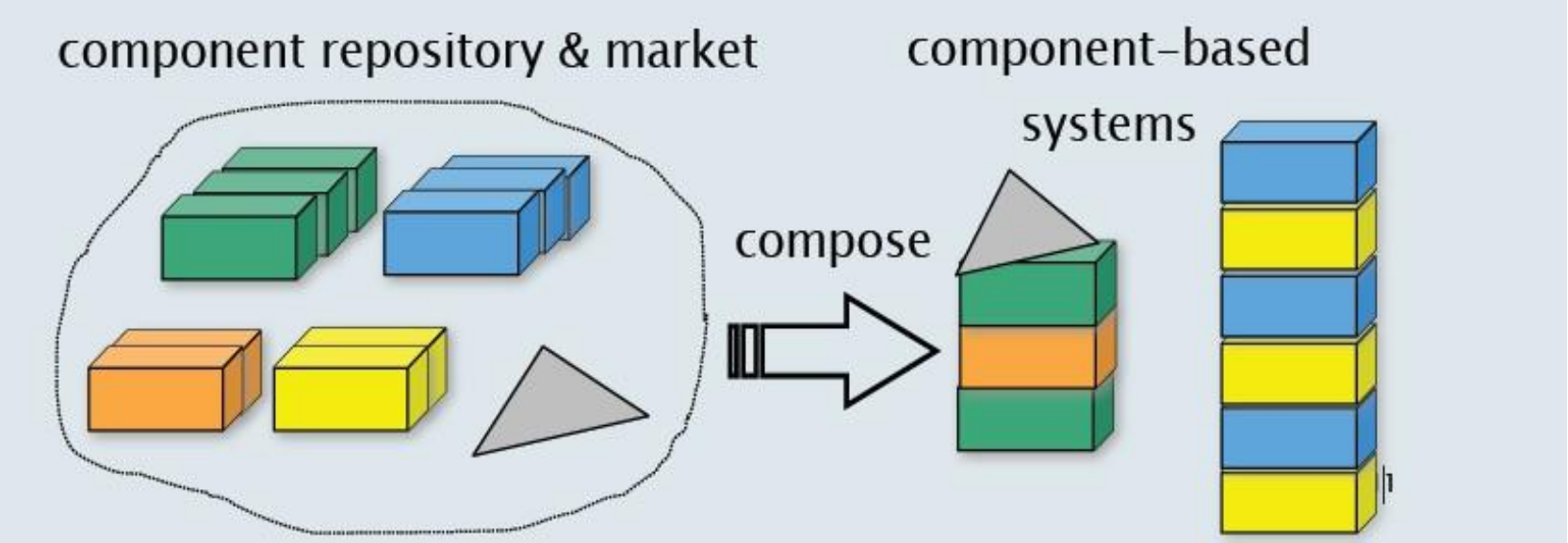
Chapter 5

Component Based Software Engineering (CBSE)



Contents

- Introduction
- Components
  - Provides and Requires Interface
  - Component characteristics
- Component models
  - Basic elements of a component model
  - Middleware support
- CBSE processes
  - CBSE for reuse
  - CBSE with reuse
- Component composition
  - Types of component composition
  - Interface Incompatibility
  - Interface semantics, Composition Trade-off

Learning Outcomes

- The objective of this chapter is to describe an approach to software reuse
- based on the composition of reusable, standardized components. When you have read this activity you will:
  - know that component-based software engineering is concerned with developing standardized components based on a component model, and composing these into application systems;
  - understand what is meant by a component and a component model;
  - know the principal activities in the CBSE process for reuse and the CBSE process with reuse;
  - understand some of the difficulties and problems that arise during the process of component composition.

Introduction…

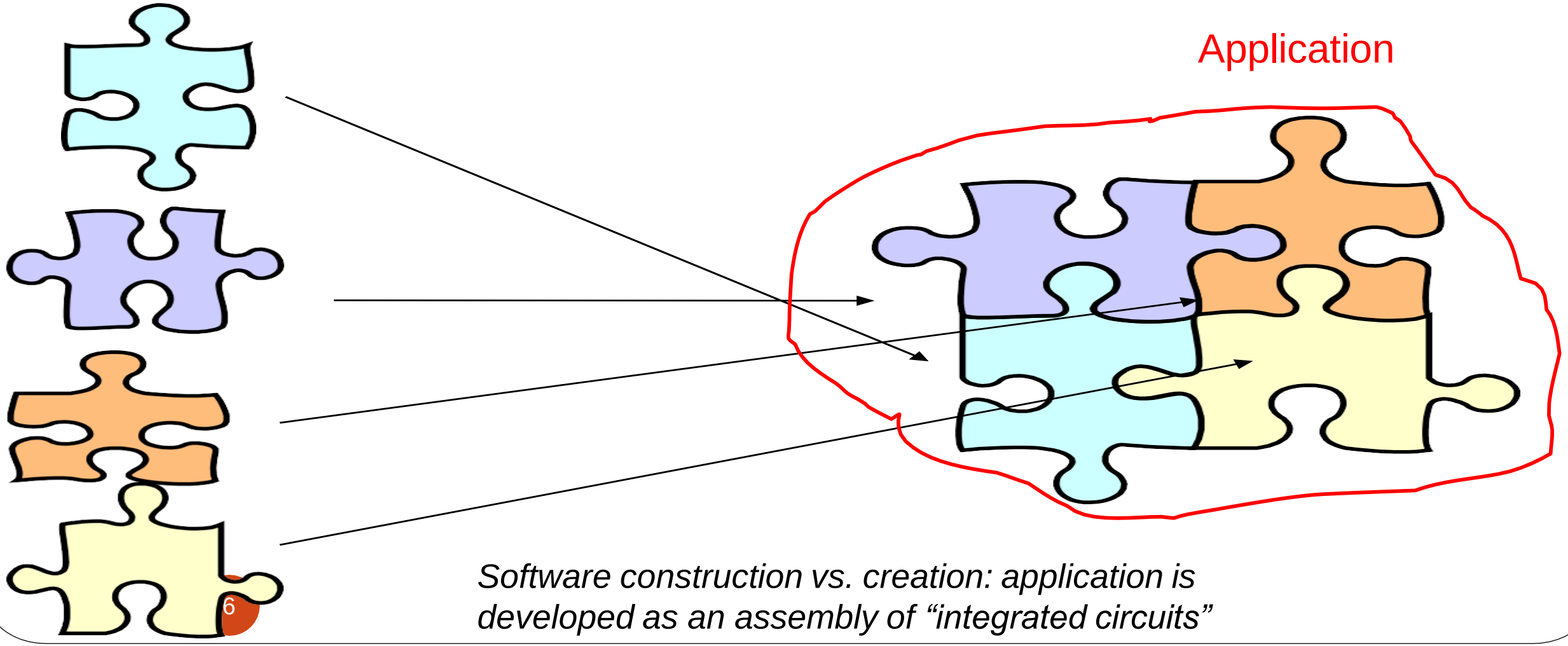
- CBSE is a process for developing computer systems using reusable software components.
- It is concerned with assembling of pre-existing software components into larger pieces of software
- It emerged from the failure of object-oriented development to support effective reuse.
  - Single object classes are too detailed and specific.
- Components are more abstract than object classes and can be considered to be stand-alone service providers.
  - They can exist as stand-alone entities.

Introduction…

- The essentials of component-based software engineering are:
- **Independent components** that are completely specified by their interfaces.
  - There should be a clear separation between the component interface and its implementation.
- **Component standards** to facilitate component integration.
  - They define, at the very minimum, how component interfaces should be specified and how components communicate.
- **Middleware** that provides support for component inter- operability.
  - To make independent, distributed components work together, you need middleware support that handles component communications.
- **A development process** that is geared to reuse

Introduction…

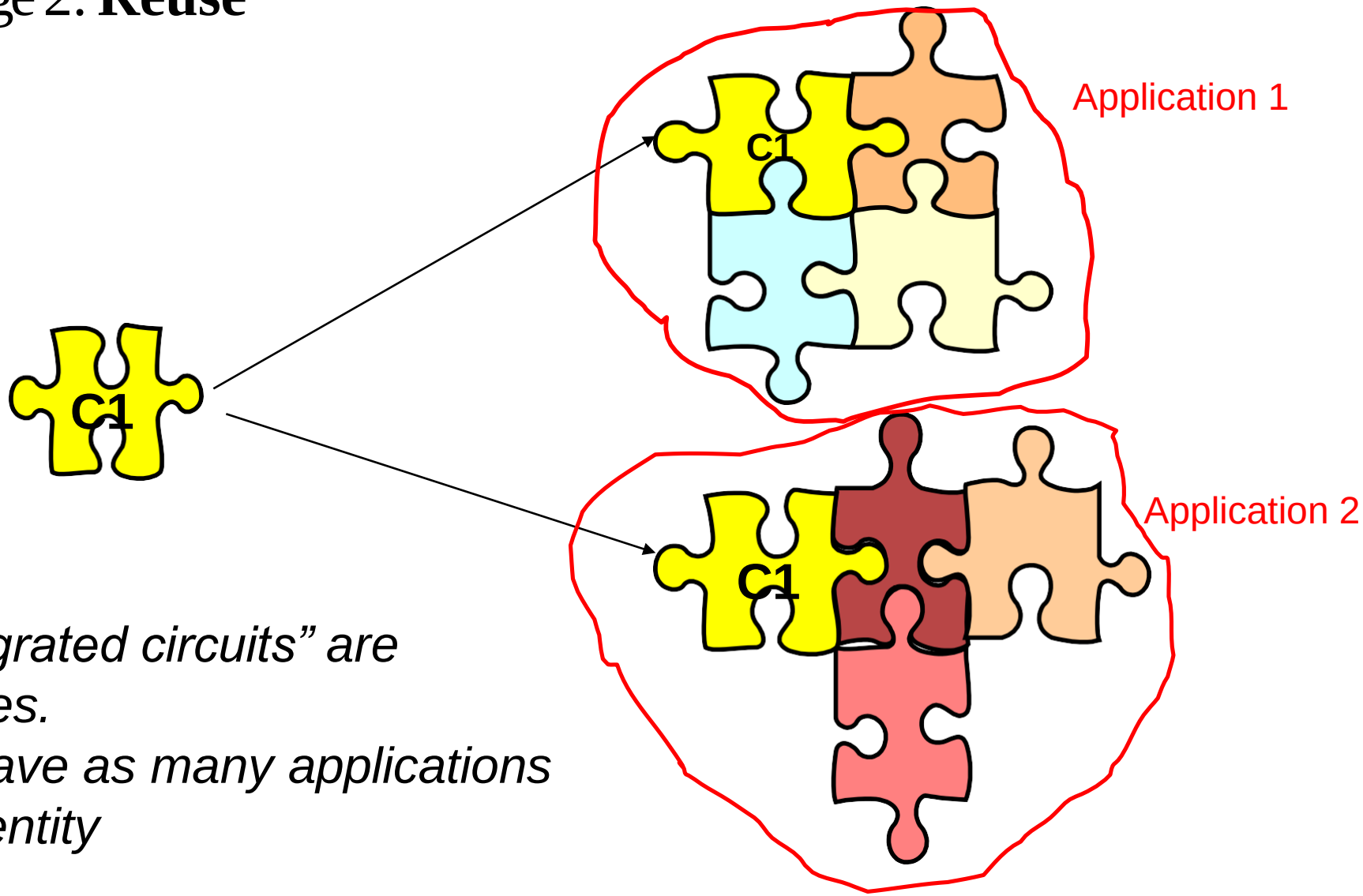
- There are a number of advantages gained with CBSE
- Advantage 1: **Software construction**: Building a system by composing "entities"



Software construction vs. creation: application is developed as an assembly of "integrated circuits"

Introduction…

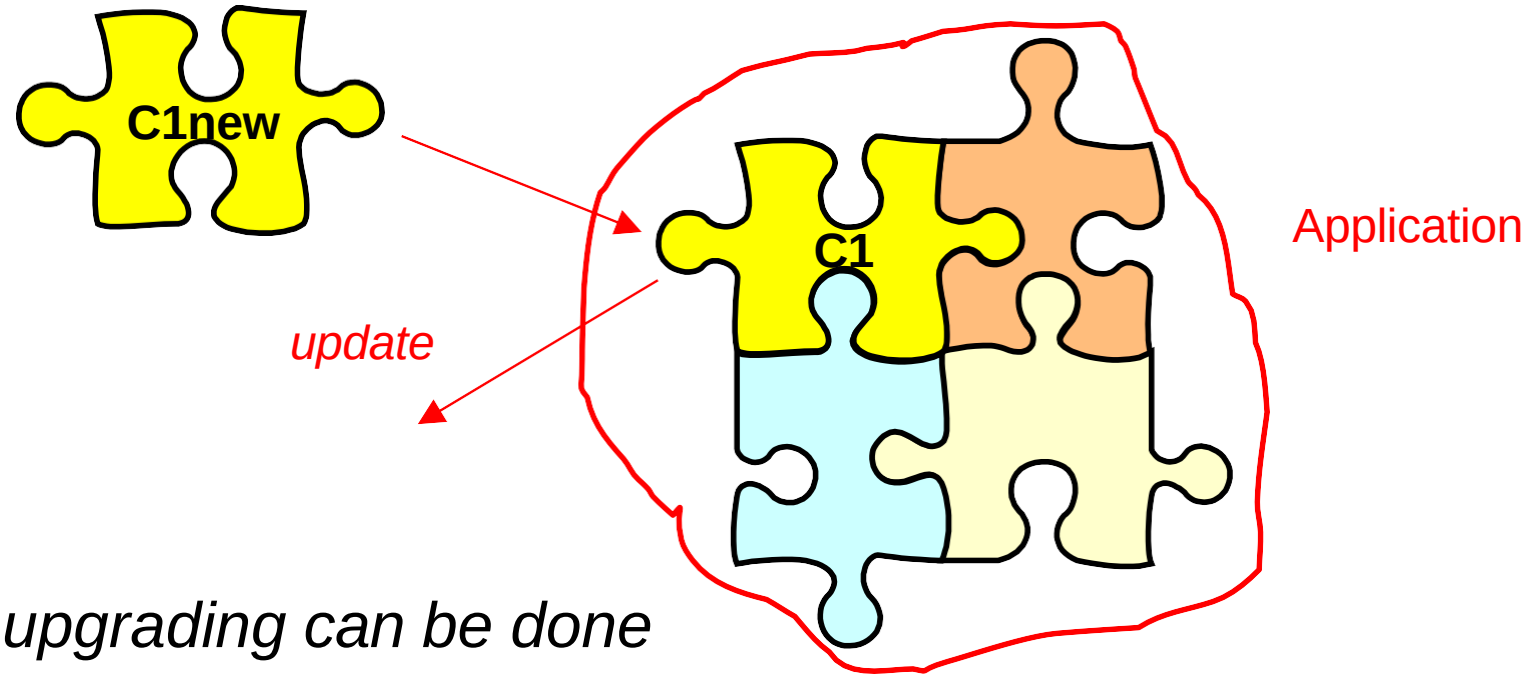
- Advantage 2: **Reuse**



Software "integrated circuits" are reusable entities.
It pays off to have as many applications that reuse an entity

Introduction…

- Advantage 3: **Maintenance & Evolution**: Maintaining a system by adding/removing/replacing "entities"



Maintenance and upgrading can be done by replacing parts, maybe even at runtime

Introduction…

- There are a number of problems with CBSE that have limited its impact
  - **Component trustworthiness** - how can a component (are black-box program units) 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?
  - **Requirements trade-offs** - how do we do trade-off analysis between the features of one component and another?
    - more structured, systematic trade-off analysis method to help designers select and configure components is important.

Components

- There is general agreement in the CBSE community that a component is an independent software unit that can be composed with other components to create a software system.

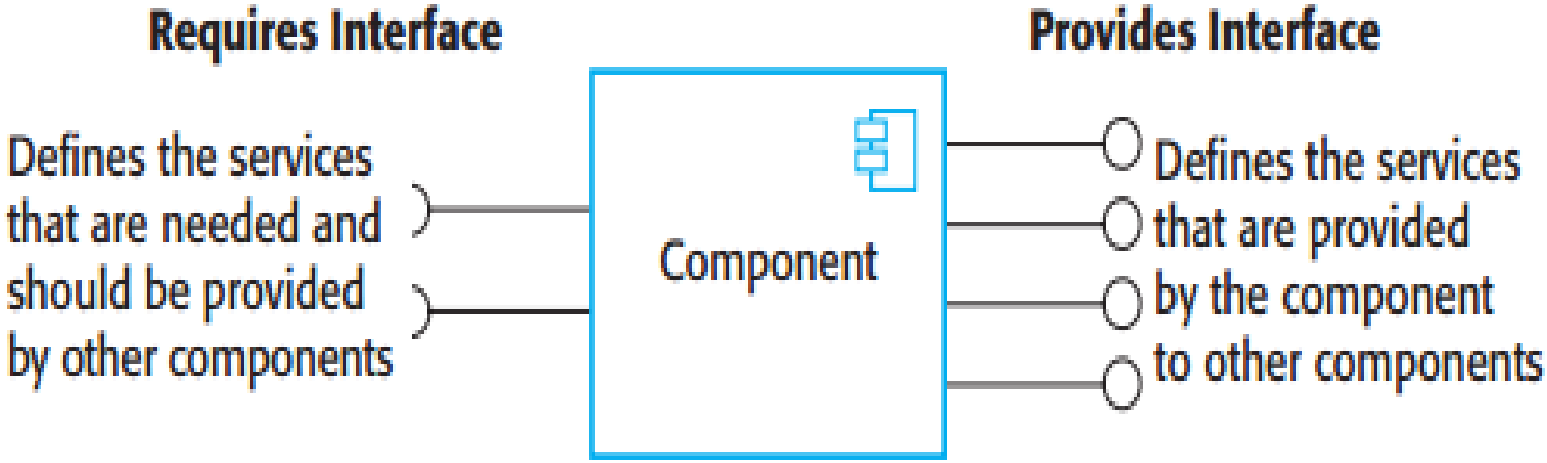
- *A software component*
  - *is a software element that conforms to a component model and can be independently deployed and composed without modification according to a composition standard.*
  - *is an independently deployable implementation of some functionality, to be reused as it is in a broad spectrum of applications.*
  - *a software package, a web service, a web resource, or a module that encapsulates a set of related functions (or data).*

Components...

- 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.
- The component interface is expressed in terms of parameterized operations and its internal state is never exposed.
- **Provides interface**
  - Defines the services that are provided by the component to other components.
  - This interface, essentially, is the component API. It defines the methods that can be called by a user of the component.

Components...

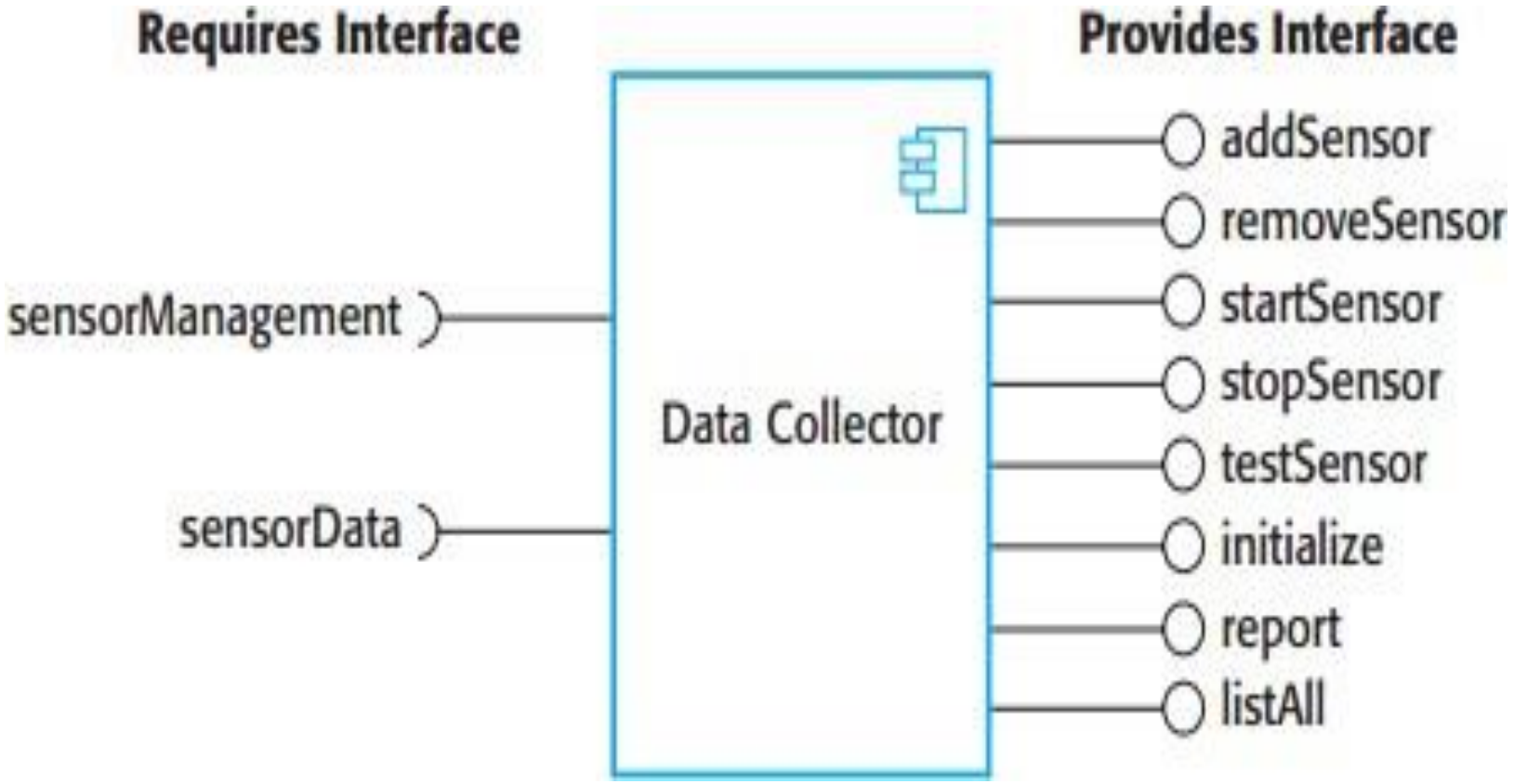
- **Requires interface**
  - Defines the services that specifies what services must be made available for the component to execute as specified.
  - This does not compromise the independence or deployability of a component because the 'requires' interface does not define how these services should be provided.



UML notation of a component. Ball and sockets can fit together.

Components...

A model of a data collector component



Components...

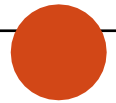
- It takes significant effort and awareness to write a software component that is effectively reusable.
- The component needs to be:
  - fully documented
  - thoroughly tested
    - robust - with comprehensive input-validity checking
    - able to pass back appropriate error messages or return codes
  - designed with an awareness that it *will* be put to unforeseen uses/not predicted uses

Component characteristics

Component characteristic	Description
Standardized	Component standardization means that a component used in a CBSE process has to conform to a standard component model. This model may define component interfaces, component metadata, 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…

Component characteristic	Description
Deployable	To be deployable, a component has to be self-contained. It must be able to operate as a stand-alone entity on a component platform that provides an implementation of the component model. This usually means that the component is binary and does not have to be compiled before it is deployed. If a component is implemented as a service, it does not have to be deployed by a user of a component. Rather, it is deployed by the service provider.
Documented	Components have to be fully documented so that potential users can decide whether or not the components meet their needs. The syntax and, ideally, the semantics of all component interfaces should be specified.

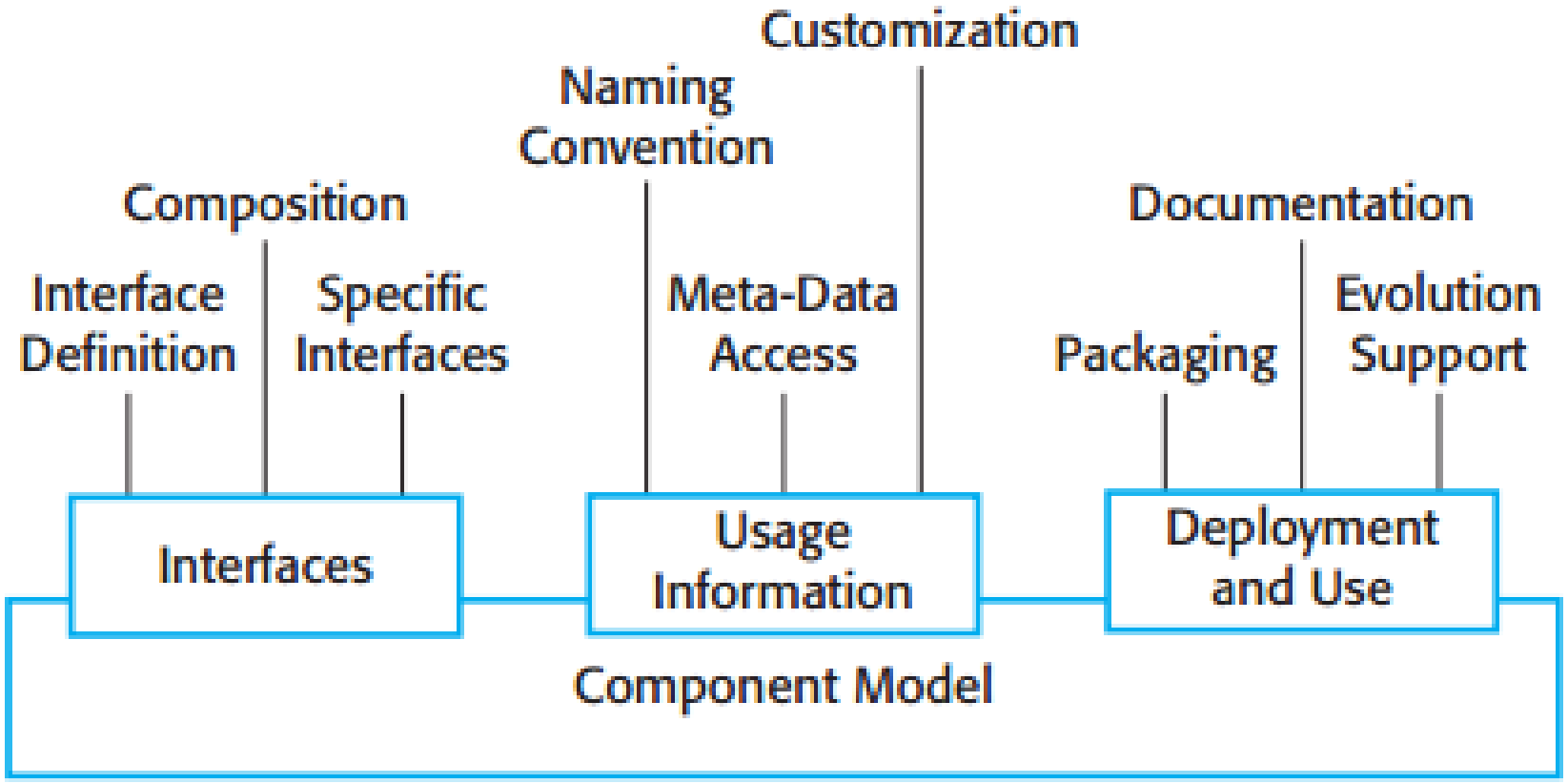


Component models

- A component model is a definition of standards for component implementation, documentation and deployment.
- These standards are for component developers to ensure that components can interoperate.
- They are also for providers of component execution infrastructures who provide middleware to support component operation.
- Many component models have been proposed, but the most important models are 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.

Component models…

Basic elements of a component model



Component models…

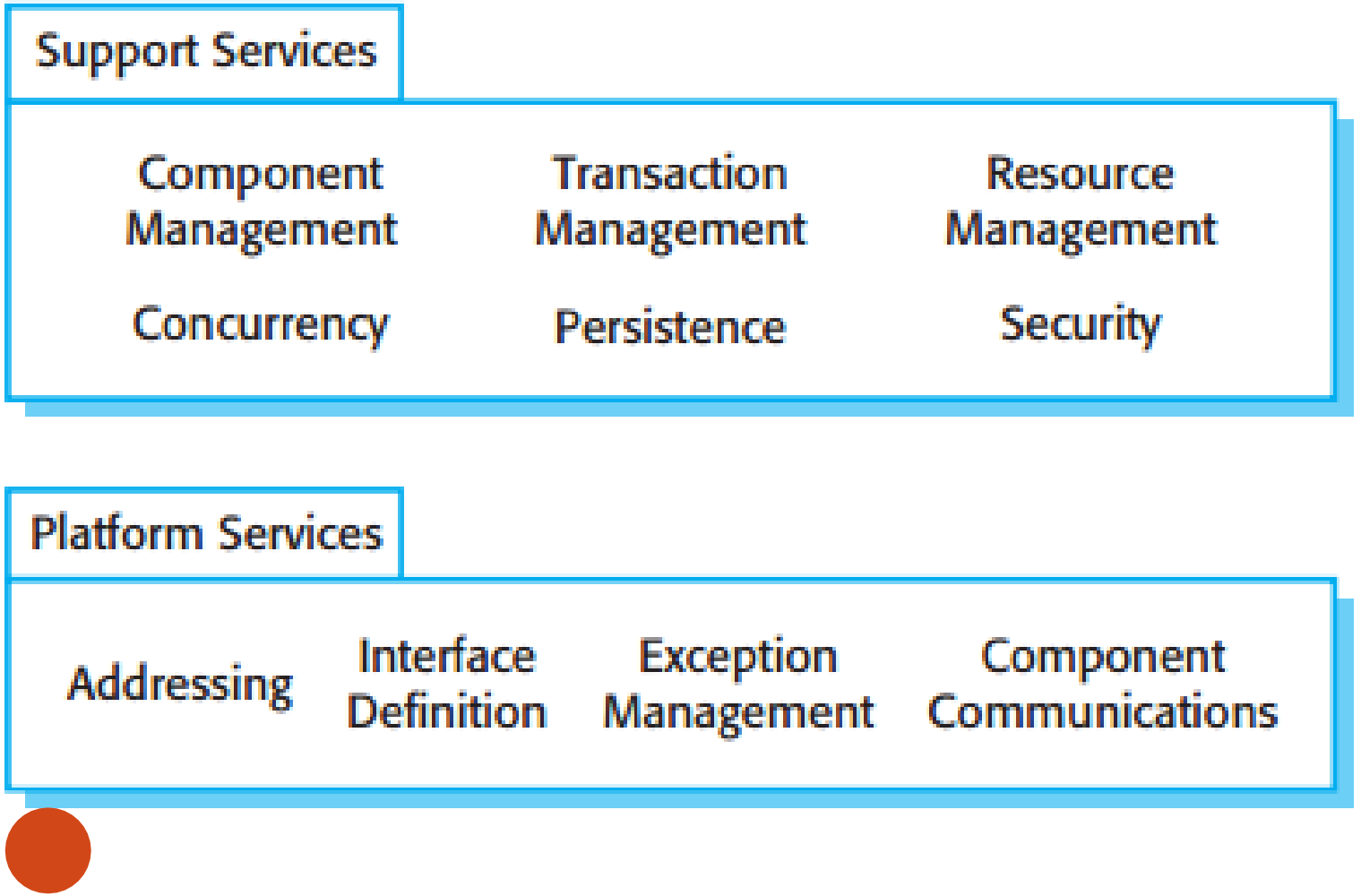
- Elements of a component model
  - Interfaces
    - Components are defined by specifying their interfaces.
    - The component model specifies how the interfaces should be defined and the elements, such as operation names, parameters and exceptions, which should be included in the interface definition.
  - Usage
    - In order for components to be distributed and accessed remotely, they need to have a unique name or handle associated with them.This has to be globally unique.
  - Deployment
    - The component model includes a specification of how components should be packaged for deployment as independent, executable entities

Component models…

- **Middleware support**
  - Middleware supports the execution of components
  - The services provided by a component model implementation fall into two categories:
    - **Platform services** that enable components to communicate and interoperate in a distributed environment.
      - These are the fundamental services that must be available in all component-based systems.
    - **Support services** are common services that are likely to be required by many different components
      - For example, many components require authentication to ensure that the user of component services is authorized.
  - 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 models…

Middleware services defined in a component model

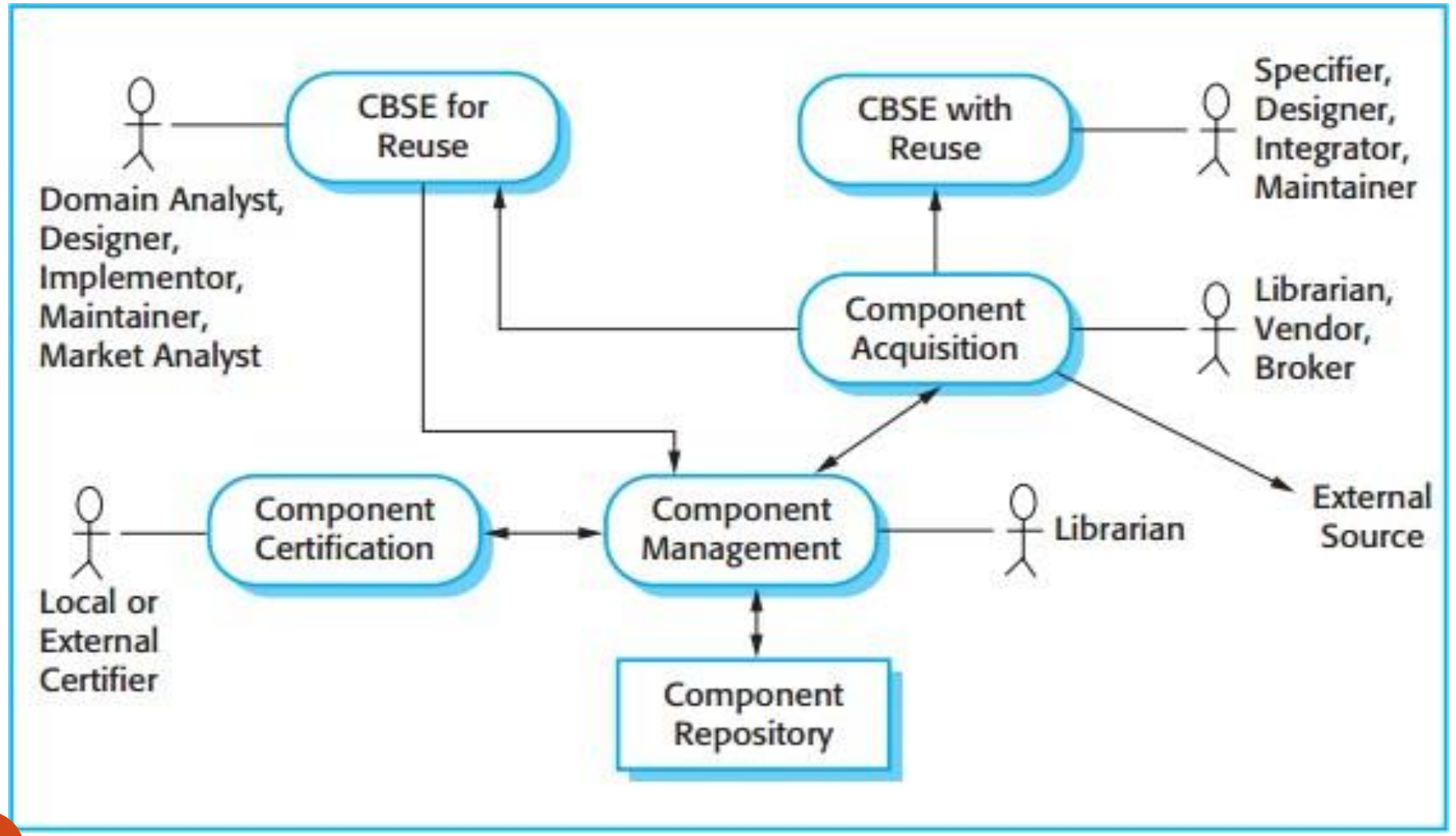


CBSE processes

- CBSE processes are software processes that support CBSE
  - They take into account the possibilities of reuse and the different process activities involved in developing and using reusable components.
- At the highest level, there are two types of CBSE processes:
  - **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.
  - **Development with reuse**
    - This process is the process of developing new applications using existing components and services.

CBSE processes…

- CBSE Processes



CBSE processes…

- Supporting processes
  - **Component acquisition** is the process of acquiring components for reuse or development into a reusable component.
    - It may involve accessing locally- developed components or services or finding these components from an external source.
  - **Component management** is concerned with managing a company's reusable components, ensuring that they are properly catalogued, stored and made available for reuse.
  - **Component certification** is the process of checking a component and certifying that it meets its specification.
  - Components maintained by an organization may be stored in a **component repository** that includes both the components and information about their use

CBSE for reuse

- CBSE for reuse focuses on component development.
- Components developed for a specific application usually have to be generalized to make them reusable.
- 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.
- Components for reuse may be specially constructed by generalising existing components.
- There is a trade-off between reusability and usability
  - The more general the interface, the greater the reusability but it is then more complex and hence less usable.(*generality and understandability*)

CBSE for reuse…

- Component reusability *should reflect stable domain abstractions, should hide state representation, should be as independent as possible,* and *should publish exceptions through the component interface*.
- To make components reusable, you have to adapt and extend the application specific components to create more generic and therefore more reusable version
- Possible 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.

CBSE for reuse…

- **Exception handling**
  - Components should not handle exceptions themselves, because each application will have its own requirements for exception handling.
    - Rather, the component should define what exceptions can arise and should publish these as part of the interface.
  - In practice, however, there are two problems with this
    - Publishing all exceptions leads to bloated interfaces that are harder to understand. This may put off potential users of the component.
    - The operation of the component may depend on local exception handling, and changing this may have serious implications for the functionality of the component.

CBSE for reuse…

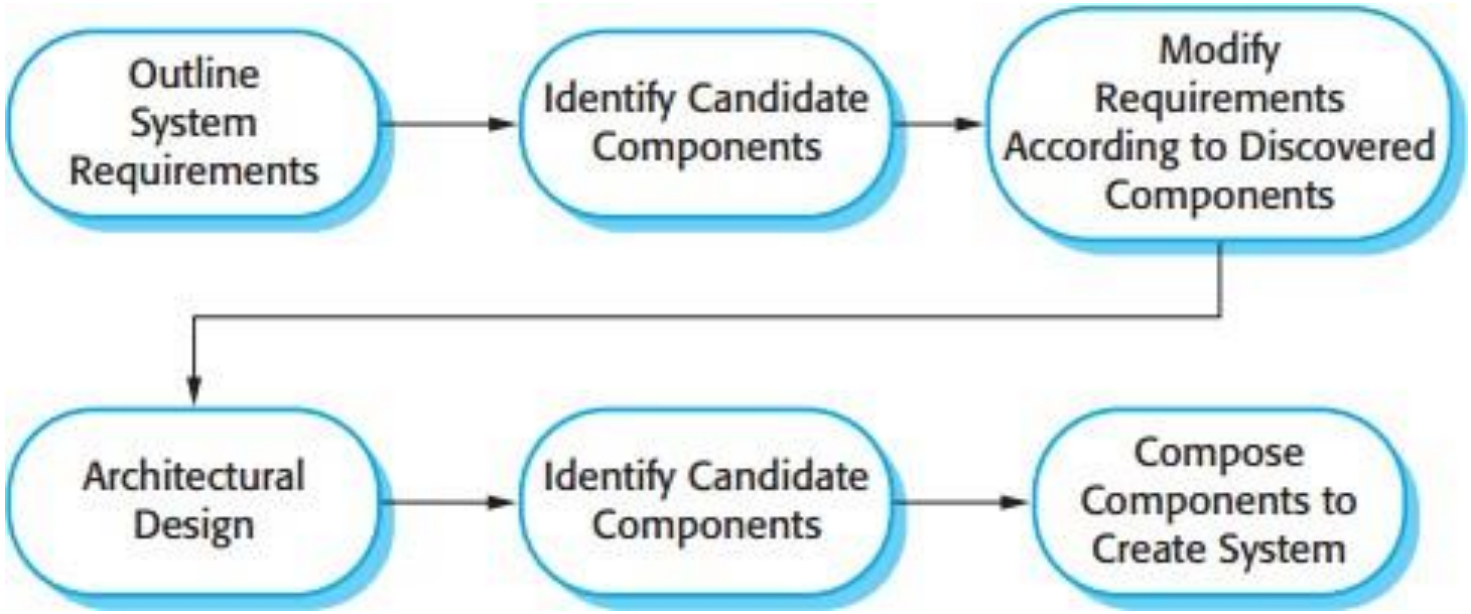
- **Legacy system components**
  - A potential source of components is existing legacy systems.
  - That is existing legacy systems that fulfill a useful business function can be re-packaged as components for reuse.
  - This involves writing a wrapper component that implements provides and requires interfaces then accesses the legacy system.
  - The wrapper hides the complexity of the underlying code and provides an interface for external components to access services that are provided.
  - Although costly, this can be much less expensive than rewriting the legacy system.

CBSE for reuse…

- **Component management**
  - Once you have developed and tested a reusable component or service, this then has to be managed for future reuse.
  - Component management involves
    - deciding how to classify the component so that it can be discovered,
    - making the component available either in a repository or as a service,
    - maintaining information about the use of the component and
    - keeping track of different component versions.
  - A company with a reuse program may carry out some form of component certification before the component is made available for reuse.
    - Certification means that someone apart from the developer checks the quality of the component.

CBSE with reuse

- CBSE with reuse process has to find and integrate reusable components. Principal activities within this process are



- When reusing components, it is essential to make trade-offs between ideal requirements and the services actually provided by available components.

CBSE with reuse…

- The component identification process

- **Component identification issues**
  - **Trust-**You need to be able to trust the supplier of a component. At best, an untrusted component may not operate as advertised; at worst, it can breach your security.
  - **Requirements -** Different groups of components will satisfy different requirements.
  - **Validation**
    - The component specification may not be detailed enough to allow comprehensive tests to be developed.
    - Components may have unwanted functionality. How can you test this will not interfere with your application?

CBSE with reuse…

- **Component validation**
  - Component validation involves developing a set of test cases for a component (or, possibly, extending test cases supplied with that component) and developing a test harness to run component tests.
    - The major problem with component validation is that the component specification may not be sufficiently detailed to allow you to develop a complete set of component tests.
  - As well as testing that a component for reuse does what you require, you may also have to check that the component does not include any malicious code or functionality that you don’t need.

CBSE with reuse…

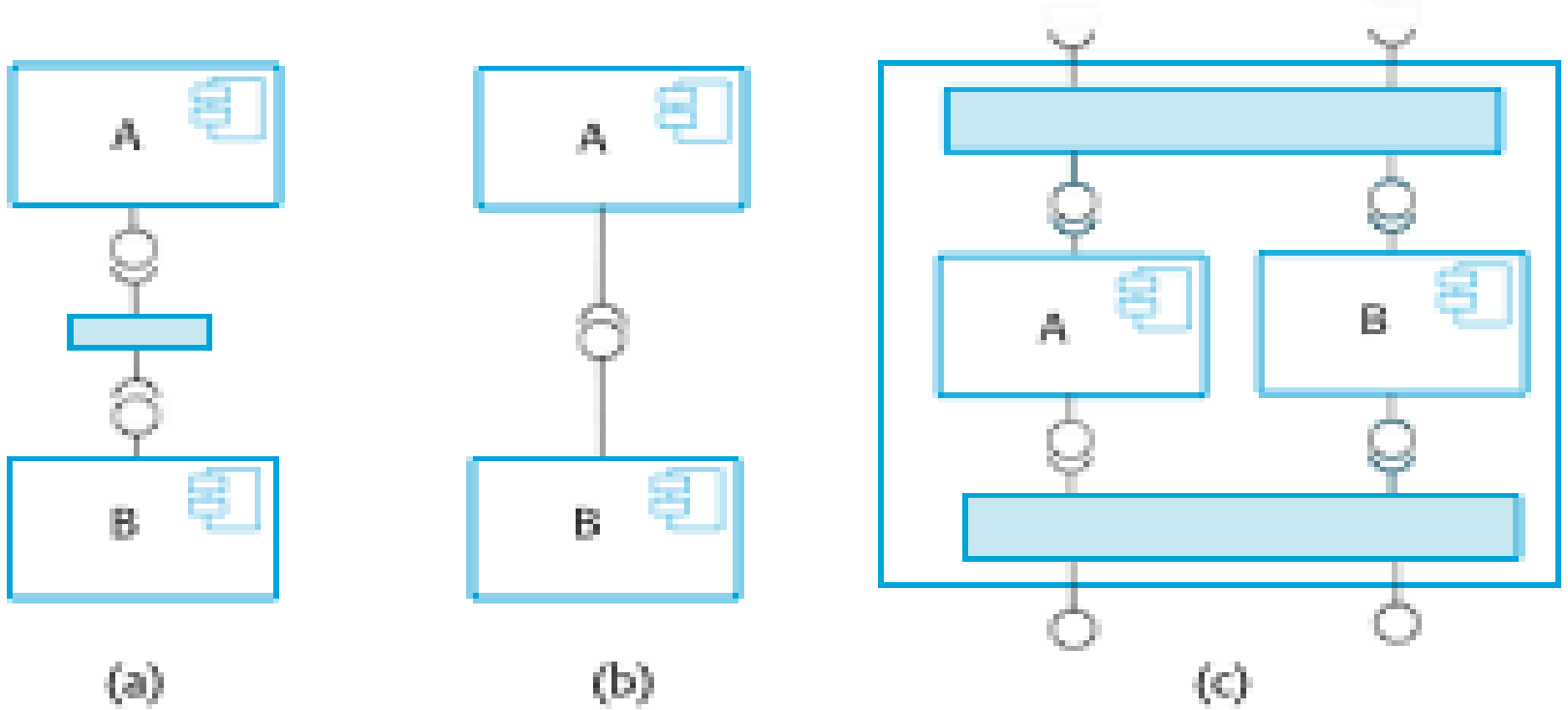
- **Ariane launcher failure – validation failure?**
  - In 1996, the 1st test flight of the Ariane 5 rocket ended in disaster when the launcher went out of control 37 seconds after take off.
  - The problem was due to a reused component from a previous version of the launcher (the Inertial Navigation System) that failed because assumptions made when that component was developed did not hold for Ariane 5.
  - The functionality that failed in this component was not required in Ariane 5.

Component composition

- The process of assembling components to create a system.
- Composition involves integrating components with each other and with the component infrastructure.
- To enable composition, components must be compatible: achieved by component model.
- **Types of composition**
  - ***Sequential composition*** where the composed components are executed in sequence.This involves composing the provides interfaces of each component.
  - ***Hierarchical composition*** where one component calls on the services of another directly.The provides interface of one component is composed with the requires interface of another.
  - ***Additive composition*** where the interfaces of two components are put together to create a new component. Provides and requires interfaces of integrated component is a combination of interfaces of constituent components.

Component composition…

Types of component composition



The components do not call each other in sequential composition

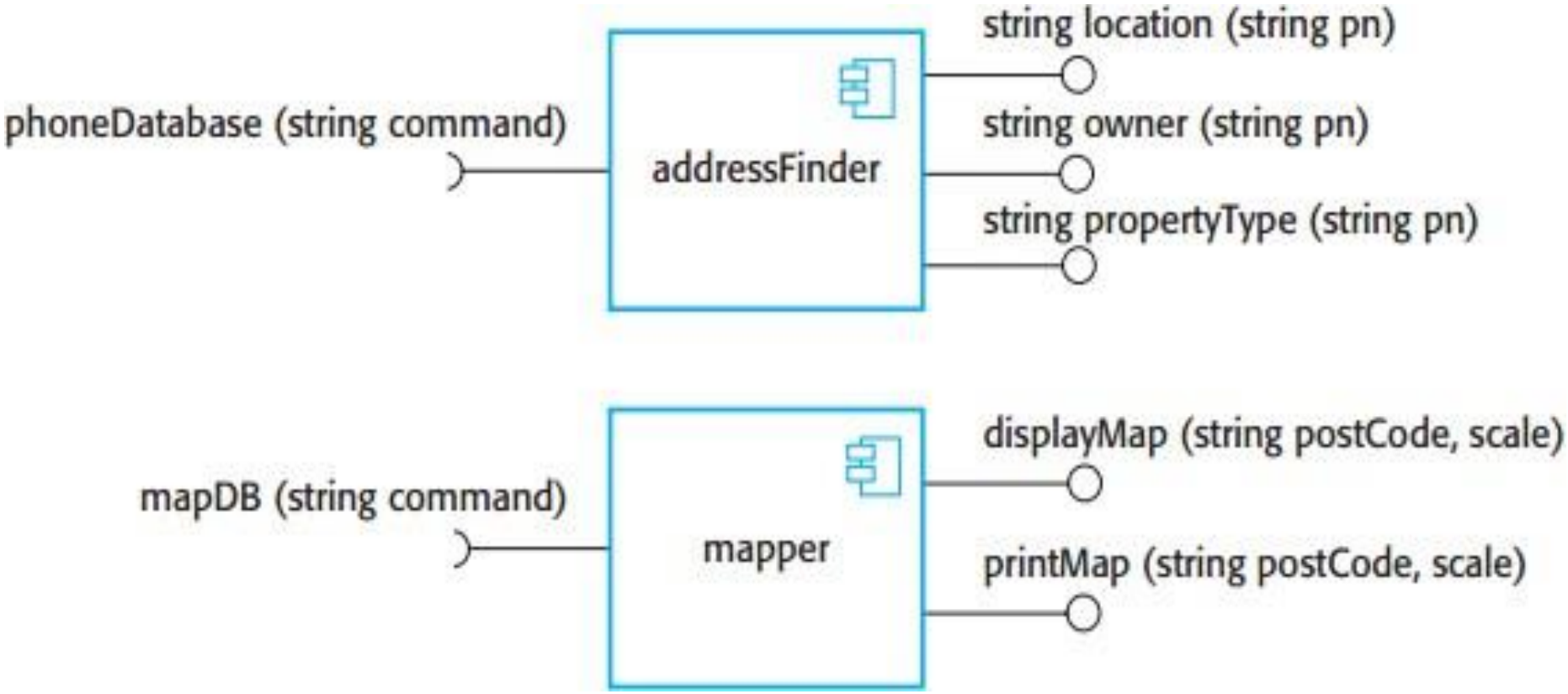
A calls B directly &if their interfaces match, there may be no need for additional code.

A and B are not dependent and do not call each other

Component composition…

- When components are developed independently for reuse, you will often be faced with *interface incompatibilities*.
- This means that the interfaces of the components that you wish to compose are not the same.
- Three types of interface incompatibility can occur:
  - **Parameter incompatibility -** the operations on each side of the interface have the same name but their parameter types or the number of parameters are different.
  - **Operation incompatibility -** the names of the operations in the 'provides' and 'requires' interfaces are different.
  - **Operation incompleteness-** the 'provides' interface of a component is a subset of the 'requires' interface of another component or vice versa.

Component composition…



Components with incompatible interfaces

Component composition…

- The problem of incompatibility is tackled by writing an adaptor that reconciles the interfaces of the two components being reused. (**Adaptor components**)
- An adaptor component converts one interface to another
- Different types of adaptor are required depending on the type of composition.
- An **addressFinder** and a **mapper** component may be composed through an adaptor that strips the postal code from an address and passes this to the mapper component.
  - These components are composable in principle because the property location includes the post or ZIP code.

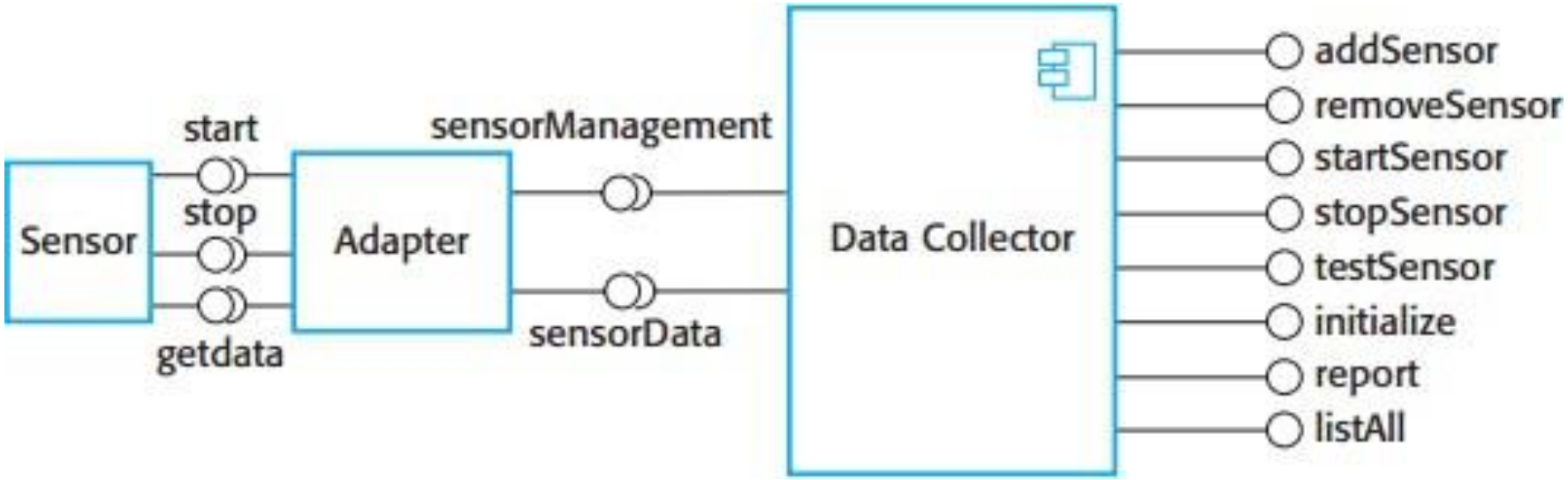
Component composition…

- However, you have to write an adaptor component called postCodeStripper that takes the location data from addressFinder and strips out the post code.
- This post code is then used as an input to mapper and the street map is displayed at a scale of 1:10,000
- The following code, which is an example of sequential composition, illustrates the sequence of calls that is required to implement this:

```
address = addressFinder.location (phonenumber) ;
postCode = postCodeStripper.getPostCode (address) ;
mapper.displayMap(postCode, 10000)
```

Component composition…

- **An adaptor linking a data collector and a sensor**

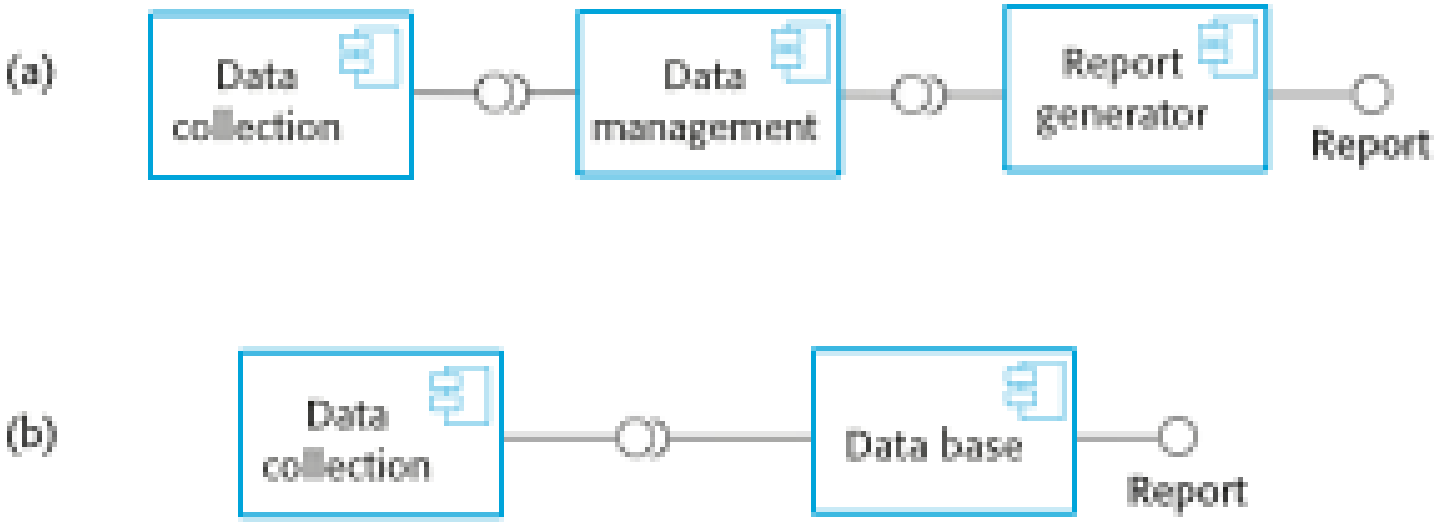


Another case in which an adaptor component may be used in hierarchical composition, where one component wishes to make use of another but there is an incompatibility between the 'provides' interface and 'requires' interface of the components in the composition.

Component composition…

- **Composition trade-offs**
  - When composing components, you may find conflicts between functional and non-functional requirements, and conflicts between the need for rapid delivery and system evolution.
  - You need to make decisions such as:
    - What composition of components is effective for delivering the functional requirements?
    - What composition of components allows for future change?
    - What will be the emergent properties of the composed system?

Component composition…



- Data collection and report generation components
  - there is a potential conflict between adaptability and performance. Composition (a) is more adaptable but composition (b) is perhaps faster and more reliable.