Unit 6 SDP

The Iteration Planning Process

Iteration Release Plan

- This plan divides the project into smaller steps or "increments." This makes the project easier to manage and track progress.
- Each increment builds on the previous one, gradually adding features and functionality. This ensures a structured and controlled development process.

Iteration Plan Goals

- **What to Build**: Clearly defines the specific features or capabilities that will be created during this iteration.
- **Addressing Risks**: Identifies potential problems that might arise and outlines strategies to minimize their impact.
- **Fixing Bugs**: Lists any known issues or errors that need to be fixed during this iteration.

Exit Criteria

- **Updated Information:** Ensures that the team has a clear understanding of the features developed and any changes to the risk management plan.
- **Release Summary:** A document that captures the key outcomes of the iteration, including the features added, issues faced, and lessons learned.
- **Testing and Results:** Includes the test cases used to verify the new features, along with the results of those tests and any defects found.

In simple terms, the Iteration Planning Process is a way to break down a large project into smaller, manageable steps. It helps in organizing the work, tracking progress, and ensuring that the project is completed in a structured and controlled manner.

Designing The User Interface

- Earlier in the project, we created basic outlines for the user interface (UI) elements. Now, we need to finalize the details, such as the number of windows, their layout, and how they will respond to user actions.
- Sequence diagrams help us understand how the different parts of the system interact with each other, including the user interface.
- The UI must be able to receive all inputs from the user and send all necessary information to other parts of the system.

Example: Select Courses to Teach: User Interface Design

- **Professor's Access:** The professor needs to enter a password to access the system.
- **Course Options:** After entering a valid password, the professor will see a window with options to add, delete, review, and print their course schedule.

- **Course Information:** The professor can enter a course name and number, and the system will retrieve and display the available course offerings.
- **Course Selection:** The professor can select a course offering, and the system will record the selection.

Adding Design Classes

- We add new classes to the system to implement specific tasks. For example, a class might be created to validate passwords.
- As we add new classes, we update the diagrams to show their relationships with other parts of the system.

The Emergence of Patterns

- **Solving Common Problems:** Design patterns are like pre-designed solutions to common problems in software development. They help in creating well-structured and maintainable systems.
- Patterns allow us to reuse successful designs, which saves time and effort.

Designing Relationships

- When designing relationships between objects, we need to decide whether the relationship should be bidirectional (objects can navigate to each other) or unidirectional (objects can only navigate in one direction). Unidirectional relationships are generally simpler to implement and maintain.
- **Owning Objects:** We also need to decide who "owns" an object. If one object owns another, it has exclusive control over the owned object's lifecycle. This is represented by a filled diamond on the owning object.
  If the relationship doesn't imply ownership, it's shown by an open diamond.
- **Dependency vs. Association:** Sometimes, a strong association (where one object needs to know the location of another) can be refined into a weaker dependency relationship. This means the client object doesn't need to know the supplier object's location beforehand and can be passed as a parameter or declared locally.
- **Multiplicity Implementation:** A one-to-one relationship can be implemented using an embedded object, a reference, or a pointer.
  A one-to-many relationship is typically implemented using a container class like a set or a list. The container can be embedded or referenced.

Designing Attributes and Operations

- **Basic Information:** During the early stages of design, we focus on defining the essential properties (attributes) and actions (operations) of objects.
- **Adding Details:** As the design progresses, we add more specific information, such as the data type of attributes, the parameters and return types of operations, and access control (who can use these attributes and operations).

Designing for Inheritance

- **Identifying Similarities:** We look for commonalities among different objects.

- **Creating Superclasses:** We create a "parent" class (superclass) to hold these commonalities, and then create "child" classes (subclasses) that inherit these properties and actions. This helps in code reuse and consistency.

Coding, Testing, and Documenting the Iteration

Implementing the Iteration

- **Method Bodies:** This step involves writing the actual code for the methods that were defined in the design phase.
- **Sequence and Collaboration Diagrams:** These diagrams help visualize how different objects interact with each other. They show who calls which methods and when.

Testing the Iteration

- **Importance of Testing:** Testing is a crucial part of the development process, even though it might not be explicitly discussed until later.
- **Test Plans and Procedures:** As the design progresses, we create detailed plans and procedures for testing the system.
- **Use Cases:** Use cases, which describe the interactions between users and the system, are an important tool for defining test cases.

Documenting the Iteration

- **Testing for Accuracy:** The iteration should be thoroughly tested to ensure that it functions as expected and meets the requirements specified in the use cases.
- **Early Integration:** Iterations should be integrated with previous iterations as soon as possible. This helps in catching errors early and ensures that the system works as a whole.
- **Risk Evaluation:** The iteration should be evaluated to determine if the identified risks have been mitigated. Any remaining risks should be reassigned to future iterations.
- **Design Documentation:** The design decisions made for the iteration should be captured in models and diagrams. This information is used to generate the documentation for the iteration.
- **Iterative Documentation:** Documentation should be created on an iterative basis. Waiting until the end of the project to document the system can lead to poor documentation or no documentation at all.