UNIT-5(SOFTWARE REUSE) COVERS

- The reuse landscape
- Design patterns
- Generator-based reuse
- Application frameworks
- Application system reuse

SOFTWAR REUSE

- Software reuse is the process of creating software systems from existing software rather than building software systems from scratch.
- The software units that are reused may be of radically different sizes.

For example

- **Application system reuse**:

The whole of an application system may be reused by incorporating it without changing into other system or by configuring the application for different customers.

- **Component reuse**:

Components of an application, ranging in size from subsystems by single objects may be reused

- **Object and function reuse**:

Software components that implement a single function, such as mathematical function, or an object class may be reused.

SOFTWAR REUSE

Benefit of software reuse

Increased dependability:

Reused software which has been tried and tested in working systems, should be more dependable than new software.

Reduced process risk:

The cost of existing software is already known, whereas the costs of development are always a matter a judgement. This is an important factor for project management because it reduces the margin of error in project cost estimation.

Effective use of specialists

Instead of doing the same work over and over again, application specialists can develop reusable software that encapsulates their knowledge.

Standards compliance

Some standards such as user interface standardsm can be implemented as a set of reusable components. For example if menus in a user interface are implemented using reusable components, all application present the same menu formats to users.

Accelerated development

Bringing a system to market as early as possible is often more important than overall development costs. Reusing software can speed up system production because both development and validation time may be reduced.

SOFTWAR REUSE

Problem with software reuse:

Increased maintenance costs:

If the source code of a reused software system or component is not available, then maintenance costs may be higher because the reused elements of the system may become increasingly incompatible with system changes.

Lack of total support

Some software tools do not support development with reuse. It may be difficult or impossible to integrate these tools with a component library system.

Not-invented syndrome

Some software engineers prefer to rewrite components because they believe they can improve on them. This is partly to do with trust and partly to do with the fact that writing original software is seen as more challenging than reusing other peoples software.

Creating, maintaining and using a component library

Populating a reusable component library and ensuring the software developers can use this library can be expensive. Development processes have to be adapted to ensure that the library is used

Finding, understanding and adapting reusable component

Software components have to be discovered in a library, understood and sometimes adapted to work in a new environment. Engineers must be reasonably confident of finding a component in the library before they include a component search as part of their normal development process.

The reuse landscape

- Although reuse is often simply thought of as the reuse of system components there are many different approaches to reuse that may by used.
- Resuse is possible at a range of levels from simple functions to complete application systems.
- The reuse landscape covers the range of possible reuse techniques.

THE REUSE LANDSCAPE

These techniques exploit the facts that systems in the same application domain are similar and have potential for reuse.

That reuse is possible at different levels from simple functions to complete applications and that standards for reusable components facilitate reuse.

Key factors that you should consider when planning reuse are

1. The development schedule for the software

- If the software has to be developed quickly, you should try to reuse off-the self systems rather than individual components.
- These are large grain reusable assests. Although the fit to requirements may be imperfect this approach minimized the amount of development required.

2. The expected software lifetime

If you are developing a long-lifetime system, you should focus on the maintainability of the system. You should not just think about the immediate benefits of reuse but also of the long-term implications.

THE REUSE LANDSCAPE

3. The background,skills, and experience of the development team:

All reuse technologies are fairly complex and you need quite a lot of time to understand and use them effectively.

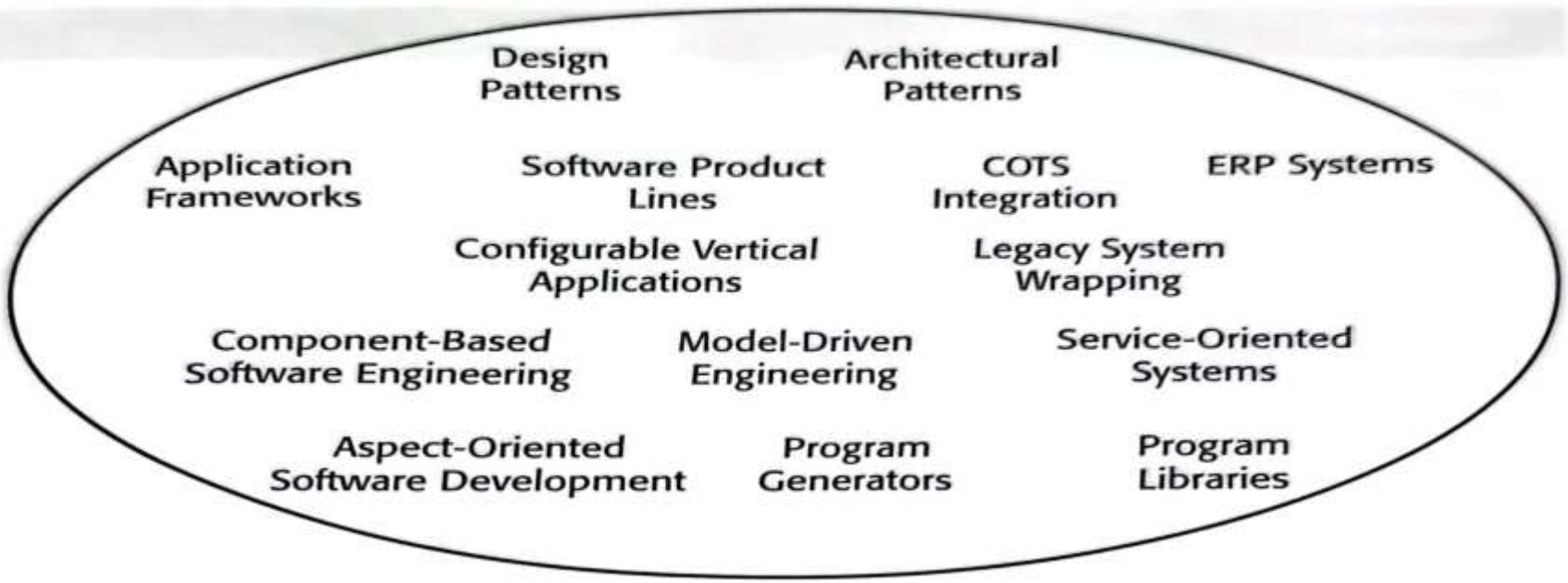
4. The criticality of the software and its non-functional requirement

For a critical system that has to be certified by an external regulator, you may have to create dependability case for the system.

5. The application domain:

In some application domain such as manufacturing and medical information system, there are several generic products that may by reused by cofirming them to a local situation.

THE REUSE LANDSCAPE



reuse

CONCEPT REUSE

- When you reuse or design components, you have to follow the design decisions made by the original developer of the component.
- This may limit the opportunities for reuse.
- However, a more abstract form of reuse is concept reuse when a particular approach is described in an implementation independent way and an implementation is then developed.
- The two main approaches to concept reuse are:
- Design patterns
- Generative programming

DESIGN PATTERNS

- Design pattern is a general repeatable solution to a commonly occurring problem in software design.
- A design pattern is not a finished design that can be transformed directly into code. Rather, it is a description or template for how to solve a problem that can be used in many different situation
- Design patterns can speed up the development process by providing tested, proven development paradigms.
- Reusing design patterns helps to improve code readability for coders and architects familiar with the patterns.
- This should be sufficiently abstract so that it can be used in different system settings.
- These patterns often rely on the characteristics of the object characterized by the inheritance and polymorphism.

DESIGN PATTERNS

Design pattern Elements

Name: A meaning pattern identifier

Problem description: The complete description of the problem to be addressed

Solution description: A template for a design solution that can be instantiated in different operational context. It is not a concrete design. It is often illustrated graphically.

Consequences: The results and trade-offs of applying the pattern. It includes the analysis and experience.

Example: The observer pattern

Name:observer

Description:seperates the display of object state from the object itself allowing alternative displays.

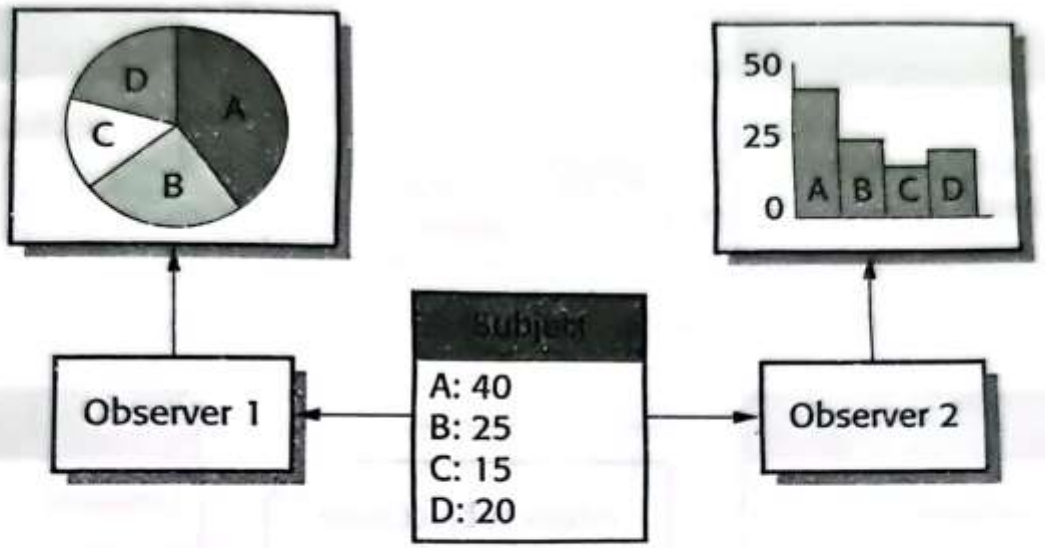
Problem description:

Used when multiple display of state are needed

Solution description: It is based on the UML description

Consequences: Object optimization to enhance the performance of a particular displays are impractical.

DESIGN PATTERNS



GENERATOR BASED REUSE

- The program generators involve the reuse of standard programs and algorithms.
- These are embedded in the generator and parameterized by the user command and a program is then automatically generated.
- Generator-based reuse is possible when domain abstraction and their mapping to executable code can be identified.
- A domain specific language is used to compose and control these abstraction

Types of program generator:

1. Application generator for business data processing
2. Parse and lexical analyzer generator for language processing
3. Code generation in CASE tools

- Generator based reuse is very cost effective but its applicability is limited to a relatively small number of application program.
- It is easier for end-users to devlop programs using generators compared to other component based approaches to reuse.

APPLICATION FRAMEWORKS

- An application framework is a software library that provides a fundamental structure to support the development of applications for a specific environment.
- It consist of a software framework used by software developers to implement the standard structure of an application.
- Became popular with the rise of graphical user interfaces since these tended to promote a standard structure for applications.
- Programmers find it much simpler to create automatic GUI creation tools when using a standard framework since this defines the underlying code structure of the application in advance.
- Developers usually use object-oriented programming techniques to implement frameworks such that the unique parts of an application can simply inherit from pre-existing classes in the framework.
- Examples of application frameworks include Django and Ruby on Rails for web development, .NET Framework and Spring Framework for enterprise applications, and React and Angular for frontend development.

FRAMEWORK CLASSES

- Frameworks are implemented as a collection of concrete and abstract object classes in an object-oriented programming language, therefore are language specific.
- There are frameworks available in all of the commonly used object-oriented programming language, therefore are language specific.
- There are framework available in all of the commonly used object-oriented programming languge like java, C#, C++ etc.
- In fact a framework can incorporate several other framework where each of these is designed to support the development of a part of the application.
- We can use a framework to create a complete application or to implement part of an application, such as the graphical user interface
- **System infrastructure frameworks**

Support the development of system infrastructures such as communications, user interfaces and compilers.

- **Middleware integration frameworks**

Standards and classes that support component communication and information exchange

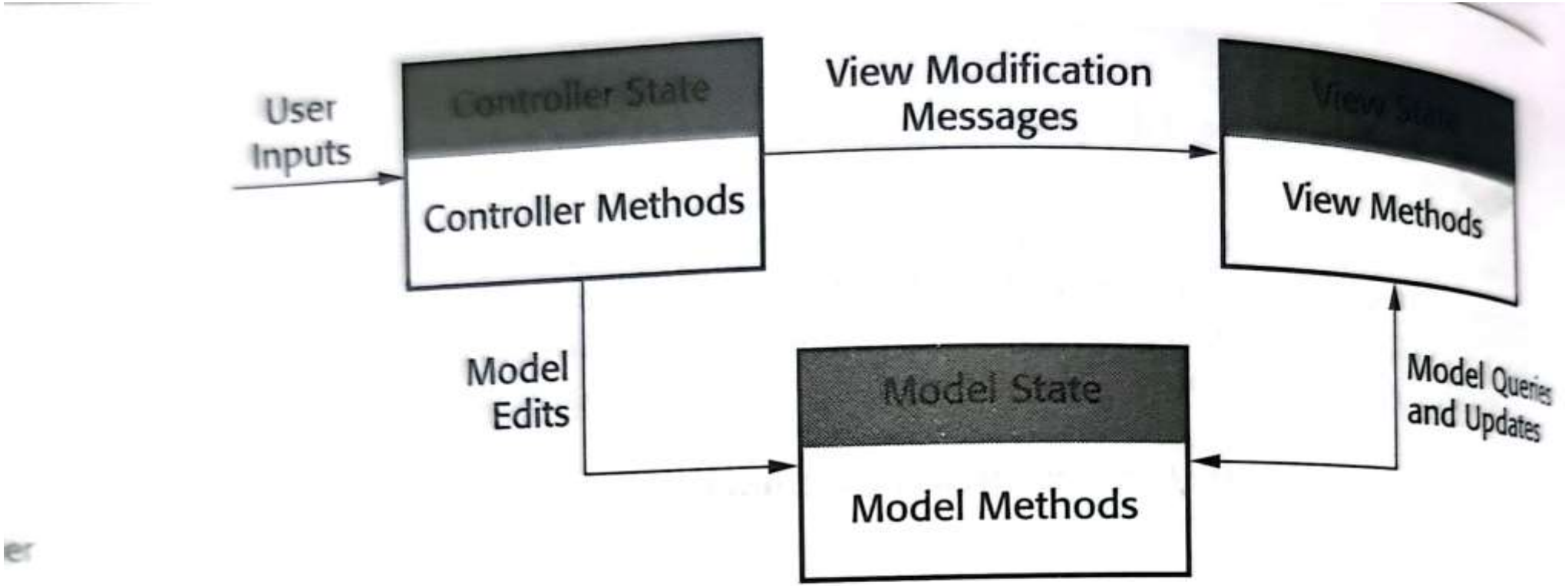
- **Enterprise application frameworks**

Support the development of specific types of application such as telecommunications or financial systems.

MODEL VIEW CONTROLLER(MVC)

- The Model View Controller(MVC) is an architectural pattern that seperates an application into three main logical components: the model, the view and the controller.
- Each of these components is built to handle specific development aspects of an application
- MVC is one of the most frequently used industry-standard web development framework to create scalable and extensible projects.
- It divides a given software application into three interconnected parts so as to separate internal representations of information from the ways that information is presented to or accepted from the user.
- Traditionally used for desktop graphical user interface this architecture has become extremely popular for designing web applications.

MODEL VIEW CONTROLLER(MVC)



MODEL VIEW CONTROLLER(MVC)

Model:

- The model component corresponds to all the data-related logic that the user works with.
- This can represent either the data that is being transferred between the view and controller components or any other business logic-related data.
- For example: a customer object will retrieve the customer information from the database, manipulate it and update it data back to the database or use it to render data.

MODEL VIEW CONTROLLER(MVC)

View

- The view component is used for all the UI logic of the application.
- For example: The customer view will include all the UI components such as textboxes, dropdowns etc. that the final user interacts with.

Controller

- Controllers act as an interface between Model and View components to process all the business logic and incoming requests, manipulate data using the model component and interact with the Views to render the final output
- For example: The customer controller will handle all the interactions and inputs from the customer view and update the database using the customer model
- The same controller will be used to view the customer data.

MODEL VIEW CONTROLLER(MVC)

View

- The view component is used for all the UI logic of the application.
- For example: The customer view will include all the UI components such as textboxes, dropdowns etc. that the final user interacts with.

Controller

- Controllers act as an interface between Model and View components to process all the business logic and incoming requests, manipulate data using the model component and interact with the Views to render the final output
- For example: The customer controller will handle all the interactions and inputs from the customer view and update the database using the customer model
- The same controller will be used to view the customer data.

APPLICATION SYSTEM REUSE

Application system Reuse involves the reuse of entire application either by configuring a system for an environment or by integrating two or more system to create a new application.

There may be two approaches for application system reuse. They are:

1. COTS production integration
2. Product line development

APPLICATION SYSTEM REUSE

1. **COTS(Commercial On The Self):**

- COTS are usually complete application system that often is an API which benefits in faster application development at lower costs.
- E-procurement System is one of the example of COTS production reuse.

APPLICATION SYSTEM REUSE

2. Product line development

- A software product line is a set of applications with a common architecture and shared components with each application specialized to reflect different requirement.
- This core system is designed to be configured and adapted to suit the needs of different system customers.
- This may involve the configuration of some components implementing additional components and modifying some of the components to reflect new requirements.
- Software product lines usually emerge from existing applications. That is , an organization develops an application then, when a similar system is required informally reuses code from this in the new applications.
- The same process is used as other similar applications are developed.

Application System Reuse

- However, change tends to corrupt application structure so as more new instances are developed it becomes increasingly difficult to create a new version. Consequently a decision to design a generic product line may then be made.
- This involves identifying common functionality in product instances and including this in a base application, which is then used for future development.
- This base application is deliberately structured to simplify reuse and reconfiguration