Component-Based Software Engineering

Andy Carpenter

School of Computer Science

(Andy.Carpenter@manchester.ac.uk)

Elements these slides come from Sommmerville, author of “Software Engineering", and are copyright Sommerville

Component-based development

- ✧ Component-based software engineering (CBSE) is an approach to software development that relies on the reuse of entities called 'software components'.
- ✧ 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.

CBSE essentials

- ✧ Independent components specified by their interfaces.
- ✧ Component standards to facilitate component integration.
- ✧ Middleware that provides support for component inter-operability.
- ✧ A development process that is geared to reuse.

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;
  - One components can be replaced by another if its interface is maintained;
  - Component infrastructures offer a range of standard services.

Component standards

- Standards need to be established so that components can communicate with each other and inter-operate.
- Unfortunately, several competing component standards were established:
  - Sun’s Enterprise Java Beans
  - Microsoft’s COM and .NET
  - CORBA’s CCM
- In practice, these multiple standards have hindered the uptake of CBSE. It is impossible for components developed using different approaches to work together.

Service-oriented software engineering

- ✧ An executable service is a type of independent component. It has a 'provides' interface but not a 'requires' interface.
- ✧ From the outset, services have been based around standards so there are no problems in communicating between services offered by different vendors.
- ✧ System performance may be slower with services but this approach is replacing CBSE in many systems.
- ✧ Covered in Chapter 18

Component definitions

- Councill and Heinmann:
  - *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.*
- Szyperski:
  - *A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third-parties.*

Component characteristics

Component characteristic	**Description**
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.
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.

Component characteristics

Component characteristic	**Description**
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.
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.
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.

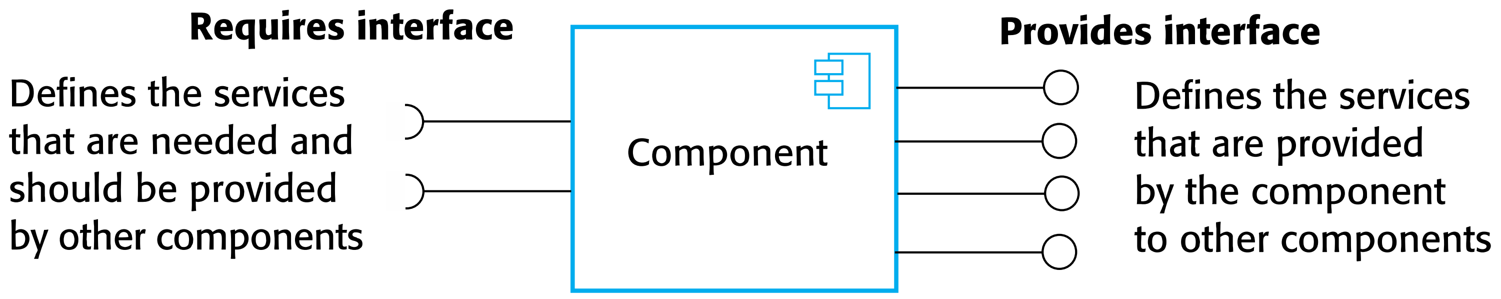
Component as a service provider

- 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.
- The component interface is expressed in terms of parameterized operations and its internal state is never exposed.

Component interfaces

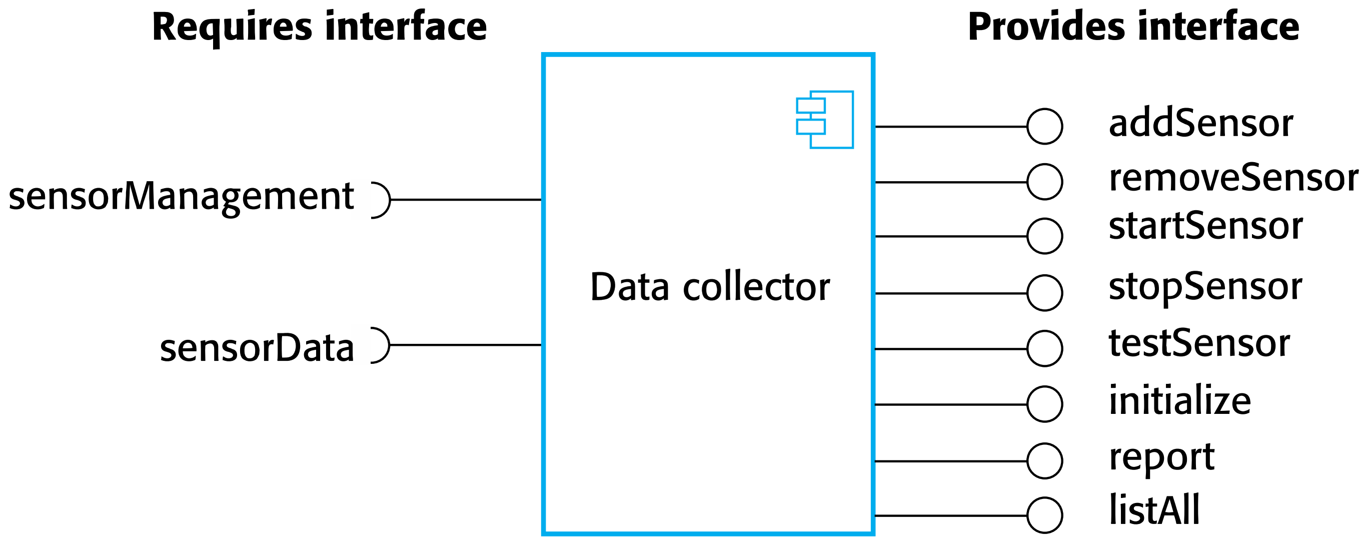
- ✧ Provides interface
  - ▪ Defines the services that are provided by the component to other components.
  - ▪ This interface, essentially, is the component API. It defines the methods that can be called by a user of the component.
- ✧ Requires interface
  - ▪ Defines the services that specifies what 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.

Component interfaces


Requires interface
Defines the services that are needed and should be provided by other components
Component
Provides interface
Defines the services that are provided by the component to other components

Note UML notation. Ball and sockets can fit together.

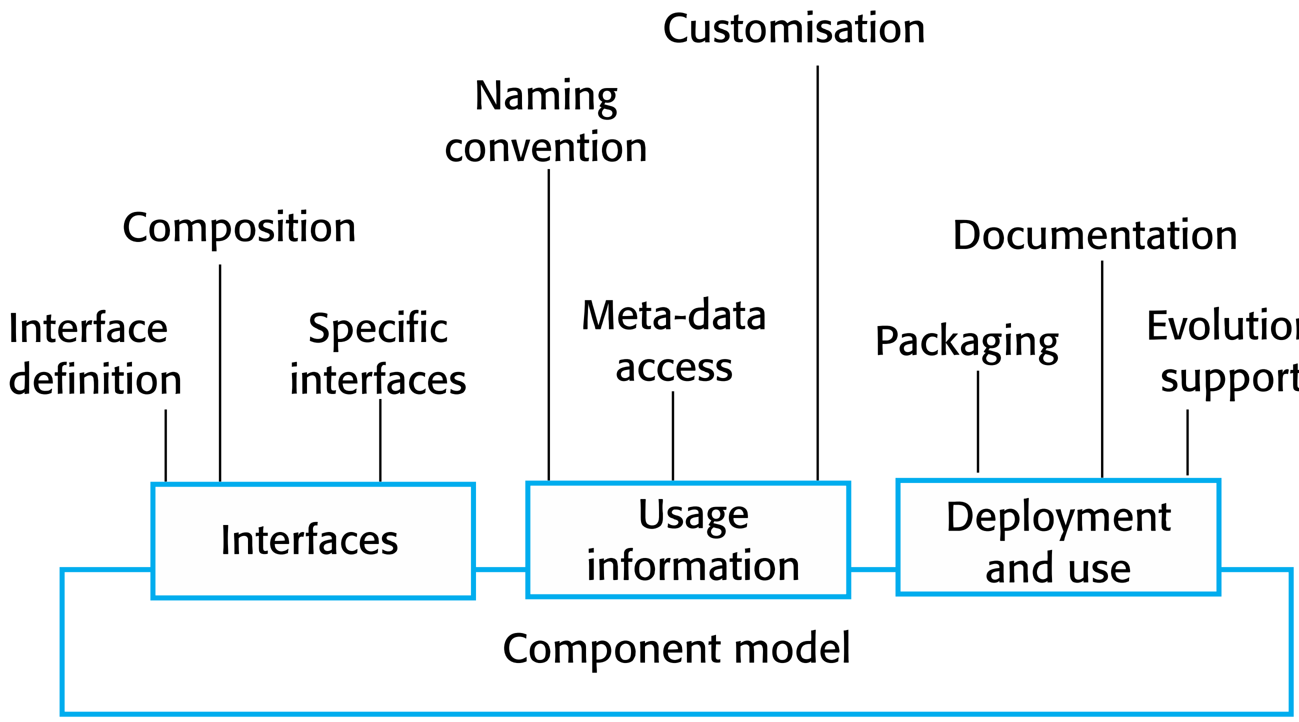
A model of a data collector component



Component models

- ✧ A component model is a definition of standards for component implementation, documentation and deployment.
- ✧ Examples of component models
  - 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.

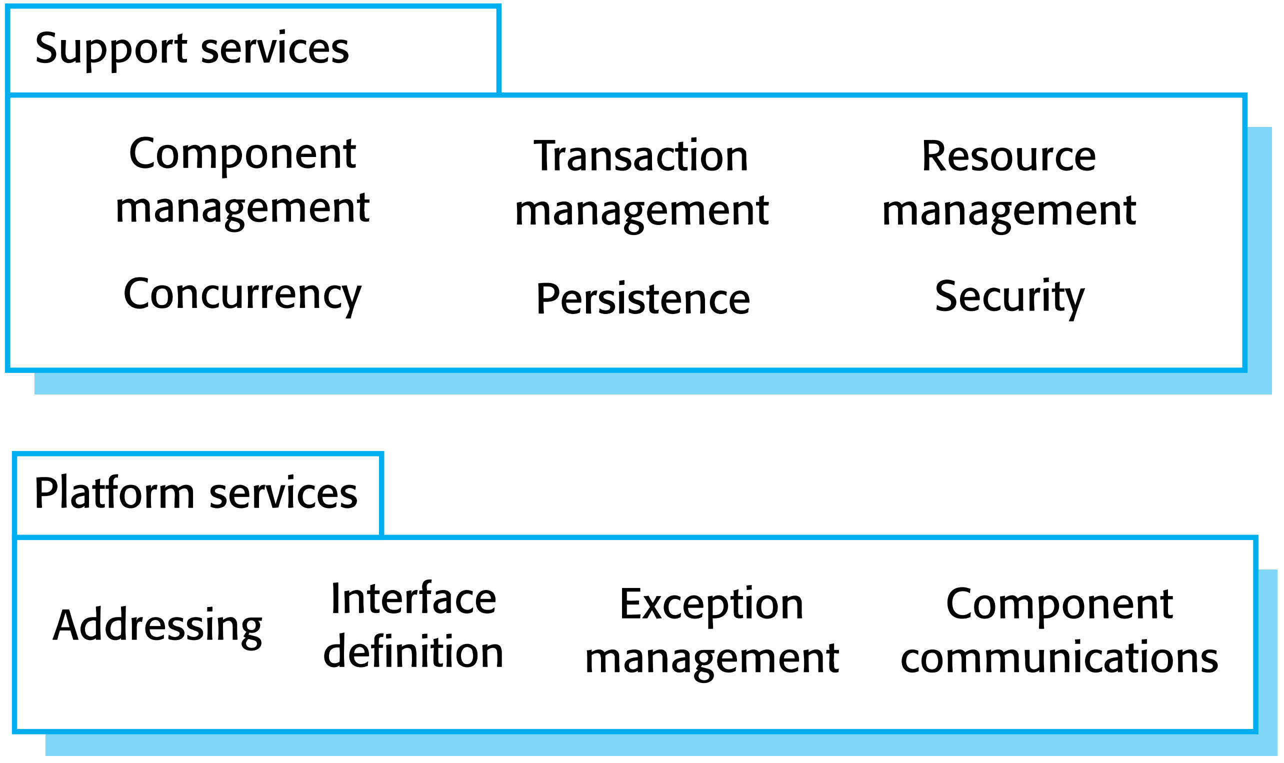
Basic elements of a component model



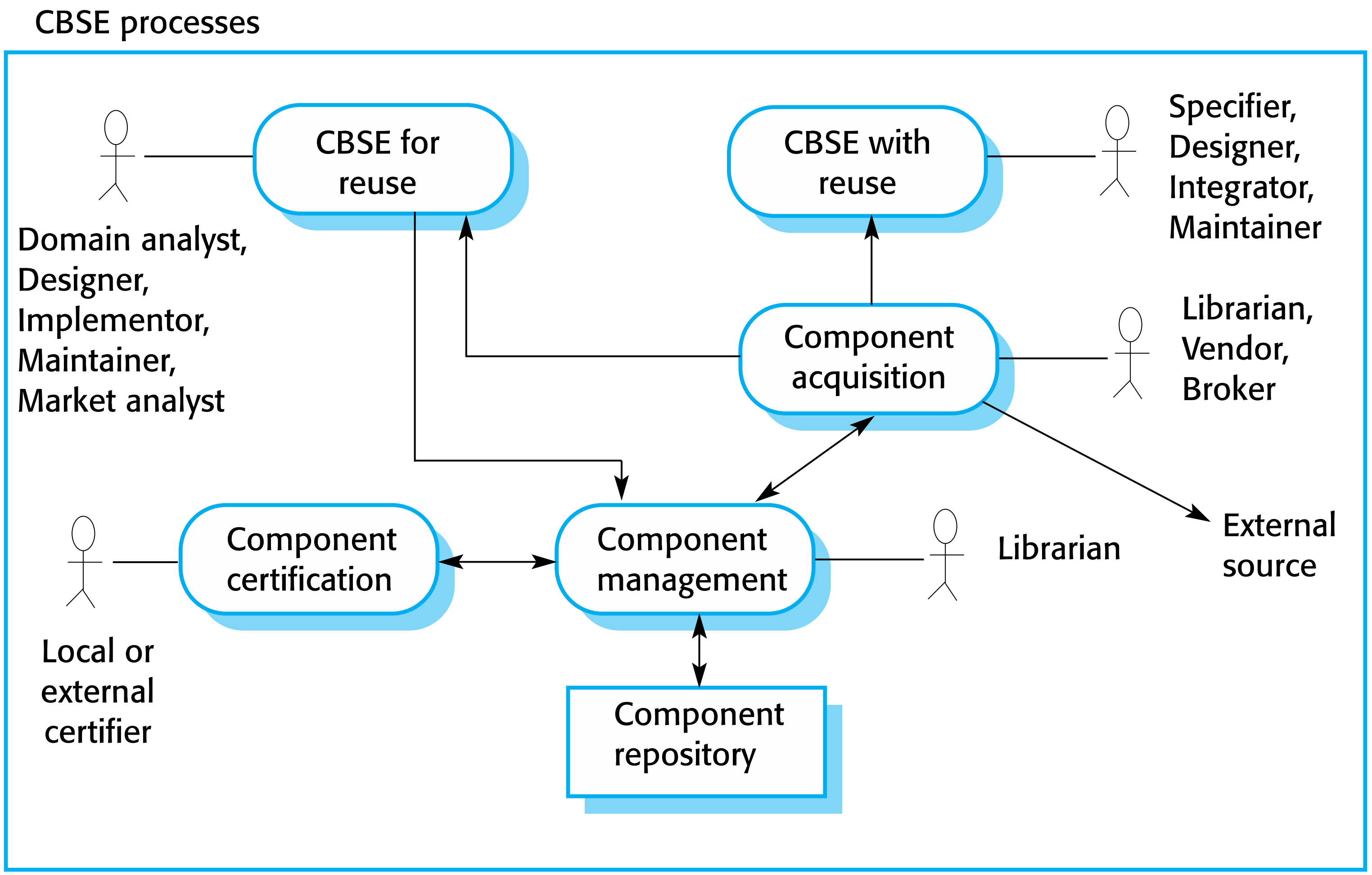
Middleware support

- 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;
  - Support 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.

Middleware services defined in a component model



CBSE processes


CBSE processes
CBSE for reuse
CBSE with reuse
Domain analyst, Designer, Implementor, Maintainer, Market analyst
Specifier, Designer, Integrator, Maintainer
Component acquisition
Librarian, Vendor, Broker
External source
Component certification
Component management
Librarian
Local or external certifier
Component repository

CBSE for reuse

- ✧ CBSE for reuse focuses on component development.
- ✧ Components developed for a specific application usually have to be generalised 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.

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.

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.

Reusable components

- ✧ The development cost of reusable components may be higher than the cost of specific equivalents. This extra reusability enhancement cost should be an organization rather than a project cost.
- ✧ Generic components may be less space-efficient and may have longer execution times than their specific equivalents.

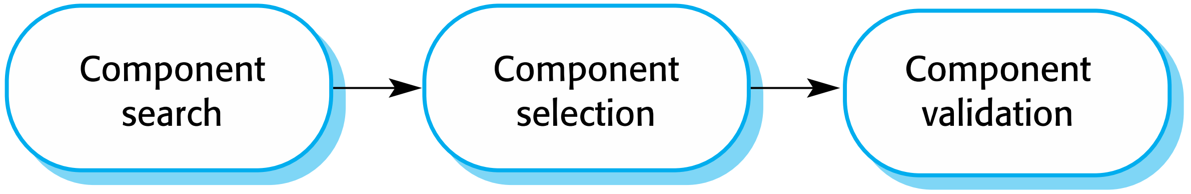
Component management

- ✧ 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.
- ✧ When reusing components, it is essential to make trade-offs between ideal requirements and the services actually provided by available components.
- ✧ This involves:
  - Developing outline requirements;
  - Searching for components then modifying requirements according to available functionality.
  - Searching again to find if there are better components that meet the revised requirements.
  - Composing components to create the system.

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?

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.

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.

Key points

- ✧ CBSE is a reuse-based approach to defining and implementing loosely coupled components into systems.
- ✧ A component is a software unit whose functionality and dependencies are completely defined by its interfaces.
- ✧ Components may be implemented as executable elements included in a system or as external services.
- ✧ A component model defines a set of standards that component providers and composers should follow.
- ✧ The key CBSE processes are CBSE for reuse and CBSE with reuse.

Key points

- ✧ During the CBSE process, the processes of requirements engineering and system design are interleaved.
- ✧ Component composition is the process of ‘wiring’ components together to create a system.
- ✧ When composing reusable components, you normally have to write adaptors to reconcile different component interfaces.
- ✧ When choosing compositions, you have to consider required functionality, non-functional requirements and system evolution.