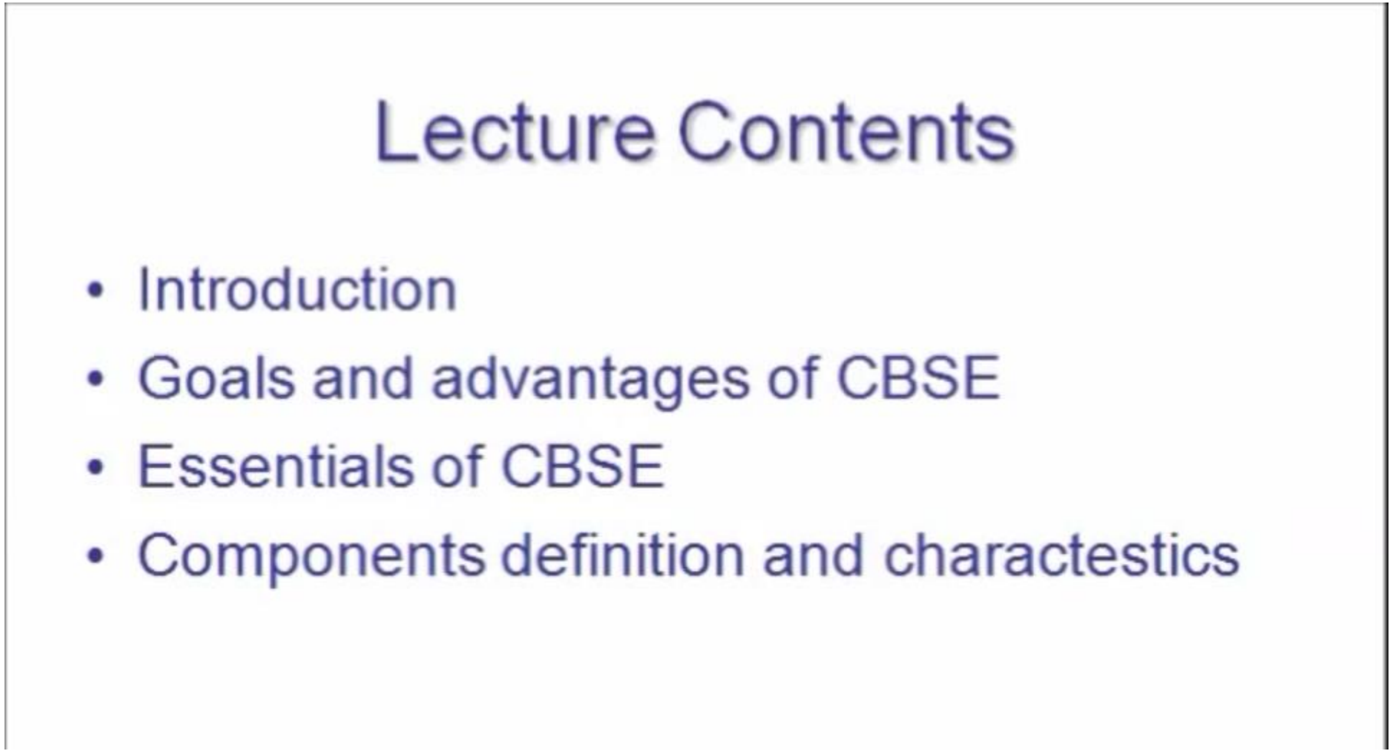
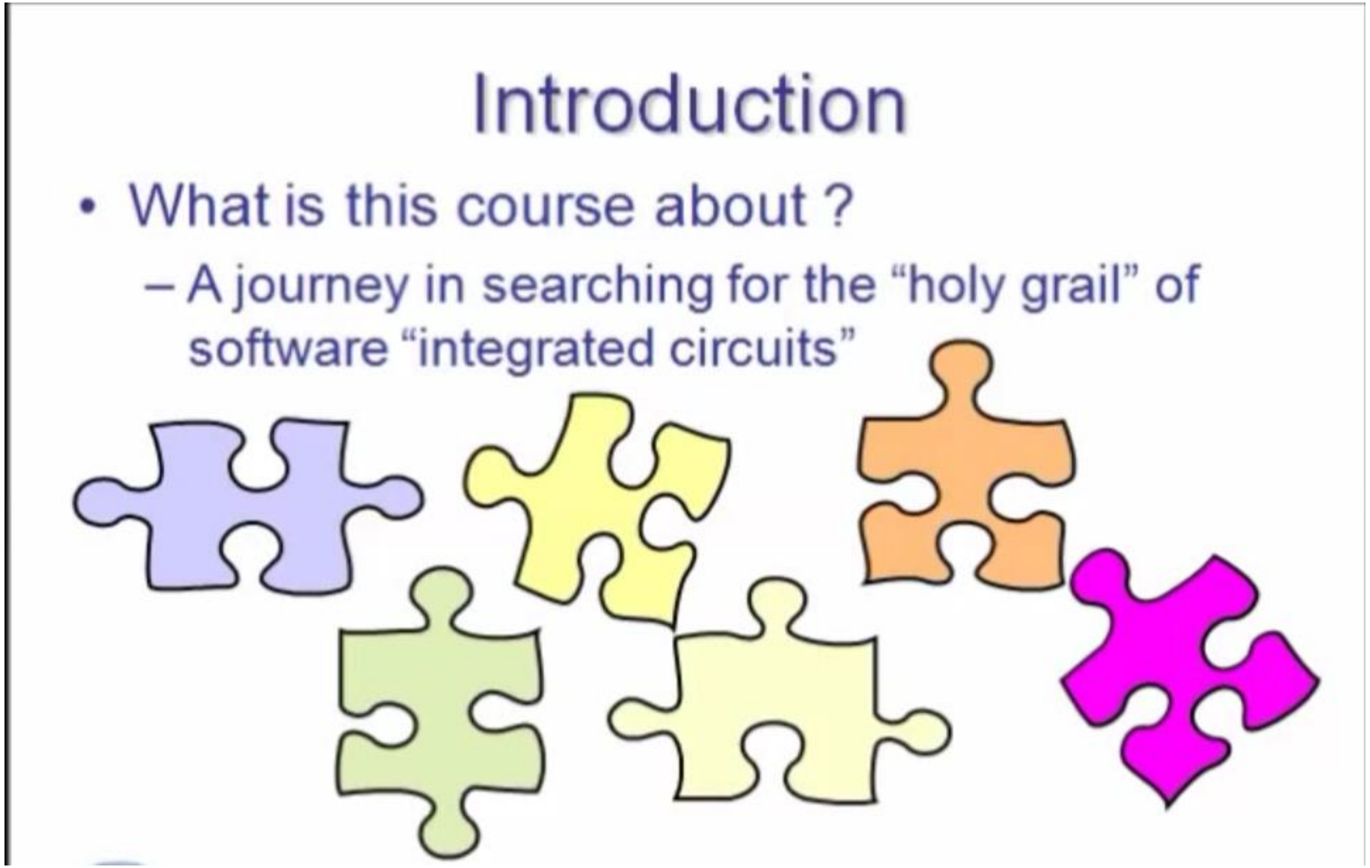
CBSE

Saloua Ben Yahia



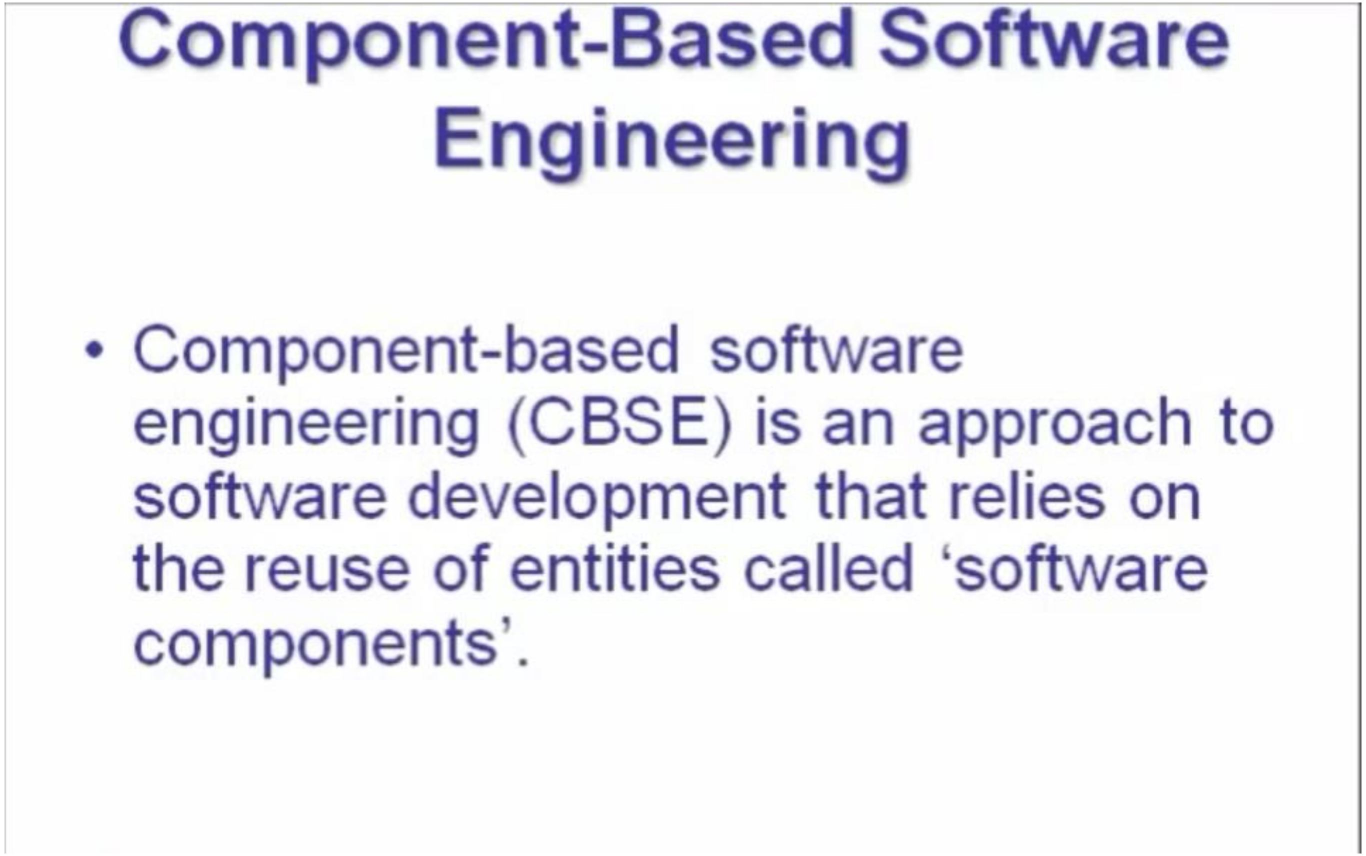
Lecture Contents

- Introduction
- Goals and advantages of CBSE
- Essentials of CBSE
- Components definition and charactestics



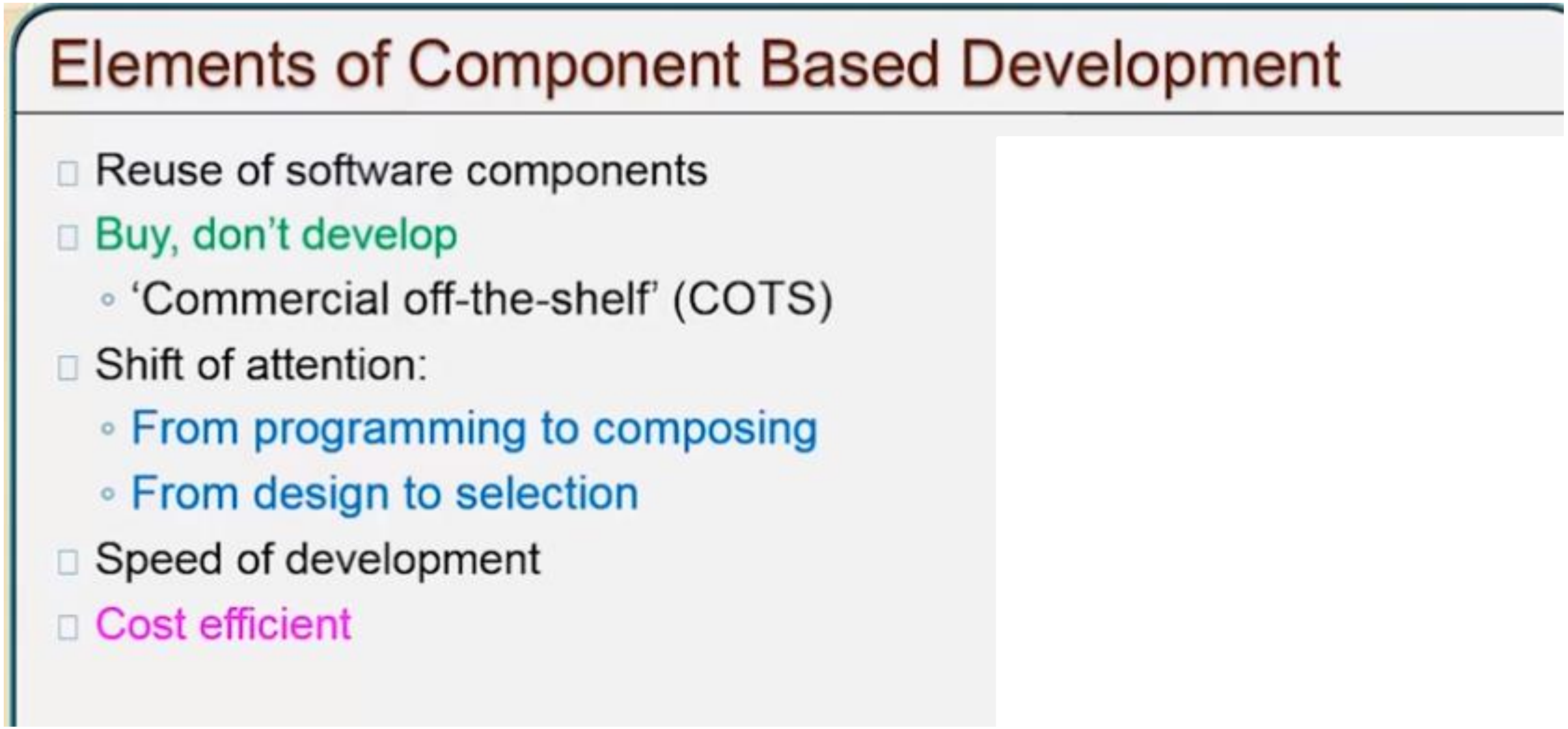
Introduction

- What is this course about ?
  - A journey in searching for the "holy grail" of software "integrated circuits"



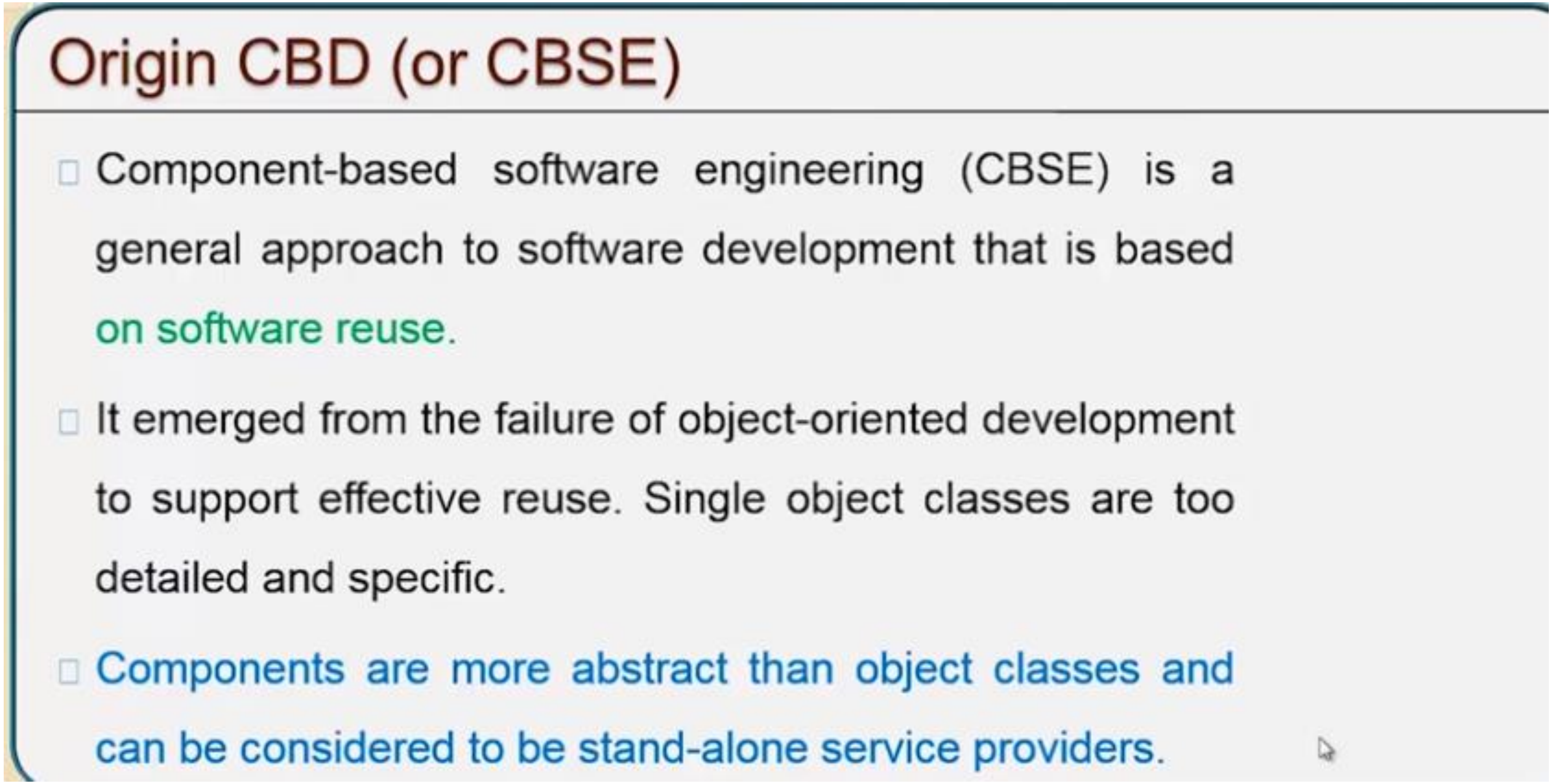
Component-Based Software Engineering

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



Elements of Component Based Development

- Reuse of software components
- Buy, don't develop
  - 'Commercial off-the-shelf' (COTS)
- Shift of attention:
  - From programming to composing
  - From design to selection
- Speed of development
- Cost efficient



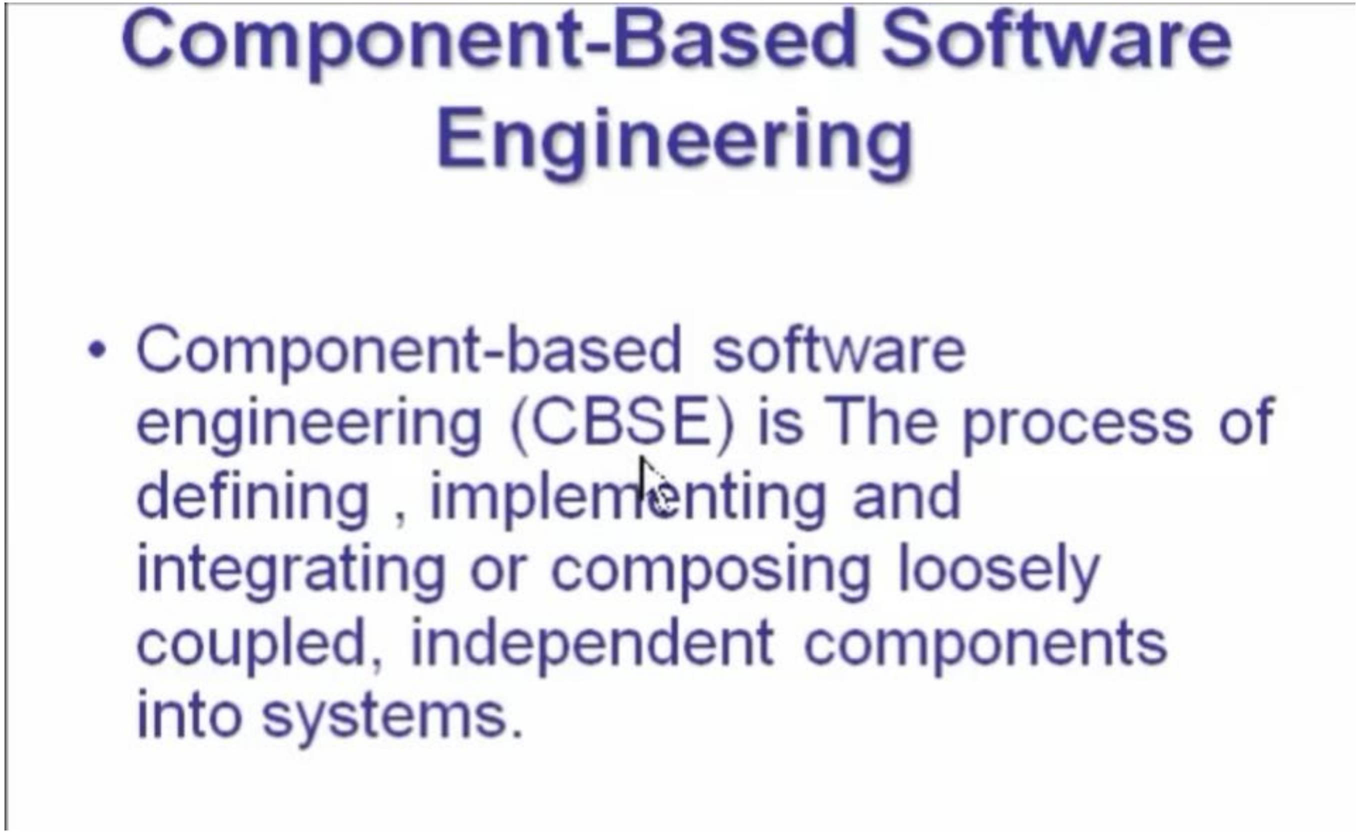
Origin CBD (or CBSE)

- Component-based software engineering (CBSE) is a general approach to software development that is based 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.

Component-Based Software Engineering

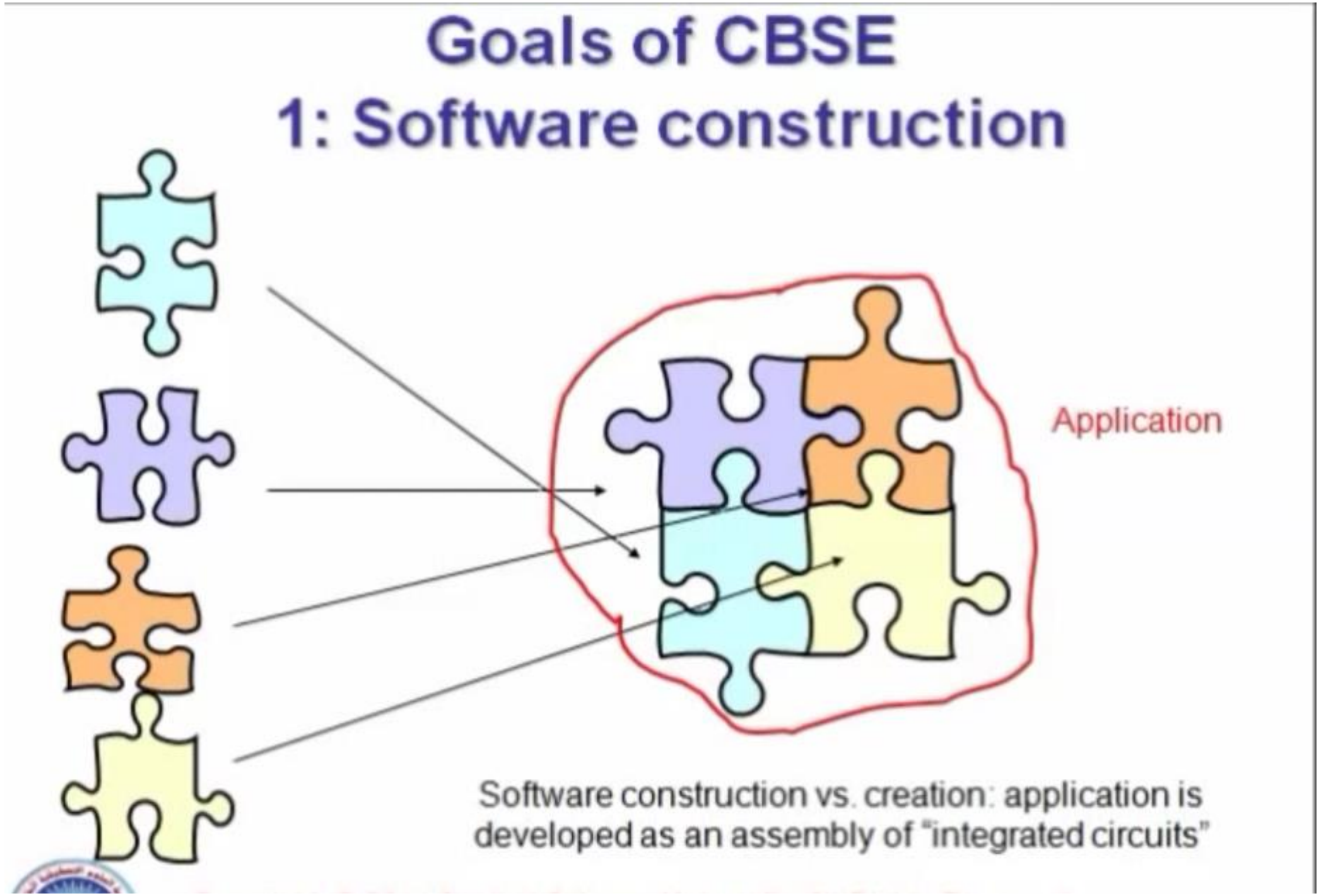
- It emerged from the failure of object-oriented development to support effective reuse. Single object classes are too detailed and specific.

- Components are more abstract than object classes and can be considered to be stand-alone service providers. They can exist as stand-alone entities.

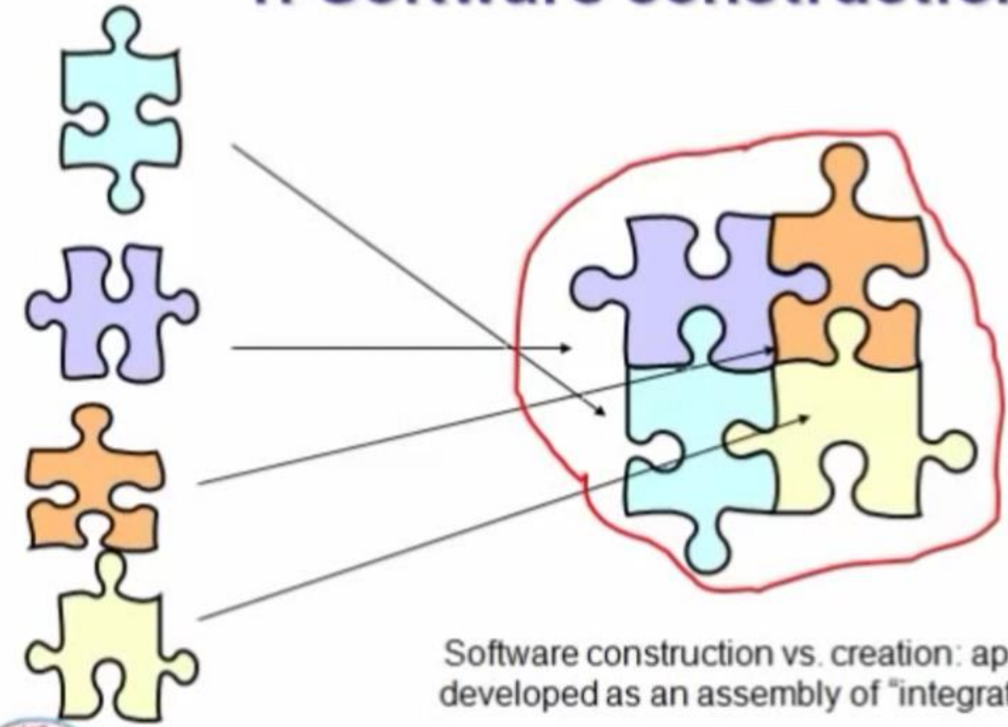


Component-Based Software Engineering

- Component-based software engineering (CBSE) is The process of defining , implementing and integrating or composing loosely coupled, independent components into systems.

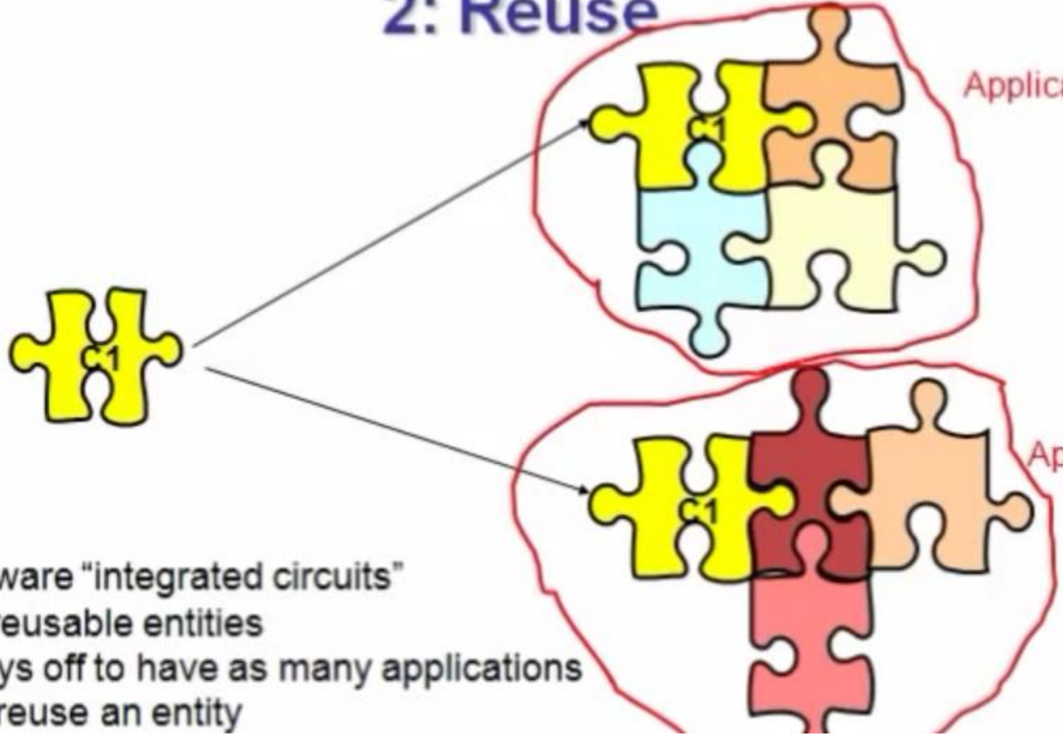


Goals of CBSE
1: Software construction



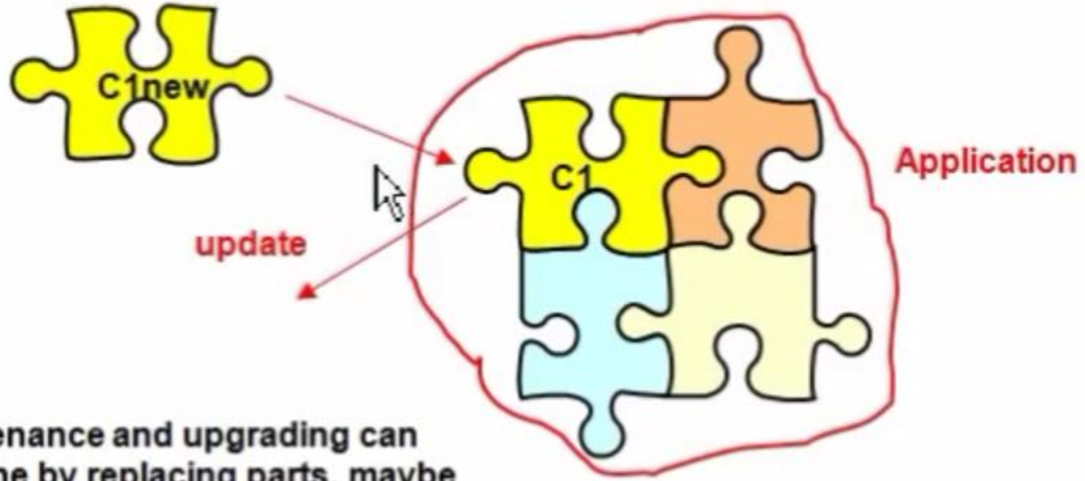
Software construction vs. creation: application is developed as an assembly of "integrated circuits"

Goals of CBSE
2: Reuse

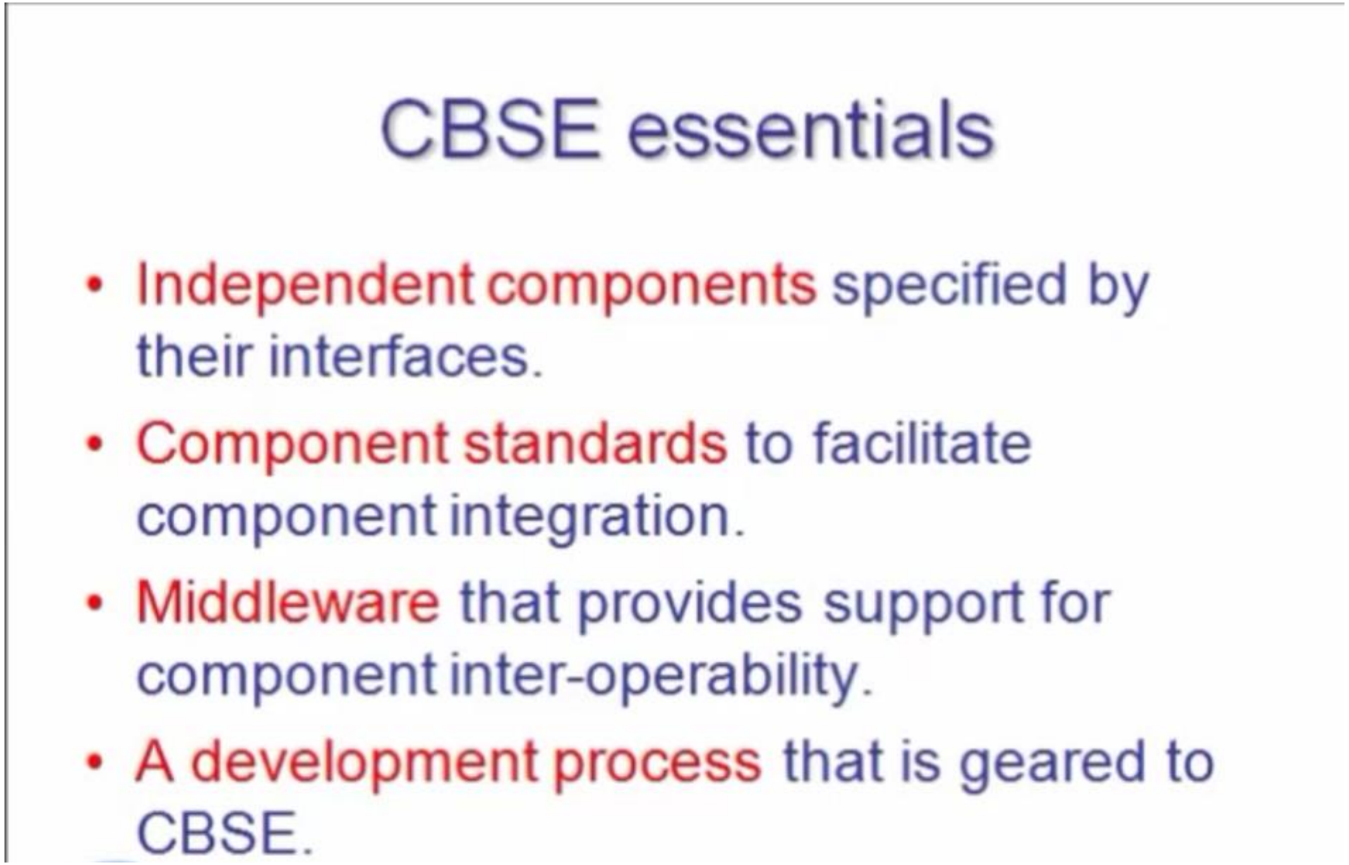


Software "integrated circuits"
are reusable entities
It pays off to have as many applications
that reuse an entity

Goals of CBSE
3: Maintenance & Evolution

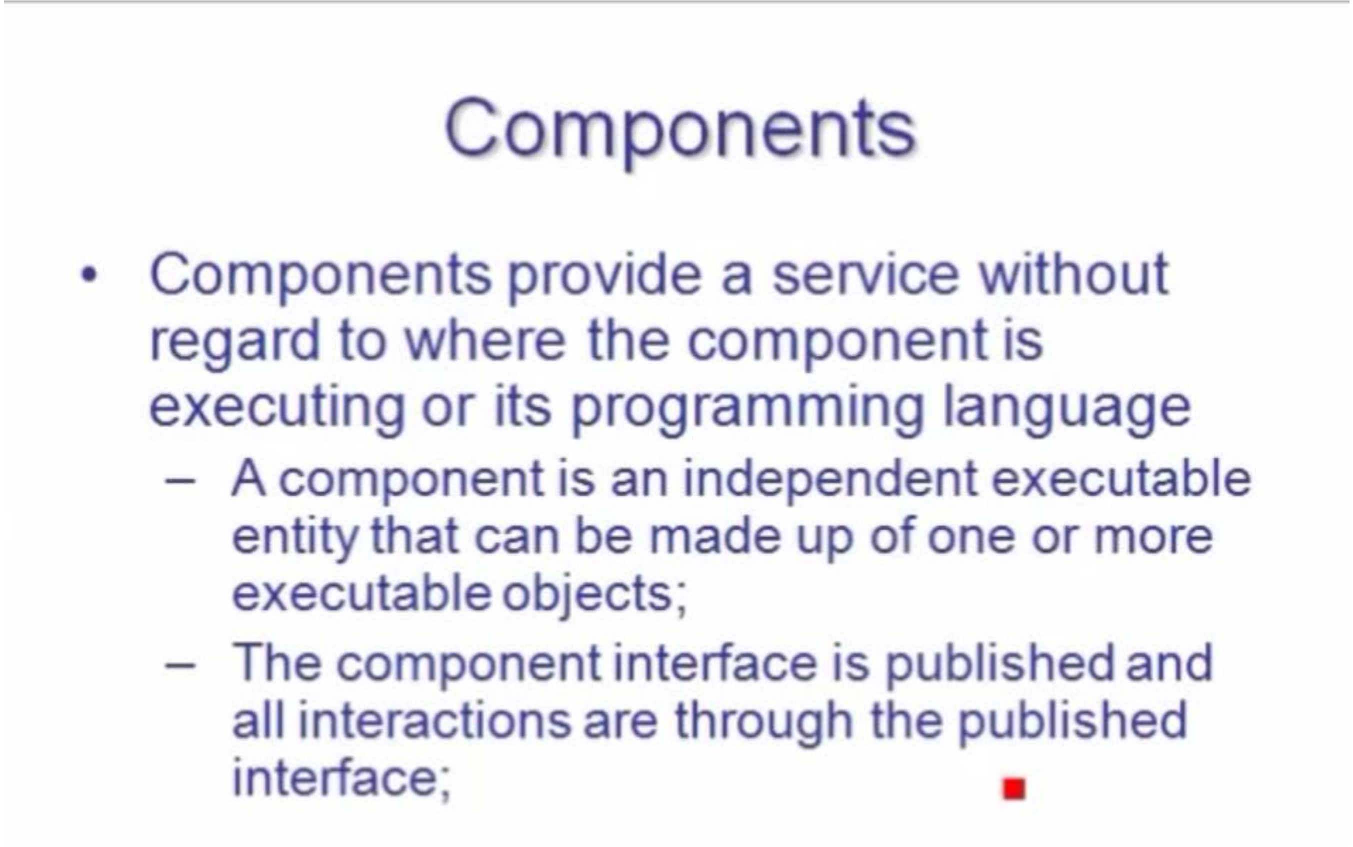


Maintenance and upgrading can be done by replacing parts, maybe even at runtime



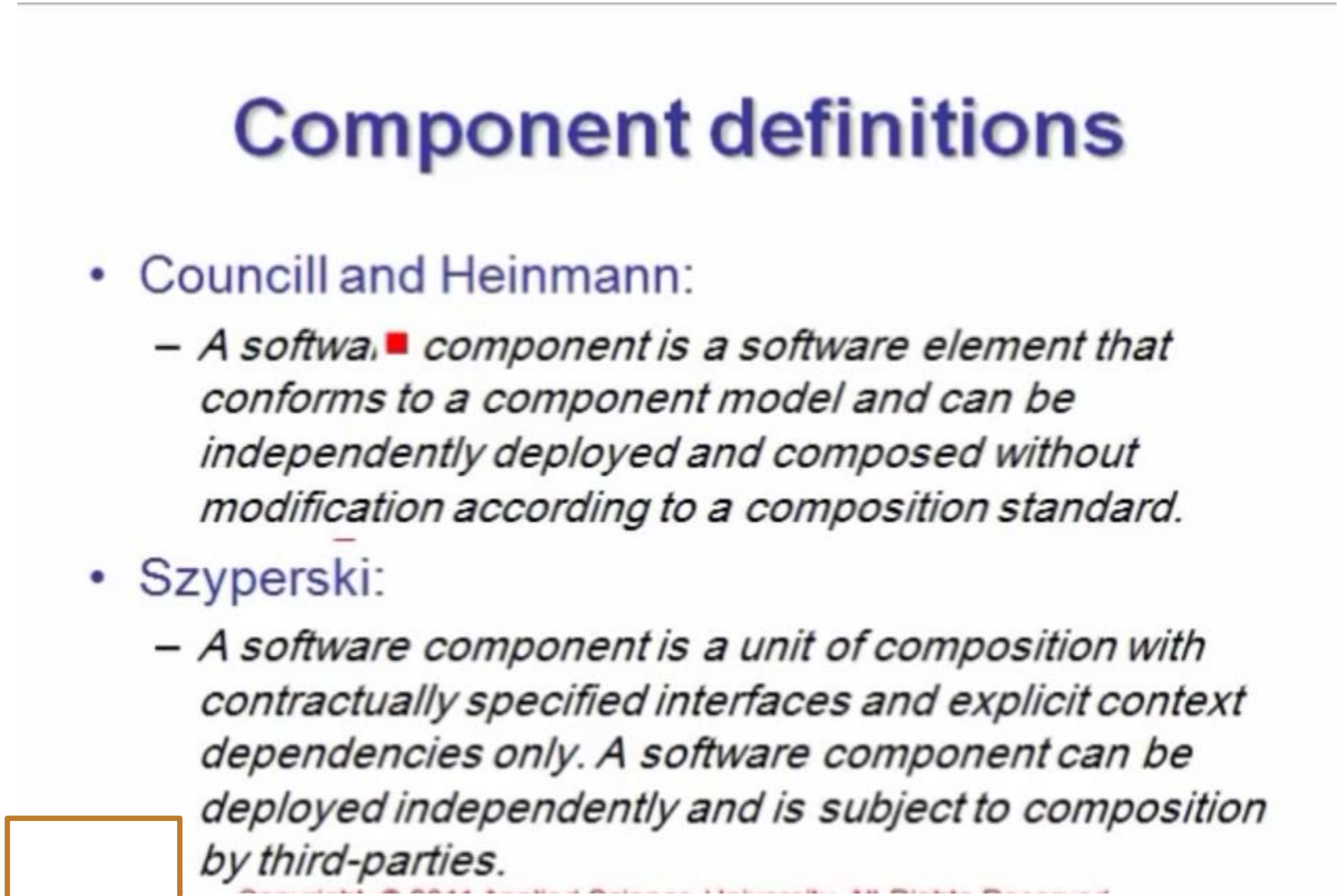
CBSE essentials

- 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 CBSE.



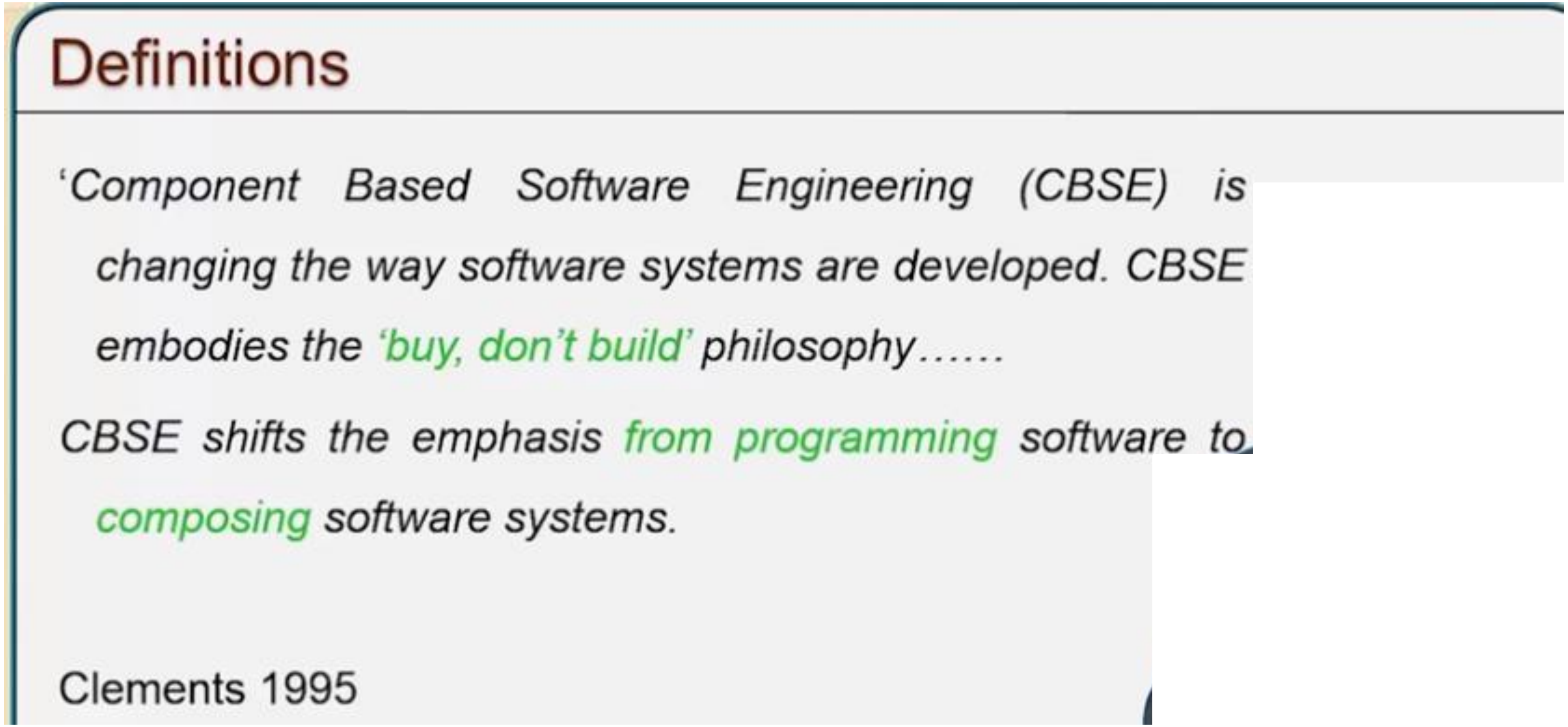
Components

- 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;



Component definitions

- Councill and Heinmann:
  - *A softwa[illegible] 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.*

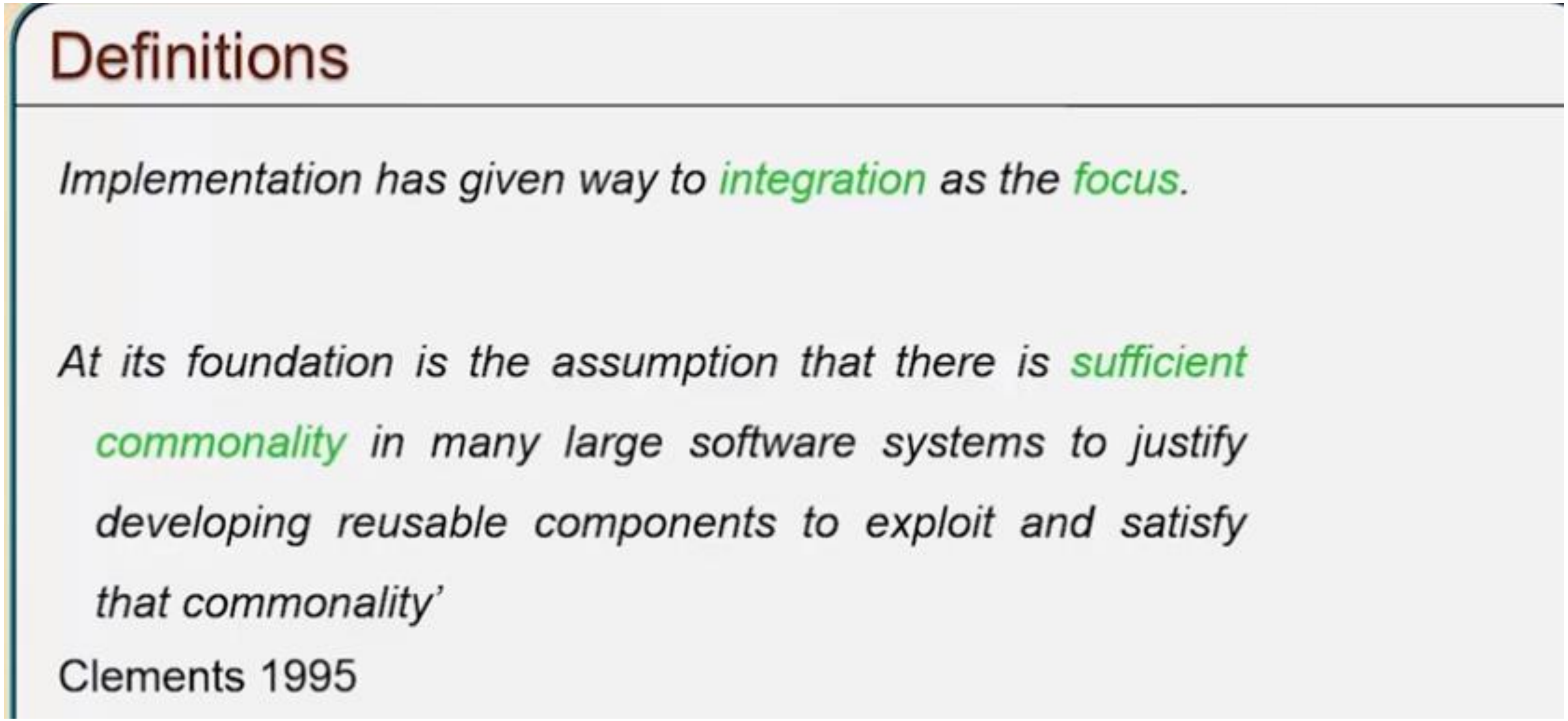


Definitions

'Component Based Software Engineering (CBSE) is changing the way software systems are developed. CBSE embodies the 'buy, don't build' philosophy……

CBSE shifts the emphasis from programming software to composing software systems.

Clements 1995

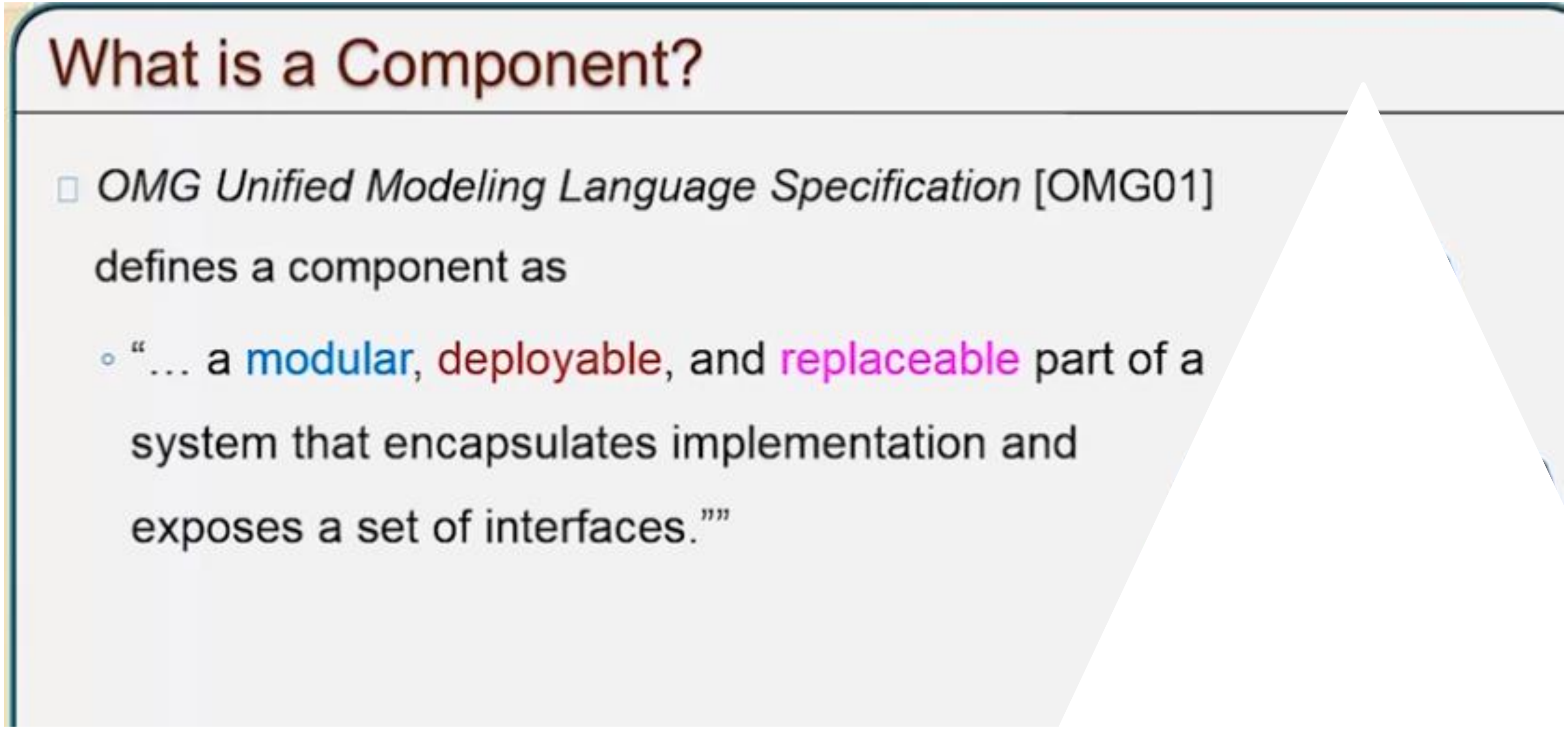


Definitions

Implementation has given way to integration as the focus.

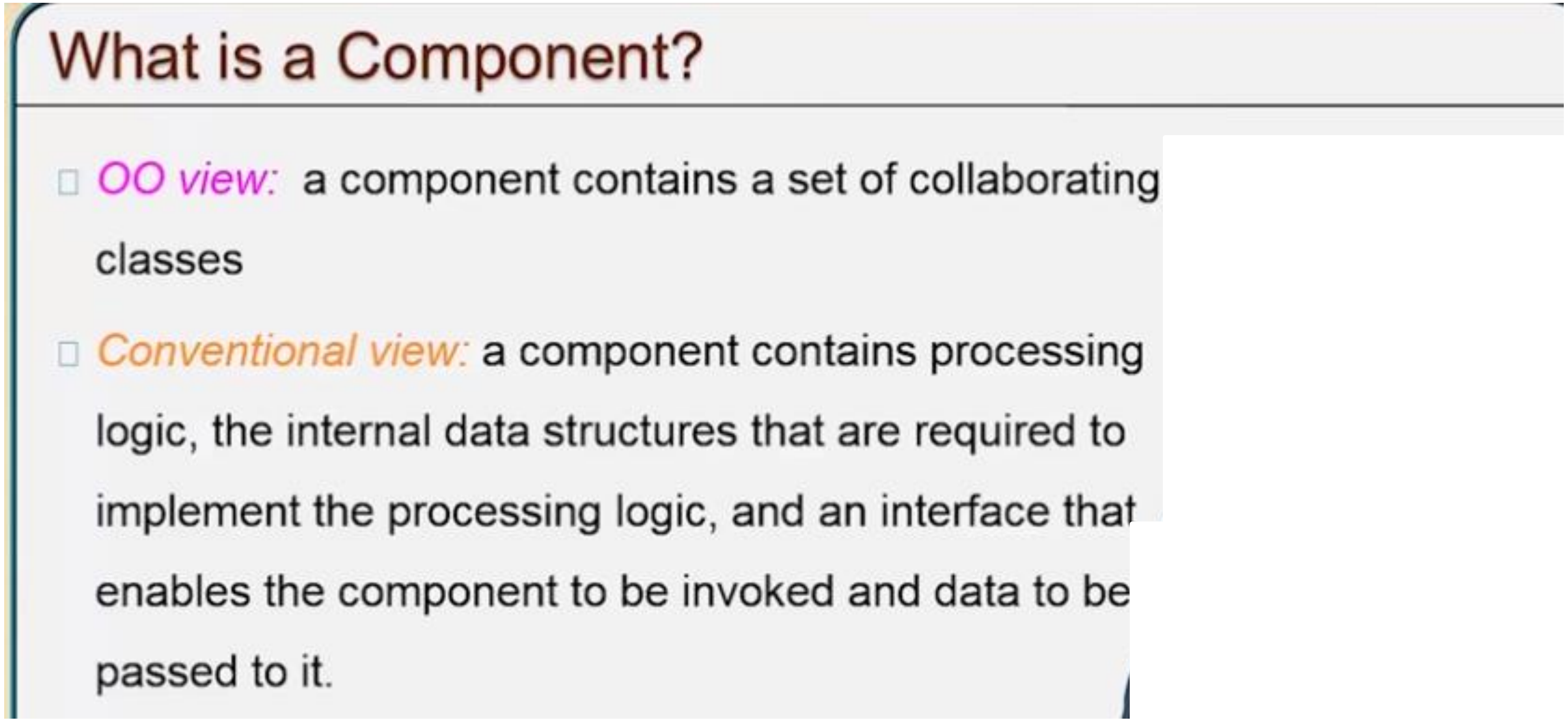
At its foundation is the assumption that there is sufficient commonality in many large software systems to justify developing reusable components to exploit and satisfy that commonality'

Clements 1995



What is a Component?

- *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.""

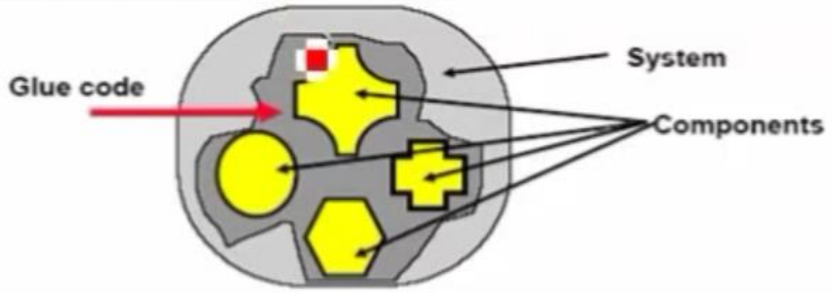


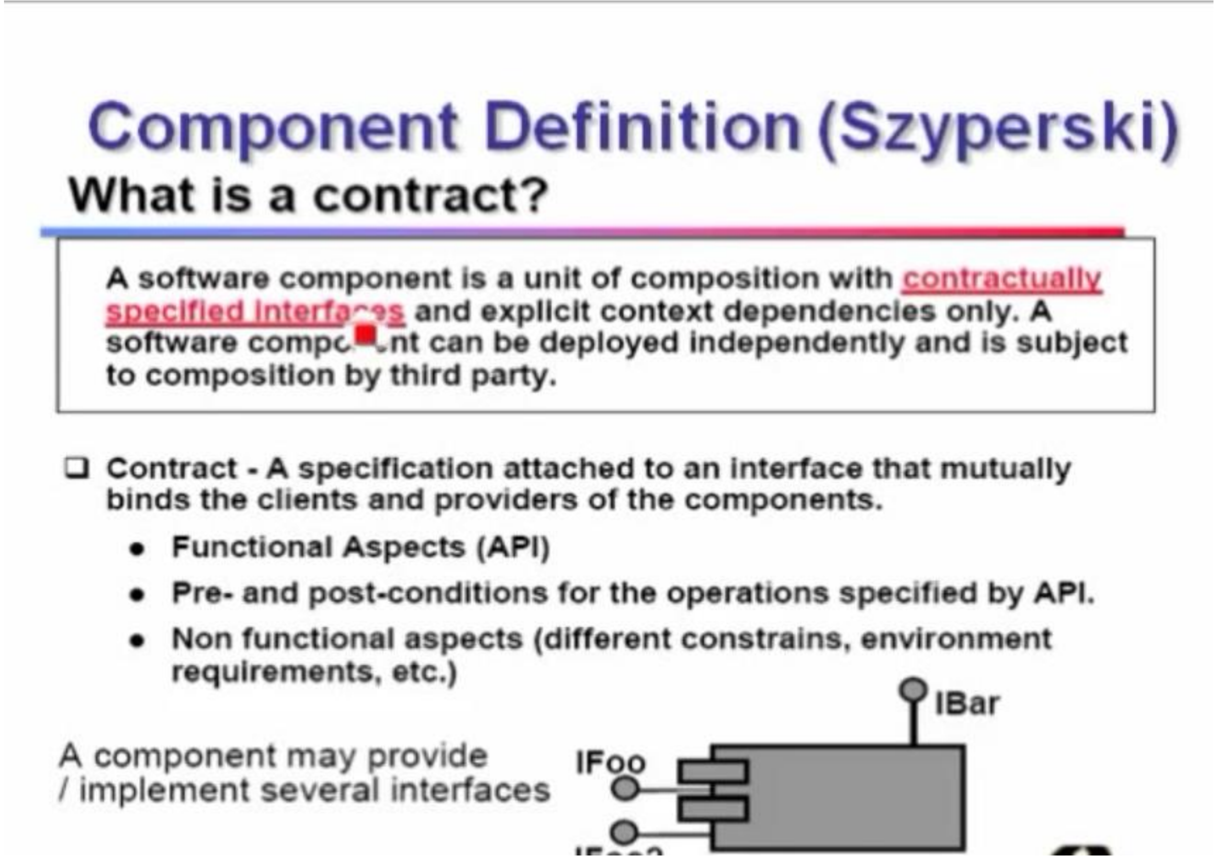
What is a Component?

- *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.

Component Definition (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 party.





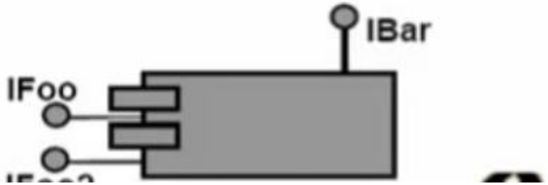
Component Definition (Szyperski)

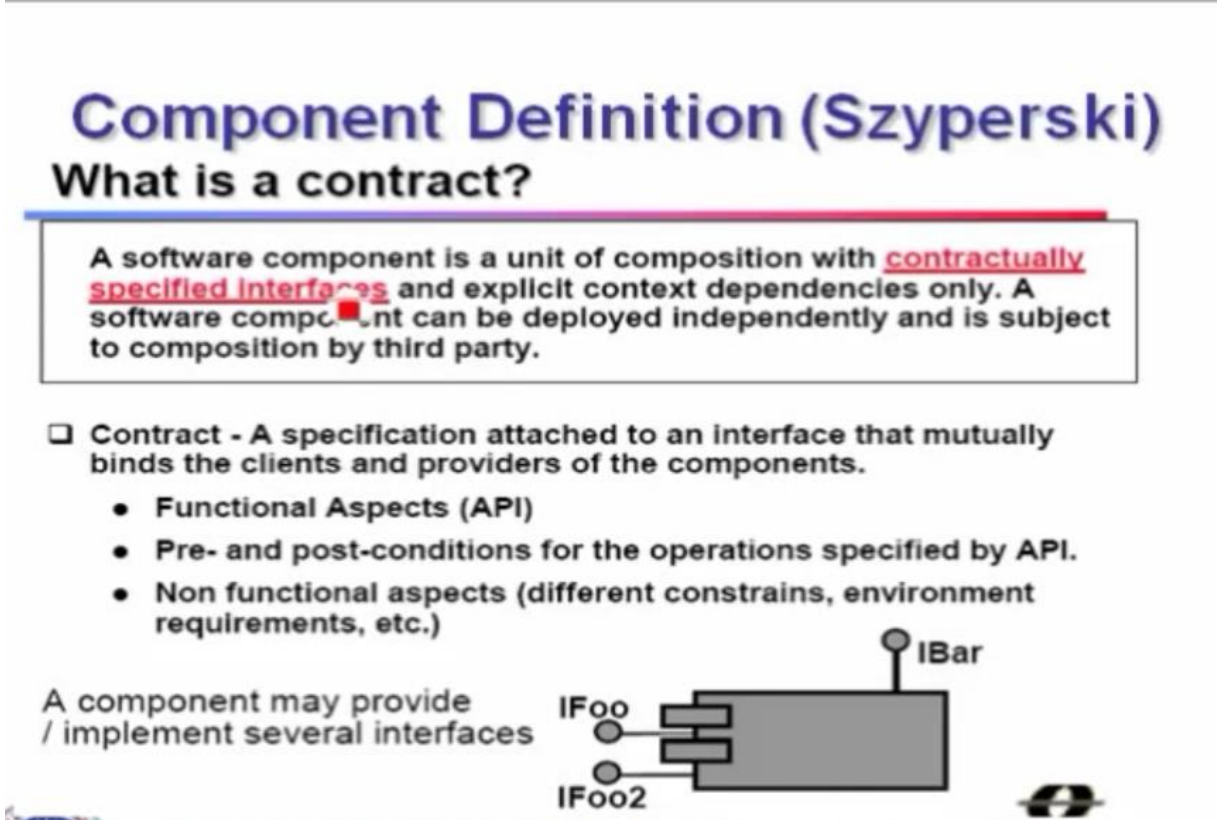
What is a contract?

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 party.

- Contract - A specification attached to an interface that mutually binds the clients and providers of the components.
  - Functional Aspects (API)
  - Pre- and post-conditions for the operations specified by API.
  - Non functional aspects (different constrains, environment requirements, etc.)

A component may provide / implement several interfaces





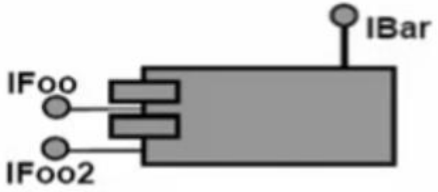
Component Definition (Szyperski)

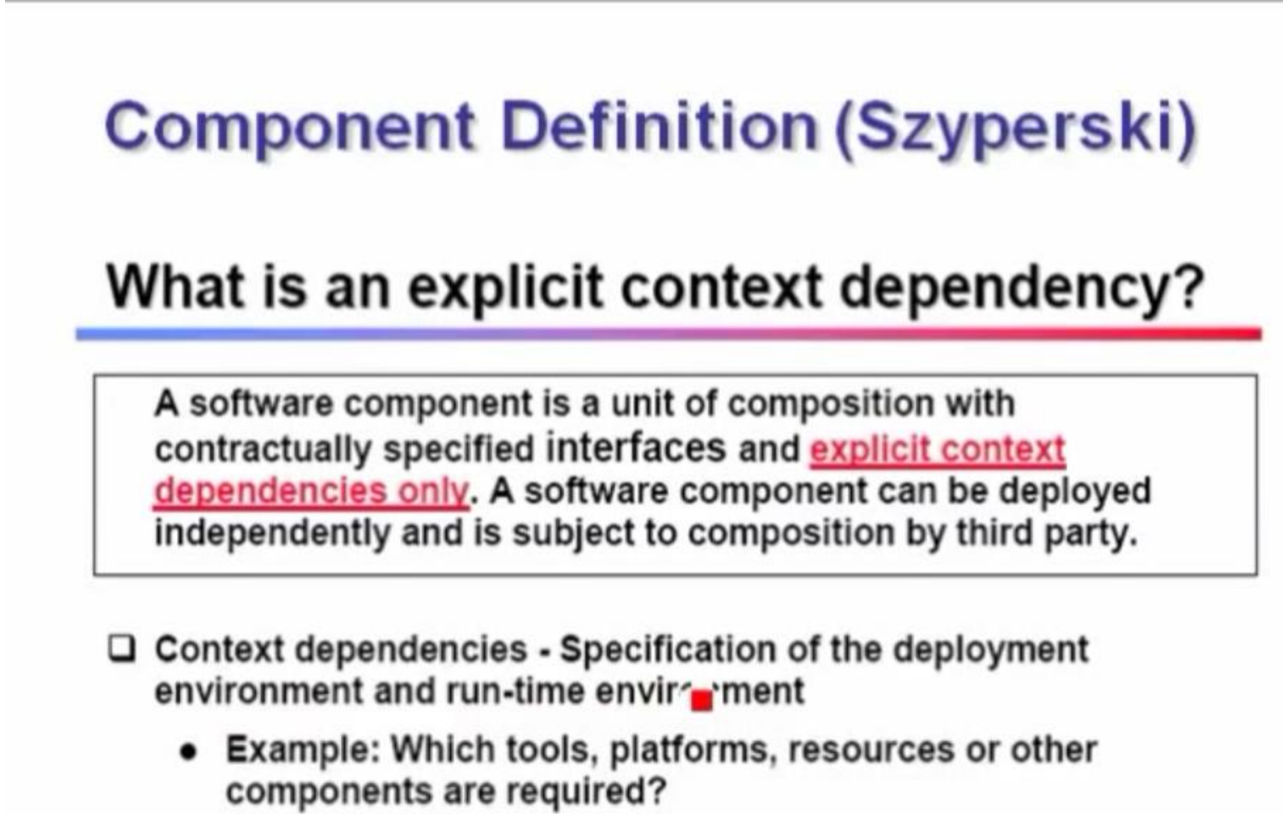
What is a contract?

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 party.

- ❑ Contract - A specification attached to an interface that mutually binds the clients and providers of the components.
  - Functional Aspects (API)
  - Pre- and post-conditions for the operations specified by API.
  - Non functional aspects (different constrains, environment requirements, etc.)

A component may provide / implement several interfaces



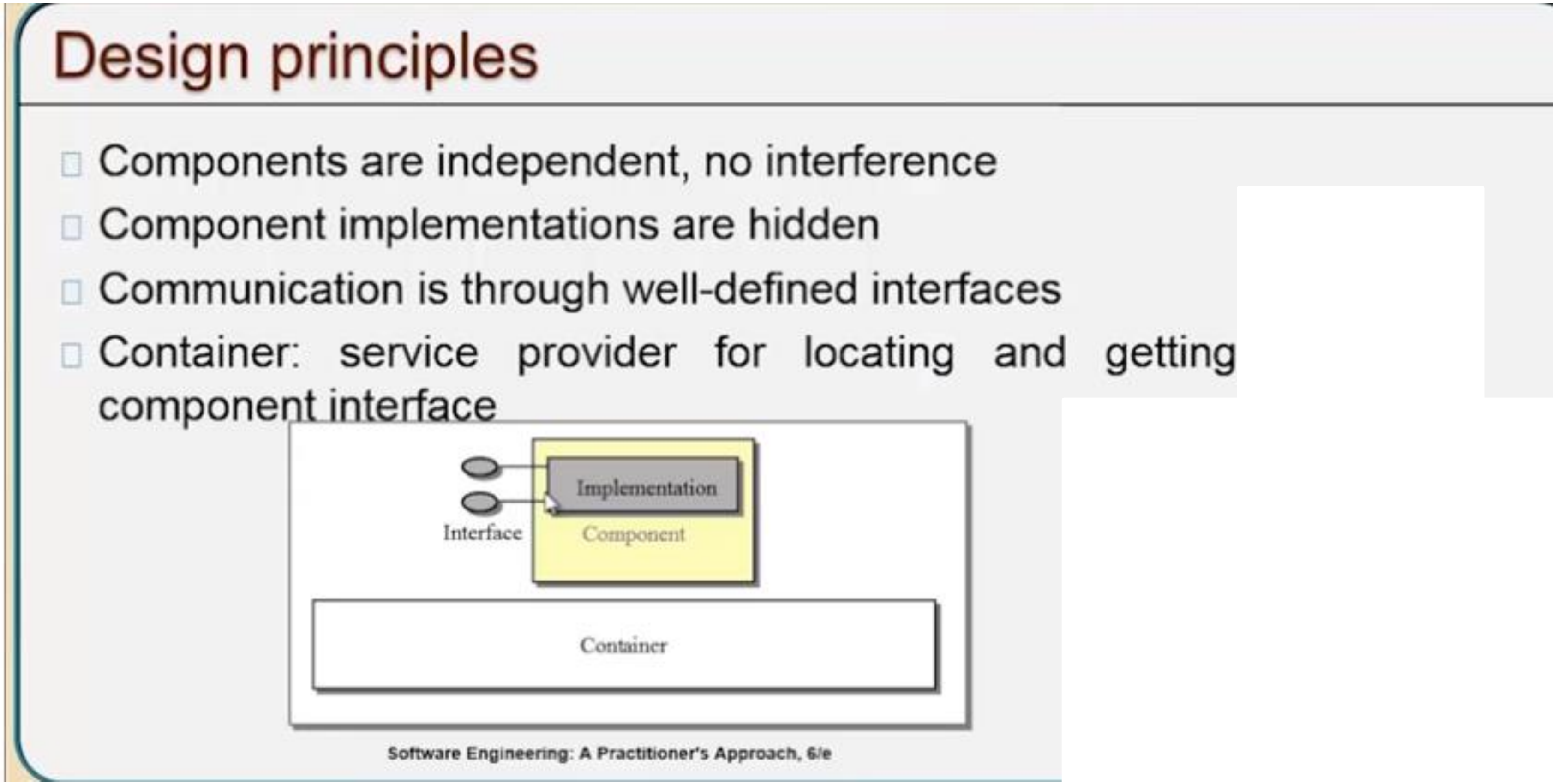


Component Definition (Szyperski)

What is an explicit context dependency?

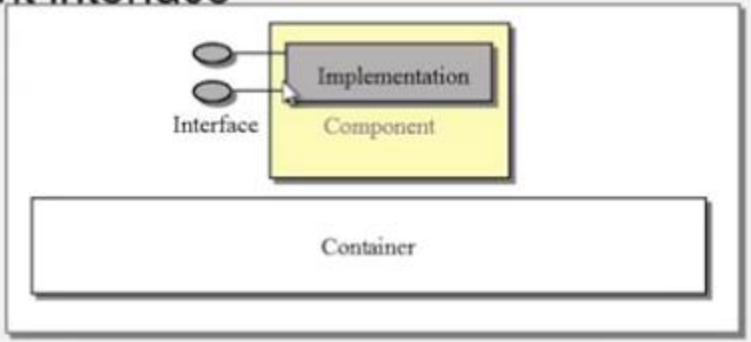
A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third party.

- ❑ Context dependencies - Specification of the deployment environment and run-time environment
  - Example: Which tools, platforms, resources or other components are required?

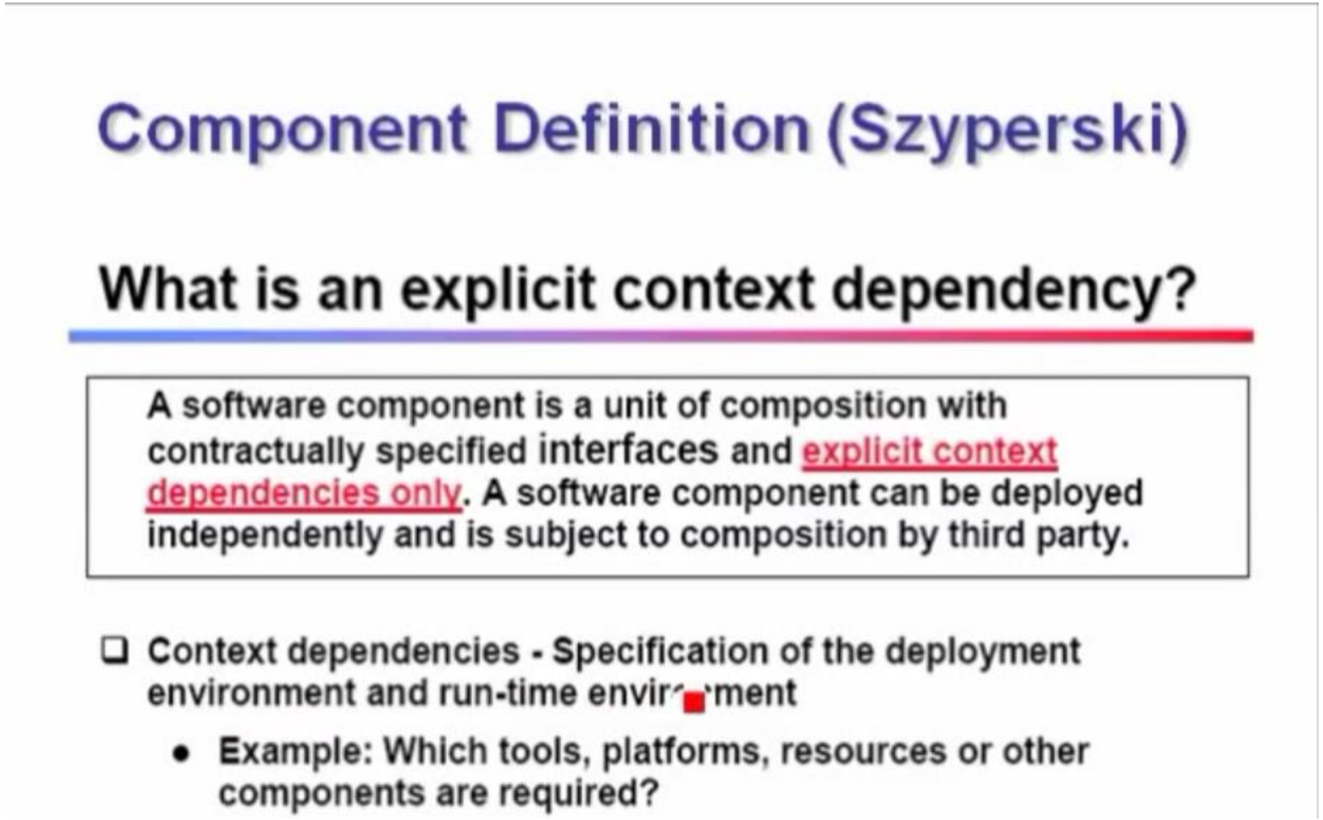


Design principles

- Components are independent, no interference
- Component implementations are hidden
- Communication is through well-defined interfaces
- Container: service provider for locating and getting component interface



Software Engineering: A Practitioner's Approach, 6/e



Component Definition (Szyperski)

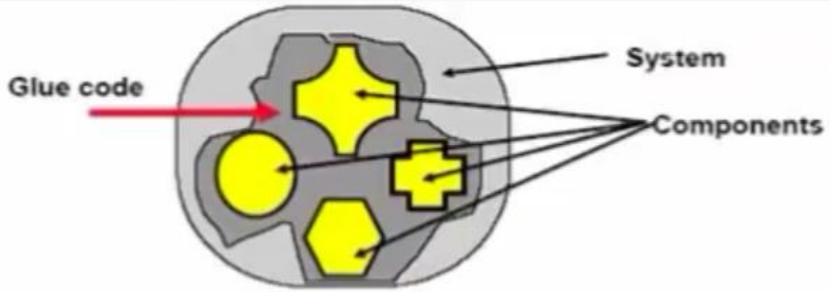
What is an explicit context dependency?

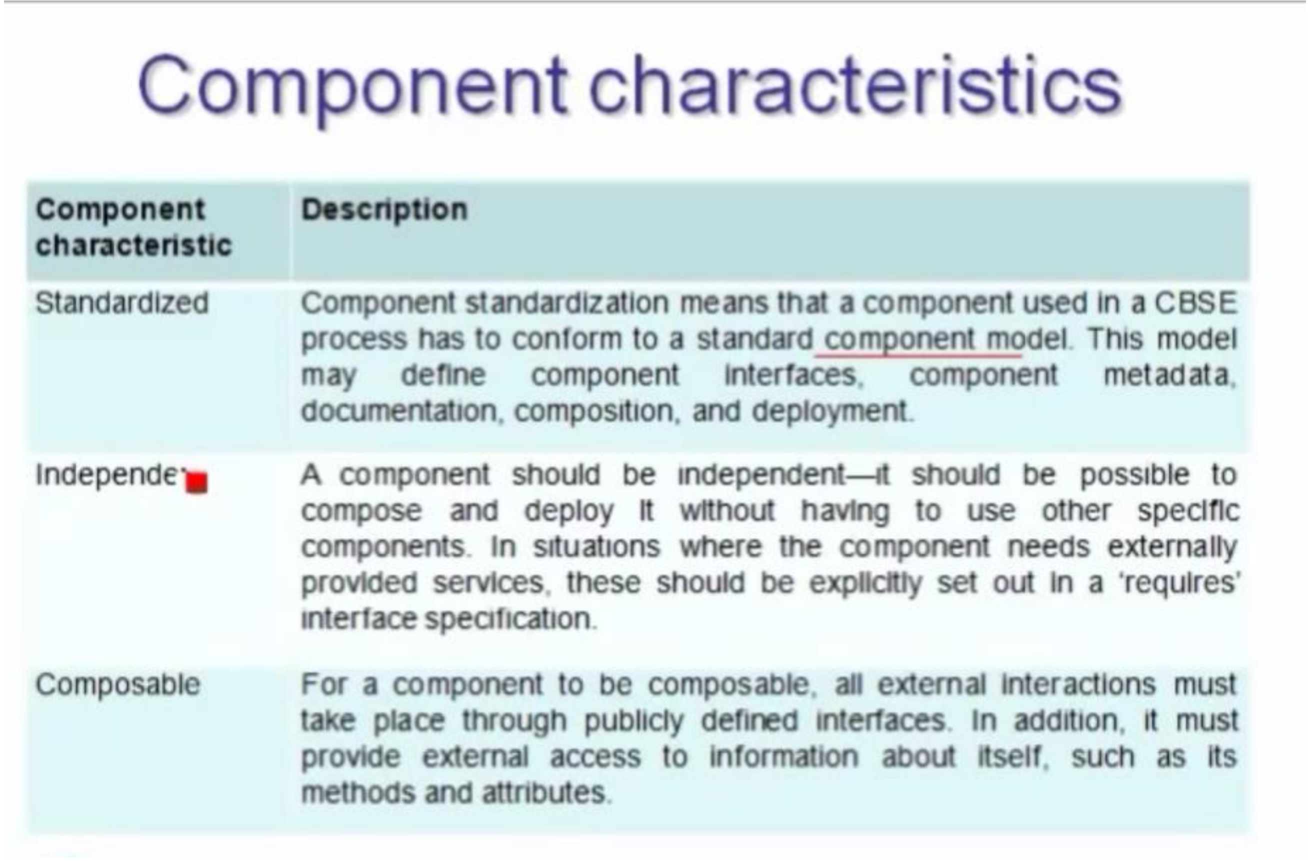
A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third party.

- Context dependencies - Specification of the deployment environment and run-time environment
  - Example: Which tools, platforms, resources or other components are required?

Component Definition (Szyperski)

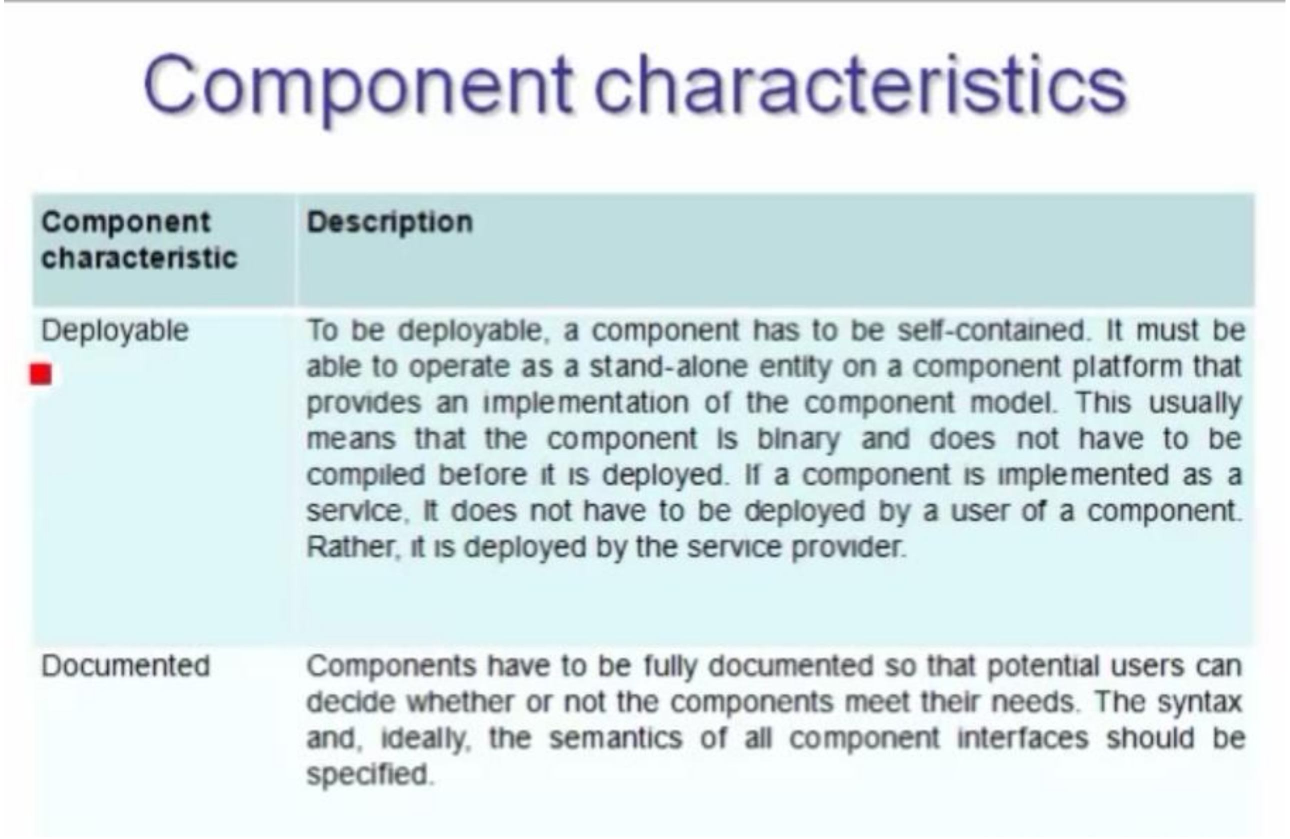
A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third party.





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.
Independe	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.