DESIGN PHASE, I: Object-Oriented Design

This part is based upon chapter 6 of Schach's "Practical Software Engineering".

Objectives:

- Appreciate that design is an essential part of the software life cycle
- Realize why a function should perform one task
- Discover what an object is
- Learn how to design a product in terms of objects
- Recognize the critical importance of reusability

WHY DESIGN?

Why one have to design a product?
There are two reasons:

1. Cost of Development

2. Cost of Maintenance

As to the cost of development:

- If there is no design one uses essentially the build-and-fix model, since until now one has only specified what has to be done, but not how that is to be done.

As to the cost of maintenance:

- It is inevitable that a product will undergo changes. Therefore the real issue is how easy it is to make changes.

- If a product is well-designed by using objects this has **two** results:

  1. It is easy to determine **what** has to be changed. This is because, using information hiding principles, any part likely to be changed (see transparencies 208-212) has been hidden from other modules.
  2. these changes can be localized using the same principles of OO design.

For these two reasons (costs of development and of maintenance) it is foolhardy to build a product without designing it.

So: HOW DOES ONE DESIGN A PRODUCT?

OBJECT-ORIENTED DESIGN

The superior use of objects in design has been documented on transparencies 173-216.

In the context of the team project "Chocoholics Anonymous", it is important to realize this project is implemented within C. Consequently "inheritance" and "information hiding" is not fully supported, and, for the duration of the project, we adhere to the following principles and definitions:

> **The fundamental building blocks of a project are $objects$; these are compilation units consisting of a data structure together with all operations that are performed on that data structure.**

Object-Oriented Design (OOD) is essentially a technique that extracts the objects and results is a design that is based on these objects. It is a four-step process:

STEP 1 Define the problem as concisely as possible.

STEP 2 Develop an informal strategy — a general sequence of steps for satisfying the specification document subject to the constraints.

STEP 3 Identify the objects; that is the data structures and the operations to be applied to those data structures.

STEP 4 Perform the detailed design.

Next the OOD is applied to the C-CC case study.

OOD CASE STUDY

STEP 1 *Concise problem description.* A definition of the problem in, preferably, one sentence:

> Monthly effectiveness reports must be reproduced for C-CC shops.

STEP 2 *Informal strategy.* Describe the strategy for solving the problem is a single paragraph:

> C-CC has a set of shops organized into regions. At the end of every month, reports have to be produced for each shop and each region, and for the corporation as a whole. The reports reflect whether or not sales objectives have been met.

STEP 3 *Formalize the strategy.* Identify the objects. That is: the data structures and the operations to be performed on them.

- This is usually done in OOD **by identifying the nouns in the informal strategy,** excluding those that lie outside the problem boundary and then using the remainder as the basis for the data structure of each object.
- Then **the verbs are identified,** and used as the basis for the operations of these objects. This usually requires **stepwise refinement.**
- Finally, as last step in **formalizing the strategy** and as a preparation upon detailed design, a top level design is produced featuring the components in term of which the design of the product will be structured.

So first the **NOUNS** in the informal strategy above are identified:

> C-CC has a SET of SHOPS organized into REGIONS. At the end of every MONTH, REPORTS have to be produced for each SHOP and each REGION, and for the CORPORATION as a WHOLE. The REPORTS reflect whether or not sales OBJECTIVES have been met.

Six of the nouns in the example can be discarded for the moment:

- SET, because it will be implemented
- MONTH, WHOLE, OBJECTIVE, because they are abstract nouns, which are within our example unlikely to correspond with an object
- C-CC, CORPORATION, because they lie outside of the problem boundary.

Consequently, three candidates for objects remain:

- REGION, REPORT, SHOP

Of the verbs, only **produce** and **meet objective** relate to these candidates.

This results in our **first refinement** of the Object-operation table:

Object	Operations
REGION REPORT SHOP	*produce* *meet objective*

Figure 6.6 Object-operations table: first refinement.

In view of our definition of data stores in the Specification document case study, see transparencies 162-163 and 151-156, this leads to the **second refinement:**

Object	Operations
REGION	*read region_record*
REPORT	*produce*
SHOP	*read shop_record*
	evaluate sales_objective

Figure 6.7 Object-operations table: second refinement.

Next, in the context of the requirements phase, it is decided that **"evaluate sales_objective"** is done by the vice-president for sales, and consequently that the reflection of this upon the product leads to identification of **produce** a REPORT and of **evaluate sales_objective**, making object REPORT redundant.

This leads in view of the Dataflow diagram for the C-CC corporation, transparency 143, to fig. 6.8 below as **third refinement**.

Object	Data Structure	Operations
REGION	region_record	*read region_record* *write region_record* *delete region_record* *generate region_report* *generate vice-president_report*
SHOP	shop_record	*read shop_record* *write shop_record* *delete shop_record* *generate shop_report*

Figure 6.8 Object-operations table: third refinement.

Now the complete top-level OOD for the C-CC management-by-objectives product is produced in fig. 6.9; this figure reflects the operations upon SHOP and REGION data occurring in the DFD for the C-CC product, cfr transparency 143.

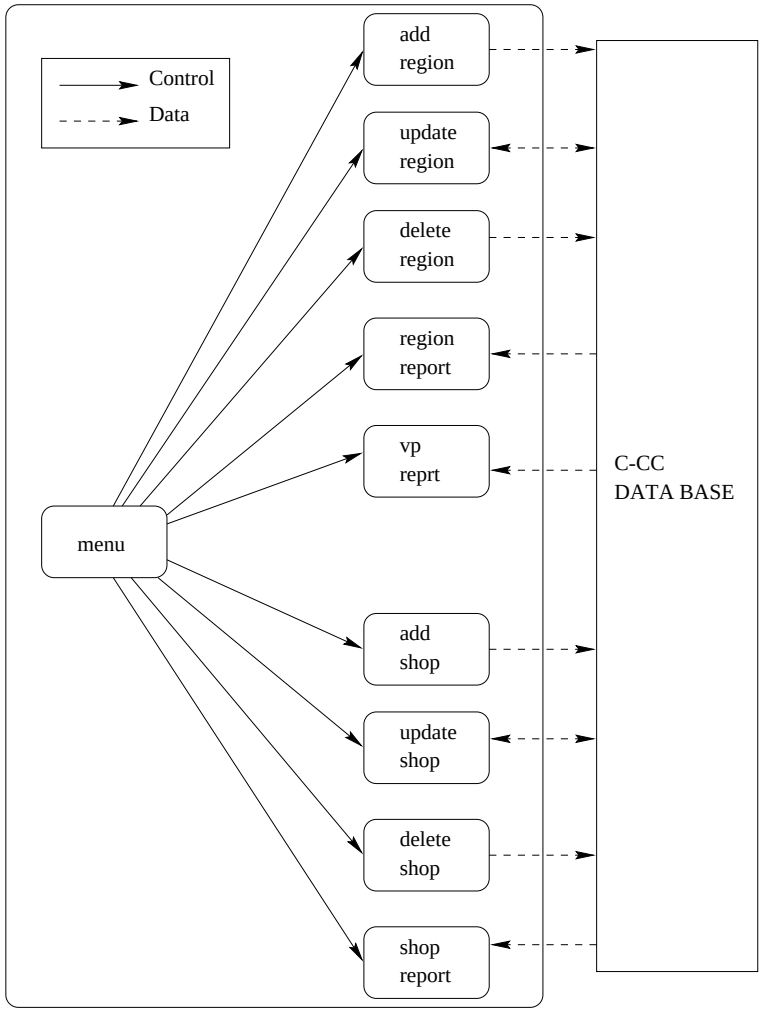

Figure 6.9 OOD of C-CC management-by-objectives project.

The refinement of these components belongs to **STEP 4** of the OOD and covers the detailed design discussed below.

REUSABILITY

Reusability is an increasingly important aspect of software engineering because it reduces development time, and, more importantly, decreases the cost of maintenance:

1. Reused functions contain fewer faults

2. Maintenance programmers tend to be familiar with reused functions, and consequently:

3. it is easier for the maintenance programmer to understand the product as a whole, as a consequence of their familiarity with these functions, and:

4. Reused functions tend to generate less regression faults.

In practise whole chunks of specification documents and manuals can be reused. Studies have shown that:

> achieving a 40% level of reusability is an attainable goal.

Summary:

- Promoting reusability decreases the very high, and still increasing, cost of software; however, its real savings occur in the maintenance phase.

- If all the products developed by an organization consist as far as possible of standard products, maintenance will no longer be a nightmare consuming two-thirds of the time and effort invested in a software product.

(TERM PROJECT) Perform the OOD of the ChocAn product.