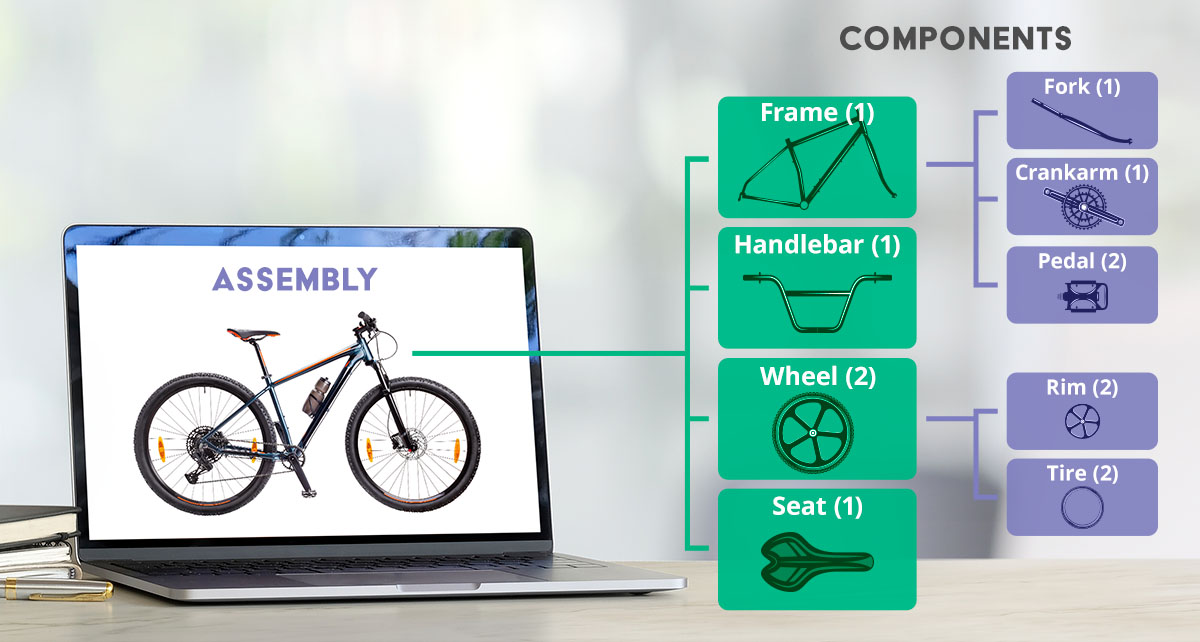
**COMPONENT LEVEL DESIGN**

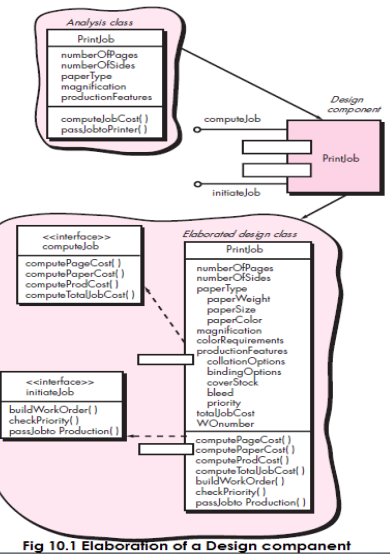
* Component level design is the definition and design of components and modules after the

architectural design phase.

* Component-level design defines the data structures, algorithms, interface characteristics, and communication mechanisms allocated to each component for the system development.
* A complete set of software components is defined during architectural design. But the internal data structures and processing details of each component are not represented at a level of abstraction that is close to code.
* Component-level design defines the data structures, algorithms, interface characteristics, and communication mechanisms allocated to each component.



* A component is a modular building block for computer software. More formally, the OMG Unified Modeling Language Specification defines a component as “a modular, deployable, and replaceable part of a system that encapsulates implementation and exposes a set of interfaces.”
* **The true meaning of the term component will differ depending on the point of view of the software engineer who uses it.**
* **An Object-Oriented View :** An Object-Oriented View: In the context of object-oriented software engineering, a component contains a set of collaborating classes. Each class within a component has been fully elaborated to include all attributes and operations that are relevant to its implementation. As part of the design elaboration, all interfaces that enable the classes to communicate and collaborate with other design classes must also be defined.

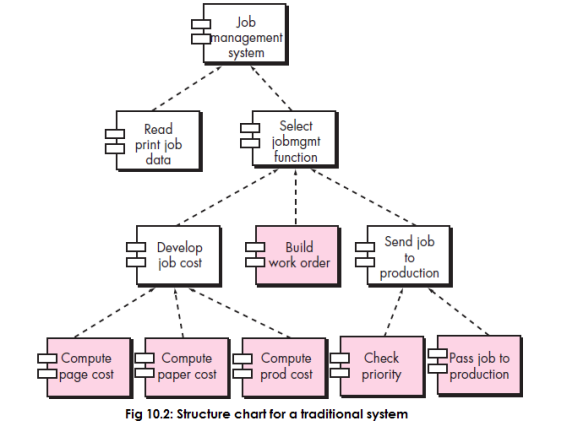


**The Traditional View:** In the context of traditional software engineering, a component is a functional element of a program that incorporates processing logic, the internal data structures that are required to implement the processing logic, and an interface that enables the component to be invoked and data to be passed to it.

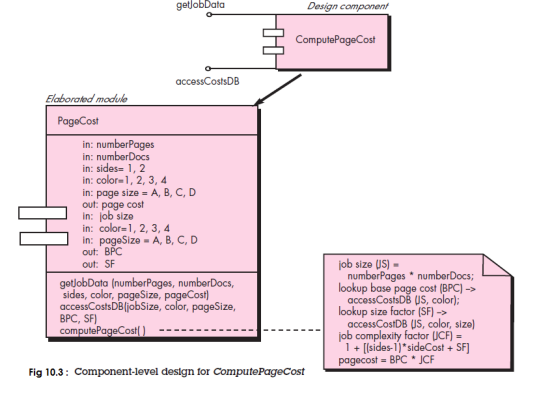
A traditional component, also called a module, resides within the software architecture and serves one of three important roles: (1) a control component that coordinates the invocation of all other problem domain components, (2) a problem domain component that implements a complete or partial function that is required by the customer, or (3) an infrastructure component that is responsible for functions that support the processing required in the problem domain.

Like object-oriented components, traditional software components are derived from the analysis model. To achieve effective modularity, design concepts like functional independence are applied as components are elaborated.

During component-level design, each module in Figure 10.2 is elaborated. The module interface is defined explicitly. That is, each data or control object that flows across the interface is represented. The data structures that are used internal to the module are defined. The algorithm that allows the module to accomplish its intended function is designed using the stepwise refinement approach. The behavior of the module is sometimes represented using a state diagram. Figure 10.3 represents the component-level design using a modified UML notation.



**A Process-Related View:** The object-oriented and traditional views of component-level design assume that the component is being designed from scratch. That is, you have to create a new component based on specifications derived from the requirements model. Over the past two decades, the software engineering community has emphasized the need to build systems that make use of existing software components or design patterns. In essence, a catalog of proven design or code-level components is made available to you as design work proceeds. As the software architecture is developed, you choose components or design patterns from the catalog and use them to populate the architecture. Because these components have been created with reusability in mind, a complete description of their interface, the function(s) they perform, and the communication and collaboration they require are all available.

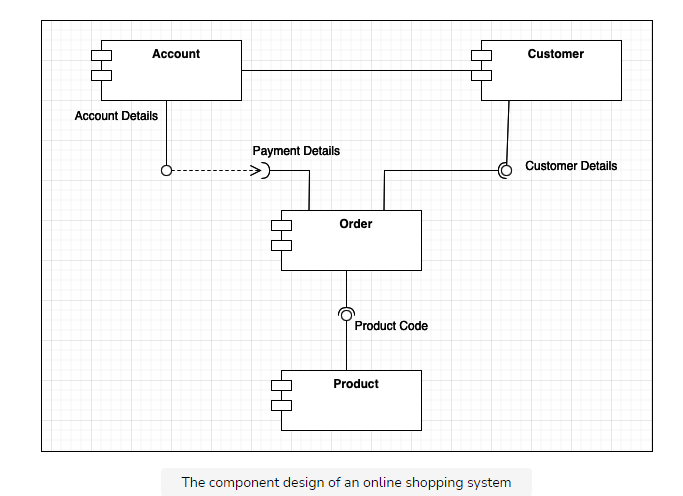


**Example**

In our example, we'll consider a component diagram of an online shopping system:

* It will have a Customer, Customer Account, Product, and Order.
* Every object is an individual component but is related to each other at the same time
* The customer will go on the online site and search the product.
* By selecting any specific product, the customer will place an order.
* After the order is placed, the system will store the customer and his account information in the Order Database.
* The order would be generated against the customer and stored in the Database.
* Every module has its own task but is interlinked with each other to complete the requirements of the software.

The component design of an online shopping system



### Characteristics of Components

* **Reusability** − Components are usually designed to be reused in different situations in different applications. However, some components may be designed for a specific task.
* **Replaceable** − Components may be freely substituted with other similar components.
* **Not context specific** − Components are designed to operate in different environments and contexts.
* **Extensible** − A component can be extended from existing components to provide new behavior.
* **Encapsulated** − A A component depicts the interfaces, which allow the caller to use its functionality, and do not expose details of the internal processes or any internal variables or state.
* **Independent** − Components are designed to have minimal dependencies on other componen

