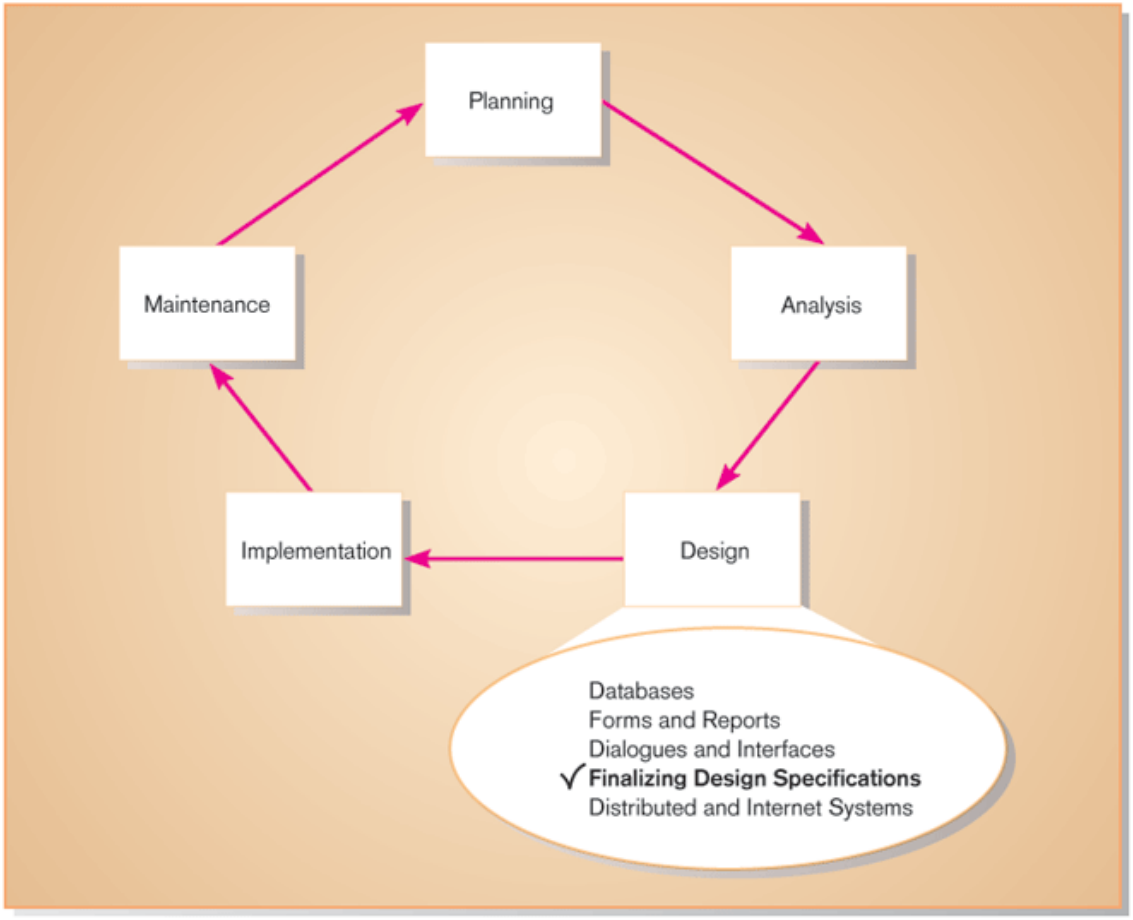
System Analysis and Design

Finalizing Design Specifications

Prof. Rania Elgohary
rania.elgohary@cis.asu.edu.eg
Dr. Yasmine Afify
yasmine.afify@cis.asu.edu.eg

Finalizing Design Specifications

Figure 13-1 The systems development life cycle with design phase highlighted



Finalizing Design Specifications

Good specifications should be stated:

- simply,
- completely,
- unambiguous,
- and have attributes that make requirements more understandable.

Deliverables and Outcomes

- A set of physical **design specifications** for the entire system, with **detailed specifications** for each separate part of the system.
  - Include functional descriptions for each part of the system.
  - Include input received and output generated for each program and its component parts.

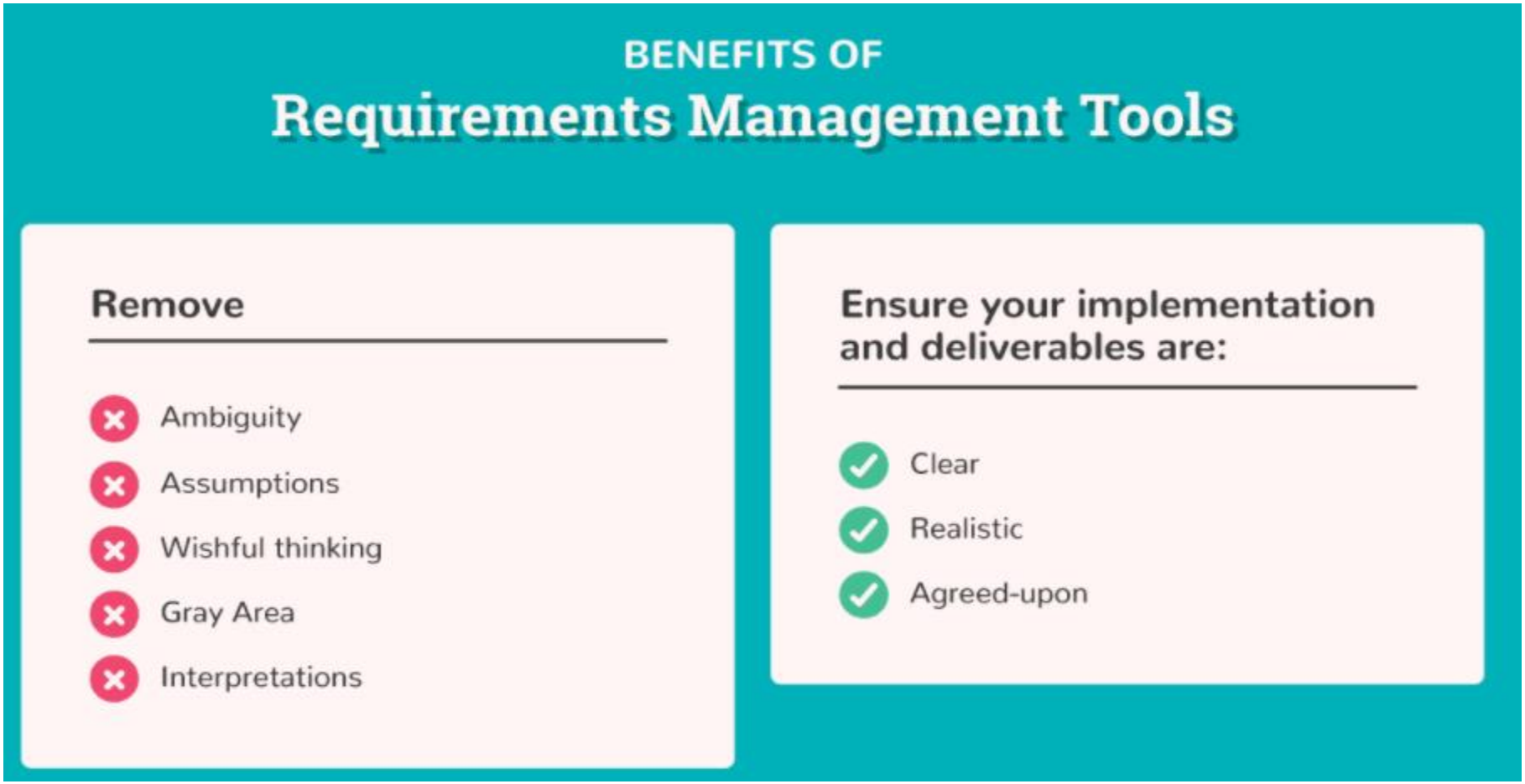
Specification Documents

- Contains:
  - Overall system description.
  - Interface requirements.
  - System features.
  - Nonfunctional requirements.
  - Supporting diagrams and models.

Requirements Management (RM) Tools

- Requirements management tools make it easier to keep documents up to date, add additional requirements and link related requirements.

Requirements Management (RM) Tools



Designing Programs

- Analysts should create a design of a maintainable system that is modular and flexible.
- Analysts can design programs in a ***top-down modular approach***.

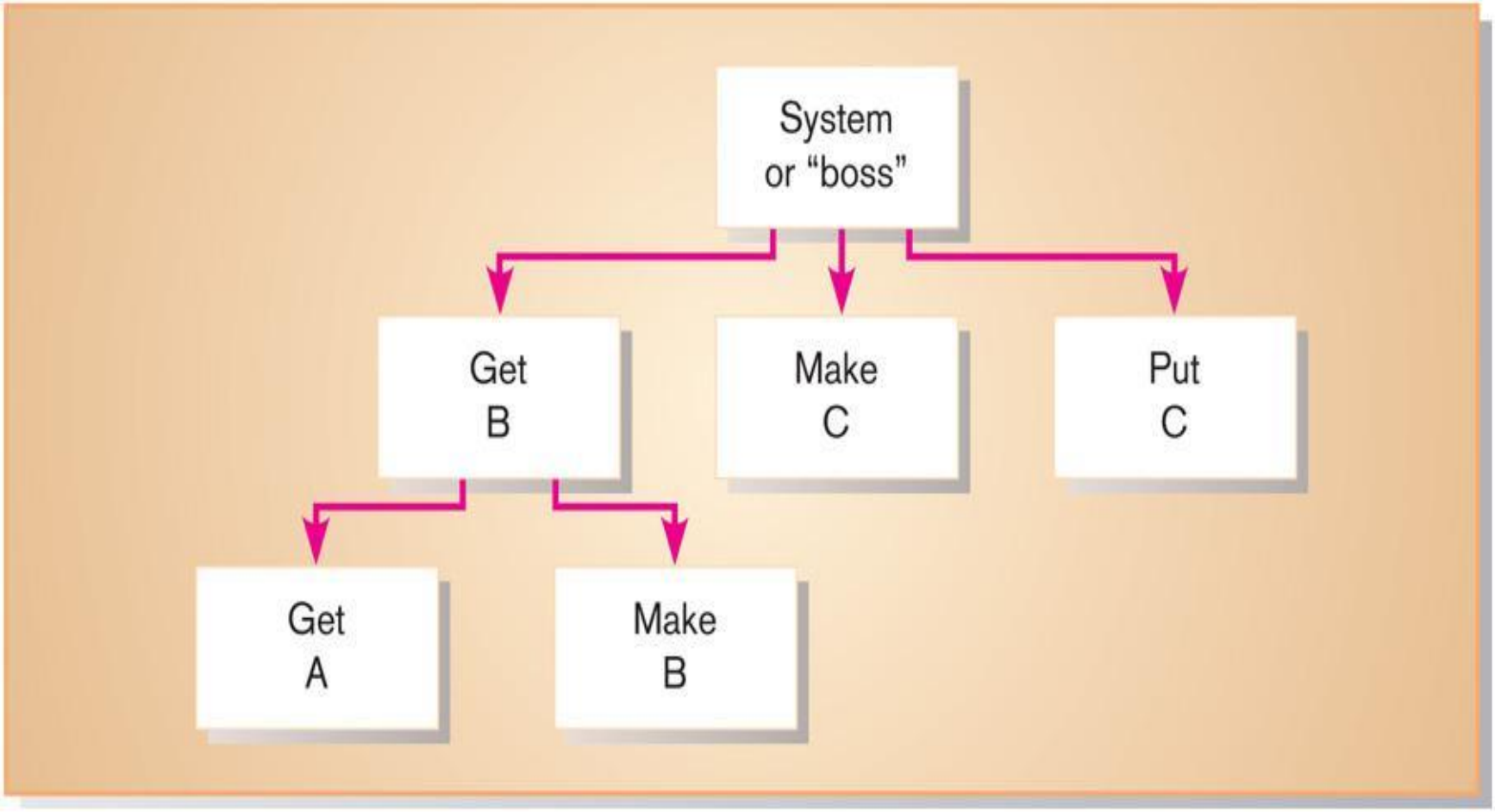
Designing Programs

- A high-level diagram, called ***structure chart***, is created to illustrate the organization and interaction of the different pieces of code within the program.
- ***Program specifications*** are written to describe what needs to be included in each program module.
- At the end of program design, the project team compiles the ***program design*** document.

Structure Chart

- A hierarchical diagram that shows how an information system is organized.
- Important program design technique that helps the analyst design the program.
- It shows all components of code in a hierarchical format that implies:
  - ***Sequence*** (order of invoking components)
  - ***Selection*** (under what condition module is invoked)
  - ***Iteration*** (how often component is repeated)

Structure Chart (Cont.)



Structure Chart (Cont.)

- Structure chart is composed of ***modules.***
- **Modules:** a self-contained component of a system that is defined by its function.
- A **control** module is a higher-level component that contains the logic for performing other modules, and the components that it calls and controls are considered **subordinate** modules.
- In previous figure, module “Get B” is the control module that directs modules “Get A” and “Make B” as its subordinates.

Linking DFDs to Structure Charts

- Each process of a DFD tends to represent one module on the structure chart.
- If leveled DFDs are used, then each DFD level tends to correspond to a different level of the structure chart hierarchy.
- The process on the context-level DFD would correspond to the top module on the structure chart.

Structure Chart Symbols

- **Data couple**:

Diagrammatic representation of the **data exchanges** between two modules.

- **Control couple (Flag)**:

Diagrammatic representation of a **message** passed between two modules.

Structure Chart (Cont.)

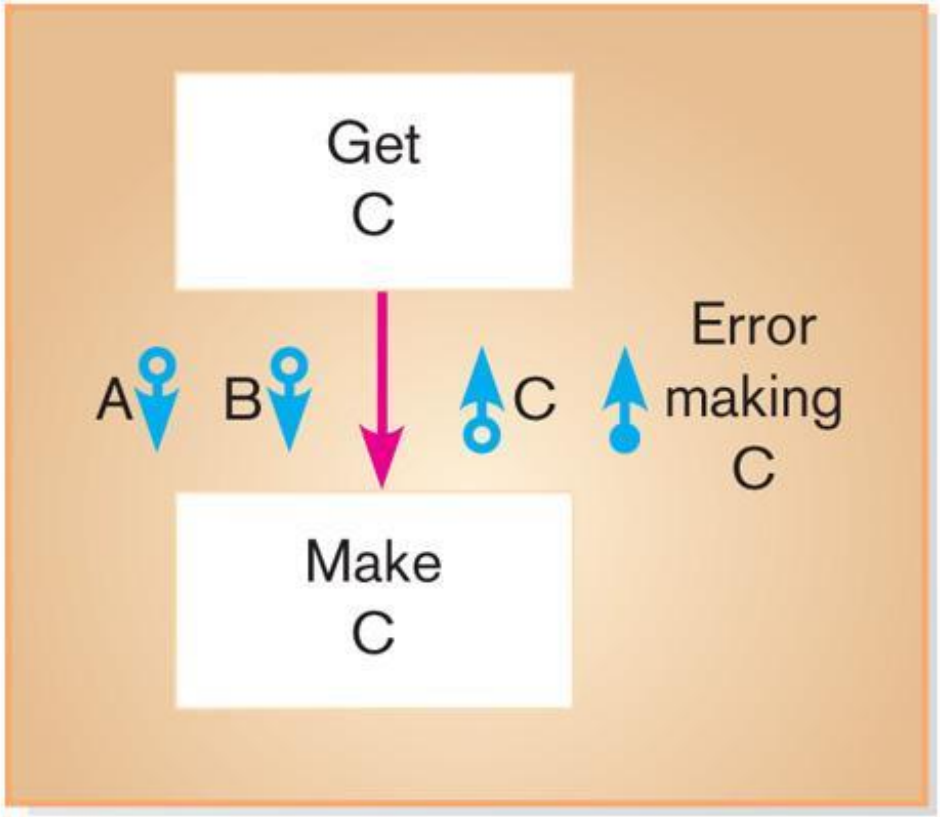


Figure 13-6 Special symbols used in structure charts – Data couples and control flag

Structure Chart (Cont.)

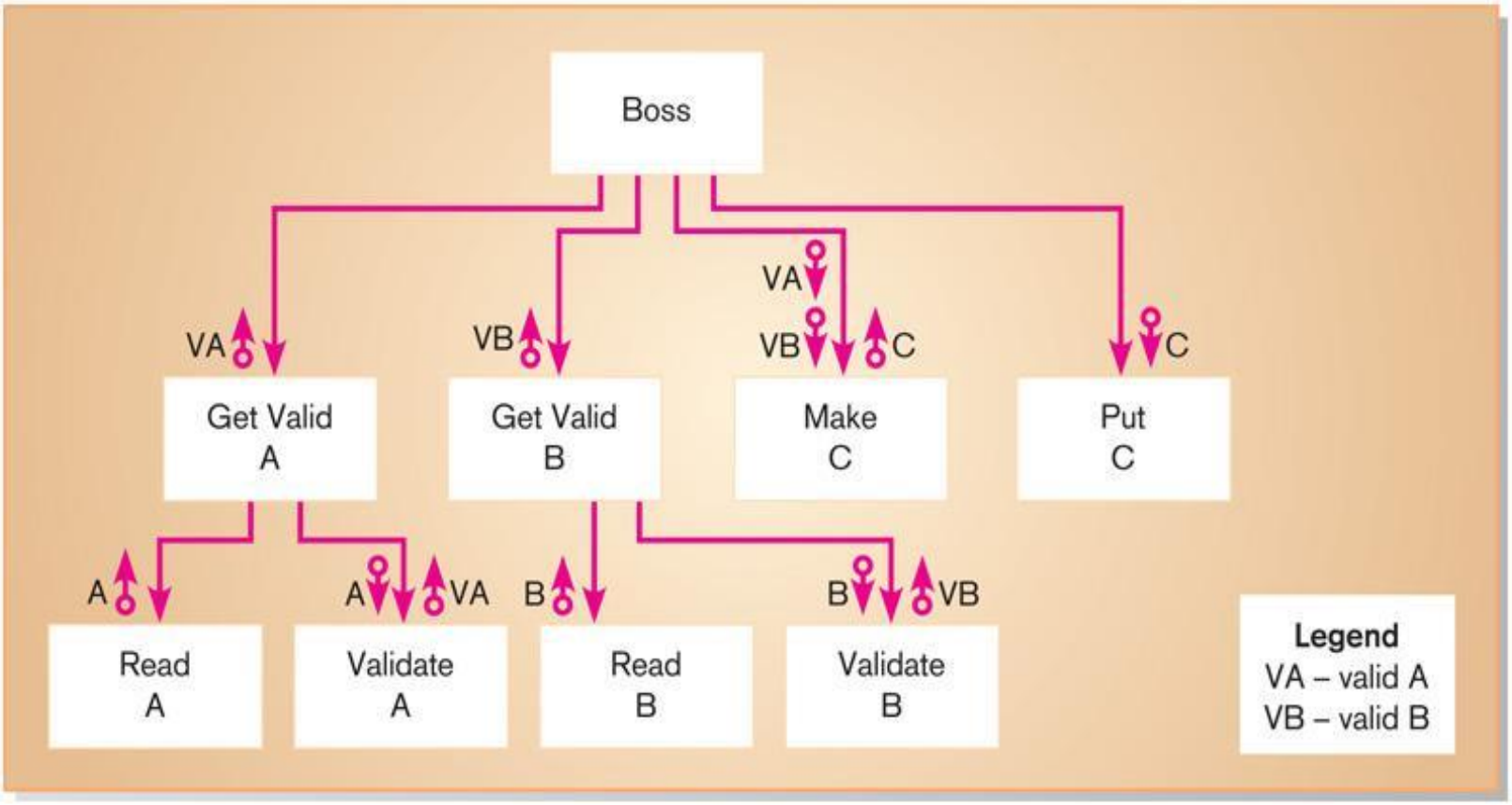


Figure 13-7 How to read a structure chart – (a) Nonoverlapping arrows

Structure Chart (Cont.)

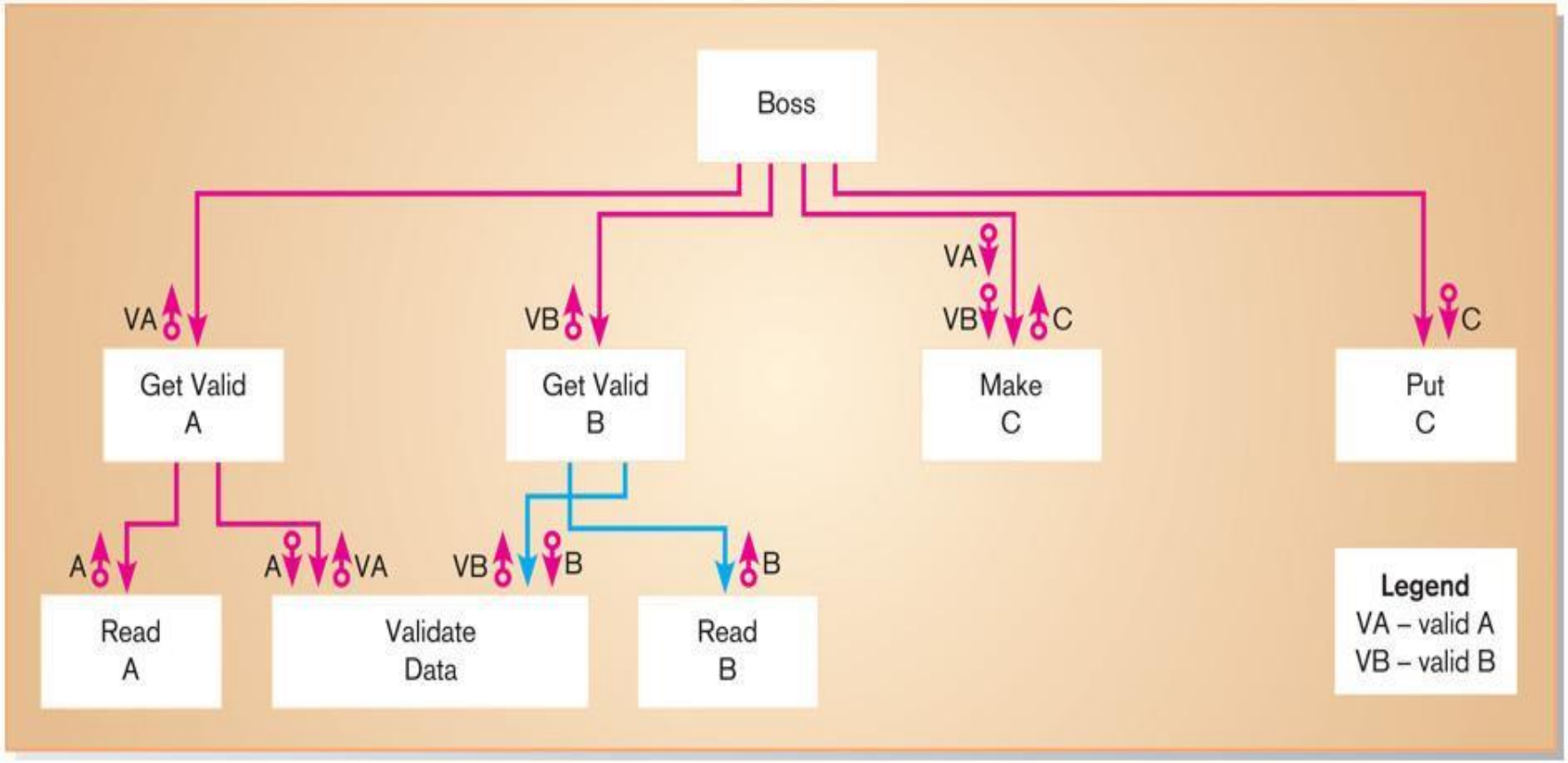


Figure 13-7 How to read a structure chart – (b) Overlapping arrows

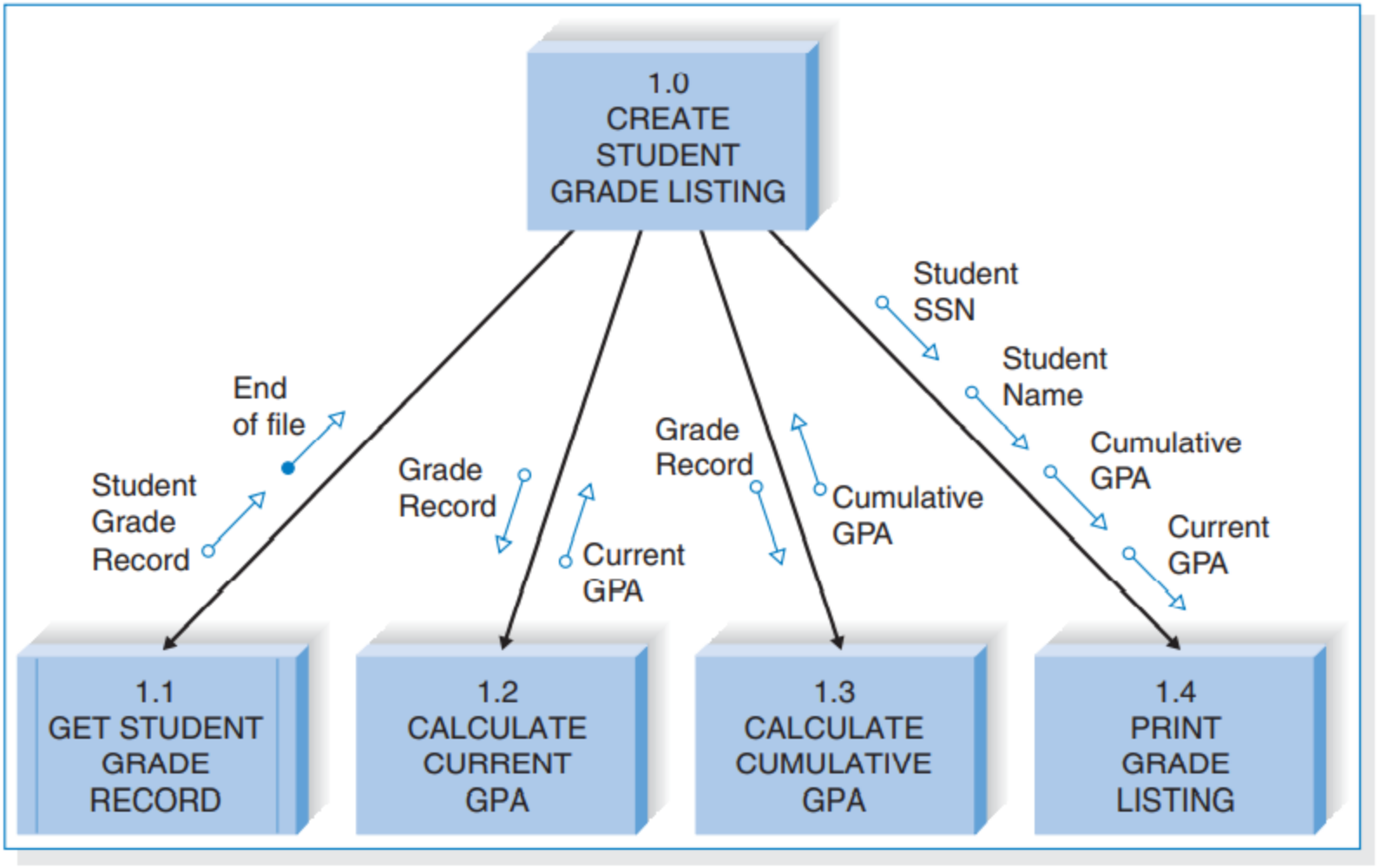
Steps in Building the Structure Chart

1. Identify top level modules and decompose them into lower levels
2. Add control connections (loops, conditional lines, etc.)
3. Add couples (identify the information that has to pass among the modules i.e. data couple and control couple)
4. Review and revise again and again until complete

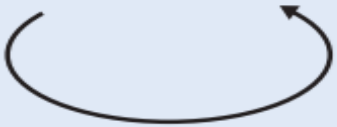
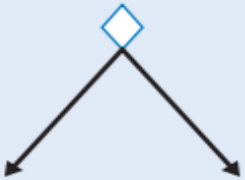
Example

- 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.
- By looking at this structure chart, 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).

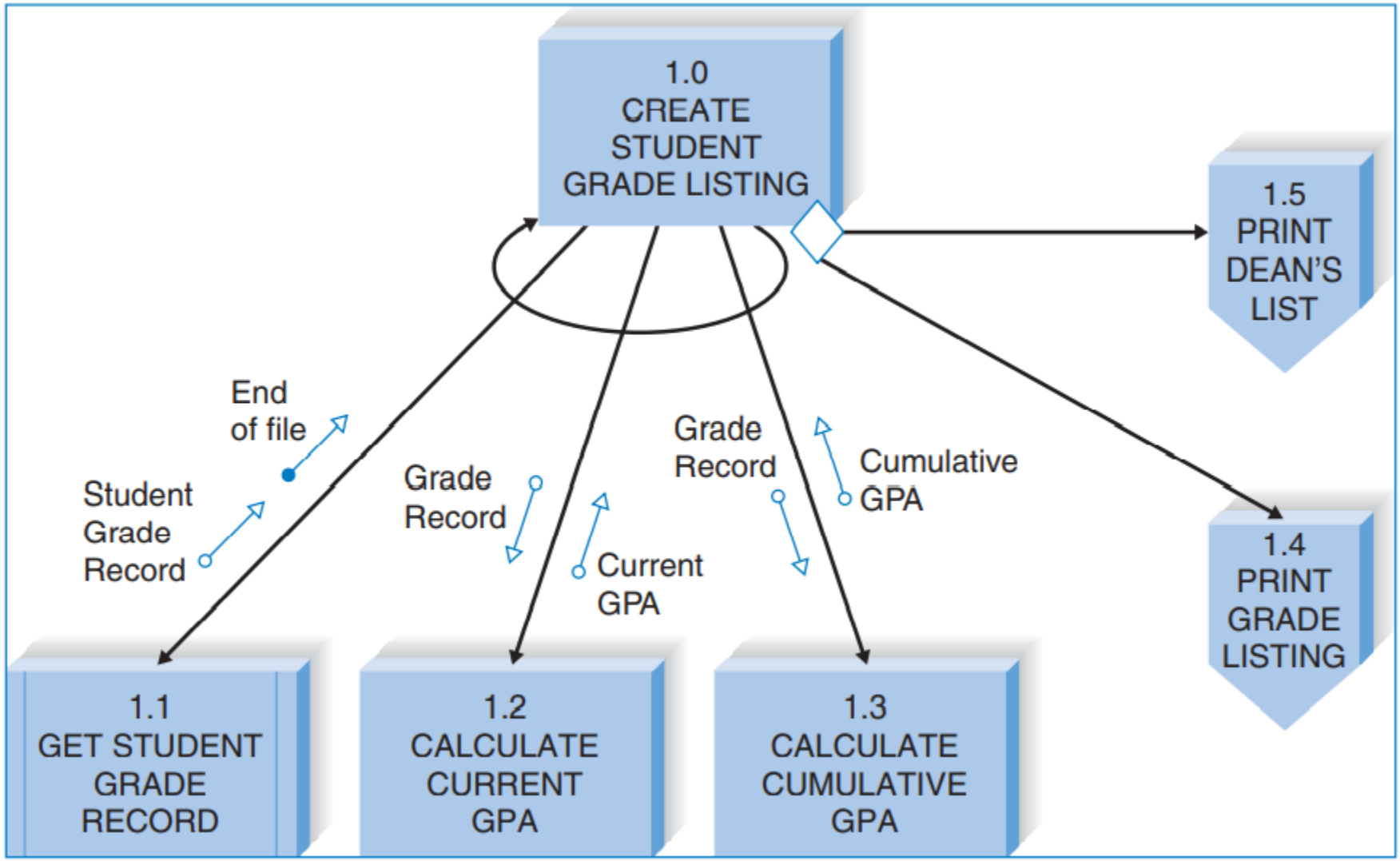
Structure Chart Example



Symbols

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	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	

Revised Structure Chart Example



Design Guidelines

- High quality structure charts result in programs that are modular, reusable, and easy to implement.
- Measures of good design 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 module relate to each other.
- Cohesive modules are easy to understand and build because their code performs one function effectively.
- Build Modules with ***High Cohesion***.

Cohesion Types

- ***Functional cohesion*** – all elements of the modules contribute to performing a single task.

- ***Coincidental cohesion*** – there is no apparent relationship among a module's functions.

Cohesion Types

	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.

Advantages of high cohesion

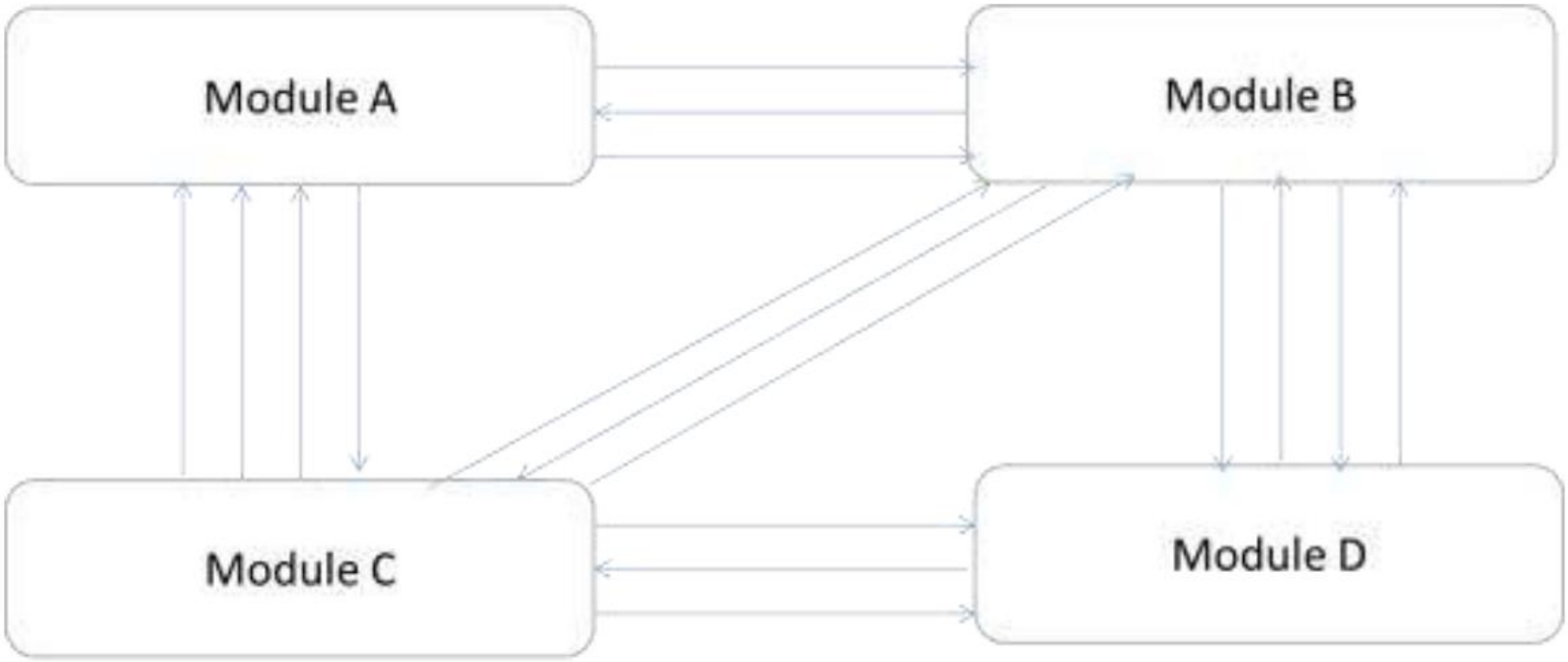
- Improved readability and understandability: High cohesion results in clear, focused modules with a single, well-defined purpose, making it easier for developers to understand the code and make changes.
- Better error isolation: High cohesion reduces the likelihood that a change in one part of a module will affect other parts, making it easier to
- Improved reliability: High cohesion leads to modules that are less prone to errors and that function more consistently, leading to an overall improvement in the reliability of the system.

Good Practice: Factoring

- ***Factoring*** is the process of separating out a function from one module into a module of its own.
- Factoring can make modules cohesive and create a better structure.

Coupling

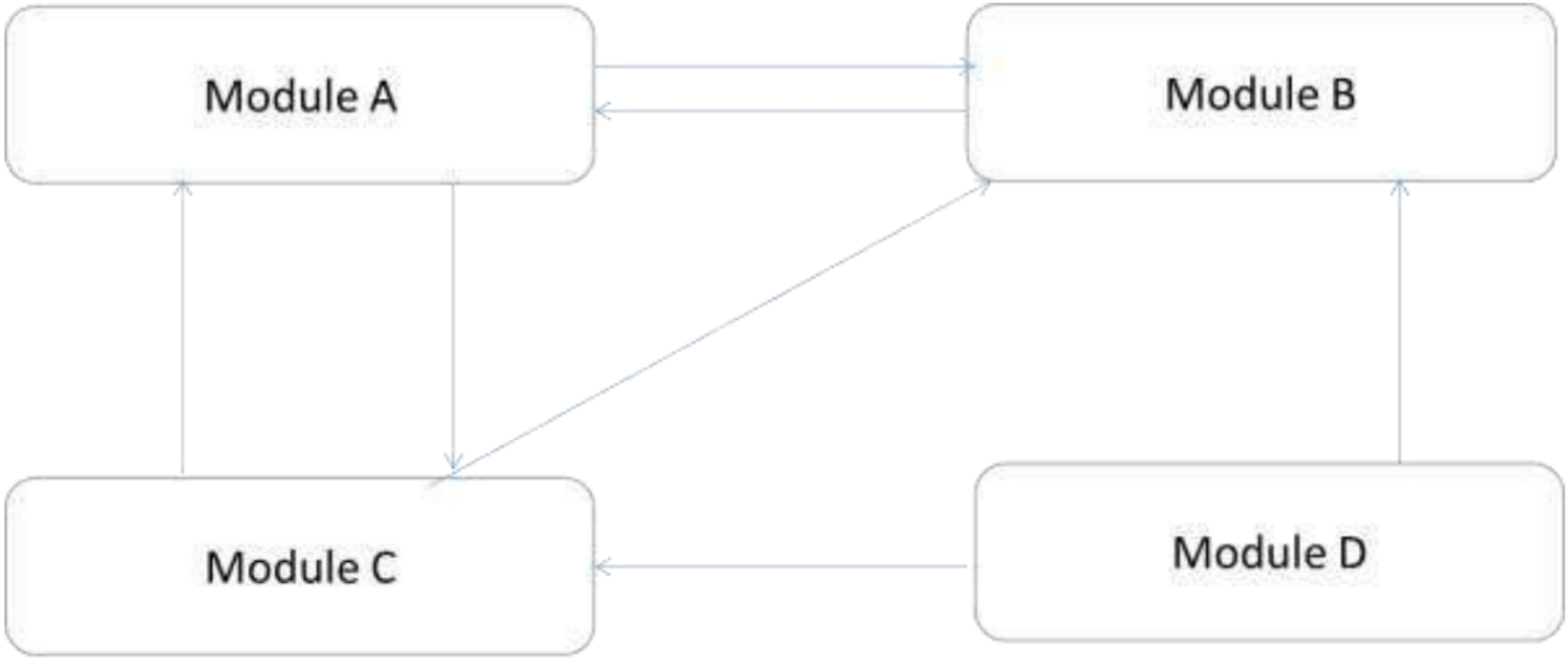
- Coupling is the measure of the independence of components. It defines the degree of dependency of each module of system development on the other.
- In practice, this means the stronger the coupling between the modules in a system, the more difficult it is to implement and maintain the system.
- Each module should have simple, clean interface with other modules, and the minimum number of data elements should be shared between modules. Modules should be ***loosely coupled***.



Highly Coupled System

Low Coupling

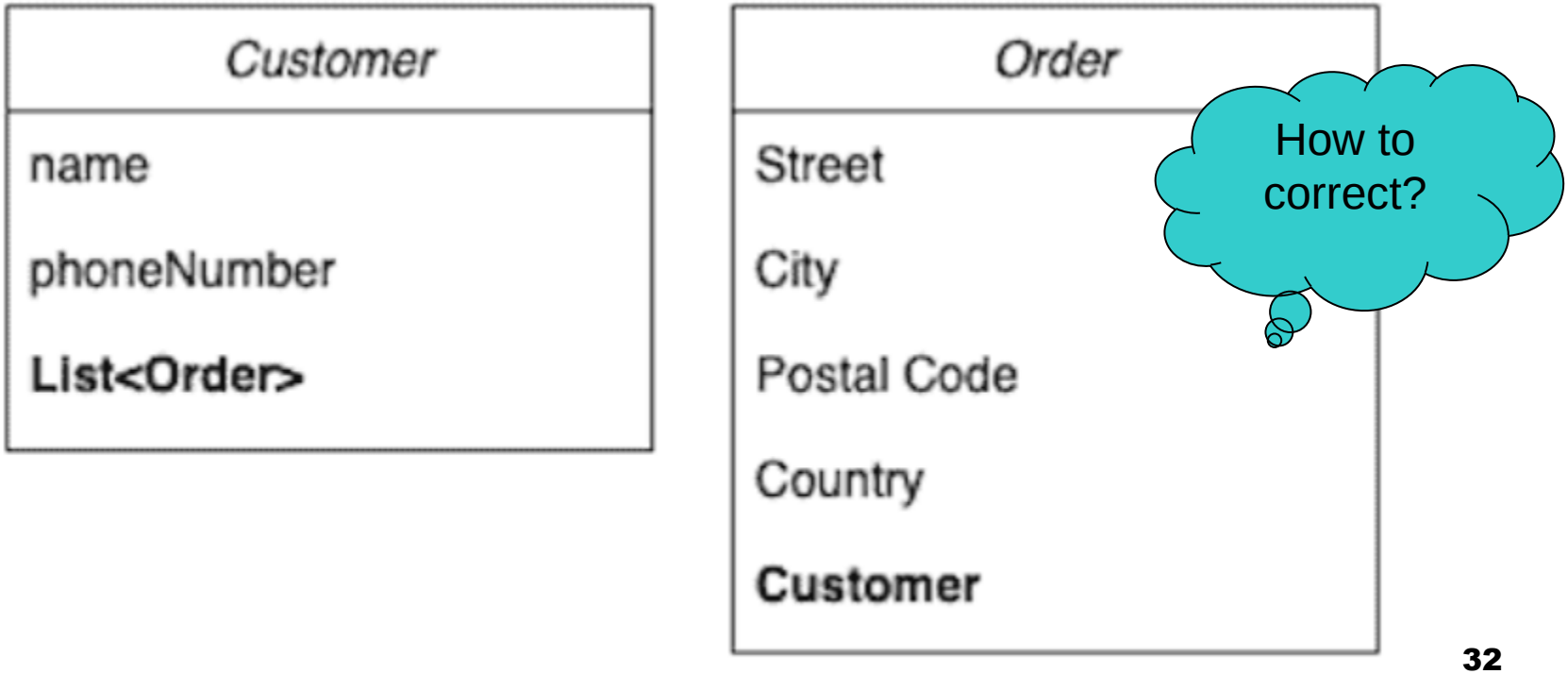
These type of systems are made up of components which are independent or almost independent. A change in one subsystem does not affect any other subsystem.



Loosely coupled system

Example on High Coupling

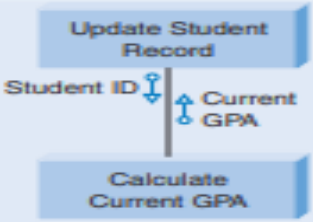
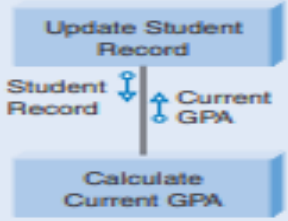
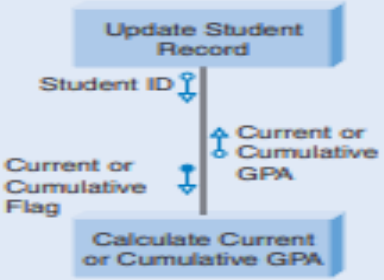
- *Customer* and *Order* are tightly coupled to each other. The *Customer* is storing the list of all the orders placed by a customer, whereas the *Order* is storing the reference to the *Customer* object.



Coupling Types

- Content Coupling: When one component actually modifies another, then the modified component is completely dependent on modifying one. It is the worst coupling type.
- Common Coupling: When amount of coupling is reduced somewhat by organizing system design so that data are accessible from a common data store.
- Control Coupling: When one component passes parameters to control the activity of another component.
- Stamp Coupling: When data structures is used to pass information from one component to another.
- Data Coupling: When only data is passed then components are connected by this coupling.

Coupling Types

	Type	Definition	Example
Good	Data	Modules pass fields of data or messages.	 All couples that are passed are used by the receiving module.
	Stamp	Modules pass record structures.	 Not all of the student record is used by the receiving module; only the *student ID* field is.
	Control	Module passes a piece of information that intends to control logic.	 The receiving module has to determine which GPA to calculate.
	Common	Modules refer to the same global data area.	Typically, common coupling cannot be shown on the structure chart; it occurs when modules access the same data areas, and errors made in those areas can ripple through all the modules that use the data.
Bad	Content	Module refers to the inside of another module.	Module A: Update Student If student = new Then go to Module B Module B: Create Student At all costs, avoid modules referring to each other in this way.

Advantages of low coupling

- Improved maintainability:

Low coupling reduces the impact of changes in one module on other modules, making it easier to modify or replace individual components without affecting the entire system.

- Enhanced modularity:

Low coupling allows modules to be developed and tested in isolation, improving the modularity and reusability of code.

- Better scalability:

Low coupling facilitates the addition of new modules and the removal of existing ones, making it easier to scale the system as needed.

Create High Fan-In

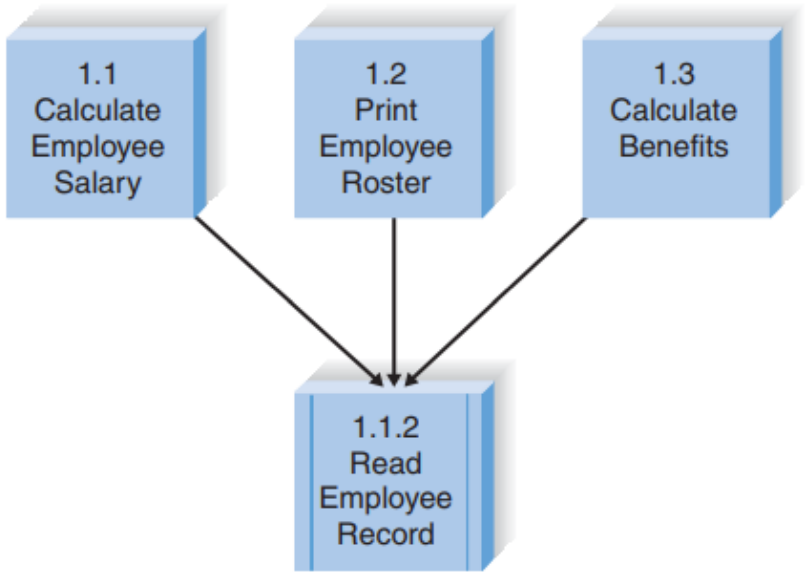
- ***Fan-in*** describes the number of control modules that communicate with a subordinate.
- A module with high fan-in has many different control modules that call it.
- This is a ***good*** situation because ***high fan-in*** indicates that a module is ***reused*** in many places on the structure chart.

Avoid High Fan-Out

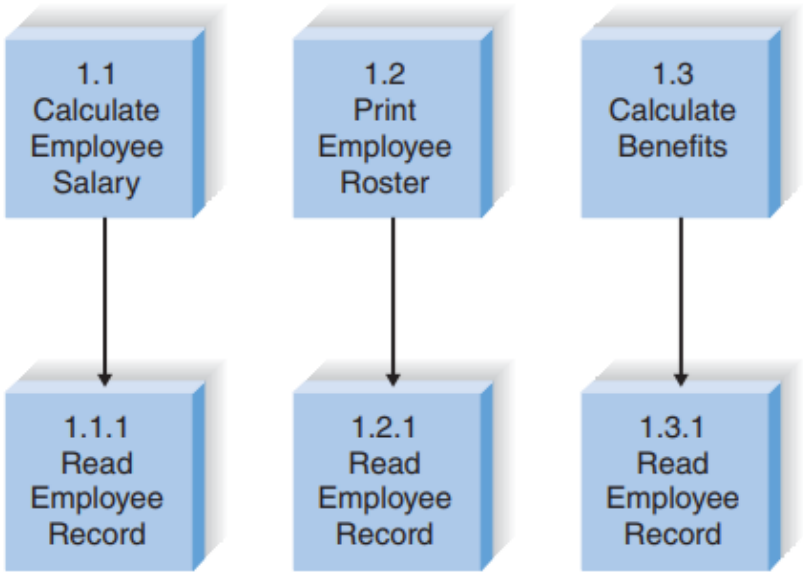
- ***Fan-out*** is the number of subordinates associated with a single control.
- A large number of subordinates associated with a single control should be ***avoided***.
- The general rule of thumb is to limit a control module's subordinates to approximately **seven**.

Fan-in / Fan-out

Two different approaches for representing the functionality of reading an employee record. Example *a* is better.



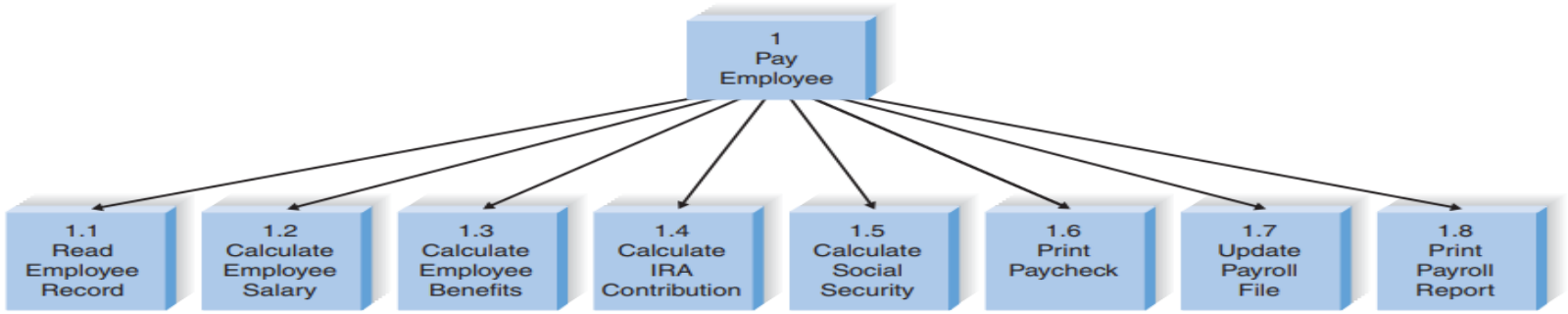
(*a*)



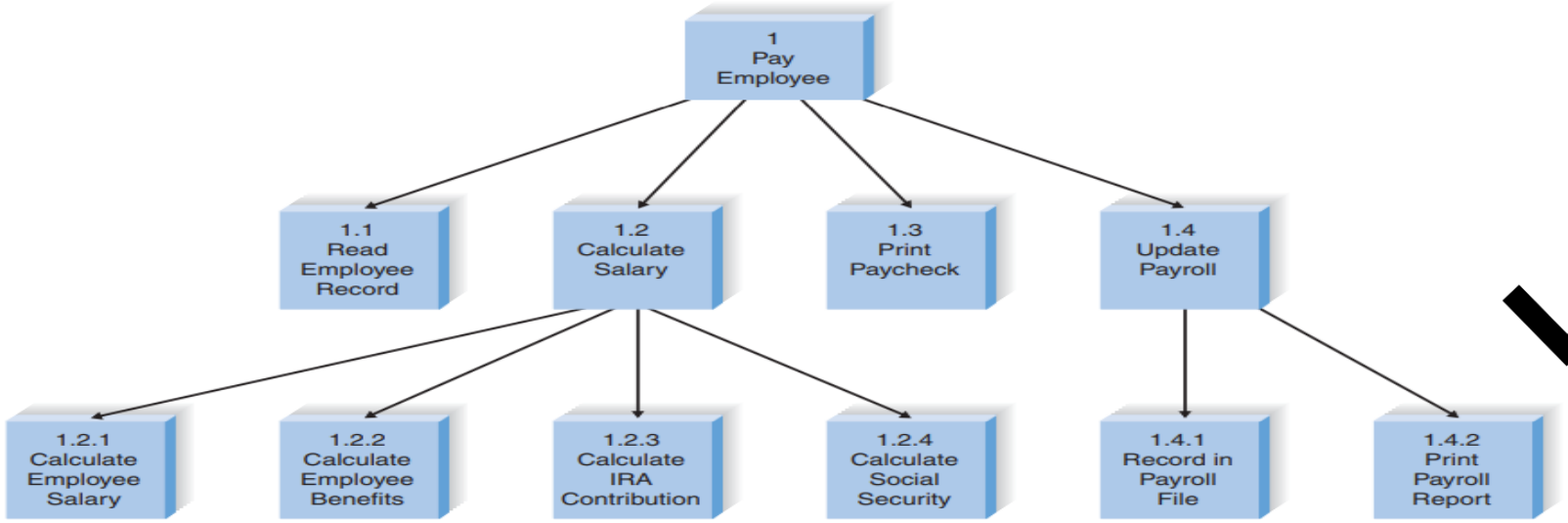
(*b*)

Fan-in / Fan-out

Figure *c* shows high fan-out situation.
Figure *d* shows low fan-out situation.



(*c*)



(*d*)

Assess the Structure Chart for 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

- Once the analyst has communicated the big picture of how the program should be put together, he must describe the *individual modules* in *enough detail* so that programmers can take over and begin writing code.
- Modules on the structure chart are described by the use of program specifications: *written documents that include explicit instructions on how to program pieces of code*.
- Typically, project team members write *one program specification for each module* on the structure chart and then pass them along to programmers.

Program Specification

- There is no formal syntax for program specification.

- Four components are essential for program specification:
  - Program information;
  - Events;
  - Inputs and outputs;
  - Pseudocode.

Program Specification Form

Program Specification 1.1 for ABC System

Module Name:

Purpose:

Progammer:

Date due:

C PowerScript HTML/PHP Visual Basic

Events

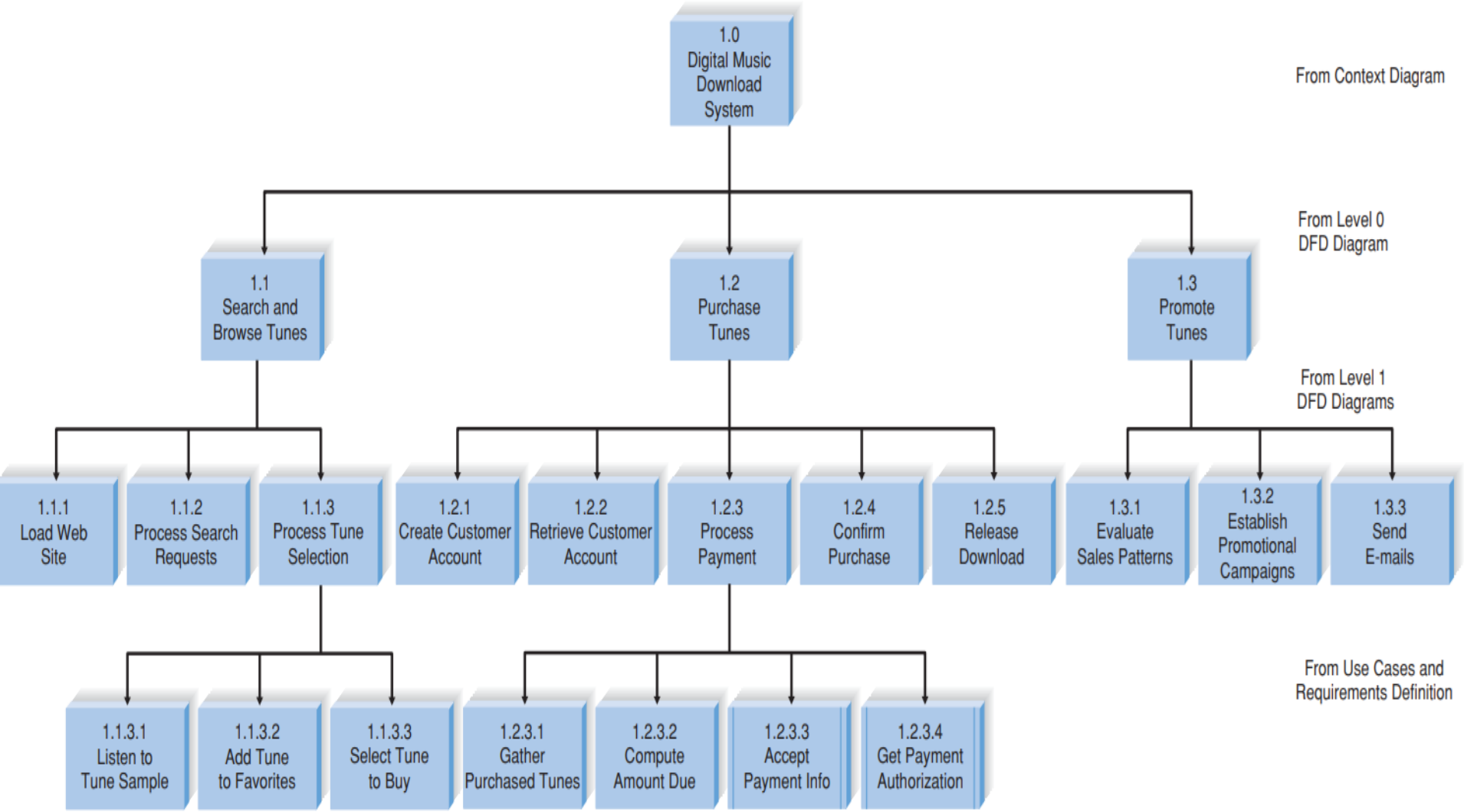
Input Name	Type	Used by	Notes

Output Name	Type	Used by	Notes

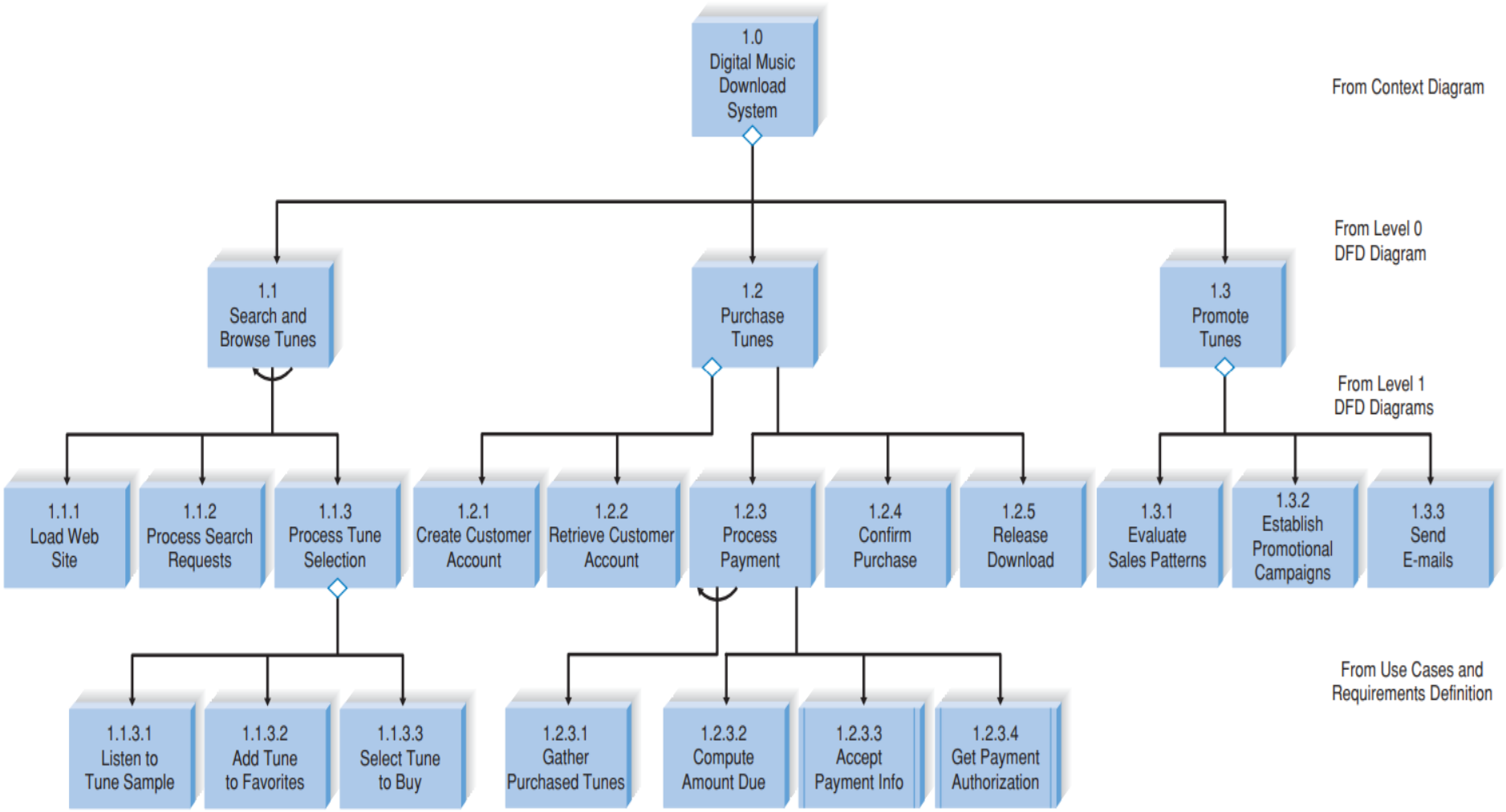
Pseudocode

Other

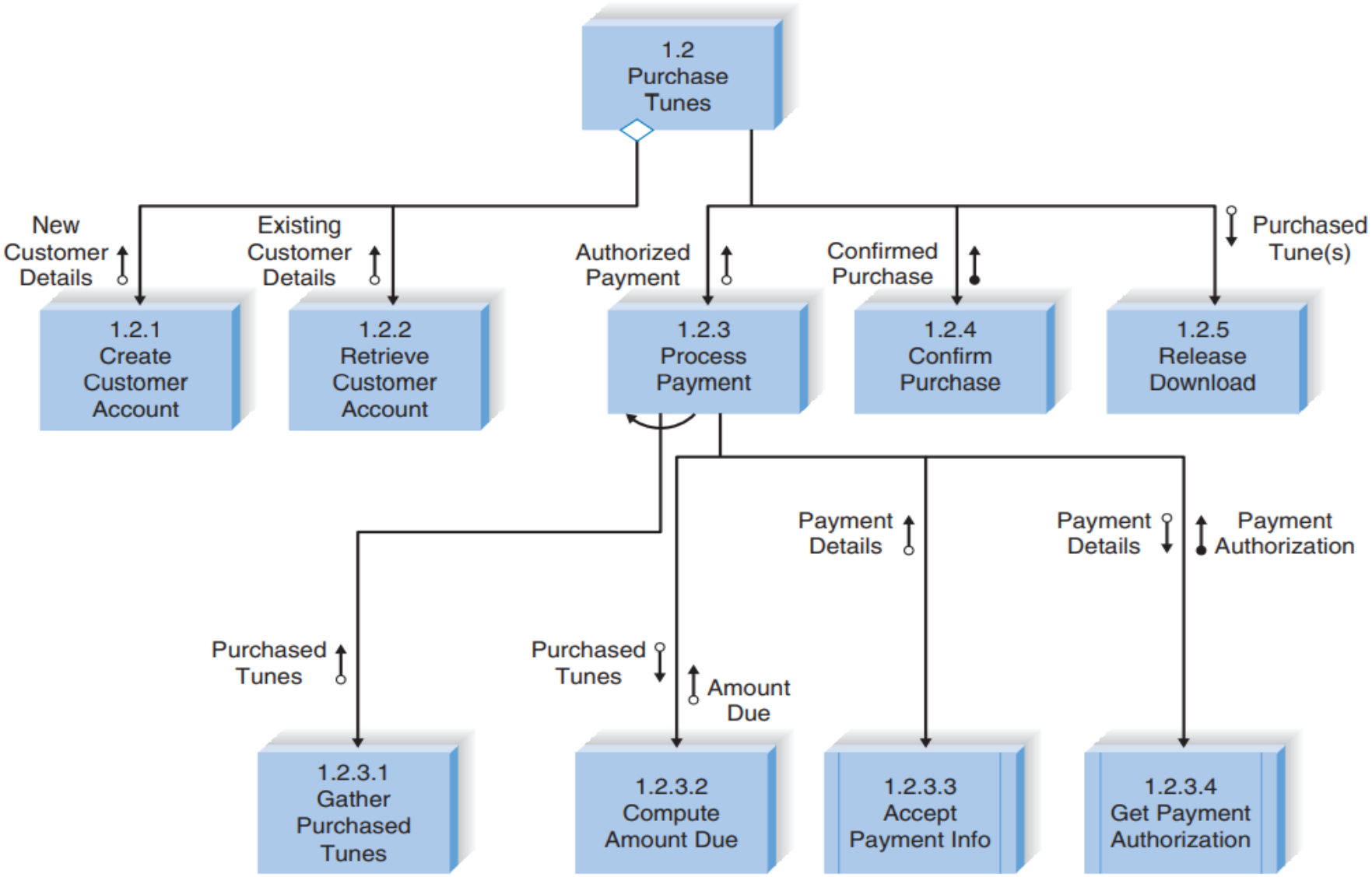
Digital Music Download System



Add Special Connections to the Structure Chart



Add Couples to Structure Chart Module: Purchase Tunes



Program Specification Form

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.

UI Design

- Many users access web material on handheld devices than on desktops.
- Some have iPhones, iPads, netbooks, Kindles —all with different screen resolutions.
- There were two main solutions:
  - Craft ***several versions*** of *one* design and make each have *fixed dimensions* (approach called **adaptive design**).
  - Work on **a *single, flexible* design** that would stretch or shrink to fit the screen (**responsive design**).

Responsive Design

- Responsive design is a **GUI design approach used to create content that adjusts smoothly to various screen sizes**.
- Designers size elements in relative units (%) and apply media queries, so their designs can automatically adapt to the browser space to ensure content consistency across devices.
- Rather than work with absolute units (e.g., pixels) on separate versions, designers were free to focus on just one design and let it flow like a liquid to fill all “containers”.

Responsive Design Core Principles

- **Fluid Grid System**
- **Fluid Image Use**
- **Media Queries**

Fluid Grid System

- Elements occupy the same **percentage** of space however large or small the screen becomes.
- It’s easier if you use a CSS (Cascading Style Sheets) grid system and generator for your design’s base.
- Calculate the target size divided by the context, as a percentage. This is your design feature’s maximum width divided by the maximum width of the users’ browser. When you apply these percentages of features to the required properties in CSS script, you’ll have a *single* design that expands or shrinks according to users’ screen size.

Fluid Image Use

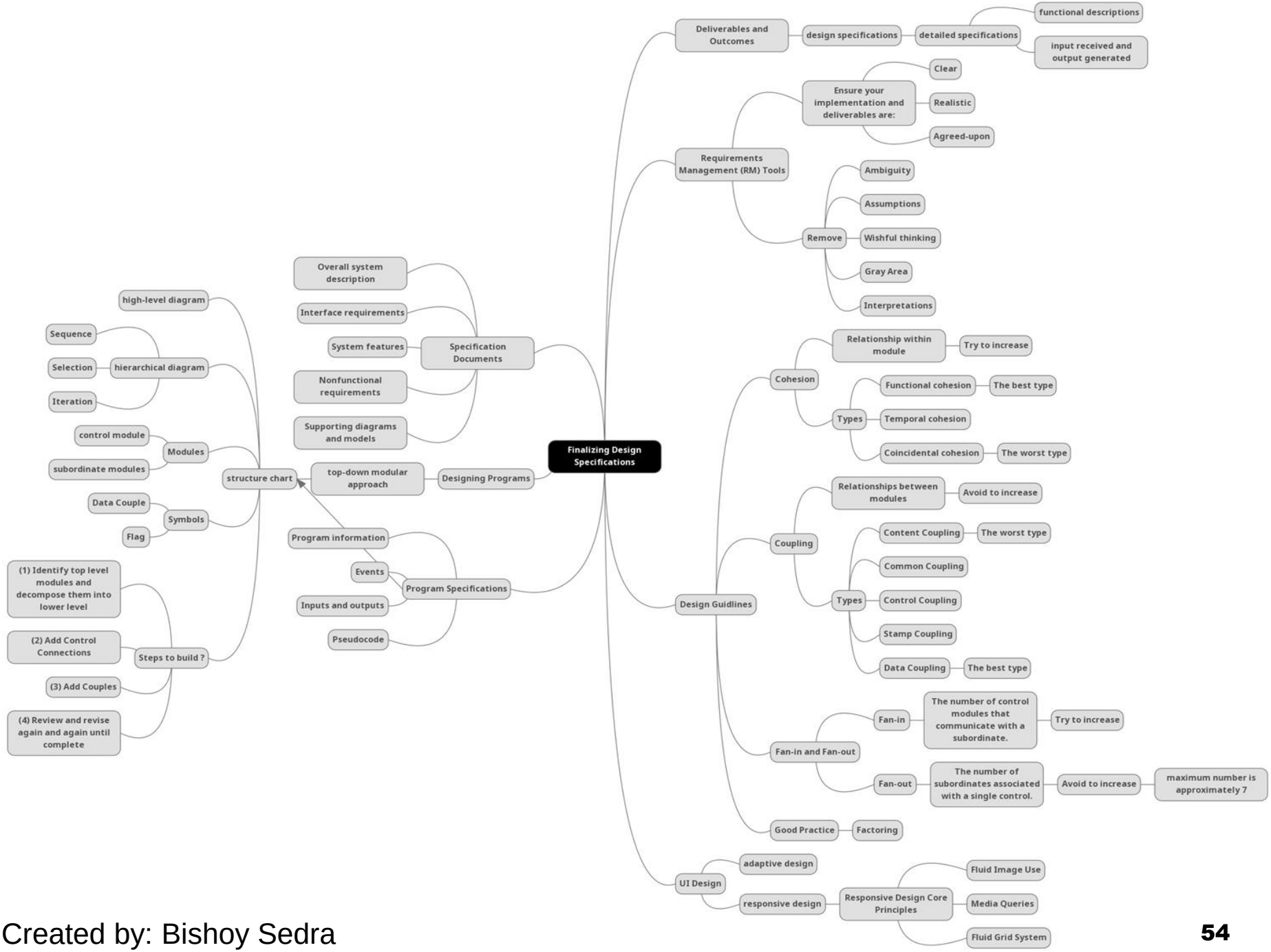
- Unlike text, images aren’t *naturally* fluid. That means they default to the same size and configuration from one device’s screen to the next.
- you need to apply a CSS command

—: img {max-width: 100%;}

to ensure an image shrinks for smaller screens.

Media Queries

- These are filters you use to detect the browsing device's dimensions and make your design appear appropriately.
- With these, you determine what size of screen a user is viewing your design on. These will alter the site layout to meet certain conditions.
- So, based on a screen's width, height, orientation, etc., you can accurately specify how your design will be rendered for different users to see.
- You can choose from a variety of tools, such as Bootstrap, H5P, Gomo and Elucidat.


Finalizing Design Specifications
Deliverables and Outcomes
design specifications
detailed specifications
functional descriptions
input received and output generated
Requirements Management (RM) Tools
Ensure your implementation and deliverables are:
Clear
Realistic
Agreed-upon
Remove
Ambiguity
Assumptions
Wishful thinking
Gray Area
Interpretations
Design Guidlines
Cohesion
Relationship within module
Try to increase
Types
Functional cohesion
The best type
Temporal cohesion
Coincidental cohesion
The worst type
Coupling
Relationships between modules
Avoid to increase
Types
Content Coupling
The worst type
Common Coupling
Control Coupling
Stamp Coupling
Data Coupling
The best type
Fan-in and Fan-out
Fan-in
The number of control modules that communicate with a subordinate.
Try to increase
Fan-out
The number of subordinates associated with a single control.
Avoid to increase
maximum number is approximately 7
Good Practice
Factoring
UI Design
adaptive design
responsive design
Responsive Design Core Principles
Fluid Image Use
Media Queries
Fluid Grid System
Designing Programs
top-down modular approach
structure chart
Specification Documents
Overall system description
Interface requirements
System features
Nonfunctional requirements
Supporting diagrams and models
Program Specifications
Program information
Events
Inputs and outputs
Pseudocode
high-level diagram
hierarchical diagram
Sequence
Selection
Iteration
Modules
control module
subordinate modules
Symbols
Data Couple
Flag
Steps to build ?
(1) Identify top level modules and decompose them into lower level
(2) Add Control Connections
(3) Add Couples
(4) Review and revise again and again until complete

Replace with Keyword(s)

- Written documents that include explicit detailed instructions on how to program pieces of code.
- Tool that makes it easier to keep documents up to date, add additional requirements and link related requirements.
- The process of separating out a function from one module into a module of its own.
- A higher-level component that contains the logic for performing other modules.
- Diagram that shows all components of code in a hierarchical format that implies sequence, iteration and selection.

Differentiate

- Cohesion and coupling
- Fan-in and fan-out
- Adaptive and responsive design

Describe

- Coupling types.
- Cohesion types.
- Aim of structure chart.
- Program specification components.

Complete

- UI elements occupy the same ------ of space however large or small the screen becomes.
- GUI design approach used to create content that adjusts smoothly to various screen sizes is called -----.
- Filters you use to detect the browsing device’s dimensions and make your design appear appropriately are called -----.
- Responsive design core principles include ---------.

Quiz

- Date: Tuesday 19/11/2024
- Time: 11 pm
- Duration: 5 minutes

- Y3 and Y4 students MUST use the link provided for their program
- Y2 students can use any link

Quiz Links

Program	Link
Information Systems	https://forms.office.com/r/dwaWEyarpZ?origin=lprLink
Computer Science	https://forms.office.com/r/ZnP4UTxSWV?origin=lprLink
Scientific Computing	https://forms.office.com/r/5dAuCCUzEC?origin=lprLink
Computer Systems	https://forms.office.com/r/5dAuCCUzEC?origin=lprLink

Good design is a lot like clear thinking made visual.

Edward Tufte

© 2010 Bibliotheca Alexandrina - Graphics Unit