#	Problem	PIC	Location	Solution
1	DIP Violation: UserValidator directly depends on concrete class UserRepository	Hà Việt Khánh	class UserValidator	Apply DIP: Create an abstraction (e.g., UserDataAccess) and inject it into UserValidator
2	SRP Violation: UserRepository handles both data access and logic like email/phone checking	Hà Việt Khánh	class UserRepository	Apply SRP: Move validation logic to a separate class (e.g., UserPolicyService)
3	SRP Violation: User class mixes user data with role parsing logic	Hà Việt Khánh	class User	Apply SRP: Extract the fromString() method into a separate class such as UserRoleFactory
4	ISP Violation: UserEmailService has more methods than needed for the test	Hà Việt Khánh	class CreateUserControllerTest	Apply ISP: Split UserEmailService into smaller, role-specific interfaces (e.g., EmailNotifier, etc.)
5	ISP Violation: UserEmailService interface may include more methods than necessary for test	Hà Việt Khánh	class CreateUserControllerTest	Apply ISP: Split UserEmailService into smaller interfaces like EmailSender
6	ISP PlaceOrderController violates the ISP because it depends on multiple UI interfaces (e.g., DeliveryFormScreen, InvoiceScreen, OrderInfoScreen) while only using a portion of their functionality in each method.	Hồ Bảo Thư	Class "PlaceOrderController"	Separating UI responsibilities into PlaceOrderInterface allows the controller to focus solely on business logic and depend only on service interfaces relevant to its core purpose
7	DIP RejectOrderController violates the DIP if it directly depends on concrete implementations like VNPayRefundService. This tight coupling to specific technologies makes the controller harder to test and less flexible. ISP RejectOrderController violates the ISP because it may depend on UI interfaces that expose more methods than necessary for a single use case.	Hồ Bảo Thư	Class "RejectOrderController"	- Create abstractions RefundService interface - Separating UI responsibility into RejectOrderInterface
8	SRP: - The Cart handles both storing products and checking product availability (checkAvailabilityOfProduct).	Nguyễn Lan Nhi	Class "Cart"	Move the availability-checking logic to a separate service: ProductAvailabilityService.
9	SRP: - This class handles too many responsibilities. OCP: - Changes to the logic for calculating shipping fees, validation, or inventory checking would require modifying the class — violating OCP.	Nguyễn Lan Nhi	Class "PlaceRushOrderController"	Split logic into dedicated classes: - RushOrderFormValidator (handles validateForm) - ShippingCalculator - RushOrderService (handles validate isRushOrder)
10	SRP: - This class handles order approval, sending confirmation emails, and sending rejection emails — which means more than one reason to change. Control Coupling: - The approval logic may rely on the orderStatus flag — introducing control flow dependence.	Nguyễn Lan Nhi	Class "OrderManagementControll er"	Split logic into dedicated classes: - OrderEmailService (handles sendApprove/RejectEmail)
11	SRP: - The class handles both data and the logic to update the status to Approved or Rejected.	Nguyễn Lan Nhi	Class "Order"	Split logic into dedicated class: - OrderStatusUpdaterService
12	SRP Violation: This class is handling multiple responsibilities: Product validation; Product creation/persistence; Error display	Đặng Văn Nhân	Class "CreateProductController"	Create three separate classes: 1. CreateProductController - coordinates the product creation flow 2. DefaultProductValidationService - handles all validation 3. ConsoleErrorHandler - handles error display
13	OCP Violation: The validation logic in validateSpecificFields() uses a factory to get a validator, which is good. However, adding new validation rules requires modifying this class.	Đặng Văn Nhân	Class "CreateProductController"	1. Create a ValidationRule interface that specific rules can implement 2. Use a composite pattern with CompositeProductValidator that contains all rules
14	DIP Violation: The controller directly depends on concrete implementations: It directly creates validators through the factory and it has a hard dependency on ProductRepository	Đặng Văn Nhân	Class "CreateProductController"	1. Define IProductRepository interface 2. Define IProductValidator interface 3. Define IErrorHandler interface 4. Inject implementations via constructor
15	SRP: This class is handling too much responsibilities from product validation to access database,...	Đặng Văn Nhân	Class "UpdateProductController"	Create separate classes: - UpdateProductController (to get request from UI and call specific services) - ProductValidator (check validity) - UpdatePolicyChecker (check quota 30/day) - ProductService (get, update product)
16	OCP: If we want to change the logic about changing quotas/day for update products, we need to change this class too	Đặng Văn Nhân	Class "UpdateProductController"	Create interface IProductLimitChecker and inject to controller.
17	DIP: This class calls "Product" class to check logic => depends on specific class, not abstraction	Đặng Văn Nhân	Class "UpdateProductController"	Use interface + service layer to solve: create ProductService
18	SRP: - This class handles too many responsibilities.	Trần Cao Phong	Class "CancelOrderController"	Split logic into dedicated classes: - CancelOrderHandleService (handle logic) - EmailService