System Analysis and Design

Eighth Edition

Alan Dennis, Barbara Wixom, Roberta M. Roth

Chapter 9

Program Design

Objectives

- Be able to revise logical DFDs into physical DFDs.
- Be able to create a structure chart.
- Be able to write a program specification.
- Be able to write instructions using pseudocode.
- Become familiar with event-driven programming.

Introduction

1. Various implementation decisions will be made about the new system, such as what programming language(s) will be used

2. The data flow diagrams (DFDs) created during analysis are modified to show these implementation decisions, resulting in a set of physical DFDs

3. The analysts then determine how the processes of the system will be organized, using a structure chart to depict their decisions

4. Detailed instructions called program specifications are developed so that during construction, the programmers know exactly what they should be creating

Moving from Logical to Physical Process Models

- During analysis, the systems analysts identified the processes and data flows that are needed to support the functional requirements of the new system

- These processes and data flows are described on the logical DFDs

- The logical DFDs do not contain any indication of how the system will actually be implemented

- During design, physical process models are created to show implementation details and explain how the final system will work

The Physical Data Flow Diagram

- The physical data flow diagram (DFD) contains the same components as the logical DFD

- The difference between the two models is that a physical DFD contains additional details that describe how the system will be built

- There are five steps to perform to make the transition to the physical DFD

# Steps to Create the Physical Data Flow Diagram

Step	Explanation
1. Add implementation references.	Using the existing logical DFD, add the way in which the data stores, data flows, and processes will be implemented to each component.
2. Draw a human–machine boundary.	Draw a line to separate the automated parts of the system from the manual parts.
3. Add system-related data stores, data flows, and processes.	Add system-related data stores, data flows, and processes to the model (components that have little to do with the business process).
4. Update the data elements in the data flows.	Update the data flows to include system-related data elements.
5. Update the metadata in the CASE repository.	Update the metadata in the CASE repository to include physical characteristics.

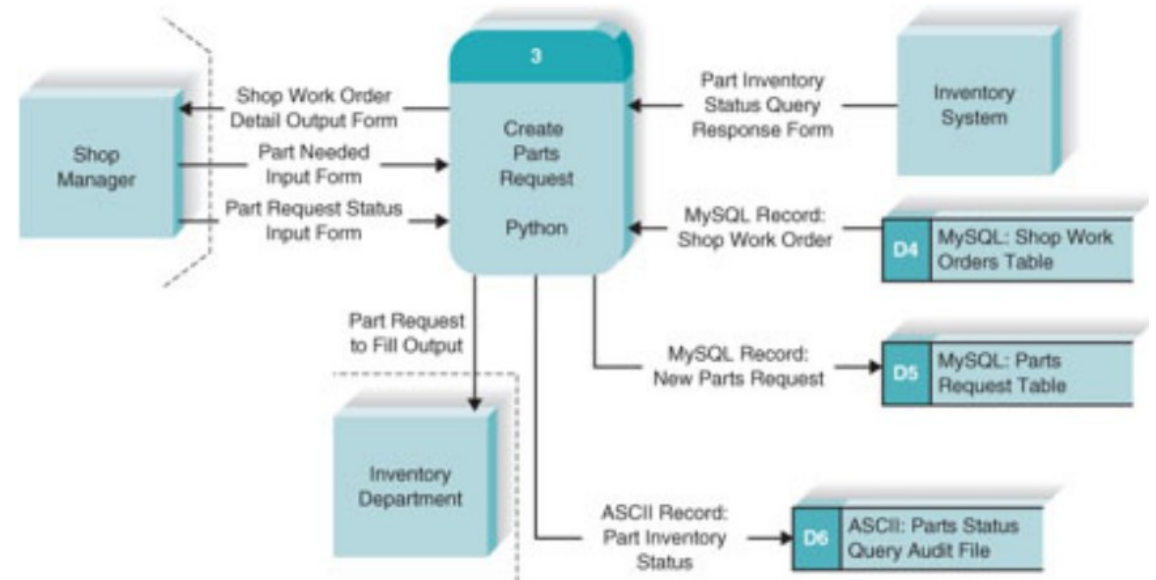# Physical DFD for Shop Manager Approval of Custom Drone Order Level 1 DFD
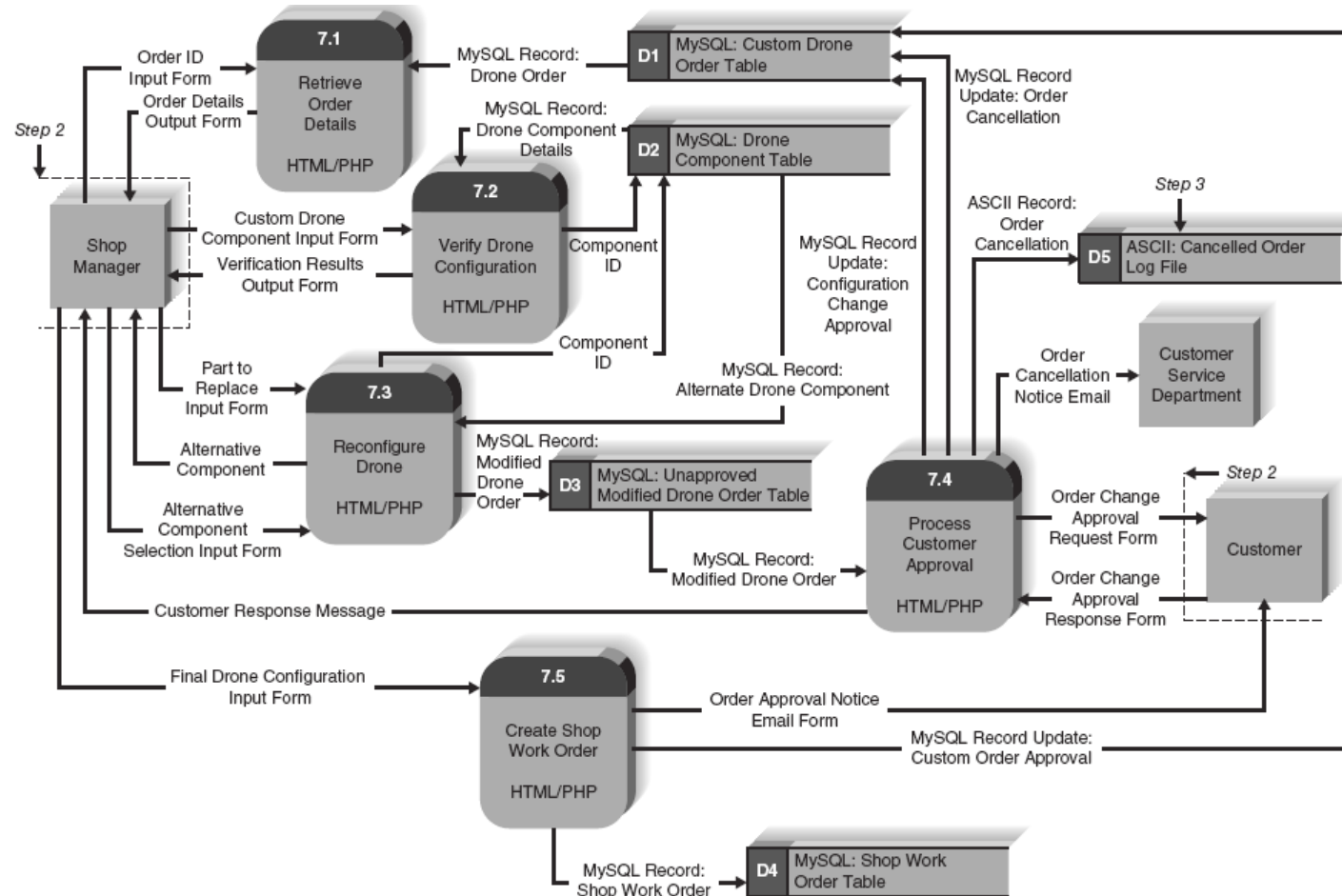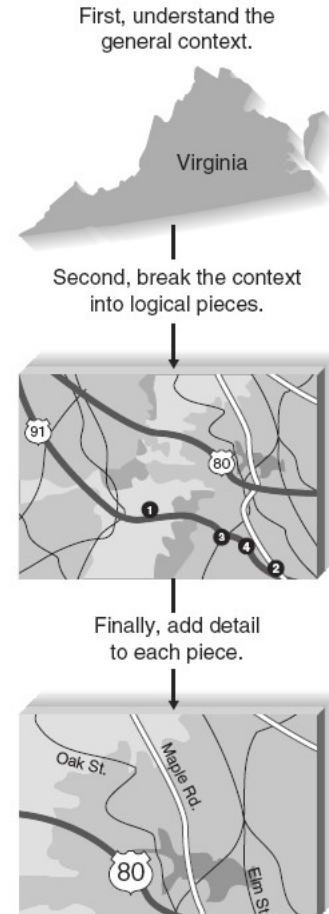
Contrasting a Logical and Physical DFD of DrōnTeq's
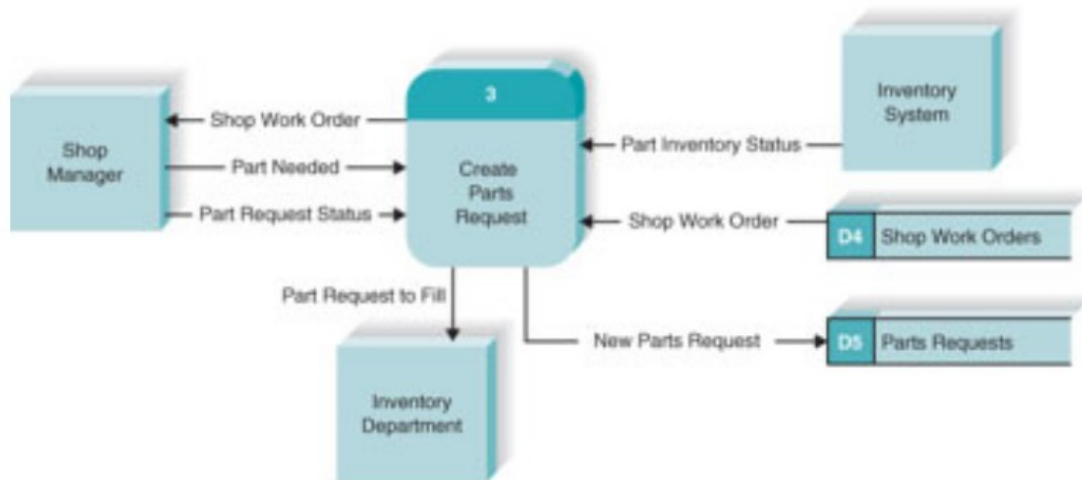
Designing Programs

- **Program design** - creating instructions for the programmers
- The **top-down, modular approach** - begin with the "big picture" and gradually add detail
- **Program design document** – all structure charts and specifications needed by programmers to implement the system

Using a Top-Down Modular Approach

First, understand the general context.

Virginia

Second, break the context into logical pieces.

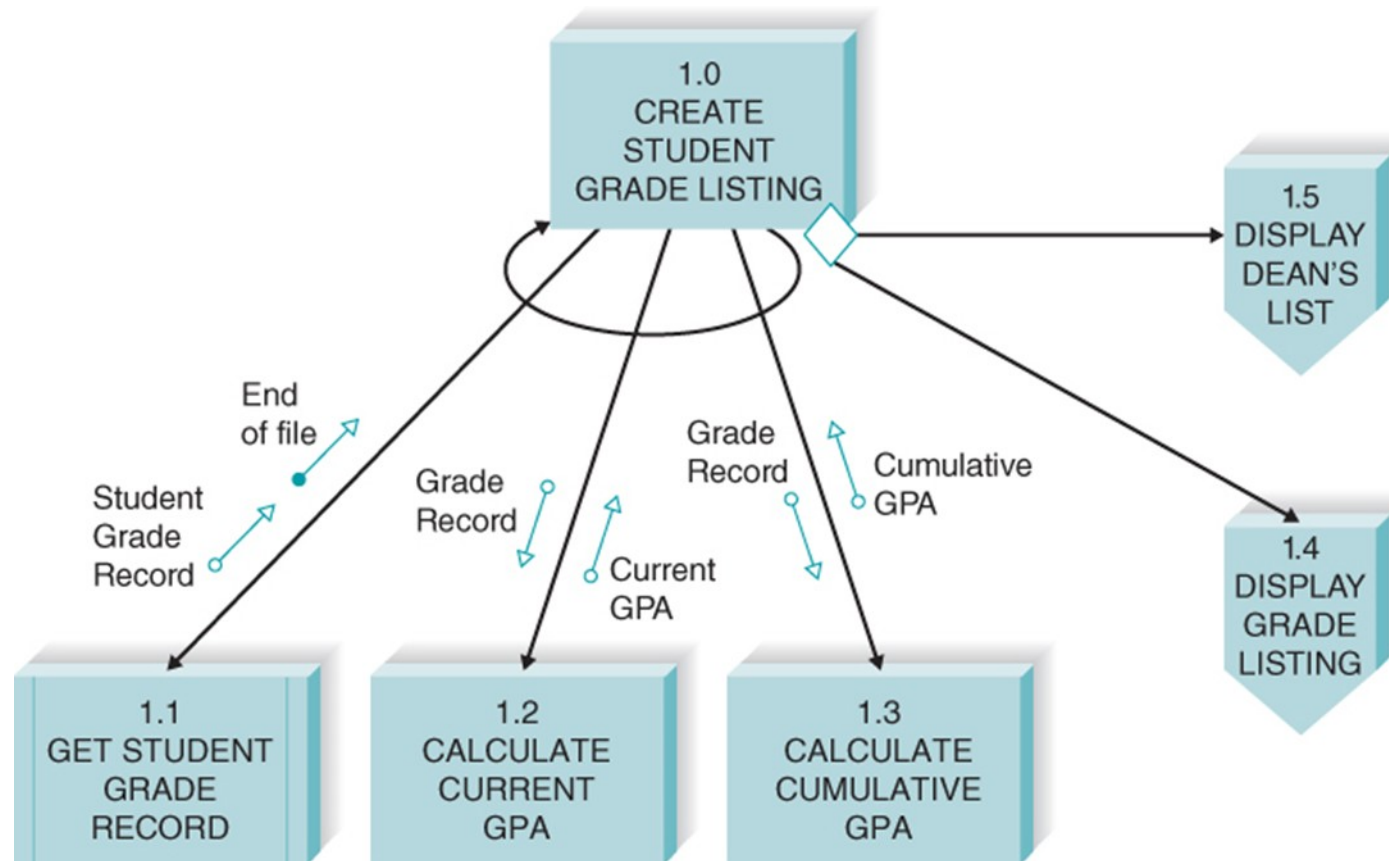Finally, add detail to each piece.

Good Program Design

- First, analysts create a high- level diagram that shows the various components of a program, how the components should be organized, and how the components interrelate
  - Known as the structure chart
- Next, program specifications are written to describe exactly what needs to be included in each program module
  - Pseudocode is a technique similar to structured English that is used to communicate what needs to be written
- At the end of program design, the project team compiles the program design document, which includes all of the structure charts and program specifications
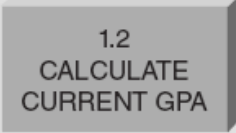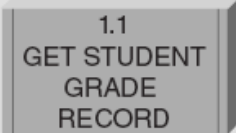
Structure Chart

- Important program design technique
- Shows all components of code in a hierarchical format
  - Sequence
  - Selection
  - Iteration
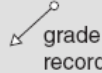- Illustrates the organization and interactions of the different program modules

Example Structure Chart

Structure Chart Elements

Structure Chart Element	Purpose	Symbol
Every *module*: • Has a number. • Has a name. • Is a control module if it calls other modules below it. • Is a subordinate module if it is controlled by a module at a higher level.	Denotes a logical piece of the program	1.2 CALCULATE CURRENT GPA
Every *library module* has: • A number. • A name. • Multiple instances within a diagram.	Denotes a logical piece of the program that is repeated within the structure chart	1.1 GET STUDENT GRADE RECORD
A *loop*: • Is drawn with a curved arrow. • Is placed around lines of one or more modules that are repeated.	Communicates that a module(s) is repeated	
A *conditional line*: • Is drawn with a diamond. • includes modules that are invoked on the basis of some condition.	Communicates that subordinate modules are invoked by the control module based on some condition	

Structure Chart Elements Continued

Structure Chart Element	Purpose	Symbol
A *data couple*: • Contains an arrow. • Contains an empty circle. • Names the type of data that are being passed. • Can be passed up or down. • Has a direction that is denoted by the arrow.	Communicates that data are being passed from one module to another	⟋○ grade record
A *control couple*: • Contains an arrow. • Contains a filled-in circle. • Names the message or flag that is being passed. • Should be passed up, not down. • Has a direction that is denoted by the arrow.	Communicates that a message or a system flag is being passed from one module to another	⟋● end of file
An *off-page connector*: • Is denoted by the hexagon. • Has a title. • Is used when the diagram is too large to fit everything on the same page.	Identifies when parts of the diagram are continued on another page of the structure chart	PRINT GRADE LISTING
An *on-page connector*: • Is denoted by the circle. • Has a title. • Is used when the diagram is too large to fit everything in the same spot on a page.	Identifies when parts of the diagram are continued somewhere else on the same page of the structure chart	PRINT GRADE LISTING

Syntax: Module

- A structure chart is composed of modules
  - These are lines of program code that perform a single function
  - The modules are depicted by a rectangle and connected by lines
- A control module is a higher-level component that contains the logic for performing other modules
  - The components that it calls and controls are considered subordinate modules
- At times, modules are standardized and used in many places throughout the system
  - These are called library modules
- The lines that connect the modules communicate the passing of control

Syntax: Module Continued

- There are two symbols that describe special types of control that can appear on the structure chart
  - The curved arrow, or loop, indicates that the execution of some or all subordinate modules is repeated
  - A conditional line (depicted by a diamond) denotes that execution of one or more of the subordinate modules occurs in some cases but not in others
- A circle is used to connect parts of the structure chart when there are space constraints and a diagram needs to be continued on another part of the page

Syntax: Couples

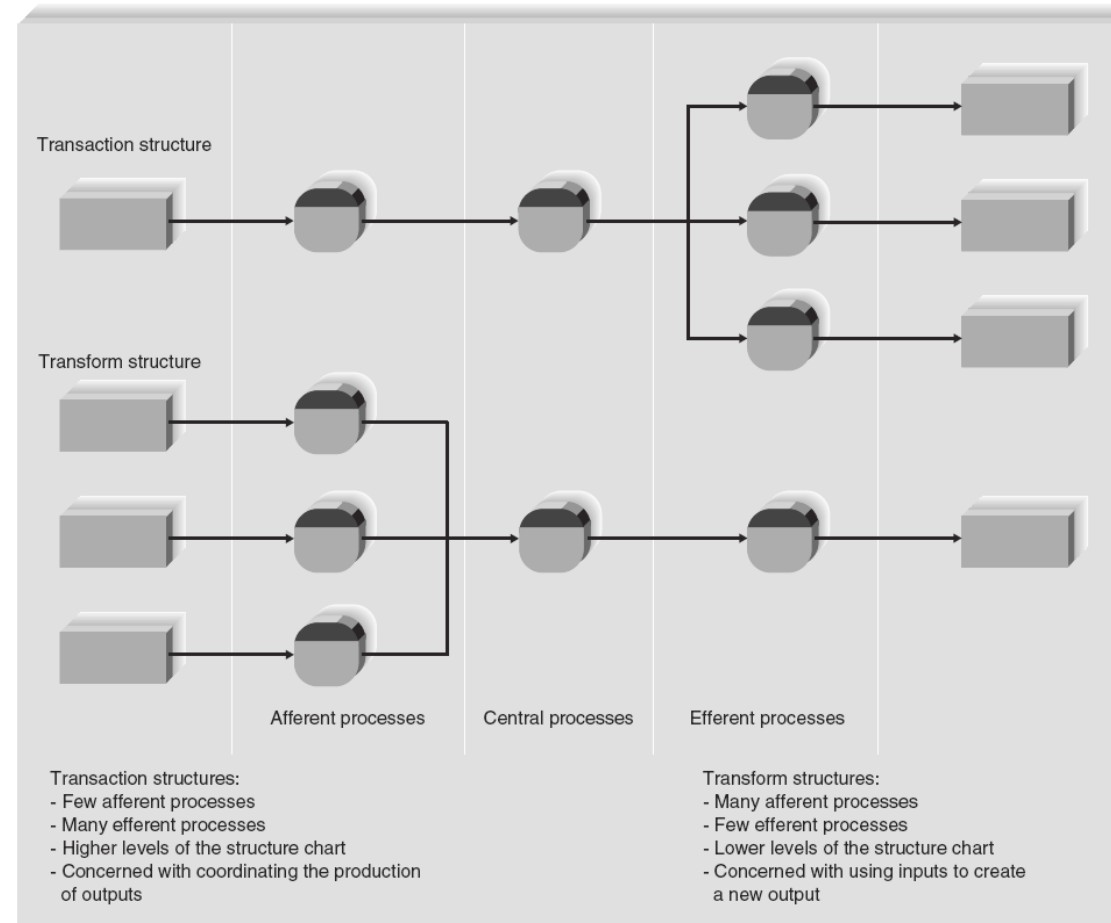- Couples, shown by arrows, are drawn on the structure chart to show that information is passed between modules
  - Arrowhead shows direction
- Control couples, drawn with the use of arrows with filled-in circles, are used to pass parameters or system-related messages back and forth among modules
- In general, control flags should be passed from subordinates to control modules, but not the other way around

Building the Structure Chart

- Processes in the DFD tend to represent one module on the structure chart
  - o Afferent processes – provide inputs to system
  - o Central processes – perform critical system operations
  - o Efferent processes – handle system outputs
- Each process of a DFD tends to represent one module on the structure chart
- The DFD leveling can correspond to the structure chart hierarchy
- The difficulty comes when determining how the components on the structure chart should be organized

Transform and Transaction Structures

Transaction structure

Transform structure

Afferent processes   Central processes   Efferent processes

Transaction structures:
- Few afferent processes
- Many efferent processes
- Higher levels of the structure chart
- Concerned with coordinating the production of outputs

Transform structures:
- Many afferent processes
- Few efferent processes
- Lower levels of the structure chart
- Concerned with using inputs to create a new output

Transaction Structure

- Transaction structure – control module calls subordinate modules, each of which handles a particular transaction

- Few afferent processes

- Many efferent processes

- Higher up levels of structure chart

- Concerned with coordinating the production of outputs
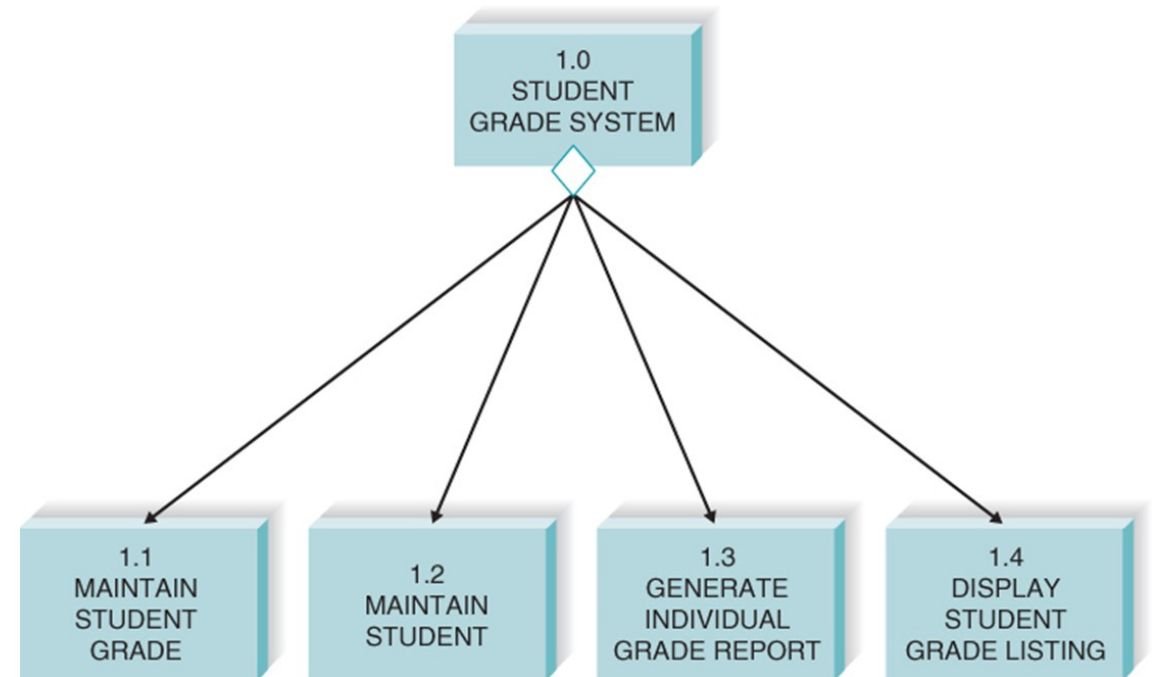
Transform Structure

- *Transform* structure – control module calls several subordinate modules in sequence
- Each subordinate performs a step in a process that transforms an input into an output
  - Many afferent processes
  - Few efferent processes
  - Lower levels of structure chart
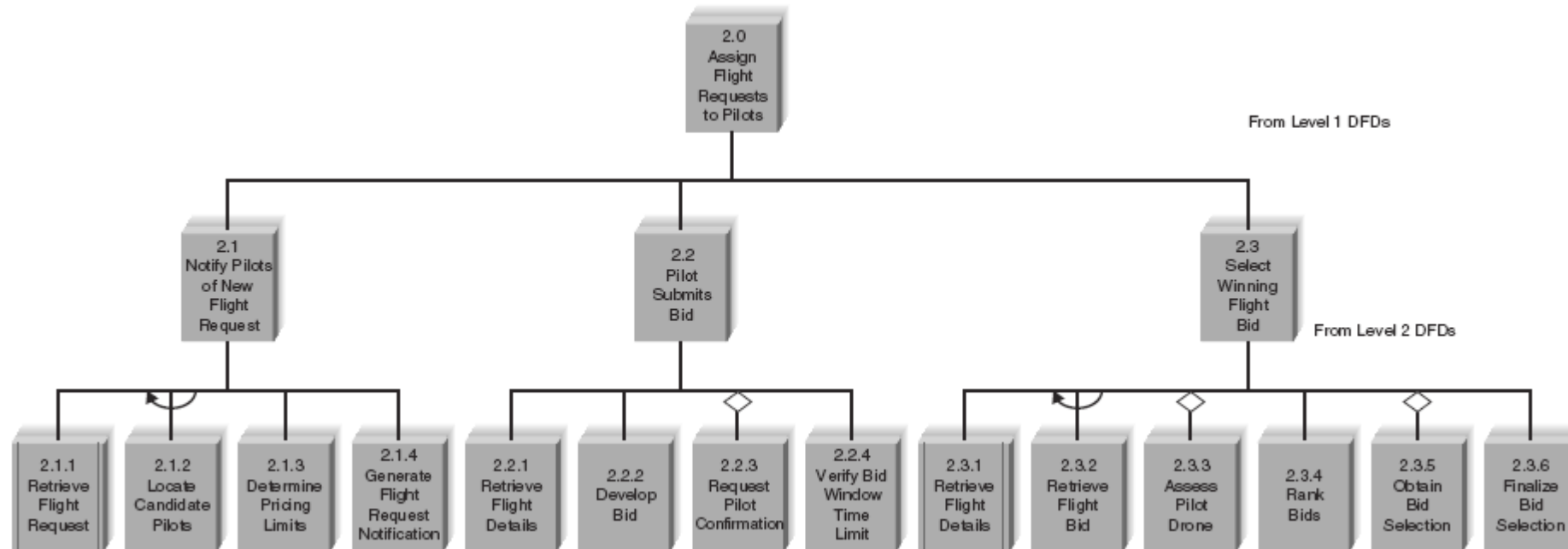 - Concerned with using inputs to create a new output
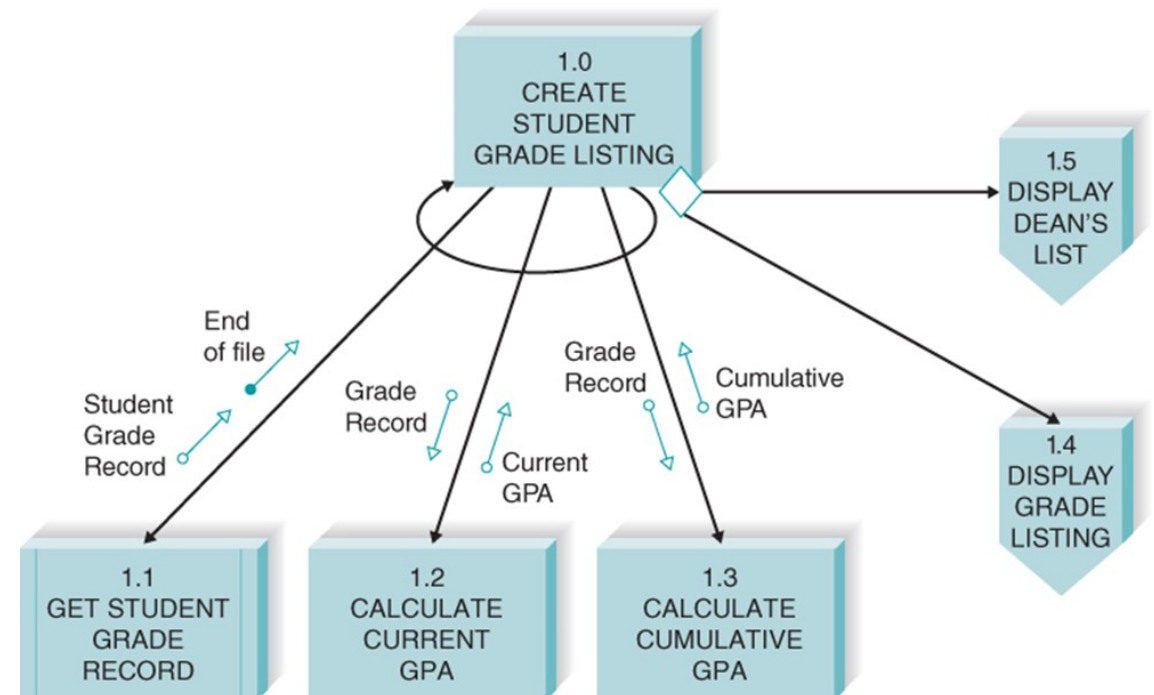
Steps in Building the Structure Chart

1. Identify top level modules and decompose them into lower levels
2. Add control connections
3. Add couples
4. Review and revise again and again until complete
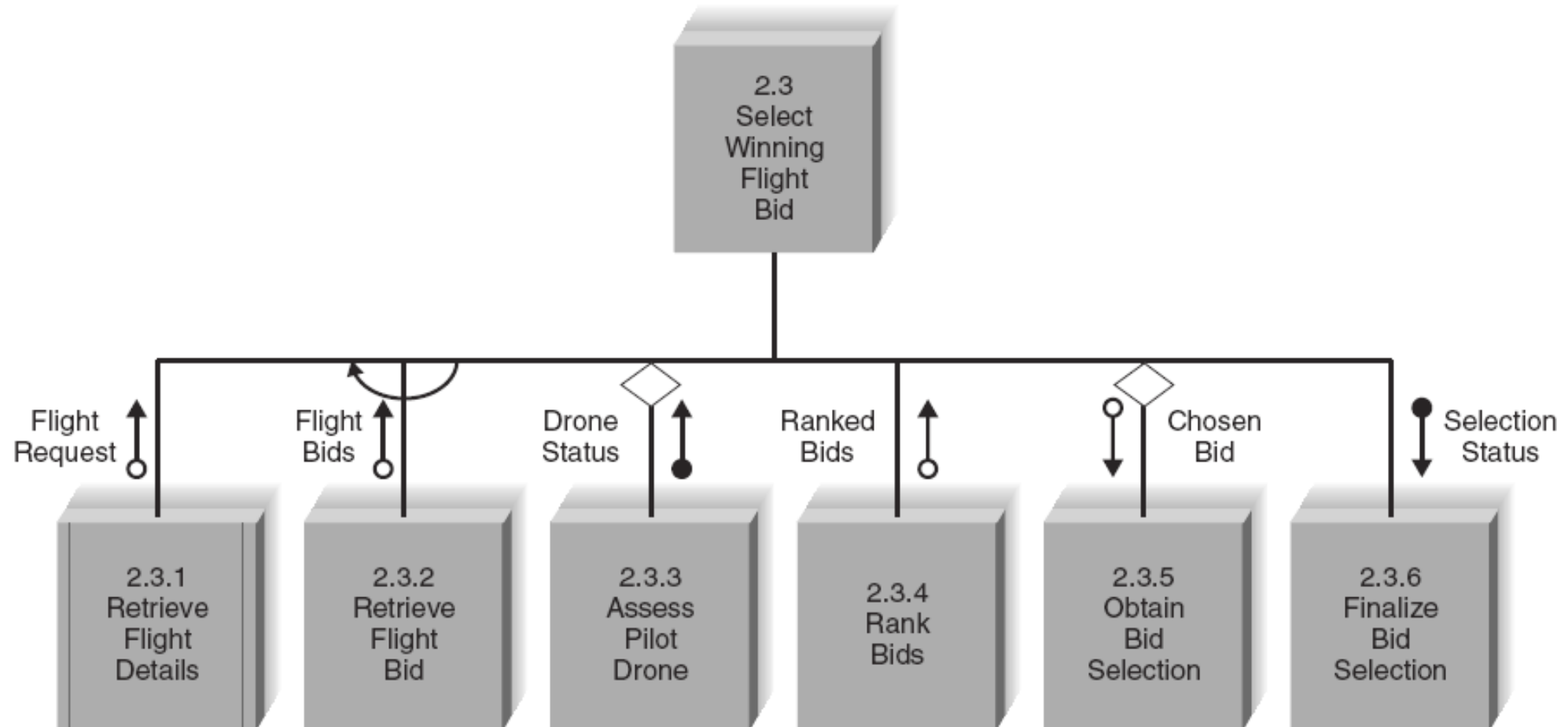
Step 1: Identify Modules and Levels for the Structure Chart
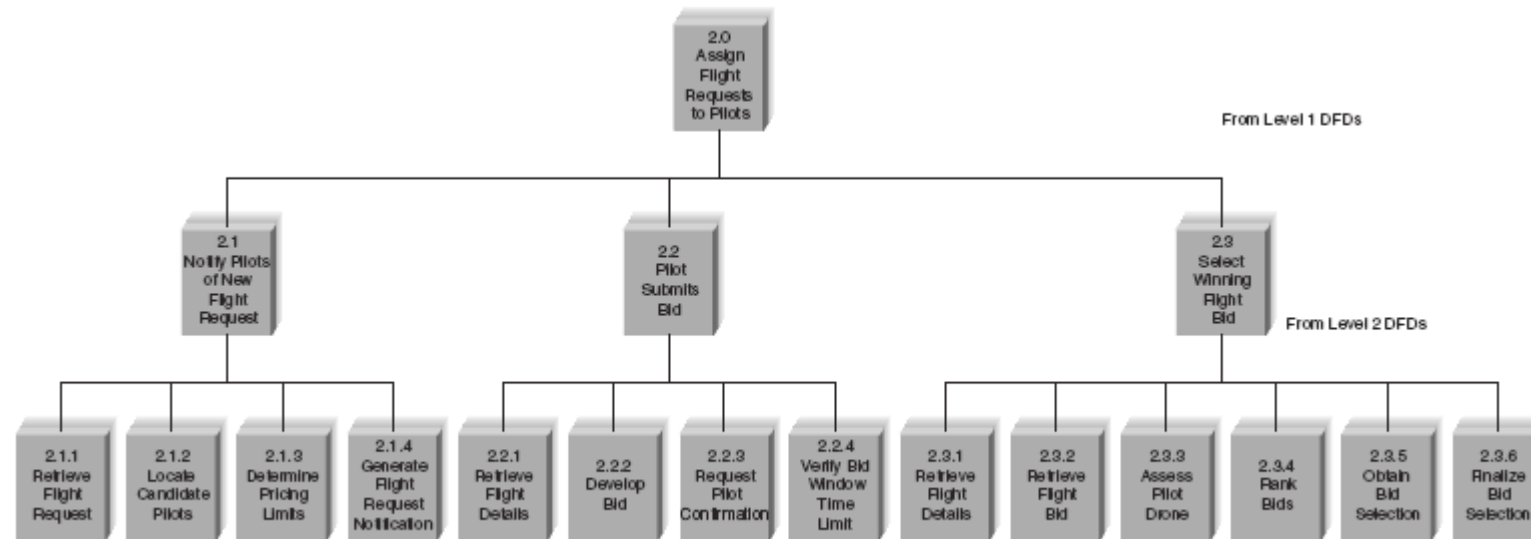
Step 2: Add Special Connections to the Structure Chart

Step 3: Add Couples to the Structure Chart

Design Guidelines

- High quality structure charts result in programs that are modular, reusable and easy to implement

- Measures include:

  - Cohesion

  - Coupling

  - Appropriate levels of fan-in and fan-out

Build Modules with High Cohesion

- Cohesion refers to how well the lines of code within each structure chart module relate to each other

- A module should perform only one task, making it highly cohesive

- There are various types of cohesion, some of which are better than others

Types of Cohesion from Good to Bad

Type of Cohesion	Definition
Functional	Module performs one problem-related task.
Sequential	Output from one task is used by the next.
Communicational	Elements contribute to activities that use the same inputs or outputs.
Procedural	Elements are performed in sequence but do not share data.
Temporal	Activities are related in time.
Logical	List of activities; which one to perform is chosen outside of module.
Coincidental	No apparent relationship.

Cohesion Decision Tree
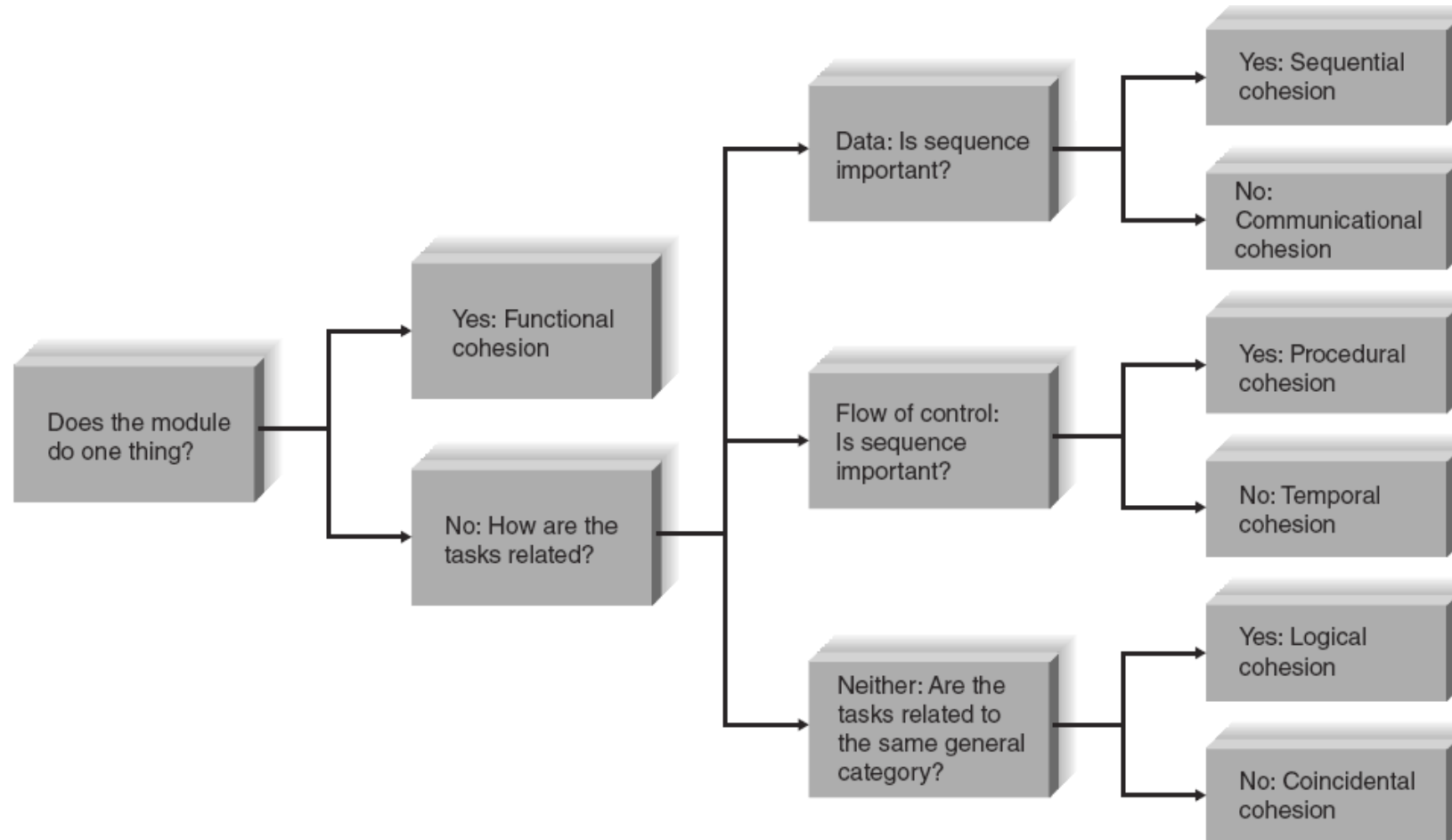
Factoring

- Process of dealing with "low" cohesion by separating out a function from one module into a module of its own
- Separates tasks into different modules
- Reduces use of control flags

Build Loosely Coupled Modules

- Coupling involves how closely modules are interrelated
- The second guideline for good structure chart design states that modules should be loosely coupled
  - That way, modules are independent from each other
- The numbers and kinds of couples on the structure chart reveal the presence of coupling between modules

Types of Coupling from Good to Bad

Type of Coupling	Definition
Data	Modules pass fields of data or messages.
Stamp	Modules pass record structures.
Control	Module passes a piece of information that intends to control logic.
Common	Modules refer to the same global data area.
Content	Module refers to the inside of another module.

Fan-In / Fan-Out

- Fan-in describes the number of control modules that communicate with a subordinate
  - A module with high fan-in has many different control modules that call it
  - This is a very good situation
  - Structures with high fan-in promote the reusability of modules and make it easier for programmers to recode
- We want to avoid a large number of subordinates associated with a single control (fan-out)

High Fan-In (a) and Low Fan-in (b)

(a)

(b)
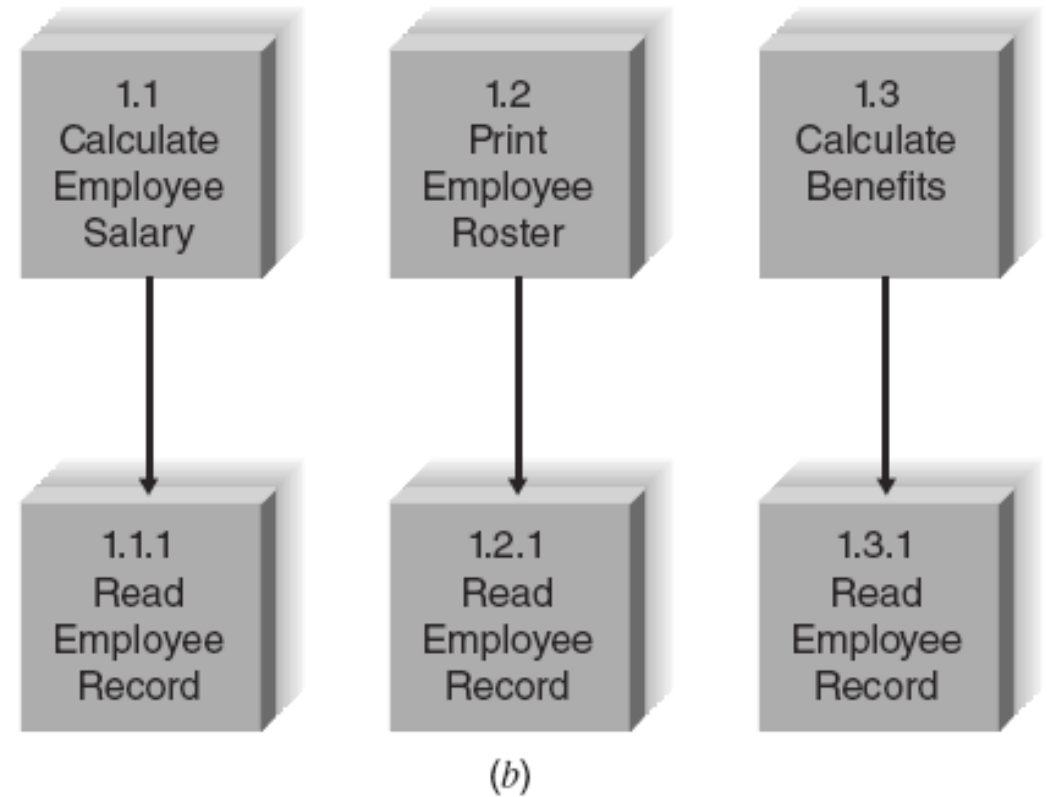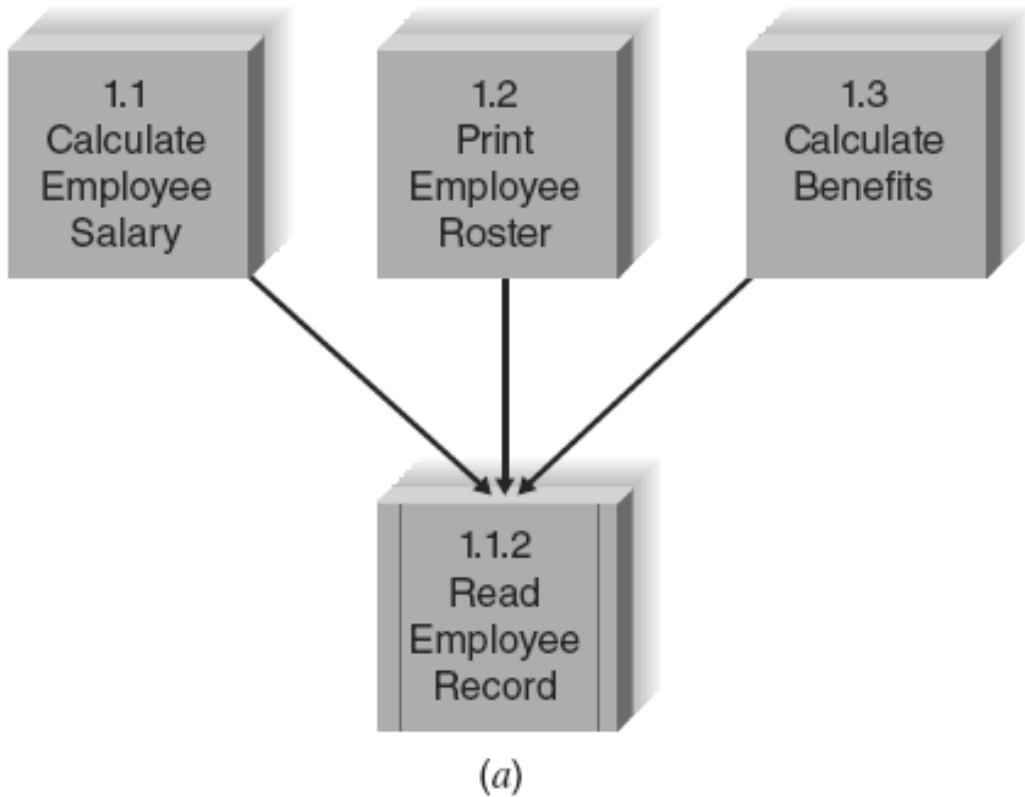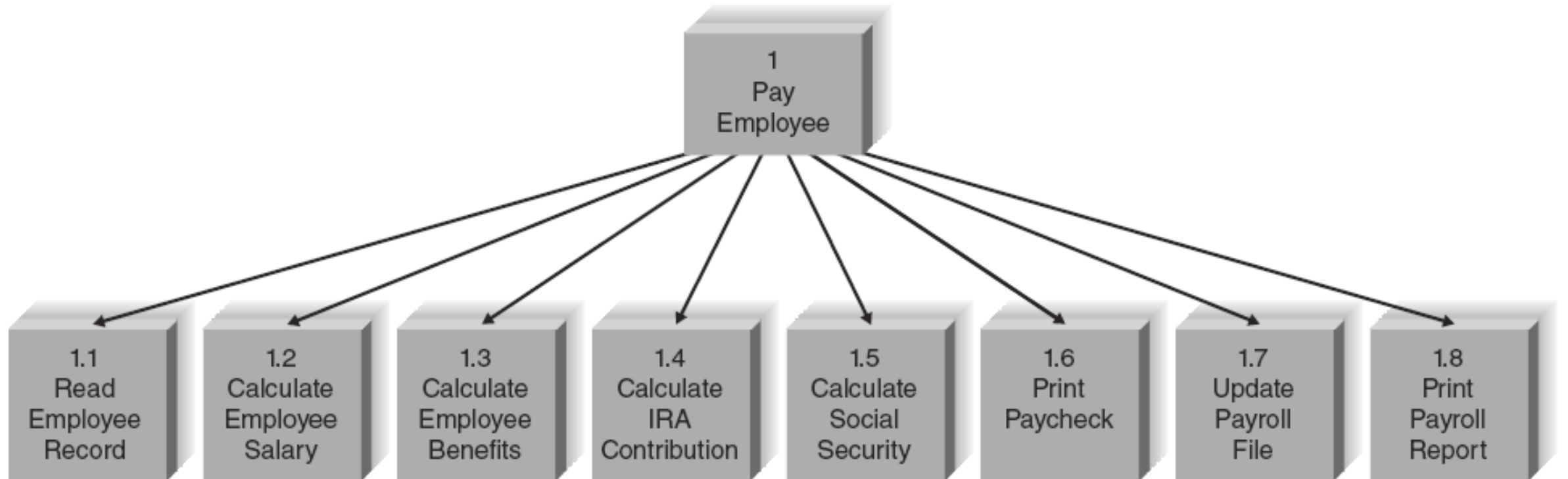
High Fan-Out

Low Fan-Out
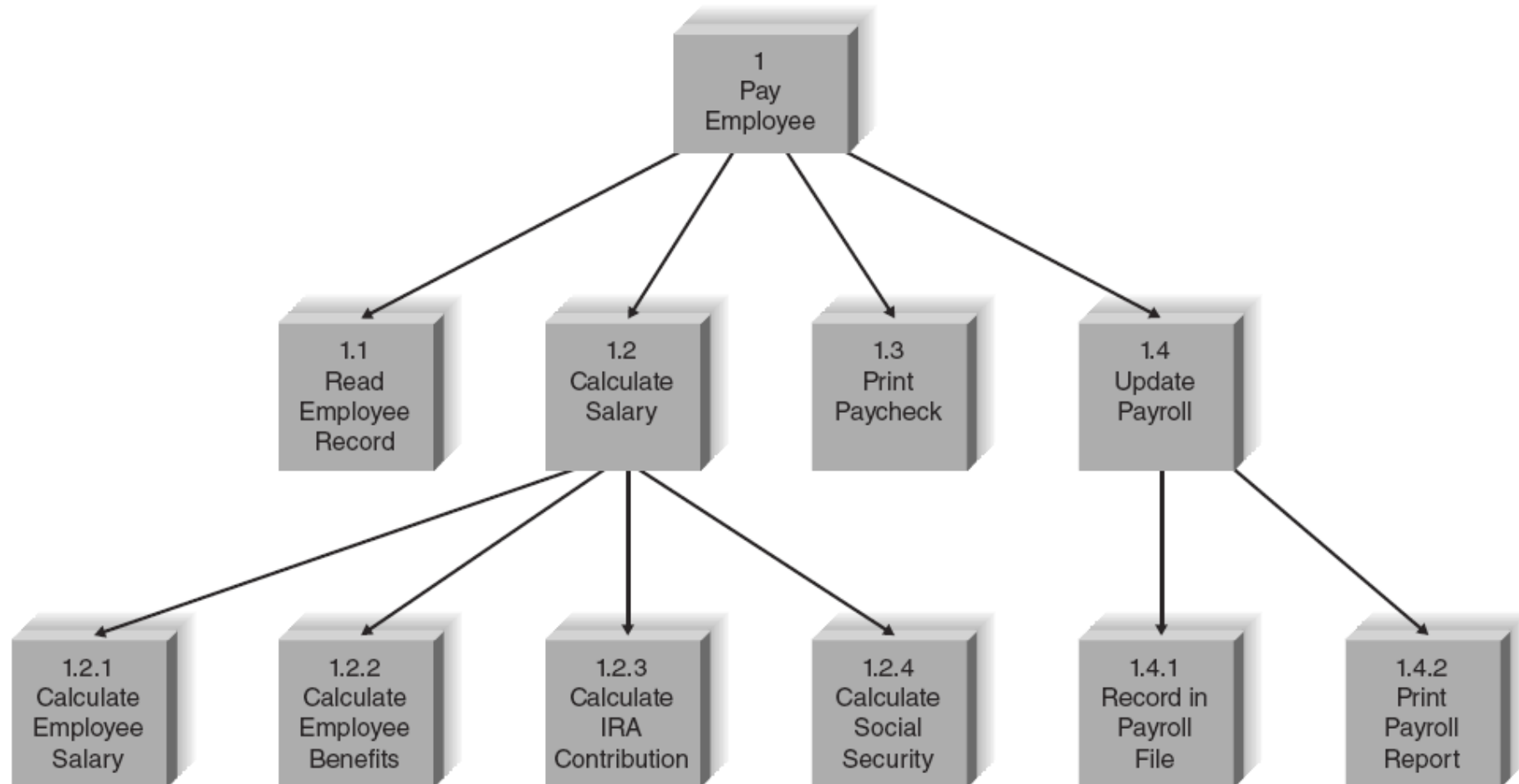
Checklist for Structure Chart Quality

- Library modules have been created whenever possible.
- The diagram has a high fan-in structure.
- Control modules have no more than seven subordinates.
- Each module performs only one function (high cohesion).
- Modules sparingly share information (loose coupling).
- Data couples that are passed are actually used by the accepting module.
- Control couples are passed from "low to high."
- Each module has a reasonable amount of code associated with it.

Program Specification

- No standard approach
- Include program information
- Note events that trigger actions
- List inputs and outputs
- Include pseudocode
- Provide additional notes and comments as needed

Sample Pseudocode

```
Calculate_discount_amount (total_price, discount_amount)
If total_price < 50 THEN
    discount_amount = 0
ELSE
If total_price < 500 THEN
    discount_amount = total_price *.10
ELSE
    discount_amount = total_price *.20
END IF
END
```

Sample Program Specification

Program Specification 2.1.2 for Assign Pilot to Flight Request

Module
Name: Locate Candidate Pilots
Purpose: Find the IDs of all pilots located in proximity to the flight location
Programmer: John Smith
Date due: April 26, 2020

☐ C   ■ Python   ☐ Visual Basic   ☐ Javascript

Events
Occurs upon submission of a new flight request by client

Input Name:	Type:	Provided by:	Notes:
Flight Location Latitude	Numeric	Program 2.1.1	

Output Name:	Type:	Used by	Notes:
Pilot ID	String (10)	Program 2.1.4	

Pseudocode
```
(Find_candidate_pilots module)
not_found = True
count = 0
Do Until not_found = False
  Define search region, radius from flight location lat/long = (radius * (count * expansion_factor))
  For all pilots in Pilot table
    If Pilot location lat/long is within search region,
      save Pilot ID
      not_found = False
    EndIf
  EndFor
  count += 1
EndDo
Return
```
Other
Business rule: Search for candidate pilots is performed until one candidate pilot ID is found.

Note: Management will need to determine if there is a maximum limit on the size of the search region. If there is, the module needs to return a control flag indicating that no pilots are capable of performing this flight request.

Chapter Review

- Identify and describe the five steps involved in converting the logical process models to physical process models.

- Describe the purpose and components of a structure chart.

- Explain the two primary structures shown in a structure chart and the purpose of each.

- Discuss the four steps involved in creating structure charts.

- Explain the structure chart design guideline of high cohesion. List and discuss the seven kinds of cohesion.

Chapter Review Continued

- Explain the structure chart design guideline of low coupling.

- List and discuss the five kinds of coupling.

- Explain how the concepts of fan-in and fan-out pertain to structure charts.

- Explain the recommended content of a program specification document.

- Describe pseudocode and explain how it differs from structured English.

Key Terms

- Afferent processes
- Central processes
- Cohesion
- Coincidental cohesion
- Conditional line
- Connector
- Content coupling
- Control couples

- Control module
- Couples
- Coupling
- Data couples
- Data coupling
- Efferent processes
- Event
- Event-driven
- Factoring

- Fan-in
- Fan-out
- Functional cohesion
- Human–machine boundary
- Iteration
- Library module
- Loop
- Module

Key Terms Continued

- Off-page connector
- On-page connector
- Physical data flow diagram (DFD)
- Physical process model
- Program design
- Program specification

- Pseudocode
- Selection
- Sequence
- Structure chart
- Subordinate module
- Temporal cohesion
- Top-down modular approach

- Transaction structure
- Transform structure