UNIT III: Software Design

CHAPTER 12 DESIGN CONCEPTS
CHAPTER 13 ARCHITECTURAL DESIGN
CHAPTER 14 COMPONENT-LEVEL DESIGN

CHAPTER 12 DESIGN CONCEPTS

12.1 Design within the Context of Software Engineering

12.2 The Design Process

12.3 Design Concepts

12.4 The Design Model

CHAPTER 13 ARCHITECTURAL DESIGN

13.1 Software Architecture

13.3 Architectural Styles

CHAPTER 14 COMPONENT-LEVEL DESIGN

14.1 What Is a Component? (Component Views)

14.2 Designing Class-Based Components

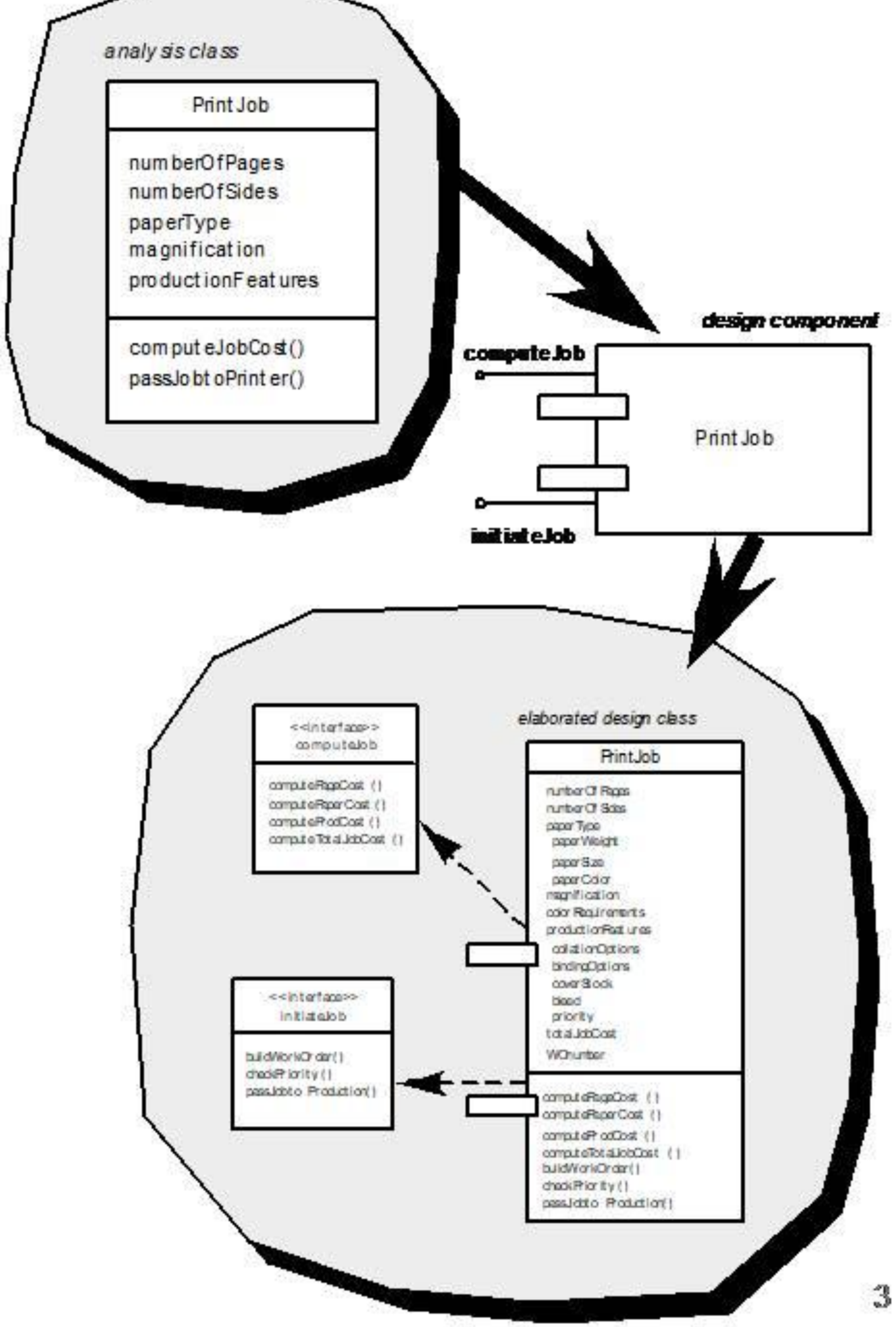
14.3 Conducting Component-Level Design

14.1 What Is a Component? (Component Views)

Component Definition

- *OMG Unified Modeling Language Specification* [OMG01] defines a component as
  - "… a modular, deployable, and replaceable part of a system that encapsulates implementation and exposes a set of interfaces.""
- *OO view:* a component contains a set of collaborating classes
- *Conventional view:* a component contains processing logic, the internal data structures that are required to implement the processing logic, and an interface that enables the component to be invoked and data to be passed to it.

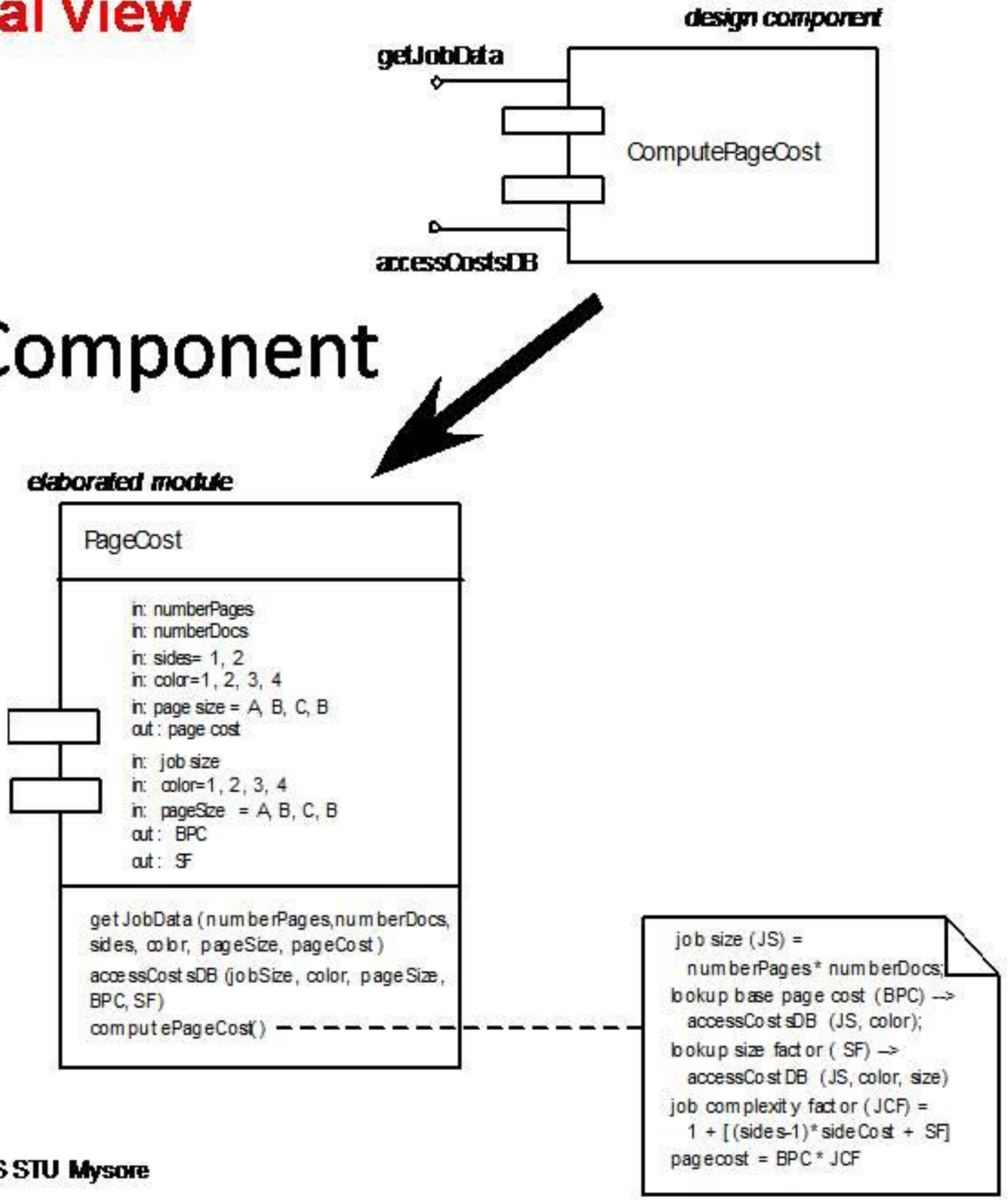
14.1.1 An Object-Oriented View



OO Component

14.1.2 The Traditional View

Conventional Component


design component
getJobData
ComputePageCost
accessCostsDB
elaborated module
PageCost
in: numberPages
in: numberDocs
in: sides= 1, 2
in: color=1, 2, 3, 4
in: page size = A, B, C, B
out : page cost
in: job size
in: color=1, 2, 3, 4
in: pageSize = A, B, C, B
out : BPC
out : SF
getJobData (numberPages,numberDocs, sides, color, pageSize, pageCost)
accessCostsDB (jobSize, color, pageSize, BPC, SF)
computePageCost()
job size (JS) = numberPages * numberDocs;
lookup base page cost (BPC) --> accessCostsDB (JS, color);
lookup size factor (SF) --> accessCostDB (JS, color, size)
job complexity factor (JCF) = 1 + [(sides-1) * sideCost + SF]
pagecost = BPC * JCF

14.1.3 A Process-Related View

- Object-oriented and traditional views of component-level design assume that **component is designed from scratch.**

- Another approach: the need to build systems that make use of existing **software components or design patterns**. A catalog of proven design or code-level components is made available for design and they are used to populate the architecture.

- These components have been created with reusability in mind, a **complete description** of their interface, the function(s) they perform, and the communication and collaboration they require are all available.

- They help in component-based software engineering **(CBSE).**

14.2 Designing Class-based Components

14.2.1 Basic Design Principles (First 4-OO Software Engineering)

- **The Open-Closed Principle (OCP).** *"A module [component] should be open for extension but closed for modification.*(New Sensor added)
- **The Liskov Substitution Principle (LSP).** *"Subclasses should be substitutable for their base classes.* (Pre and post condition)
- **Dependency Inversion Principle (DIP).** *"Depend on abstractions. Do not depend on concretions."* (Virtual class like interface, extension difficult)
- **The Interface Segregation Principle (ISP).** *"Many client-specific interfaces are better than one general purpose interface.*
- **The Release Reuse Equivalency Principle (REP).** *"The granule of reuse is the granule of release."*
- **The Common Closure Principle (CCP).** *"Classes that change together belong together."*
- **The Common Reuse Principle (CRP).** *"Classes that aren't reused together should not be grouped together."*

14.2 Designing Class-based Components

14.2.2 Component-Level Design Guidelines for

- **Components**
  - Naming conventions should be established for components that are specified as part of the architectural model and then refined and elaborated as part of the component-level model.

- **Interfaces**
  - Interfaces provide important information about communication and collaboration (as well as helping us to achieve the OPC).

- **Dependencies and Inheritance**
  - it is a good idea to model dependencies from left to right and inheritance from bottom (derived classes) to top (base classes).

14.2 Designing Class-based Components

14.2.3 Cohesion

- **Conventional view:**
  - the "single-mindedness" of a module

- **OO view:**
  - *cohesion* implies that a component or class encapsulates only attributes and operations that are closely related to one another and to the class or component itself

- **Levels of cohesion (Types)**
  - Functional (1 function)
  - Layer (Higher layer access lower, no vice versa ex: Safe home telephone call if they don't respond to alram)
  - Communicational (Operations accessing same data are in 1 class)
  - Sequential
  - Procedural
  - Temporal
  - utility

14.2 Designing Class-based Components

14.2.4 Coupling

- **Conventional view:**
  - The degree to which a component is connected to other components and to the external world
- **OO view:**
  - a qualitative measure of the degree to which classes are connected to one another
- **Level of coupling**
  - Content (content modification permission)
  - Common
  - Control (A invokes B and passes control to B)
  - Stamp
  - Data
  - Routine call
  - Type use
  - Inclusion or import
  - External (Infrastructure components : OS)

14.3 Conducting Component-Level Design

Transform requirements and architectural models into a design representation. The following steps represent a **task set for component-level design**, when it is applied for an OO system.

- **Step 1. Identify all design classes that correspond to the problem** domain based on analysis and architectural model.

- **Step 2. Identify all design classes that correspond to the infrastructure domain** (GUI, OS, Object & data management components)

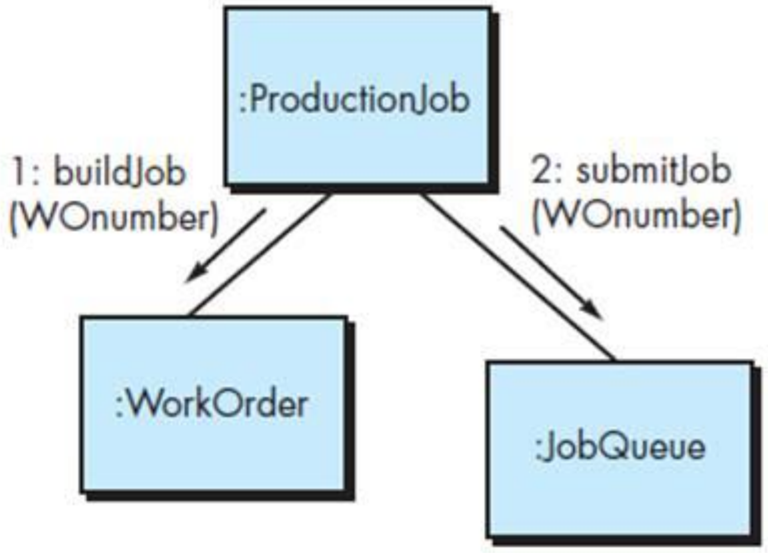
- **Step 3. Elaborate all design classes that are not acquired as reusable components.** Elaboration requires that all interfaces, attributes, and operations necessary to implement the class be described in detail. Design heuristics (e.g., component cohesion and coupling) must be considered as this task is conducted.

14.3 Conducting Component-Level Design (Cont'd)

- **Step 3a. Specify message details when classes or component collaborate.**

FIGURE 14.6

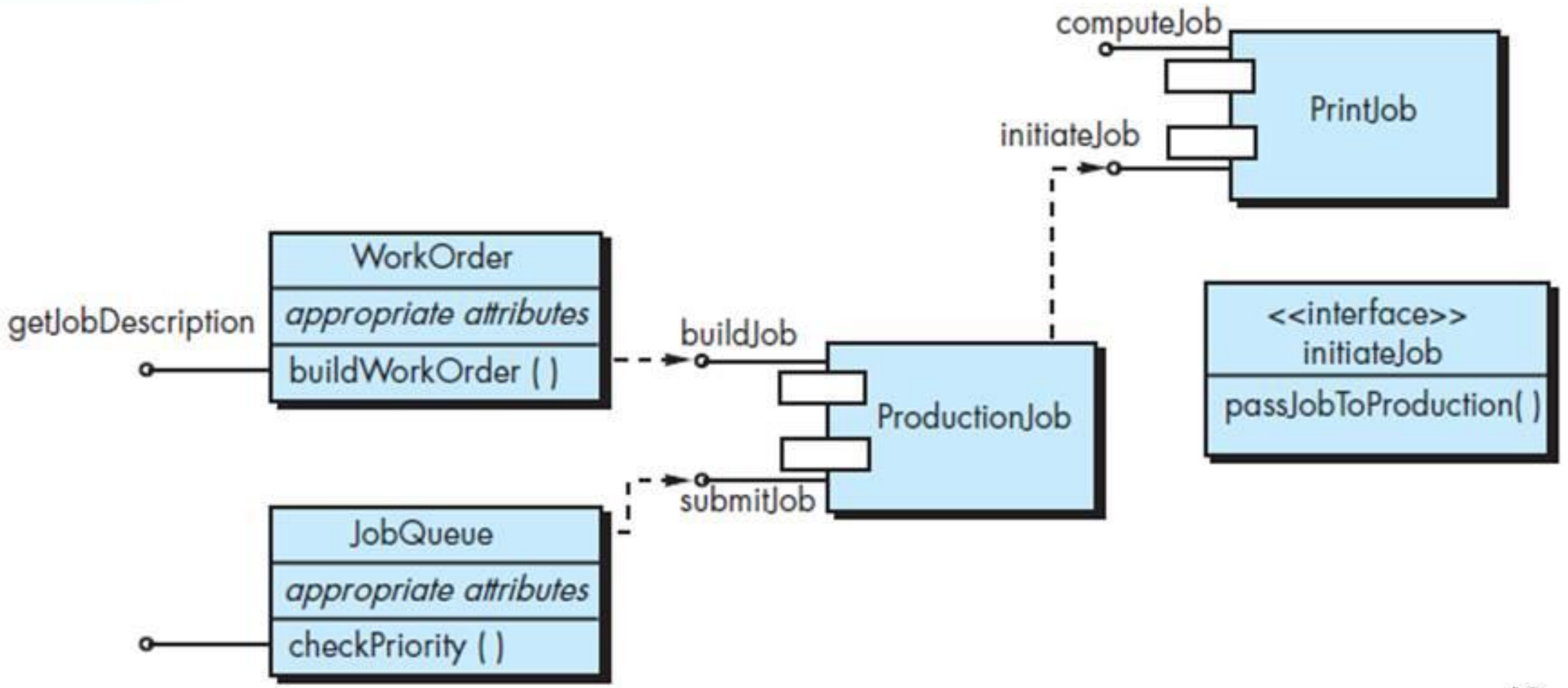
Collaboration diagram with messaging



14.3 Conducting Component-Level Design (Cont'd)

- **Step 3b. Identify appropriate interfaces for each component.** Interface is "a group of externally visible operations, provides a controlled connection between design classes. Single minded classes and cohesion.

FIGURE 14.7 Refactoring interfaces and class definitions for PrintJob



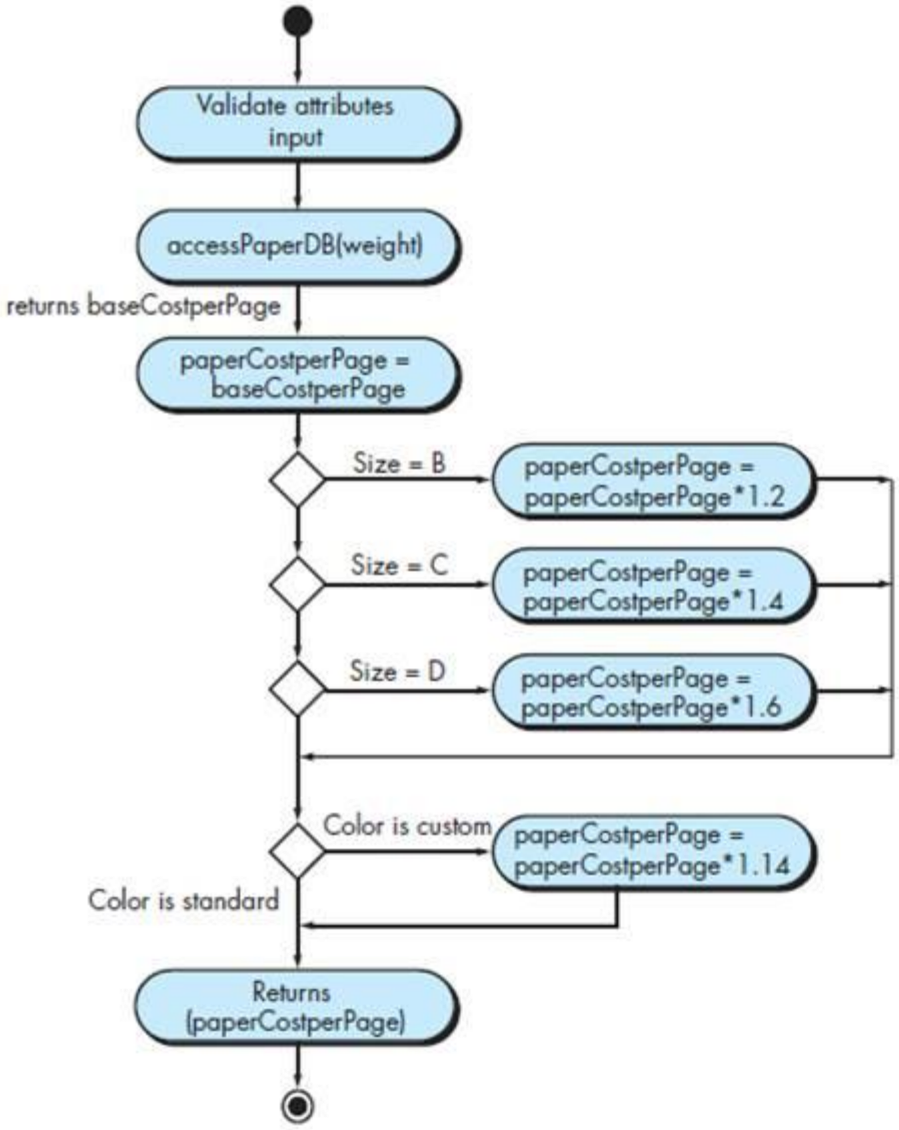
14.3 Conducting Component-Level Design (Cont'd)

- **Step 3c.** Elaborate attributes and define data types and data structures required to implement them.

- **Step 3d.** Describe processing flow within each operation in detail. Fig. 14.8

FIGURE 14.8

UML activity diagram for *compute-PaperCost()*


Validate attributes input
accessPaperDB(weight)
returns baseCostperPage
paperCostperPage = baseCostperPage
Size = B
paperCostperPage = paperCostperPage*1.2
Size = C
paperCostperPage = paperCostperPage*1.4
Size = D
paperCostperPage = paperCostperPage*1.6
Color is custom
paperCostperPage = paperCostperPage*1.14
Color is standard
Returns (paperCostperPage)

14.3 Conducting Component-Level Design

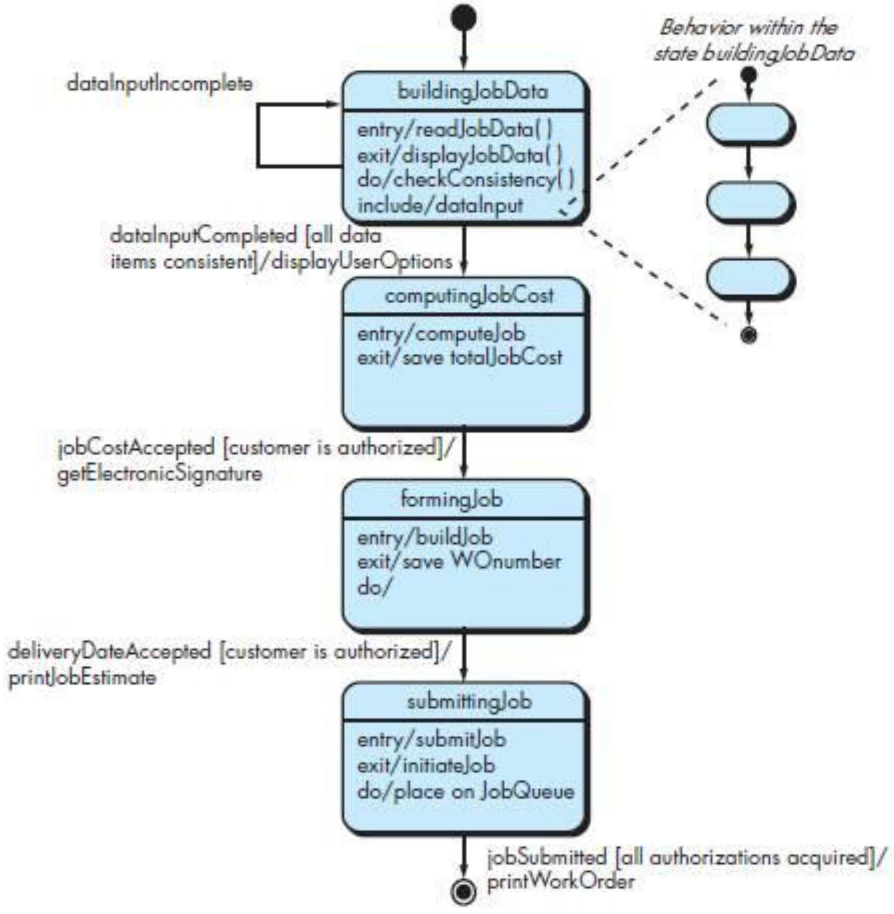
- **Step 4. Describe persistent data sources (databases and files) and identify the classes required to manage them.** Databases and files normally transcend the design description of an individual component.

14.3 Conducting Component-Level Design

- **Step 5. Develop and elaborate behavioral representations for a class or component.** The dynamic behavior of an object (an instantiation of a design class as the program executes) is affected by events that are external to it and the current state (mode of behavior) of the object. Examine all use cases Figure 14.9 .

FIGURE 14.9

Statechart fragment for PrintJob class



Behavior within the state buildingJobData
dataInputIncomplete
buildingJobData
entry/readJobData( )
exit/displayJobData( )
do/checkConsistency( )
include/dataInput
dataInputCompleted [all data items consistent]/displayUserOptions
computingJobCost
entry/computeJob
exit/save totalJobCost
jobCostAccepted [customer is authorized]/ getElectronicSignature
formingJob
entry/buildJob
exit/save WOnumber
do/
deliveryDateAccepted [customer is authorized]/ printJobEstimate
submittingJob
entry/submitJob
exit/initiateJob
do/place on JobQueue
jobSubmitted [all authorizations acquired]/ printWorkOrder

14.3 Conducting Component-Level Design

- **Step 6. Elaborate deployment diagrams to provide additional implementation detail.** Elaborated to represent the location of key packages of components. In some cases, deployment diagrams are elaborated into instance form at this time. **This means that the specific hardware and operating system environment(s) that will be used is (are) specified and the location of component packages within this environment is indicated.**

- **Step 7. Factor every component-level design representation and always consider alternatives.**