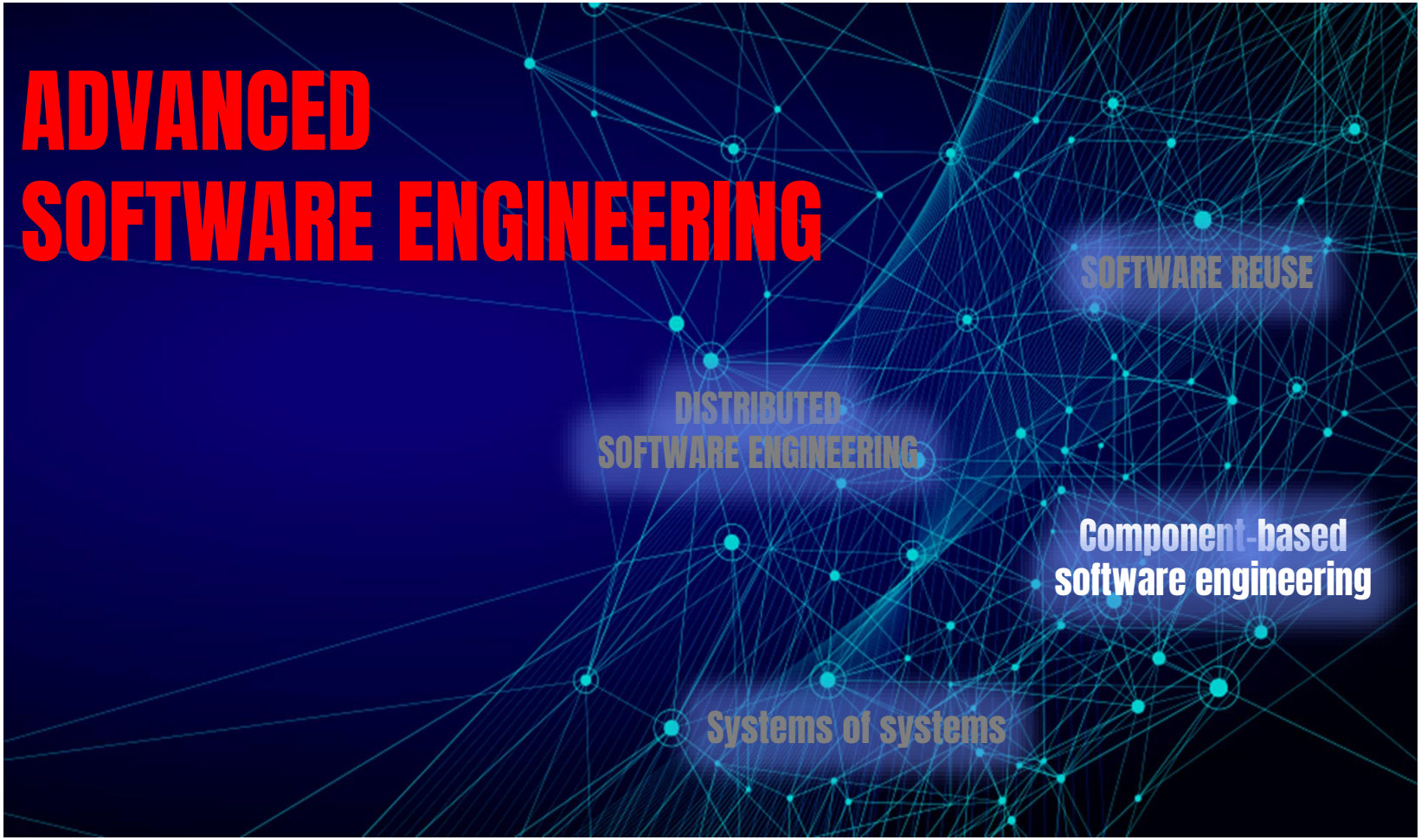
Class8: Component-Based Software Engineering (CBSE) Chapter 16

Adeeb Noor, PhD
IT Department
Faculty of Computing and Information Technology
King Abdulaziz University
Fall 2025

Setting the Stage



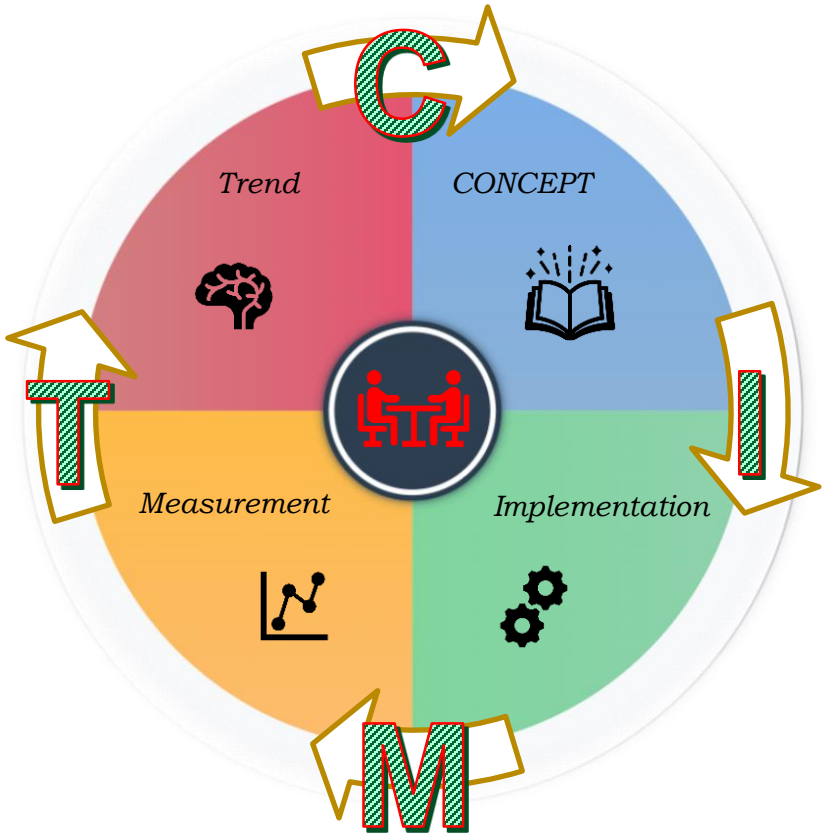
Our Roadmap

Concept: Definitions, principles, and theoretical foundations of CBSE

Implementation: Practical processes (development for/with reuse, composition)

Measurement: Metrics for success (cost, time) and risks (e.g., validation failures)

Trend: Modern applications (AI integration, federated learning)



The Core Idea

"*The key to building large, complex systems is to break them down into smaller, independent components that can be built and tested separately*." - Philippe Kruchten

Key Takeaways

- The Key CLO: Describe an approach to software reuse based on the composition of standardized and reusable components
- Components and component models (16.1): Components are independent software units with defined interfaces
- CBSE Processes (16.2): Development with and for reuse, along with supporting processes
- Component Composition (16.3): Assembling components, often requiring glue code to address interface compatibility
- CBSE is important in the AI era for reusing and adapting models

The Big Why

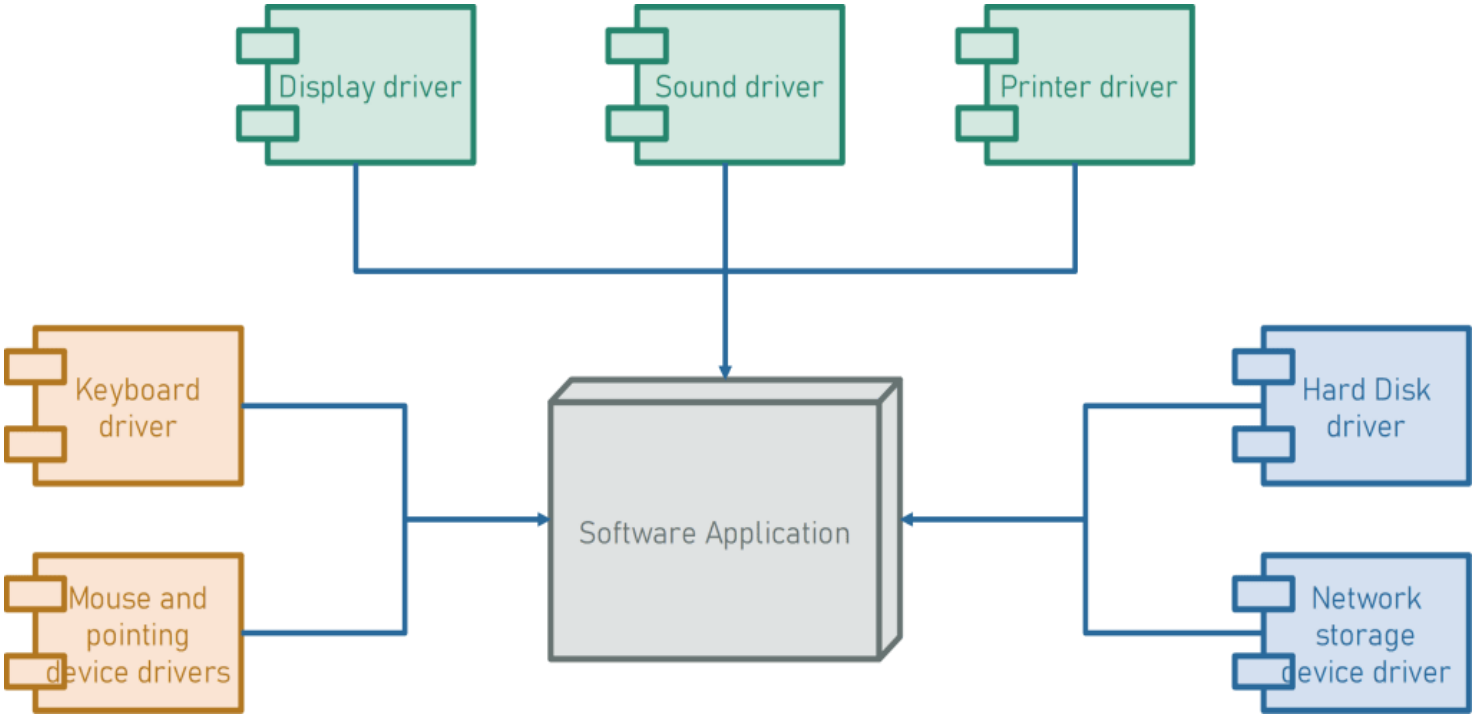
So, why is CBSE so important? Because it offers a more efficient and reliable way to build software. By reusing components, we can reduce development time, improve quality, and make it easier to maintain and update our systems. But beware, CBSE requires careful planning and thorough validation to avoid unexpected issues.

A Cautionary Tale

The Ariane 5 rocket failure highlights the importance of thorough validation in CBSE, as reusing components can introduce unexpected issues if not properly checked

Defining CBSE

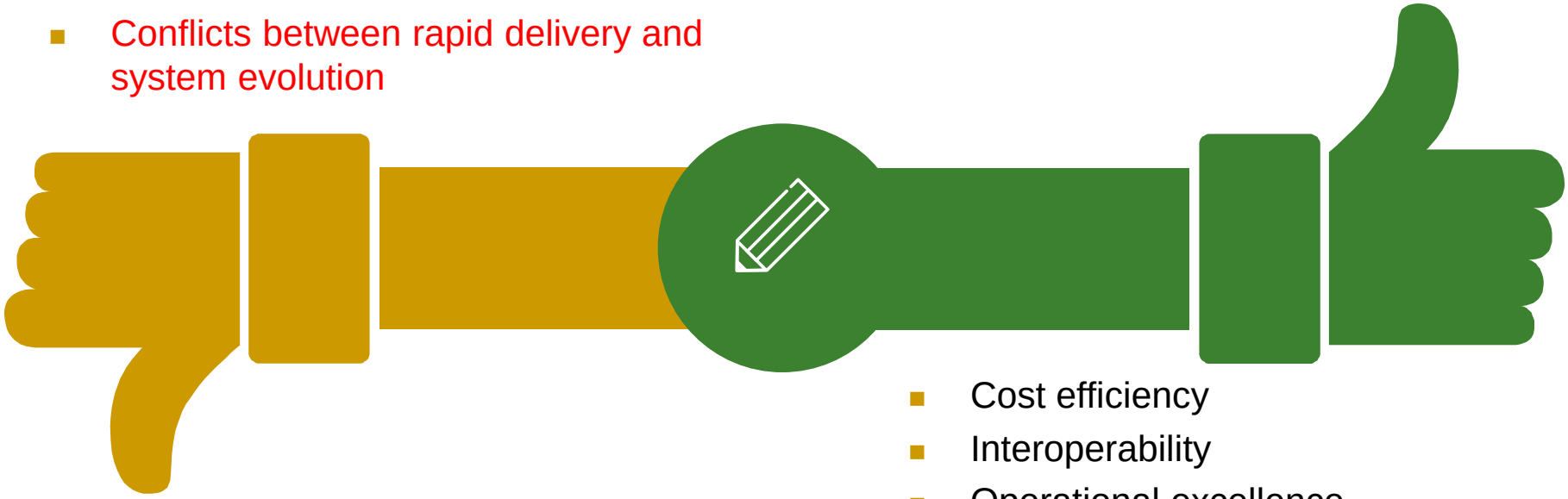
- **Component-based software engineering (CBSE)** is a software engineering methodology that prioritizes composing software applications through integration over implementing these from scratch every time



https://softwaredominos.com/home/software-design-development-articles/part-6-component-based-software-engineering-15-questions-and-answers-on-developing-business-architecture-for-the-future/

Weighing the Pros & Cons

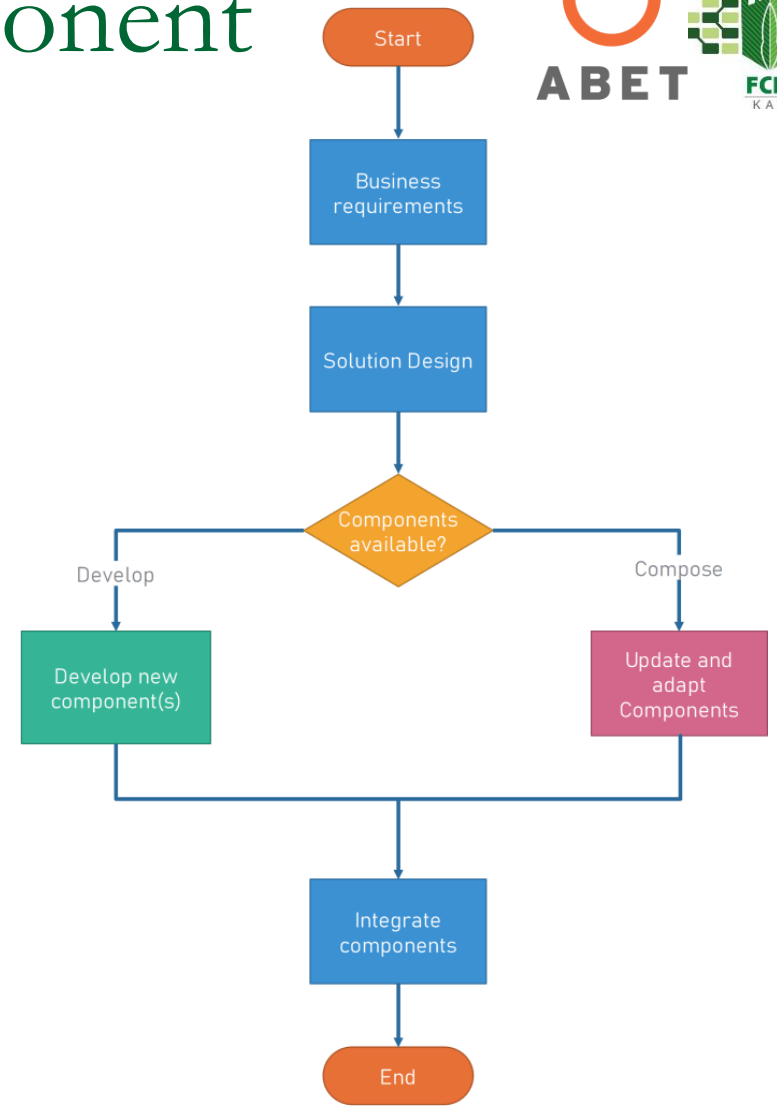
- Finding and integrating reusable components
- Conflicts between functional and non-functional requirements
- Conflicts between rapid delivery and system evolution



- Cost efficiency
- Interoperability
- Operational excellence
- Delivery of large software systems

The Life of a Component

"When *new* *business requirements* come in, solution architects design a solution that fits those requirements by either updating existing components that are suitable for providing that desired functionality or developing new components from scratch."



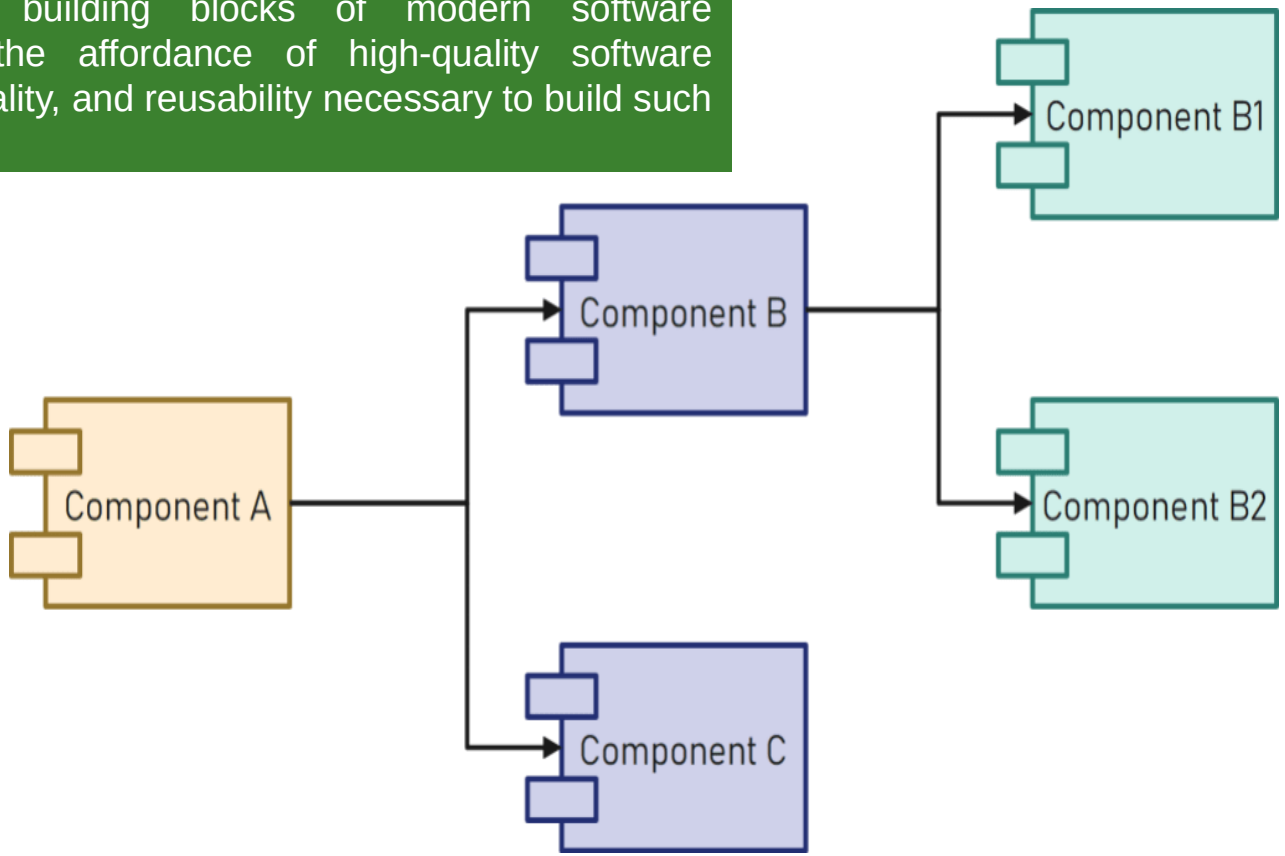
https://softwaredominos.com/home/software-design-development-articles/part-6-component-based-software-engineering-15-questions-and-answers-on-developing-business-architecture-for-the-future/

16.1 Components & Component Models

Components are the building blocks of modern software applications, providing the affordance of high-quality software engineering, rich functionality, and reusability necessary to build such applications

Component characteristics:

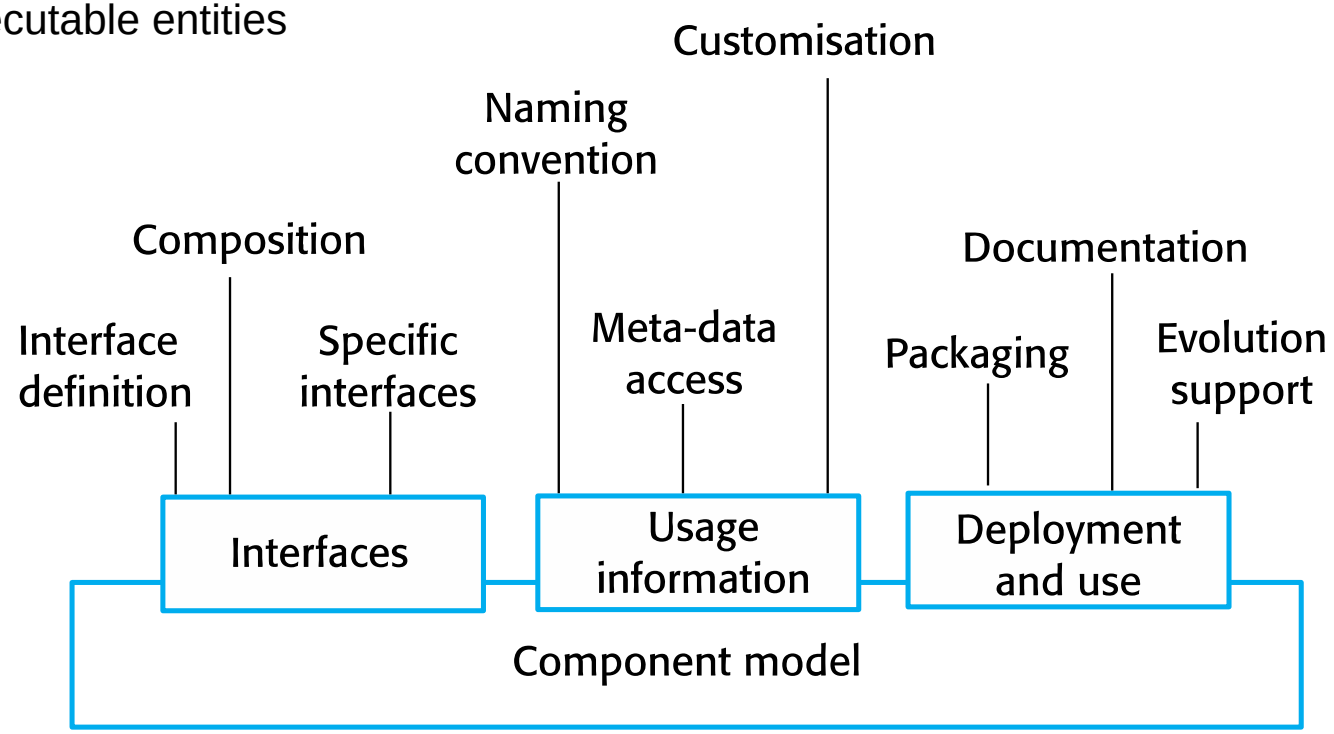
- Composable
- Deployable
- Documented
- Independent
- Standardized



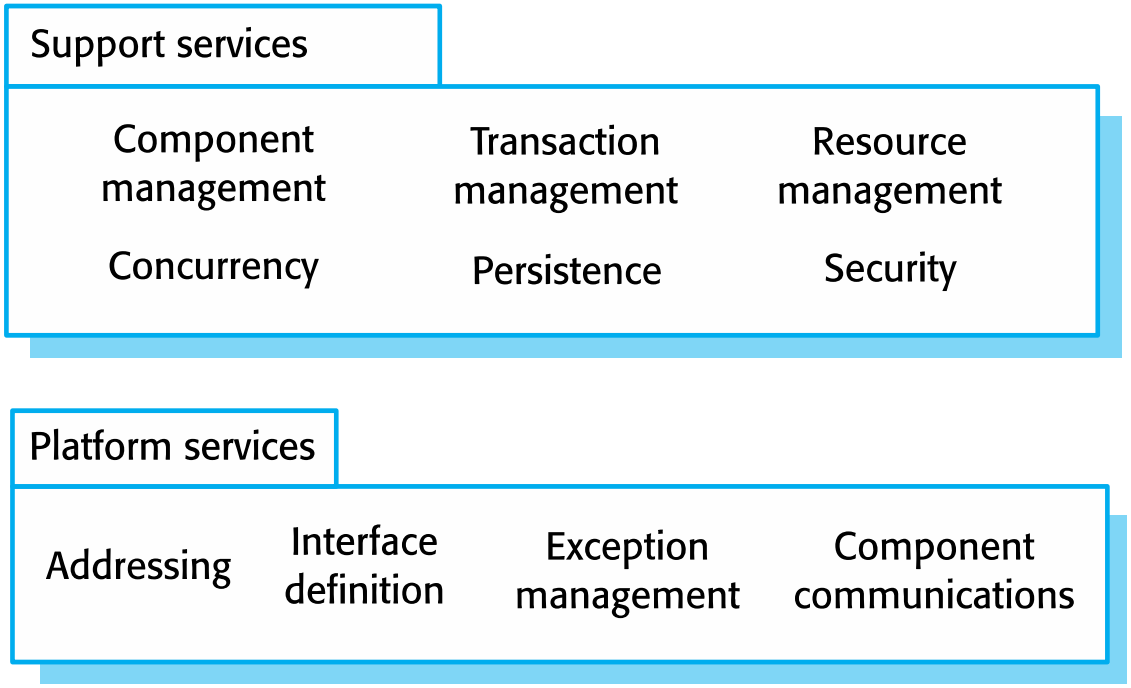
https://softwaredominos.com/home/software-design-development-articles/part-6-component-based-software-engineering-15-questions-and-answers-on-developing-business-architecture-for-the-future/

Blueprints for Components

- **Interfaces**: How interfaces should be defined, including operation names, parameters, and exceptions
- **Usage**: Unique names or handles for components, necessary for remote access and distribution
- **Deployment**: How components should be packaged for deployment as independent, executable entities



The Glue That Holds It Together

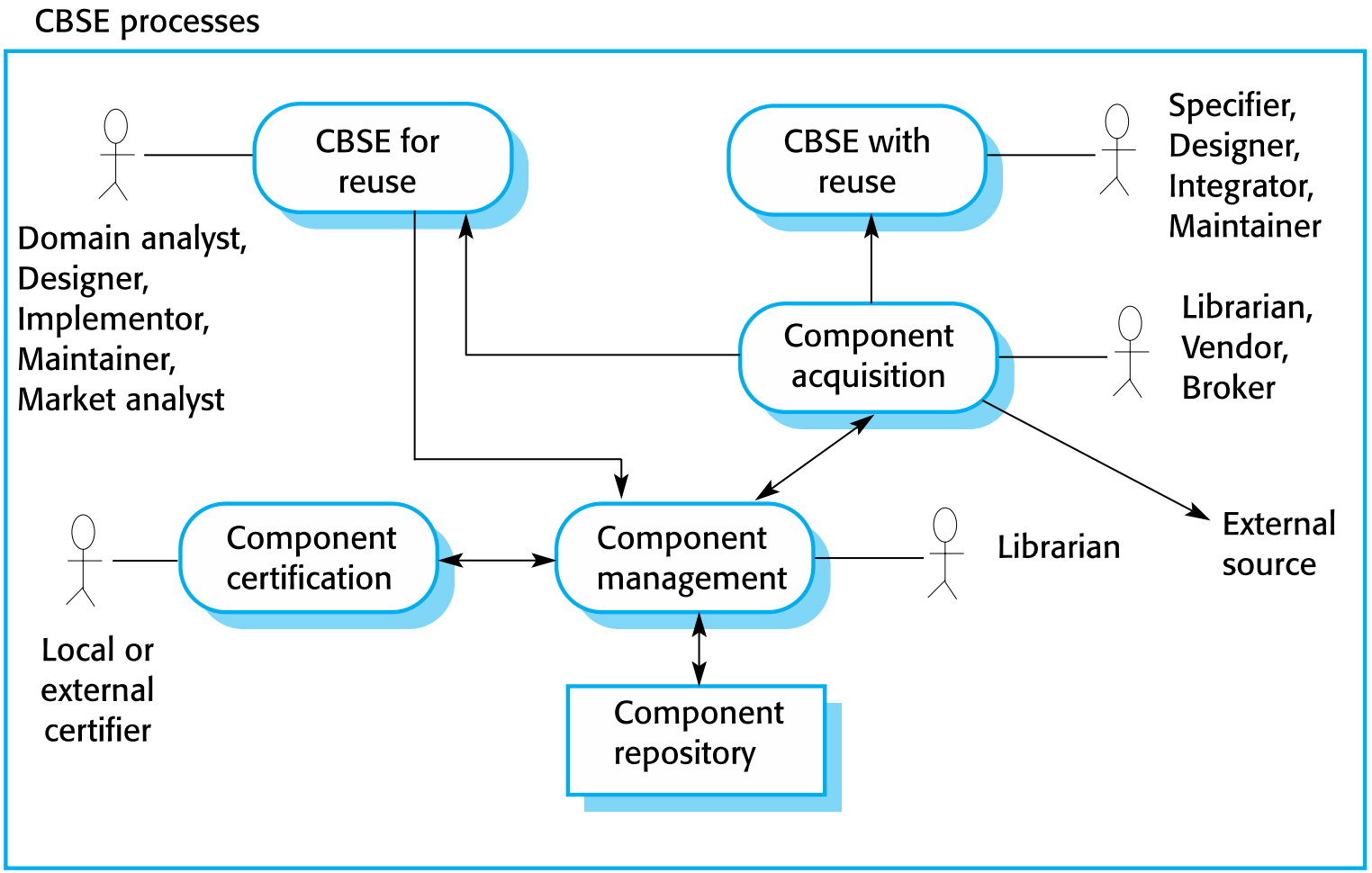


- Component models are the basis for middleware that provides support for executing components; component model implementations provide:
  - **Platform services** that allow components written according to the model to communicate
  - **Support services** that are application-independent services used by different components

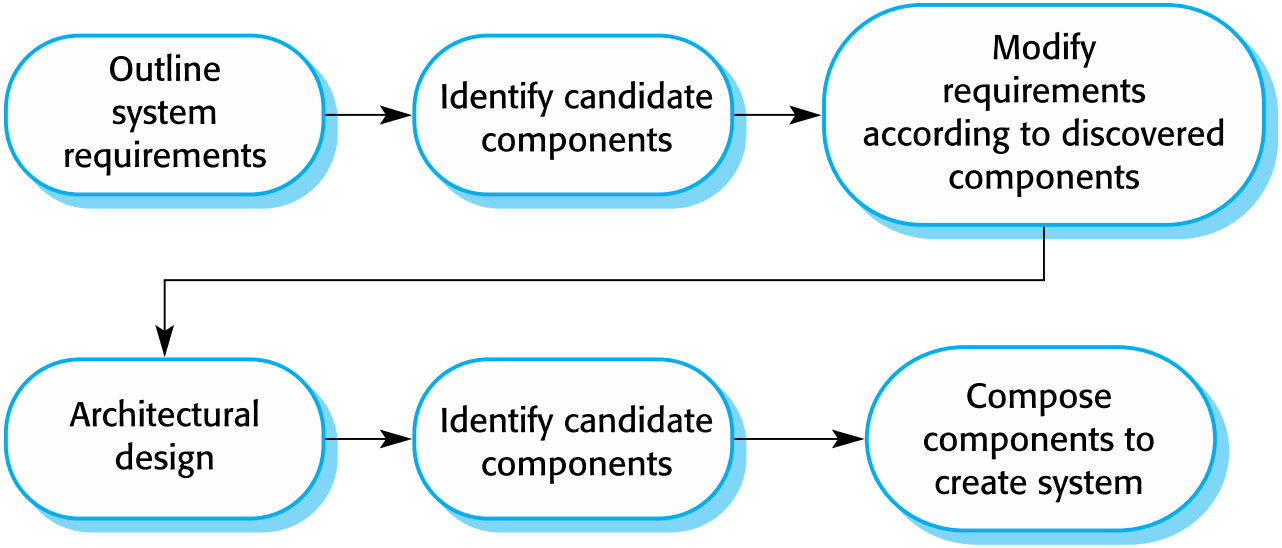
16.2 The CBSE Process

- **Development for reuse**: This process is concerned with developing components or services that will be reused in other applications
- **Development with reuse**: This process is the process of developing new applications using existing components and services
- **Component acquisition**: This process is about acquiring components for reuse or developing them into a reusable component
- **Component management**: This process is concerned with managing a company's reusable components, ensuring that they are properly catalogued, stored, and made available for reuse
- **Component certification**: This process is about checking a component and certifying that it meets its specification

Visualizing the Process



Reusing Components



The reuse methods:

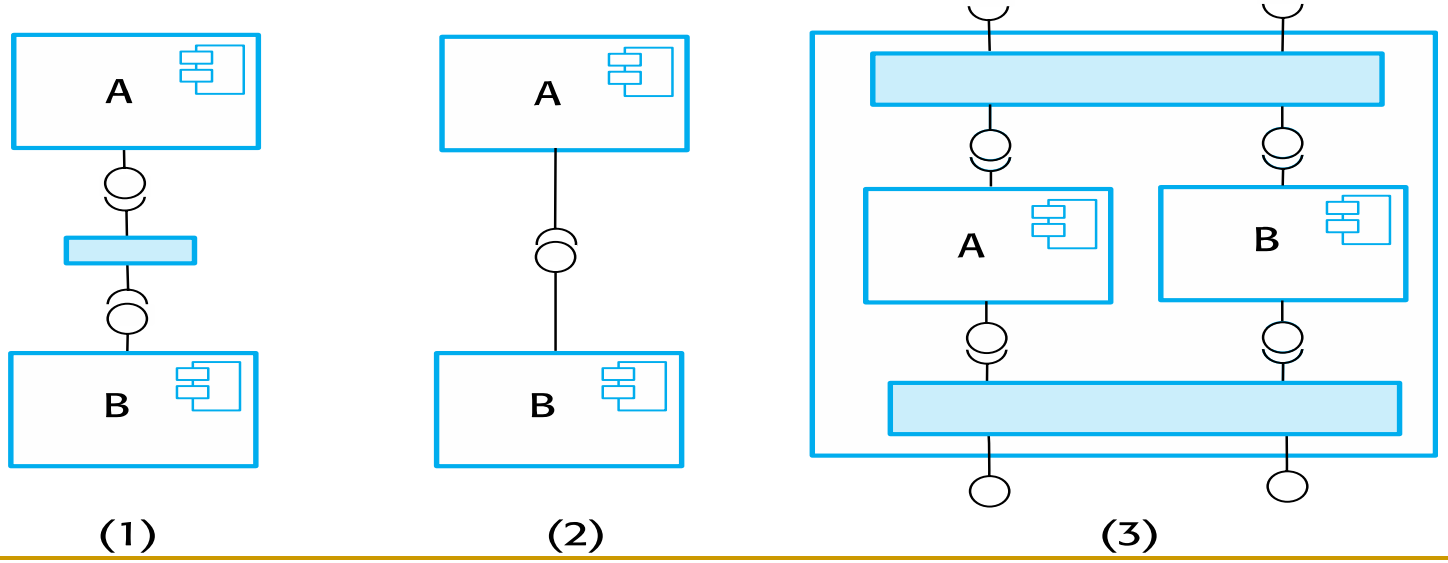
- Find and integrate reusable components.
- Make trade-offs between ideal requirements and available components.
- Develop outline requirements.
- Search for components and modify requirements based on available functionality.
- Search again for better components after revising requirements.
- Compose components to create the system.

Component identification issues:

- Trust: Ensure the component supplier is trustworthy to avoid security breaches.
- Requirements: Different components meet different requirements.
- Validation: Component specifications may be insufficient for testing, components may have unwanted functionality, develop test cases and a test harness to validate components, and check for malicious code or unneeded functionality.

16.3 Component Composition

- **Component composition** is the process of assembling components to create a system
  - (1) **Sequential**: Components executed in order. Example: Online shopping (product selection -> total calculation -> payment) - Glue code
  - (2) **Hierarchical**: One component uses another's services. Example: Music player (main control -> audio decoding, playlist)
  - (3) **Additive**: Combining interfaces to create a new component. Example: Spell check + grammar check = writing assistance



When Components Don't Agree

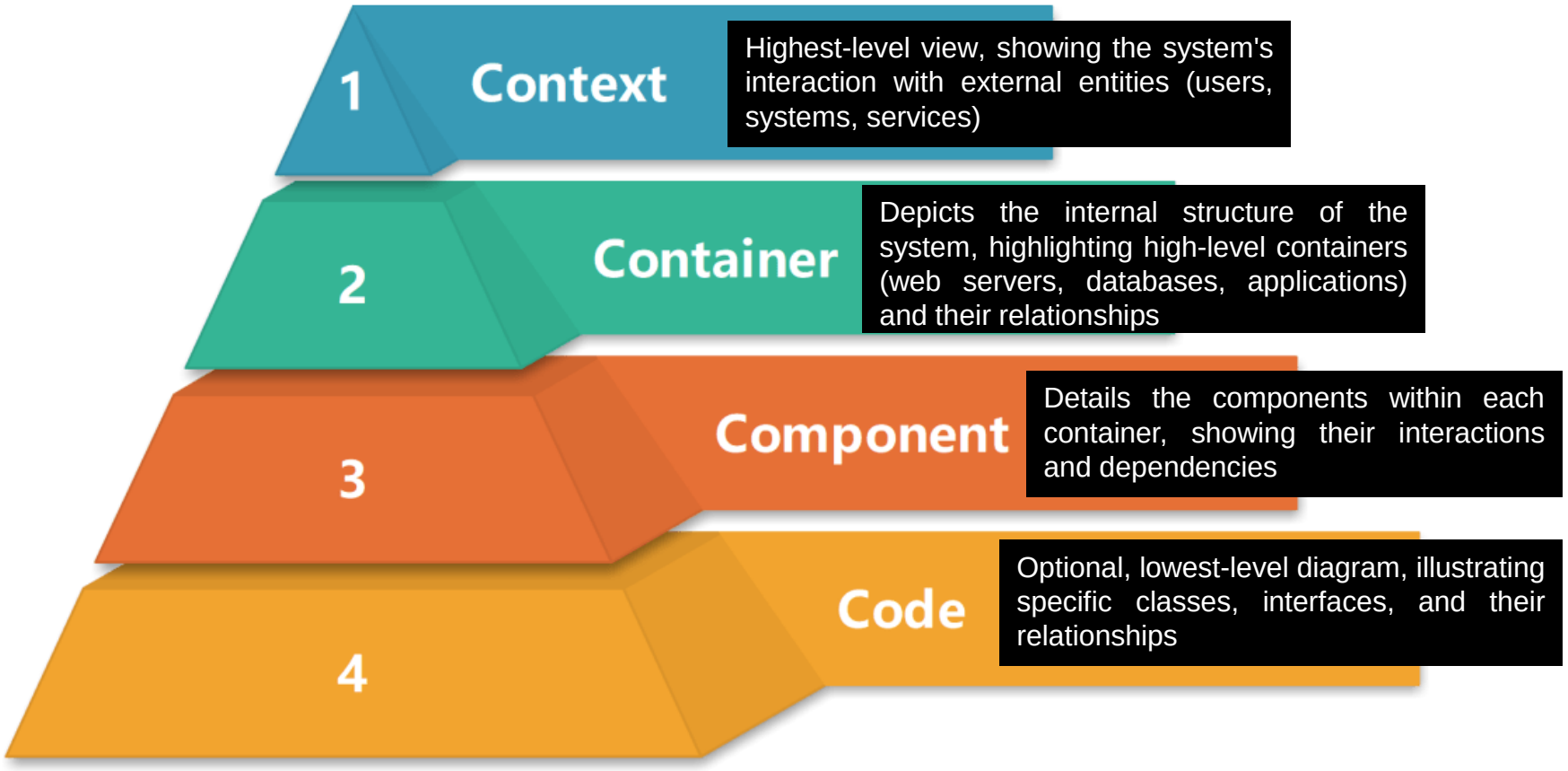
- **Parameter incompatibility** where operations have the same name but are of different types
- **Operation incompatibility** where the names of operations in the composed interfaces are different
- **Operation incompleteness** where the provides interface of one component is a subset of the requires interface of another
- **Adaptor components** address the problem of component incompatibility by reconciling the interfaces of the components that are composed

Bridging the Gap - Glue Code

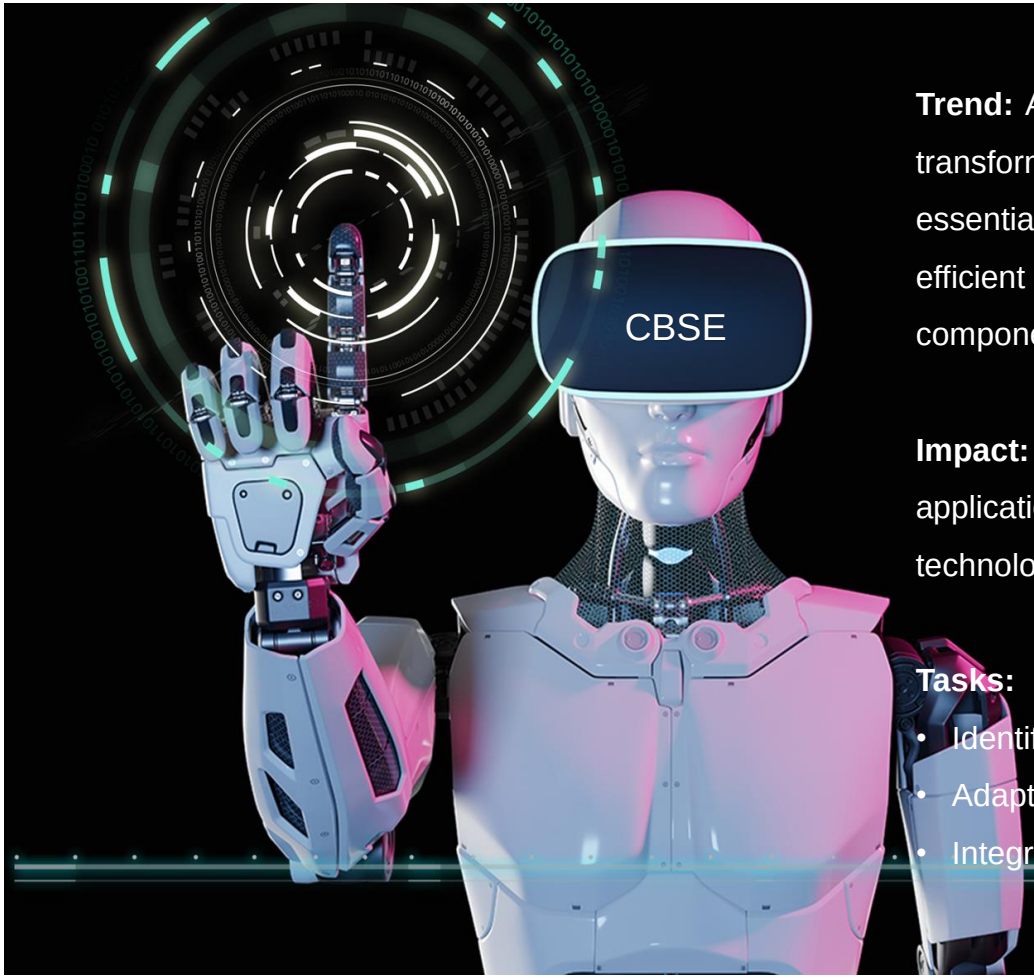
```java
// Component A: Outputs temperature data in XML format
class WeatherSensor {
    public String getData() {
        return "<temperature>25</temperature>";
    }
}

// Component B: Accepts temperature data in JSON format
class DisplayDashboard {
    public void showData(String jsonData) {
        System.out.println("Displaying: " + jsonData);
    }
}

// Glue Code: Converts XML to JSON and bridges the components
public class GlueCodeAdapter {
    public static void main(String[] args) {
        WeatherSensor sensor = new WeatherSensor();
        DisplayDashboard dashboard = new DisplayDashboard();

        // Extract temperature from XML
        String xmlData = sensor.getData();
        String temp = xmlData.split(">")[1].split("<")[0]; // Extracts "25"

        // Convert to JSON
        String jsonData = "{ \"temperature\": " + temp + " }";

        // Pass to DisplayDashboard
        dashboard.showData(jsonData); // Output: Displaying: { "temperature": 25 }
    }
}
```

- The WeatherSensor outputs XML, while the DisplayDashboard requires JSON
- The glue code (GlueCodeAdapter) extracts the temperature value from XML and reformats it into JSON
- This resolves parameter incompatibility (data format mismatch)

A Bird's-Eye View



CBSE and the AI Revolution



Trend: AI and Large Language Models (LLMs) are transforming software development. CBSE is essential for harnessing their power by enabling efficient reuse and adaptation of AI models and components

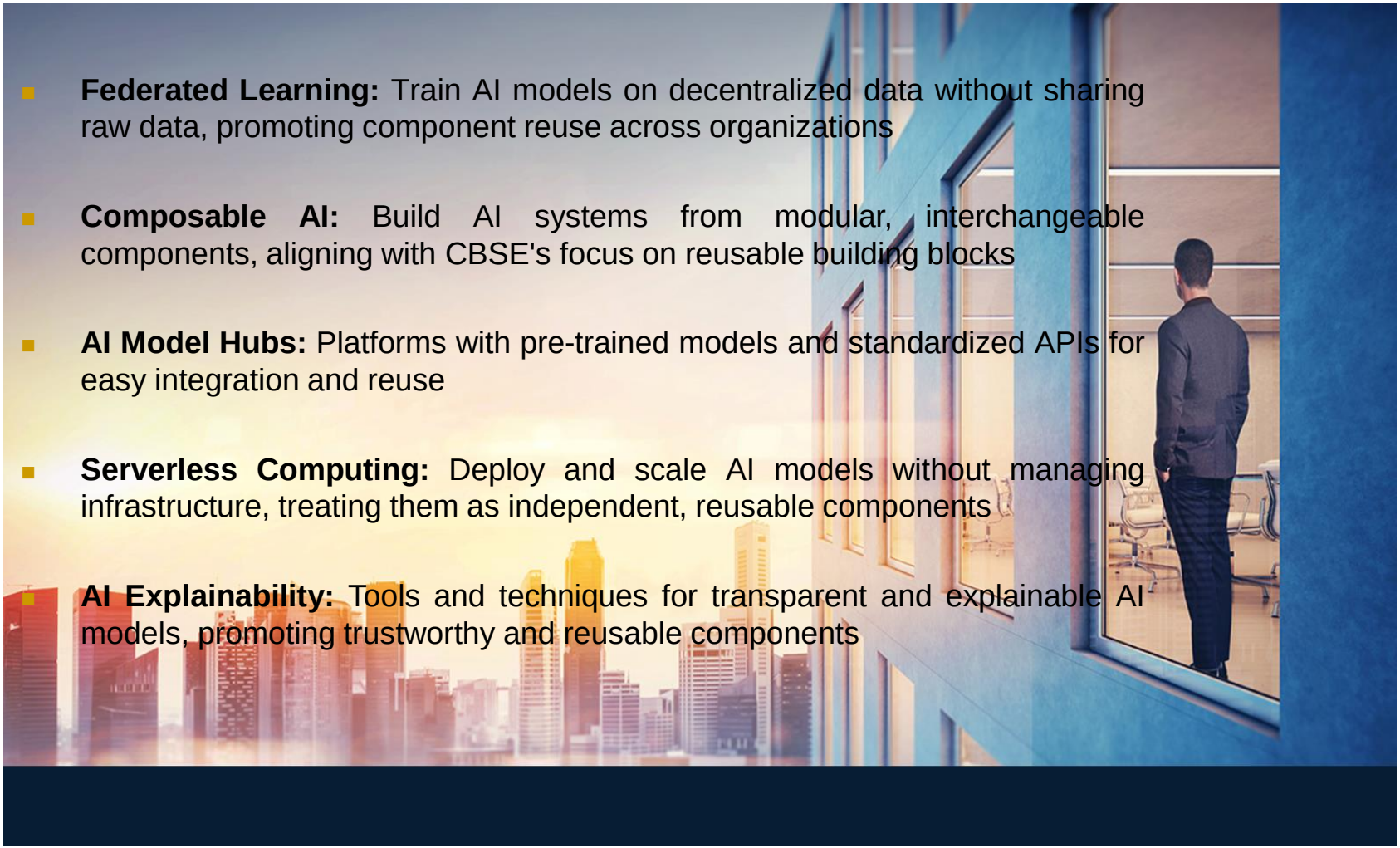
Impact: Accelerated development of AI applications, and Increased accessibility of AI technologies

Tasks:

- Identifying reusable AI models and components
- Adapting components to new application needs
- Integrating components into new AI applications

Ex; Meta Employ CBSE to Manage Components for Sentiment Analysis, Fake News

Keeping Up with Trends



- **Federated Learning:** Train AI models on decentralized data without sharing raw data, promoting component reuse across organizations
- **Composable AI:** Build AI systems from modular, interchangeable components, aligning with CBSE's focus on reusable building blocks
- **AI Model Hubs:** Platforms with pre-trained models and standardized APIs for easy integration and reuse
- **Serverless Computing:** Deploy and scale AI models without managing infrastructure, treating them as independent, reusable components
- **AI Explainability:** Tools and techniques for transparent and explainable AI models, promoting trustworthy and reusable components

Your Challenge

- Assignment 6 is designed to assess your mastery of EGTH (CLO): Describe an approach to software reuse based on the composition of standardized and reusable components
- Total Marks: 5
- Absolutely no late submissions will be accepted. All submissions are due by 8:00 PM on Saturday, March 1st
- You can use any LLM (ChatGPT, Claude, etc.), but you must follow the IT department's LLM ethics guidelines
- Collaboration allowed but submit individual work