufacturer's on-line shopping (web) page to view the products offered and their prices.

1.2 Actor

> Customer

1.3 Use Case

> Display standard product list and product details

From such "descriptions" a use-case diagram representing a "use-case model" is constructed in which "requirements" and their associated "actors" and "use-cases" are related to one another.

Next, the flow of events in a use-case provide the basis for generating an "activity model" (in more recent versions of UML) in order that, ultimately, interaction models composed of sequence and collaboration diagrams can be derived from the activity model and then represented in UML.

In addition to the above, other "stages" in the use of UML involve the description of classes, attributes, associations between "things" represented by aggregation and composition, and the notion of an "interaction" introduces the need for an "interaction model", etc, etc, etc.

Thinking and OO Design

Some idea of the kind of "thinking" that is directly ascribable to the author of a book on UML (we do not need to know which author) can be gleaned from the following statement:-

"The *system state* is what the system consists of. The state is the function of the systems information content at any one time - it is the function of the system's current set of instance objects."

----

[1] A most unfortunate acronym.

[2] The notion of a *transformation* in this context implies that many alternative design models can be derived from the analysis model.