Unit 5. Industry, Handicraft and Tools

Objectives of the unit

At the unit end the student should know

- Batch and interactive programs
- Programming evolution
- Objects-oriented programming
- Programming evolution during the decades

Batch & Interactive Programs

```
BEGIN
        ....                    (A)
        ....
        DO ....
           ....
           ....
           ....                 (R)
           ....
           ....
           ....
        END-DO
        ....
        ....                    (F)
END-BEGIN
```

Figure 5-1

Programs are thousands and thousands. They calculate the salaries, they forecast the sales, they make reservations on the flight, they play etc. Algorithms are countless and vary widely at a cursory glance. Everlasting invention seems to push programming.
We cannot settle for this view, as we have to get our bearings in the area. We cannot look like those who travel and ignore geography. Landscapes are unlike anything they have seen before and they are unable to orientate themselves due to ignorance Disorientation holds back computer professionals and users, who instead are to advance in the field. Thus we seek for the main directions of software programming despite the colored production.

As first we examine the programs written with traditional and most aged languages such as: Cobol, Pascal, PL/1, Basic, RPG etc. They constitute two groups:

- ***BATCH PROGRAMS***
- ***INTERACTIVE PROGRAMS***.

The former are chiefly structured by DO…END-DO and the latter by SELECT…END-SELECT. These control macros arrange the operations in two basic ways and form two algorithms.

*** * ***

The batch algorithm begins with the starting phase (A) which makes ready the machine. For ease (A) fixes the counters, it sets up work-areas with the initial values, it gets the first record. The batch program steadily performs the cycle (R); and terminates with the end phase (F) that gives sums, synthesis and final conclusions. The batch algorithm usually processes a sequential file and assumes the following shape. The loop ends when the records finish at the end of the input file.

```
....                    (A) prepares the whole process
....
DO UNTIL the end-file
    ....
    ....
    ....                (R) calculates a salary
    ....
    ....
    ....
END-DO
....
....                    (F) concludes
```

The program that calculates the salaries provides a valuable example. The input files regard the worked hours and the registry of employees. After the initialization (A), the program computes the salary of every employee through the block (R). When the file finishes, (F) computes the grand total of the salaries to be paid.

Common batch programs process a large number of records and obey to standard procedures. E.g. The computation of the salaries complies with general rules and cannot be customized.

Batch programs were overwhelming in the past and are still very common. Companies, business, institutions handle a large amount of data using general criteria and cannot forgo batch programs.

Batch & Interactive Programs (contd.)

```
BEGIN
  ....
  SELECT
      IF  A
      THEN SELECT
                IF   K
                THEN ....
                IF   J
                THEN ....
           END-SELECT
      IF  B
      THEN SELECT
                IF   W
                THEN ....
                IF   Z
                THEN ....
           END-SELECT
  END-SELECT
  ....
END-BEGIN
```

Figure 5-2

The interactive program offers a menu to the user, then another choice and so on. The selections of the human operator determine substantially the outcome of the interactive algorithm, which personalizes by definition.
To exemplify, the first menu displays the buttons A and B; A leads to K and J; B leads to W and Z. (Figure 5-2b).The SELECT macro handles the choices in the software programs(Figure 5-2).

The interactive algorithm ties several SELECT macros, which make three main schemes

- **Simple Tree**: The menus are given in succession with a common root. Figure 5-2b and 5-2 provide an example. Figure 5-2c exhibits the implementation in general.
- **Cyclic Tree**: Selections may be repeated.
- **Acyclic Tree**: Selections follow no specific pathway.

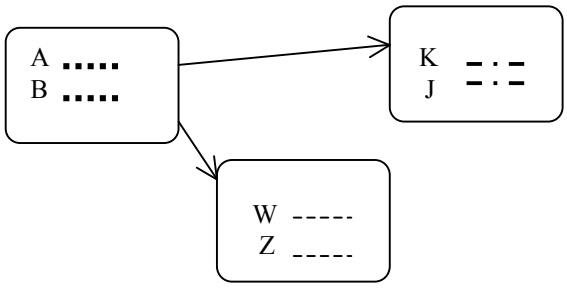


Figure 5-2b

The interactive algorithm manages chained choices and also asynchronous and unexpected tasks that break off the selections. An interrupt occurs if the user makes a mistake, he presses an abnormal key or something else.

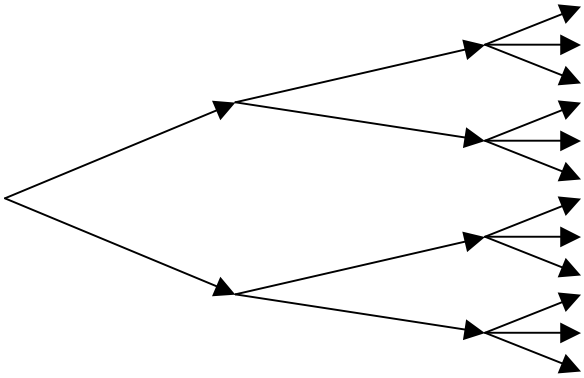


Figure 5-2c

The interactive program asks the user to decide and to contribute to the information process again and again. It opens and keeps a long dialogue with people; hence software developer's attention turns to human and subjective aspects. Ergonomic issues heavily condition interactive programming.
Conversely batch programming needs a rationale organization of tasks independent on psychological feeling. The programmer apply criteria absolutely rationale and works like the organizer of a plant.

Batch & Interactive Programs (contd.)

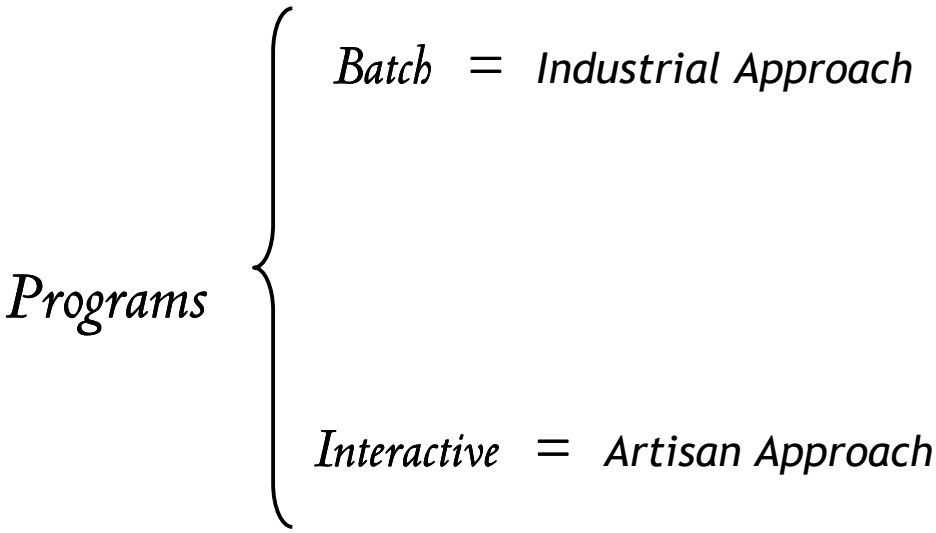


Figure 5-3

Which is the meaning of batch and interactive programs? Which duties do these software products fulfil?

Before answering, we make a preliminary stage.
Dynamical systems assume two different and somewhat opposite behaviors. They bring about either **industrial** or **artisan** production. They observe two diverse standards that we outline as follows.

i) - ***Industrial System***

1 The operations are repetitive (e.g. the assembly line).
2 It produces large quantities and achieves the economy of scale.
3 It has scarce interaction with customers.
4 The production is rather rigid.
5 Customers' tastes are filtered.
6 The processes are scheduled in the time.
7 The quality of results is standard.

ii) - ***Artisan System***

1 It is rather creative (e.g. the tailor).
2 It produces limited quantities and ignores the economy of scale.
3 The system interacts with customers.
4 The production diversifies.
5 The customer tastes are repeatedly felt.
6 The processes run in real-time.
7 The results reach a high quality.

Every statement of the second group clashes with the corresponding line in i), and brings evidence of their opposite qualities. Computer systems process information that is physical hence our logical framework holds that they comply with the industrial and artisan styles.

Experience proves that *the batch program complies with the industrial standard* and *the interactive program suits the handicraft approach*. We provide some remarks.

I) **The batch program follows the industrial standard** since it processes a large amount of data, namely thousands and thousands of records (see point i-2). It accomplishes general rules (point i-1). Inputs are somewhat regular, as the algorithm treats sequential files with fixed records. Exceptions are managed manually or by specific procedures (point i-4). Programmers evaluate with care any request to personalize the process (point i-3,5). Several batch programs are scheduled e.g. the computation of salaries is scheduled in a fixed day of the month.
Last we realize the industrial style through the algorithm (Figure 5-1). It starts with the preparatory phase, then works steadily, at the end it closes the production. These three stages are typical of the industrial manufacture that carries on mass production.

II) **The interactive program suits the handicraft approach** since it interacts with the user again and again through menus, windows and frames. The user takes a decision and passes from one screen to another. He can go forward and backward at his will (see point ii-3,5). Interactive programs produce a small amount of information that however is sophisticated (point ii-2 and ii-7). The artisan serves also a distant client; similarly the interactive program runs in a network. Quick reactions qualify the interactive application and "real-time program" is synonymous with interactive program (point ii-6).
Finally, the algorithm illustrates the articulated talk with the user. Take for example Figure 5-2.

Files and Data Bases

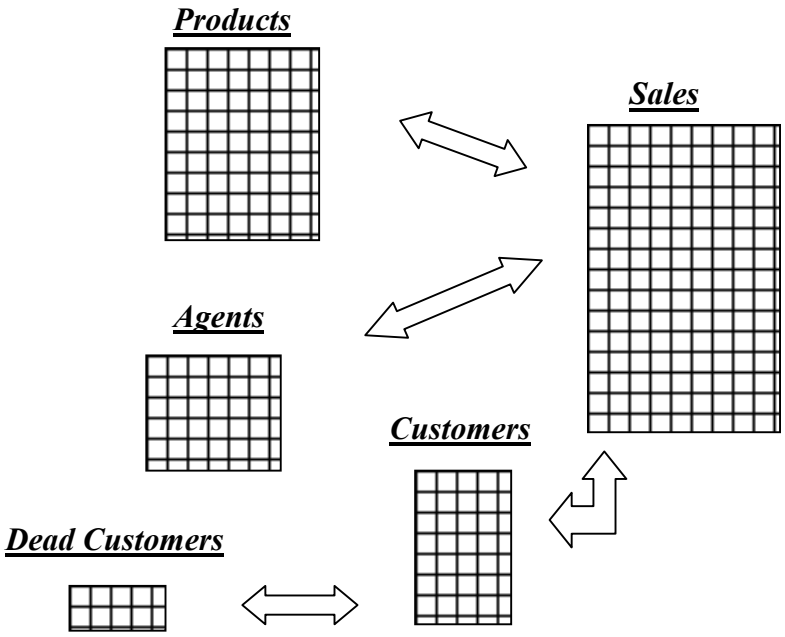


Figure 5-3

Plants and artisan workshops need warehouses to store feedstock, materials, accessories, end-products etc. Batch and interactive programs require symmetrical resources to store information that is physical indeed.

Broadly speaking, two main criteria lead the preparation of depots.

1) The form of the storehouse must conform to the tasks that exploit it. For instance, the super-market offers self-service goods hence the products lie in open and low racks; the forecourt is designed for cars ready to exit; the basin receives the water near the land to irrigate. The store harmonizes with the operators it serves.

2) Stores do not stock scattered and spread items, instead they must keep them inside special units. For instance the library keeps the books in bookshelves; the harbor collects commodities into the containers; the plant seals the products into the pallet. Bookshelves, containers and pallets are storing-units.

Experts of ICT can do nothing but apply ***1)*** and ***2).*** Now we summarize what they do.

1) Automatic tasks fall into two main classes: they follow precise algorithms or otherwise an unpredictable logic. As a consequence the storage of data serve two main software operators. There are two forms of organization in a computer system: the **file** and the **database**.
   - The file collects information for a specific program or a group of programs, that it is to say, it is designed for precise algorithms. In consequence of this

service, the file has a rather rigid structure. Any change requires the re-installation of the file. The file merely gathers the contents necessary for some programs thus it is not exhaustive.

- The database serves any elaboration: the present and the next ones. It is suitable for inquiries and free searches as well. In short, the database is capable of serving an unpredictable logic. This storage collects all the data regarding a precise matter. In practice we have the "Marketing Database", the "Production Database" in a firm; the "Student Database" in a school; "Earthquakes Database" in a geographic institute. These exhaustive collections include numerical and verbal information, along with photos, pictures etc. The file is rigid instead the database is flexible and may be easily modified according to new requirements. In practice people *manipulate* the database by means of special instructions. For ease, the user extracts the inactive customers from "Customers" and creates the new table "Dead Customers".

2) Both the file and the database have storing-units.
   - The file depends on the algorithm, hence this one dictates the contents of the storage-unit. E.g. the program computes the salary and the input file must contain all the data of the employee necessary for this calculation such as name, address, title, seniority, qualification etc. The storage-unit, named **record,** is the sequence of fields that the algorithm displays or receives.
   - The database concerns a broad field and the **table** is the storing-unit specialized in a subject of the major topic. To exemplify the tables of the "Marketing Database" (Figure 5-3) are devoted to the customers, the sales, the products, the agents etc. Note how the table "Customers" has all the attributes of the customer instead the file "Customers" barely contains the attributes needed for the program processing. Special links connect the tables so that a user is able to find out the customers with top sales, which products they have sold, the agents who have conducted best business etc. These relationships enable the complete exploration of the major topic.

Any technology allows a certain degree of freedom. Practitioners loosely use files and databases namely they apply the point 1) at discretion. Abuses and unwarranted decisions cause frequent underutilization or overuse of these resources. E.g. they restrict a database to one software application; they exploit one file for several software programs.

PROS & CONS

FILE:
- Optimized for a specific duty.
- Incomplete and rigid collection of information.
- Simple resource.

DATABASE:
- Open to a large number of algorithms and investigations.
- Exhaustive and flexible collection of information.
- It requires a huge amount of storage and special programs of control.

Way Toward Reduction

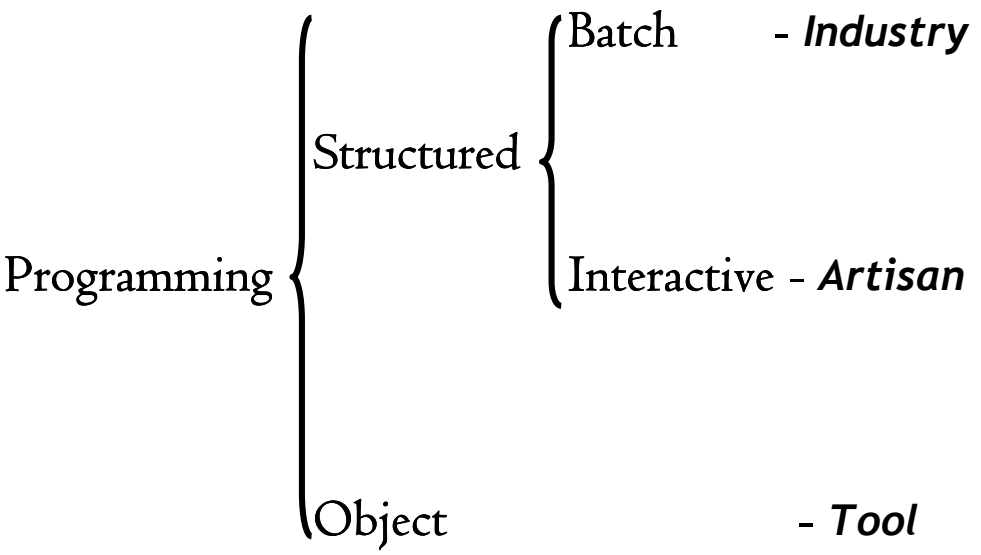


Figure 5-4

We wonder: For which direction do software programming head?

Industrial plants are the most complex systems and we cannot conceive a software structure more huge than the batch program. We have no choice but to seek for more simple arrangements. Moreover the advance of microelectronics pushes toward reduction.
We find the tool and the prosthesis in the road toward simplification after the industry and the artisan shop. In fact, productive units rage themselves as follows:

- **Industry**
- **Handicraft**
- **Tool**
- **Prosthesis**.

Experience holds our conclusions are true. We find the do-yourself methods besides batch and interactive processes in the world. The third technique makes

the computers to run as a simple instrument in the user's hands. Moreover we have tiny devices that conform to the human body as prostheses.

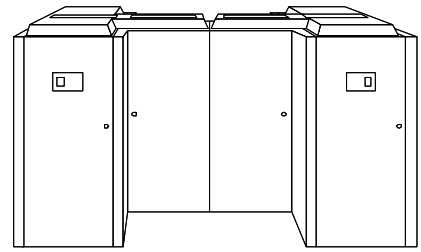
These products emerge in a tentative and experimental stage so far as now. Miniature will improve in the years to come. We bring the third software technique, amply in use, to attention.

The computer works as an instrument. It looks like the hammer operating on the nail, the screwdriver on the screw. Tools modify the objects in the world, hence the third programming technology is necessarily based on **objects.** In practice the user manipulates objects by means of the computer. Software developers design and make up software objects instead of programs.

The O-O specialists do not prepare a work-program for the machine
Instead they make several objects that the user will manipulate.

The **Objects-Oriented (O-O)** applications make the computer as a typewriter (i.e. WinWord), a tool for painting (i.e. PaintShopPro), a player for music (i.e. WinAmp), a instrument that gauges a physical quantity, a toy etc.

Programming Evolution (contd.)



years

Figure 5-5

The OO approach is really a revolution which becomes clearer when we analyze the history of the computer market from the beginnings up to now.

Batch programs, capable of carrying on large throughput, emerged since the pioneer age. They achieved economies of scale. The savings due to the mechanical processing of information, which substituted human manpower, balanced the high costs of systems in those years.

When the hardware became less expensive due to the progress of electronics, the return of investment was less demanding and the process of a small amount of information became economical. Interactive programs expanded. As first they run with terminal devices, later with mini-computers, and last with personal computers.

In the eighties the circuit performances kept improving, while the prices plummeted. The financial benefits lead software advance to its extreme

consequences. Very low costs of electronic appliances allowed using them as bare tools and the entire spectrum of software techniques was covered. Concluding.

The progress of microelectronics
Pushed the hardware engineering toward integration,
In turn the software engineering has gone toward reduction.

This scenario clarifies that the oldest techniques are not obsolete because batch, interactive and object programming pursues distinguished and different purposes. They constitute three separate software technologies and today people select the most appropriate according to the requirements.
In particular, some software applications are batch of necessity. E.g. the computation of salaries is batch. Others are evidently oriented to objects E.g. the user prepares a page, a text, an image that are objects with the editor. This application is evidently OO.
Other projects are not strictly bound to a method and developers tune the technique to the special will of the customer. As an example, the bank calculates the deposits of the clients and pays the interests due to them. The bank analysts may follow three ways:

- If the managers of the bank want to process the whole amount of data at the same time, specialists prepare a batch program whose cycle computes information of a single client.

- If the management allows calculating the interests on demand, the interactive program asks the user to enter a precise sequence of data. The system guides the user lastly it provides the results.

- If the managers offer a service on-line, the customer introduces its personal data and obtains the outcomes according to his/her choices. He/she is not forced toward the conclusion and may even interrupt the process. The software application is objects-oriented.

In conclusion, engineers plan a batch program if they want a systematic and standard production of data. Practitioners prepare an interactive program if they establish a procedural dialog. Specialists set up an O-O application in order to put the software functions in the hands of the user.

Objects-Oriented Programming

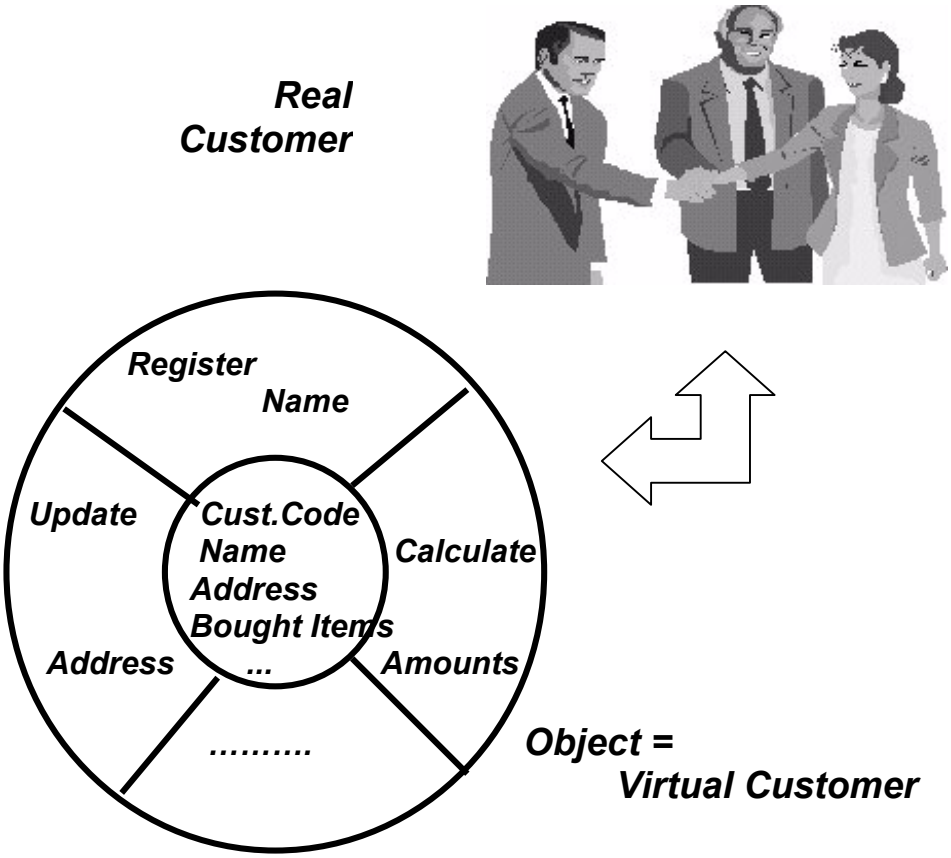


Figure 5-6

What is the software object?

An example facilitates the explanation.
The objects application schedules the calls on customers. The physical customer has a code, the name, the address, the list of bought items and the developer includes these attributes inside the software object “Customer” along with small routines that process these attributes such as “Register Name”, “Update Address”, “Calculate Amounts”. May be said that the virtual object Xo truly and univocally simulates the real object Xr.

Real Object Xr ⟷ **Virtual Object Xo**

The specialist implements the data and the software programs strictly bound to the physical object Xr and makes the software object Xo with them.
In short, the object Xo includes

- A set of data called **Attributes** (E.g. Customer Code, etc.)
- A set of programs named **Methods** or **Services** (E.g. Register Name, Update Address, etc.).

The programmer goes along with the ensuing mandatory rule: the services of Xo process the attributes of Xo and no one else can process them. E.g. “Register Name” manipulates “Name”. The services of other objects cannot process “Name”. The software object is an **encapsulated set** of items due to this exclusive relation between attributes and services. No one can penetrate inside Xo due to this protection. We draw a software object like a coconut to symbolize this property.
Encapsulation discriminates O-O programming from traditional techniques, which conversely handles programs and files as distinct products. As an example, the Cobol programmer loads the file "Customers" and implements the program concerning the clients of the company. Data and operations are disjointed components. Instead the Java programmer collects all the data and the services within the object "Customer".

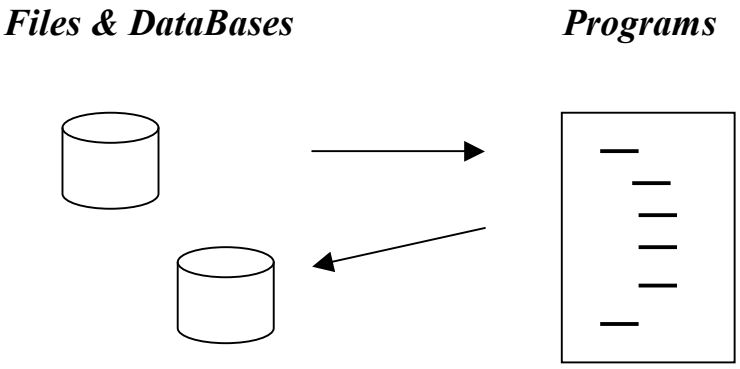


Figure 5-6b

In conclusion data and process lie apart with structured techniques, instead they are absolutely integrated in objects-oriented products.

Object-Oriented Programming (contd.)

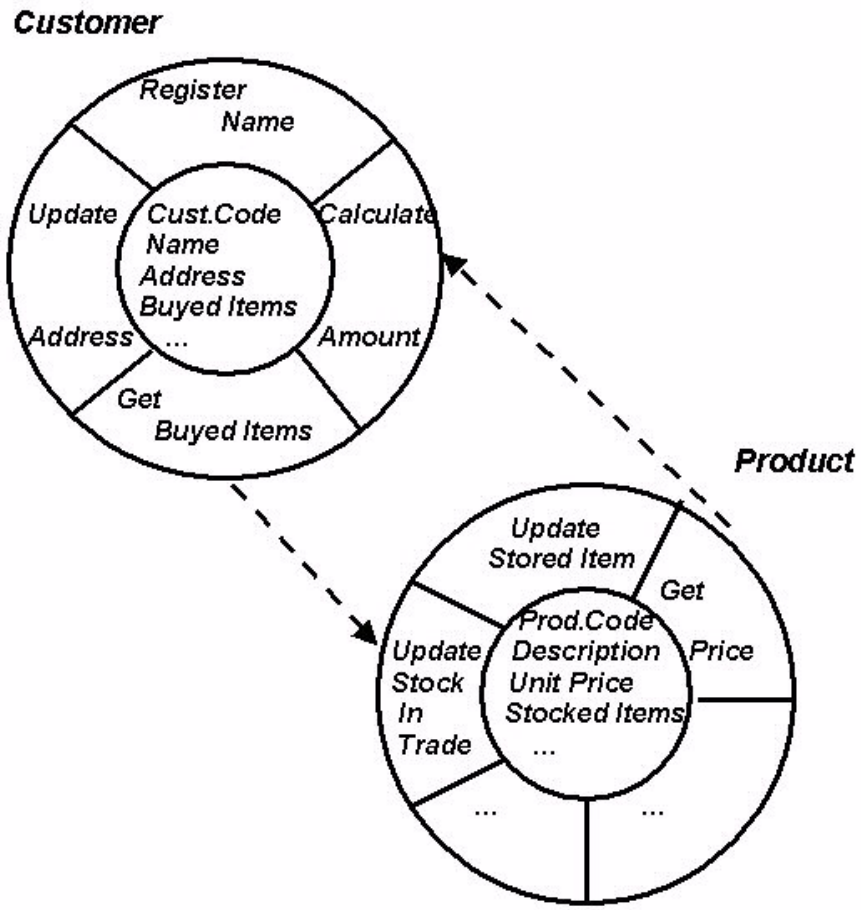


Figure 5-7

Practitioners encapsulate the objects and the services are just capable of processing internal information.

When the service of the object Xo_1 is complex and needs the data of Xo_2, it calls for the support from Xo_2. For instance, “Calculate Amounts” provides the grand total of the items bought by the customer. "Customer" has the list of the bought items but does not have the unitary prices of the products. It seeks them from “Products” that provides the prices by means of “Get Price”. May be said that “Product” **serves** “Customer”.
In general, the object A asks a service to B: the former is **client** and the latter **server**.

Objects Programming shares the client/server architecture.

In the previous example we have

"Customer" = **CLIENT**
"Product" =**SERVER**

Relations change in the client/server approach (see Chapter 3), they even invert. An example should make this clear. The object "Product" brings up to-date the stock-in-trade. The method "Update Stock-in-Trade" needs information about the items bought by the customer and asks for assistance to "Customer". This object provides the data by means of "Get Bought Items" and the objects are inverting their roles

"Customer" = **SERVER**
"Product" = **CLIENT**

OO programming complies with the client/server architecture. This basic feature makes this technology essential for the Internet. It enables amazing processes around the world. For example, the client-object runs into the host A, while the server-object runs in B miles far away.

Object-Oriented Programming (contd.)

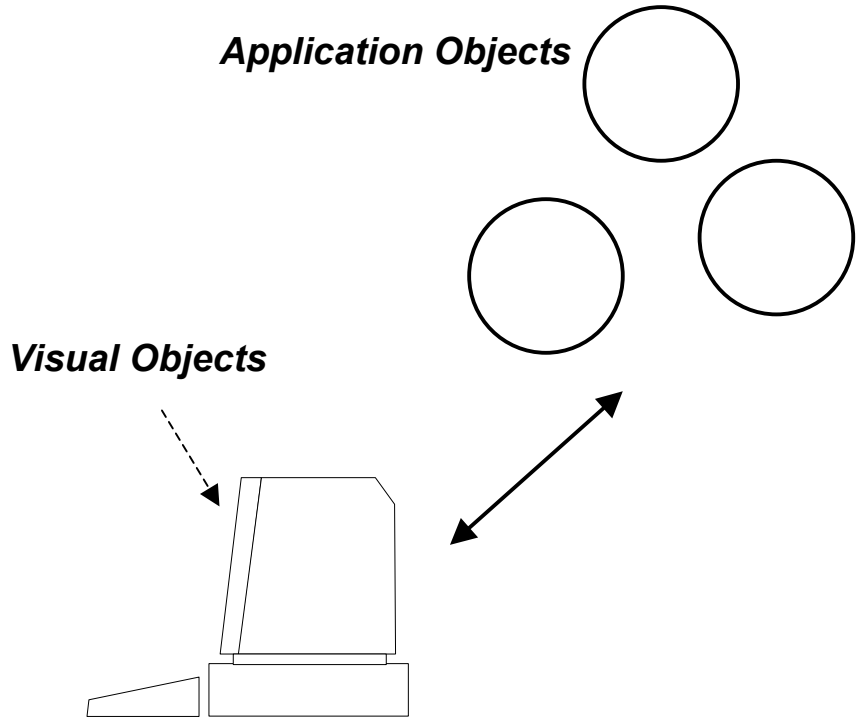


Figure 5-8

People do not manipulate directly "Customer", “Products” etc. instead they push buttons, they open windows, menus etc. This means that buttons and windows are objects. OO applications include

- **Logical (or Application) Objects** such as “Customer”, “Product”, “Count”
- **Visuals Objects** such as “Button”, “Window”, “Frame” and even “Figure”, “Panorama”, “Portrait” etc.

The user operates upon the logical objects by means of the visual objects. These accomplish lively interactions and constitute the **human-computer interfaces (hci)**.

Video games and other products for leisure require vivid interface and visual objects are appealing and attractive. Colors, shapes, sounds and other sensational features have so much weight that an OO implementation is entrusted to creative. On the contrary hci is not required to be astonishing in a bank, an office or a

company. Programmers inherit standard visual items such as buttons and windows and never invent new ones.
We consider the software applications that fall into two main classes:

- **Products for Free Time** (= games, animations, simulations etc.) are vivid, astonishing and creative.
- **Products for Work** (= calculations, editors etc.) are regular and standard.

Business applications share a modest look and put forward a precise pathway to men/women. This communication is **modal** as it flows in a regular and consecutive mode. It observes a planned procedure and may be implemented with interactive and OO technologies.
The leisure communication breaks down into countless interactions; it does not follow any rigid path. For ease the player invents ever-new actions to win and is **amodal**.

Linguistic Remark: People associate the term *program* with the concepts of "plan" and "schedule" that are meaningless in the OO environment. In fact a developer makes the computer to become a tool and the work-plan for the computer is nonsensical. He does not implement any pre-configured procedure as the classical developer does.

PROS & CONS

BATCH PROGRAM:
- Rigid procedures
- Economy of scale
- Unattended running
- Uneasy maintenance.

INTERACTIVE PROGRAM:
- Flexible procedures
- Small workload
- Constant manual intervention during running
- Uneasy maintenance.

OBJECT-ORIENTED APPLICATION:
- Robust product
- Client/server architecture
- Constant manual intervention during running.

Program Development

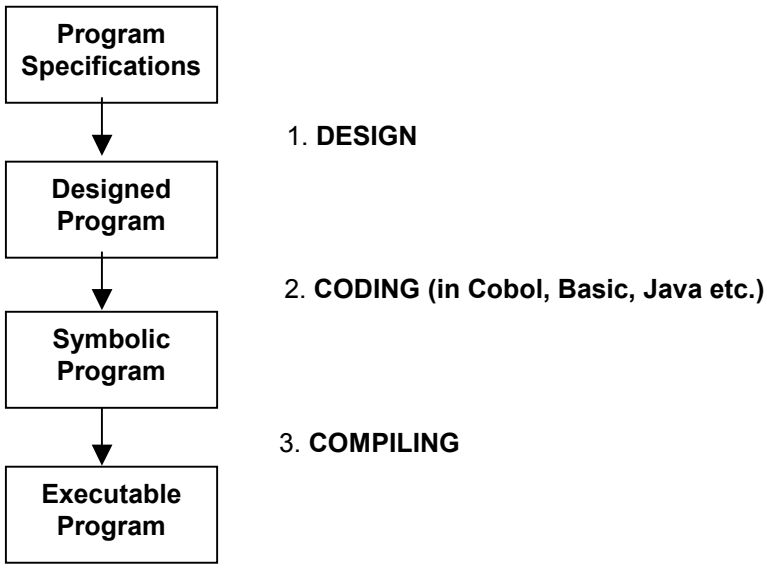


Figure 5-9

Software packages appear intricate and cannot be done all at once. We have already dealt with this topic in Chapter 3, now we return to this point.

Developers take three main steps.

1. They project the software product on the basis of the **program specifications** in advance prepared by software analysts or by the customer or others. The specifications passed to the programmer are rarely perfect and do not illustrate exactly the algorithm. The programmer makes up for this lack of particulars with an accurate job.

2. The project is translated into the programming language such as Cobol, Basic or Java.

3. The symbolic program is compiled, namely it is made ready to run by using the **compiler** and some other products.

In short, the first two stages are manual, the last is performed by automatic resources.

Program Development (contd.)

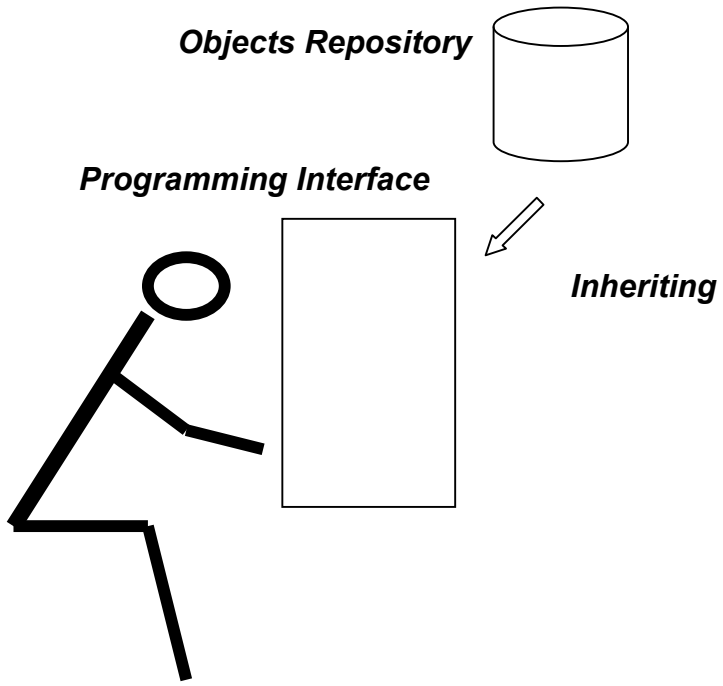


Figure 5-10

Classical developers provide the storage MC1 with instructions and MC2 with declaratives. Batch and interactive programs set up the hardware machine through direct intervention.

The developers of objects cannot immediately determine the hardware machine because there is no “object” inside it. The software objects do not match with the hardware architecture in Figure 2-6 and the steps 1, 2 and 3 become much more demanding.

We sum up the different courses in the two fields.

- **Structured Programming**: The programmer considers the hardware system as is. He prepares the software product, which directly complies with the machine architecture. The project is rather immediate and he needs a few

automatic aids such as the compiler and linker, to accomplish the tasks 2 and 3.

❖ **O-O Programming:** The programmer designs and codes objects that do not consist with the hardware resources. Hence he must follow a special approach to fulfil his duty.

- He needs a powerful platform and the producer of the O-O language provides this sophisticated interface. Some platforms are **visual,** so that a practitioner arranges the objects through an easier way.

- The programmer does not code the objects all the way through, instead he copies the objects, prepared in advance, and customizes them. We tell **inheritance** this procedure that ensures **very robust** software products. In fact inherited objects are fully tested in advance by the producer and are absolutely correct. As counterpart, inheritable objects are thousands and thousands; the specialist must study, select and use them with care. These operations cause a heavy workload.

Software Overview

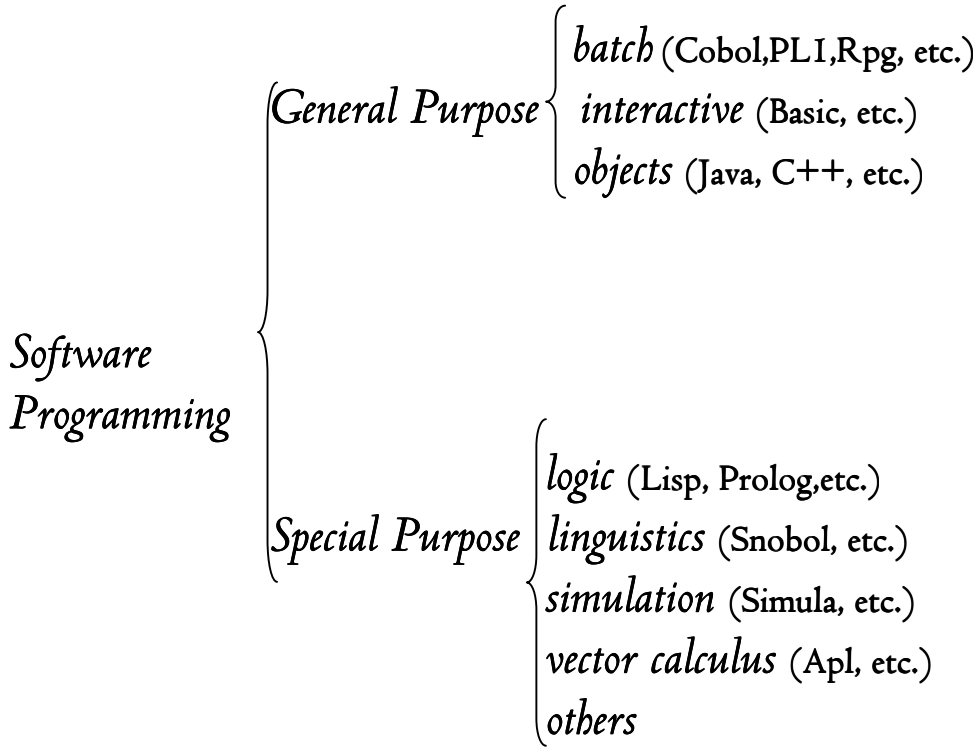


Figure 5-11

The plant, the artisan shop and the tool are organizations necessary in any environment. Batch, interactive and objects applications concern banks, business, universities, offices etc. **General-purpose techniques** group them because they tune up the computer according to three basic productive schemes.

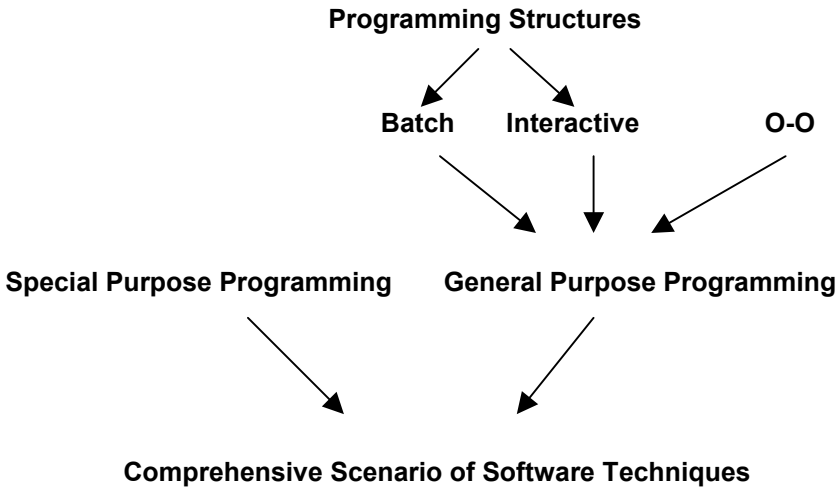
Small teams of specialists develop software products that demonstrate mathematical theorems, that manipulate linguistic texts, control the satellite, simulate an explosion, calculate broad formulas etc. These software products foster keen interests in scientists but make a minority group. They are far less important from the business viewpoints, even though they are very innovative and challenging in point of sciences.

This complete scenario produces several effects. We merely plunge into the educational and cultural corollaries.
The special-purpose tuition follows a special path. The student may settle for a brief study on computing due to his restricted responsibility. On the contrary, professionals who operate for institutions, companies, administrations etc. are to gain the complete comprehension of the software technologies. General-purpose applications involve a variety of users and customers, and entail the ample culture

necessary to cooperate with them. The contents of this chapter constitute the prized basis for further progress.

Unit 5 SUMMARY

We have introduced the batch and the interactive programs. Later we have commented some features of objects-oriented programming.



These techniques carry on the general-purpose software production. The complete scenario including special products closes this unit.