Software Engineering (2)

Final Revision

2022

<u>تنبيهات هاامة جداا!!</u>

١. يحظر ويمنع منعا باتاا الظهور بهذه المذكرة داخل الجامعة.
٢. يحظر ويمنع منعا باتاا نسخ او نقل او تصوير أي جزء من هذه المذكرة حتى لا تعرض نفسك للمسائلة القانونية .

Chapter 1
Software Reuse

Define the following Software Reuse Concepts:

1) ***Reuse-based software engineering*** is a software engineering strategy where the software development process is geared to reusing existing software.
2) ***System reuse*** is a software engineering approach that refers to reusing a complete software system as a part of system of system.
3) ***Application reuse*** is a software engineering approach that refers to reusing an application by incorporating it without any change into other systems or by configuring the application to different customers.
4) ***Component reuse*** is a software engineering approach that refers to reusing a single subsystem or set of objects to in an application.
5) ***Object and function reuse*** is a software engineering approach that refers to reusing a single function, object, class, or a mathematical function in a software component.
6) ***The reuse landscape*** refers to a variety of software engineering techniques which can be reused, for examples, Enterprise Resource Planning (ERP) system, and Model Driven Engineering software**.**
7) ***Application Framework*** is a software reuse approach refers to Collections of abstract and concrete classes are adapted and extended to create application systems.
8) ***Application system integration*** is a software reuse approach refers to integrating two or more application systems to provide extended functionality.
9) ***Architectural patterns*** are standard software architectures that support common types of application system are used as the basis of applications.
10) ***Configurable application systems*** is a software reuse approach that refers to that domain-specific systems are designed so that they can be configured to the needs of specific system customers.
11) ***Design patterns*** is a generic abstractions that occur across applications and provide description or template for how to solve a design problem that can be used in many different situations (i.e. showing abstract and concrete objects and interactions.
12) ***ERP systems (Enterprise Resource Planning)*** are Large-scale systems that encapsulate generic business functionality and rules to be configured for an organization.
13) ***A legacy system*** is an old or out-dated system, technology or software application that continues to be used by an organization because it still performs the functions it was initially intended to do.
14) ***Legacy system wrapping*** is a software engineering process that defines a set of new interfaces and providing access to these legacy systems through these interfaces.
15) ***Model-driven engineering*** is a software development methodology that focuses on creating and exploiting domain models and can generate "code" from these models.
16) ***Program generators*** is a program that generates other programs with less cost and knowledge through embedding knowledge of a type of application and can be used to generate other applications and systems in a specific domain.
17) ***Generator-based reuse*** involves incorporating reusable concepts and knowledge into automated tools and providing an easy way for tool users to integrate specific code with this generic knowledge.
18) ***Program libraries*** are Class and function libraries that implement commonly used software abstractions and are available for several reuse software purposes.
19) ***Application Framework*** is an integrated set of software artifacts (such as classes, objects and components) that collaborate to provide a reusable architecture for a family of related applications
20) ***Software product lines*** is an application type which is generalized around a common architecture so that it can be adapted to meet specific requirements for functionality, target platform, or operational configuration.

21) ***System Configuration*** is a software engineering process that refers to adding or removing components from the system, defining parameters and constraints for system components, and including knowledge of business processes to adapt the system with the new environment of work.
22) ***Design-time configuration*** is a configuration stage that refers to developing, selecting, or adapting components to create a new system for a customer by modifying the common product-line core component.
23) ***Deployment-time configuration*** is a configuration stage that refers to adapting an application before deploying the application into the target environment without modifying the common product-line core component.
24) ***COTS (Commercial Off-the Shelf System) products*** are classified as application system products which can be defined as a software system that can be adapted to the needs of different customers without changing the source code of the system and developed by a system vendor for a general market.
25) ***Configurable- application systems*** are generic application systems that may be designed to support a particular business type, business activity, or, sometimes, a complete business enterprise. For example, a system produced for dentists may handle appointments, reminders, dental records, patient recall, and billing as a configurable application system.
26) ***Integrated application systems*** include two or more application systems or, sometimes, legacy systems. You may use this approach when no single application system meets all of your needs or when you wish to integrate a new application system with systems that you are already in use. The component systems may interact through set of APIs or service interfaces.

Answer The Following Questions:

1) What are the software units that can be reused?

Answer:

- ***System reuse*** is a software engineering approach that refers to reusing a complete software system as a part of system of system.
- ***Application reuse*** is a software engineering approach that refers to reusing an application by incorporating it without any change into other systems or by configuring the application to different customers.
- ***Component reuse*** is a software engineering approach that refers to reusing a single subsystem or set of objects to in an application.
- ***Object and function reuse*** is a software engineering approach that refers to reusing a single function, object, class, or a mathematical function in a software component.

2) What are the benefits of software reuse?

Answer:

- **Accelerated development:** Reusing software can speed up system production because both development and validation time may be reduced
- **Effective use of specialists:** Instead of doing the same work repeatedly, application specialists can develop reusable software that encapsulates their knowledge.
- **Increased dependability:** Reused software, which has been tried and tested in working systems, should be more dependable and free of implementation errors than new software.
- **Lower development costs:** Reusing software means that fewer lines of code have to be written, this lead to reduce the development costs.
- **Reduced process risk:** Since the cost of reused software is already known, reusing software system reduces the margin of error in project cost estimation.
- **Standards compliance:** Some standards, such as user interface standards, can be implemented as a set of reusable components. The use of standard user interfaces improves dependability because users make fewer mistakes when presented with a familiar interface.

3) What are the challenges of software reuse?

Answer:

- **Creating, maintaining, and using a component library**
- **Finding, understanding, and adapting reusable components**
- **Increased maintenance costs**
- **Lack of tool support**
- **Rewrite software reuse is a challenging issue**

4) What are the SIX Key factors that you should consider when planning software reuse?

Answer:

1. **The development schedule for the software**
2. **The expected software lifetime**
3. **The background, skills and experience of the development team**
4. **The criticality of the software and its non-functional requirements**
5. **The application domain**
6. **The platform on which the system will run**

5) What are the main functionalities of web applications framework (WAF)?

Answer:

1. ***Security***
2. ***Dynamic web pages***
3. ***Database integration***
4. ***Session management***
5. ***User interaction***

6) Explain how a system can be implemented using an application framework based on *'inversion of control' approach*'? Sketch a model to clarify the 'inversion of control' in application framework

Answer:

To implement a system using a framework, you add concrete classes that inherit operations from abstract classes in the framework. In addition, you define "callbacks"— methods that are called in response to events recognized by the framework. The framework objects, rather than the application-specific objects, are responsible for control in the system. This approach is called ' ***inversion of control***' (see figure 1). In response to events from the user interface and database framework objects invoke "hook methods" that are then linked to user-provided functionality. For example, a framework will have a method that handles a mouse click from the environment. This method is called the hook method, which you must configure to call the appropriate application methods to handle the mouse click

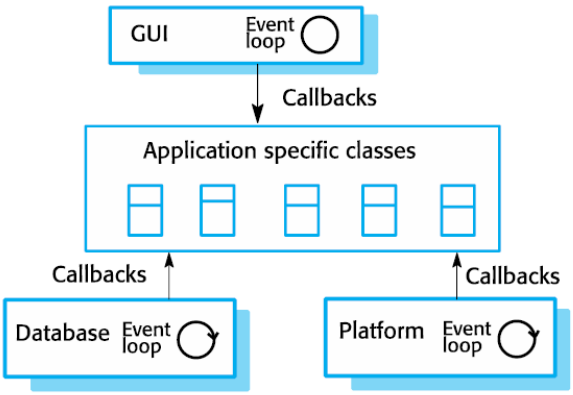

Figure 1: Inversion of control approach in Application frameworks

7) What are the THREE classes of application framework?

Answer:

1. ***System infrastructure frameworks ,*** such as such as communications networks, user interfaces, and compilers
2. ***Middleware integration frameworks,*** such as Microsoft's .NET and Enterprise Java Beans (EJB)
3. ***Enterprise application frameworks***, such as telecommunications or financial systems

8) Explain the THREE components of software product line. Clarify your answer by sketching the software product line architecture

Answer:

The base application of software product line includes (see figure 2):

1. **Core components** that provide infrastructure support. These components are not usually modified when developing a new instance of the software product line.
2. **Configurable components** that may be modified and configured to specialize them to a new application. Sometimes it is possible to reconfigure these components without changing their code by using a built-in component configuration language.
3. **Specialized, domain-specific components** some or all of which may be replaced when a new instance of a product line is created.

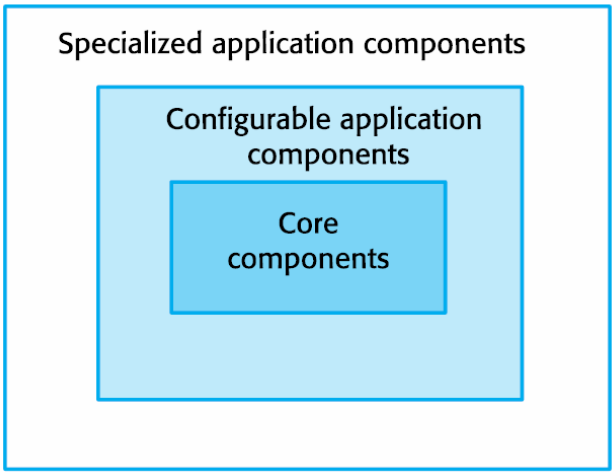

Figure 2: Base design for a software product line

9) *"Frameworks are a very effective approach to reuse. However, they are expensive to introduce into software development processes."* Explain the major challenges of software application framework

Answer:

1. Application frameworks are inherently complex and it can take several months to learn to use them.
2. It can be difficult and expensive to evaluate available frameworks to choose the most appropriate one.
3. Debugging framework-based applications is more difficult than debugging original code because you may not understand how the framework methods interact.
4. Debugging tools may provide information about the reused framework components, which the developer does not understand.

10) Explain the main differences between application framework and software product line

Answer:

Application Framework	Software Product line
Application frameworks rely on object-oriented features such as inheritance and polymorphism to implement extensions to the framework. The code is not modified, and the possible modifications are limited to whatever is supported by the framework.	***Software product lines*** are not necessarily created using an object-oriented approach. Application components code can be changed, deleted, or rewritten. There are no limits for modification
Application frameworks provide general support rather than domain-specific support. For example, there are application frameworks to create web based applications.	***Software product lines*** are usually embeds detailed domain and platform information. For example, there could be a software product line for web-based applications for *health record management system*
Application frameworks are usually *software-oriented*, and they do not usually include hardware interaction components. For example, Microsoft office package is an application framework	***Software product lines*** are *hardware oriented*. It often provides control applications for hardware devices. For example, there may be a software product line for a family of printers.

11) Explain how software product lines can be implemented based on application frameworks. Give an example to clarify your answer.

Answer:

If you are developing a software product line using an object-oriented programming language, then you may use an application framework as a basis for the system. You create the core of the product line by extending the framework with domain specific components using its built-in mechanisms. There is then a second phase of development where versions of the system for different customers are created.

For example, you can use a web-based framework to build the core of a software product line that supports web-based help desks. This "help desk product line" may then be further specialized to provide particular types of help desk support. For example, consider a product-line system that is designed to handle *vehicle dispatching for emergency services*. Operators of this system take calls about incidents, find the appropriate vehicle to respond to the incident, and dispatch the vehicle to the incident site. The developers of such a system may market versions of it for police, fire, and ambulance services. Therefore, the vehicle *dispatching system* is an example of a *generic resource allocation and management* architecture as depicted below in figure 3 (a,b)

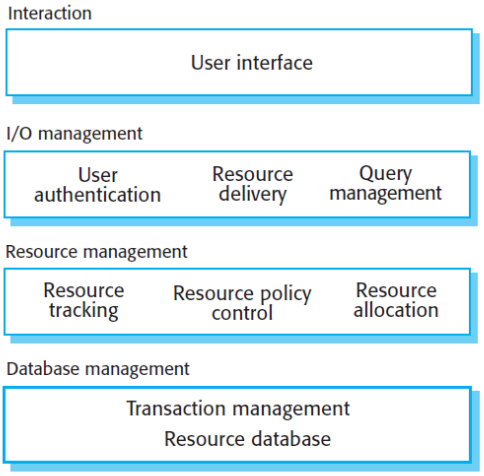

Figure 3 (a): The architecture of a *resource management system*

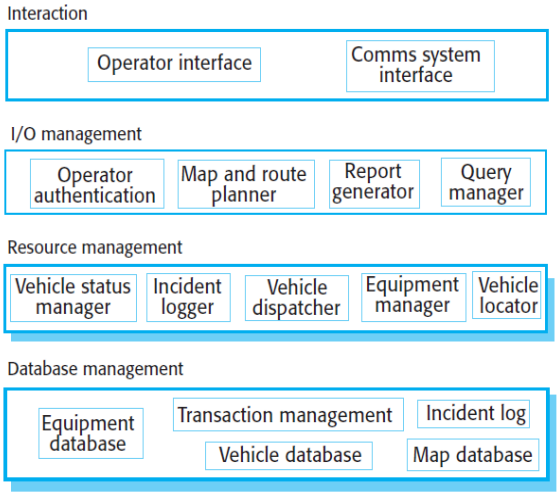

Figure 3(b): A product line architecture of *a vehicle dispatcher system*

12) What are the FOUR types of specialization of a software product line?

Answer:

- ***Platform specialization*** Versions of the application may be developed for different platforms. For example, versions of the application may exist for Windows, Mac OS, and Linux platforms.
- ***Environment specialization*** Versions of the application may be created to handle different operating environments and peripheral devices. For example, a system for the emergency services may exist in different versions, depending on the communications hardware used by each service.
- ***Functional specialization*** Versions of the application may be created for specific customers who have different requirements. For example, a library automation system may be modified depending on whether it is used in a public library, a reference library, or a university library.
- ***Process specialization*** The system may be adapted to cope with specific business processes. For example, an ordering system may be adapted to cope with a centralized ordering process in one company and with a distributed process in another.

13) Explain the FIVE phases software engineering process for extending a software product line to create a new version of software system?

Answer: (see figure 4)

1. **Elicit stakeholder requirements**
2. **Select the existing system that is the closest fit to the requirements**
3. **Renegotiate requirements**
4. **Adapt existing system**
5. **Deliver new product family member**

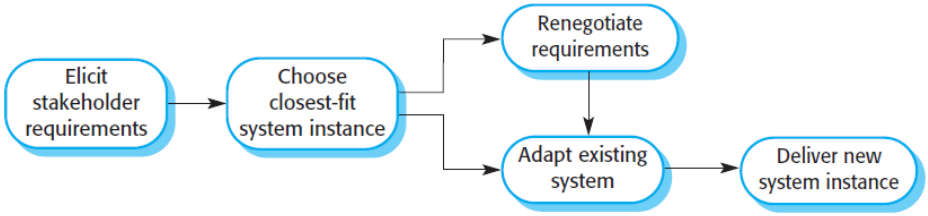

Figure 4: Software Product instance development Process

14) Explain what is the meaning of *software product line configuration* , and what are the two types of configurations processes that may occur in developing software product lines

Answer:

Software product lines are designed to be reconfigurable. This reconfiguration may involve adding or removing components from the system, defining parameters and constraints for system components, and including knowledge of business processes. This configuration may occur at different stages in the development process:

1. ***Design-time configuration*** The organization that is developing the software modifies a common product line core by developing, selecting, or adapting components to create a new system for a customer.
2. ***Deployment-time configuration*** A generic system is designed for configuration by a customer or consultants working with the customer. Knowledge of the customer's specific requirements and the system's operating environment is embedded in the configuration data used by the generic system. It Involves using a configuration tool to create a specific system configuration that is recorded in a configuration database or as a set of configuration files as in figure 5

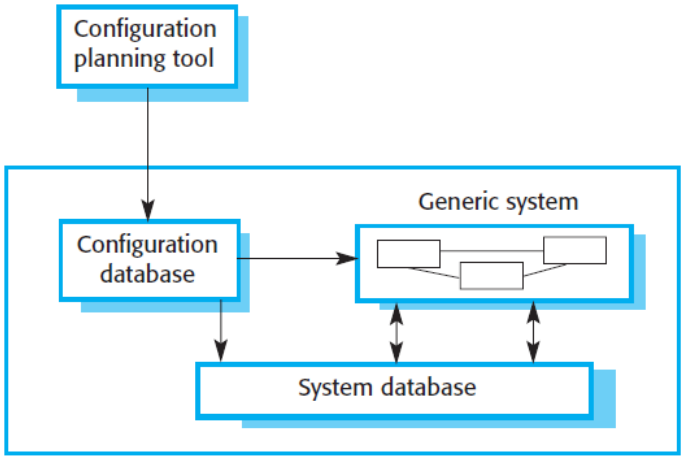

Figure 5: Deployment-time configuration Process.

15) What is meaning of an *application system product reuse*? In addition, what are its benefits and challenges a software engineering process?

Answer:

An application system product reuse is a software system that can be adapted to the needs of different customers without changing the source code of the system. Application systems are developed by a system vendor for a general market; they are not specially developed for an individual customer. These system products are sometimes known as COTS (Commercial Off-the Shelf System) products.

The benefits of application system product reuse approach are:

1. Rapid deployment
2. Easier to specify system functionality
3. Development risks are avoided
4. Development can focus on their core activity without having to devote a lot of resources to IT systems development
5. Easy to add new updates to the system as these are the responsibility of the application system vendor rather than the customer.

The challenges of application system product reuse approach are:

1. Requirements usually have to be adapted to reflect the functionality and mode of operation of the off-the-shelf application system.
2. It is very difficult to change the source code
3. Choosing the right application system for an enterprise can be a difficult process, especially as many of these systems are not well documented
4. There may be a lack of local expertise to support systems development. Consequently, the customer has to rely on the vendor and external consultants for development advice
5. The system vendor controls system support and evolution. It may go out of business be taken over, or make changes that cause difficulties for customers

16) Compare between *Configurable application systems* and *Application system integration*, and give an example for each approach

Answer:

Configurable application systems	Application system integration
Single product that provides the functionality required by a customer	Several different application systems are integrated to provide customized functionality
Based on a generic solution and standardized processes	Flexible solutions may be developed for customer processes
Development focus is on system configuration	Development focus is on system integration
System vendor is responsible for maintenance	System owner is responsible for maintenance
System vendor provides the platform for the system	System owner provides the platform for the system

- **Enterprise Resource Planning (ERP) systems** is a good example of *configurable application systems*. A generic ERP system includes a number of modules that may be composed in different ways to create a system for a customer. The configuration process involves choosing which modules are to be included, configuring these individual modules, defining business processes and business rules, and defining the structure and organization of the system database. A model of the overall architecture of an ERP system that supports a range of business functions is shown in figure 6:

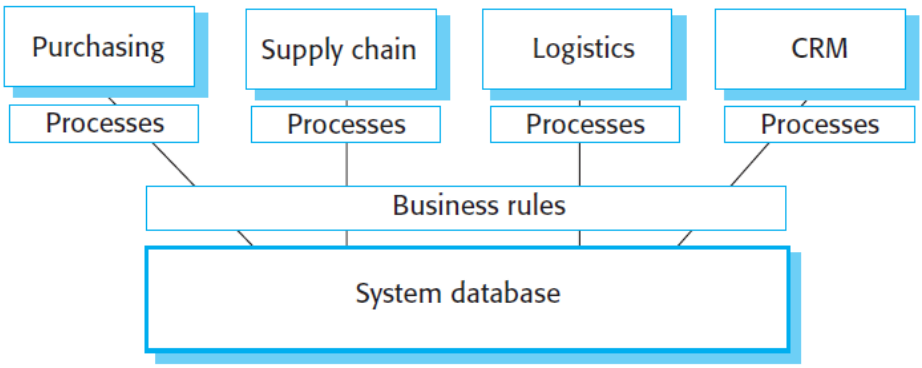

Figure 6: Enterprise Resource Planning (ERP) systems

- **Integrated procurement system** is a good example of Application system integration. It is designed as a client-server design model that integrates different systems types, which can communicate using 'Adapters'. The client side includes web browser and email systems, and the server side includes Ecommerce system, ordering invoicing system and email system. The variance in data format exchange between these systems is adapted using adapter systems. The general architecture of Integrated procurement system is shown in figure 7

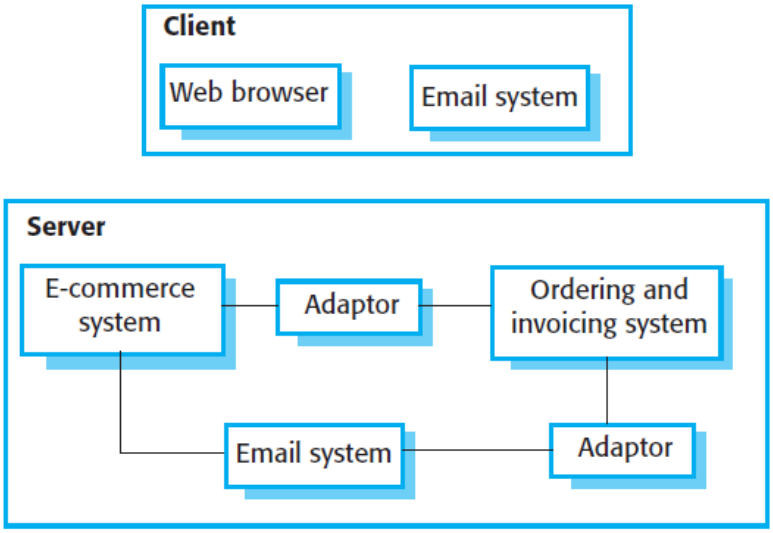

Figure 7: An integrated procurement system

17) What are the FOUR major Application system integration problems?

Answer:

1. **Lack of control over functionality and performance**
2. **Problems with system interoperability**
3. **No control over system evolution**
4. **Support from system vendors**

Chapter 2
Component-based Software Engineering

Define the following Software Reuse Concepts:

1) ***Component-based software engineering (CBSE)***: is an approach to software development emerged in the 1990's that relies on the reuse of entities called 'software components'. It emerged from the failure of object-oriented development to support effective reuse. Components are more abstract than object classes and can be considered stand-alone services.
2) ***A software component:*** A software element that conforms to a standard component model and can be independently deployed and composed without modification according to a composition standard.
3) ***Component interface:*** is a device or program that enable software components communicate with each other. Each component has two interfaces, *provide*, and *require*.
4) ***Provides interface***: Defines the services that are provided by the component to other components.
5) ***Requires interface:*** Defines the services that specifies what services must be made available for the component to execute as specified.
6) ***A component model:*** is a definition of standards for component implementation, documentation, and deployment. It specifies how interfaces should be defined and the elements that should be included in an interface definition
7) ***Remote Procedure Call (RPC):*** is a software communication protocol that one component can use to request a service from another software component located in another computer based on a unique identifier (usually a URL) and can be referenced from any networked computer.
8) ***Middleware:*** is a technology that implements an infrastructure for supporting component functionality, and communication for easing the use and implementation of component-based software.
9) ***CBSE Processes:*** are software processes that support component-based software engineering. They take into account the possibilities of reuse and the different process activities involved in developing and using reusable components.
10) ***CBSE for reuse*** is concerned with developing components or services that will be reused in other applications. It usually involves generalizing existing components.
11) ***CBSE with reuse***: is the process of developing new applications using existing components and services.
12) ***Component composition*** is the process of assembling components to create a system. Composition involves integrating components with each other and with the component infrastructure. Normally you have to write ***'glue code'*** to integrate components, and to reconcile the different component interfaces.
13) **Sequential composition:** where the composed components are executed in sequence. This involves composing the provides interfaces of each component.
14) ***Hierarchical composition***: where one component calls on the services of another. The provides interface of one component is composed with the requires interface of another.
15) ***Additive composition:*** where the interfaces of two components are put together to create a new component. *Provides* and *requires* interfaces of integrated component is a combination of interfaces of constituent components

Answer The Following Questions:

1) What are the essentials of component-based software engineering? And what are CBSE design principles as a software engineering process?

Answer:

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.

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.

2) What are the main characteristics of software components?

Answer:

Component characteristics are:

- **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 that 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.
- **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.

3) *"The services offered by a software component are made available through Two interfaces":*

- **What are the two interfaces of a software component? Explain using a model how all component interactions take place through those two interfaces.**

Answer:

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.

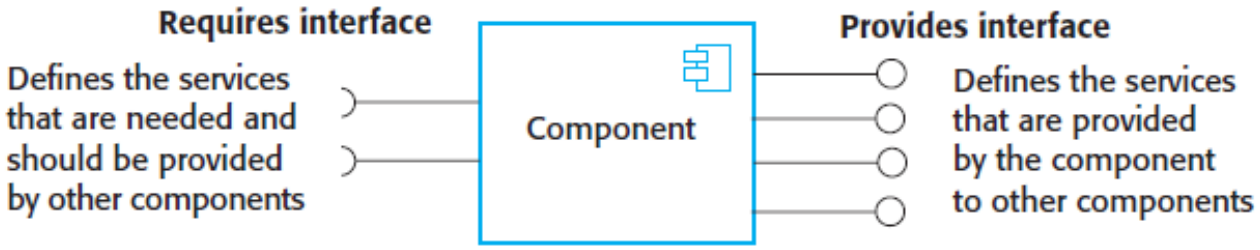

Figure 1: Software component interfaces.

4) Explain using a diagram what is software component model? In addition, define the main three elements of a component model.

Answer:

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 as depicted in figure 2

Elements of a component model:

Interfaces
Components are defined by specifying their interfaces. The component model specifies how the interfaces should be defined and the elements, such as operation names, parameters and exceptions, which should be included in the interface definition.

Usage
In order for components to be distributed and accessed remotely, they need to have a unique name or handle associated with them. This has to be globally unique.

Deployment
The component model includes a specification of how components should be packaged for deployment as independent, executable entities.

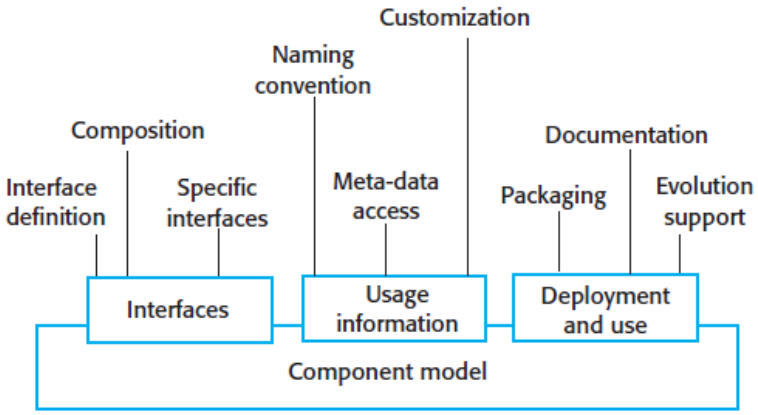

Figure 2: Basic Elements of Software component Model.

5) *"Middleware implements the common component services and provides interfaces to them for easing the use and implementation of component-based software*."

- **Explain using a diagram the TWO Middleware services defined in a software component model**

Answer:

The services provided by a component model implementation fall into two categories:

- **Platform services**, which enable components to communicate and interoperate in a distributed environment
- **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.

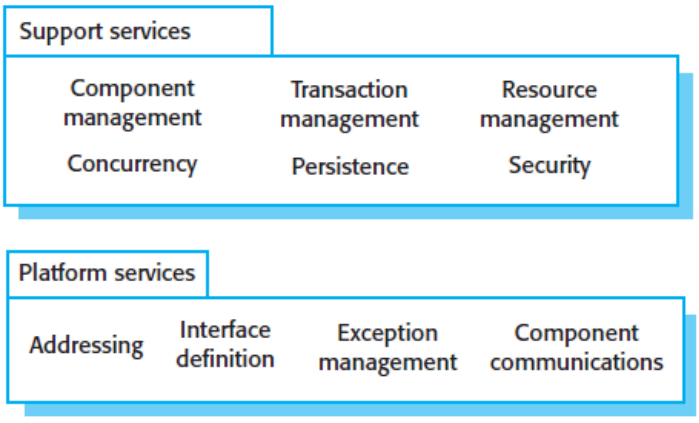

Figure 3: Middleware services defined in a component model.

6) Define what is what is CBSE processes?, In addition, explain the main THREE activities of CBSE processes

Answer:

CBSE processes are software processes that support component-based software engineering. They take into account the possibilities of reuse and the different process activities involved in developing and using reusable components. There are two types of CBSE processes:

- **CBSE for reuse** is concerned with developing components or services that will be reused in other applications. It usually involves generalizing existing components.
- **CBSE with reuse** is the process of developing new applications using existing components and services.

CBSE activities are:

- ***Component acquisition*** is the process of acquiring components for reuse or development into a reusable component.
- ***Component management*** is concerned with managing a company's reusable components, ensuring that they are properly catalogued, stored, and made available for reuse
- ***Component certification*** is the process of checking a component and certifying that it meets its specification.

7) What is the meaning of *Component reusability*?

Answer:

Component reusability:

- Should reflect stable domain abstractions;
- Should hide state representation;
- Should be as independent as possible;
- Should publish exceptions through the component interface.

8) *"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***.**

- **What are changes that you may make to a software component to make it more reusable?**

Answer:

To make an existing component reusable:

- 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.

9) Explain using a diagram the main activities of CBSE with reuse?

Answer:

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 (as in figure 4):

- 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.

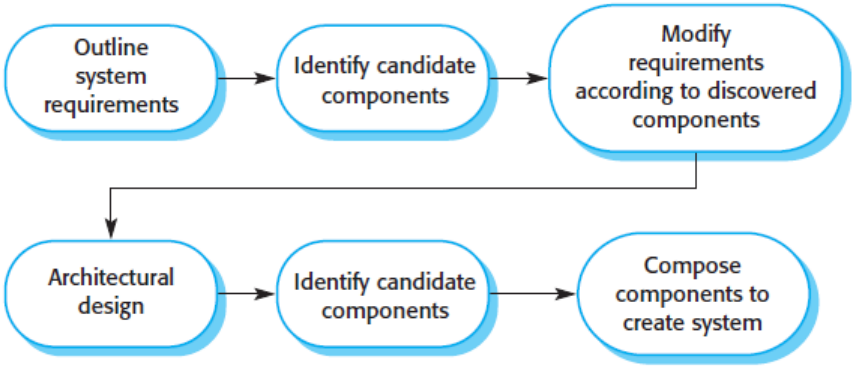

Figure 4: CBSE with reuse

10) Explain what is the meaning of each of the following?

- **Component Identification**
- **Component Validation**

Answer:

Component identification issues:

- 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.
- 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 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.

11) Explain Three types of component composition (clarify your answer using a diagrams)

Answer:

Three types of component composition:

1. ***Sequential composition*** where the composed components are executed in sequence. This involves composing the provides interfaces of each component.
2. ***Hierarchical composition*** where one component calls on the services of another. The provides interface of one component is composed with the requires interface of another.
3. ***Additive composition*** where the interfaces of two components are put together to create a new component. Provides and requires interfaces of integrated component is a combination of interfaces of constituent components.

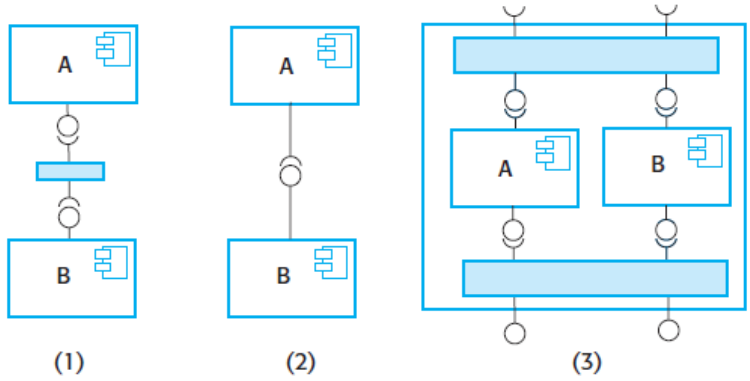

Figure 5: Types of component composition

12) What are the three types of incompatibility problems while developing and composing software components? How to solve these problems?

Answer:

When components are developed independently for reuse, their interfaces are often incompatible. Three types of incompatibility can occur:

- **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.

These problems can be solved using Adaptor. Adaptor components address the problem of component incompatibility by reconciling the interfaces of the components that are composed. Different types of adaptor are required depending on the type of composition.