Learning Resource

On

Software Engineering

Chapter-6

Software Design: Function Oriented Design

Prepared By:

Kunal Anand, Asst. Professor

SCE, KIIT, DU, Bhubaneswar-24

Chapter Outcomes:

After completing this chapter successfully, the students will be able to:

- Identify the goal of function oriented design.
- Distinguish between structured analysis and structured design.
- Explain data flow diagram (DFD) and its types.
- List the components of DFD.
- List commonly made errors while drawing DFD.
- Draw DFD for a given problem statement.
- Discuss the shortcomings of DFD model.
- Explain structured design
- Draw structure chart for a given DFD.
- Discuss detailed design

Organization of the Chapter

- Introduction to Function oriented design
- Structured Analysis and Structured Design
- Data Flow Diagram
- Structure Chart
- Examples
- Detailed Design
- Design Review

Introduction to Function Oriented Design

- These techniques are very popular and are being widely used in software development process.
- Here, the software product is viewed as a "**Black box**", that performs a set of high-level functions.
- During design process, the high-level functions are decomposed into individual modules, with **high cohesion and low coupling**, that can be implemented using any suitable programming language.
- Broadly, the design techniques are:
  - **Structured Analysis**
  - **Structured Design**

SA/SD Technique

- **Structured Analysis** transforms a **textual problem** description into a **graphical representation** by performing **top-down clean decomposition in neat arrangement** manner.
- **Structured Design** maps the identified functions to a module structure that is also known as **"Software Architecture"**.
  - This is performed on each module in order to get the detailed design.

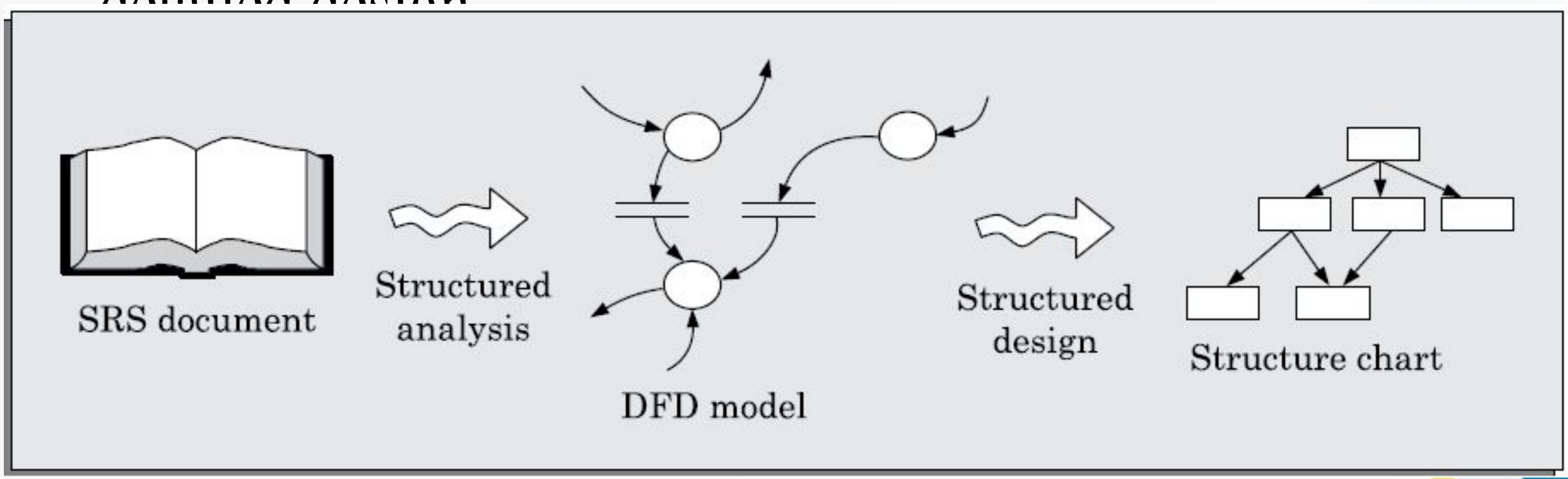


Fig. 6.1: SA and SD Methodology

Data Flow Diagram (DFD)

- DFD, also known as **Bubble Chart**, is a graphical representation of a system that shows different **functions** performed by the system, and the **data interchange** among these functions.
- DFD depicts **incoming data** flow, **outgoing data** flow and **stored data**.
- Here, each function of the system is considered as a **process** that takes some **input data** and produces some **output data** by performing some **operation** on them.
- **Difference between DFD and Flowchart**
  - The flowchart depicts **flow of control** in program modules whereas, DFDs depict **flow of data** in the system at various levels.
  - DFD does not contain any control or branch elements.

DFD Components

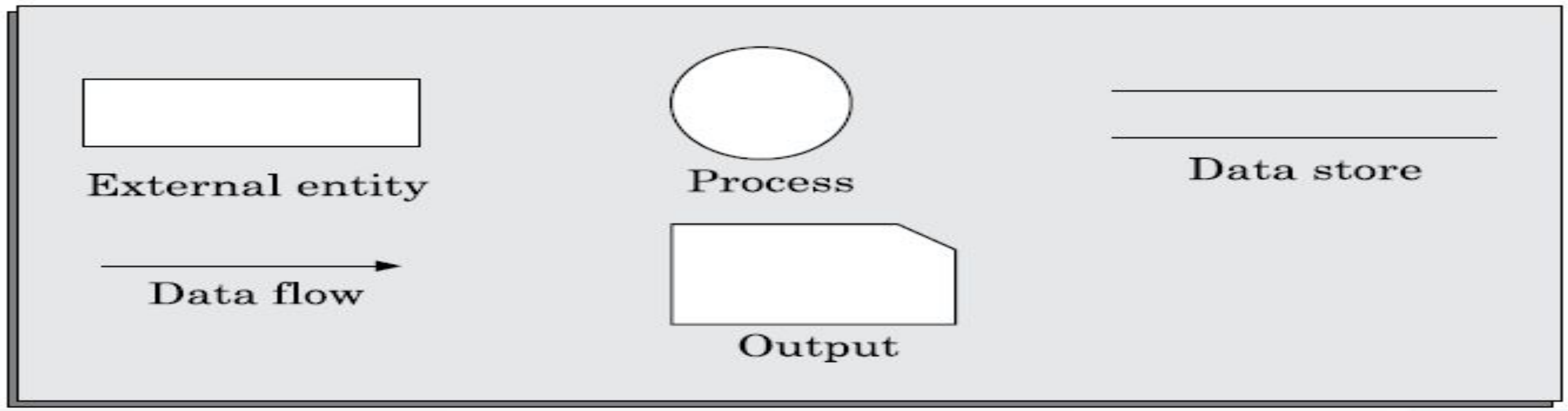


Fig. 6.2: Symbols used for DFDs

- **Entities** - Entities are source and destination of information data.
- **Process** - Activities and action taken on the data are represented by Circle or Round-edged rectangles. It represents a function.
- **Data Storage** - There are two variants of data storage - it can either be represented as a rectangle with absence of both smaller sides or as an open-sided rectangle with only one side missing.
- **Data Flow** - Movement of data is shown by pointed arrows.
  – Data movement is shown from the base of arrow as its source towards head of the arrow as destination.

Levels of DFD

- **Level 0** - Highest abstraction level DFD is known as **Level 0 DFD**, also known as **Context Diagram**.
- Here, the entire system is represented as a **single functionality** i.e., context diagram will have **only one bubble**. This bubble is named as the **name of the software**, and it is leveled as **Level 0.**
- The various **entities** associated with the system is also represented in the context diagram only. The entity may be **any user** or **any external system** that may provide some input or may consume the output.
- The context diagram **conceals** any information about the functionality of the system.

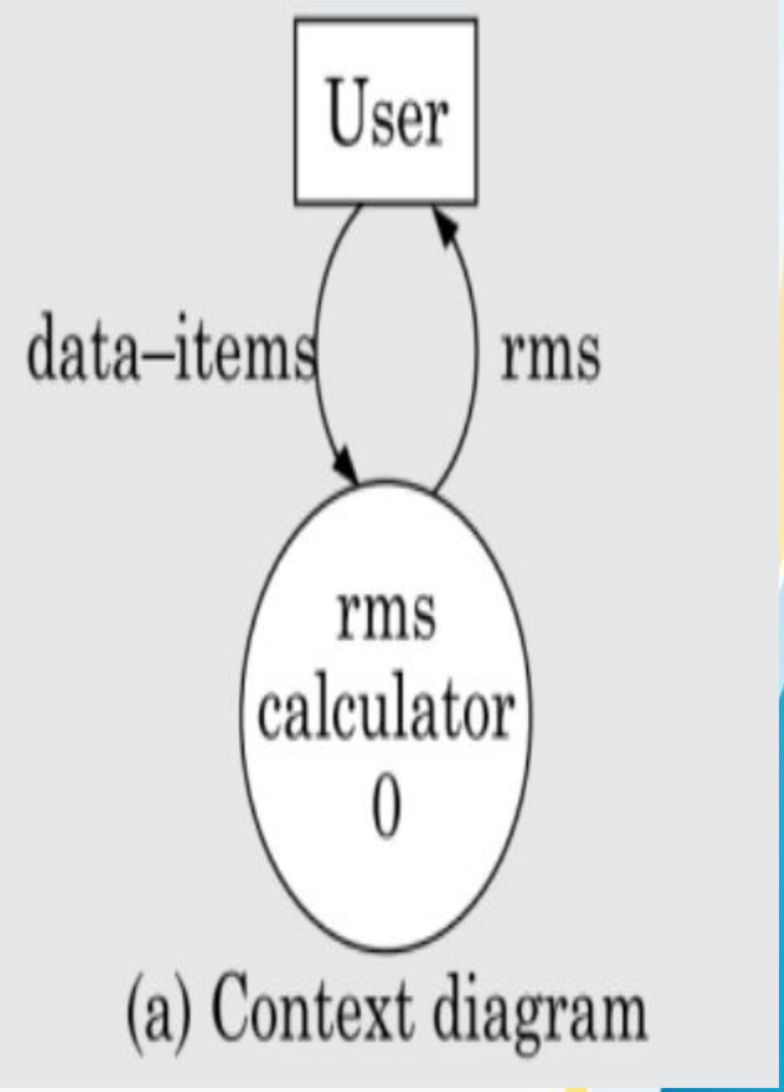
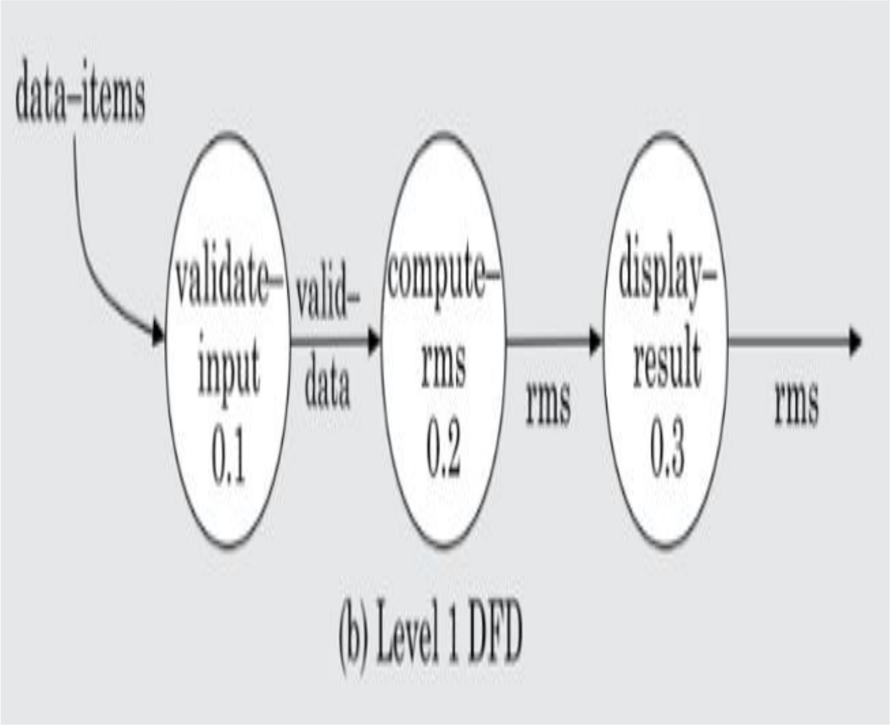


Fig. 6.3: Level 0 DFD

Level-1 and Level-2 DFD

- **Level-1:** Level 1 DFD depicts **basic modules** in the system and **flow of data** among various modules.

- **Level-2:** At this level, DFD shows how data flows **inside the modules** mentioned in Level 1.



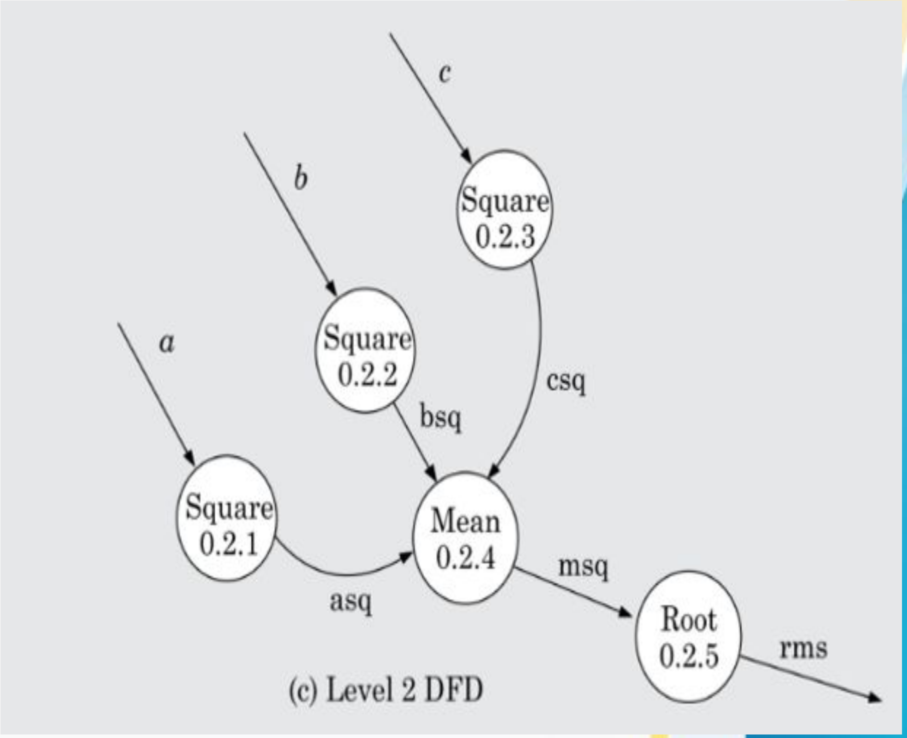


Fig. 6.4: Level 1 and Level 2 DFD

Decomposition and Numbering

- Each **bubble** in a DFD represents a **function**. the bubbles are decomposed into sub functions at the successive levels of the DFD.
- The **decomposition** of a bubble is also known as **factoring or expansion**.
- A bubble at any level of DFD is usually decomposed between **3 to 7 bubbles**. This process continues till we get the bubbles that can be described using a simple algorithm.
- **Numbering** of the DFD is very important. The **context diagram** is numbered as **"0"** as it is level 0 DFD.
- When a **bubble(x)** is decomposed, the sub functions are numbered as **x.1, x.2, x.3** etc. in level-1 DFD.
- In level-2 DFD, When a **bubble(x.y)** is decomposed, the sub functions are numbered as **x.y.1, x.y.2, x.y.3** etc.

Points to be noted in DFD

- **Synchronous and Asynchronous Operation:**
  - If two bubbles are **directly related** by a data arrow, then they are **synchronous** in nature. This means that they operate at the **same speed.**
  - In case of **asynchronous** operation, the bubbles are **connected to a data store** which means that the operating bubbles operate at **different speed**.

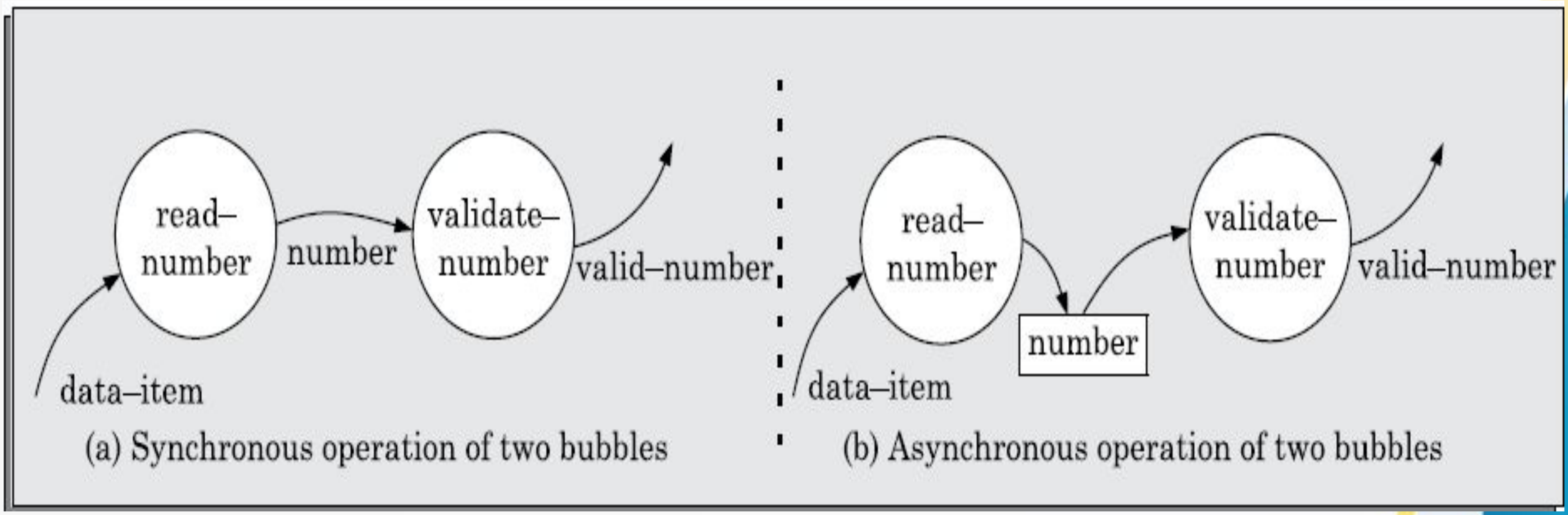


Fig. 6.5: Synchronous and Asynchronous data flow

Important points (contd..)

- **Data Dictionary:** Every DFD model must be accompanied by a data dictionary.
  - A data dictionary lists **all items** appearing in the DFD model of a system.
  - It includes **all data flows** and the content of **all data stores** appearing in all DFDs of a DFD model.
  - A **DFD model** may consists of several DFD like **level 0, level 1, level 2** etc. However, **only one data dictionary** exists for the entire DFD model.
  - Data dictionary provides better **consistency** due to standard naming convention used.
  - For the **smallest units** of data items, the data dictionary simply lists their **name and type**.

Important points (contd..)

- **Data Definition:** Composite data items can be defined using primitive data items with the help of following operators:
  - "+" : denotes **composition** of two data items. e.g., a+b represents data "a" and "b".
  - **[, ,]** : It represents **selection** i.e., any one data item listed inside the bracket. e.g., [a,b,c] means either a or b or c.
  - **()** : The content inside the () is **optional**. e.g., a+(b) means either a or a+b.
  - **{ }** : It represents **iterative** data definition. e.g., {name}5 means five name data.
  - **"="** : Represents **equivalence**. e.g., a=b+c means a represents b and c.
  - /*...*/ : Represents comments

Important points (contd..)

- **Balancing DFDs:** The data flow into or out of a bubble must match the data flow at the next level of the DFD for the said bubble.

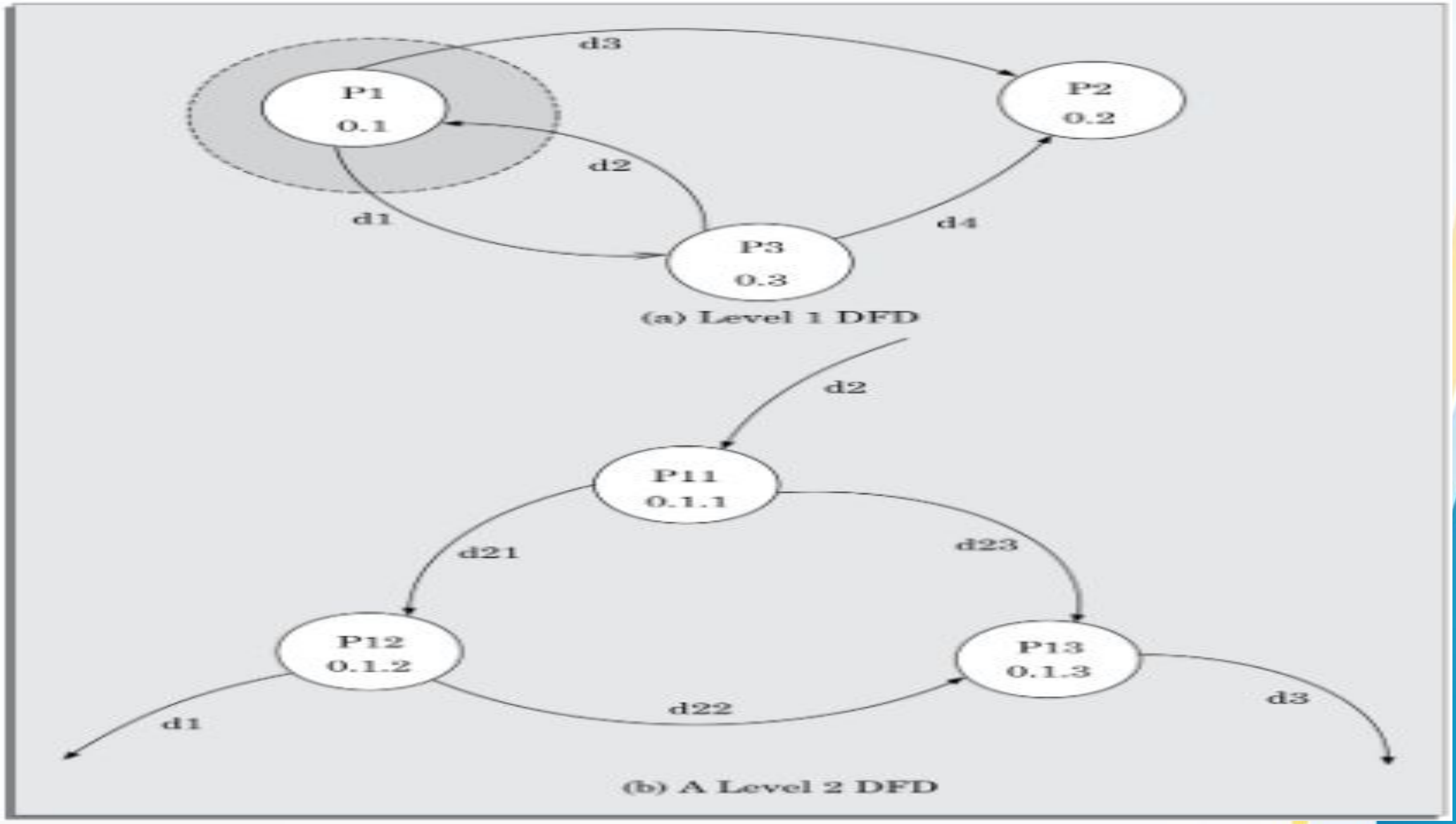


Fig. 6.6: Balancing of DFD

Guidelines For Constructing DFDs

- **Context diagram** should represent the system as a **single bubble**.
  - Many beginners commit the mistake of drawing **more than one bubble** in the context diagram.
- A DFD must be **balanced**.
- All **external entities** should be represented in the **context diagram** i.e., external entities **must not** appear at any other level of DFD.
- Only **3 to 7 bubbles** per diagram should be drawn.
- A common mistake committed by many beginners is to represent **control information** in a DFD.
  - Ex: trying to represent the order in which different functions are executed. **This must not be done**.

Commonly made errors

- Unbalanced DFDs
- Forgetting to mention the names of the data flows
- Unrepresented functions or data
- External entities appearing at higher level DFDs
- Trying to represent control aspects
- Context diagram having more than one bubble
- A bubble decomposed into too many bubbles in the next level
- Terminating decomposition too early
- Nouns used in naming bubbles

Example-1: RMS Calculator

- **Problem Statement:** A RMS Calculating software reads three integers from the users in the range of -1000 and +1000 and would determine the “Root Mean Square” of the given three input numbers and display it.
- **Context Diagram**

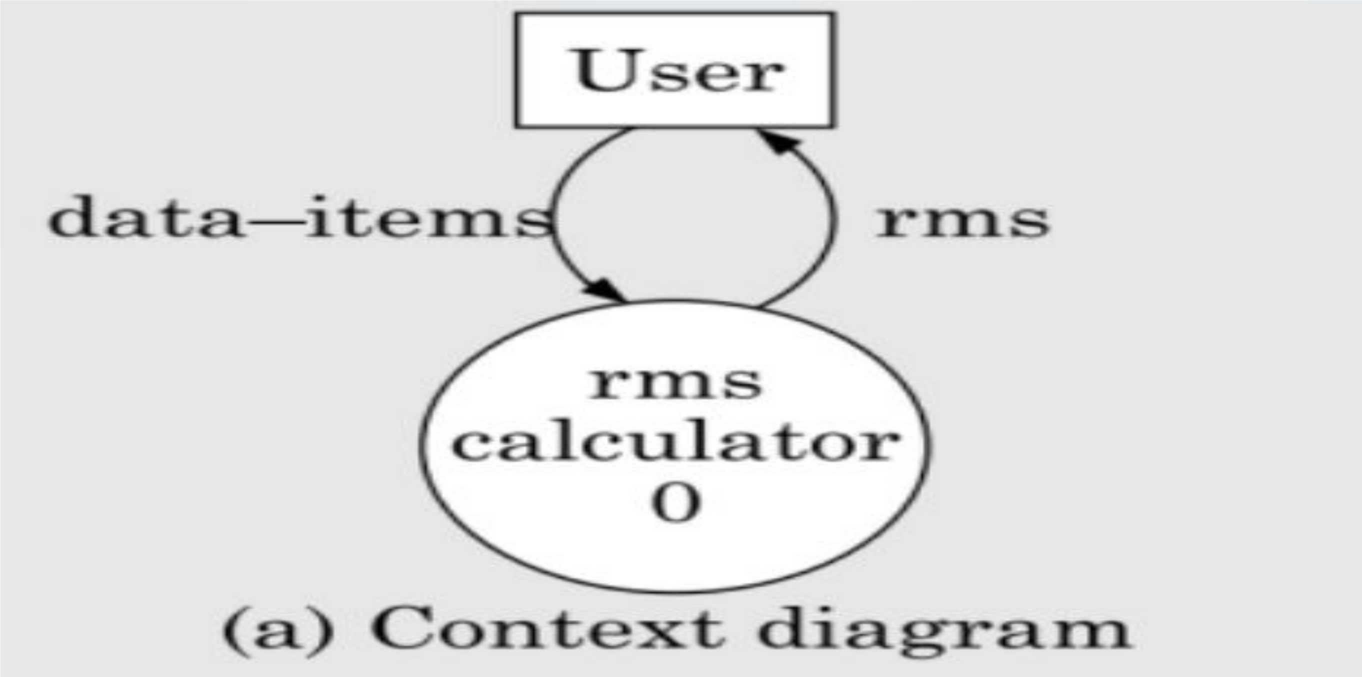


Fig. 6.7: Level 0 DFD for RMS calculator

contd..

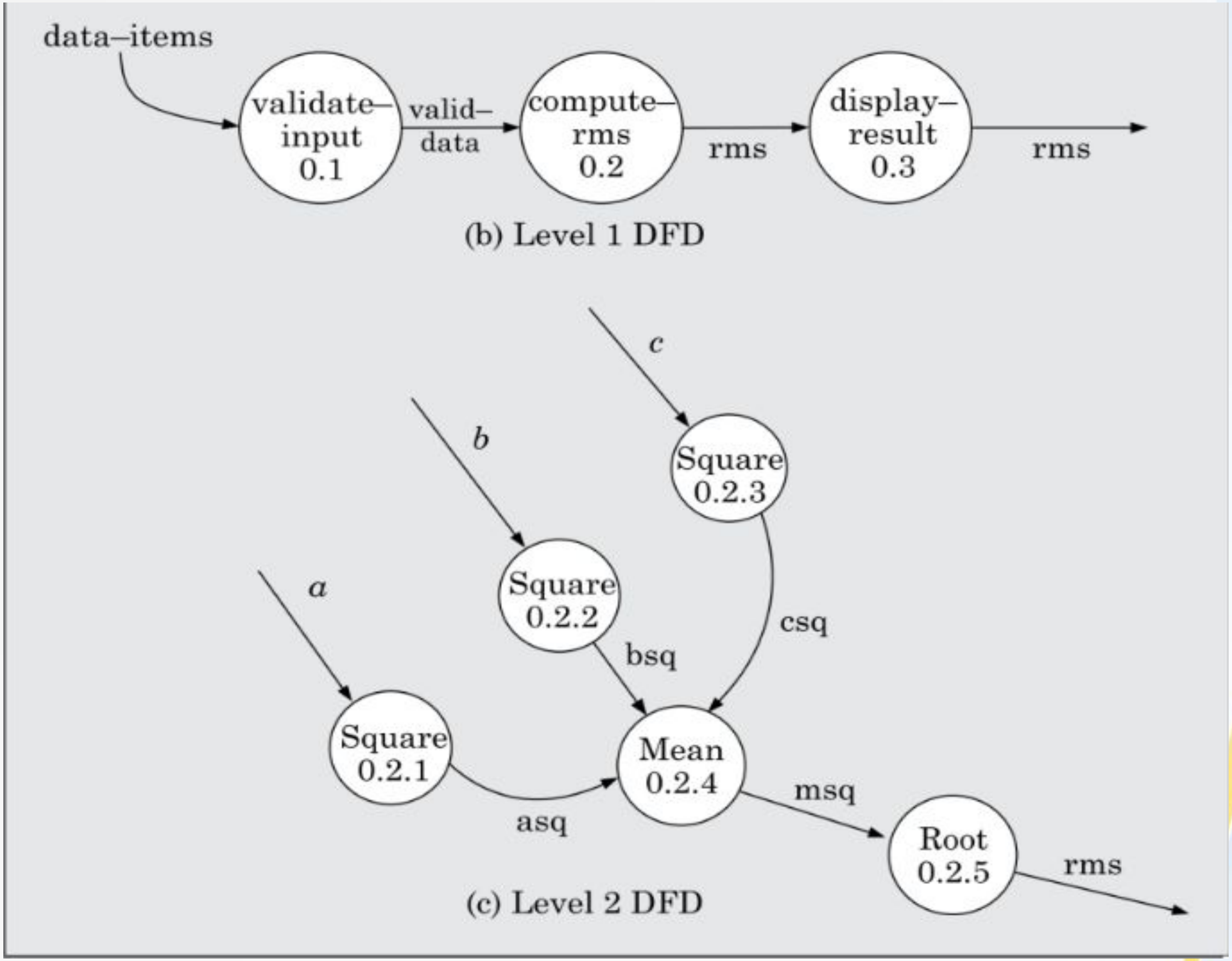


Fig. 6.8: Level 1 and Level 2 DFD for RMS calculator

Data Dictionary for RMS Calculator

- data_item: {integer}3
- rms: float
- valid_data: data_item
- a: integer
- b: integer
- c: integer
- asq: integer
- bsq: integer
- csq: integer
- msq: integer

Example-2: Supermarket Prize Scheme

- **Problem Statement**
  - A supermarket needs to develop a software that would help it to develop a scheme to that it plans to introduce to encourage its regular customers.
  - In this scheme, a customer would have to first register themselves by providing their basic personal details, following which they will be allotted a unique "Customer Number" (CN).
  - A customer can produce his/her CN to the checkout staff whenever he/she makes the purchase. In this case, the value of his purchase is credited against his/her purchase.
  - At the end of each year, the supermarket intends to award the surprise gifts to the 10 customer who make the highest total. purchase. Also, it intends to award a 22-caret gold coin to every customer who purchase exceeded Rs. 10000.
  - The entries against the CN are reset at the last day of the year after declaration of the winners.

DFD for Supermarket Prize Scheme

Context Diagram

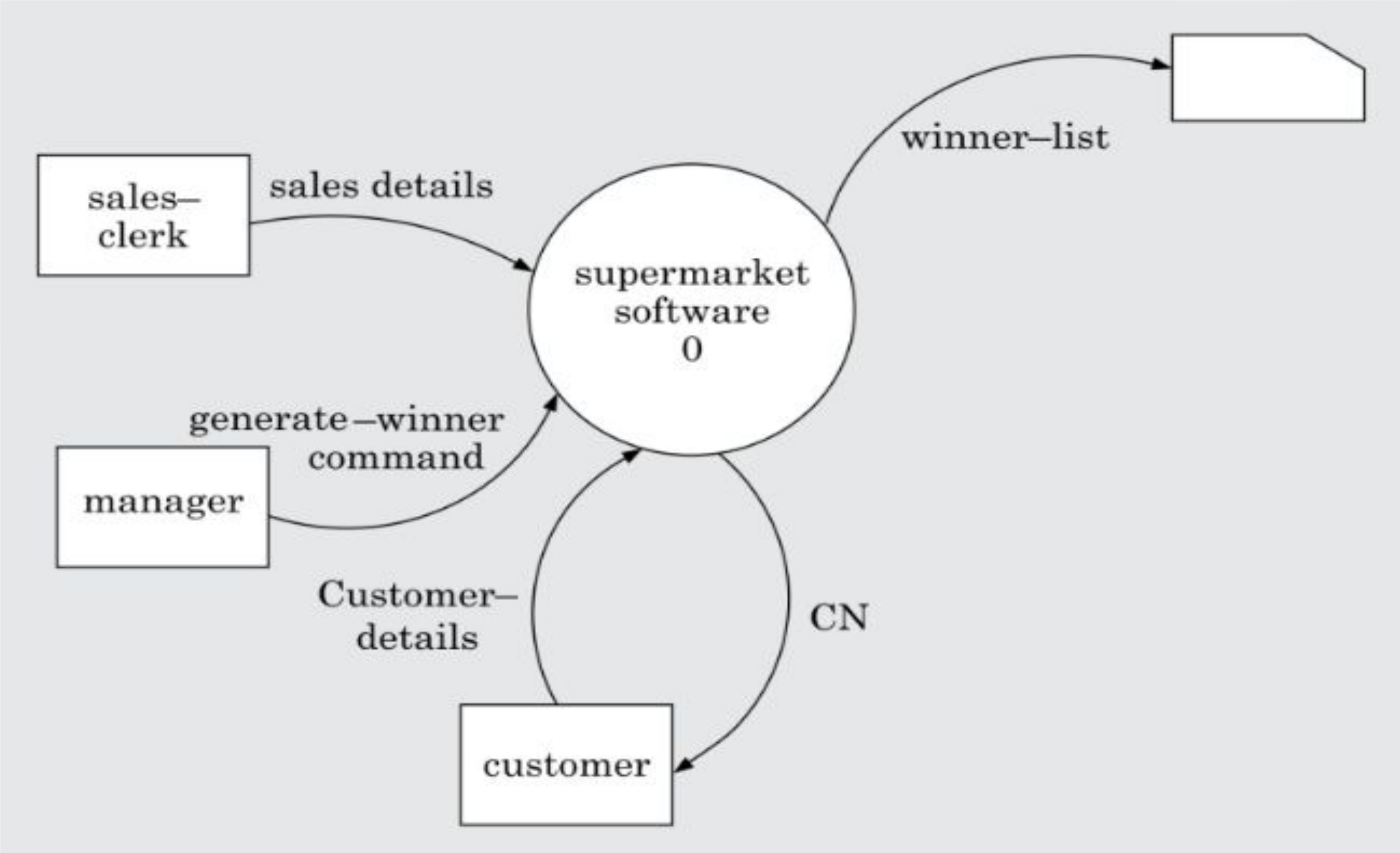


Fig. 6.9: Level 0 DFD for Supermarket problem

Level-1 DFD

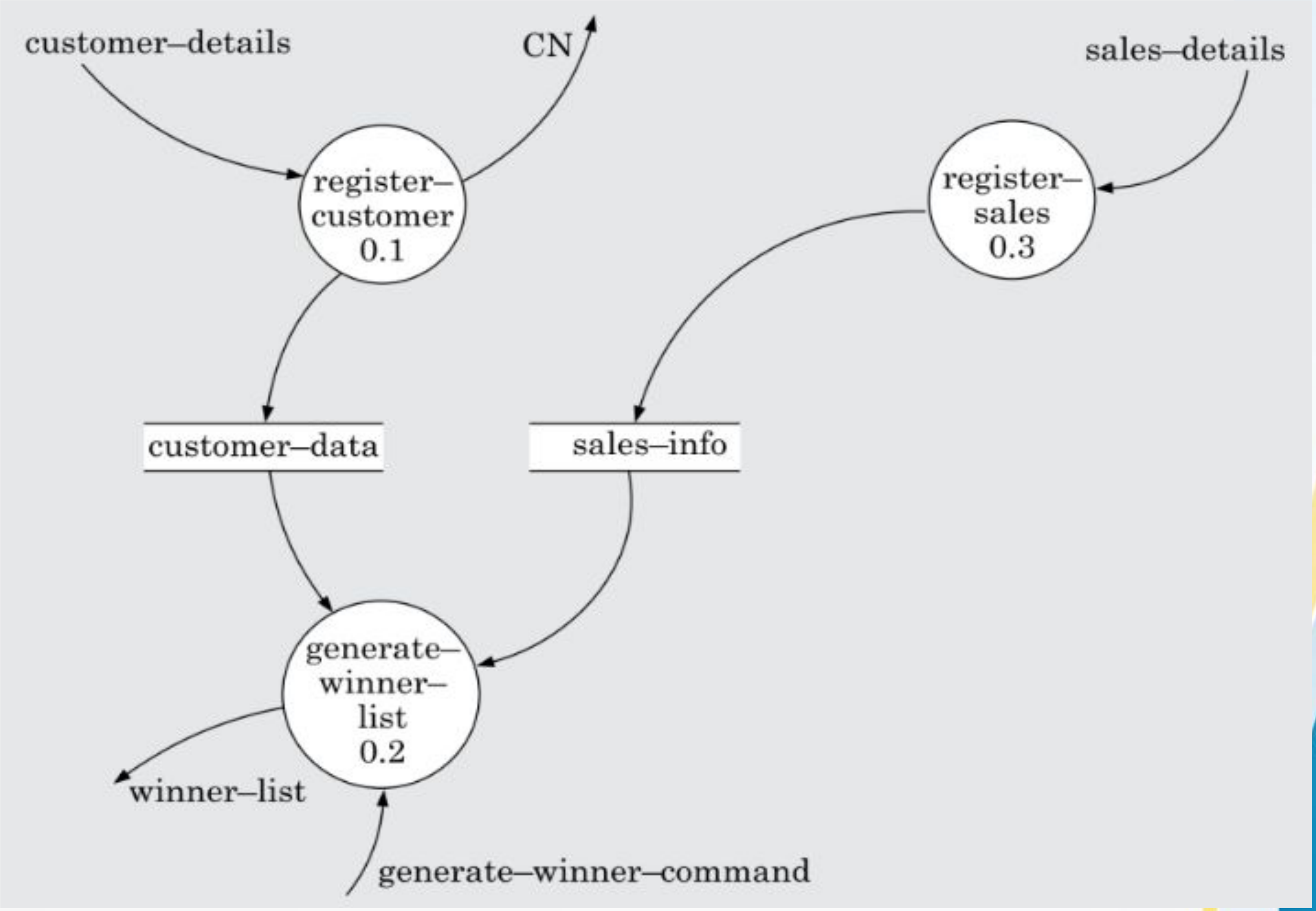


Fig. 6.10: Level 1 DFD for Supermarket problem

Level-2 DFD

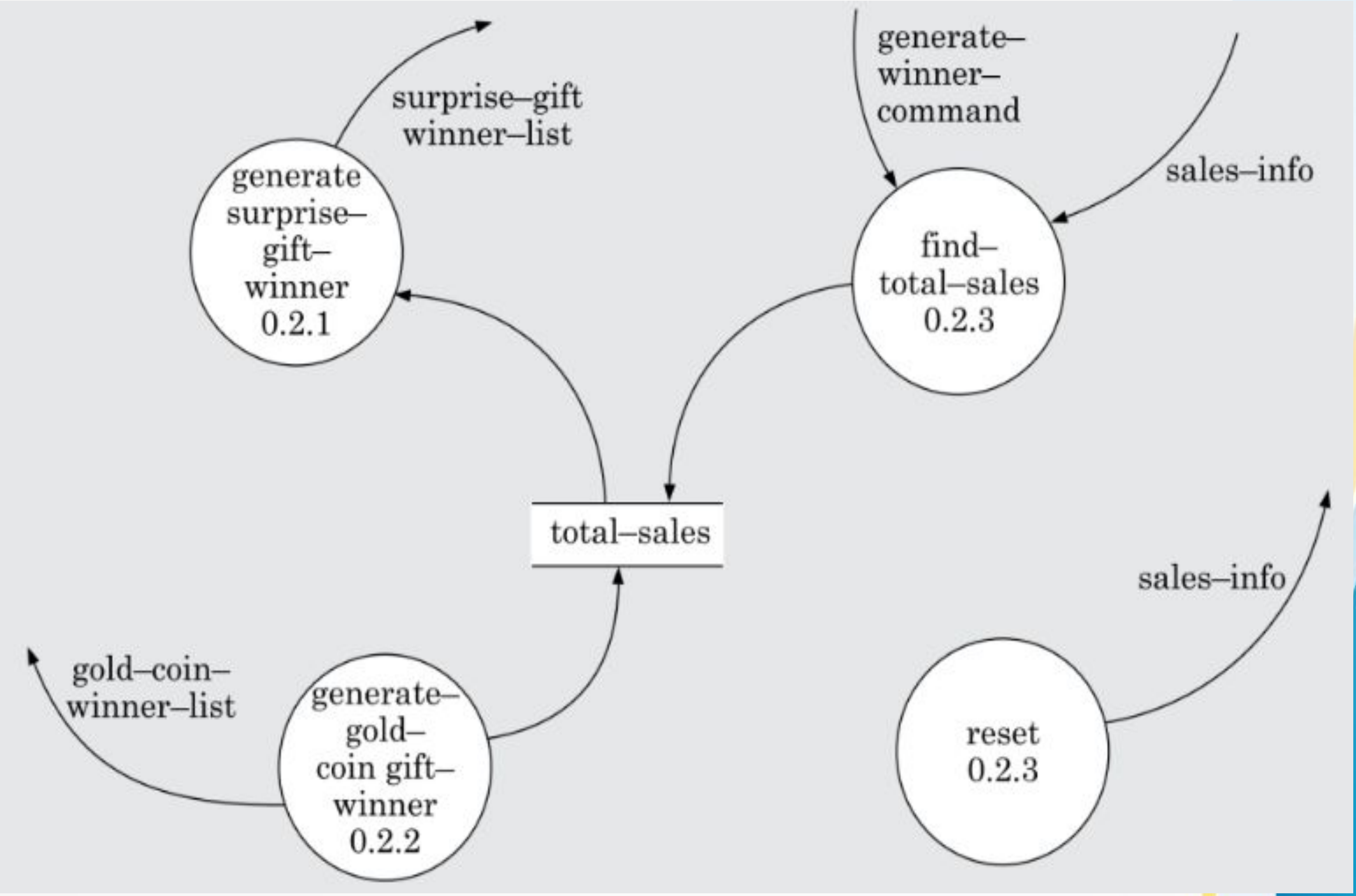


Fig. 6.11: Level 2 DFD for Supermarket problem

Data Dictionary

- address: name+house_no+street+city+pin
- sales_details: {item+amount}* + CN
- CN: integer
- cust_details: {address+CN}*
- sales_info: {sales_details}*
- winner_list: [surprise_gift+gold_coin]
- surprise_gift: {cust_details}*
- gold_coin: {cust_details}*
- gen_winner_command: command
- total_sales: {CN+integer}*

Note: Kindly go through solved **examples 6.4 and 6.5** at page no. **259-263** from given text book.

Shortcomings of the DFD Model

- DFD models suffer from several shortcomings:
  - DFDs leave ample scope to be **imprecise**. In a DFD model, we infer about the function performed by a bubble from its label. A label may not capture all the functionality of a bubble.
  - DFD model does not represent the **control information**, which might be needed while designing the algorithms.
  - The method of carrying out decomposition is **subjective** and largely depends upon the choice and judgement of an individual, due to which several DFD may be possible for the same problem.

Structured Design (SD)

- The main aim of structured design is to transform the results of structured analysis (i.e., a **DFD** representation) into a **“Structure Chart”**.

- A **structure chart** represents the **software architecture** which includes various **modules** making up the system, **module dependency** (i.e., which module calls which other modules), **parameters** passed among different modules.

- The **structure chart** is the desirable outcome as it can be easily implemented using a suitable programming language.

Structure Chart

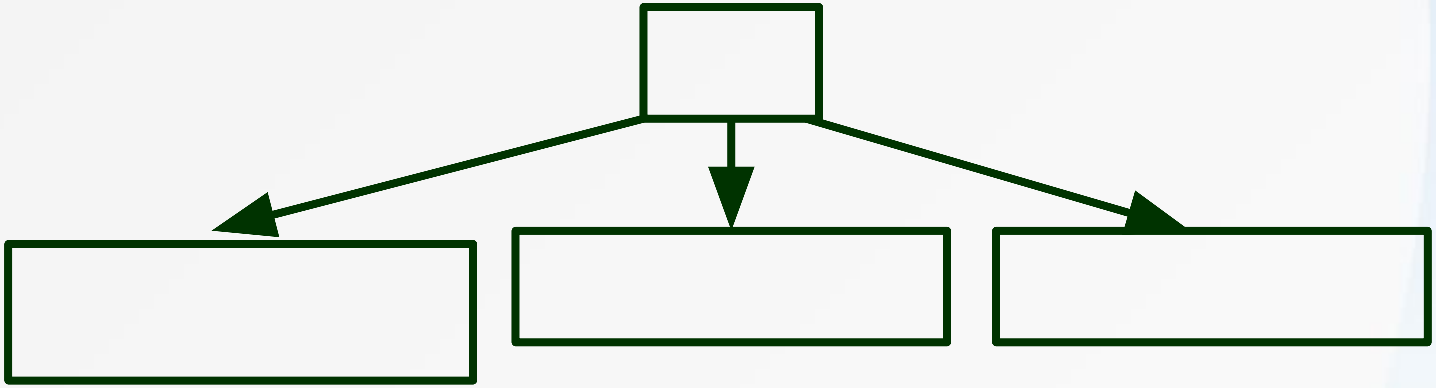
- There is only **one module** at the top i.e., the **root module.**
- There is at most **one control relationship** between any two modules i.e., if module A invokes module B, then module B cannot invoke module A.
  - The main reason behind this restriction is the **layered arrangement** in which the modules are arranged in layers or levels.
  - The principle of abstraction **does not allow lower-level modules to invoke higher-level modules**.
- However, two higher-level modules can invoke the same lower-level module.

Basic building blocks of structure chart

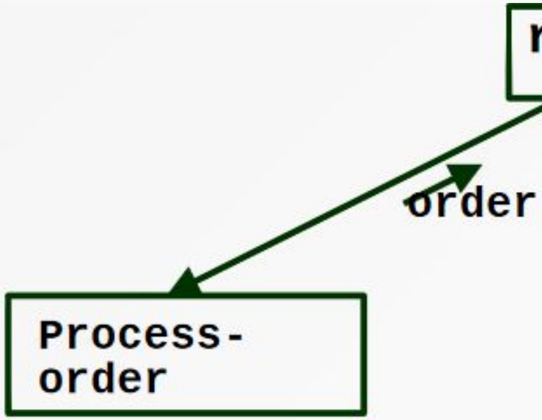
- **Rectangular box:**
  - A rectangular box represents a **module**.
  - It is annotated with the **name of the module** it represents.

- **Module Invocation Arrows:**
  - An **arrow** between two modules implies that during execution, **control is passed** from one module to the other in the direction of the arrow.
  - However, it does not tell that **how many times** a module has invoked another module, or in which **order** the execution took place.

contd..



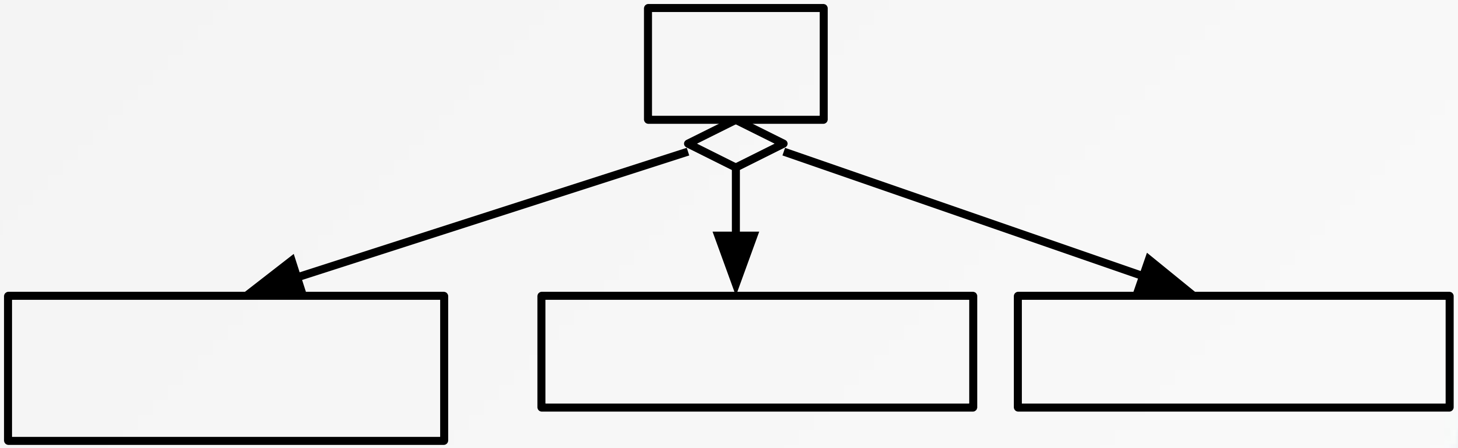
- **Data Flow Arrows:** These are the **small arrows** appearing along side the module invocation arrows. They are annotated with the corresponding **data name**. It implies that the **named data** passes from one module to another module in the **direction of the arrow**.



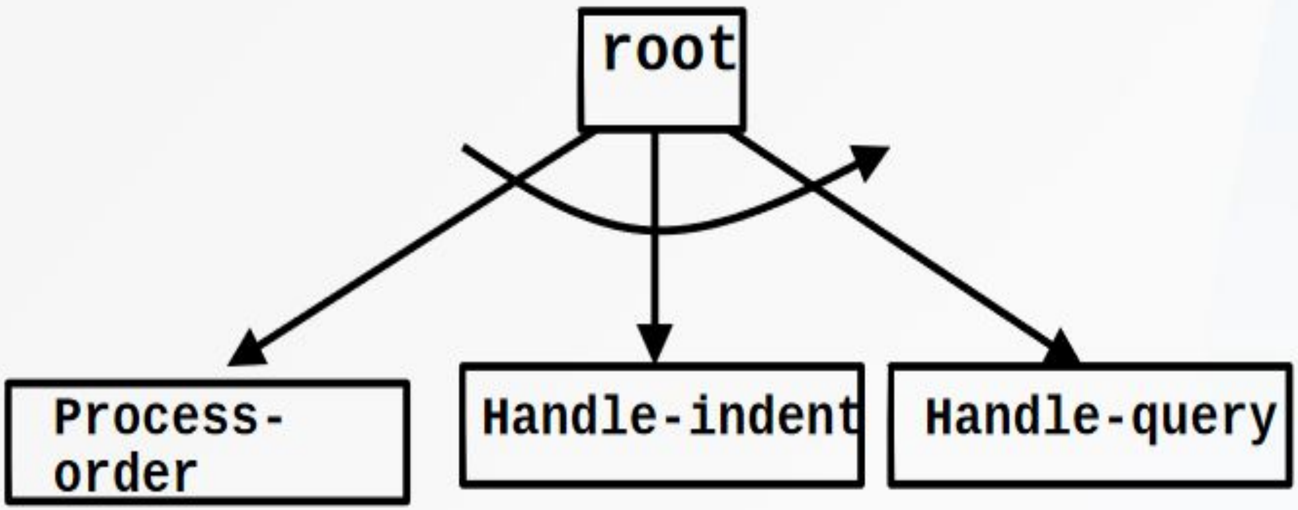
contd..

- **Library Modules:** Library modules represent **frequently called modules**:
  - a rectangle with double side edges.
  - Simplifies drawing when a module is called by several modules.

- **Selection:** The diamond symbol represents **selection** which implies that one module of several modules connected to the diamond symbol is invoked depending on some condition.



- **Repetition:** A loop around control flow arrows denotes that the concerned modules are invoked **repeatedly**.



Structure Chart vs. Flow Chart

- Structure chart represents the **data interchange** among different modules, whereas flow chart **does not** do the same.

- Structure chart does not represent the **sequence** in which the modules will be executed. On the other hand, the flow chart represents the **sequence** of tasks.

- Using structure chart, we can **easily identify** the different modules while we can not do the same using flow chart.

Transformation of DFD into Structure Chart

- Structured Design provides following two strategies to transform the DFD into structure chart.
  - **Transform Analysis**
  - **Transaction Analysis**

- At each level, first we need to determine whether we need to apply transform analysis or transaction analysis to a DFD.

Transform Analysis

- The first step in transform analysis is to divide the DFD into 3 parts named **input, logical processing, and output.**
  - **Input**
    - processes which convert input data from **physical to logical form**. **e.g.,** read characters from the terminal and store in internal tables or lists.
    - Each input portion is called an **<u>afferent branch</u>**.
    - It is Possible to have **more than one** afferent branch in a DFD.
  - **Output**
    - transforms output data from **logical form to physical form**. **e.g.,** from list or array into output characters.
    - Each output portion is called an **<u>efferent branch</u>**.
  - The remaining portions of a DFD called **<u>central transform</u>**

contd..

- Derive structure chart by drawing **one functional component** for:
  - the central transform,
  - each afferent branch,
  - each efferent branch.
- **First level of structure chart:**
  - draw a box for each input and output units
  - a box for the central transform.
- **Next, refine the structure chart:**
  - add **sub-modules** required by each high-level module. This is referred to as **"factoring"**.
  - Many levels of modules may be required to be added.

Example-1 RMS Calculator

- From a cursory analysis of the problem description,
  - easy to see that the system needs to perform:
    - **read-input:** accept the input numbers from the user,
    - **validate-input:** validate the numbers,
    - **compute-rms:** calculate the root mean square of the input numbers,
    - **write-result:** display the result.
- Here, we can say that **"get-good-data" is** the **afferent branch**, whereas **"write-result"** is the **efferent branch**.
  - **"get-good-data"** is further factored into **"read-input"** and **"validate-input"**.
- **compute-rms** is the **central transform**.

Structure Chart for RMS Calculating Software

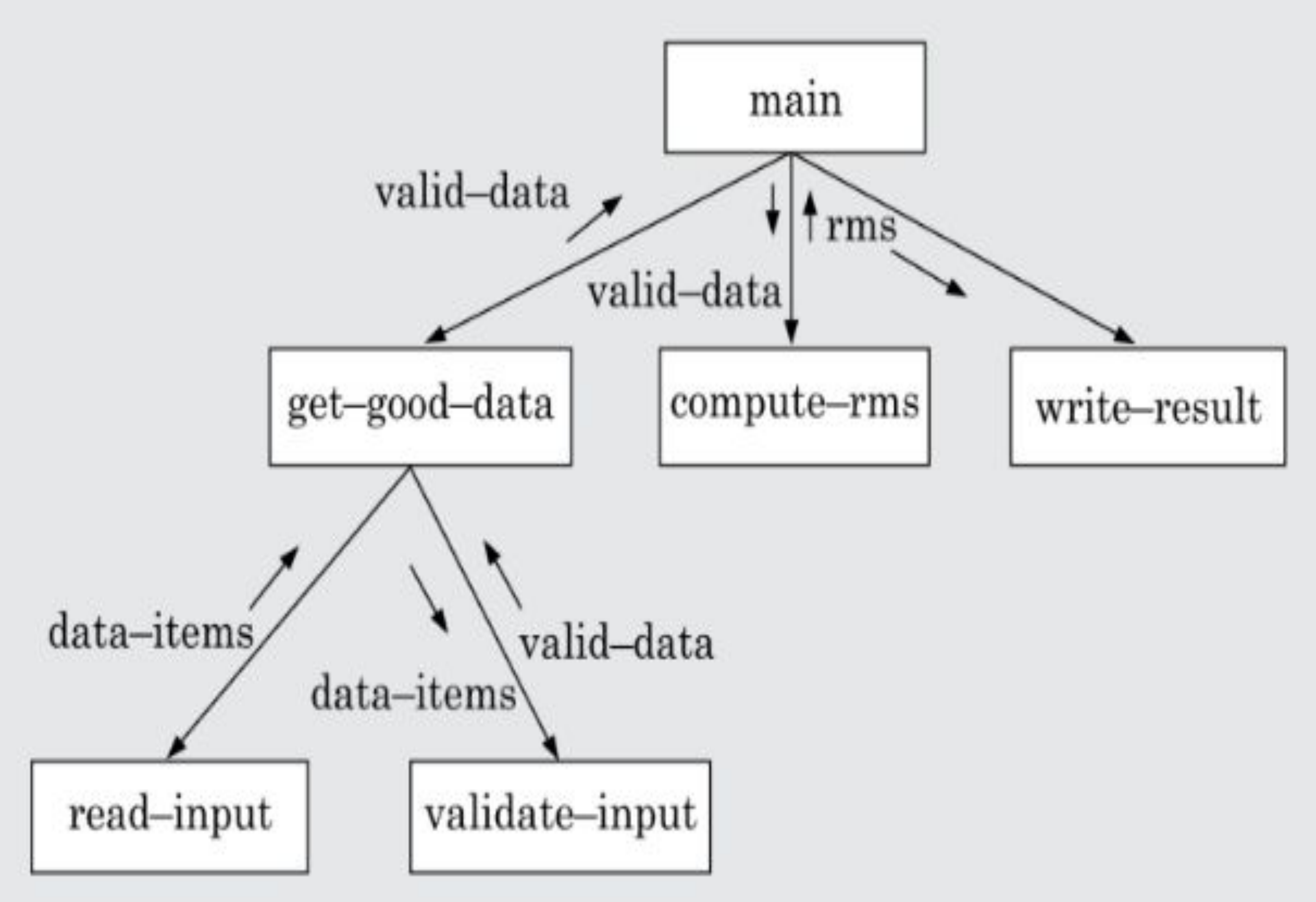


Fig. 6.12: Structure chart for RMS calculator

Example-2: Super Market Prize Scheme

- From a cursory analysis of the problem description,
  - easy to see that the system needs to perform:
    - accept customer details and generate customer number (CN)
    - accept sales data and record the sales data internally.
    - generate winner list
- Here, we can say that **“register-customer” and “register-sales”** are the **afferent branches.**
  - **“register-customer”** is further factored into **“get-customer -details”** and **“generate-CN”**.
  - **“register-sales”** is further factored into **“get-sales-details”** and **“record-sales-details”**
- **“generate-winner-list”** is the **central transform**.
  - it is further factored into **“find-total-sales”**, **“gen-surprise -gift list”** and **“gen-gold-coin-winner list”**.

contd..

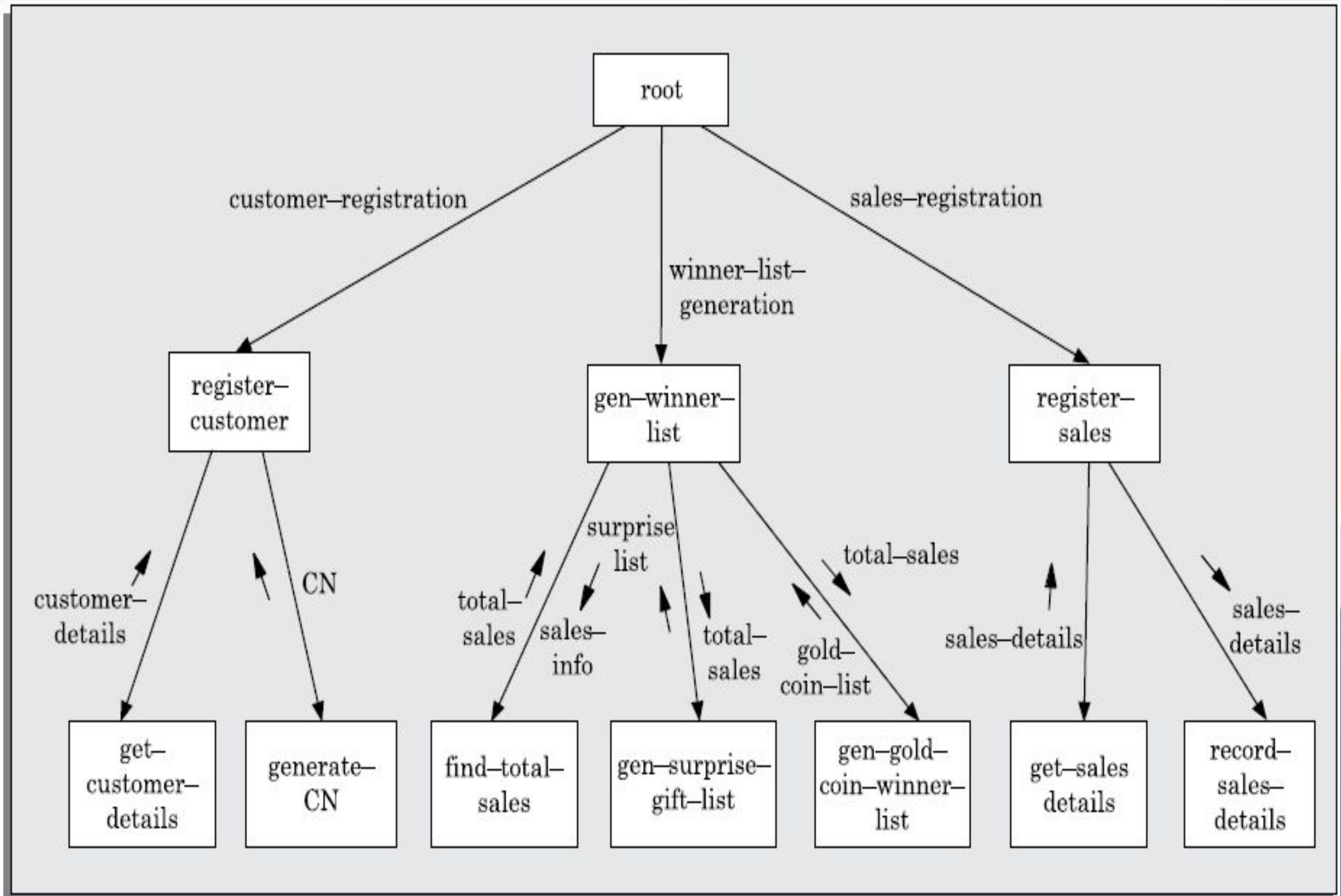


Fig. 6.13: Structure chart for Supermarket problem

Transaction Analysis

- **Transaction analysis** is an **alternative** to transform analysis and is useful in designing **transaction processing systems**.
- In a transaction driven system, different data items may pass through different computation paths through the DFD. Each different path refers to a **transaction**.
  - A **transaction** allows the user to perform some specific type of work by using the software.
  - For **each identified transaction**, the input data is traced to the output. All the traversed bubbles belong to the transaction.
  - These bubbles should be mapped to the same module in the structure chart.
- In structure chart, draw a root module and below this module, draw each identified transaction as a module.

Transaction Analysis

- **Useful for designing transaction processing programs.**
  - **Transform-centred systems:**
    - characterized by similar processing steps for every data item processed by input, process, and output bubbles. **Ex: Supermarket Software**
  - **Transaction-driven systems,**
    - one of several possible paths through the DFD is traversed depending upon the input data value. **Ex: Personal Library Software**
      - **Refer to pg nos. 262 and 263 for DFD Model;**
      - **Refer to pg. no. 271 for structure chart.**

Shortcomings of Structure Chart

- By looking at a structure chart, we can not say whether a module calls another module just once or many times.

- Also, by looking at a structure chart, we can not tell the order in which the different modules are invoked.

Detailed Design

- During detailed design, the pseudo code description of the processing and the different data structures are designed for the different modules of the structure chart.
- They are usually described in the form of **Module Specifications (MSPEC).**
  - MSPEC is usually written using structured english.
  - MSPEC for the non leaf modules describe the different conditions under which the responsibilities are delegated to the lower-level modules.
  - The MSPEC for the leaf-level modules should describe in algorithmic form how the primitive processing steps are carried out.
  - To develop the MSPEC of a module, it is usually necessary to refer to the DFD model and the SRS document to determine the functionality of the module

Design Review

- After a design is complete, the design is required to be reviewed.
- The review team usually consists of members with design, implementation, testing, and maintenance perspectives, who may or may not be the members of the development team.
- Normally, members of the team who would code the design, and test the code, the analysts, and the maintainers attend the review meeting.
- The review team checks the design documents especially for the following aspects:
  - Traceability
  - Correctness
  - Maintainability
  - Implementation