**Problem Statement 9:**
**Coupling & Cohesion :**
Focus on the user authentication module of EduLearn Pro. Evaluate the current level of coupling and cohesion in the design. Propose and implement modifications to improve cohesion and reduce coupling, ensuring a more modular and maintainable authentication system.

**Solution :**
**To evaluate the current level of coupling and cohesion in the user authentication module of EduLearn Pro, it's essential to understand these concepts:**

**Cohesion:**

- Cohesion refers to how closely the components within a module are related to each other. High cohesion means that the components in a module are closely related and work together to achieve a common goal.
  Coupling:
- Coupling refers to the degree of interdependence between modules. Low coupling means that modules are independent of each other, making the system more modular and maintainable.

**Evaluation of Current Design:**

**Cohesion:**

The cohesion in the current authentication module can be assessed based on the responsibilities of its components. Cohesion can be categorized into different types:

**Functional Cohesion:**

- If the module is focused on a single, well-defined task (e.g., user authentication), it exhibits functional cohesion.
- Sequential Cohesion:
- If the components are organized in a sequence of execution, it indicates sequential cohesion.

**Communicational Cohesion:**

- If components share common data, it exhibits communicational cohesion.
- Procedural Cohesion:
- If components work together to perform a specific procedure, it indicates procedural cohesion.

**Coupling:**

The coupling in the current design can be assessed by looking at how components within the authentication module interact with each other and with external modules. Low coupling involves minimizing dependencies between modules.

**Proposed Modifications:**

**Separation of Concerns:**

- Introduce a clear separation of concerns by dividing the authentication module into distinct components for user data management, password hashing, and session handling. This will improve functional cohesion.

**Use Dependency Injection:**

- Reduce direct dependencies between components by implementing dependency injection. For example, inject a user data management service into the authentication service instead of having the authentication service directly access the user data.

**Single Responsibility Principle (SRP):**

- Ensure that each class or component has a single responsibility. Extract any unrelated functionality from the authentication module into separate modules.

**Encapsulate Database Access:**

- Encapsulate database access within a dedicated data access layer. This reduces coupling by isolating database-related code from the authentication logic.

**Adopt Interface-Based Programming:**

- Use interfaces to define contracts between different components. For instance, define an interface for the user data management service, allowing for flexibility in the implementation and reducing coupling.

**Decouple Authentication and Authorization:**

- Separate authentication and authorization responsibilities. Authentication focuses on verifying user identity, while authorization determines access rights. This separation enhances modularity.

**Event-Driven Architecture:**

- Implement an event-driven architecture where components communicate through events. This reduces direct dependencies and allows for better modularity.

**Encourage Loose Coupling through Messaging:**

- Use messaging patterns (e.g., publish/subscribe) to decouple components. Events or messages can be used for communication between authentication and other modules.

**Encourage Dependency Inversion:**

- Apply the Dependency Inversion Principle by depending on abstractions rather than concrete implementations. This promotes low-level modules depending on high-level abstractions, reducing coupling.

**Implementation Steps:**

**Refactor Code:**

- Refactor the existing codebase to implement the proposed changes gradually.

**Unit Testing:**

- Develop unit tests to ensure that each component works independently and as part of the larger authentication module.
  Integration Testing:
- Conduct integration tests to verify that the modified components work seamlessly together with reduced coupling.

**Documentation:**

- Update documentation to reflect the changes made to the authentication module, ensuring that future developers understand the design and its principles.

**Code Reviews:**

- Conduct code reviews to gather feedback from team members and ensure that the modifications align with best practices and design principles.

By focusing on improving cohesion and reducing coupling in the user authentication module, EduLearn Pro can achieve a more modular and maintainable authentication system, making it easier to understand, extend, and maintain in the long run.