Software Engineering

Component-Based Software Engineering

Based on Software Engineering, 7th Edition by Ian Sommerville

Objectives

- To explain that CBSE is concerned with developing standardised components and composing these into applications
- To describe components and component models
- To show the principal activities in the CBSE process
- To discuss approaches to component composition and problems that may arise

Component-based development

- Component-based software engineering (CBSE) is an approach to software development that relies on software reuse.
- 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.
- 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;
  - Component platforms are shared and reduce development costs.
- 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?
  - Requirements trade-offs - how do we do trade-off analysis between the features of one component and another?

Components

- Components provide a service without regard to where the component is executing or its programming language
  - A component is an independent executable entity that can be made up of one or more executable objects
  - The component interface is published and all interactions are through the published interface
- Definitions:
  - *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.*
  - *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.*

Stan Kurkovsky

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.

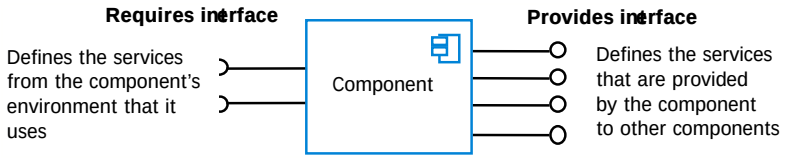
Characteristics

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.
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.

Stan Kurkovsky

Component interfaces

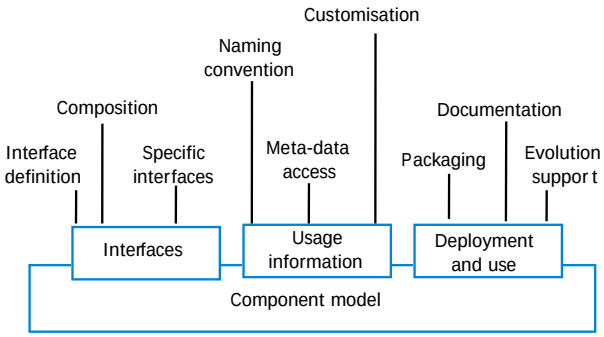
- Provides interface
  - Defines the services that are provided by the component to other components
- Requires interface
  - Defines the services that specifies what services must be made available for the component to execute as specified



Stan Kurkovsky

Component models

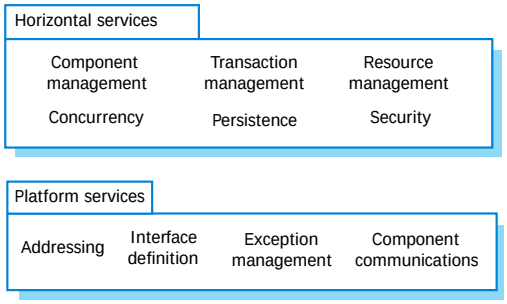
- A component model is a definition of standards for component implementation, documentation and deployment.
- Examples of component models
  - EJB model (Enterprise Java Beans)
  - COM+ model (.NET model)
  - Corba Component Model
- The component model specifies how interfaces should be defined and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



Stan Kurkovsky

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;
  - 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.



Stan Kurkovsky

Component development for reuse

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

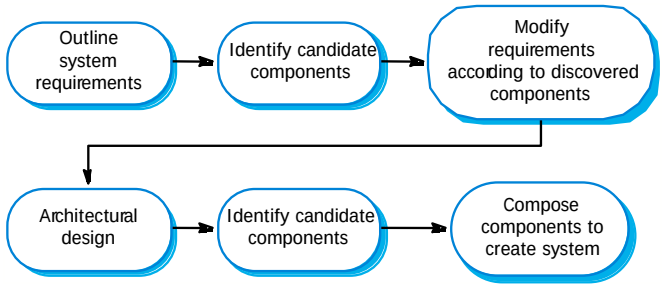
Stan Kurkovsky

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.

The CBSE process

- 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.



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.
- A component model defines a set of standards that component providers and composers should follow.
- 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.