10. Component Based Software Engineering

IT 3106– Object Oriented Analysis and Design

Level II - Semester 3

Overview

In this section students will learn

- the key activities in the component-based software engineering (CBSE) process,
- what is meant by a software component that may be included in a program as an executable element;
- the key elements of software component models and the support provided by middleware for these models;
- three different types of component composition and some of the problems that have to be resolved when components are composed to create new components or systems.

Intended Learning Outcomes

At the end of this lesson students will be able to

- understand the component-based software engineering (CBSE) process concerned with developing standardized components based on a component model, and composing these into application systems;
- know the principal activities in the CBSE process for reuse and the CBSE process with reuse;
- describe the three different types of component composition and some of the problems that have to be resolved when components are composed to create new components or systems.

List of sub topics

10 Component Based Software Engineering (3 hours)

- 10.1 Introduction to Component Based Software Engineering (CBSE) [Ref2: 465-467]
- 10.2 Components and Component Models [Ref 2: Pg. 467-473]
- 10.3 CBSE Processes [Ref 2: Pg. 473-480]
  - 10.3.1 CBSE for Reuse
  - 10.3.2 CBSE with Reuse
- 10.4 Component Composition [Ref 2: Pg. 480-486]

10.1 Introduction to Component Based Software Engineering (CBSE)

- Many new systems – now developed by configuring COTS (Commercial off the shelf)
- Cannot use if it does not meet their requirements.
- For custom software, CBSE is an effective , reuse oriented way to develop new enterprise systems.

10.1 Introduction to Component Based Software Engineering (CBSE)

- CBSE emerged in late 1990s.

Motivation:

- OO development had not led to extensive reuse as originally suggested.
- Single object classes were too detailed and specific to reuse.
- Selling or distributing objects as individual reusable components were practically impossible.

Component- based Software Engineering (CBSE).

- Components are higher level abstractions than objects.
- They are defined by their interfaces.
- Larger than individual objects.

CBSE: is the process of

- **defining,**
- **Implementing**
- **Integrating**

Loosely coupled , independent components into systems.

Essentials of Component- based Software Engineering (CBSE).

1. **Independent Components that are completely specified by their interfaces.**
   - There should be a clear separation between the component interface and its implementation.

Essentials of
Component- based Software Engineering (CBSE).

2. Component standards that facilitates the integration of components.

- These standards are embedded in a component model.
- They define (minimum): How component interfaces should be specified, How components communicate.
- If components confirm to standards, their operation is independent of the Programming Language.
- Components written in different languages can be integrated into the same system.

Essentials of
Component- based Software Engineering (CBSE).

3. Middleware that provides software support for component integration

- To make independent, distributed components work together, you need middleware support that handles component communications.

Essentials of
Component- based Software Engineering (CBSE).

4. A development process that is geared to component-based software engineering.

- Allows requirements to evolve, depending on the functionality of available components.

CBSE and Design Principles

- Apart from the benefits of reuse, CBSE is based on sound software engineering design principles:
  - Components are independent so do not interfere with each other;
  - Component implementations are hidden;
  - Communication is through well-defined interfaces;
  - Component platforms are shared and reduce development costs.

CBSE problems

- Component trustworthiness - how can a component with no available source code be trusted?
- Component certification - who will certify the quality of components?
- Emergent property prediction - how can the emergent properties of component compositions be predicted? Especially when they are integrated with other components.

CBSE problems

- Multiple standards (eg. COM, EJB) have hindered the uptake of CBSE.
  - Impossible for components developed using different approaches to work together.
  - Components that are developed for different platforms, such as .NET or J2EE, cannot interoperate.
  - The standards and protocols proposed were complex and difficult to understand.
- In response to these problems, the notion of a **component as a service** was developed.

Component as a Service Provider

- The component is an independent, executable entity. It does not have to be compiled before it is used with other components.
- The services offered by a component are made available through an interface and all component interactions take place through that interface.

Component characteristics 1

Standardised	Component standardisation means that a component that is used in a CBSE process has to conform to some standardised component model. This model may define component interfaces, component meta-data, documentation, composition and deployment.
Independent	A component should be independent – it should be possible to compose and deploy it without having to use other specific components. In situations where the component needs externally provided services, these should be explicitly set out in a 'requires' interface specification.
Composable	For a component to be composable, all external interactions must take place through publicly defined interfaces. In addition, it must provide external access to information about itself such as its methods and attributes.

Component characteristics 2

Deployable	To be deployable, a component has to be self-contained and must be able to operate as a stand-alone entity on some component platform that implements the component model. This usually means that the component is a binary component that does not have to be compiled before it is deployed.
Documented	Components have to be fully documented so that potential users of the component can decide whether or not they meet their needs. The syntax and, ideally, the semantics of all component interfaces have to be specified.

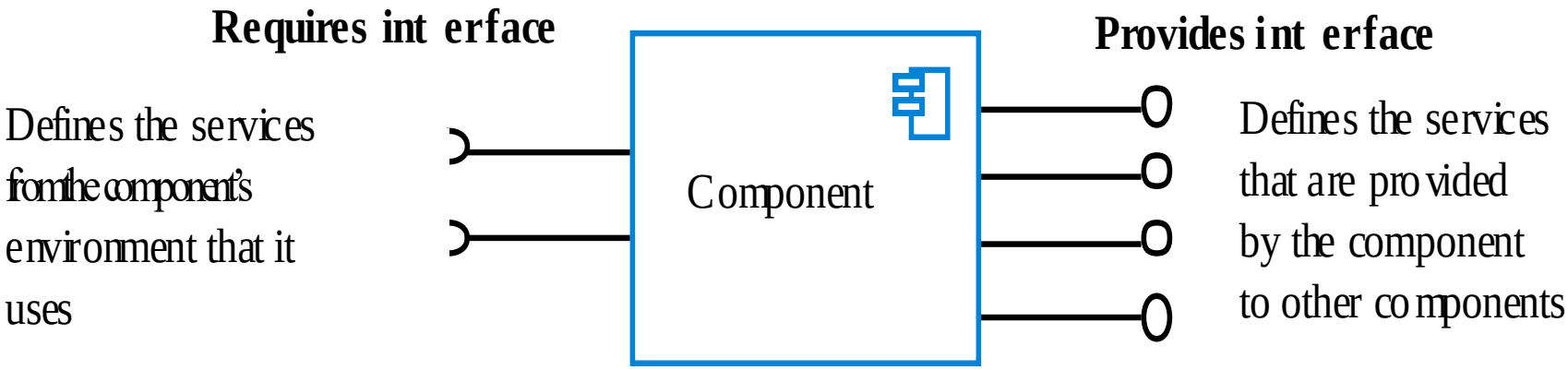
Component interfaces

Components have two related interfaces.

- Provides interface
  - Defines the services that are provided by the component to other components.

    In UML – *Circle* is used
- Requires interface
  - Defines the services that specifies what services must be made available for the component to execute as specified.
  - In UML – *Semi Circle* is used

Component interfaces

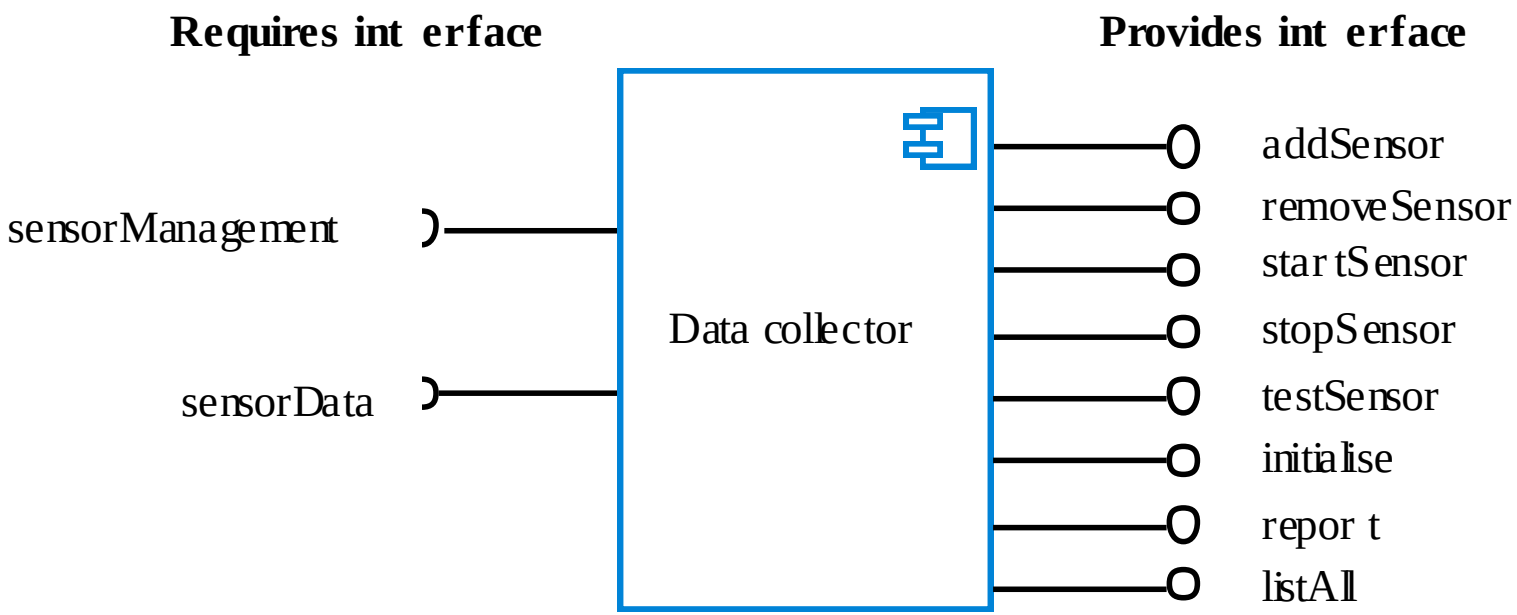


In a UML component diagram, the "**provides**" interface for a component is Indicated by a circle at the end of a line from the component icon

In the UML the symbol for a "**requires**" interface is a semicircle at the end of a line from the component icon.

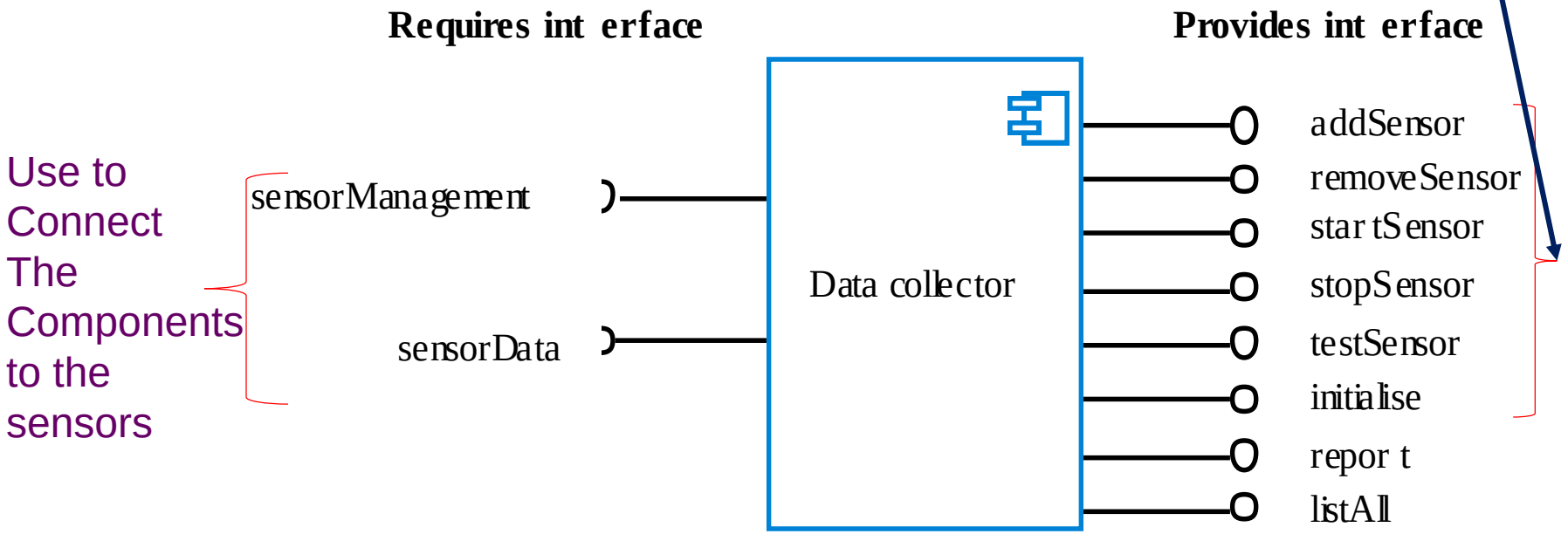
A data collector component

Designed to collect and collate information from an array of sensors over a period.

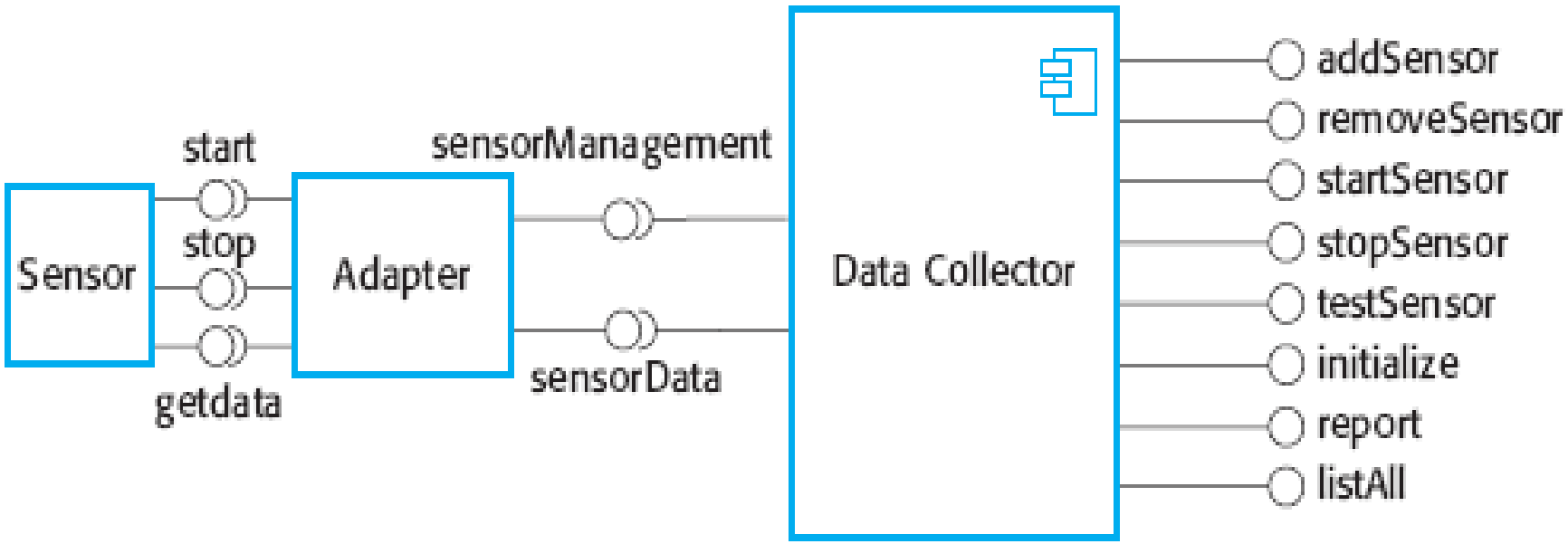


A data collector component

- The provides interface include methods to ***initialize, add, remove, start, stop, and test sensors.***
- The ***report*** method returns the sensor data that has been collected.
- ***listAll*** method provides information about the attached sensors.

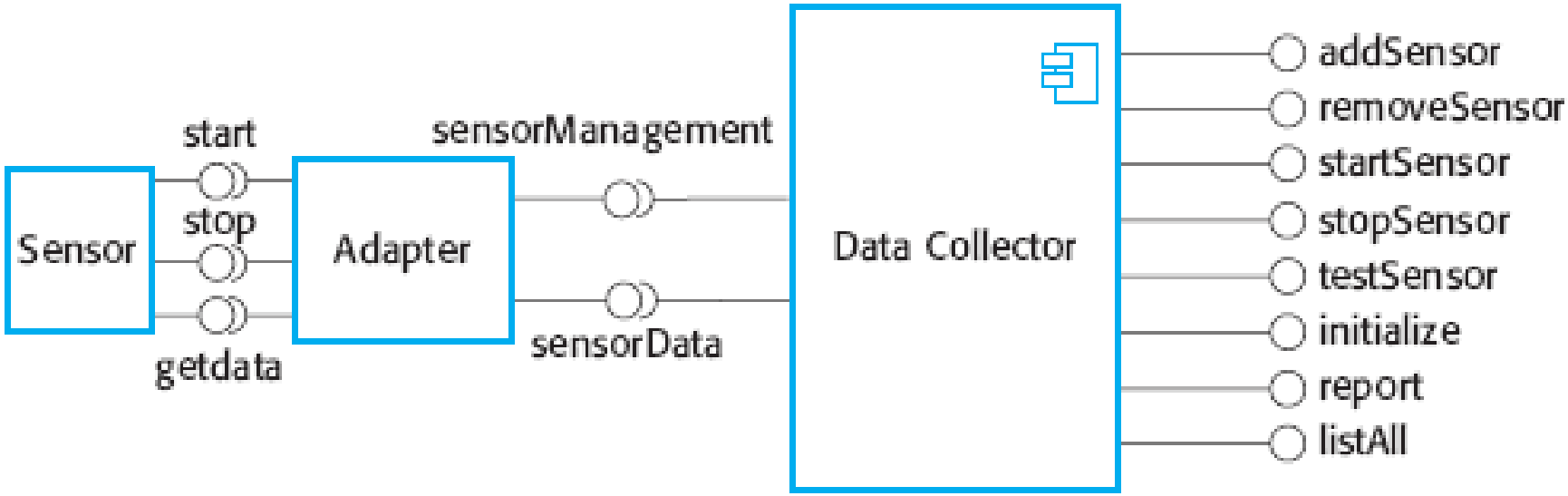


Adaptor linking a Data Collector and a Sensor



- Adaptor component can be used when a component wishes to make use of another but there is an *incompatibility between the provides and required interfaces* of the components.

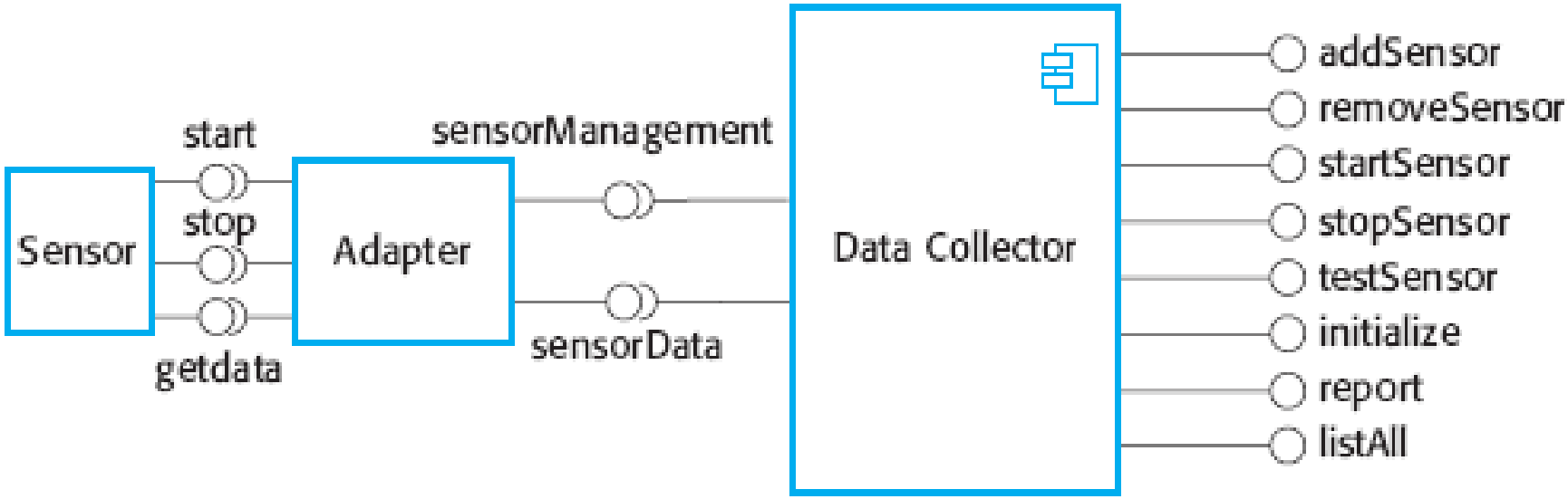
Adaptor linking a Data Collector and a Sensor



Eg. To issue a collect command sensorData("Collect")

- Adaptor passes the input string, identifies the command (eg. "Collect") and calls the sensor.getData to collect the sensor value. It then returns the result to the Data Collector.

Adaptor linking a Data Collector and a Sensor

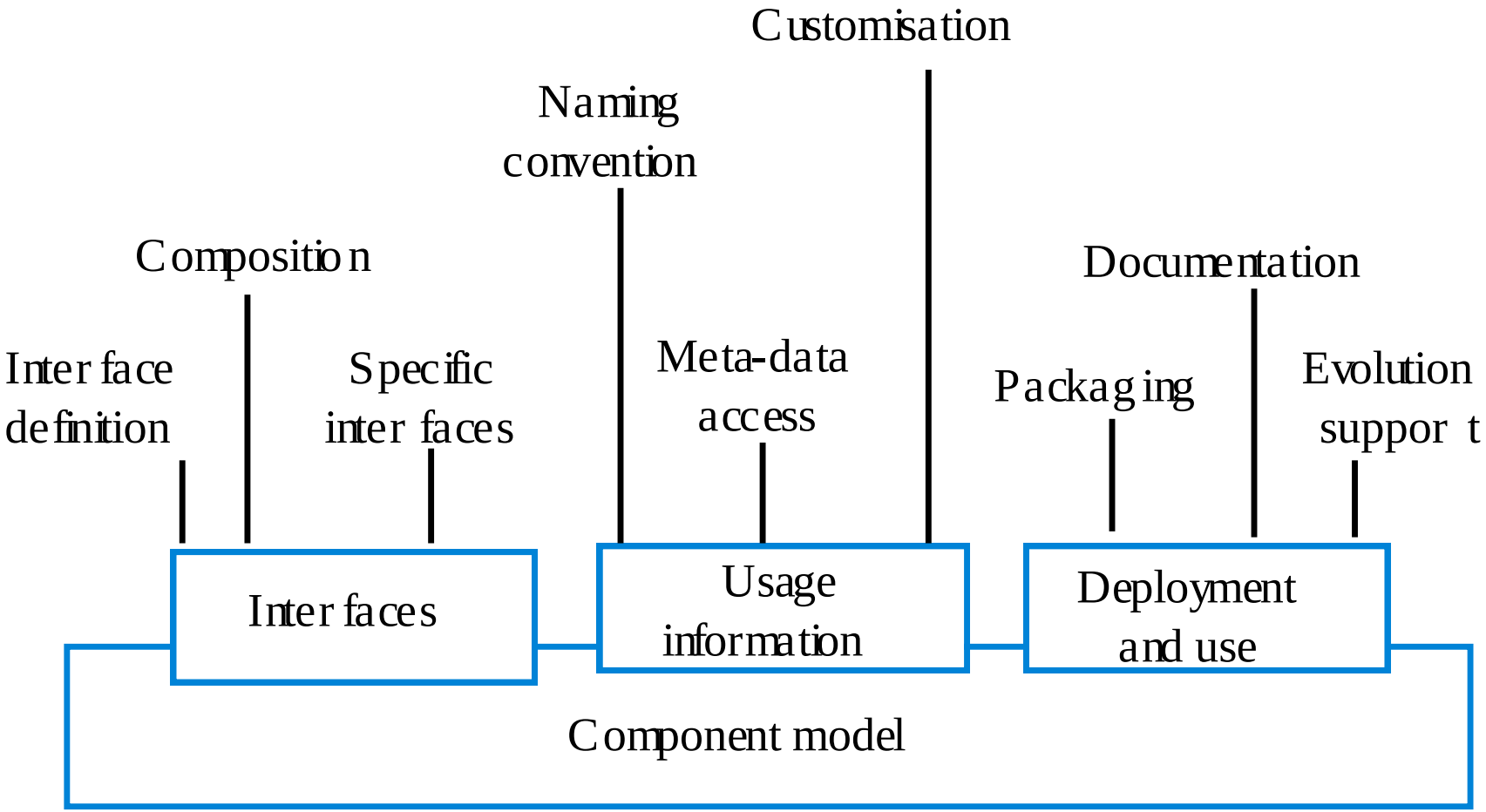


Similarly, Data Collector can interact with different types of sensors. A separate **Adaptor** which converts the sensor command from Data Collector to actual sensor interface is implemented for each type of sensor.

Component models

- A component model is a definition of standards for component implementation, documentation and deployment.
- Examples of component models
  - WebServices model
  - EJB model (Enterprise Java Beans)
  - COM+ model (.NET model)
  - CORBA Component Model
- The component model specifies how interfaces should be defined and the elements that should be included in an interface definition.

Basic elements of a component model



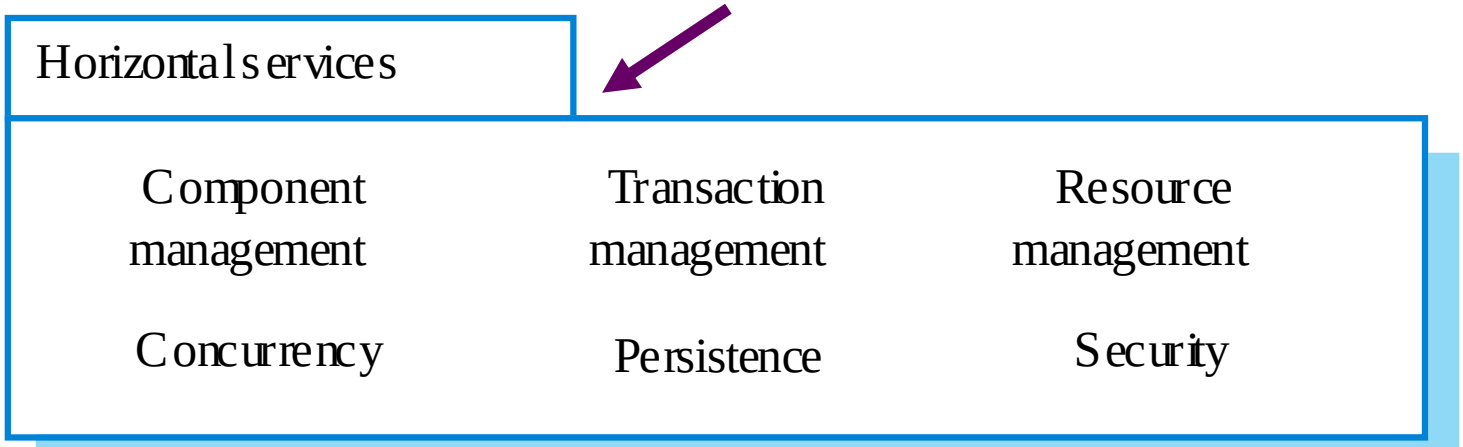
Middleware support – Components implemented as program units

- Component models are the basis for middleware that provides support for executing components.
- Component model implementations provide:
  - Platform services that allow components written according to the model to communicate;
  - Horizontal services that are application-independent services used by different components.
- To use services provided by a model, components are deployed in a ‘container’.
- This is a set of interfaces used to access the service implementations.

Component model services

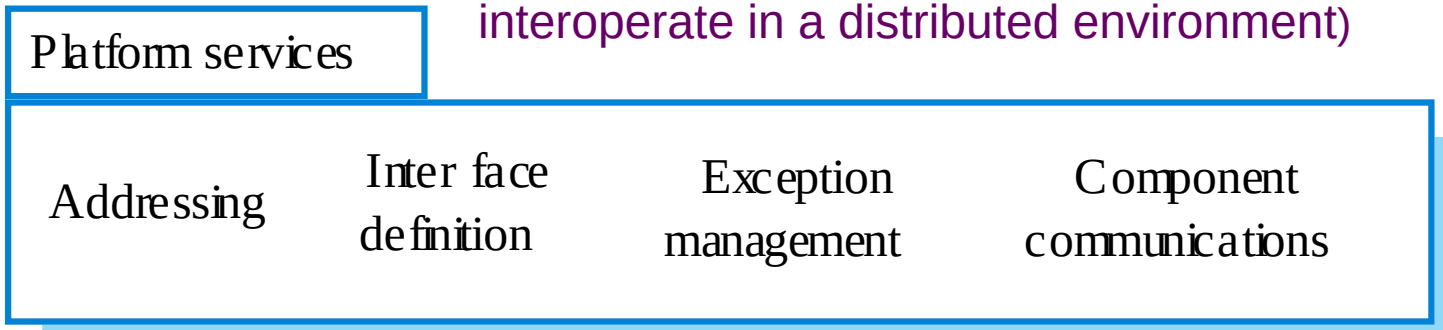
Support Services

(Application-independent services used by different components)



Fundamental Services

(enable components to communicate and interoperate in a distributed environment)



CBSE processes

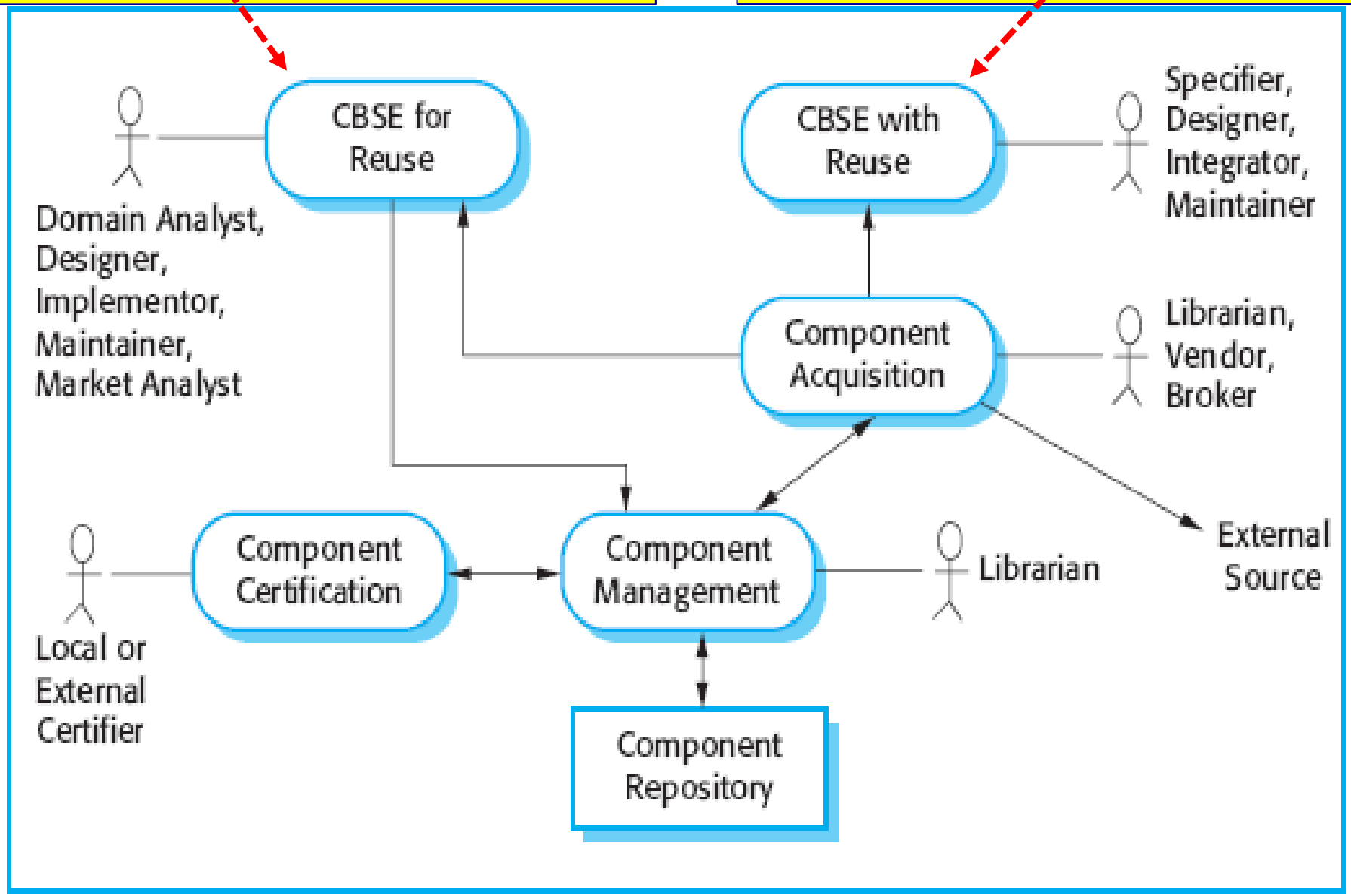
There are two types of CBSE processes:

1. **Development for reuse** This process is concerned with developing components or services that will be reused in other applications. It usually involves generalizing existing components. Knows the components that you will be working with.

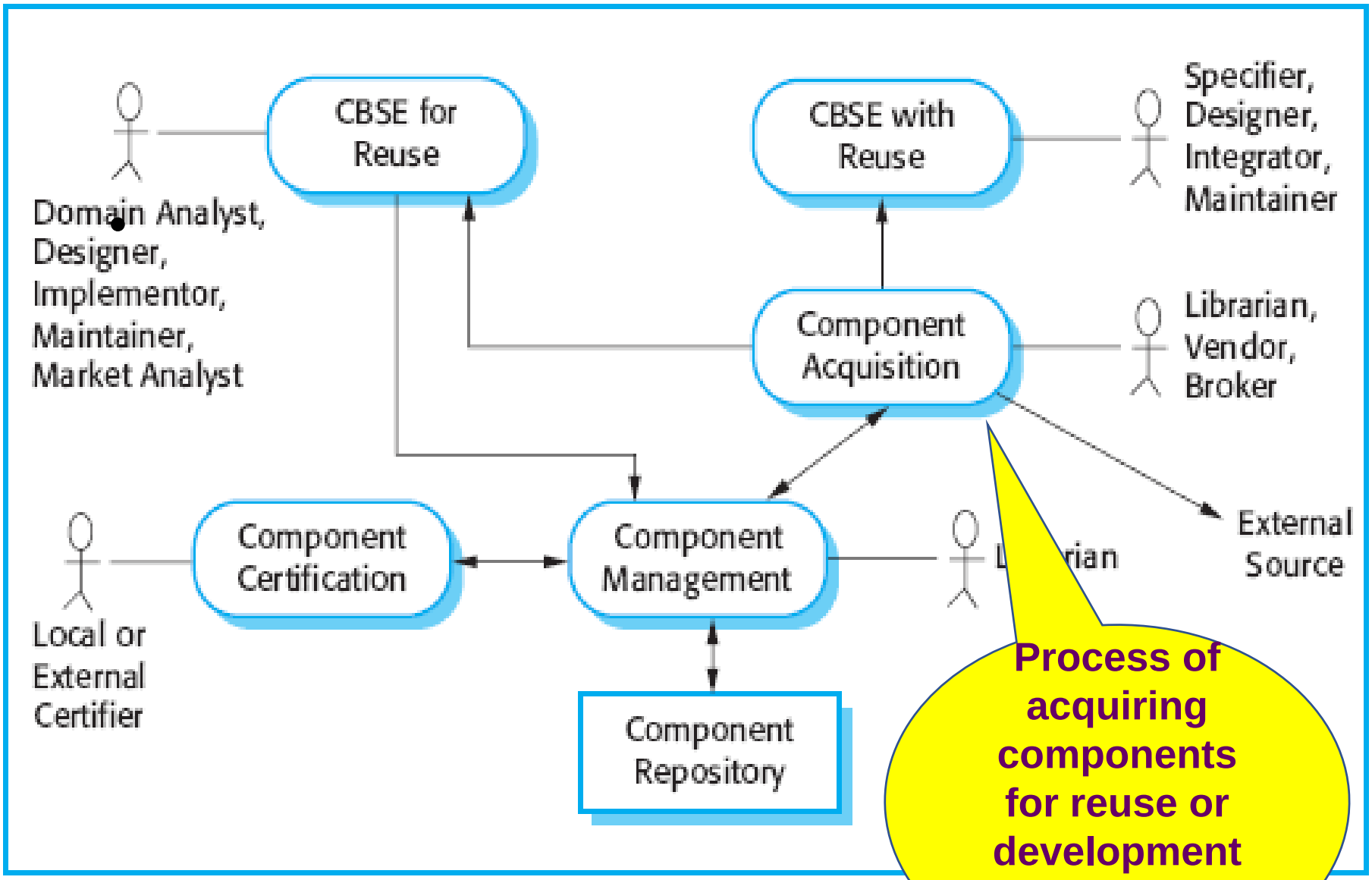
2. **Development with reuse** This is the process of developing new applications using existing components and services. You do not know what components are available.

Have access to source code to generalize them

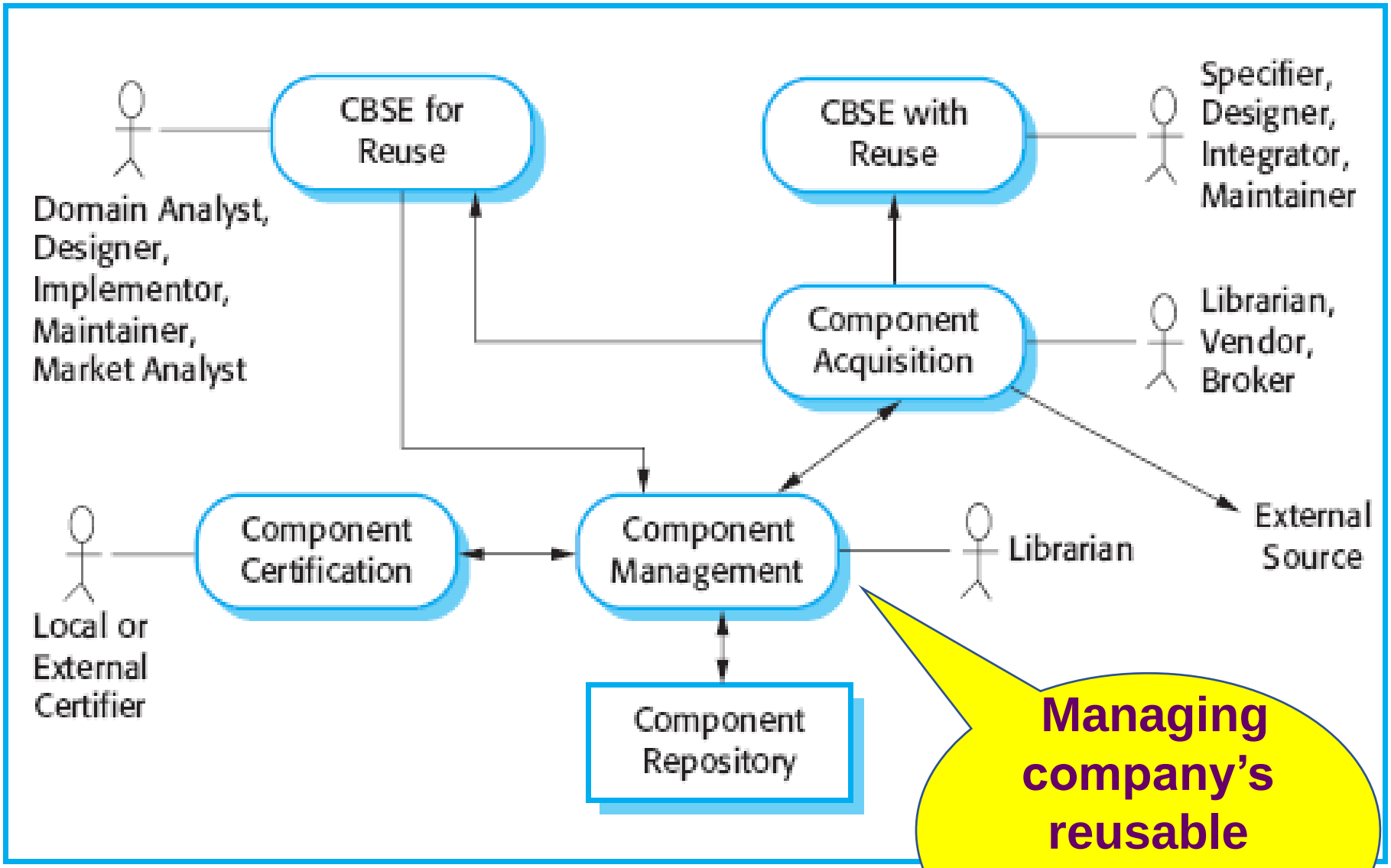
You may not have access to source code



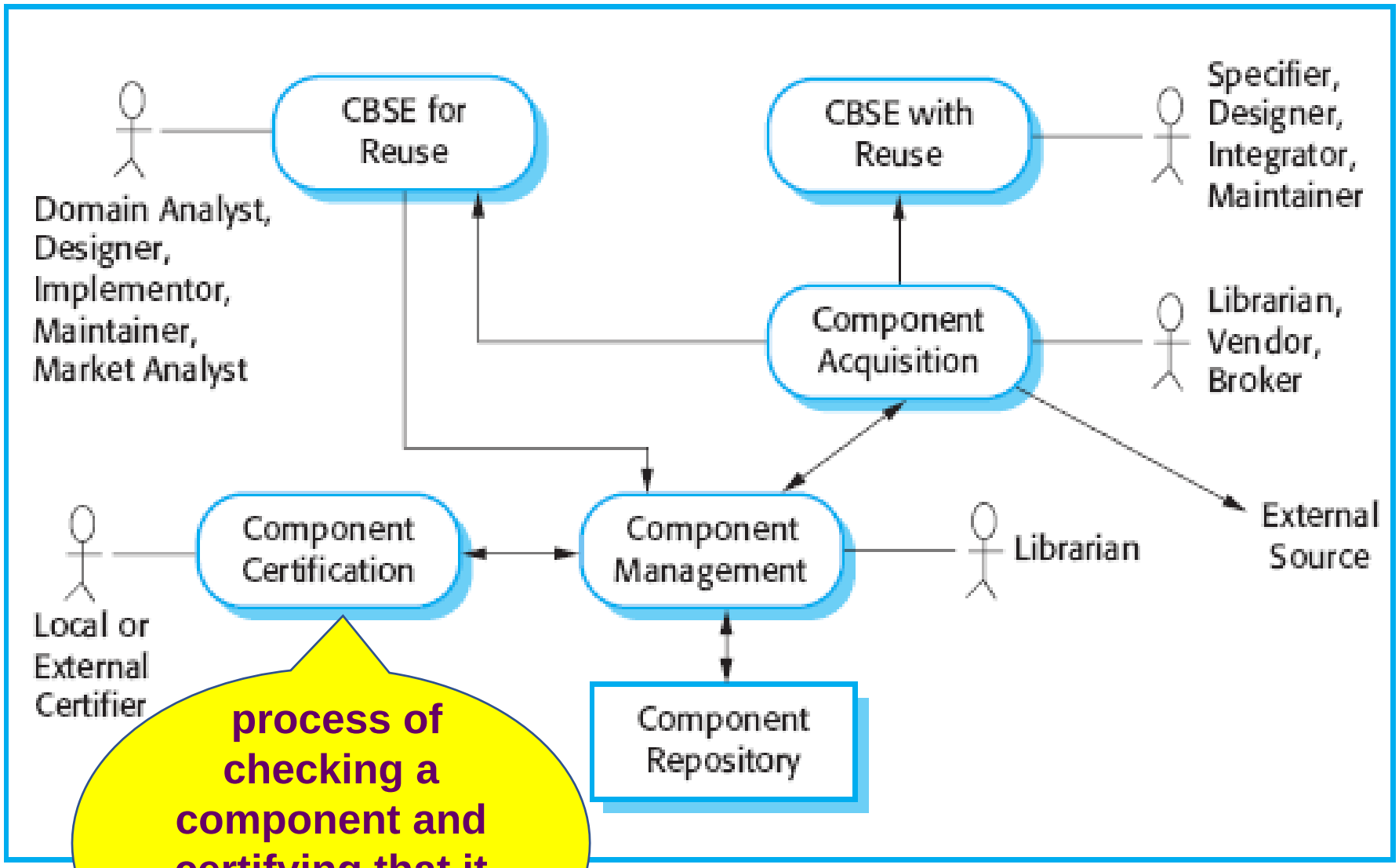
CBSE Processes



CBSE Processes

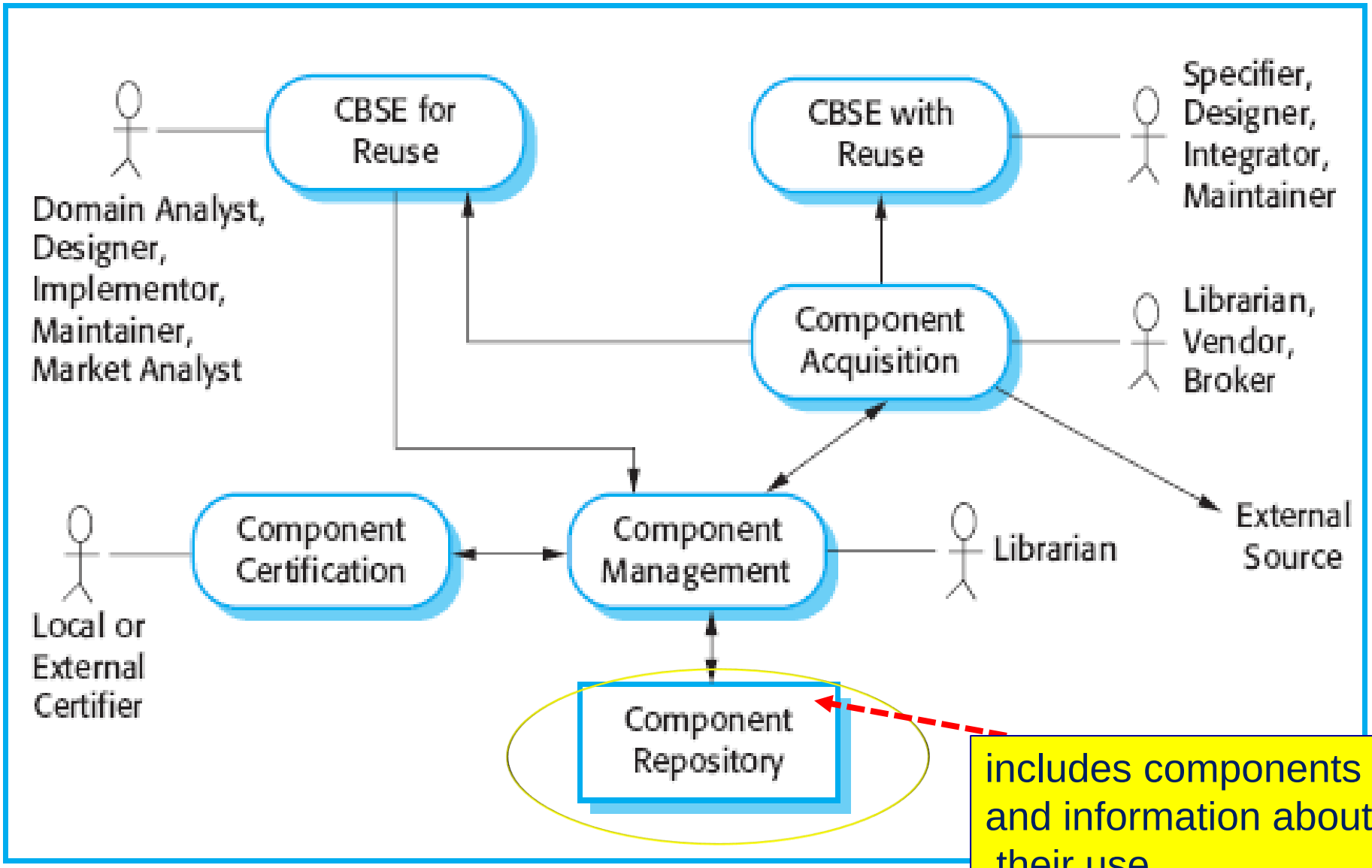


CBSE Processes



process of checking a component and certifying that it meets its specification

CBSE Processes



Component development for reuse

- Components developed for a specific application usually have to be generalized to make them reusable. –associated cost
- A component is most likely to be reusable if it associated with a stable domain abstraction (business object).
- For example, in a hospital stable domain abstractions are associated with the fundamental purpose - nurses, patients, treatments, etc.

Component development for reuse

- Components for reuse may be specially constructed by generalising existing components.
- Component reusability should
  - reflect stable domain abstractions;
  - hide state representation;
  - be as independent as possible;
  - publish exceptions through the component interface.

To make a component cost effective

- Cost savings from future reuse vs cost of making
- Cost of Making:
  - Documentation cost,
  - Component validation,
  - Making it more generic

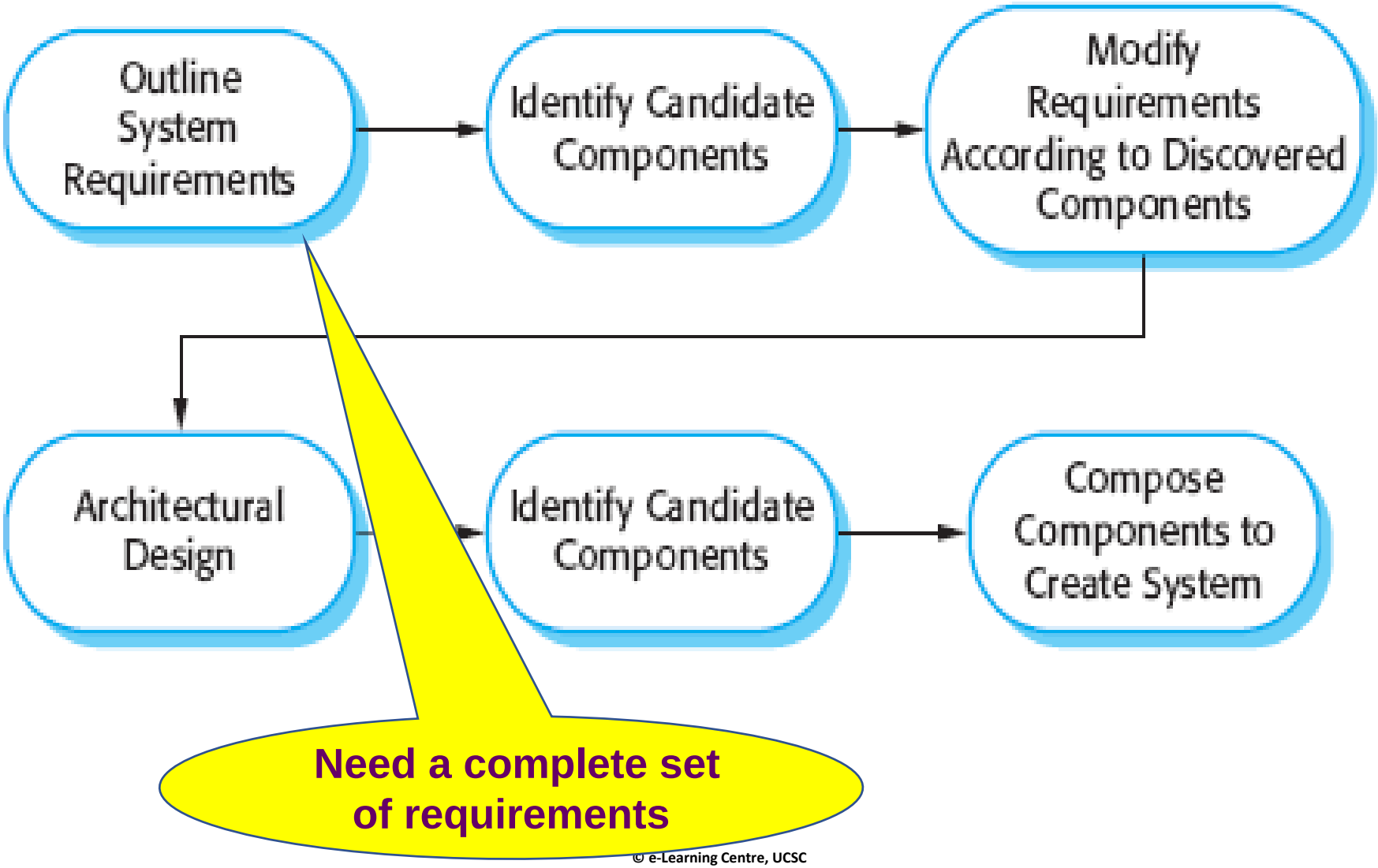
Changes for reusability

- Remove application-specific methods.
- Change names to make them general.
- Add methods to broaden coverage.
- Make exception handling consistent.
- Add a configuration interface for component adaptation.
- Integrate required components to reduce dependencies.

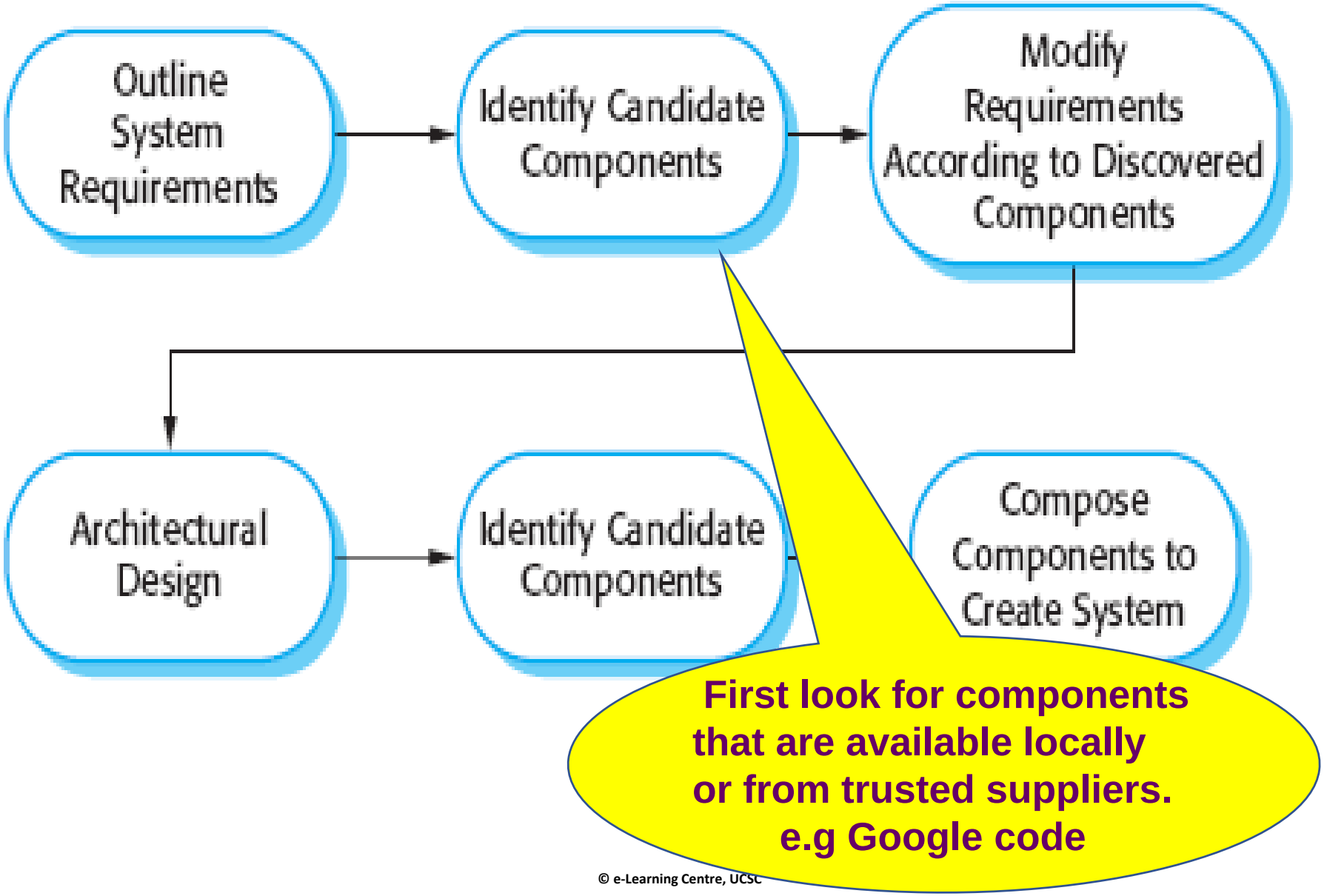
Legacy system components

- Existing legacy systems that fulfil a useful business function can be re-packaged as components for reuse.
- This involves writing a wrapper component that implements provides and requires interfaces then accesses the legacy system.
- Although costly, this can be much less expensive than rewriting the legacy system.

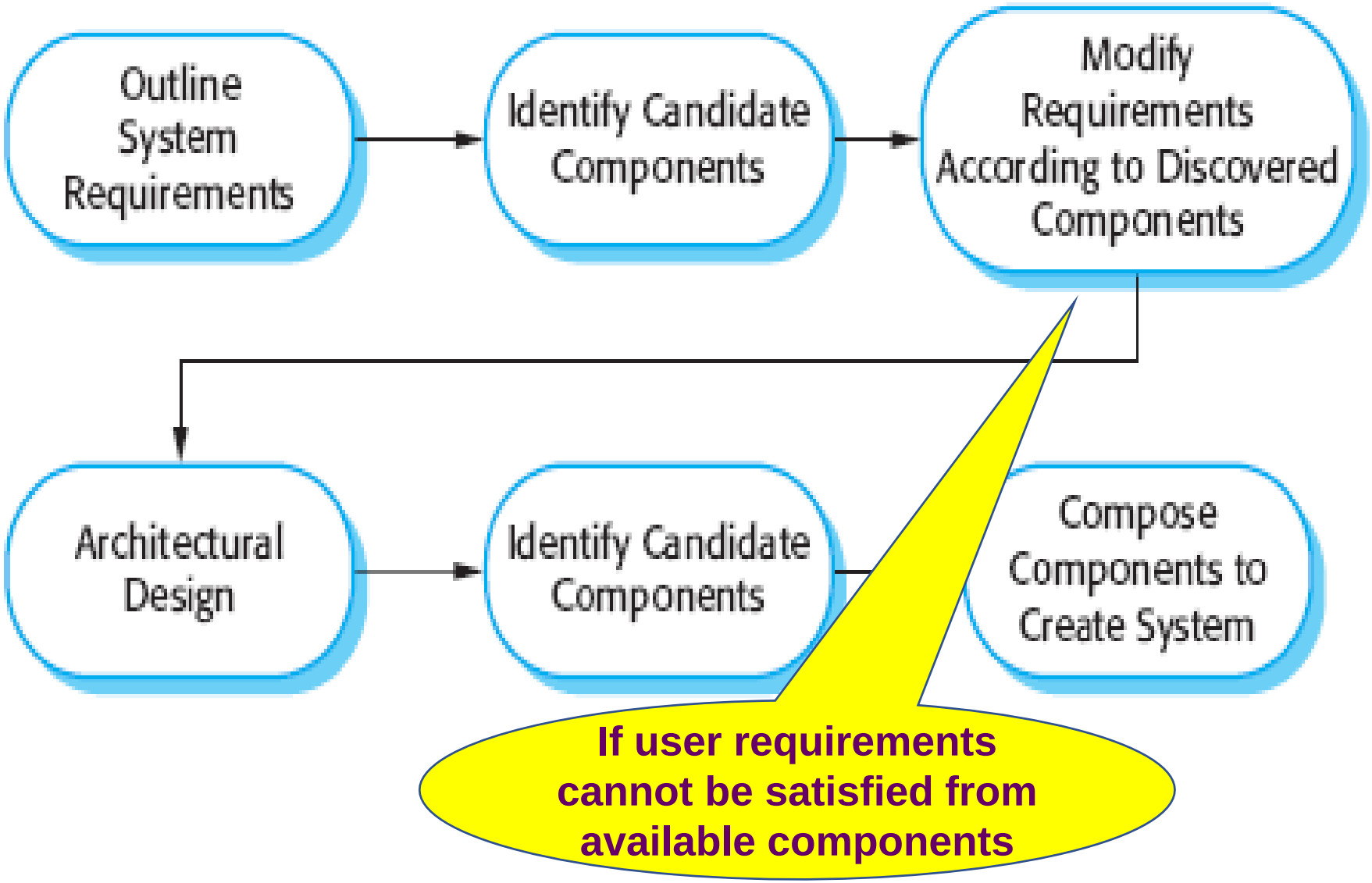
Component Development with reuse



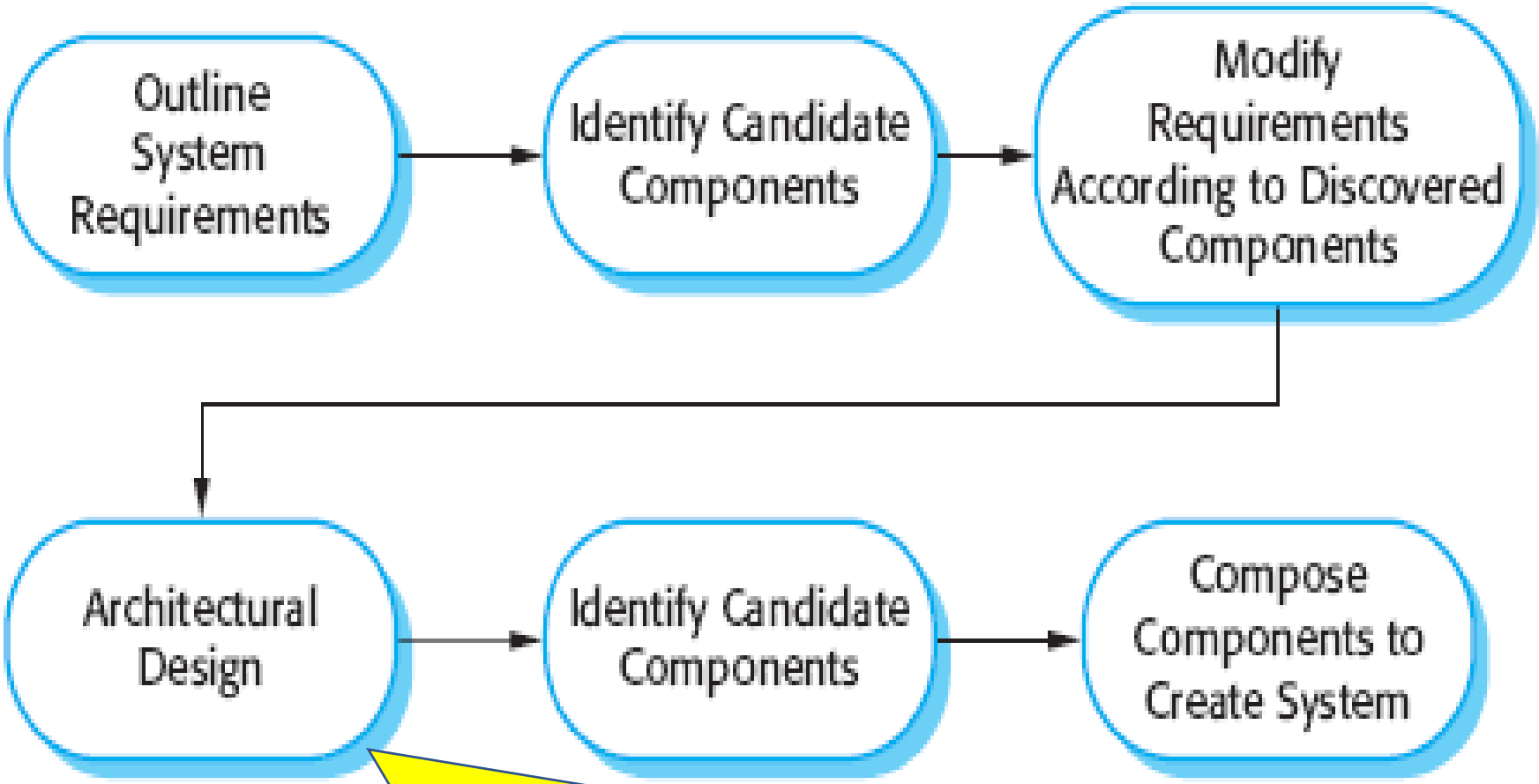
Component Development with reuse



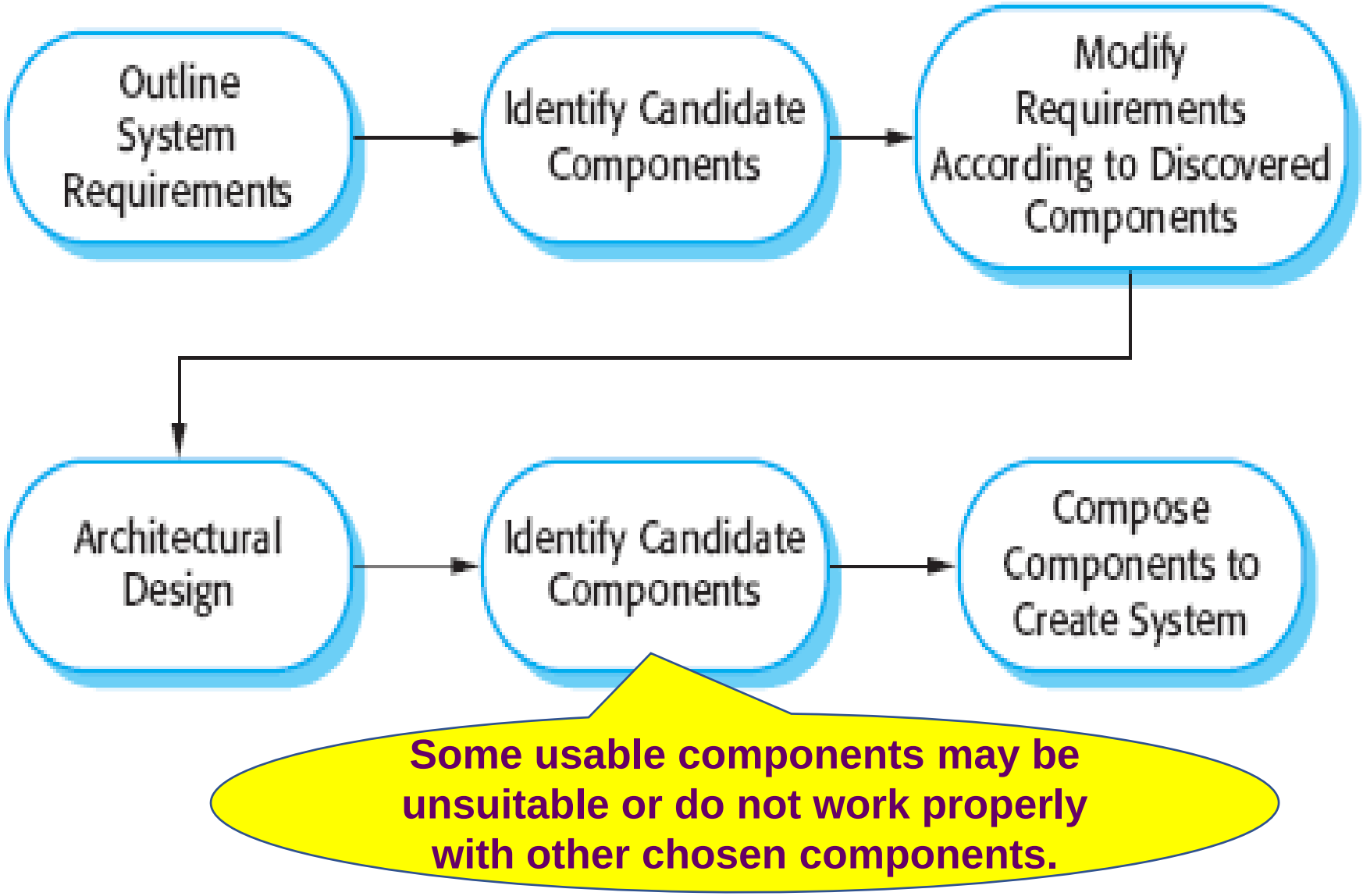
Component Development with reuse



Component Development with reuse



Component Development with reuse

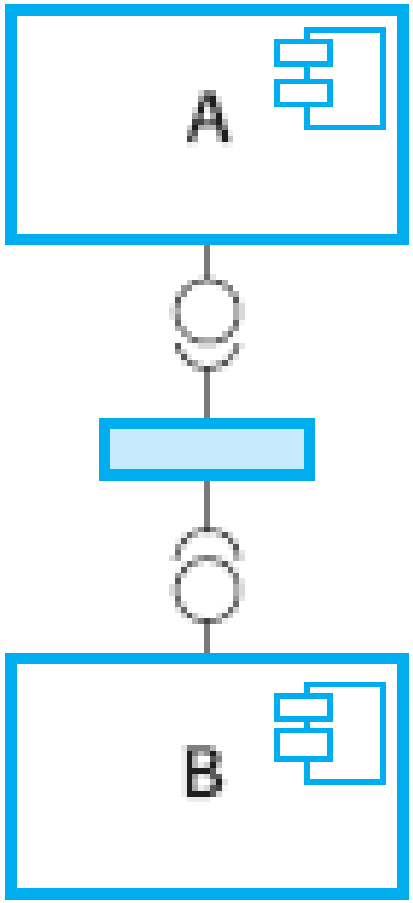


Component Composition

- Process of integrating components with each other (with or without glue code) to create a system or another component.
- Several different ways exist
  - **Sequential Composition**
  - **Hierarchical Composition**
  - **Additive Composition**

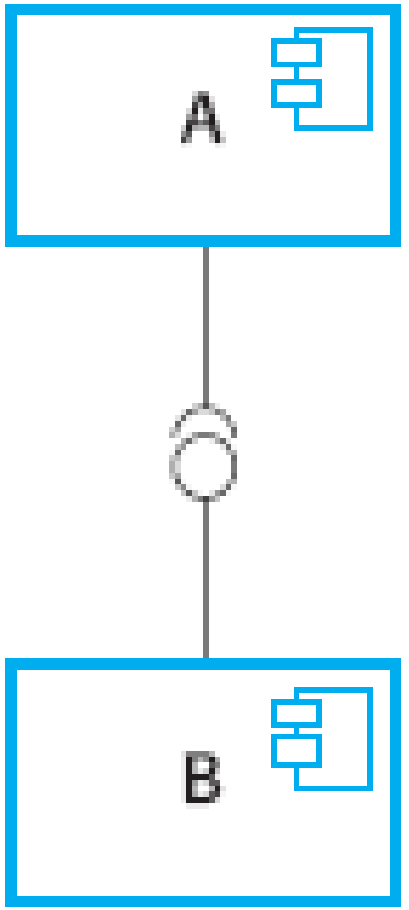
Component Composition (Sequential)

- The services offered by Component A are called and the results returned by A are then used in the call to the services offered by component B.
- Extra glue code required to call the services in the right order and ensure compatibility.



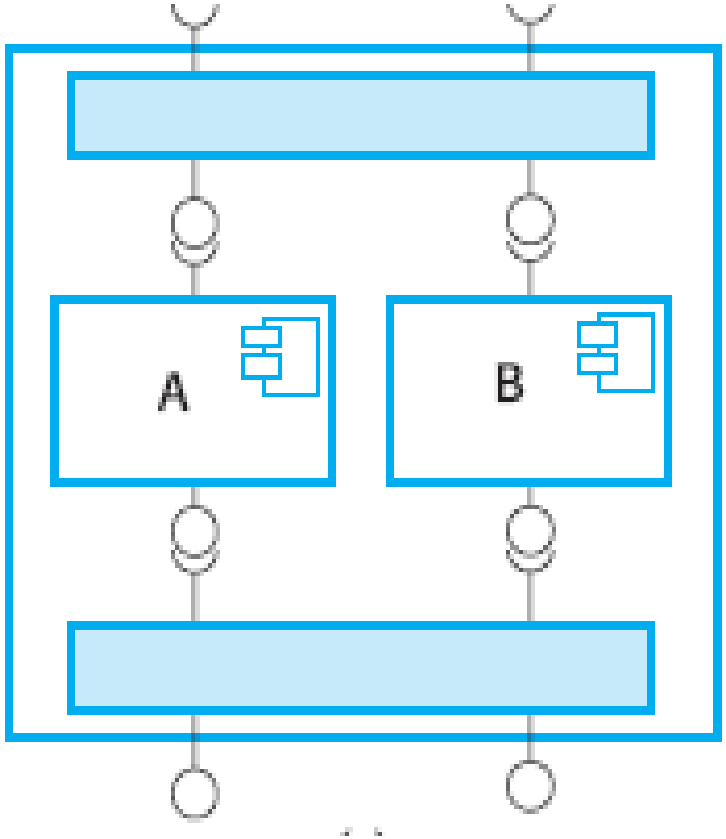
Component Composition (Hierarchical)

- This type of composition occurs when one component calls directly on the services provided by another component.
- The ‘provides’ interface of the called component must be compatible with the ‘requires’ interface of the calling component.



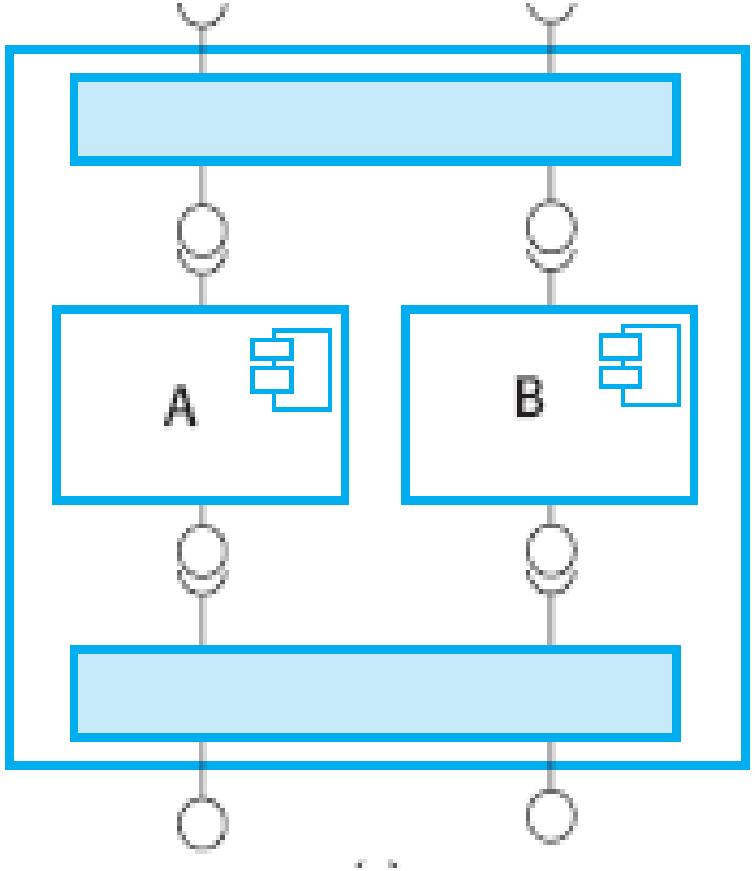
Component Composition (Additive)

- This occurs when two or more components are put together (added) and,
- Create a new component, which combines their functionality.



Component Composition (Additive)

- The components are called separately through the external interface of the composed component.
- A and B are not dependent and do not call each other.



Component Composition

- When composing reusable components that have not been written for your application:
  - You may need to write adaptors or ‘glue code’ to reconcile the different component interfaces.
- When choosing compositions, you have to consider
  - required functionality of the system
  - non-functional requirements and
  - how easy is to replace one component when the system is changed.

Summary

- CBSE is the process of defining, Implementing and Integrating, loosely coupled, independent components into systems.
- Components are higher level abstractions than objects and they are defined by their interfaces.
- Independent Components that are completely specified by their interfaces so that there should be a clear separation between the component interface and its implementation.
- There should be component standards that facilitates the integration of components.
- To make independent, distributed components work together, you need middleware support that handles component communications.
- A development process that is geared to component-based software engineering allows requirements to evolve, depending on the functionality of available components.
- Characteristics of components are standardized, independent, composable, deployable and documented.

Summary cont..

- *Provides* component interface defines the services that are provided by the component to other components. In UML *Circle* symbol is used for the *Provides* interface.
- *Requires* interface defines the services that specifies what services must be made available for the component to execute as specified. In UML Semi Circle is used for the Requires interface
- *Component composition is the process of "wiring" components together to create a system.*
- *Types of composition include sequential composition, hierarchical composition, and additive composition.*
- *When composing reusable components that have not been written for your application, you may need to write adaptors or "glue code" to reconcile the different component interfaces.*

*****