

# **Function-Oriented Software Design (continued):**

# Organization of this Lecture

- ⌘ A larger example of Structured Analysis

- ⌘ Structured Design

  - ☐ A major objective of this lecture is that you should be able to develop structured design from any DFD model.

- ⌘ Examples

- ⌘ Summary

# Review of Last Lecture

## ⌘ The goal of structured analysis:

- ☒ perform functional decomposition.

- ☒ represent using Data Flow Diagrams (DFDs).

## ⌘ DFDs are a hierarchical model:

- ☒ We examined why any hierarchical model is easy to understand

## ⌘ During structured analysis:

- ☒ Functional decomposition takes place

- ☒ in addition, data decomposition takes place.

## ⌘ At the most abstract level:

- ☒ context diagram

- ☒ refined to more detailed levels.

# DFD

- ⌘ Data flow diagram is graphical representation of flow of data in an information system.
- ⌘ It is capable of depicting incoming data flow, outgoing data flow and stored data.
- ⌘ The DFD does not mention anything about how data flows through the system.
- ⌘ Difference between DFD and Flowchart.
  - ☒ The flowchart depicts flow of control in program modules.
  - ☒ DFDs depict flow of data in the system at various levels. DFD does not contain any control or branch elements.

# Types of DFD

⌘ Data Flow Diagrams are either Logical or Physical.



☒ **Logical DFD** - This type of DFD concentrates on the system process, and flow of data in the system.

☒ For example in a Banking software system, how data is moved between different entities.

☒ **Physical DFD** - This type of DFD shows how the data flow is actually implemented in the system.

☒ It is more specific and close to the implementation.

# DFD Components

- ⌘ DFD can represent Source, destination, storage and flow of data using the following set of components –



- ⌘ **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.
- ⌘ **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.

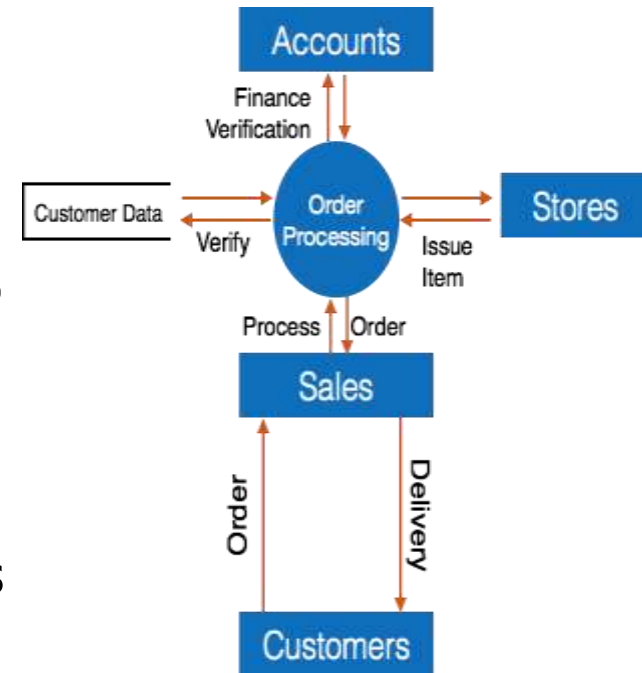
# Levels of DFD

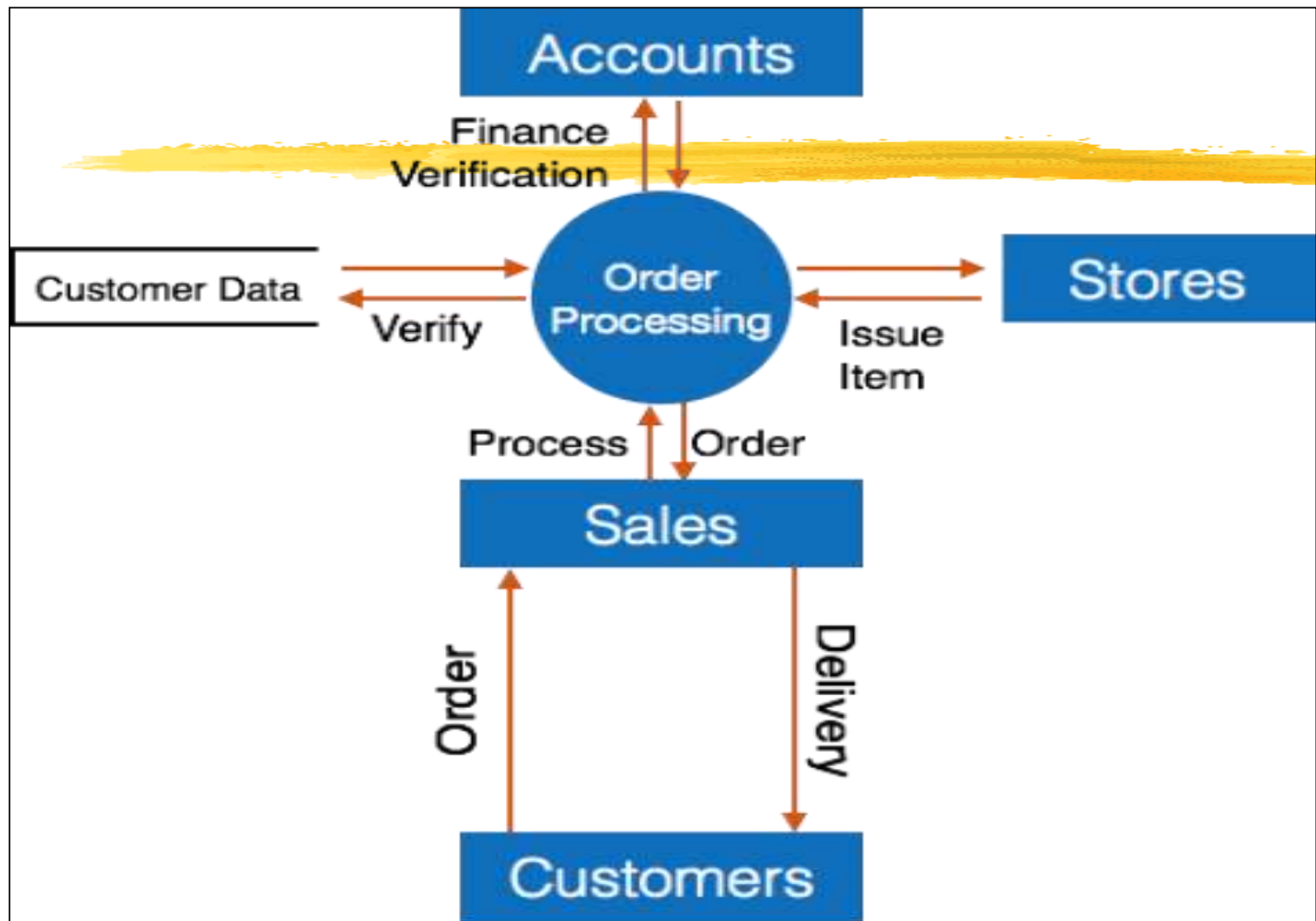
⌘ **Level 0** - Highest abstraction level DFD is known as Level 0 DFD, which depicts the entire information system as one diagram concealing all the underlying details.

⌘ Level 0 DFDs are also known as **context level** DFDs.

⌘ **Level 1** - The Level 0 DFD is broken down into more specific, Level 1 DFD. Level 1 DFD depicts basic modules in the system and flow of data among various modules.

⌘ Level 1 DFD also mentions basic processes and sources of information.



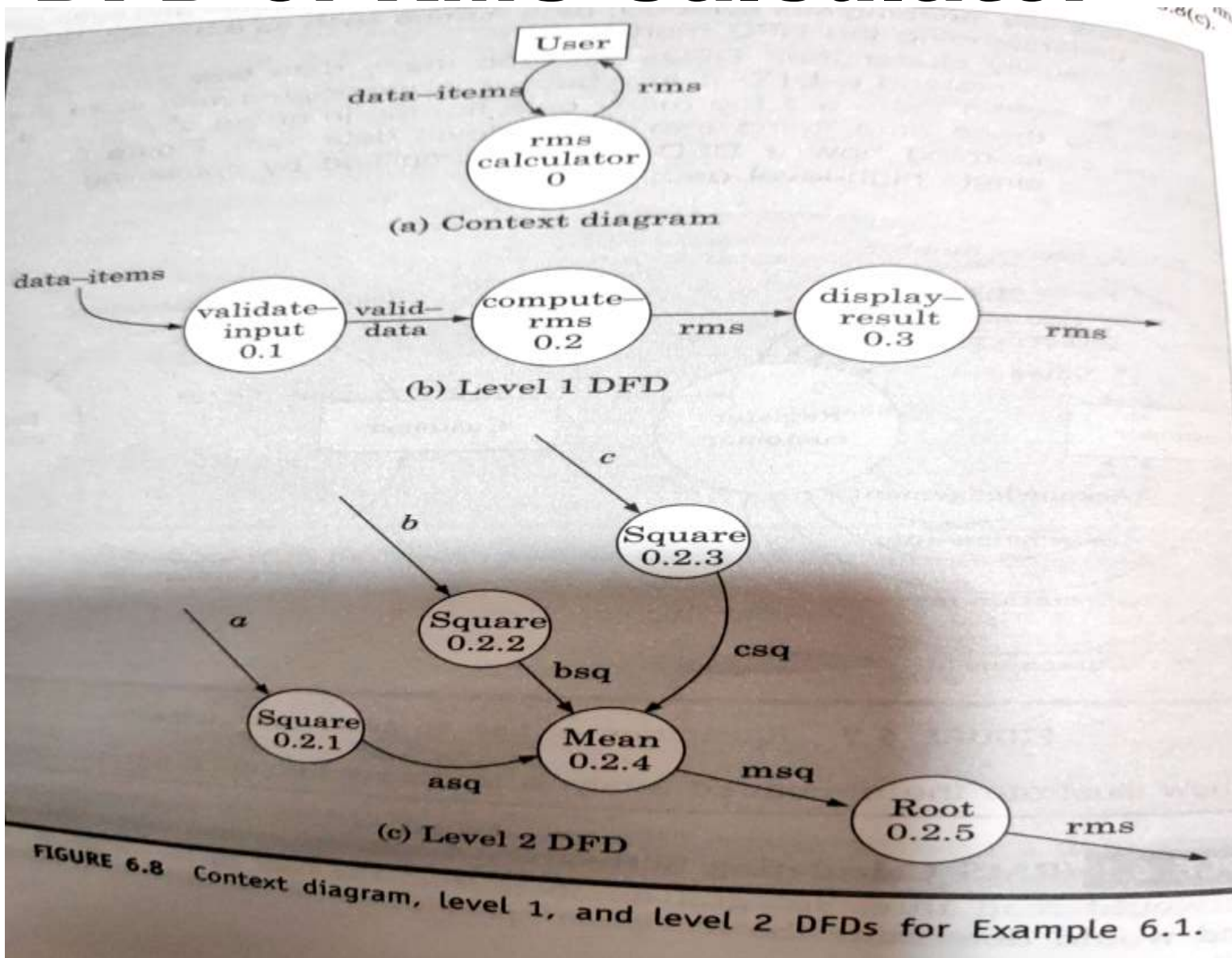




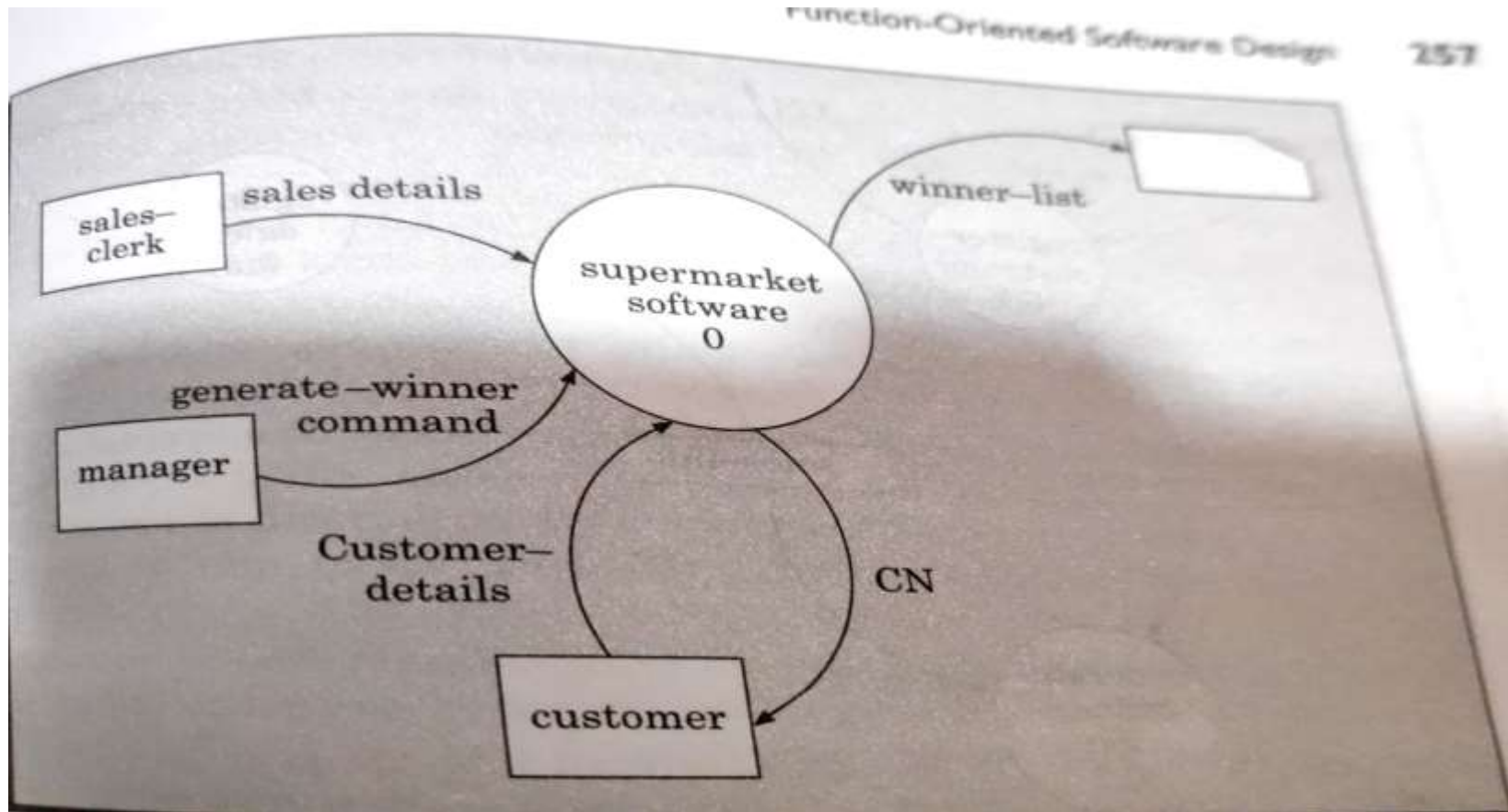
# Levels of DFD

- ⌘ **Level 2** - At this level, DFD shows how data flows inside the modules mentioned in Level 1.
- ⌘ Higher level DFDs can be transformed into more specific lower level DFDs with deeper level of understanding unless the desired level of specification is achieved.

# DFD of RMS Calculator



# Supermarket prize scheme context diagram



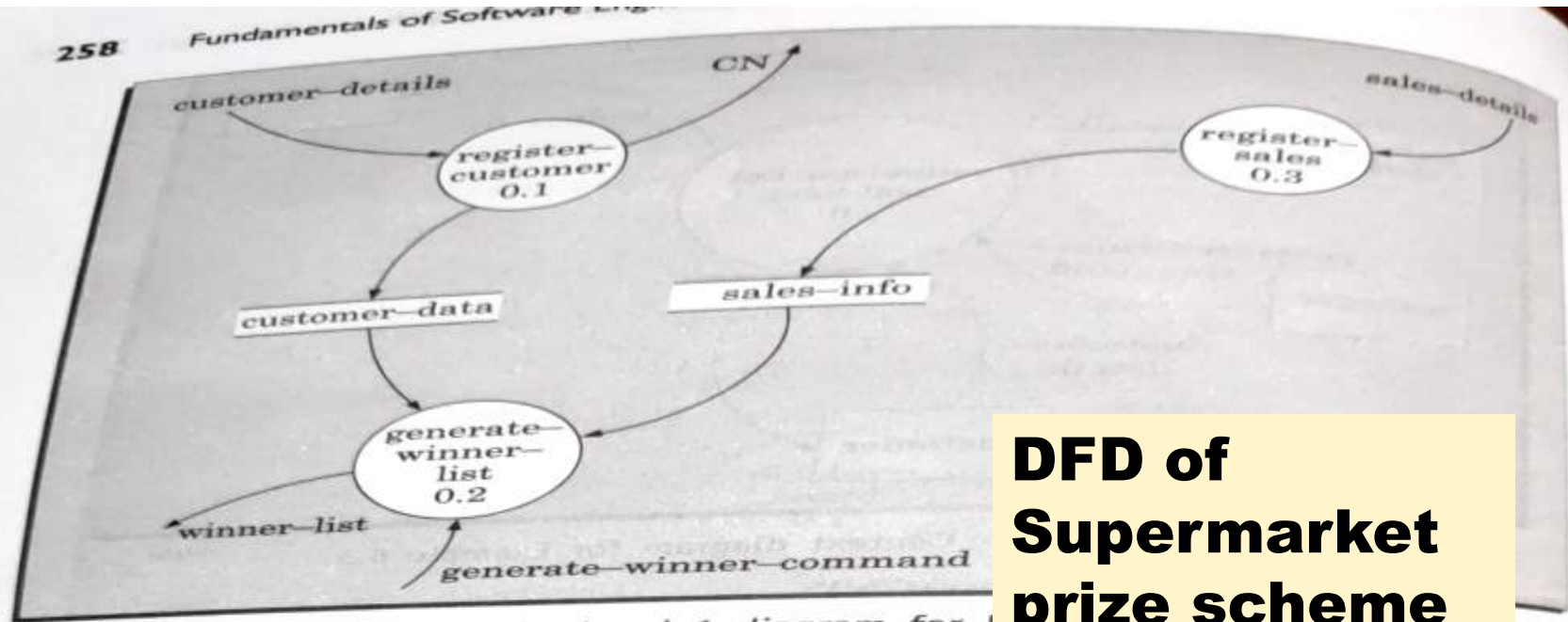


FIGURE 6.11 Level 1 diagram for

## DFD of Supermarket prize scheme

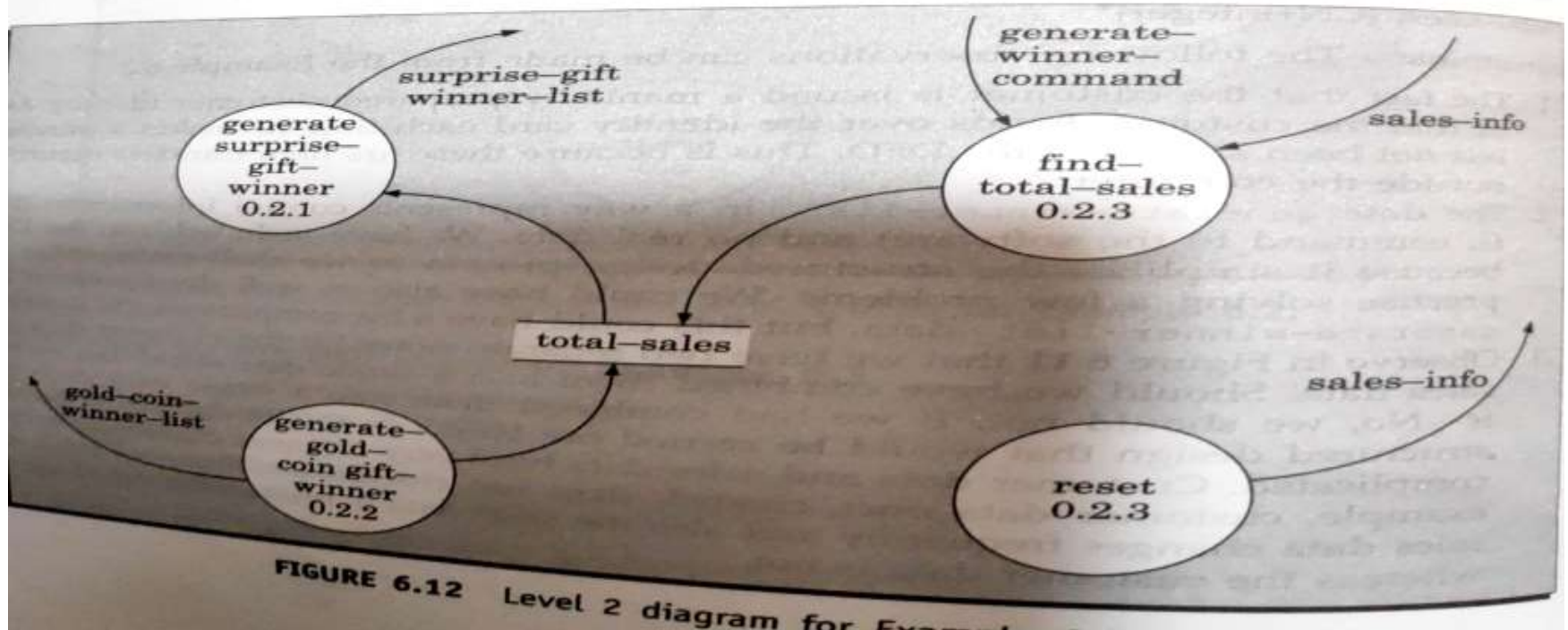


FIGURE 6.12 Level 2 diagram for

## **Example 3: Trading-House Automation System (TAS)**

- ⌘ A large trading house wants us to develop a software:
  - ☒ to automate book keeping activities associated with its business.
- ⌘ It has many regular customers:
  - ☒ who place orders for various kinds of commodities.
- ⌘ The trading house maintains names and addresses of its regular customers.
- ⌘ Each customer is assigned a unique customer identification number (CIN).
- ⌘ As per current practice when a customer places order:
  - ☒ the accounts department first checks the credit-worthiness of the customer.

# **Example: Trading-House Automation System (TAS)**

- ⌘ **The credit worthiness of a customer is determined:**
  - ☒ **by analyzing the history of his payments to the bills sent to him in the past.**
- ⌘ **If a customer is not credit-worthy:**
  - ☒ **his orders are not processed any further**
  - ☒ **an appropriate order rejection message is generated for the customer.**
- ⌘ **If a customer is credit-worthy:**
  - ☒ **items he/she has ordered are checked against the list of items the trading house deals with.**
- ⌘ **The items that the trading house does not deal with:**
  - ☒ **are not processed any further**
  - ☒ **an appropriate message for the customer for these items is generated.**



# **Example: Trading-House Automation System (TAS)**

- ⌘ **The items in a customer's order that the trading house deals with:**
  - ☒ **are checked for availability in the inventory.**
- ⌘ **If the items are available in the inventory in desired quantities:**
  - ☒ **a bill with the forwarding address of the customer is printed.**
  - ☒ **a material issue slip is printed.**
- ⌘ **The customer can produce the material issue slip at the store house:**
  - ☒ **take delivery of the items.**
  - ☒ **inventory data adjusted to reflect the sale to the customer.**

# Example: Trading-House Automation System (TAS)

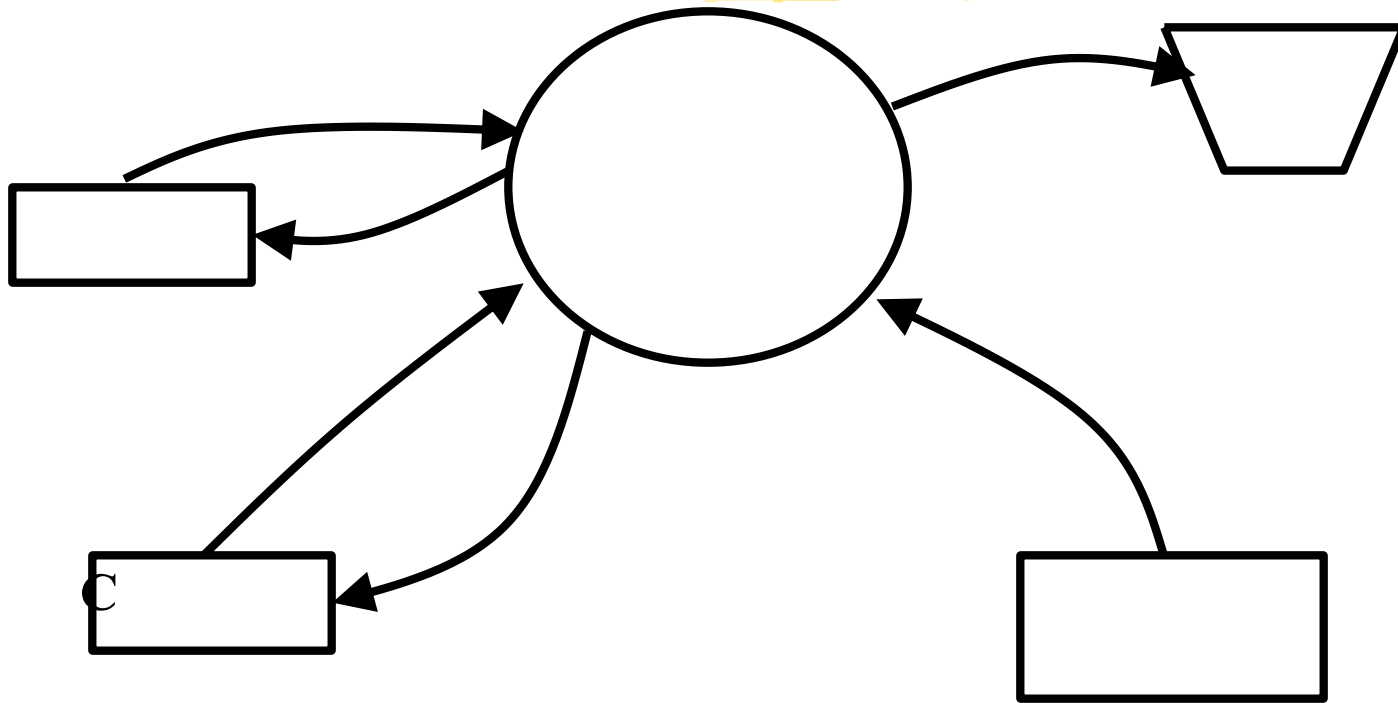
- ⌘ If an ordered item is not available in the inventory in sufficient quantity:
  - ⊞ to be able to fulfill pending orders store details in a "pending-order" file :
    - ⊗ out-of-stock items along with quantity ordered.
    - ⊗ customer identification number
- ⌘ The purchase department:
  - ⊞ would periodically issue commands to generate indents.
- ⌘ When **generate indents** command is issued:
  - ⊞ the system should examine the "pending-order" file
  - ⊞ determine the orders that are pending
  - ⊞ total quantity required for each of the items.



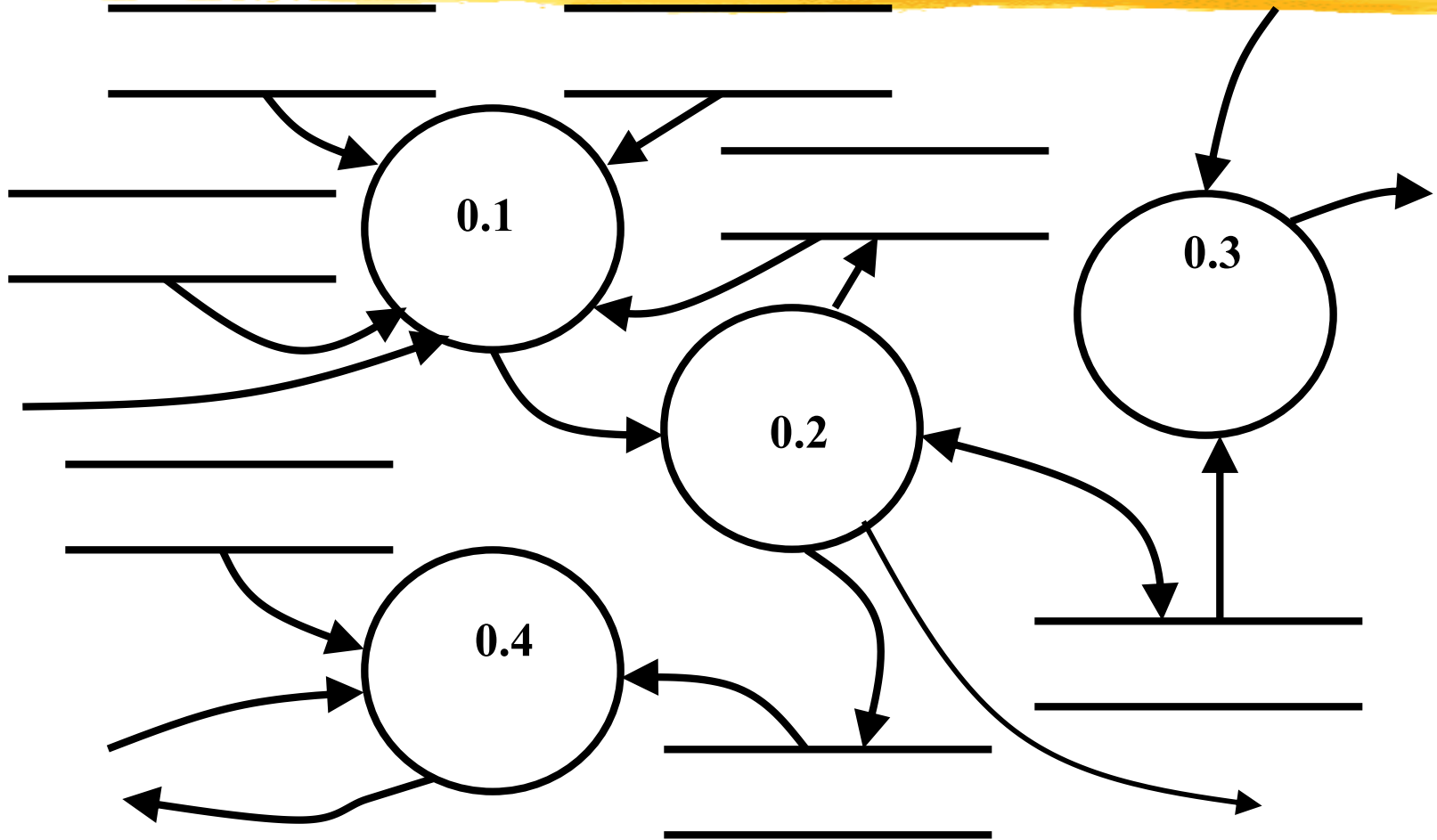
# **Example: Trading-House Automation System (TAS)**

- ⌘ **TAS should find out the addresses of the vendors who supply the required items:**
  - ☐ **examine the file containing vendor details (their address, items they supply etc.)**
  - ☐ **print out indents to those vendors.**
  
- ⌘ **TAS should also answers managerial queries:**
  - ☐ **statistics of different items sold over any given period of time**
  - ☐ **corresponding quantity sold and the price realized.**

# Context Diagram



# Level 1 DFD



# Example: Data Dictionary

- ⌘ **response: [bill + material-issue-slip, reject-message]**
- ⌘ **query: period /\* query from manager regarding sales statistics\*/**
- ⌘ **period: [date+date,month,year,day]**
- ⌘ **date: year + month + day**
- ⌘ **year: integer**
- ⌘ **month: integer**
- ⌘ **day: integer**
- ⌘ **order: customer-id + {items + quantity}\***
- ⌘ **accepted-order: order /\* ordered items available in inventory \*/**
- ⌘ **reject-message: order + message /\* rejection message \*/**
- ⌘ **pending-orders: customer-id + {items+quantity}\***
- ⌘ **customer-address: name+house#+street#+city+pin**

# Example: Data Dictionary

- ⌘ **item-name: string**
- ⌘ **house#: string**
- ⌘ **street#: string**
- ⌘ **city: string**
- ⌘ **pin: integer**
- ⌘ **customer-id: integer**
- ⌘ **bill: {item + quantity + price}\* + total-amount + customer-address**
- ⌘ **material-issue-slip: message + item + quantity + customer-address**
- ⌘ **message: string**
- ⌘ **statistics: {item + quantity + price }\***
- ⌘ **sales-statistics: {statistics}\***
- ⌘ **quantity: integer**

# Observation

⌘ From the examples,

☐ observe that DFDs help create:

☒ data model

☒ function model

⌘ As a DFD is refined into greater levels of detail:

☐ the analyst performs an implicit functional decomposition.

☐ At the same time, refinements of data takes place.

# 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.
- ⌘ **All external entities should be represented in the context diagram:**
  - ☒ external entities should not appear at any other level of DFD.
- ⌘ **Only 3 to 7 bubbles per diagram should be allowed:**
  - ☒ each bubble should be decomposed to between 3 and 7 bubbles.

# Guidelines For Constructing DFDs

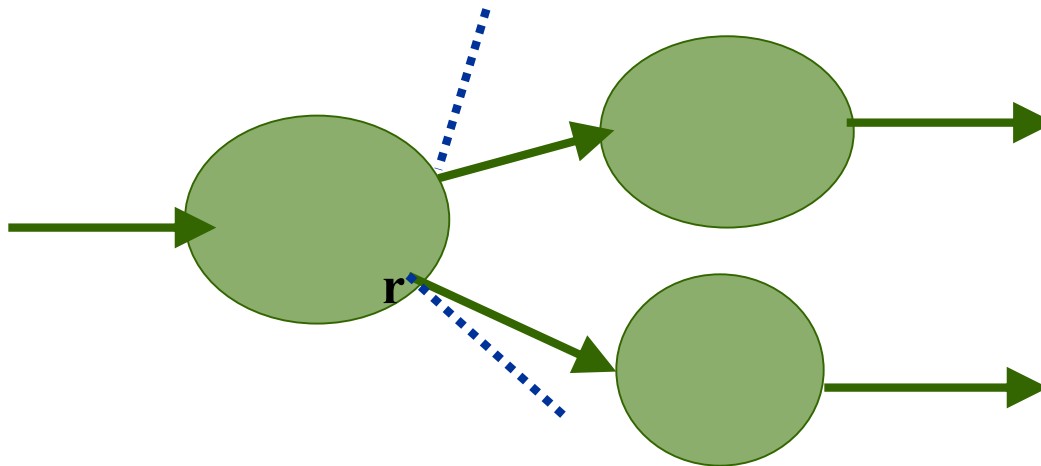
- ⌘ **A common mistake committed by many beginners:**
  - ⊞ attempting to represent control information in a DFD.
  - ⊞ e.g. trying to represent the order in which different functions are executed.
  
- ⌘ **A DFD does not represent control information:**
  - ⊞ when or in what order different functions (processes) are invoked
  - ⊞ the conditions under which different functions are invoked are not represented.
  - ⊞ For example, a function might invoke one function or another depending on some condition.



# Example-1

## ⌘ Check the input value:

- ☑ If the input value is less than -1000 or greater than +1000 generate an error message
- ☑ otherwise search for the number



# Guidelines For Constructing DFDs

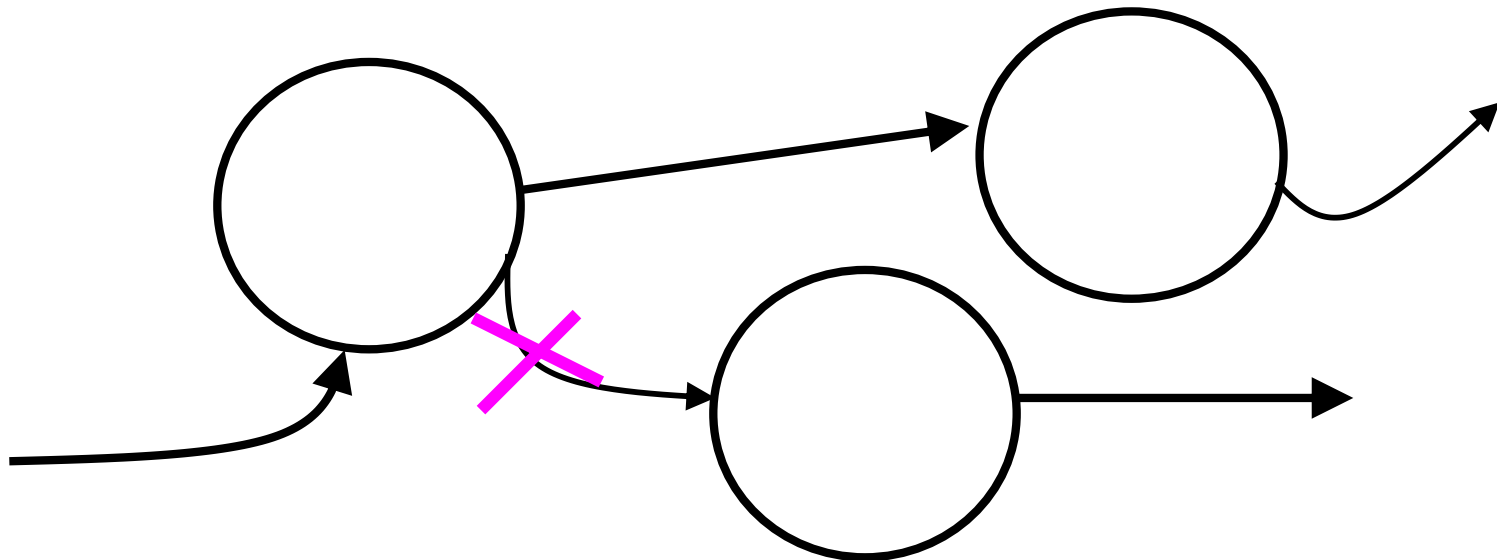
⌘ If a bubble **A** invokes either bubble **B** or bubble **C** depending on some conditions:

☐ represent the data that flows from bubble **A to bubble B and bubbles A to C**

☐ not the conditions depending on which a process is invoked.

# Example-2

- ⌘ A function accepts the book name to be searched from the user
- ⌘ If the entered book name is not a valid book name
  - ⏏ generates an error message,
- ⌘ If the book name is valid,
  - ⏏ searches the book name in the database.



# Guidelines For Constructing DFDs

⌘ **All functions of the system must be captured in the DFD model:**

☐ **no function specified in the SRS document should be overlooked.**

⌘ **Only those functions specified in the SRS document should be represented:**

☐ **do not assume extra functionality of the system not specified by the SRS document.**

# 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**

# 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.

# Shortcomings of the DFD Model

⌘ For example, a bubble named find-book-position has only intuitive meaning:

☒ does not specify several things:

☒ what happens when some input information is missing or is incorrect.

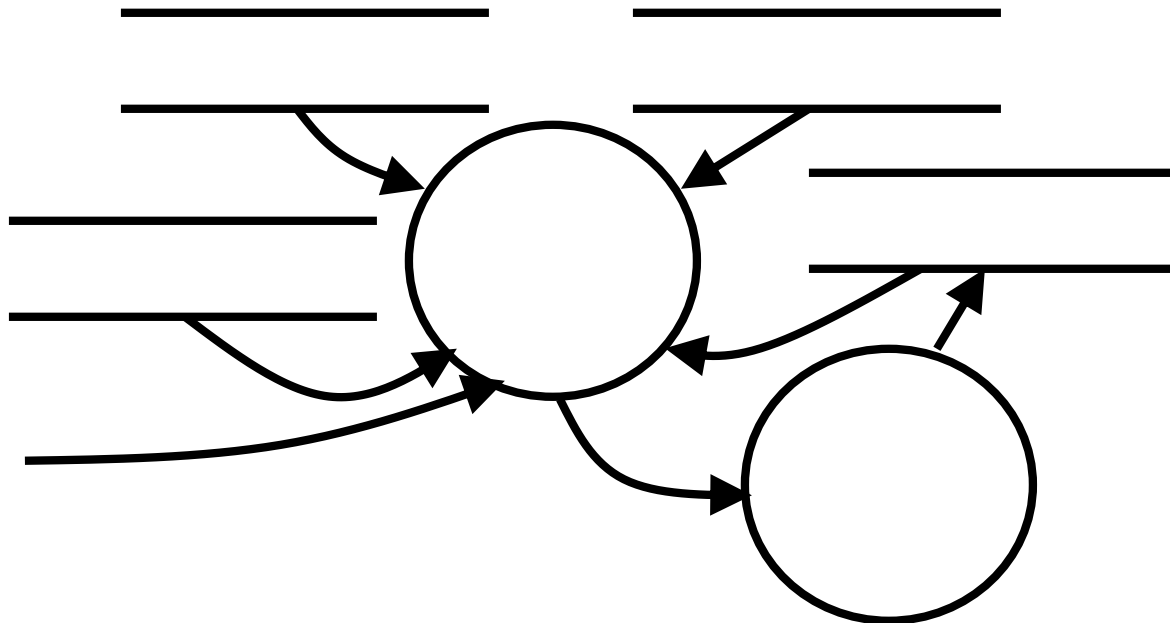
☒ Does not convey anything regarding what happens when book is not found

☒ or what happens if there are books by different authors with the same book title.

# Shortcomings of the DFD Model

⌘ **Control information is not represented:**

☒ For instance, order in which inputs are consumed and outputs are produced is not specified.





# Creating Data Flow Diagrams

## Lemonade Stand Example



# Creating Data Flow Diagrams

## Example

The operations of a simple lemonade stand will be used to demonstrate the creation of dataflow diagrams.



## Steps:

- 1. Create a list of activities**
  - Old way: no Use-Case Diagram**
  - New way: use Use-Case Diagram**
- 2. Construct Context Level DFD (identifies sources and sink)**
- 3. Construct Level 1 DFD (identifies manageable sub processes )**
- 4. Construct Level 2- n DFD (identifies actual data flows and data stores )**

# Creating Data Flow Diagrams

## Example

Think through the activities that take place at a lemonade stand.



## 1. Create a list of activities

**Customer Order**  
**Serve Product**  
**Collect Payment**  
**Produce Product**  
**Store Product**

# Creating Data Flow Diagrams

## Example

Also think of the additional activities needed to support the basic activities.



## 1. Create a list of activities

**Customer Order**  
**Serve Product**  
**Collect Payment**  
**Produce Product**  
**Store Product**  
**Order Raw Materials**  
**Pay for Raw Materials**  
**Pay for Labor**

# Creating Data Flow Diagrams

## Example

Group these activities in some logical fashion, possibly functional areas.



## 1. Create a list of activities

**Customer Order**  
**Serve Product**  
**Collect Payment**

**Produce Product**  
**Store Product**

**Order Raw Materials**  
**Pay for Raw Materials**

**Pay for Labor**

# Creating Data Flow Diagrams

## Example

Create a context level diagram identifying the sources and sinks (users).

**Customer Order**  
**Serve Product**  
**Collect Payment**

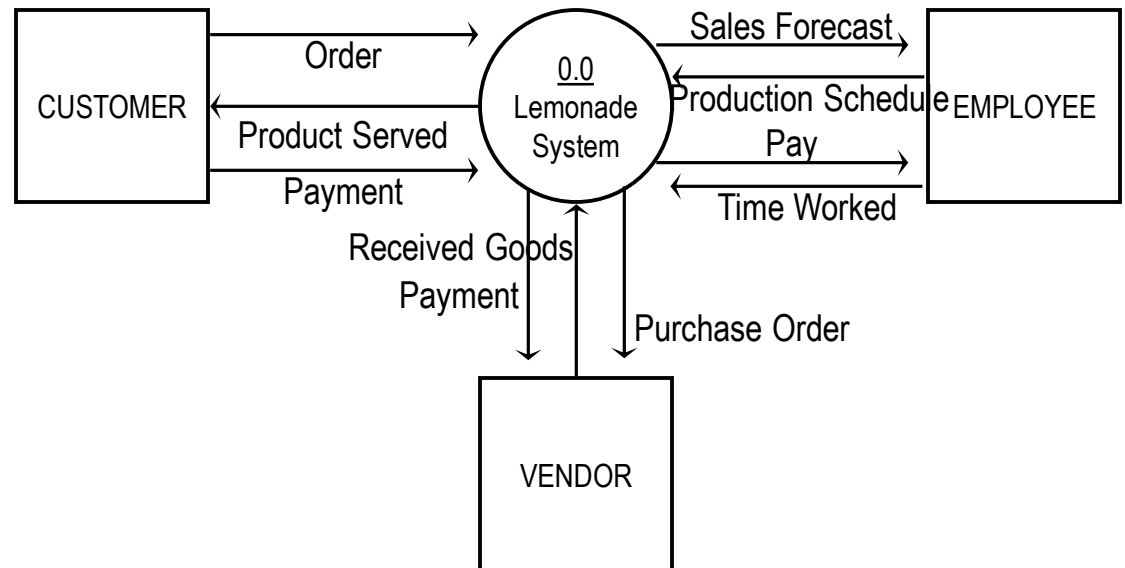
**Produce Product**  
**Store Product**

**Order Raw Materials**  
**Pay for Raw Materials**

**Pay for Labor**

## 2. Construct Context Level DFD (identifies sources and sink)

### Context Level DFD



# Creating Data Flow Diagrams

## Example

Create a level 1 diagram identifying the logical subsystems that may exist.

**Customer Order**  
**Serve Product**  
**Collect Payment**

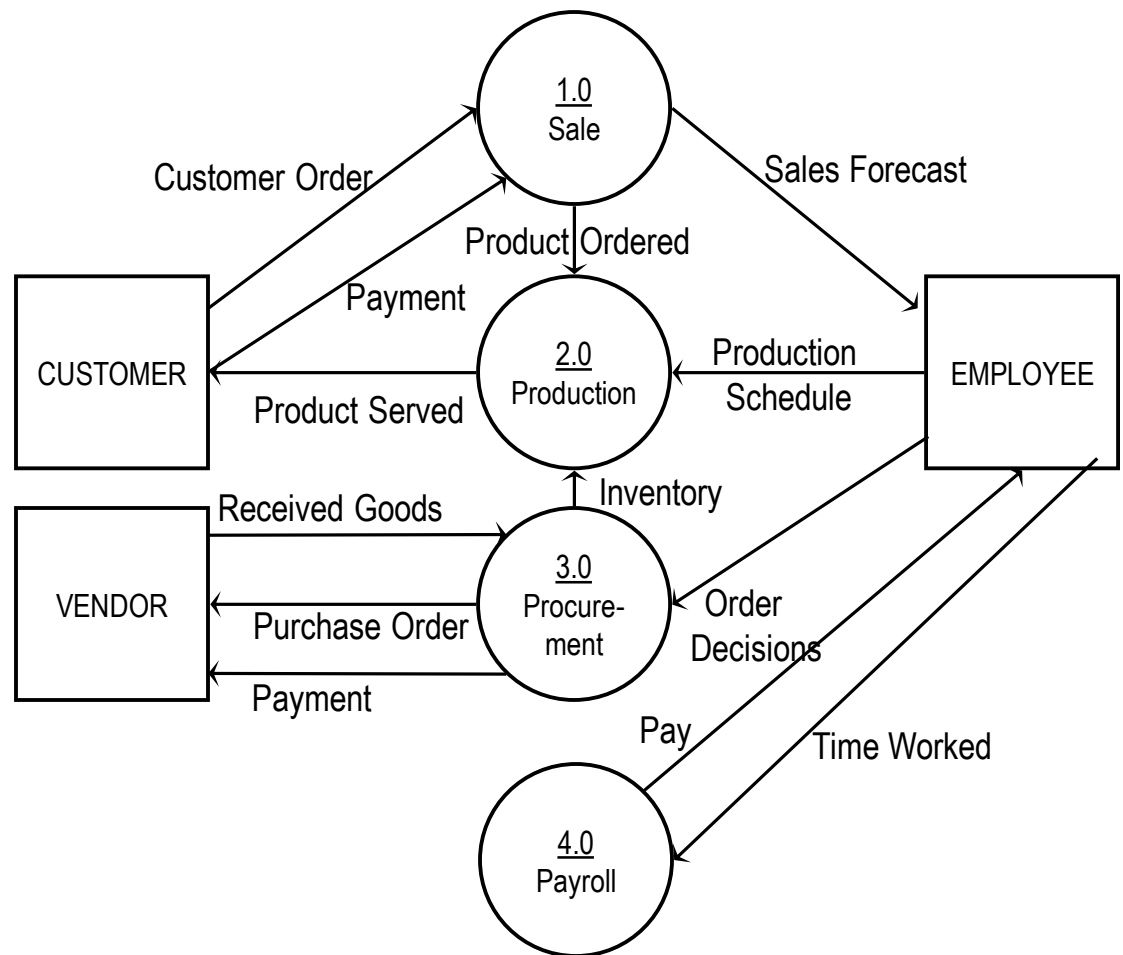
**Produce Product**  
**Store Product**

**Order Raw Materials**  
**Pay for Raw Materials**

**Pay for Labor**

## 3. Construct Level 1 DFD (identifies manageable sub processes )

### Level 1 DFD



# Creating Data Flow Diagrams

## Example

Create a level 2 decomposing the processes in level 1 and identifying data stores.

**Customer Order**  
**Serve Product**  
**Collect Payment**

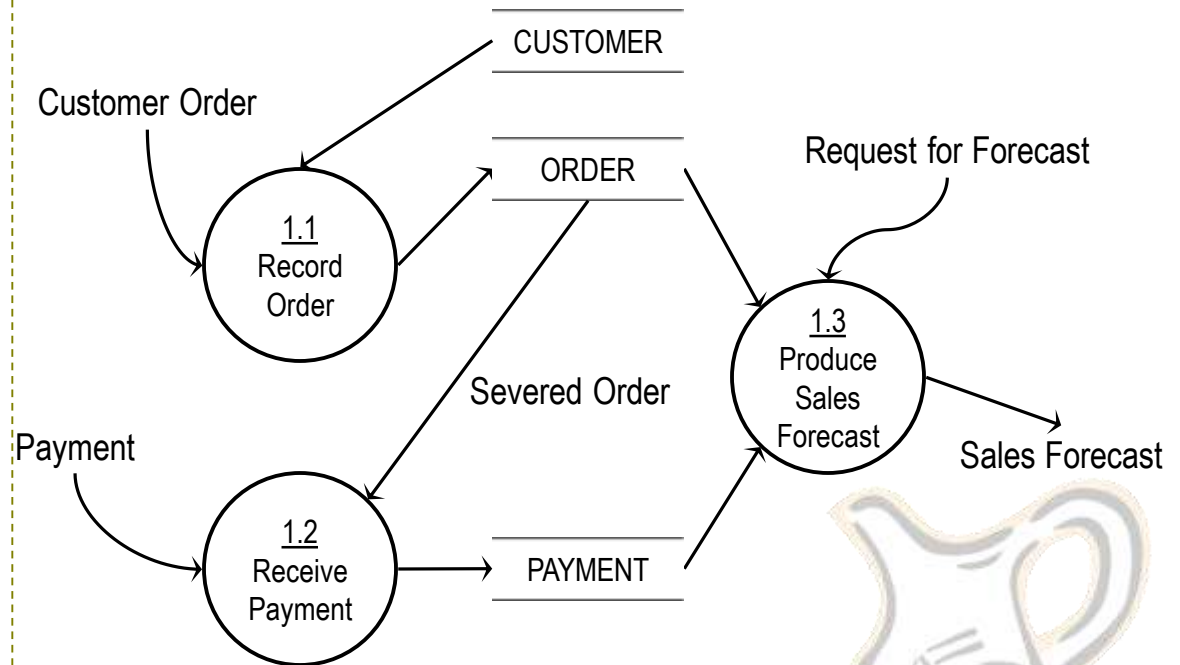
**Produce Product**  
**Store Product**

**Order Raw Materials**  
**Pay for Raw Materials**

**Pay for Labor**

## 4. Construct Level 2- n DFD (identifies actual data flows and data stores )

### Level 2 DFD





# Creating Data Flow Diagrams

## Example

Create a level 2 decomposing the processes in level 1 and identifying data stores.

**Customer Order**  
**Serve Product**  
**Collect Payment**

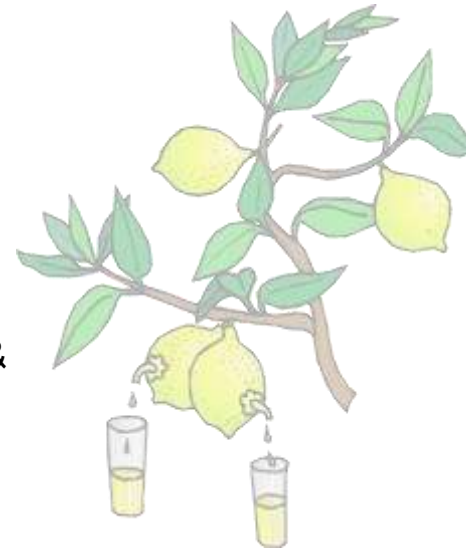
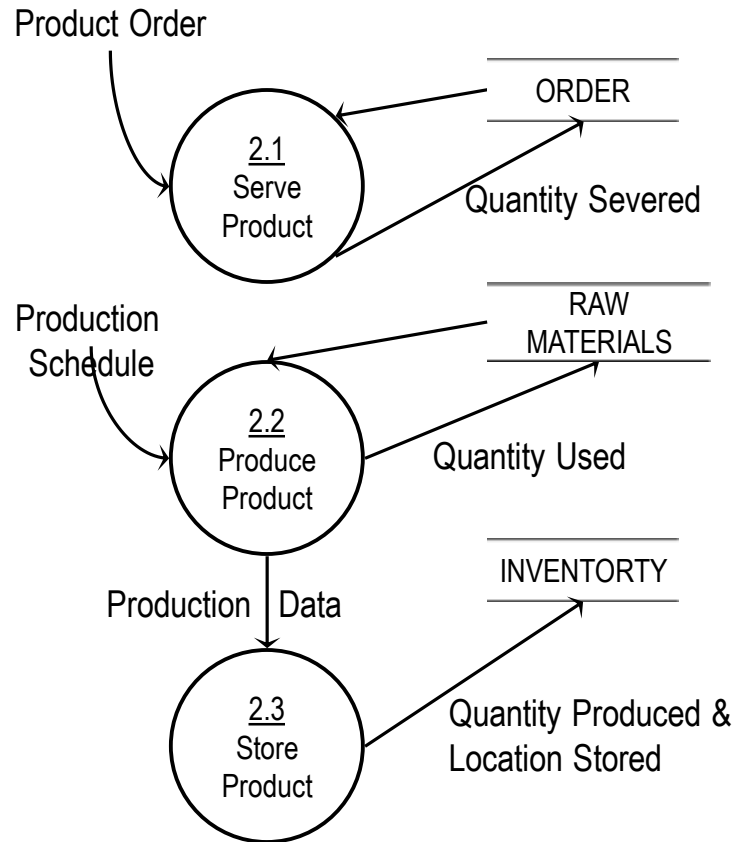
**Produce Product**  
**Store Product**

**Order Raw Materials**  
**Pay for Raw Materials**

**Pay for Labor**

## 4. Construct Level 2 (continued)

### Level 2 DFD



# Creating Data Flow Diagrams

## Example

Create a level 2 decomposing the processes in level 1 and identifying data stores.

**Customer Order**  
**Serve Product**  
**Collect Payment**

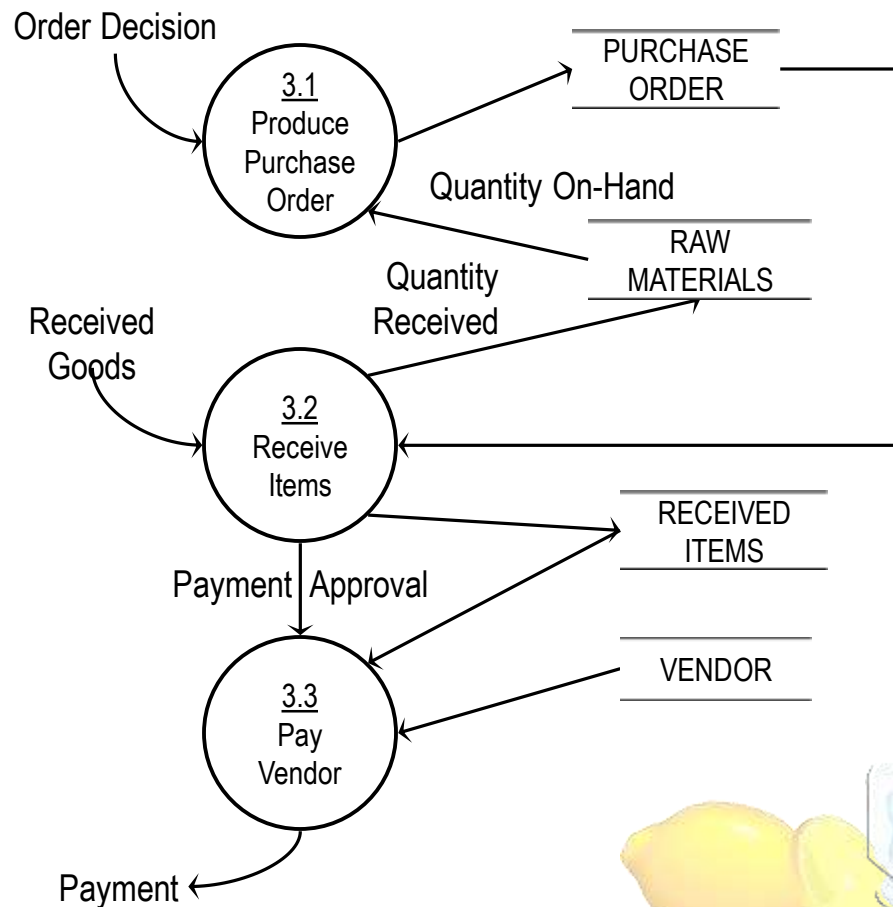
**Produce Product**  
**Store Product**

**Order Raw Materials**  
**Pay for Raw Materials**

**Pay for Labor**

## 4. Construct Level 2 (continued)

### Level 2 DFD



# Creating Data Flow Diagrams

## Example

Create a level 2 decomposing the processes in level 1 and identifying data stores.

**Customer Order**  
**Serve Product**  
**Collect Payment**

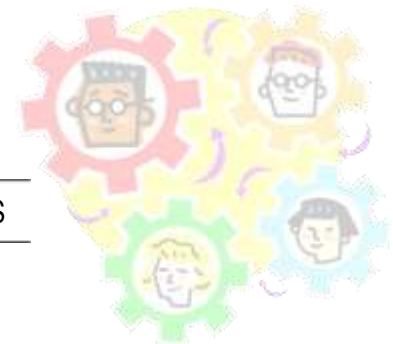
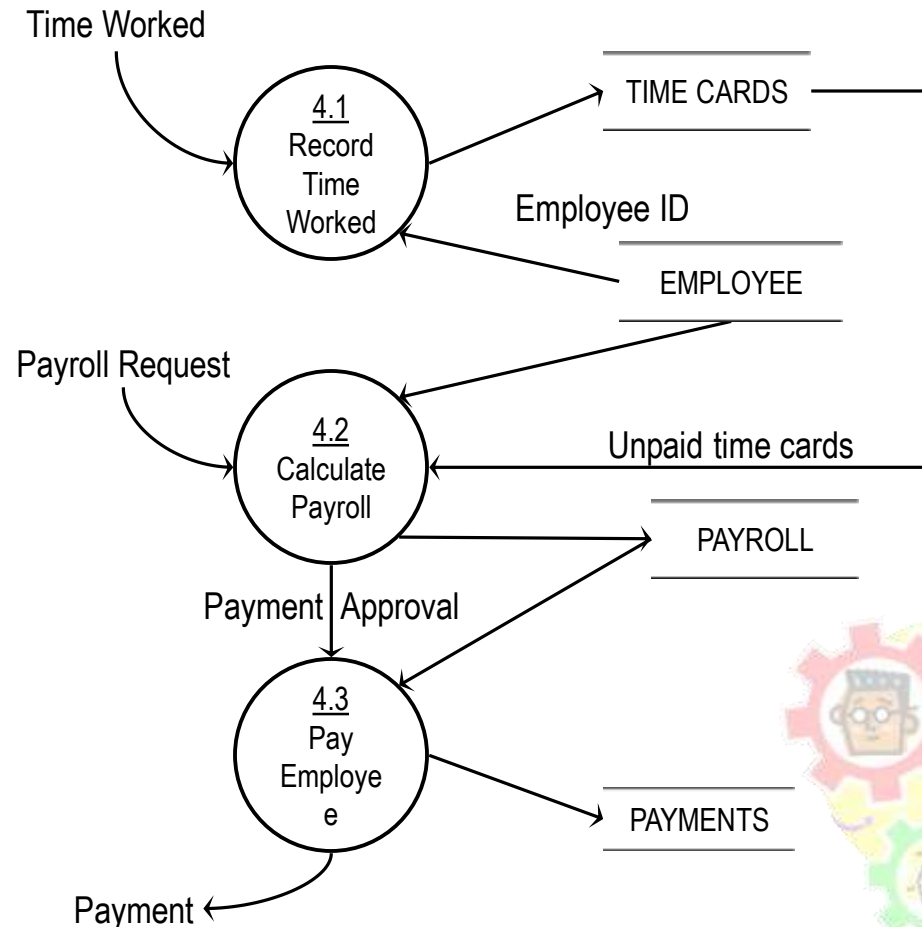
**Produce Product**  
**Store Product**

**Order Raw Materials**  
**Pay for Raw Materials**

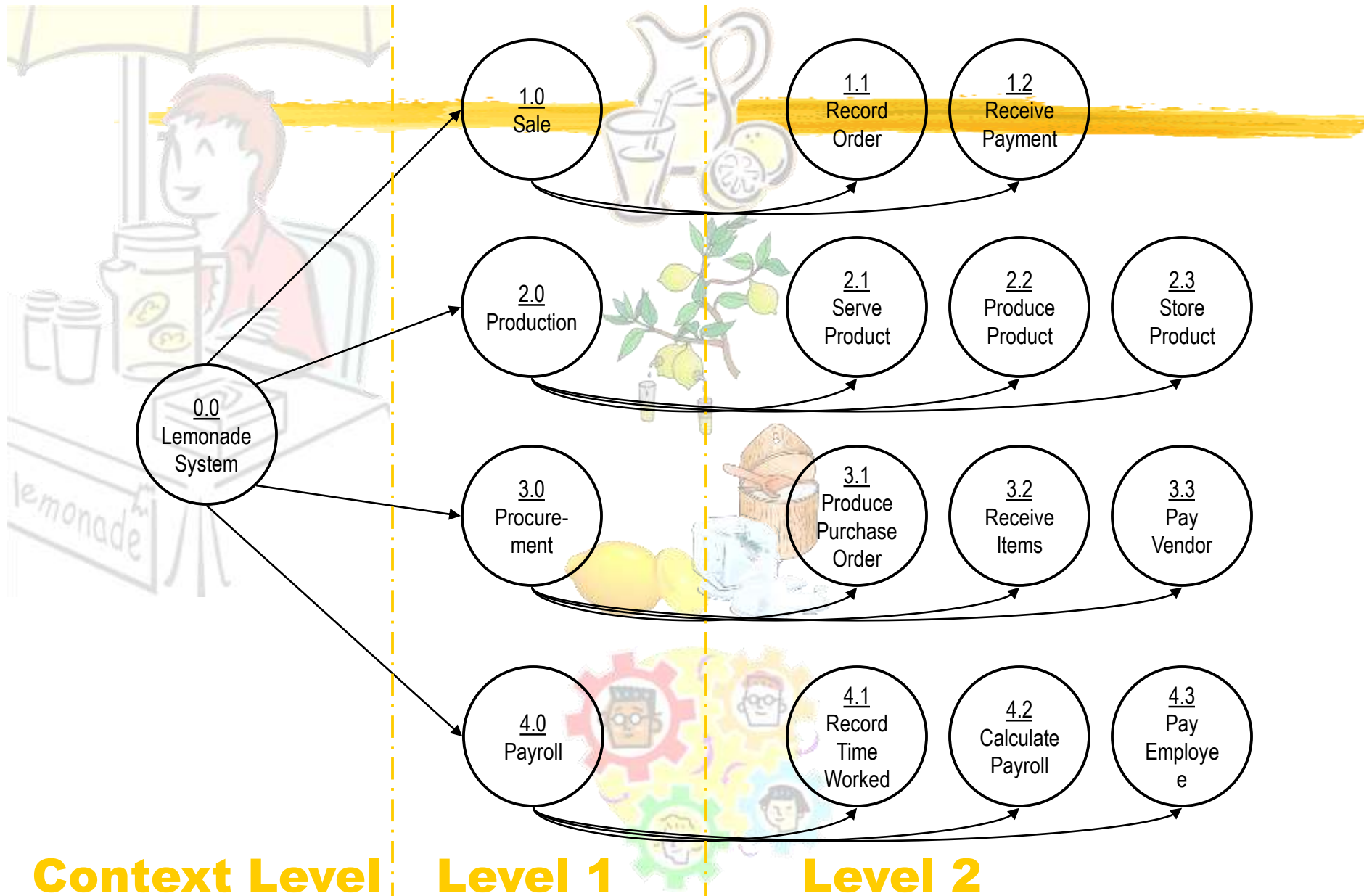
**Pay for Labor**

## 4. Construct Level 2 (continued)

### Level 2 DFD



# Process Decomposition



# 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**

# Shortcomings of the DFD Model

⌘ DFD technique does not provide:

- ☒ any clear guidance as to how exactly one should go about decomposing a function:

- ☒ one has to use subjective judgement to carry out decomposition.

⌘ Structured analysis techniques do not specify when to stop a decomposition process:

- ☒ to what length decomposition needs to be carried out.

# Extending DFD Technique to Real-Time Systems

⌘ For real-time systems (systems having time bounds on their actions),

☒ essential to model control flow and events.

☒ Widely accepted technique: **Ward and Mellor technique.**

☒ a type of process (bubbles) that handles only control flows is introduced.

☒ These processes are represented using dashed circles.

# Structured Design

## ⌘ The aim of structured design

- ☒ transform the results of structured analysis (i.e., a DFD representation) into a structure chart.

## ⌘ A structure chart represents the software architecture:

- ☒ various modules making up the system,
- ☒ module dependency (i.e. which module calls which other modules),
- ☒ parameters passed among different modules.



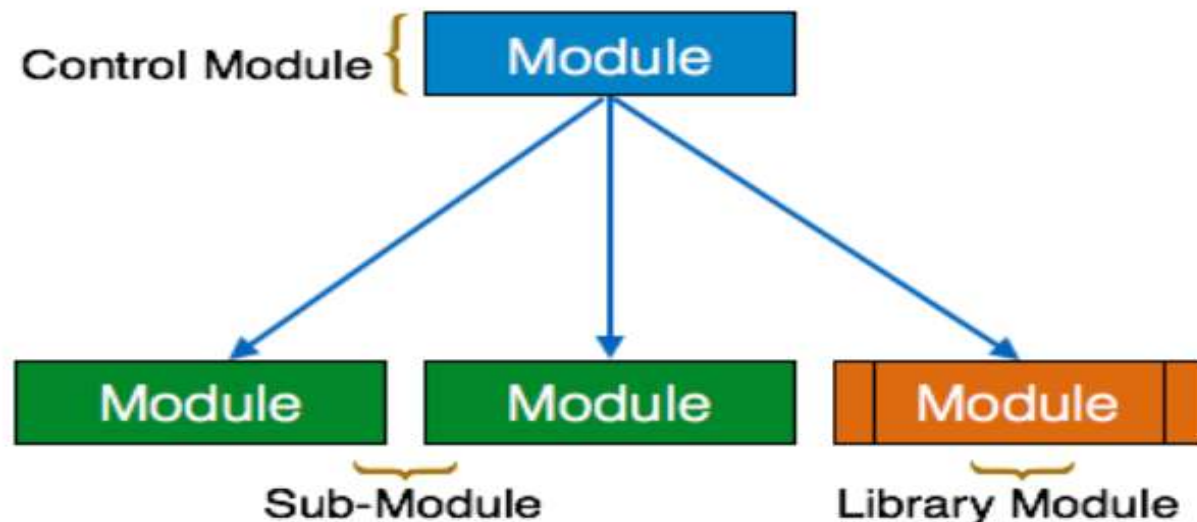
# Structure Charts

Structure chart is a chart derived from Data Flow Diagram. It represents the system in more detail than DFD. It breaks down the entire system into lowest functional modules, describes functions and sub-functions of each module of the system to a greater detail than DFD.

Structure chart represents hierarchical structure of modules. At each layer a specific task is performed.

Here are the symbols used in construction of structure charts -

- **Module** - It represents process or subroutine or task. A control module branches to more than one sub-module. Library Modules are re-usable and invokable from any module.



# Structure Chart

## ⌘ Structure chart representation

- ☑ easily implementable using programming languages.

## ⌘ Main focus of a structure chart:

- ☑ define the module structure of a software,

- ☑ interaction among different modules,

- ☑ procedural aspects (e.g, how a particular functionality is achieved) are not represented.

# 1. Basic building blocks of structure chart

## ⌘ Rectangular box:

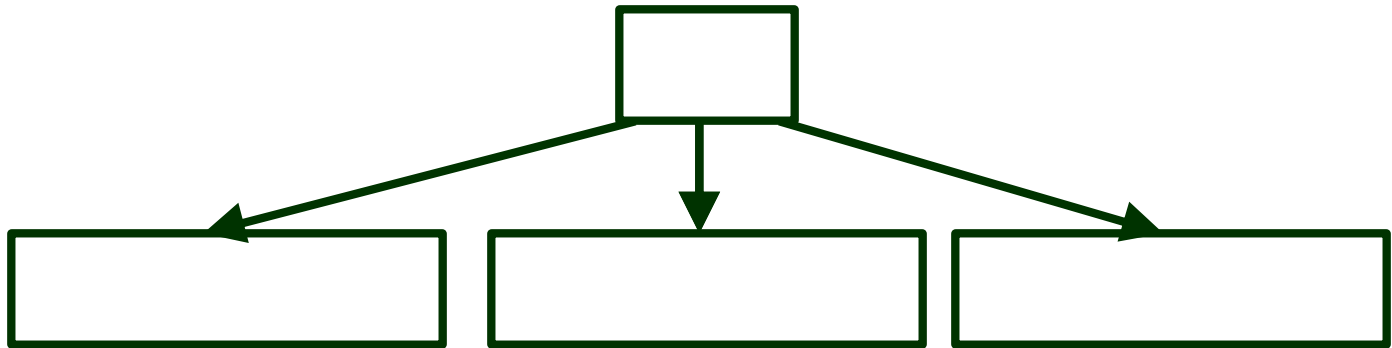
☐ A rectangular box represents a module.

☐ annotated with the name of the module it represents.



# 2. Arrows

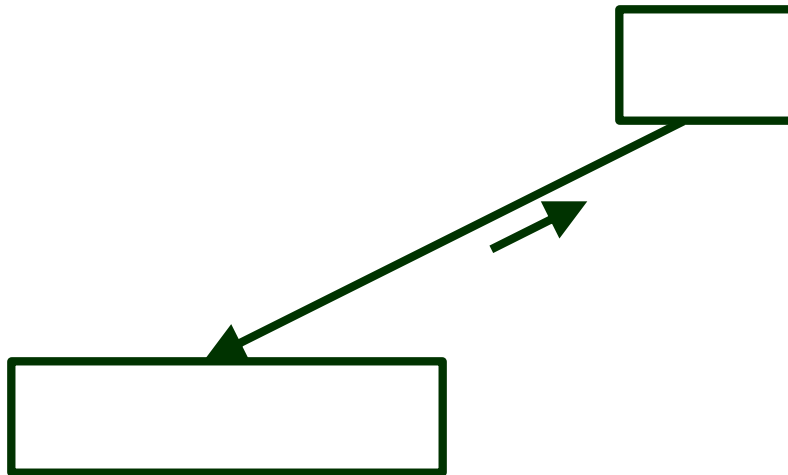
- ⌘ An arrow between two modules implies:
  - 📁 during execution control is passed from one module to the other in the direction of the arrow.



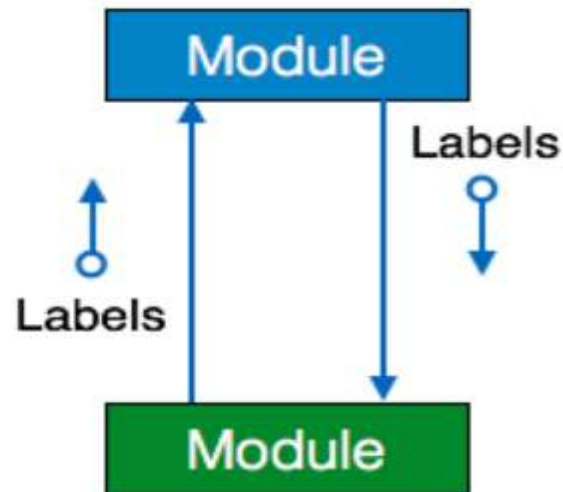
# 3. Data flow Arrows

⌘ Data flow arrows represent:

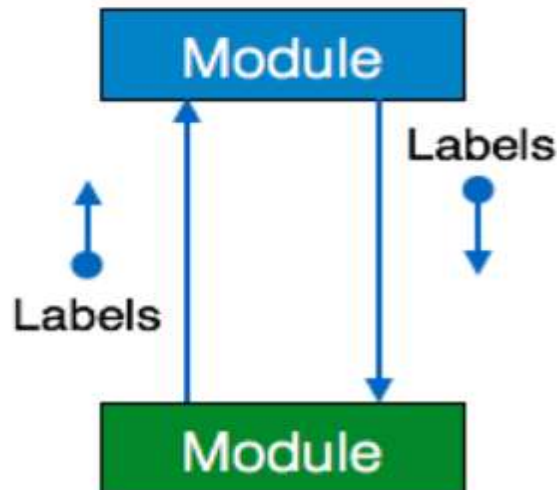
📁 data passing from one module to another in the direction of the arrow.



- **Data flow** - A directed arrow with empty circle at the end represents data flow.



- **Control flow** - A directed arrow with filled circle at the end represents control flow.



# 4. Library modules

⌘ **Library modules represent frequently called modules:**

☐ a rectangle with double side edges.

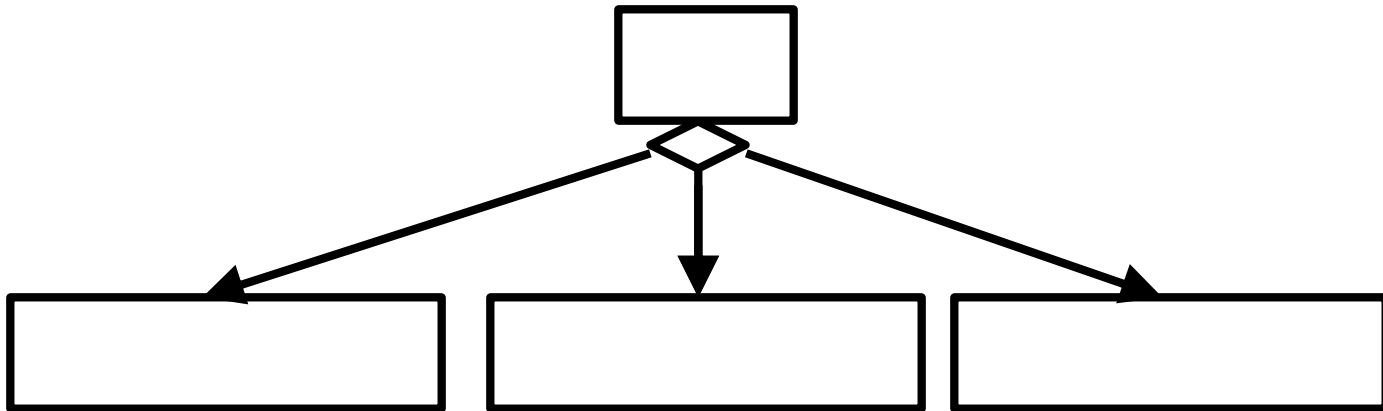
☐ Simplifies drawing when a module is called by several modules.



# 5. Selection

⌘ The diamond symbol represents:

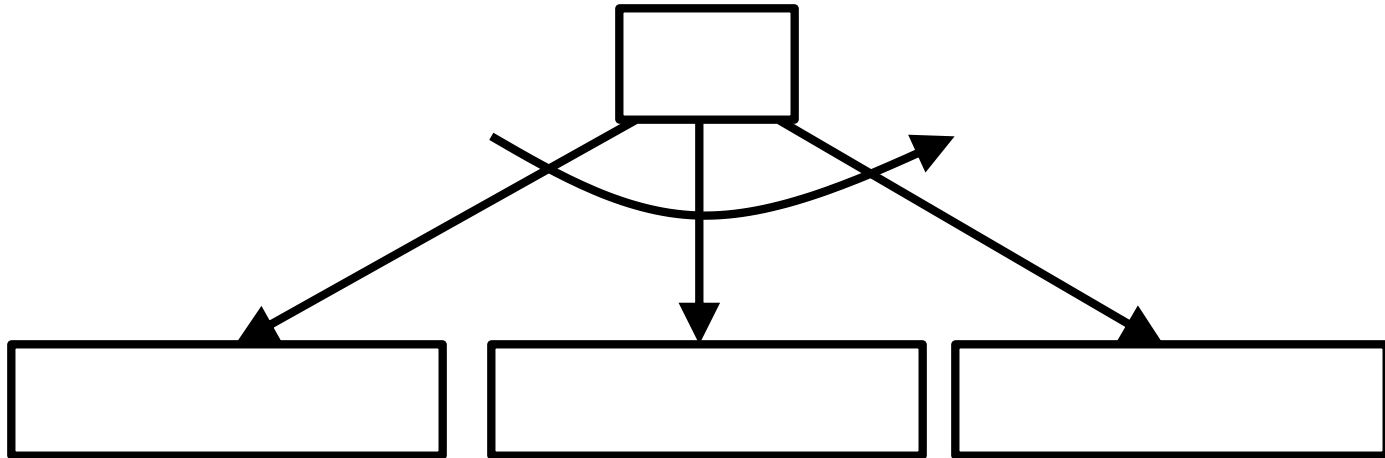
⏏ one module of several modules connected to the diamond symbol is invoked depending on some condition.





# 6. Repetition

⌘ A loop around control flow arrows denotes that the concerned modules are invoked repeatedly.



# Structure Chart

- ⌘ There is only one module at the top:
  - ☐ the **root module**.
- ⌘ There is at most one control relationship between any two modules:
  - ☐ if module A invokes module B,
  - ☐ module B cannot invoke module A.
- ⌘ The main reason behind this restriction:
  - ☐ **consider modules in a structure chart to be arranged in layers or levels.**

# Structure Chart

**⌘ The principle of abstraction:**

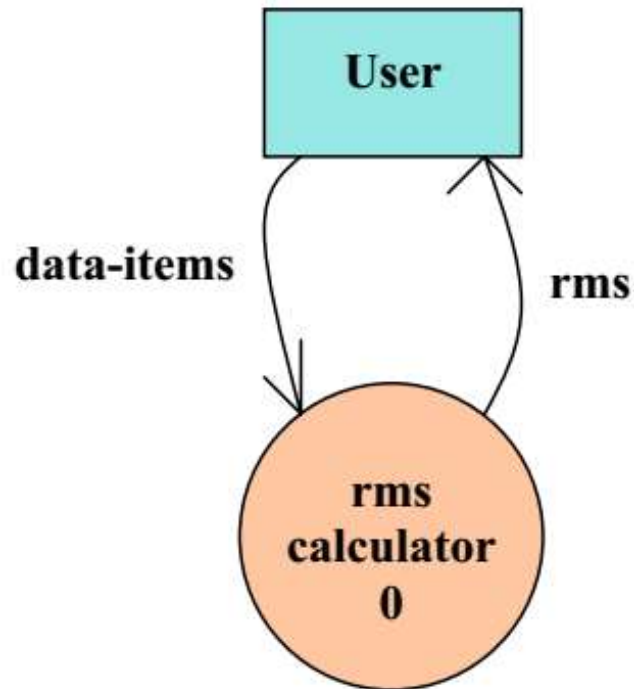
**☐ does not allow lower-level modules to invoke higher-level modules:**

**☐ But, two higher-level modules can invoke the same lower-level module.**

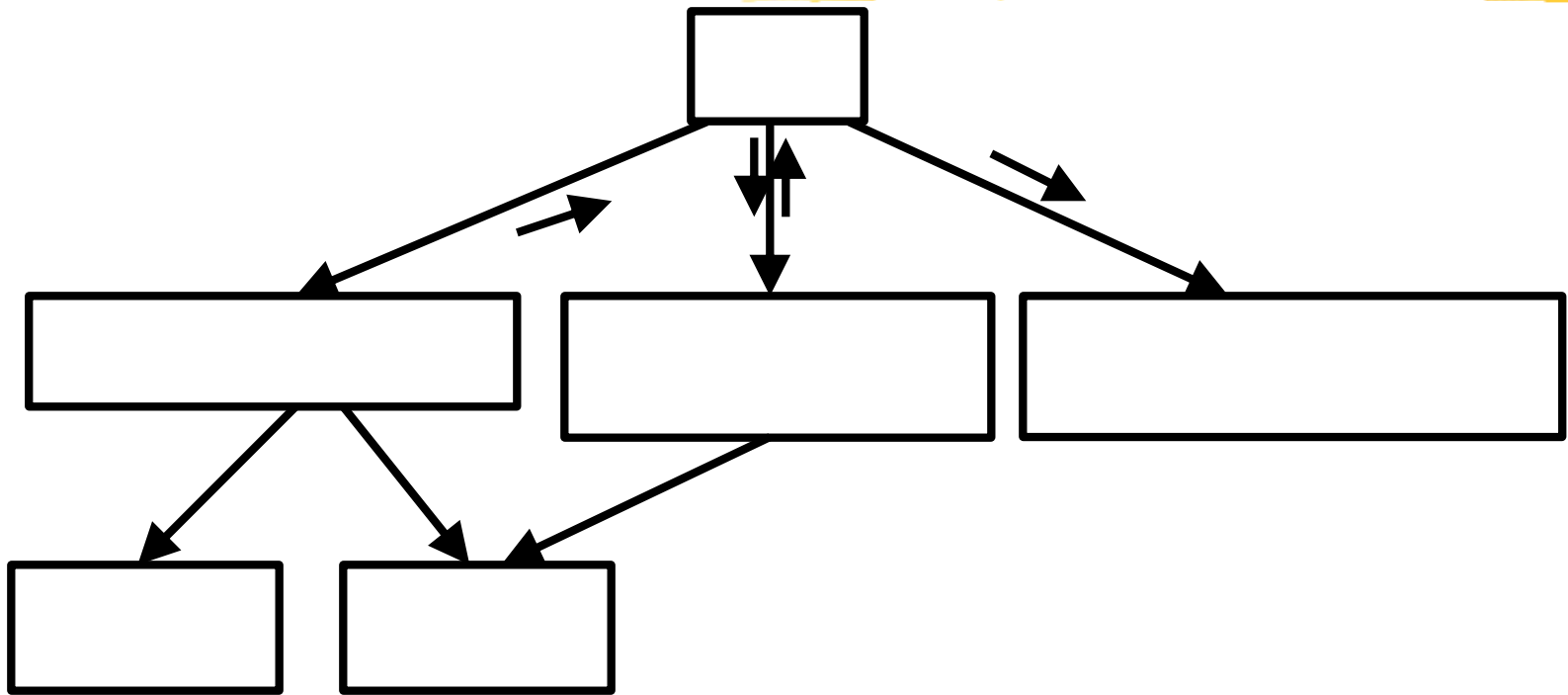
# RMS calculator

**RMS Calculator :**

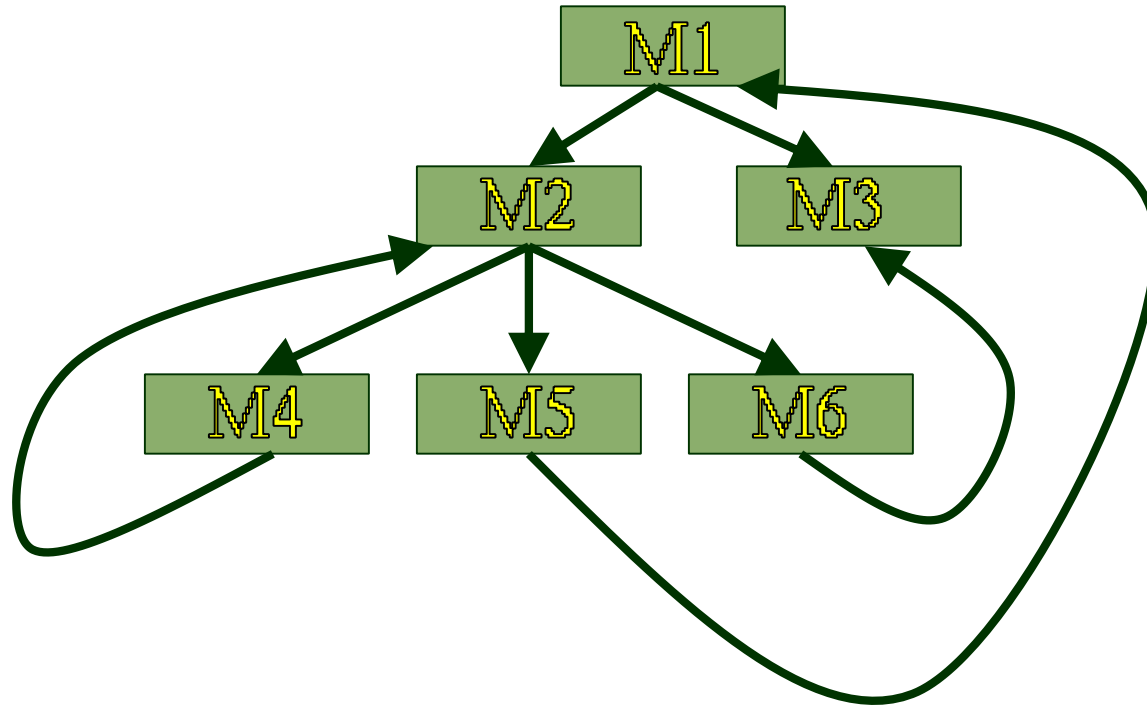
**Reads three integers and determine Root Mean Square**



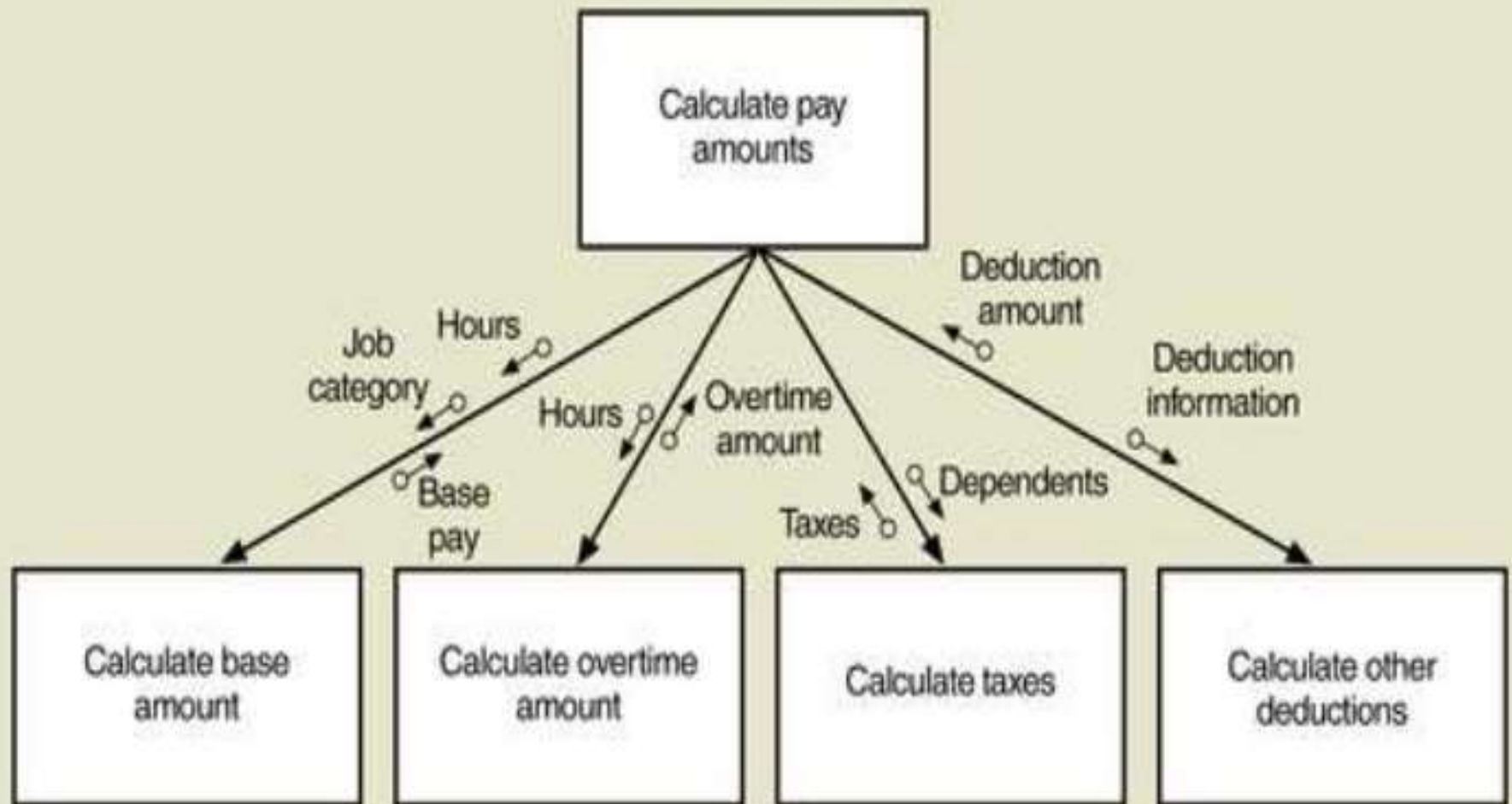
# Example



# Bad Design



# Structure Chart for Calculate Pay Amount Routine



# 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.



# Flow Chart (Aside)

⌘ We are all familiar with the flow chart representations:

⏏ Flow chart is a convenient technique to represent the flow of control in a system.

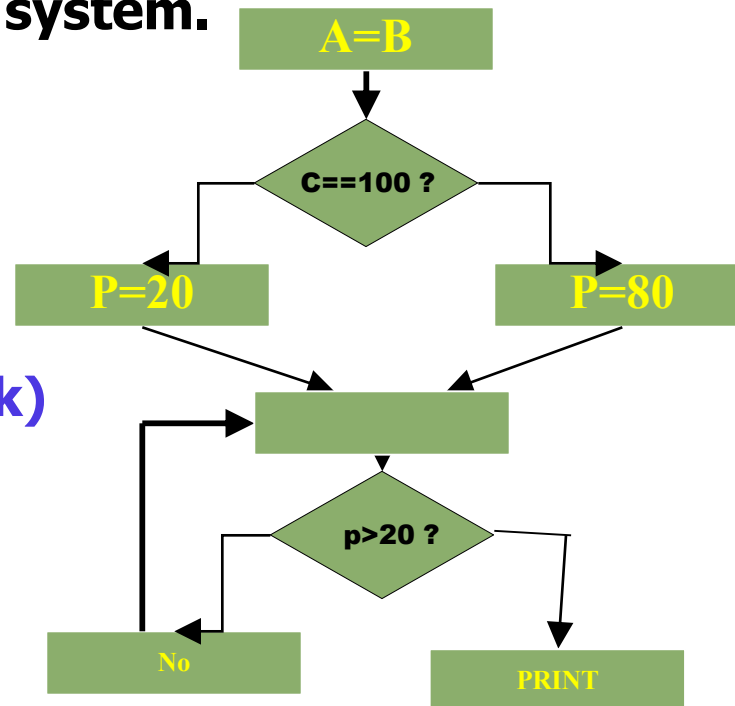
⌘ `if(c == 100)`

⌘ `P=20`

⌘ `else p= 80`

⌘ `while(p>20)`

⌘ `print(student mark)`



# Flow Chart versus Structure Chart

⌘ A structure chart differs from a flow chart in three principal ways:

☐ It is difficult to identify modules of a software from its flow chart representation.

☐ Data interchange among the modules is not represented in a flow chart.

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

# **Transformation of a DFD Model into Structure Chart**

**⌘ Two strategies exist to guide transformation of a DFD into a structure chart:**

**☒ Transform Analysis**

**☒ Transaction Analysis**

# Transform Analysis

⌘ The first step in transform analysis:

☐ Divide the DFD into 3 types of parts:

☒ input,

☒ logical processing,

☒ output.

# Transform Analysis

## ⌘ Input portion in the DFD:

- ☒ 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:

- ☒ called an afferent branch.

- ☒ Possible to have more than one afferent branch in a DFD.

# Transform Analysis

## ⌘ Output portion of a DFD:

- ☐ transforms output data from logical form to physical form.

  - ☒ e.g., from list or array into output characters.

- ☐ Each output portion:

  - ☒ called an efferent branch.

## ⌘ The remaining portions of a DFD

- ☐ called central transform

# Transform Analysis

**⌘ Derive structure chart by drawing one functional component for:**

**☐ the central transform,**

**☐ each afferent branch,**

**☐ each efferent branch.**

# Transform Analysis

⌘ **Identifying the highest level input and output transforms:**

☐ requires experience and skill.

⌘ **Some guidelines:**

☐ trace the inputs until a bubble is found whose output cannot be deduced from the inputs alone.

☐ Processes which validate input are not central transforms.

☐ Processes which sort input or filter data from it are.



# Transform Analysis

## ⌘ 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 subfunctions required by each high-level module.
- ☑ Many levels of modules may required to be added.

# Factoring

⌘ **The process of breaking functional components into subcomponents.**

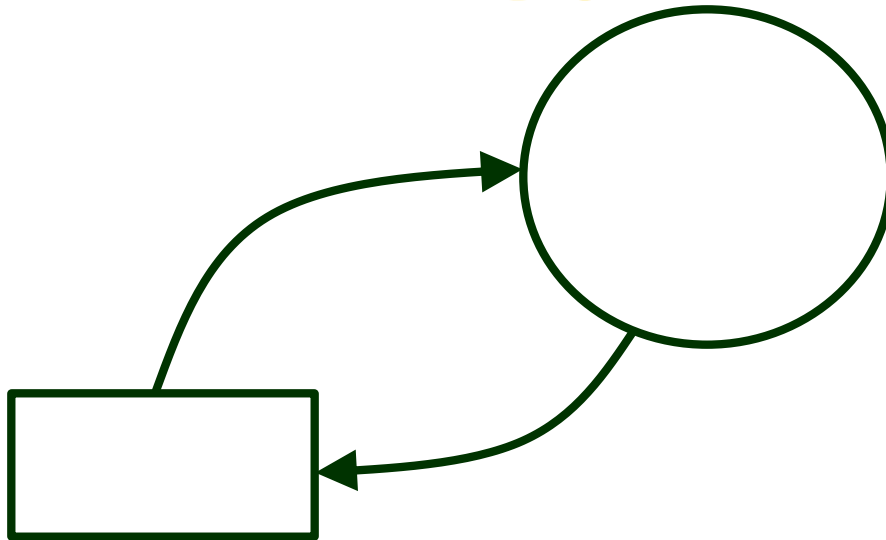
⌘ **Factoring includes adding:**

- ☐ **read and write modules,**
- ☐ **error-handling modules,**
- ☐ **initialization and termination modules, etc.**

⌘ **Finally check:**

- ☐ **whether all bubbles have been mapped to modules.**

# Example 1: RMS Calculating Software



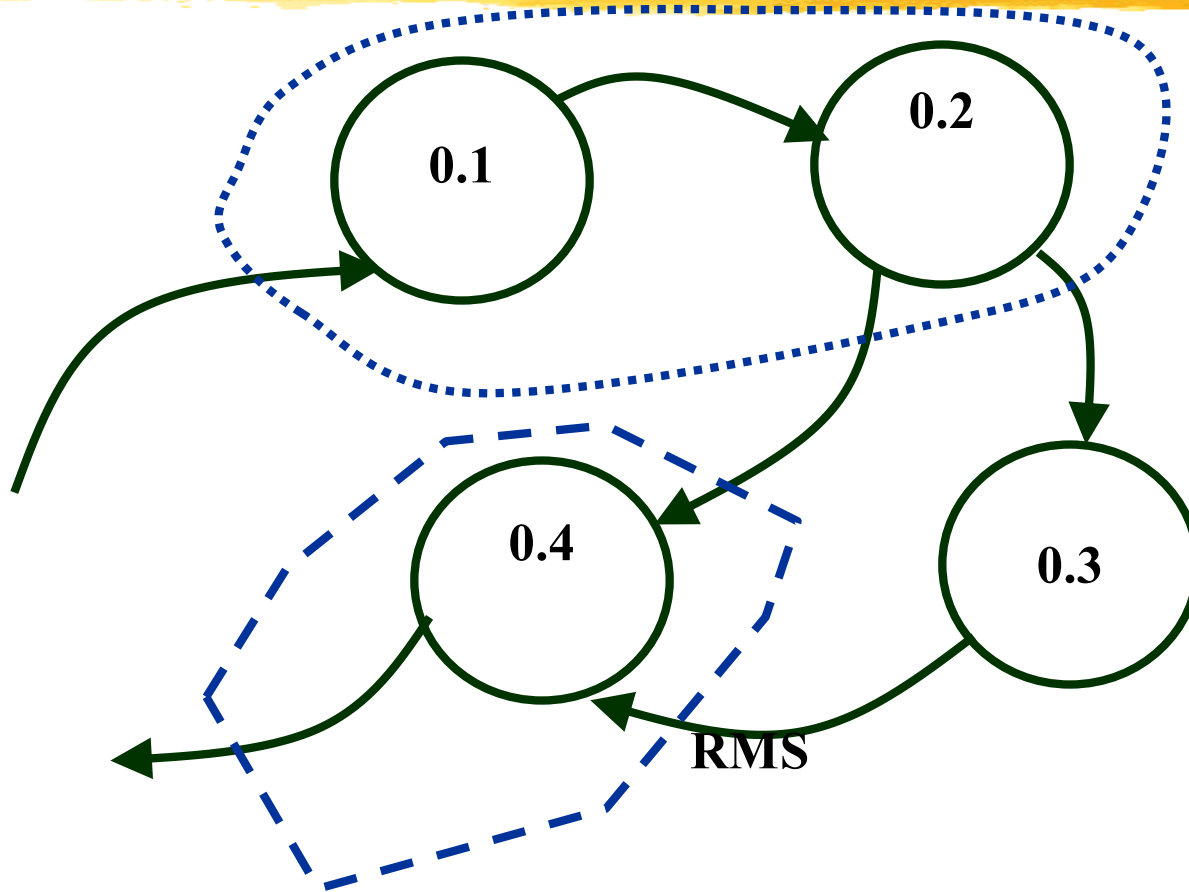
# **Example 1: RMS Calculating Software**

⌘ From a cursory analysis of the problem description,

☑ easy to see 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,
- ☒ display the result.

# Example 1: RMS Calculating Software



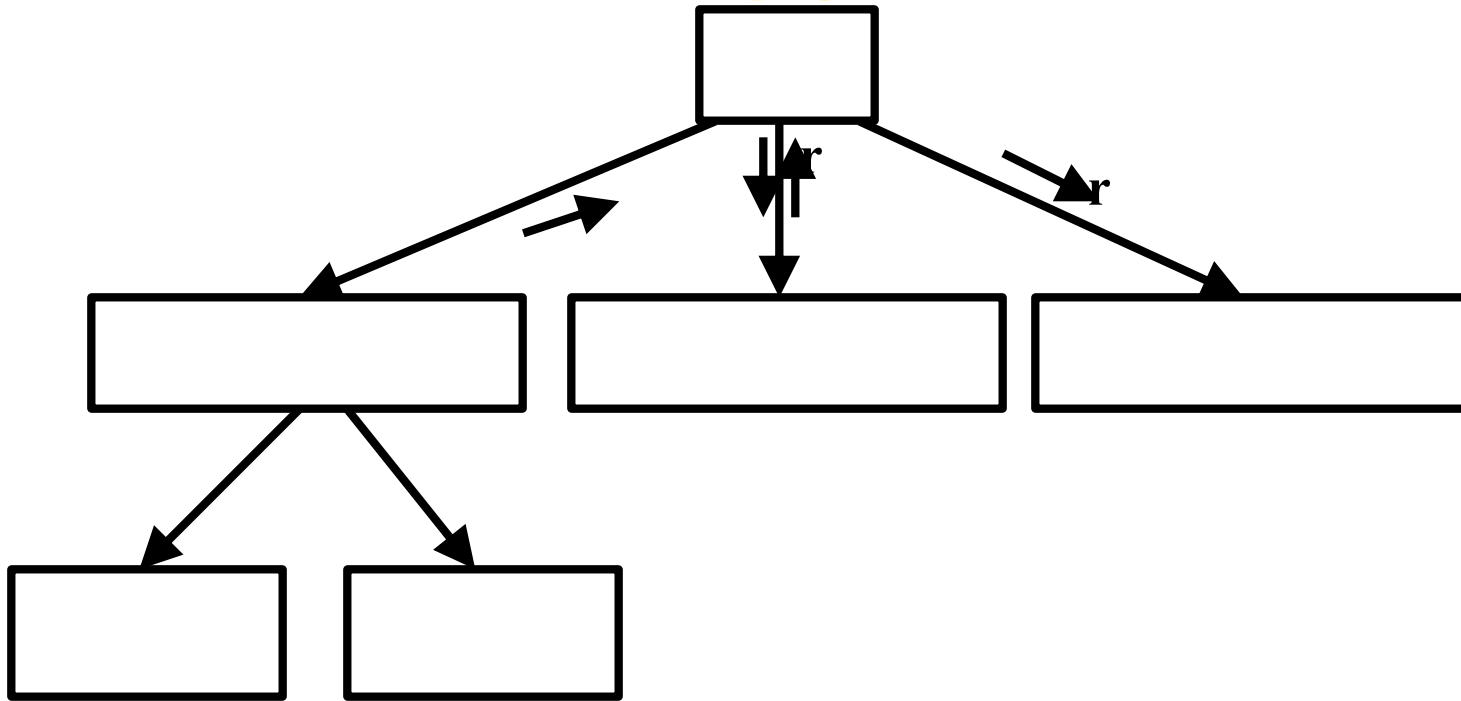
# **Example 1: RMS Calculating Software**

**⌘ By observing the level 1  
DFD:**

**☑ identify read-number and  
validate-number bubbles as  
the afferent branch**

**☑ display as the efferent  
branch.**

# Example 1: RMS Calculating Software

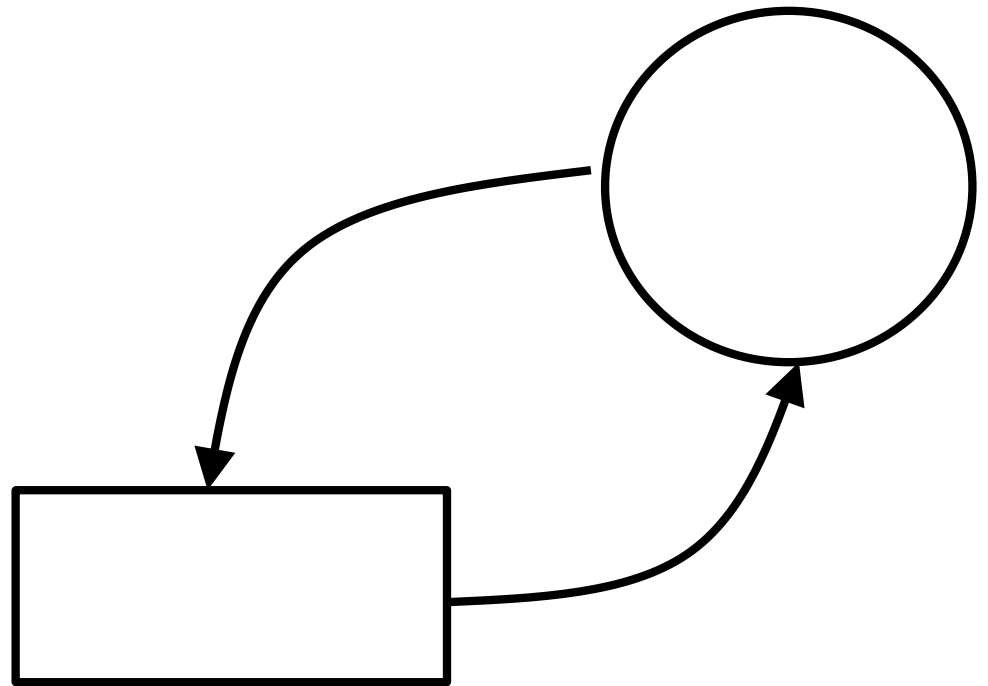


# **Example 2: Tic-Tac-Toe Computer Game**

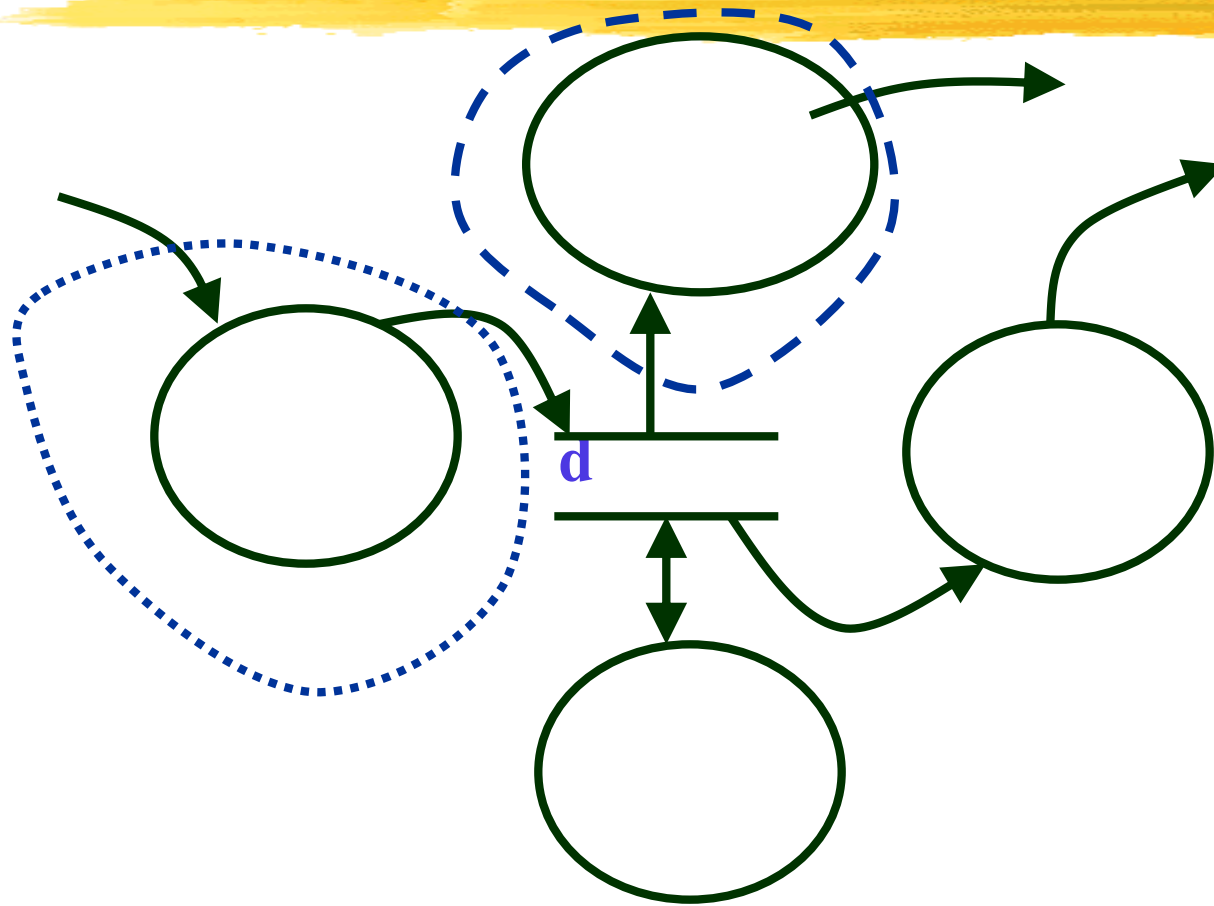
- ⌘ **As soon as either of the human player or the computer wins,**
  - ☑ **a message congratulating the winner should be displayed.**
- ⌘ **If neither player manages to get three consecutive marks along a straight line,**
  - ☑ **and all the squares on the board are filled up,**
  - ☑ **then the game is drawn.**
- ⌘ **The computer always tries to win a game.**



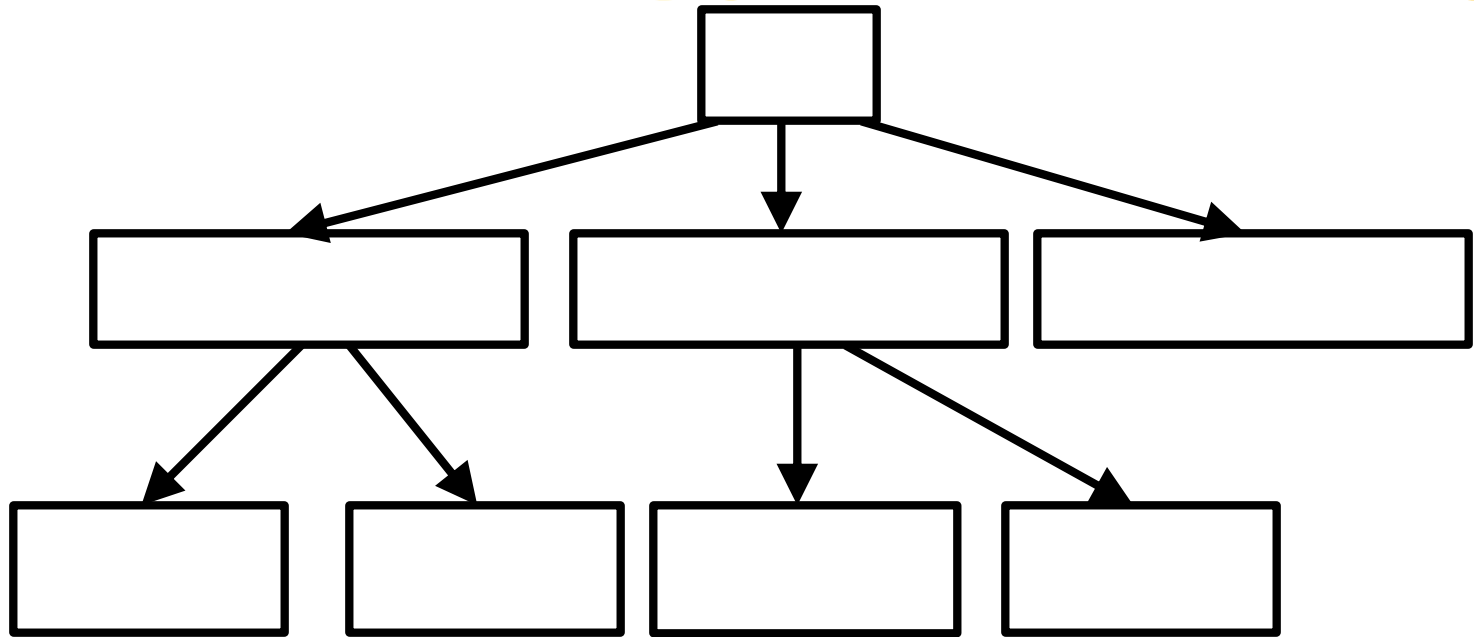
# Context Diagram for Example 2



# Level 1 DFD



# Structure Chart



# Transaction Analysis

⌘ Useful for designing transaction processing programs.

☐ Transform-centered systems:

☒ characterized by similar processing steps for every data item processed by input, process, and output bubbles.

☐ Transaction-driven systems,

☒ one of several possible paths through the DFD is traversed depending upon the input data value.

# Transaction Analysis

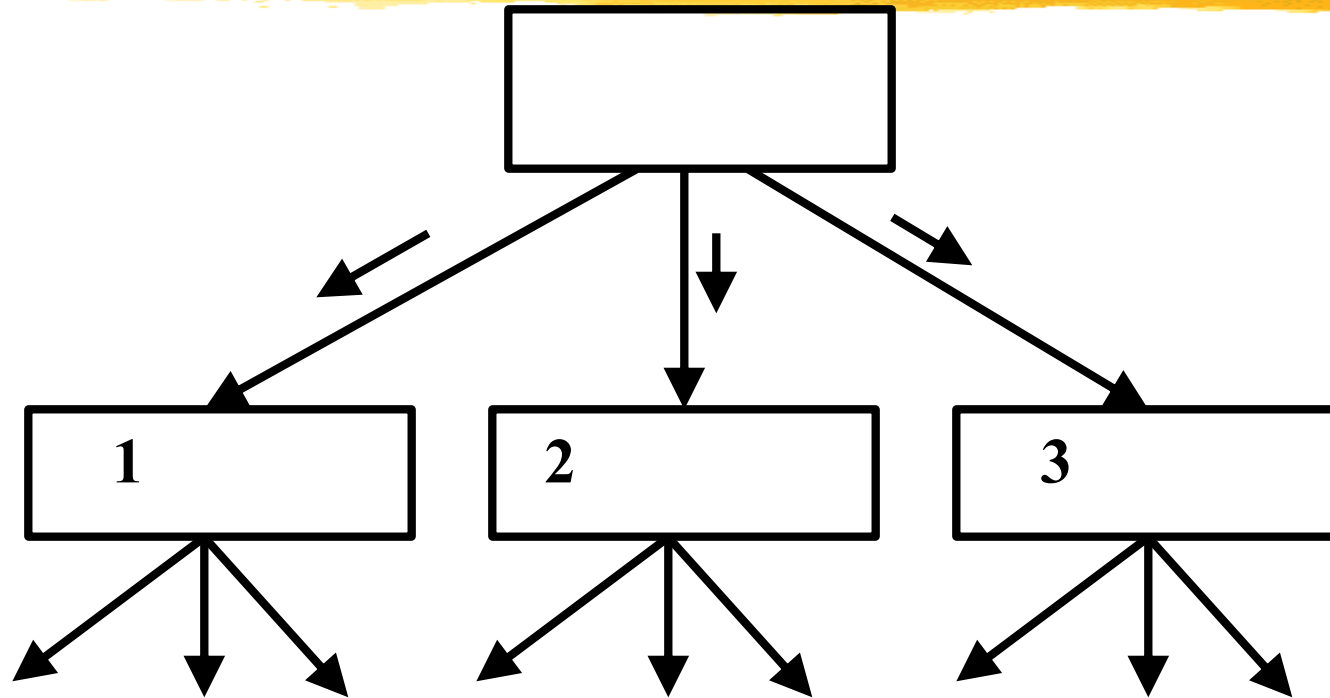
## ⌘ Transaction:

- ☒ any input data value that triggers an action:
- ☒ For example, selected menu options might trigger different functions.
- ☒ Represented by a tag identifying its type.

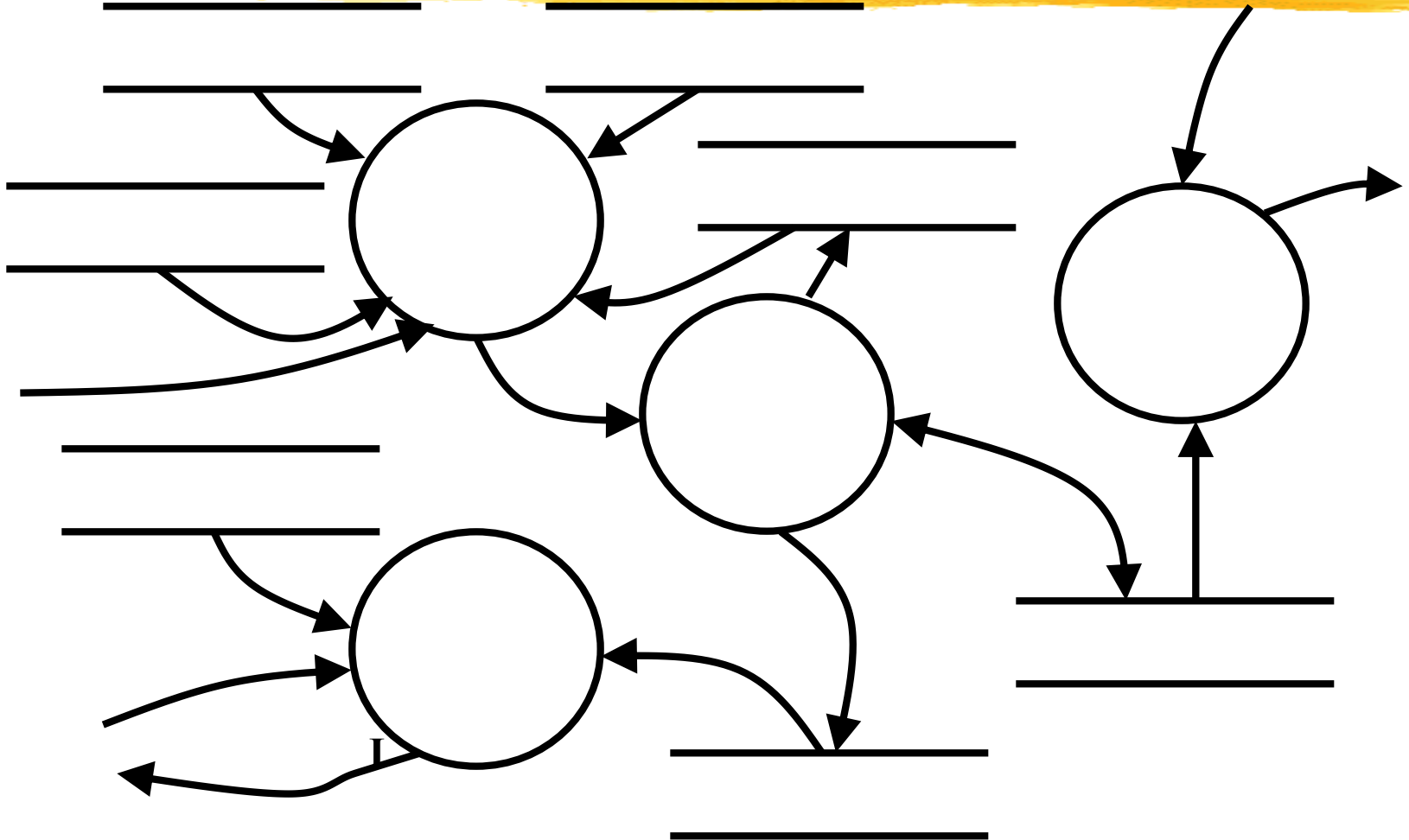
## ⌘ Transaction analysis uses this tag to divide the system into:

- ☒ several transaction modules
- ☒ one transaction-center module.

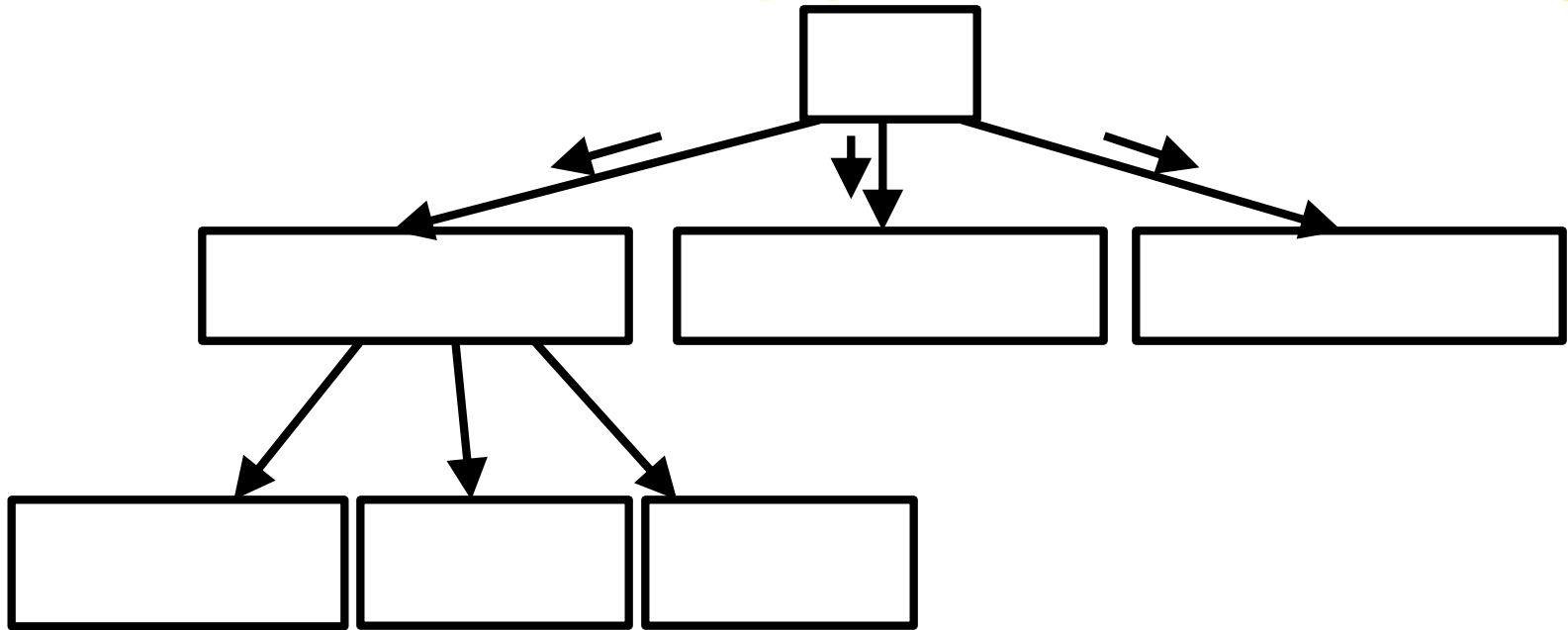
# Transaction analysis



# Level 1 DFD for TAS



# Structure Chart





# Summary

- ⌘ We first discussed structured analysis of a larger problem.
- ⌘ We defined some general guidelines
  - ☐ for constructing a satisfactory DFD model.
- ⌘ The DFD model though simple and useful
  - ☐ does have several short comings.
- ⌘ We then started discussing structured design.

# Summary

## ⌘ Aim of structured design:

- ☑ transform a DFD representation into a structure chart.

## ⌘ Structure chart represents:

- ☑ module structure
- ☑ interaction among different modules,
- ☑ procedural aspects are not represented.

# Summary



**⌘ Structured design provides two strategies to transform a DFD into a structure chart:**

**☑ Transform Analysis**

**☑ Transaction Analysis**

# Summary

- ⌘ **We Discussed three examples of structured design.**
- ⌘ **It takes a lot of practice to become a good software designer:**
  - ☑ **Please try to solve all the problems listed in the Exercise section of Text Book**