Module 5

Function-Oriented Software Design

Lesson 12

Structured Design

Specific Instructional Objectives

At the end of this lesson the student will be able to:

- Identify the aim of structured design.
- Explain what a structure chart is.
- Differentiate between a structure chart and a flow chart.
- Identify the activities carried out during transform analysis with examples.
- Explain what is meant by transaction analysis.

Structured Design

The aim of structured design is to transform the results of the structured analysis (i.e. a DFD representation) into a structure chart. Structured design provides two strategies to guide transformation of a DFD into a structure chart.

- Transform analysis
- Transaction analysis

Normally, one starts with the level 1 DFD, transforms it into module representation using either the transform or the transaction analysis and then proceeds towards the lower-level DFDs. At each level of transformation, it is important to first determine whether the transform or the transaction analysis is applicable to a particular DFD. These are discussed in the subsequent sub-sections.

Structure Chart

A structure chart represents the software architecture, i.e. the various modules making up the system, the dependency (which module calls which other modules), and the parameters that are passed among the different modules. Hence, the structure chart representation can be easily implemented using some programming language. Since the main focus in a structure chart representation is on the module structure of the software and the interactions among different modules, the procedural aspects (e.g. how a particular functionality is achieved) are not represented.

The basic building blocks which are used to design structure charts are the following:

- **Rectangular boxes:** Represents a module.
- **Module invocation arrows:** Control is passed from one module to another module in the direction of the connecting arrow.

- **Data flow arrows:** Arrows are annotated with data name; named data passes from one module to another module in the direction of the arrow.
- **Library modules:** Represented by a rectangle with double edges.
- **Selection:** Represented by a diamond symbol.
- **Repetition:** Represented by a loop around the control flow arrow.

Structure Chart vs. Flow Chart

We are all familiar with the flow chart representation of a program. Flow chart is a convenient technique to represent the flow of control in a program. A structure chart differs from a flow chart in three principal ways:

- It is usually difficult to identify the different modules of the software from its flow chart representation.
- Data interchange among different modules is not represented in a flow chart.
- Sequential ordering of tasks inherent in a flow chart is suppressed in a structure chart.

Transform Analysis

Transform analysis identifies the primary functional components (modules) and the high level inputs and outputs for these components. The first step in transform analysis is to divide the DFD into 3 types of parts:

- Input
- Logical processing
- Output

The input portion of the DFD includes processes that transform input data from physical (e.g. character from terminal) to logical forms (e.g. internal tables, lists, etc.). Each input portion is called an afferent branch.

The output portion of a DFD transforms output data from logical to physical form. Each output portion is called an efferent branch. The remaining portion of a DFD is called the central transform.

In the next step of transform analysis, the structure chart is derived by drawing one functional component for the central transform, and the afferent and efferent

branches. These are drawn below a root module, which would invoke these modules.

Identifying the highest level input and output transforms requires experience and skill. One possible approach is to trace the inputs until a bubble is found whose output cannot be deduced from its inputs alone. Processes which validate input or add information to them are not central transforms. Processes which sort input or filter data from it are. The first level structure chart is produced by representing each input and output unit as boxes and each central transform as a single box.

In the third step of transform analysis, the structure chart is refined by adding sub-functions required by each of the high-level functional components. Many levels of functional components may be added. This process of breaking functional components into subcomponents is called factoring. Factoring includes adding read and write modules, error-handling modules, initialization and termination process, identifying customer modules, etc. The factoring process is continued until all bubbles in the DFD are represented in the structure chart.

Example: Structure chart for the RMS software

For this example, the context diagram was drawn earlier.

To draw the level 1 DFD (fig. 5.11), from a cursory analysis of the problem description, we can see that there are four basic functions that the system needs to perform – accept the input numbers from the user, validate the numbers, calculate the root mean square of the input numbers and, then display the result.

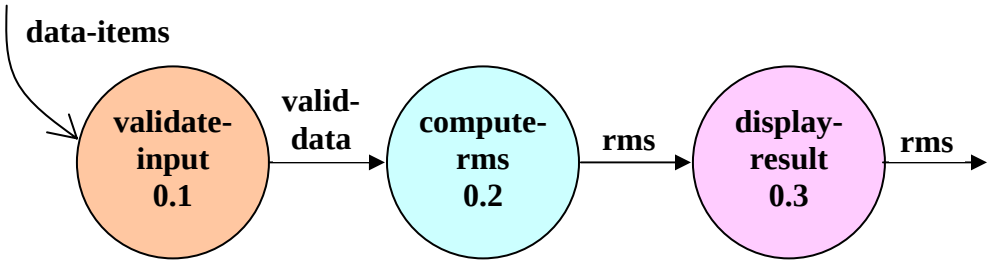

Fig. 5.11: Level 1 DFD

By observing the level 1 DFD, we identify the validate-input as the afferent branch and write-output as the efferent branch. The remaining portion (i.e. compute-rms) forms the central transform. By applying the step 2 and step 3 of transform analysis, we get the structure chart shown in fig. 5.12.

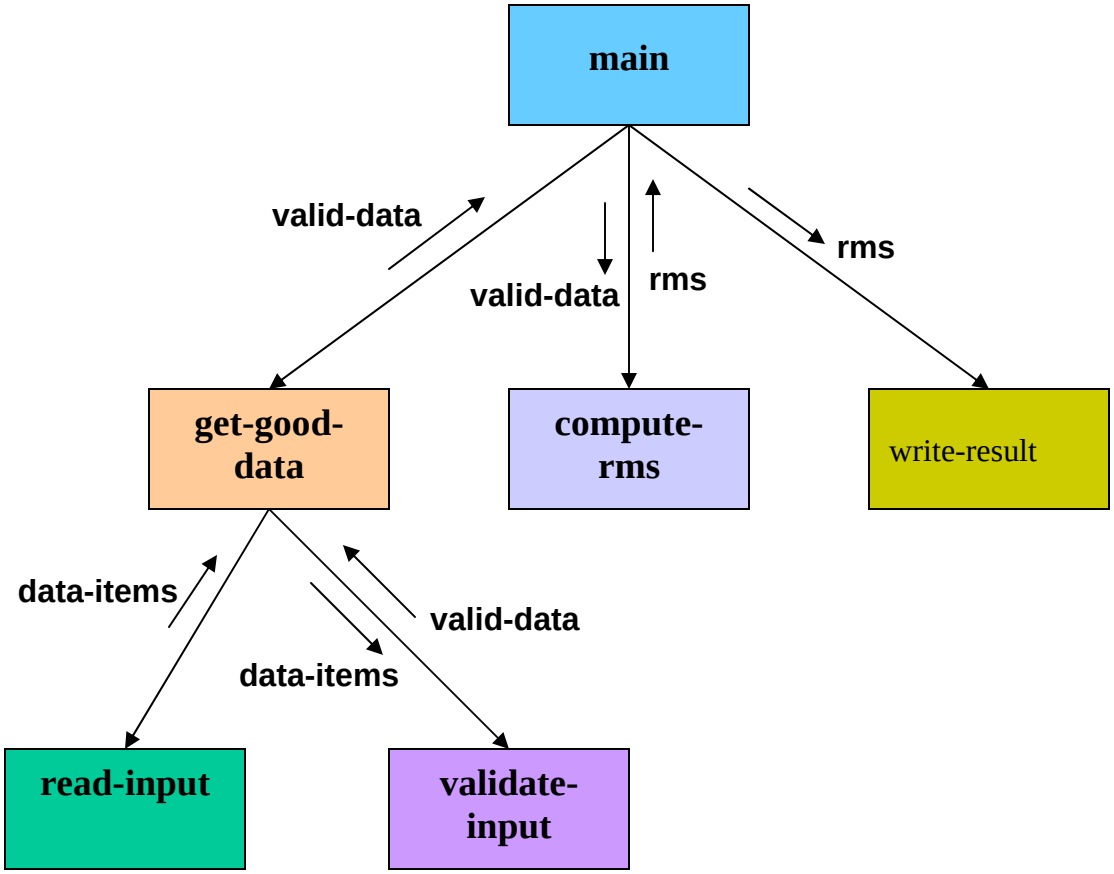

Fig. 5.12: *Structure chart*

Transaction Analysis

A transaction allows the user to perform some meaningful piece of work. Transaction analysis is useful while designing transaction processing programs. In a transaction-driven system, one of several possible paths through the DFD is traversed depending upon the input data item. This is in contrast to a transform centered system which is characterized by similar processing steps for each data item. Each different way in which input data is handled is a transaction. A simple way to identify a transaction is to check the input data. The number of bubbles on which the input data to the DFD are incident defines the number of transactions. However, some transaction may not require any input data. These transactions can be identified from the experience of solving a large number of examples.

For each identified transaction, trace the input data to the output. All the traversed bubbles belong to the transaction. These bubbles should be mapped to the same module on the structure chart. In the structure chart, draw a root

module and below this module draw each identified transaction a module. Every transaction carries a tag, which identifies its type. Transaction analysis uses this tag to divide the system into transaction modules and a transaction-center module.

The structure chart for the supermarket prize scheme software is shown in fig. 5.13.

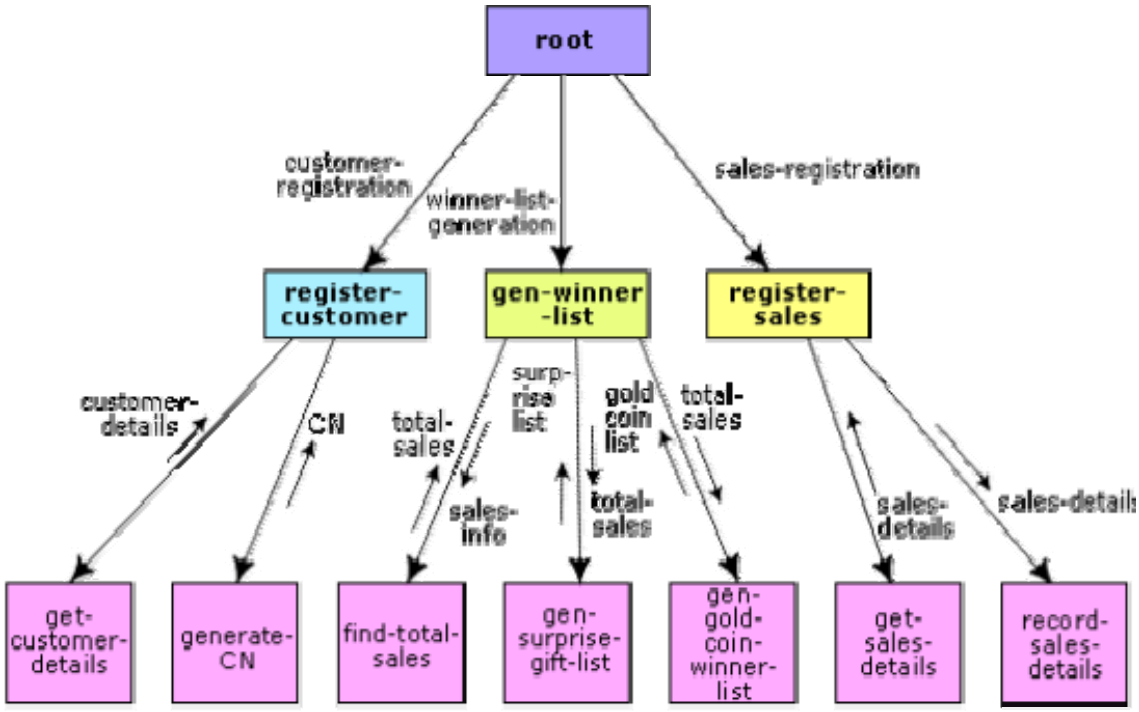

Fig. 5.13: Structure chart for the supermarket prize scheme

The following questions have been designed to test the objectives identified for this module:

1. Identify the aim of the structured analysis activity. Which documents are produced at the end of structured analysis activity?

Ans.: - The aim of the structured analysis activity is to transform a textual problem description into a graphic model. Structured analysis is used to carry out the top-down decomposition of the set of high-level functions depicted in the problem description and to represent them graphically. During structured analysis, functional decomposition of the system is achieved. That is, each function that the system performs is analyzed and hierarchically decomposed into more detailed functions.

During structured analysis, the major processing tasks (functions) of the system are analyzed, and the data flow among those processing tasks is represented graphically. Structured analysis technique is based on the following essential underlying principles**:**

- Top-down decomposition approach.
- Divide and conquer principle. Each function is decomposed independently.
- Graphical representation of the analysis results using Data Flow Diagrams (DFDs).

2. Identify the necessity of constructing DFDs in the context of a good software design.

Ans.: - Data Flow Diagram (DFD) is a very simple formalism. It is simple to understand and use. Starting with a set of high-level functions that a system performs, a DFD model hierarchically represents various sub-functions. The data flow diagramming technique follows a very simple set of intuitive concepts and rules. DFD is an elegant modeling technique that turns out to be useful not only to represent the results of structured analysis of a software problem but also useful for several other applications such as showing the flow of documents or items in an organization.

3. Write down the importance of data dictionary in the context of good software design.

Ans.: - A data dictionary plays a very important role in any software development process because of the following reasons:

- A data dictionary provides a standard terminology for all relevant data for use by the engineers working in a project. A consistent vocabulary for data items is very important, since in large projects, different engineers of the project have a tendency to use different terms to refer to the same data, which unnecessary causes confusion.
- The data dictionary provides the analyst with a means to determine the definition of different data structures in terms of their component elements.

4. What does the term "balancing a DFD" mean? Give an example to explain your answer.

Ans.: - The data that flow into or out of a bubble must match the data flow at the next level of DFD. This is known as balancing a DFD. The concept of balancing a DFD has been illustrated in fig. 5.2. In the level 1 of the DFD, data items d1 and d3 flow out of the bubble 0.1 and the data item d2 flows into the bubble P1. In the

next level, bubble 0.1 is decomposed. The decomposition is balanced, as d1 and d3 flow out of the level 2 diagram and d2 flows in.

5. Write down some essential activities required to develop the DFD of a system more systematically.

Ans.: - A DFD model of a system can be systematically developed in the following way:

1. The SRS document is examined to determine:
   - Different high-level functions that the system needs to perform.
   - Data input to every high-level function.
   - Data output from every high-level function.
   - Interactions (data flow) among the identified high-level functions.

   These aspects of the high-level functions are then represented in a diagrammatic form. This forms the top-level Data Flow Diagram (DFD), usually called the DFD 0.

2. The high-level functions described in the SRS document are examined. If there are between 3 to 7 high-level requirements in the SRS document, then each of the high-level function can be represented in the form of a bubble, if there are more than 7 bubbles, then some of them have to be combined. If there are less than 3 bubbles, then some of these have to be split.

3. Each high-level function is decomposed into its constituent sub-functions through the following set of activities:
   - Different sub-functions of the high-level function are identified.
   - Data input to each of these sub-functions are identified.
   - Data output from each of these sub-functions are identified.
   - Interactions (data flow) among these sub-functions are identified.

Step 3 is repeated recursively for each sub-function until a sub-function can be represented by using a simple algorithm.

6. What do you understand by top-down decomposition in the context of structured analysis? Explain your answer using a suitable example.

Ans.:- In the context of function-oriented design, top-down decomposition starts with the high-level functional requirements. Then it successively decomposes those high-level functions into more detailed functions.

7. Identify some commonly made errors while constructing of a DFD model.

Ans.:- The different types of mistakes that users usually make while constructing the DFD model of systems are as follows:

- Many beginners commit the mistake of drawing more than one bubble in the context diagram. A context diagram should depict the system as a single bubble.
- Many beginners have external entities appearing at all levels of DFDs. All external entities interacting with the system should be represented only in the context diagram. The external entities should not appear at other levels of the DFD.
- It is a common oversight to have either too less or too many bubbles in a DFD. Only 3 to 7 bubbles per diagram should be allowed, i.e. each bubble should be decomposed to between 3 and 7 bubbles.
- Many beginners leave different levels of DFD unbalanced.
- A common mistake committed by many beginners while developing a DFD model is attempting to represent control information in a DFD. It is important to realize that a DFD is the data flow representation of a system, and it does not represent control information. For an example mistake of this kind: click here.

  - Consider the following example. A book can be searched in the library catalog by inputting its name. If the book is available in the library, then the details of the book are displayed. If the book is not listed in the catalog, then an error message is generated. While generating the DFD model for this simple problem, many beginners commit the mistake of drawing an arrow (as shown in fig. 5.10) to indicate the error function is invoked after the search book. But, this is a control information and should not be shown on the DFD.

  - Another error is trying to represent when or in what order different functions (processes) are invoked and neither does it represent the conditions under which different functions are invoked.

  - If a bubble A invokes either the bubble B or the bubble C depending upon some conditions, we need only to represent the data that flows between bubbles A and B or bubbles A and C and not the conditions based on which the two modules are invoked.

- A data store should be connected only to bubbles through data arrows. A data store cannot be connected to another data store or to an external entity.
- All the functionalities of the system must be captured by the DFD model. No function of the system specified in its SRS document should be overlooked.
- Only those functions of the system specified in the SRS document should be represented, i.e. the designer should not assume

functionality of the system not specified by the SRS document and then try to represent them in the DFD.

- Improper or unsatisfactory data dictionary.
- The data and function names must be intuitive. Some students and even practicing engineers use symbolic data names such a, b, c, etc. Such names hinder understanding the DFD model.

8. Identify some important shortcomings of the DFD model.

Ans.: - DFD models suffer from several shortcomings. The important shortcomings of the DFD models are the following:

- DFDs leave ample scope to be imprecise. In the DFD model, the function performed by a bubble is judged from its label. However, a short label may not capture the entire functionality of a bubble. For example, a bubble named find-book-position has only intuitive meaning and does not specify several things, e.g. what happens when some input information are missing or are incorrect. Further, the find-book-position bubble may not convey anything regarding what happens when the required book is missing.
- Control aspects are not defined by a DFD. For instance, the order in which inputs are consumed and outputs are produced by a bubble is not specified. A DFD model does not specify the order in which the different bubbles are executed. Representation of such aspects is very important for modeling real-time systems.
- The method of carrying out decomposition to arrive at the successive levels and the ultimate level to which decomposition is carried out are highly subjective and depend on the choice and judgment of the analyst. Due to this reason, even for the same problem, several alternative DFD representations are possible. Further, many times it is not possible to say which DFD representation is superior or preferable to another one.
- The data flow diagramming technique does not provide any specific guidance as to how exactly to decompose a given function into its sub-functions and we have to use subjective judgment to carry out decomposition.

9. Differentiate between a structure chart and a flow chart.

Ans.: - A structure chart differs from a flow chart in three principal ways**:**

- It is usually difficult to identify the different modules of the software from its flow chart representation.

- Data interchange among different modules is not represented in a flow chart.

- Sequential ordering of tasks inherent in a flow chart is suppressed in a structure chart.

For the following, mark all options which are true.

1. The purpose of structured analysis is

- □ to capture the detailed structure of the system as perceived by the user √
- □ to define the structure of the solution that is suitable for implementation in some programming language
- □ all of the above

2. Structured analysis technique is based on

- □ top-down decomposition approach √
- □ bottom-up approach
- □ divide and conquer principle √
- □ none of the above

3. Data Flow Diagram (DFD) is also known as a:

- □ structure chart
- □ bubble chart √
- □ Gantt chart
- □ PERT chart

4. The context diagram of a DFD is also known as

- □ level 0 DFD √
- □ level 1 DFD
- □ level 2 DFD
- □ none of the above

5. Decomposition of a bubble is also known as

- □ classification
- □ factoring √
- □ exploding √
- □ aggregation

6. Decomposition of a bubble should be carried on

- □ till the atomic program instructions are reached
- □ upto two levels
- □ until a level is reached at which the function of the bubble can be described using a simple algorithm √
- □ none of the above

7. The bubbles in a level 1 DFD represent

- exactly one high-level functional requirement described in SRS document
- more than one high-level functional requirement
- part of a high-level functional requirement
- any of the above depending on the problem √

8. By looking at the structure chart, we can

- say whether a module calls another module just once or many times
- not say whether a module calls another module just once or many times √
- tell the order in which the different modules are invoked
- not tell the order in which the different modules are invoked √

9. In a structure chart, a module represented by a rectangle with double edges is called

- root module
- library module √
- primary module
- none of the above

10. A structure chart differs from a flow chart in which of the following ways

- it is always difficult to identify the different modules of the software from its flow chart representation √
- data interchange among different modules is not presented in a flow chart √
- sequential ordering of tasks inherent in a flow chart is suppressed in a structure chart √
- none of the above

11. The input portion in the DFD that transform input data from physical to logical form is called

- central transform
- efferent branch
- afferent branch √
- none of the above

12. If during structured design you observe that the data entering a DFD are incident on different bubbles, then you would use:

- transform analysis
- transaction analysis √
- combination of transform and transaction analysis

- neither transform nor transaction analysis

13. During structured design, if all the data flow into the diagram are processed in similar ways i.e. if all the input data are incident on the same bubble in the DFD, the one have to use:

- transform analysis √
- transaction analysis
- combination of transform and transaction analysis
- neither transform nor transaction analysis

14. Which of the following types of bubbles may belong to the central transform ?

- input validation
- adding information to the input
- sorting input √
- filtering data √

15. During detailed design which of the following activities take place?

- the pseudo code for the different modules of the structure chart are developed in the form of MSPECs √
- data structures are designed for the different modules of the structure chart √
- module structure is designed
- none of the above

Mark the following as either True or False. Justify your answer.

1. A DFD model of a system represents the functions performed by the system and the data flow taking place among these functions.

Ans.: - True

Explanation: - A DFD in simple words, is a hierarchical graphical model of a system that shows the different processing activities or functions that the system performs and the data interchange among these functions.

2. A data dictionary lists the purpose of all data items and the definition of all composite data items in terms of their component data items.

Ans.: - True.

Explanation: - A data dictionary lists the purpose of all data items and the definition of all composite data items in terms of their component data items. For example, a data dictionary entry may represent that the data **grossPay** consists of the components **regularPay** and **overtimePay**.

grossPay = regularPay + overtimePay

3. The context diagram of a system represents it using more than one bubble.

Ans.: - False.

Explanation: - The context diagram is the most abstract data flow representation of a system. It represents the entire system as a single bubble. This bubble is labeled according to the main function of the system. The various external entities with which the system interacts and the data flow occurring between the system and the external entities are also represented.

4. External entities may appear at all levels of DFDs.

Ans.: - False.

Explanation: - All external entities interfacing with the system should be represented only in the context diagram. The external entities should not appear at other levels of the DFD.

5. A DFD captures the order in which the processes (bubbles) operate.

Ans.: - False.

Explanation: - A DFD does not capture the order in which the processes (bubbles) operate.

6. DFDs enable a software engineer to develop the data domain and functional domain decomposition of the system at the same time.

Ans.: - True.

Explanation: - As the DFD is refined into greater levels of detail, the analyst performs an implicit functional decomposition. At the same time, the DFD refinement automatically results in refinement of corresponding data items.

7. There should be at most one control relationship between any two modules in a properly designed structure chart.

Ans.: - True.

Explanation: - It can be considered the different modules of a structure chart to be arranged in layers or levels. The principle of abstraction does not allow lower-level modules to be aware of the existence of the high-level modules. However, it is possible for two higher-level modules to invoke the same lower-level module.