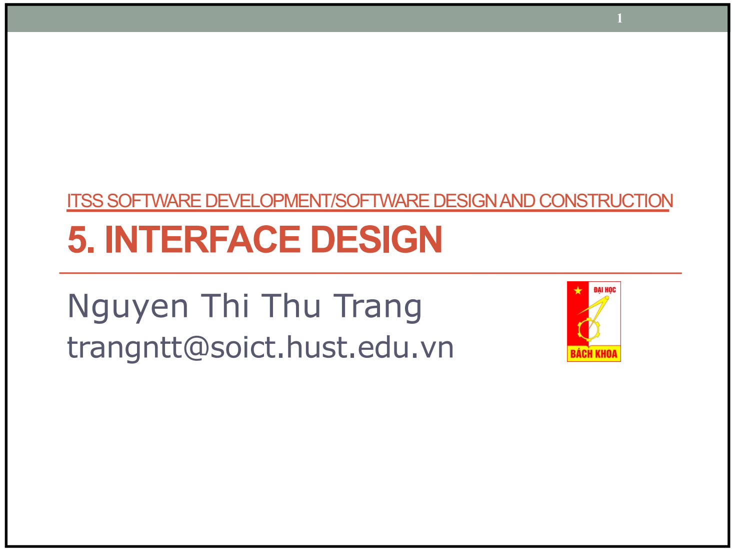
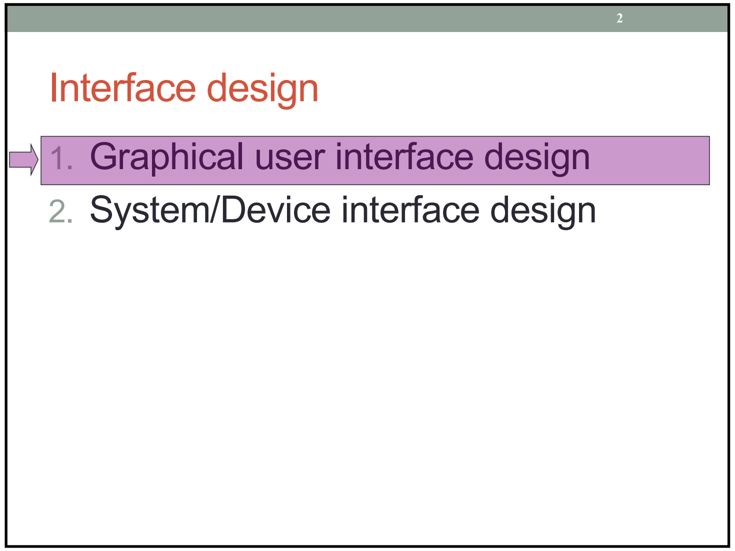


ITSS SOFTWARE DEVELOPMENT/SOFTWARE DESIGN AND CONSTRUCTION

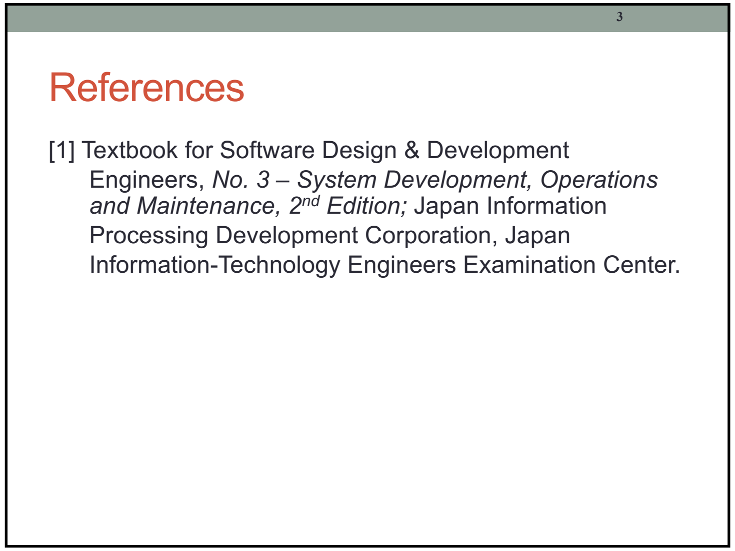
5. INTERFACE DESIGN

Nguyen Thi Thu Trang
trangntt@soict.hust.edu.vn



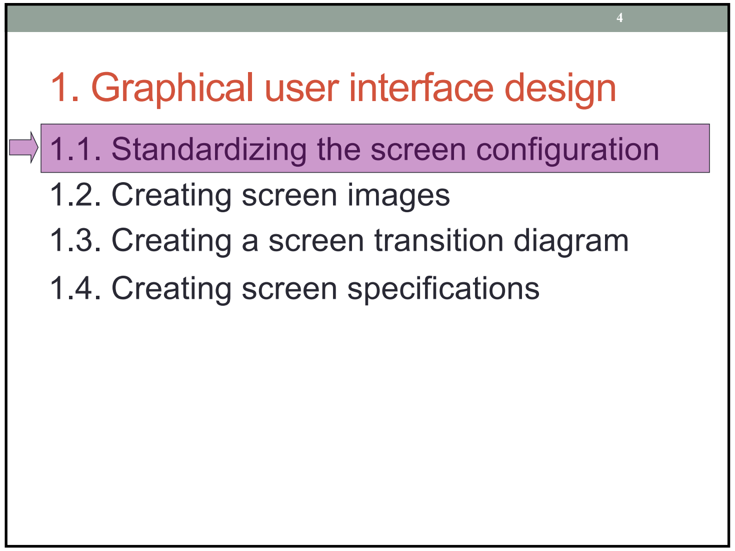
Interface design

1. Graphical user interface design
2. System/Device interface design



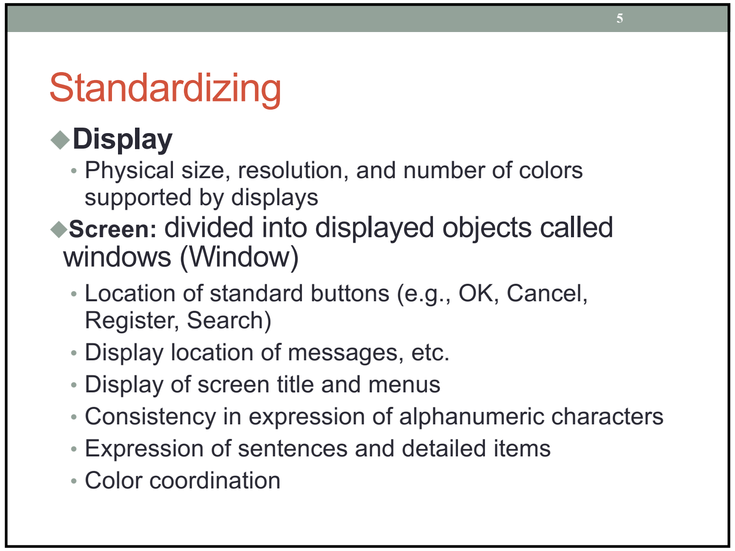
References

[1] Textbook for Software Design & Development Engineers, *No. 3 – System Development, Operations and Maintenance, 2nd Edition;* Japan Information Processing Development Corporation, Japan Information-Technology Engineers Examination Center.



1. Graphical user interface design

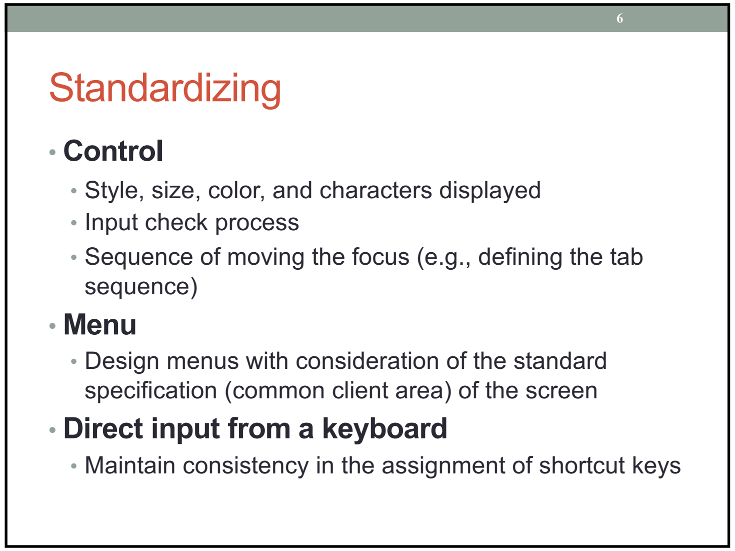
1.1. Standardizing the screen configuration
1.2. Creating screen images
1.3. Creating a screen transition diagram
1.4. Creating screen specifications



Standardizing

- **Display**
  - Physical size, resolution, and number of colors supported by displays
- **Screen:** divided into displayed objects called windows (Window)
  - Location of standard buttons (e.g., OK, Cancel, Register, Search)
  - Display location of messages, etc.
  - Display of screen title and menus
  - Consistency in expression of alphanumeric characters
  - Expression of sentences and detailed items
  - Color coordination

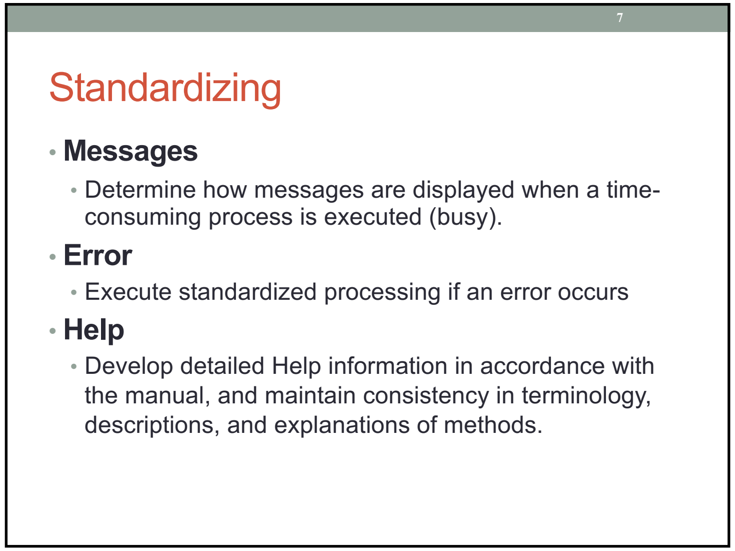
5



Standardizing

- **Control**
  - Style, size, color, and characters displayed
  - Input check process
  - Sequence of moving the focus (e.g., defining the tab sequence)
- **Menu**
  - Design menus with consideration of the standard specification (common client area) of the screen
- **Direct input from a keyboard**
  - Maintain consistency in the assignment of shortcut keys

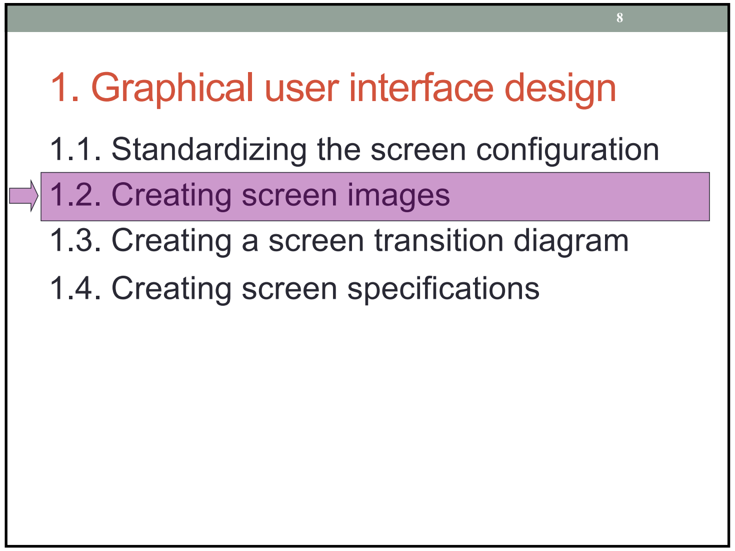
6



Standardizing

- **Messages**
  - Determine how messages are displayed when a time-consuming process is executed (busy).
- **Error**
  - Execute standardized processing if an error occurs
- **Help**
  - Develop detailed Help information in accordance with the manual, and maintain consistency in terminology, descriptions, and explanations of methods.

7



1. Graphical user interface design

1.1. Standardizing the screen configuration

⇨ 1.2. Creating screen images

1.3. Creating a screen transition diagram

1.4. Creating screen specifications

8

From use case

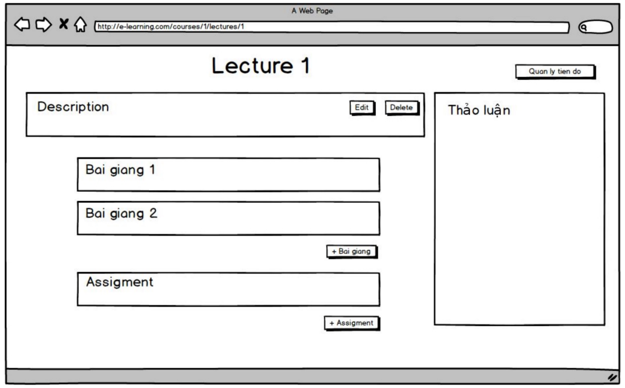
- Based on use case and boundary classes which interact with users
  - Map these boundary classes to screens
- Based on input/output description in use case specification/scenario

=> Design screen using tools

GUI Design tools

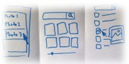
- Simple tools
  - Notepad
  - Microsoft Excel/Powerpoint/Word/FrontPage
- Professional tools
  - Free
    - InVision
    - IDEs: Eclipse, NetBean
  - Commercial
    - Adobe Dreamweaver
    - Axure RP
    - Photoshop
    - IDEs: Visual Studio

Example of wireframe

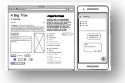


SKETCH, WIREFRAME, MOCKUP, PROTOTYPE

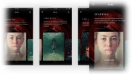
Sketch
- Freehand drawings
- On paper or digital tool
- Very fast to represent concepts



Wireflow
- Wireframe:
  - ✓Mostly black and white UI elements
  - ✓Skeleton for UI design: outline and structure
- Userflow: Simulate steps
- Fast to represent concepts



Mockup
- Detail design for UI
- Mocked data
- Color, logo, icon, image, style



Prototype
- Working model with UX
- Simulate user interaction
- User journey, usability

1. Graphical user interface design

1.1. Standardizing the screen configuration

1.2. Creating screen images

1.3. Creating a screen transition diagram

1.4. Creating screen specifications

13

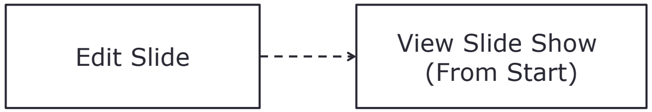
Display transition diagram

- Summarize the correlation of screens in the screen transition diagram
  - Classify the screens into the four patterns by focusing on the transition pattern
  - Link the screens in accordance with the classifications

14

Four transition patterns

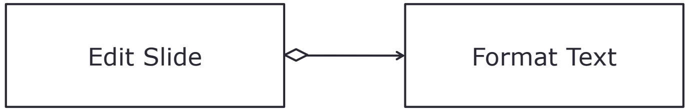
- *1. Simple screen transition:*
  - A conventional simple transition to an independent screen



15

Four transition patterns (2)

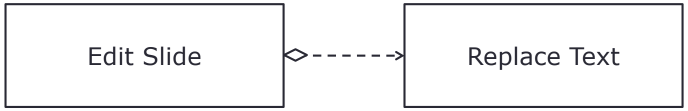
- *2. Transition to a dependent child screen:*
  - Move to a pop-up screen
  - When a child screen is displayed on the parent screen, the underlying parent screen cannot be operated



16

Four transition patterns (3)

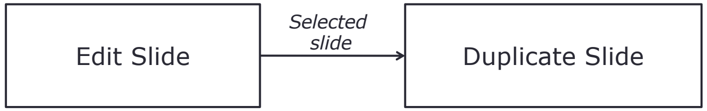
- *3. Transition to an independent child screen:*
  - Move to a pop-up screen,
  - Parent screen and other screens can be operated while the child screen is displayed.



17

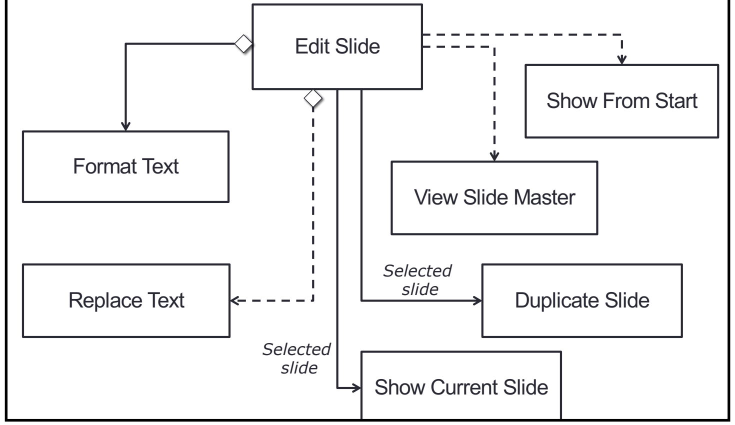
Four transition patterns (4)

- *4. Transition to a dependent screen:*
  - Start a dependent new screen with data



18

Link the screen: Screen transition diagram



19

1. Graphical user interface design

1.1. Standardizing the screen configuration

1.2. Creating screen images

1.3. Creating a screen transition diagram

⇨ 1.4. Creating screen specifications

20

4. Screen specification

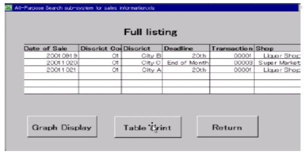
- Decide on a detailed format for a screen specification
- Define field attributes based on the new screen information identified while deciding on screen images and the screen transition diagram

Screen specification

- Screen image
  - This is the screen image to be displayed. If screen images are created in advance with the screen design tool, attach a hardcopy.
- List of functions
  - Defines the names of parts such as the buttons on the screen, and summarizes their functions.
  - Provide descriptions of events for individual screens, attributes of parts, input check specifications and output specifications, etc.
- Defining the field attributes

Liquor sales basic system (general-purpose search subsystem for sales information)		Date of creation	Approved by	Reviewed by	Person in charge
Screen specification	Displaying detail table				

Screen specification example



All-Purpose Search sub-system for sales information.xls

Full listing

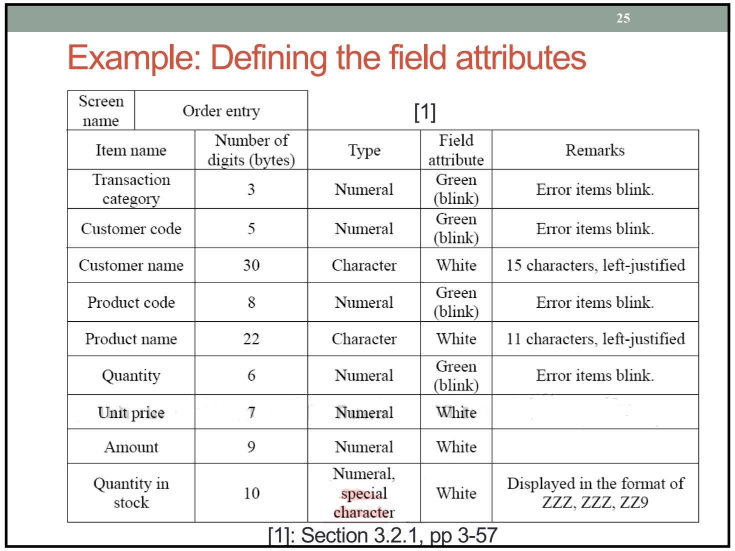
Date of Sale	Discrict Co	Discrict	Deadline	Transaction	Shop
20010819	01	City B	20th	00001	Liquor Shop
20011020	01	City C	End of Month	00003	Super Market
20011021	01	City A	20th	00001	Liquor Shop

Control	Operation	Function
Area for displaying detail table	Initial	-Displays in a table information meeting the conditions defined in the search specification screen. -This follows the setting specified in the display settings screen for display items and sequence of display.
Graph display button	Click	Displays the graph display screen
Table print button	Click	Displays the print preview screen
Return button	Click	Displays the search specification screen

[1]: Section 3.2.1, pp 3-54

Defining the field attributes

- Decide on the field attributes of input and output items
- Summarize them in descriptions of items for screen display.
- The screen consists of multiple fields.
- Each field consists of a one-byte (equivalent to a single character) attribute at the beginning and a variable item

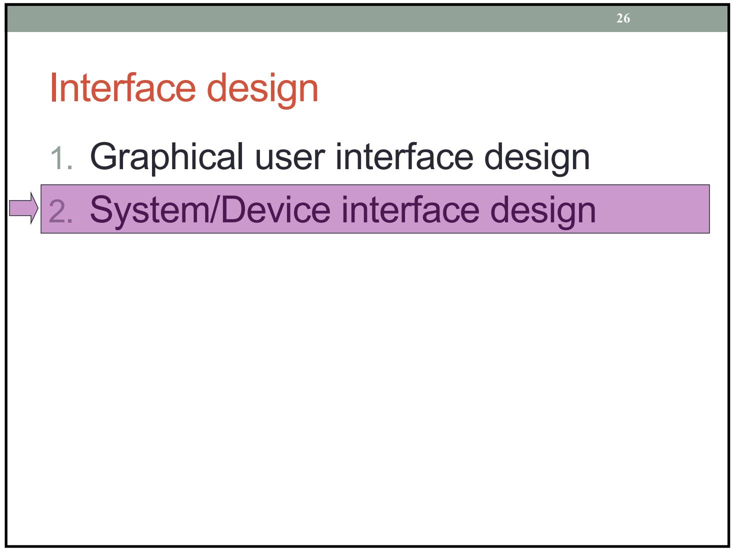


Example: Defining the field attributes

Screen name	Order entry	[1]		
Item name	Number of digits (bytes)	Type	Field attribute	Remarks
Transaction category	3	Numeral	Green (blink)	Error items blink.
Customer code	5	Numeral	Green (blink)	Error items blink.
Customer name	30	Character	White	15 characters, left-justified
Product code	8	Numeral	Green (blink)	Error items blink.
Product name	22	Character	White	11 characters, left-justified
Quantity	6	Numeral	Green (blink)	Error items blink.
Unit price	7	Numeral	White	
Amount	9	Numeral	White	
Quantity in stock	10	Numeral, special character	White	Displayed in the format of ZZZ, ZZZ, ZZ9

[1]: Section 3.2.1, pp 3-57

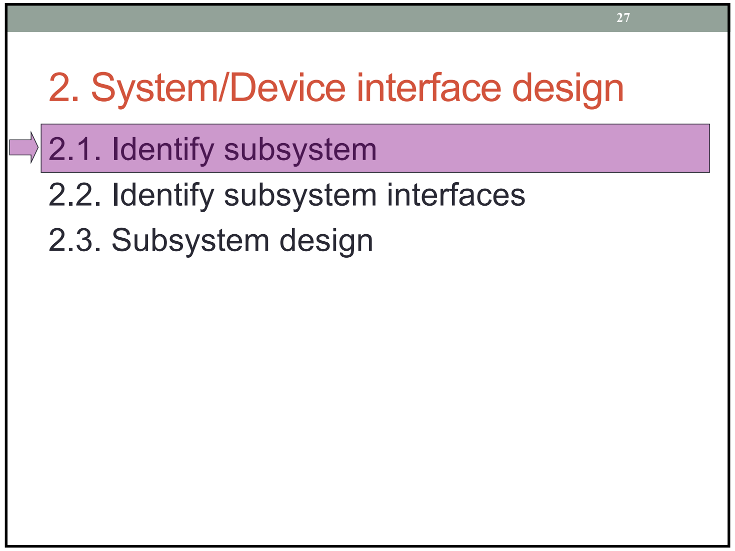
25



Interface design

1. Graphical user interface design
2. System/Device interface design

26



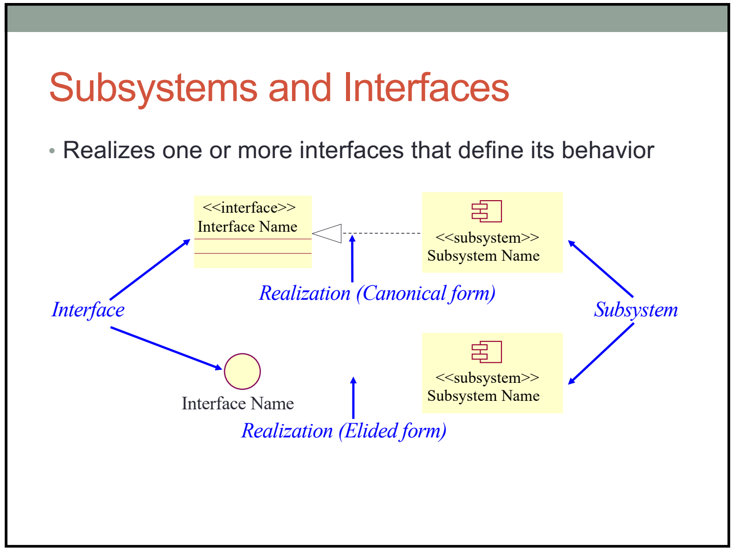
2. System/Device interface design

2.1. Identify subsystem

2.2. Identify subsystem interfaces

2.3. Subsystem design

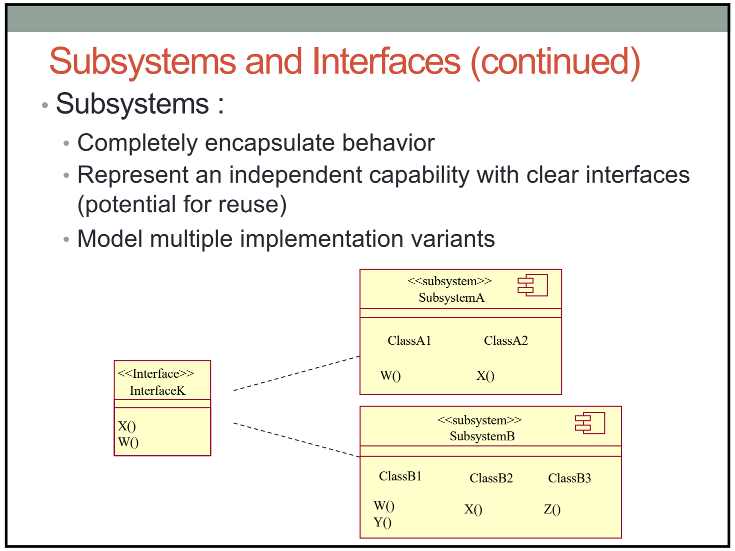
27



Subsystems and Interfaces

- Realizes one or more interfaces that define its behavior

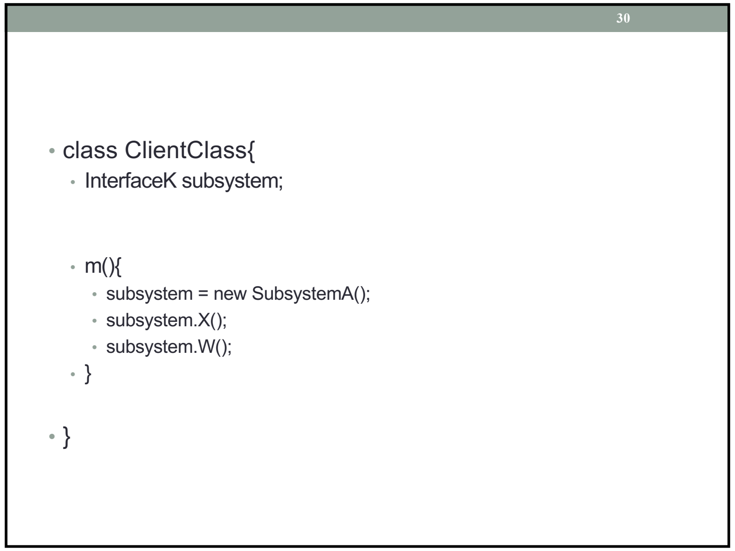
28



Subsystems and Interfaces (continued)

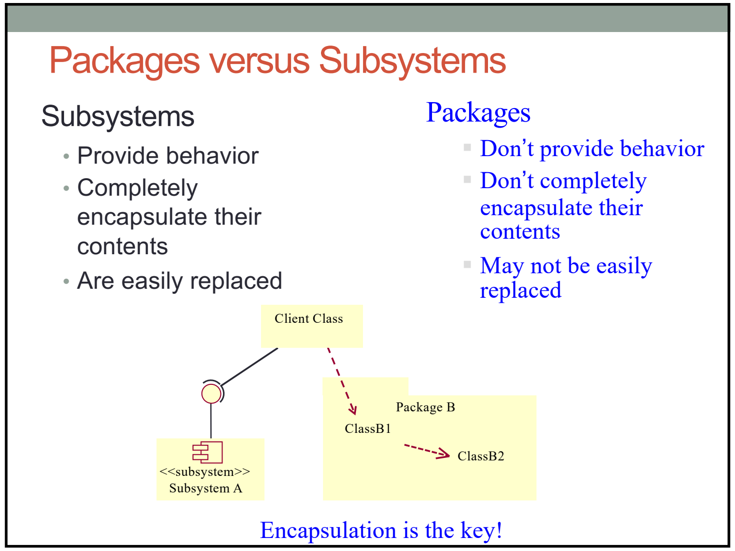
- Subsystems :
  - Completely encapsulate behavior
  - Represent an independent capability with clear interfaces (potential for reuse)
  - Model multiple implementation variants

29



- class ClientClass{
  - InterfaceK subsystem;
  - m(){
    - subsystem = new SubsystemA();
    - subsystem.X();
    - subsystem.W();
  - }
- }

30



Packages versus Subsystems

Subsystems

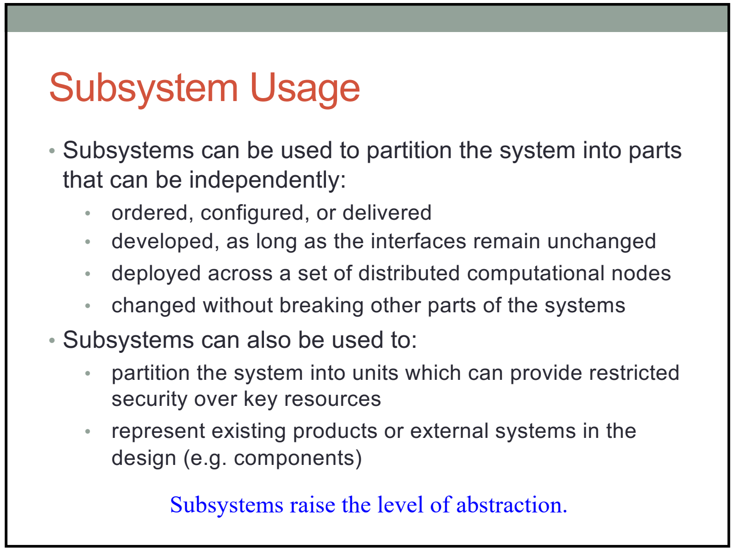
- Provide behavior
- Completely encapsulate their contents
- Are easily replaced

Packages

- Don’t provide behavior
- Don’t completely encapsulate their contents
- May not be easily replaced

Encapsulation is the key!

31

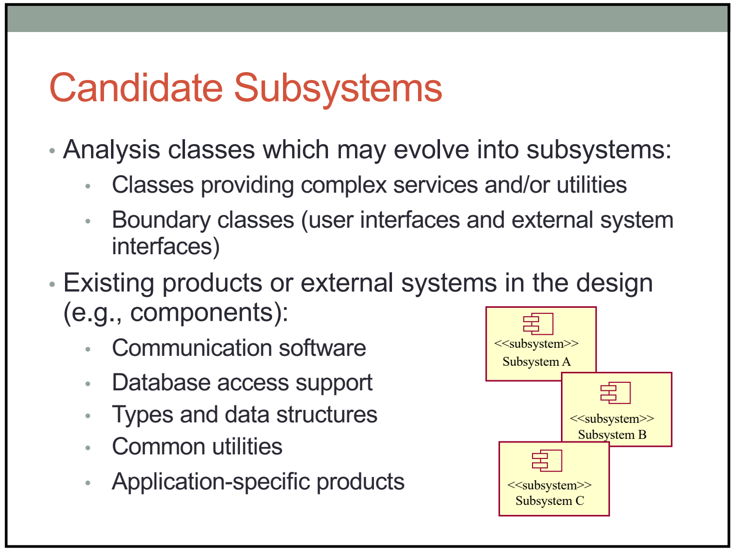


Subsystem Usage

- Subsystems can be used to partition the system into parts that can be independently:
  - ordered, configured, or delivered
  - developed, as long as the interfaces remain unchanged
  - deployed across a set of distributed computational nodes
  - changed without breaking other parts of the systems
- Subsystems can also be used to:
  - partition the system into units which can provide restricted security over key resources
  - represent existing products or external systems in the design (e.g. components)

Subsystems raise the level of abstraction.

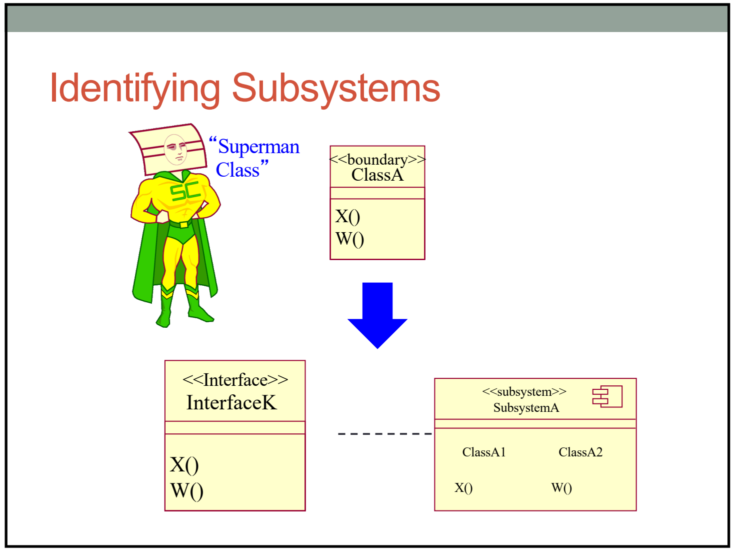
32



Candidate Subsystems

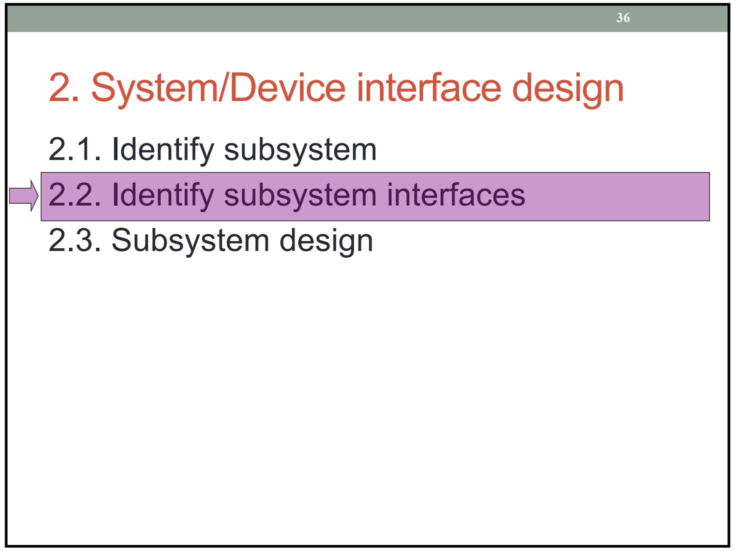
- Analysis classes which may evolve into subsystems:
  - Classes providing complex services and/or utilities
  - Boundary classes (user interfaces and external system interfaces)
- Existing products or external systems in the design (e.g., components):
  - Communication software
  - Database access support
  - Types and data structures
  - Common utilities
  - Application-specific products

34



Identifying Subsystems

35



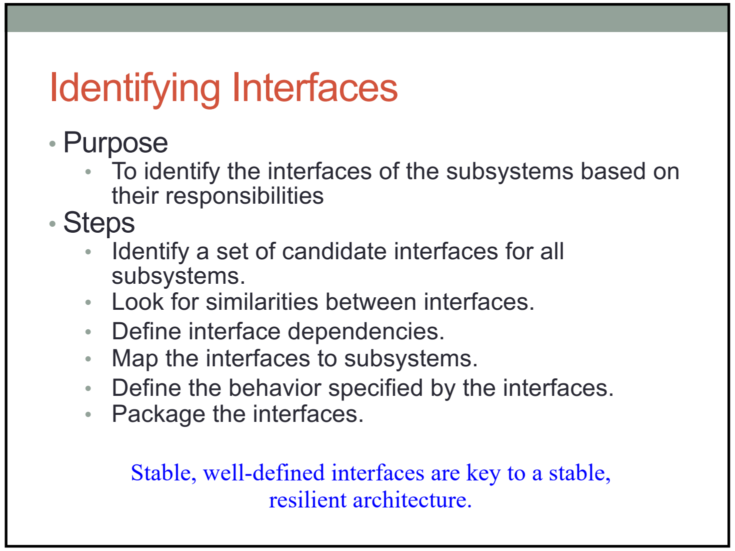
2. System/Device interface design

2.1. Identify subsystem

2.2. Identify subsystem interfaces

2.3. Subsystem design

36



Identifying Interfaces

- Purpose
  - To identify the interfaces of the subsystems based on their responsibilities
- Steps
  - Identify a set of candidate interfaces for all subsystems.
  - Look for similarities between interfaces.
  - Define interface dependencies.
  - Map the interfaces to subsystems.
  - Define the behavior specified by the interfaces.
  - Package the interfaces.

Stable, well-defined interfaces are key to a stable, resilient architecture.

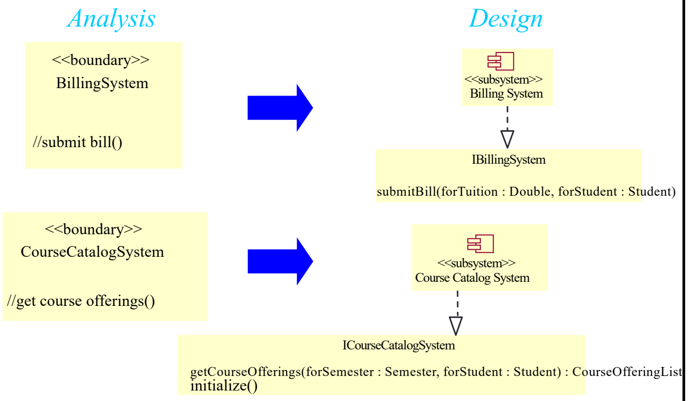
37

Interface Guidelines

- Interface name
  - Reflects role in system
- Interface description
  - Conveys responsibilities
- Operation definition
  - Name should reflect operation result
  - Describes what operation does, all parameters and result
- Interface documentation
  - Package supporting info: sequence and state diagrams, test plans, etc.

38

Example: Design Subsystems and Interfaces

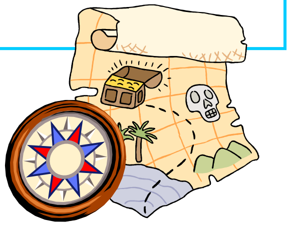


All other analysis classes map directly to design classes.

39

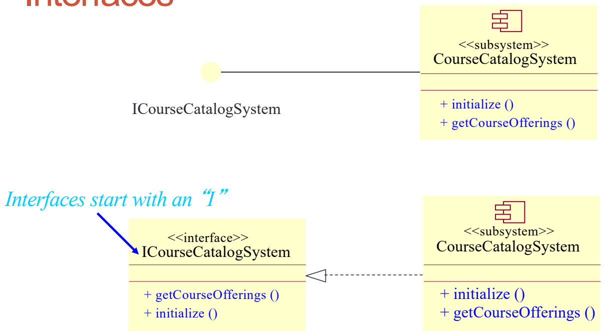
Example: Analysis-Class-To-Design-Element Map

Analysis Class	**Design Element**
CourseCatalogSystem	CourseCatalogSystem Subsystem
BillingSystem	BillingSystem Subsystem
All other analysis classes map directly to design classes	



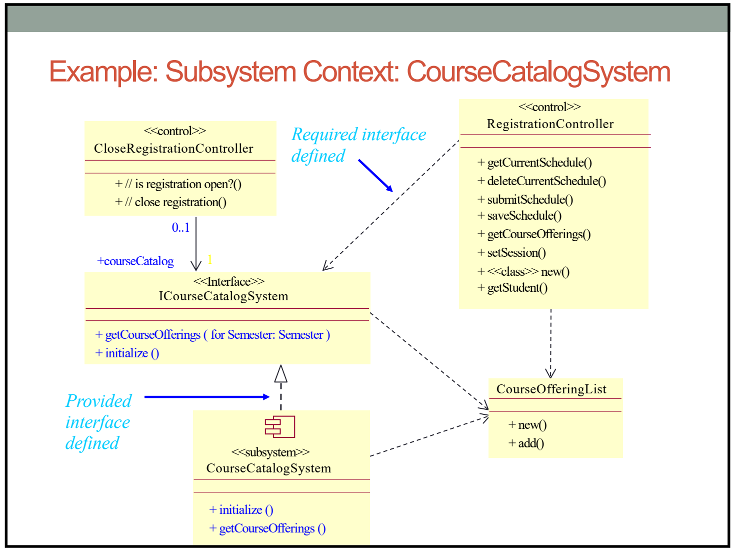
40

Modeling Convention: Subsystems and Interfaces



Interfaces start with an "I"

41



Example: Subsystem Context: CourseCatalogSystem

<<control>>
CloseRegistrationController
+ // is registration open?()
+ // close registration()

0..1
+courseCatalog
1

<<Interface>>
ICourseCatalogSystem
+ getCourseOfferings (for Semester: Semester)
+ initialize ()

Required interface defined

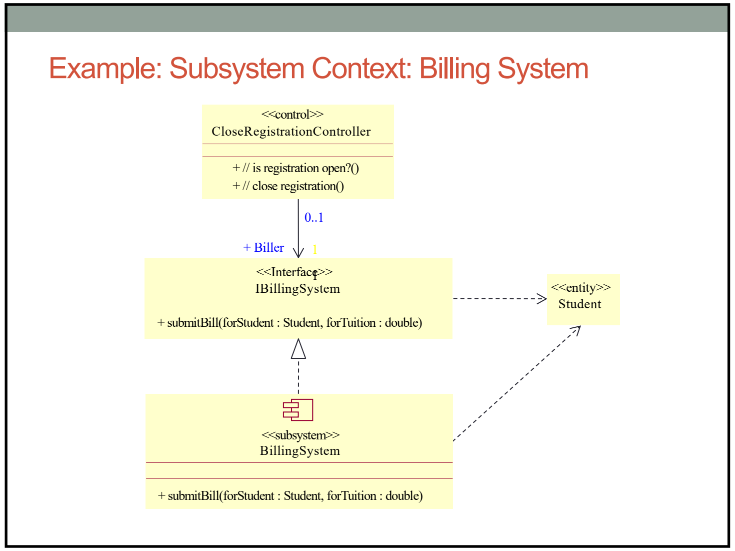
<<control>>
RegistrationController
+ getCurrentSchedule()
+ deleteCurrentSchedule()
+ submitSchedule()
+ saveSchedule()
+ getCourseOfferings()
+ setSession()
+ <<class>> new()
+ getStudent()

Provided interface defined

<<subsystem>>
CourseCatalogSystem
+ initialize ()
+ getCourseOfferings ()

CourseOfferingList
+ new()
+ add()

42



Example: Subsystem Context: Billing System

<<control>>
CloseRegistrationController
+ // is registration open?()
+ // close registration()

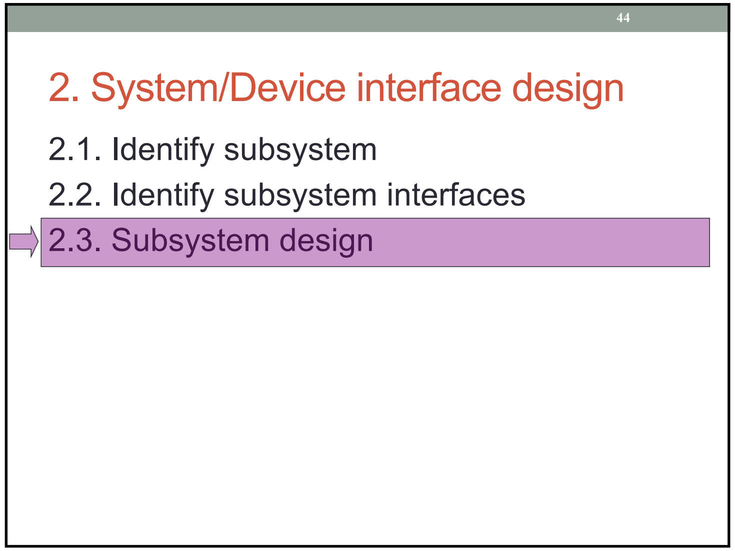
0..1
+ Biller
1

<<Interface>>
IBillingSystem
+ submitBill(forStudent : Student, forTuition : double)

<<entity>>
Student

<<subsystem>>
BillingSystem
+ submitBill(forStudent : Student, forTuition : double)

43



44

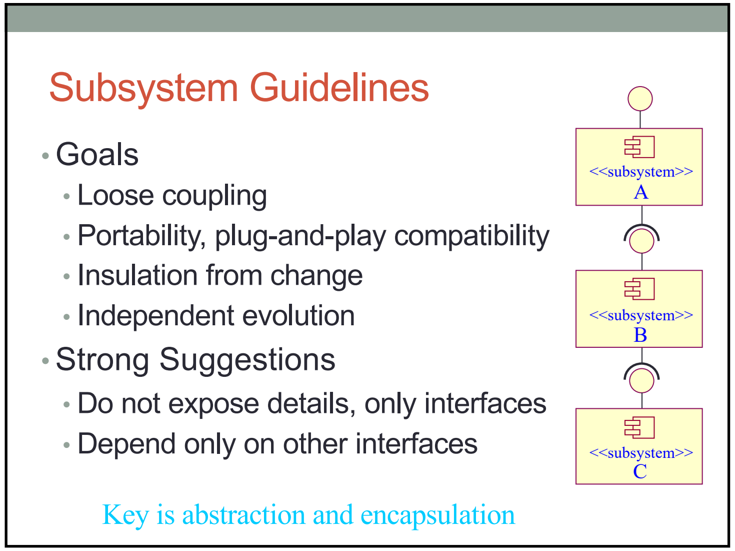
2. System/Device interface design

2.1. Identify subsystem

2.2. Identify subsystem interfaces

2.3. Subsystem design

44



Subsystem Guidelines

- Goals
  - Loose coupling
  - Portability, plug-and-play compatibility
  - Insulation from change
  - Independent evolution
- Strong Suggestions
  - Do not expose details, only interfaces
  - Depend only on other interfaces

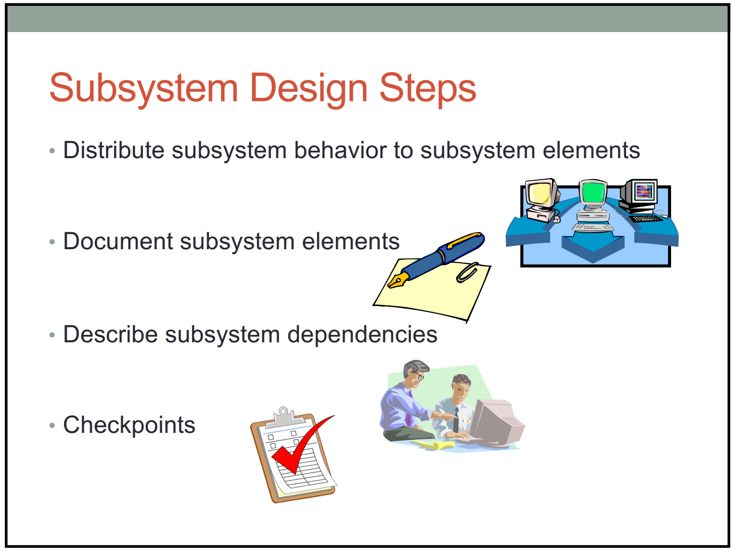
<<subsystem>>
A

<<subsystem>>
B

<<subsystem>>
C

Key is abstraction and encapsulation

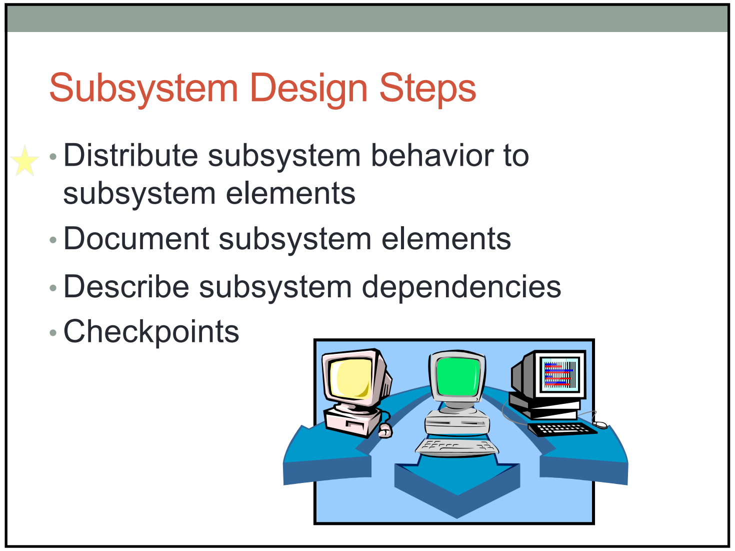
45



Subsystem Design Steps

- Distribute subsystem behavior to subsystem elements
- Document subsystem elements
- Describe subsystem dependencies
- Checkpoints

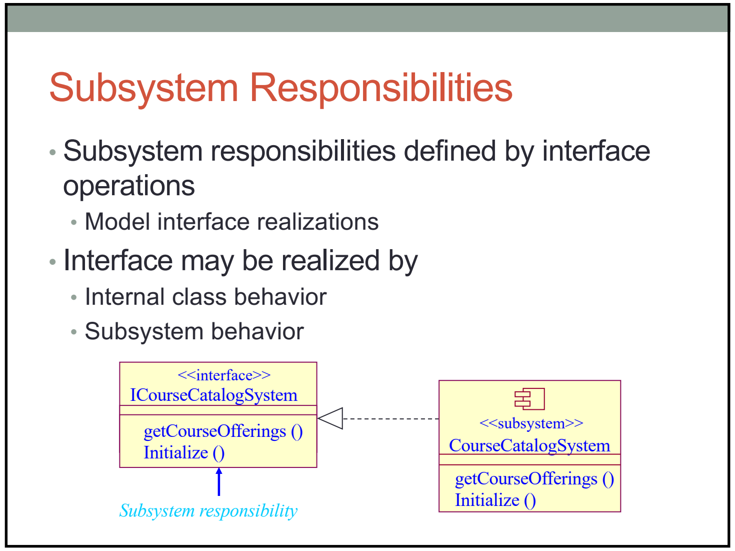
46



Subsystem Design Steps

- Distribute subsystem behavior to subsystem elements
- Document subsystem elements
- Describe subsystem dependencies
- Checkpoints

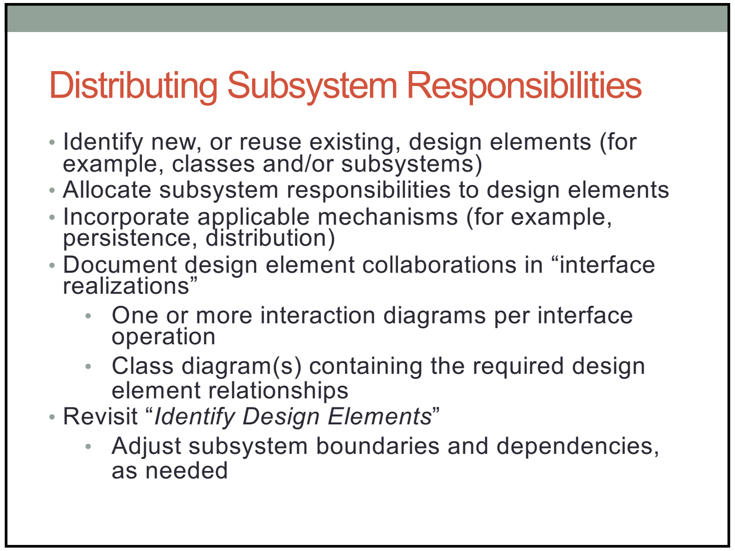
47



Subsystem Responsibilities

- Subsystem responsibilities defined by interface operations
  - Model interface realizations
- Interface may be realized by
  - Internal class behavior
  - Subsystem behavior

48



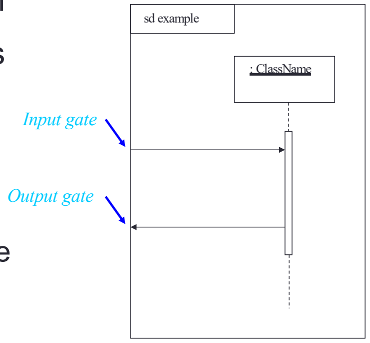
Distributing Subsystem Responsibilities

- Identify new, or reuse existing, design elements (for example, classes and/or subsystems)
- Allocate subsystem responsibilities to design elements
- Incorporate applicable mechanisms (for example, persistence, distribution)
- Document design element collaborations in "interface realizations"
  - One or more interaction diagrams per interface operation
  - Class diagram(s) containing the required design element relationships
- Revisit "*Identify Design Elements*"
  - Adjust subsystem boundaries and dependencies, as needed

49

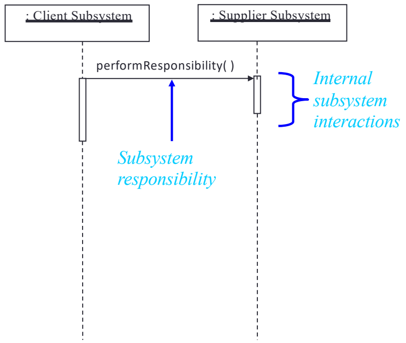
What Are Gates?

- A connection point in an interaction for a message that comes into or goes outside the interaction.
  - A point on the boundary of the sequence diagram
  - The name of the connected message is the name of the gate



50

Subsystem Interaction Diagrams

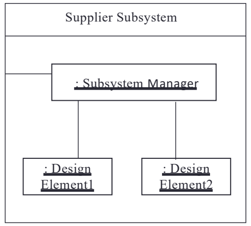


Black box view of subsystems

51

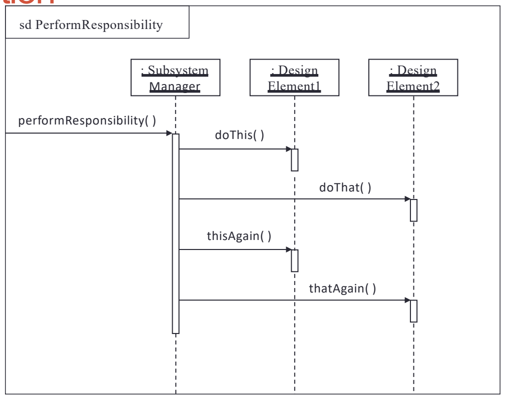
Internal Structure of Supplier Subsystem

- Subsystem Manager coordinates the internal behavior of the subsystem.
- The complete subsystem behavior is distributed amongst the internal Design Element classes.



52

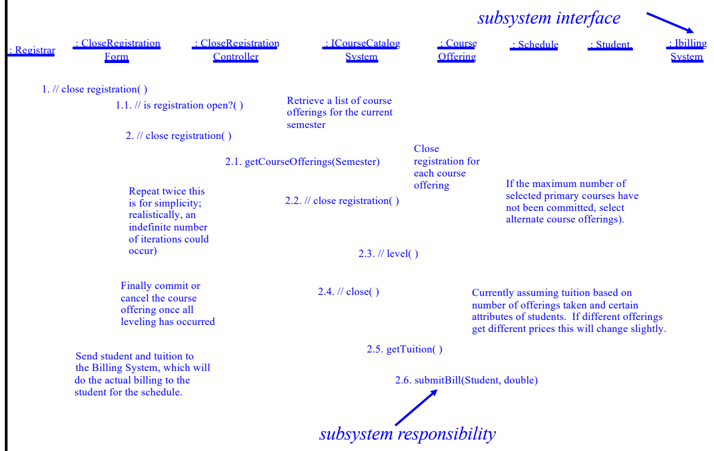
Modeling Convention: Internal Subsystem Interaction



White box view of Supplier Subsystem

53

Example: Billing System Subsystem In Context



subsystem interface

: Registrar

: CloseRegistration Form

: CloseRegistration Controller

: ICourseCatalog System

: Course Offering

: Schedule

: Student

: Ibilling System

1. // close registration( )

1.1. // is registration open?( )

Retrieve a list of course offerings for the current semester

2. // close registration( )

2.1. getCourseOfferings(Semester)

Close registration for each course offering

Repeat twice this is for simplicity; realistically, an indefinite number of iterations could occur)

2.2. // close registration( )

If the maximum number of selected primary courses have not been committed, select alternate course offerings).

2.3. // level( )

Finally commit or cancel the course offering once all leveling has occurred

2.4. // close( )

Currently assuming tuition based on number of offerings taken and certain attributes of students. If different offerings get different prices this will change slightly.

2.5. getTuition( )

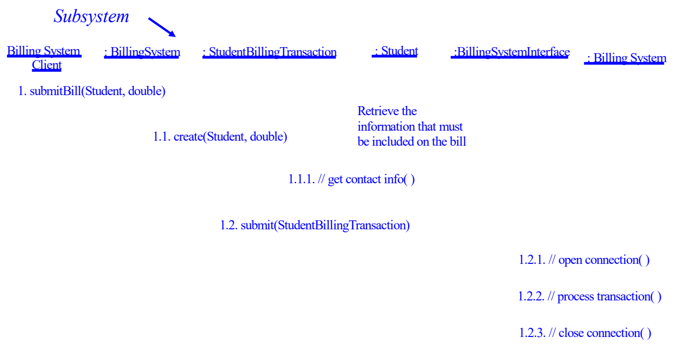
Send student and tuition to the Billing System, which will do the actual billing to the student for the schedule.

2.6. submitBill(Student, double)

subsystem responsibility

54

Example: Local BillingSystem Subsystem Interaction



Subsystem

Billing System Client

: BillingSystem

: StudentBillingTransaction

: Student

:BillingSystemInterface

: Billing System

1. submitBill(Student, double)

1.1. create(Student, double)

Retrieve the information that must be included on the bill

1.1.1. // get contact info( )

1.2. submit(StudentBillingTransaction)

1.2.1. // open connection( )

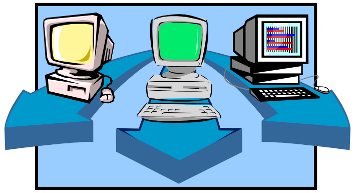
1.2.2. // process transaction( )

1.2.3. // close connection( )

55

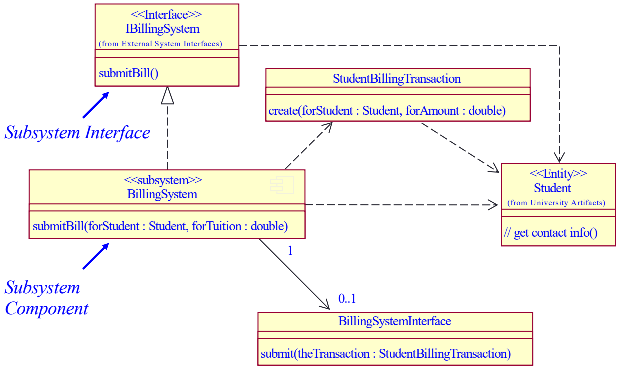
Subsystem Design Steps

- Distribute subsystem behavior to subsystem elements
- Document subsystem elements
- Describe subsystem dependencies
- Checkpoints



56

Example: Billing System Subsystem Elements



<<Interface>>
IBillingSystem
(from External System Interfaces)

submitBill()

StudentBillingTransaction

create(forStudent : Student, forAmount : double)

Subsystem Interface

<<subsystem>>
BillingSystem

submitBill(forStudent : Student, forTuition : double)

<<Entity>>
Student
(from University Artifacts)

// get contact info()

1

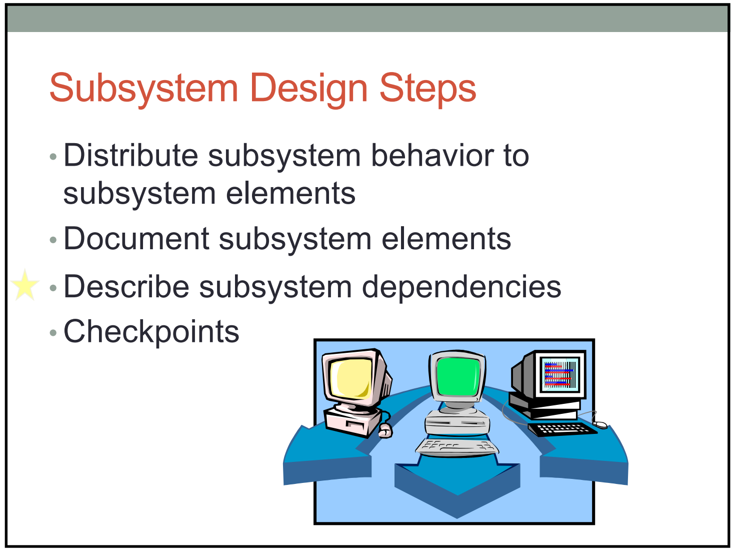
0..1

Subsystem Component

BillingSystemInterface

submit(theTransaction : StudentBillingTransaction)

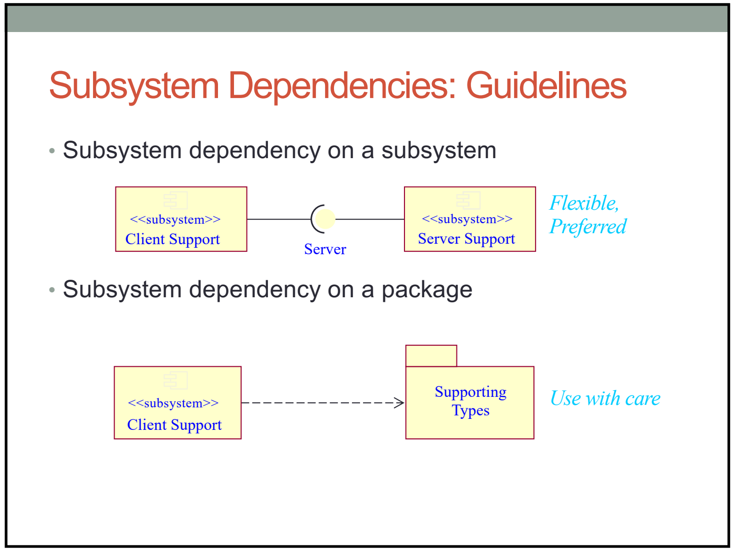
57



Subsystem Design Steps

- Distribute subsystem behavior to subsystem elements
- Document subsystem elements
- ★ Describe subsystem dependencies
- Checkpoints

58



Subsystem Dependencies: Guidelines

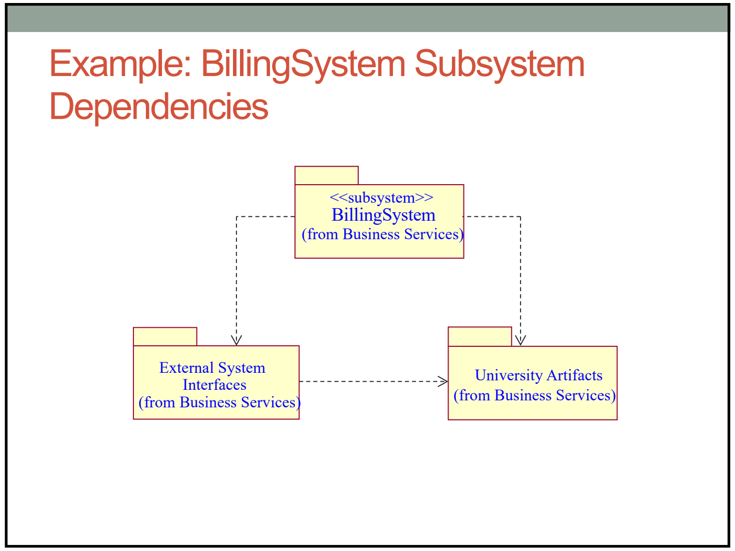
- Subsystem dependency on a subsystem

Flexible, Preferred

- Subsystem dependency on a package

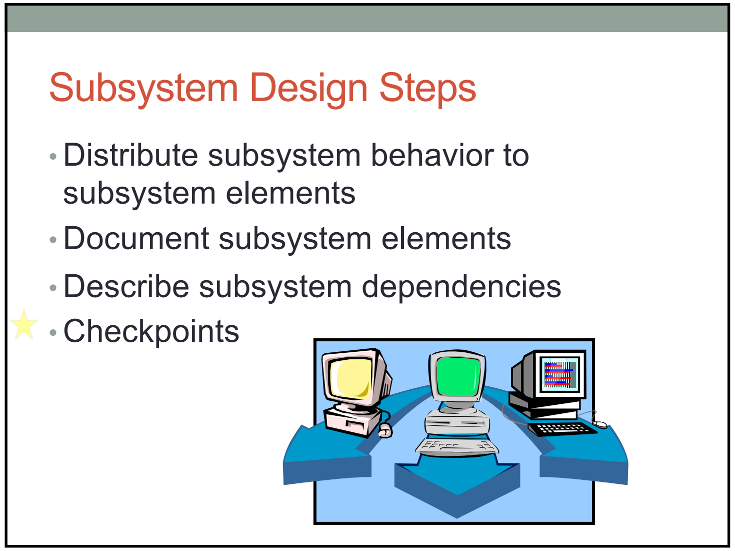
Use with care

59



Example: BillingSystem Subsystem Dependencies

60



Subsystem Design Steps

- Distribute subsystem behavior to subsystem elements
- Document subsystem elements
- Describe subsystem dependencies
- ★ Checkpoints

61

Checkpoints: Design Subsystems

- Is a realization association defined for each interface offered by the subsystem?
- Is a dependency association defined for each interface used by the subsystem?
- Are you sure that none of the elements within the subsystem have public visibility?
- Is each operation on an interface realized by the subsystem documented in a interaction diagram? If not, is the operation realized by a single class, so that it is easy to see that there is a simple 1:1 mapping between the class operation and the interface operation?

62

Review: Subsystem Design

- What is the purpose of Subsystem Design?
- What are gates?
- Why should dependencies on a subsystem be on the subsystem interface?

63

Question?

64