Systems Analysis and Design

5th Edition

Program Design

Alan Dennis, Barbara Haley Wixom, and Roberta Roth

Chapter 10 Outline

- Moving from logical to physical process models.
- Designing programs.
- Structure chart.
- Program specification.

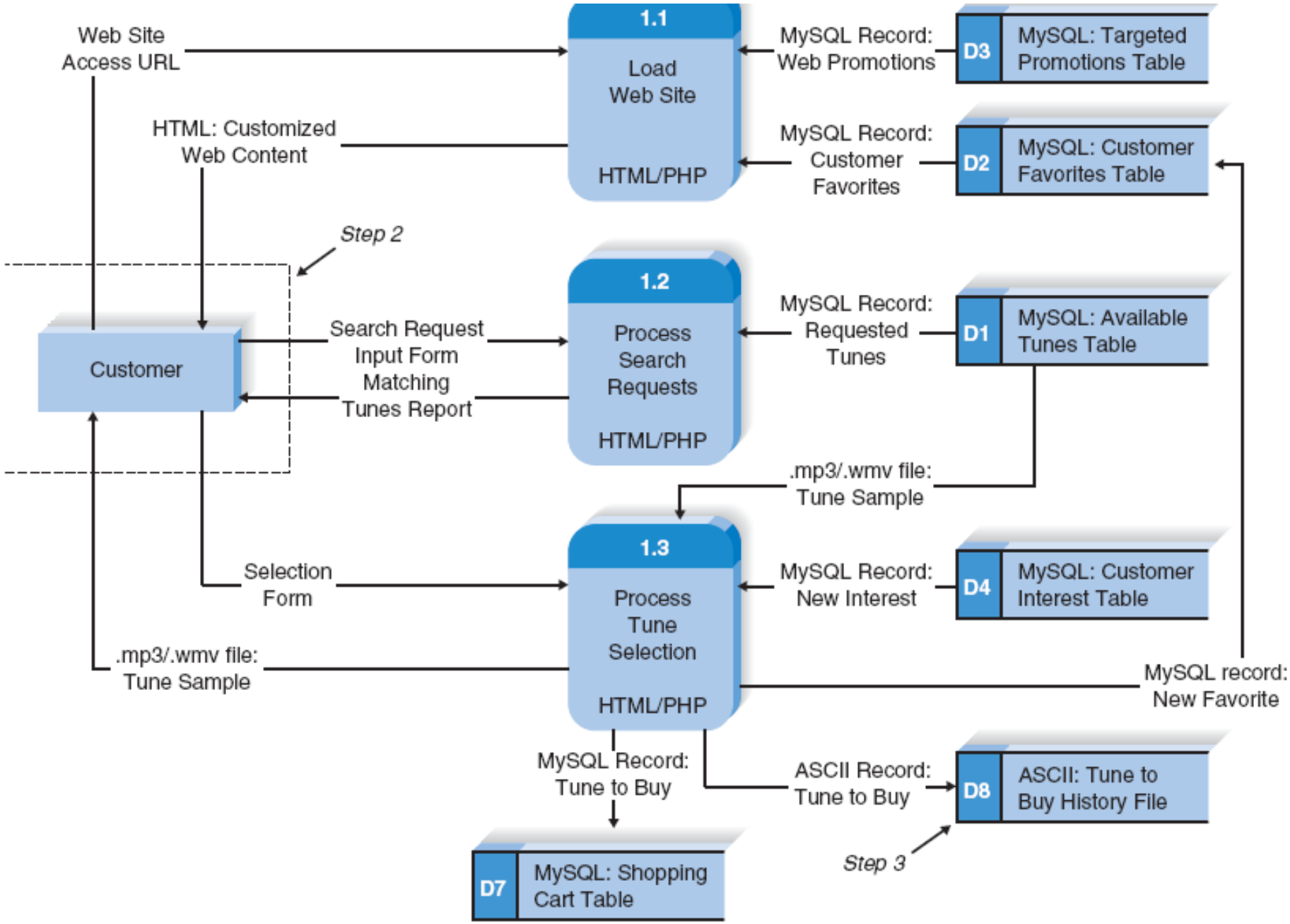
INTRODUCTION

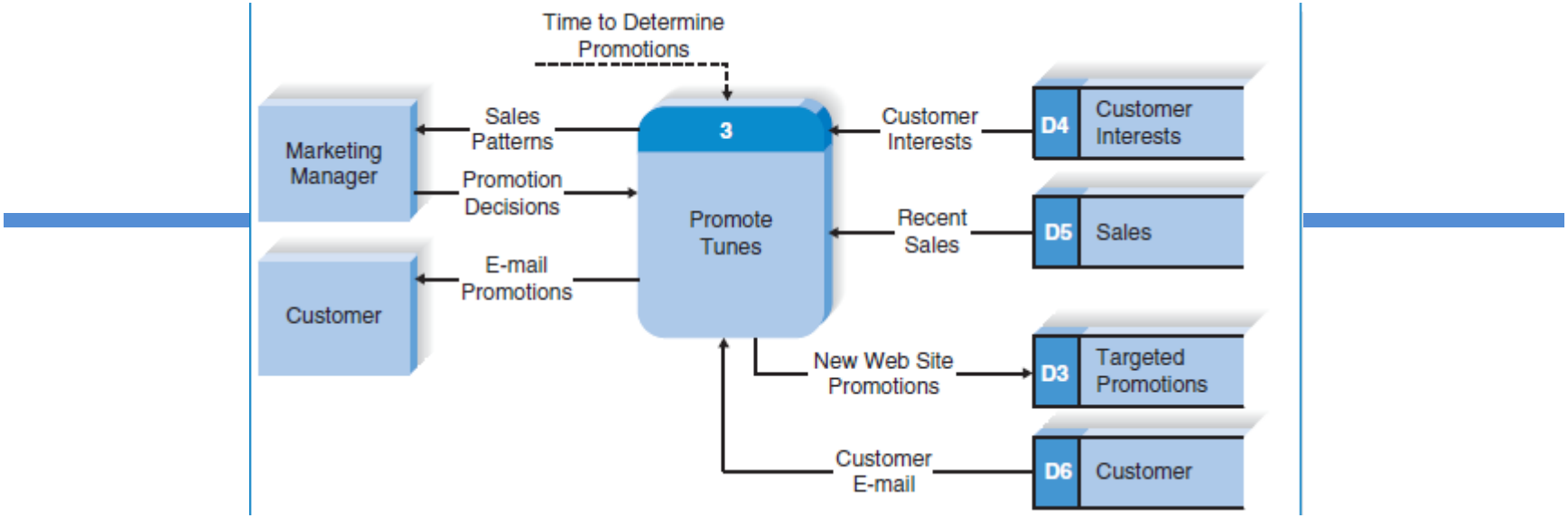
- **Program design** – Analysts determine what programs will be written and create instructions for the programmers.
- Various implementation decisions are made about the new system (e.g., what programming language(s) will be used.)
- The DFDs created during analysis are modified to show these implementation decisions, resulting in a set of ***physical data flow diagrams***.
- The analysts determine how the processes are organized, using a ***structure chart*** to depict the decisions.
- Detailed instructions called ***program specifications*** are developed.

Steps to Create the Physical Data Flow Diagram

Step	Explanation
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.
Draw a human–machine boundary.	Draw a line to separate the automated parts of the system from the manual parts.
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).
Update the data elements in the data flows.	Update the data flows to include system-related data elements.
Update the metadata in the CASE repository.	Update the metadata in the CASE repository to include physical characteristics.

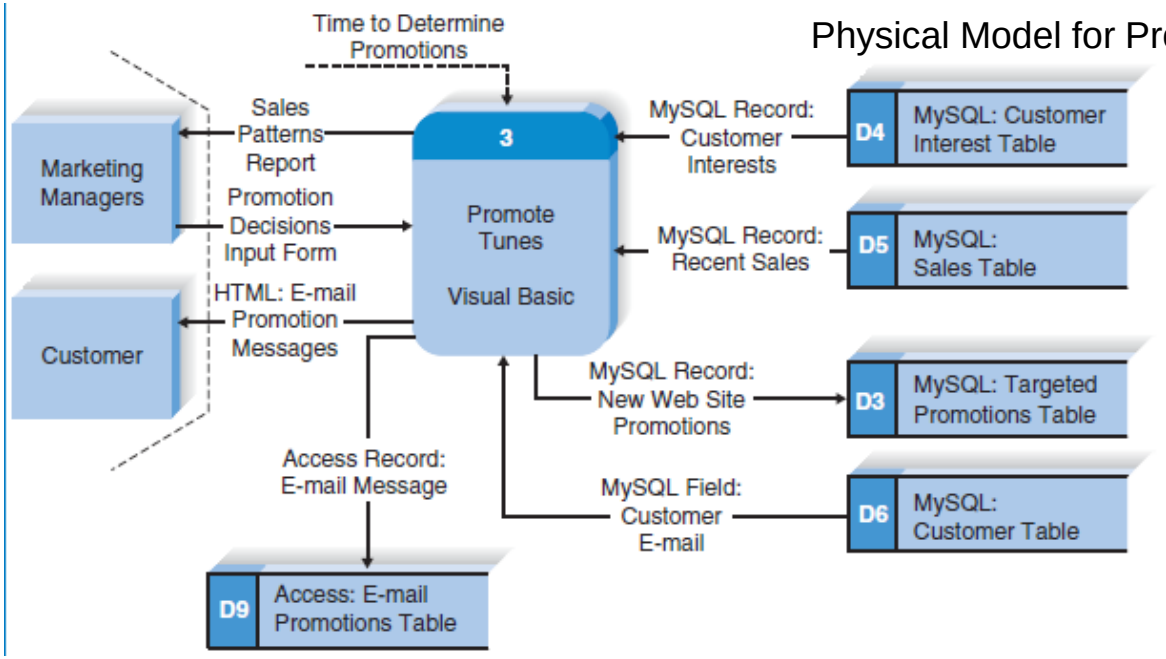
CASE = computer-aided software engineering; DFD = data flow diagram.


Web Site Access URL
1.1
Load Web Site
HTML/PHP
MySQL Record: Web Promotions
D3
MySQL: Targeted Promotions Table
HTML: Customized Web Content
MySQL Record: Customer Favorites
D2
MySQL: Customer Favorites Table
Step 2
Customer
Search Request Input Form
Matching Tunes Report
1.2
Process Search Requests
HTML/PHP
MySQL Record: Requested Tunes
D1
MySQL: Available Tunes Table
.mp3/.wmv file: Tune Sample
Selection Form
1.3
Process Tune Selection
HTML/PHP
MySQL Record: New Interest
D4
MySQL: Customer Interest Table
.mp3/.wmv file: Tune Sample
MySQL record: New Favorite
MySQL Record: Tune to Buy
ASCII Record: Tune to Buy
D8
ASCII: Tune to Buy History File
Step 3
D7
MySQL: Shopping Cart Table



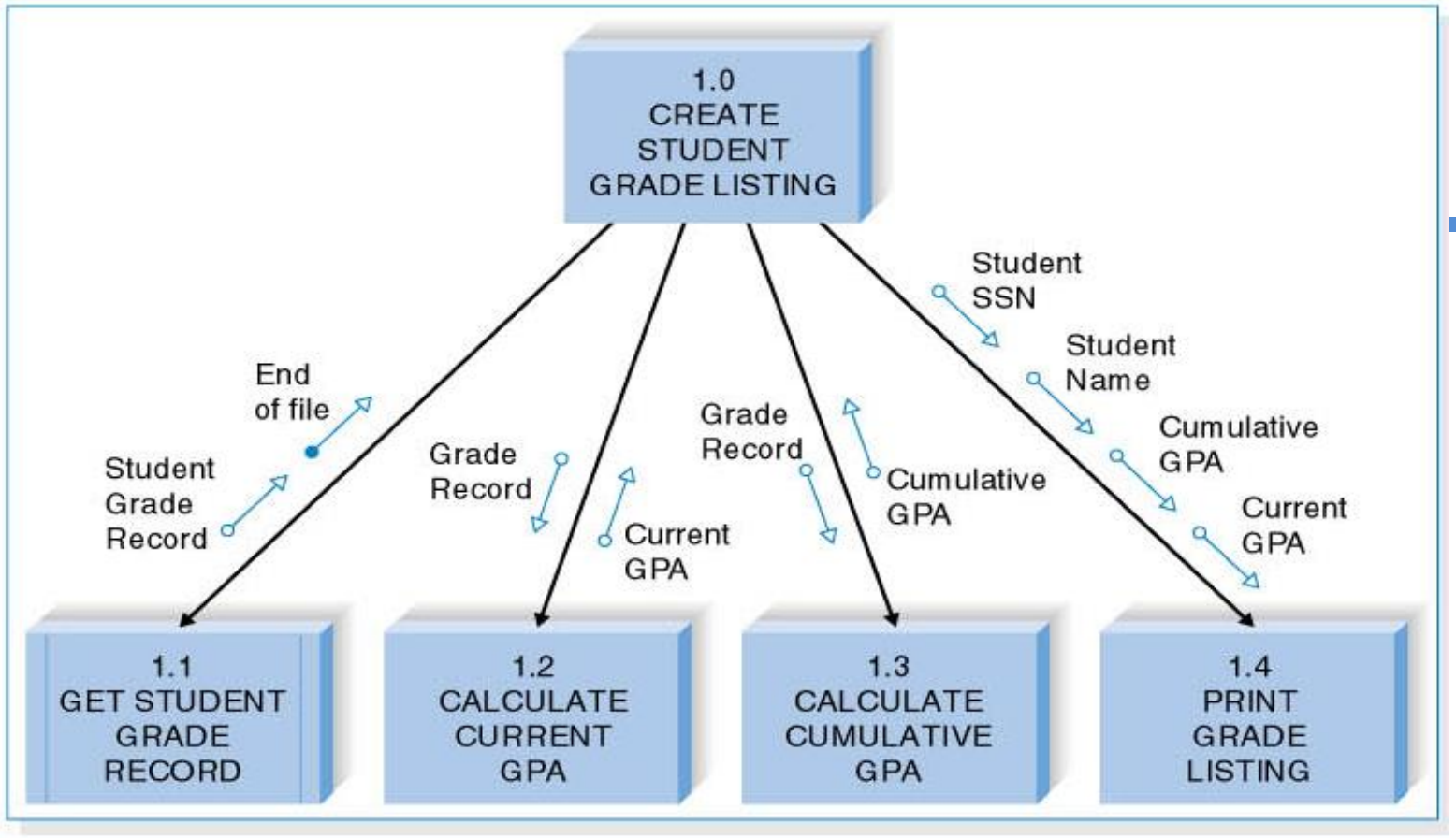
Logical Model of the Promote Tunes Process

Physical Model for Promote Tunes Process



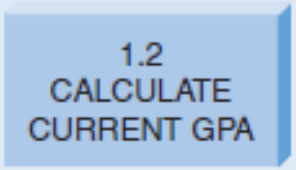
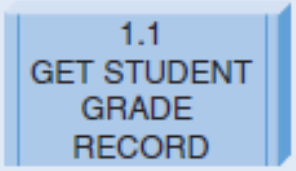
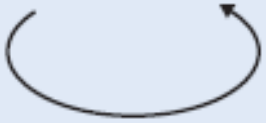
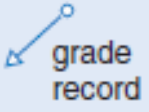
STRUCTURE CHART

- The ***structure chart*** is an Important program design technique that help the analyst design the program.
- It shows all components of code in a hierarchical format that implies
  - ***Sequence (in what order components are invoked)***,
  - ***Selection (under what condition a module is invoked)***,
  - ***Iteration (how often a component is repeated)***

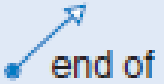


An academic system needs a program that will print a listing of students along with their grade point averages (GPAs), both for the current semester and overall. First, the program must retrieve the student grade records; then it must calculate the current and cumulative GPAs; finally, the grade list can be printed.

- a programmer can tell that there are four main code modules involved in creating a student grade listing: getting the student grade records, calculating current GPA, calculating cumulative GPA, and printing the listing. Also, there are various pieces of information that are either required by each module or created by it (e.g., the grade record, the cumulative GPA).

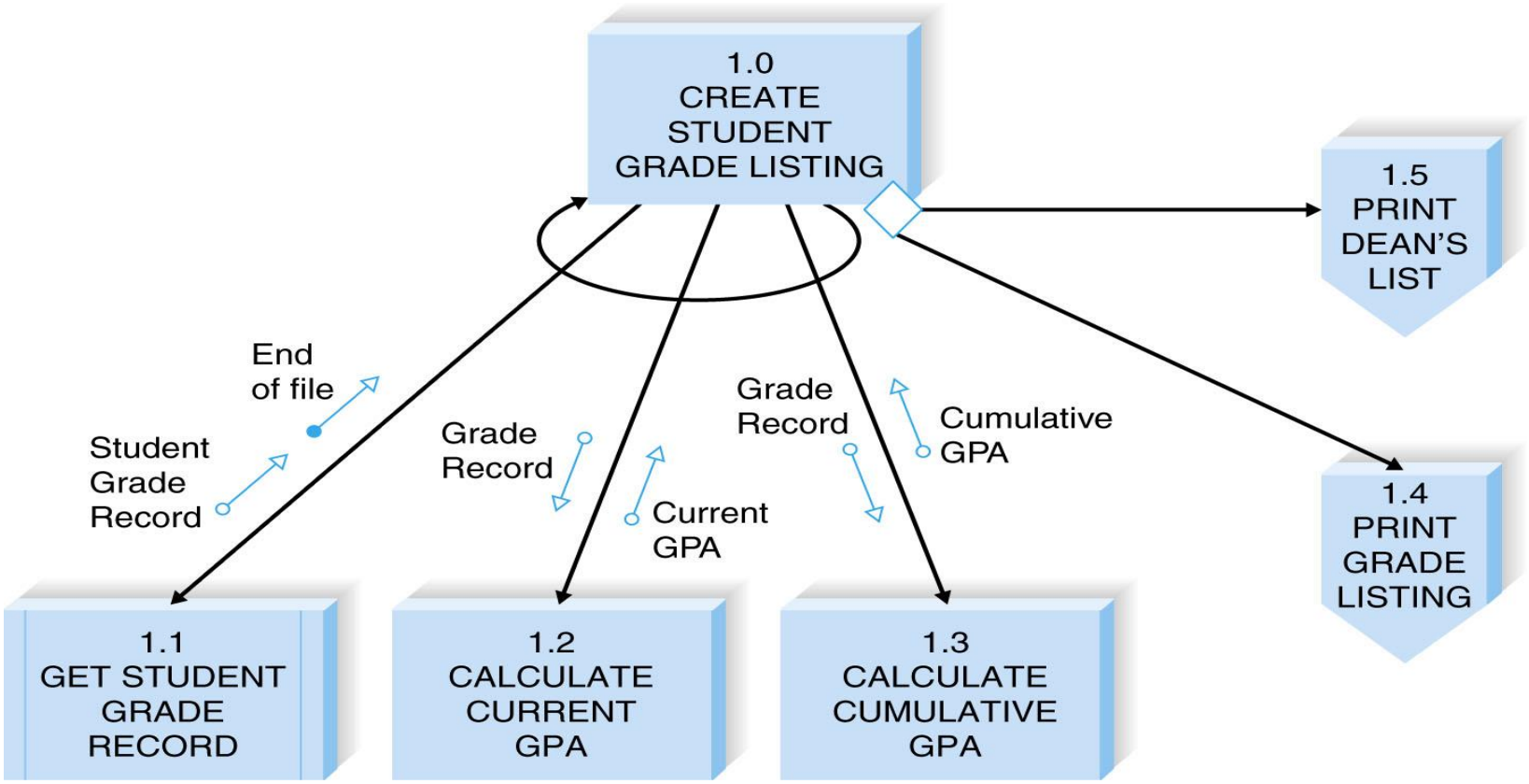
Structure Chart Element	Purpose	Symbol
Every *module*: • Has a number. • Has a namec. • Is a control module if it calls other modules below it. • Is a subordinate module if it is controlled by a module at a higher level.	Denotes a logical piece of the program	
Every *library module* has: • A number. • A name. • Multiple instances within a diagram.	Denotes a logical piece of the program that is repeated within the structure chart	
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	
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	

(cont'd)

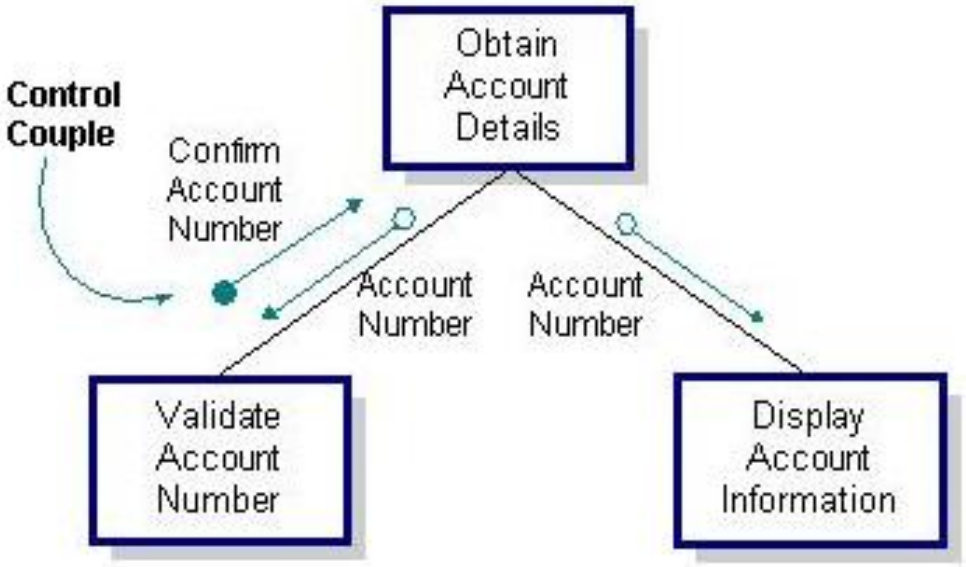
A *control couple*: • Contains an arrow. • Contains a filled-in circle. • Names the message or flag that is being passed. • Should be passed up, not down. • Has a direction that is denoted by the arrow.	Communicates that a message or a system flag is being passed from one module to another	
An *off-page 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	
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	

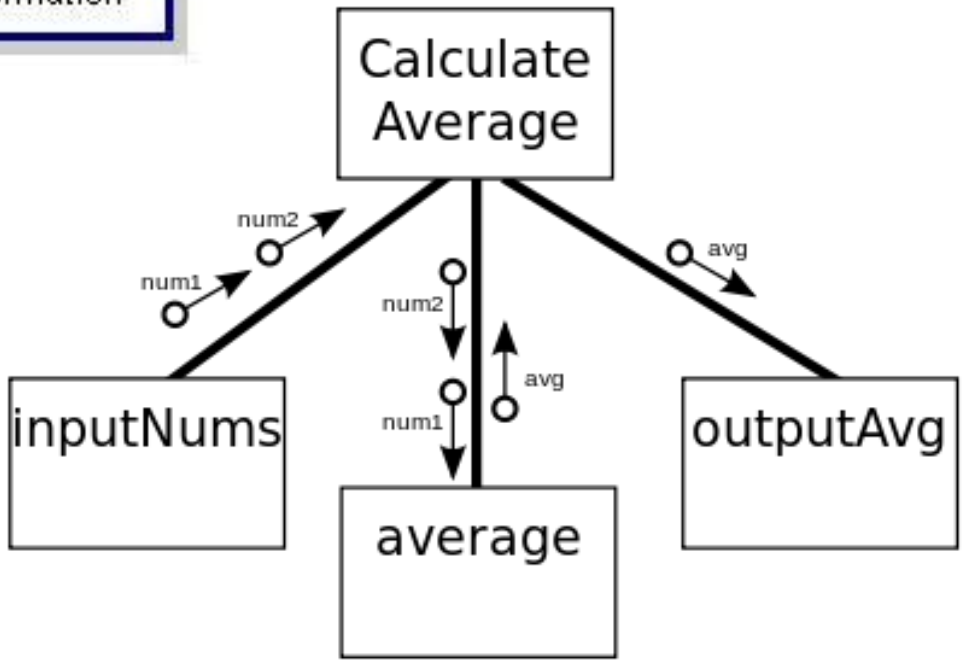
(cont'd)

Revised structure chart example



CONTROL COUPLE ON A STRUCTURE CHART





Design Guidelines

- High quality structure charts result in programs that are modular, reusable, and easy to implement.
- Measures of good design include
  - cohesion,
  - appropriate levels of fan-in and fan-out.

Build Modules with High Cohesion

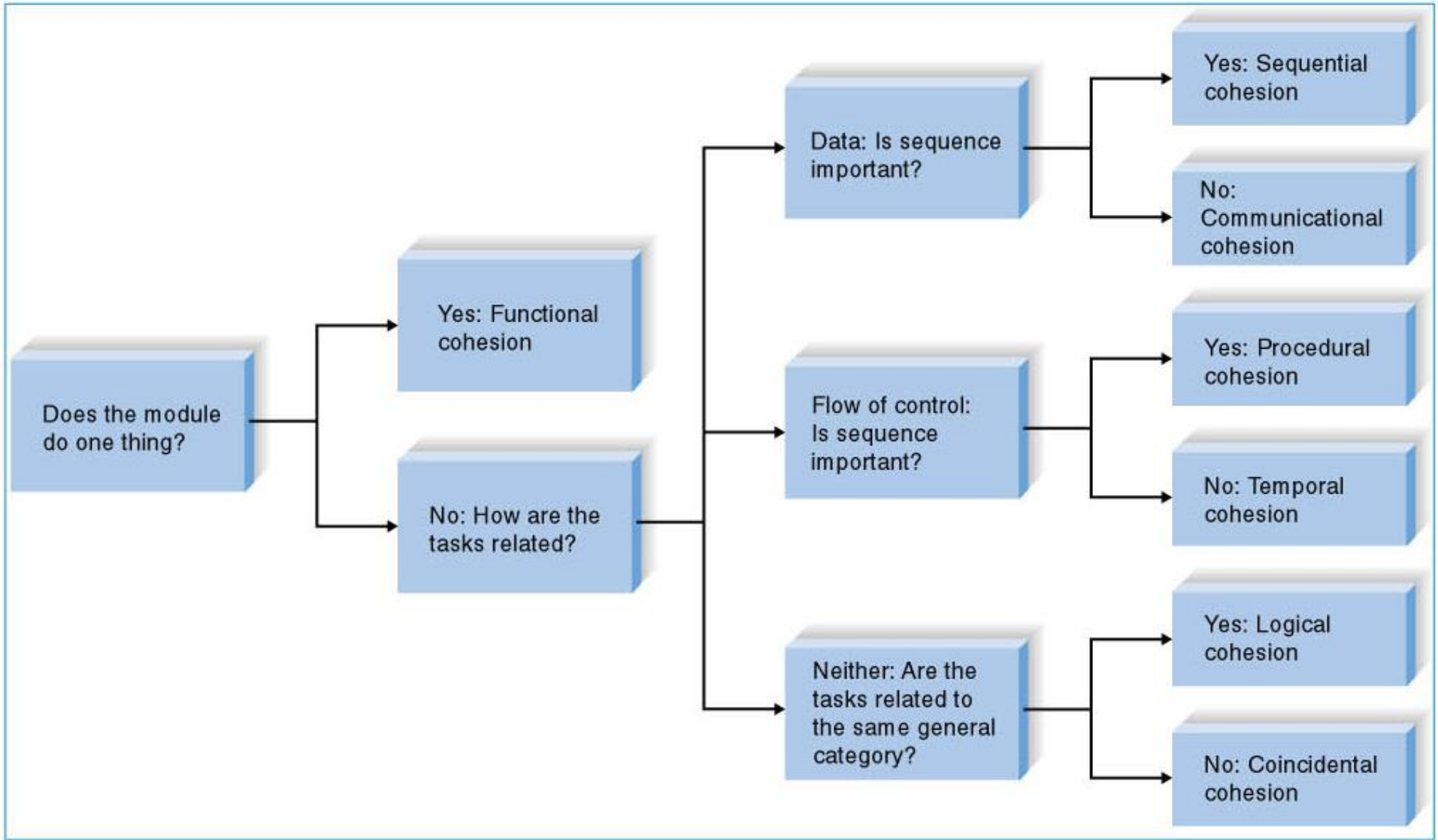
- ***Cohesion*** refers to how well the lines of code within each module relate to each other.
- Cohesive modules are easy to understand and build because their code performs one function effectively.

(cont'd)

- There are various types of cohesion.
- ***Functional cohesion*** – all elements of the modules contribute to performing a single task.
- ***Temporal cohesion*** – functions are invoked at the same time.
- ***Coincidental cohesion*** – there is no apparent relationship among a module's functions.

	Type	Definition	Example
Good	Functional	Module performs one problem-related task	Calculate Current GPA The module calculates current GPA only
	Sequential	Output from one task is used by the next	Format and Validate Current GPA Two tasks are performed, and the formatted GPA from the first task is the input for the second task
	Communicational	Elements contribute to activities that use the same inputs or outputs	Calculate Current and Cumulative GPA Two tasks are performed because they both use the student grade record as input
	Procedural	Elements are performed in sequence but do not share data	Print Grade Listing The module includes the following: housekeeping, produce report
	Temporal	Activities are related in time	Initialize Program Variables Although the tasks occur at the same time, each task is unrelated
	Logical	List of activities; which one to perform is chosen outside of module	Perform Customer Transaction This module will open a checking account, open a savings account, or calculate a loan, depending on the message that is sent by its control module
Bad	Coincidental	No apparent relationship	Perform Activities This module performs different functions that have nothing to do with each other: update customer record, calculate loan payment, print exception report, analyze competitor pricing structure

Cohesion Decision Tree



Fan-In

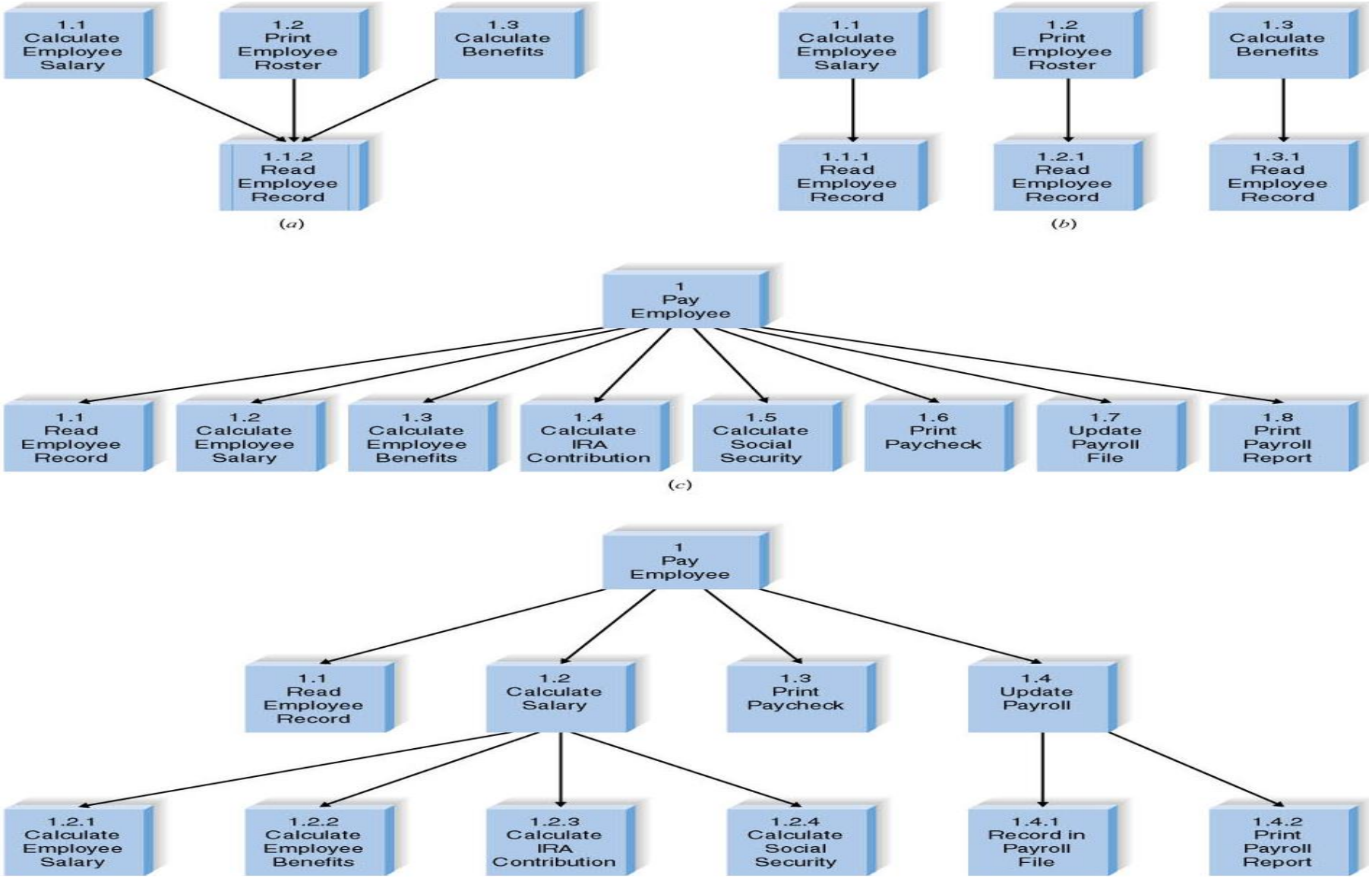
- ***Fan-in*** describes the number of control modules that communicate with a subordinate.
- A module with high fan-in has many different control modules that call it. This is a good situation because high fan-in indicates that a module is reused in many places on the structure chart.

Fan-Out

- A large number of subordinates associated with a single control should be avoided.
- The general rule of thumb is to limit a control module's subordinates to approximately seven for low ***fan-out***.

- ***fanin*** *is call by other modules.*
  - **Create High Fan-In**
- ***fanout*** *is call from that module.*
  - **Avoid High Fan-Out**

Examples of Fan-in and Fan-out



Assess the Chart for Quality

- Check list 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

- ***Program Specifications*** are documents that include explicit instructions on how to program pieces of code.
- There is no formal syntax for program specification.
- Four components are essential for program specification:
  - Program information;
  - Events;
  - Inputs and outputs; and
  - ***Pseudocode*** – a detailed outline of lines of code that need to be written.
- Additional notes and comments can also be included.

Program specification form

Program Specification 1.1 for ABC System

Module _____

Name:

Purpose:

Progammer:

Date due:

C PowerScript HTML/PHP Visual Basic

Events _____

Input Name	Type	Used by	Notes

Output Name	Type	Used by	Notes

Pseudocode _____

Other _____

Program Specification 1.1.2.2 for Digital Music Download System

Module

Name: Find_tune_by_Title
Purpose: Display basic tune information, using a title input by the user
Progammer: John Smith
Date due: April 26, 2009

☐ C ■ HTML/PHP ☐ Visual Basic ☐ Javascript

Events

search by title push-button is clicked
search by title hyperlink is selected

Input Name:	Type:	Provided by:	Notes:
Tune title	String (50)	Program 1.1.2	

Output Name:	Type:	Used by	Notes:
Tune ID	String (10)	Program 1.1.2	
Not_found	Logical	Program 1.1.2	Used to communicate when tune is not found

Pseudocode

```
(Find_tune module)
not_found =True
  For all tune titles in Available Tunes table
      If user title matches tune title, save tune ID
      not-found = False
      End If
End For
Return
```

Other

Business rule: If no matching tunes are found, the "Artist of the week" will appear to the user.

Note: A control couple containing a not _found flag should be included from 1.1.2.2 to 1.1.2 to instruct 1.1.2 to display a not-found message to the user and the Artist of the week.

(cont'd)

Pseudocode Example

```
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
```

SUMMARY

- **Moving from logical to physical process models**.
  - Physical DFDs show implementation details.
- **Structure chart**.
  - The structure chart shows all of the functional components needed in the program at a high level.
- **Building structure chart**.
  - Module, control connection, couples, review.
- **Structure chart design guidelines**.
  - Cohesion, coupling, and fan-in/fan-out.
- **Program specifications**.
  - Program specifications provide more detailed instructions to the programmers.