Patrones de Software y Programación - Laboratorio 02

Ferrán Rojas Andreu 21642668-1 y Maximo Sarno Jiménez 21853202-0.

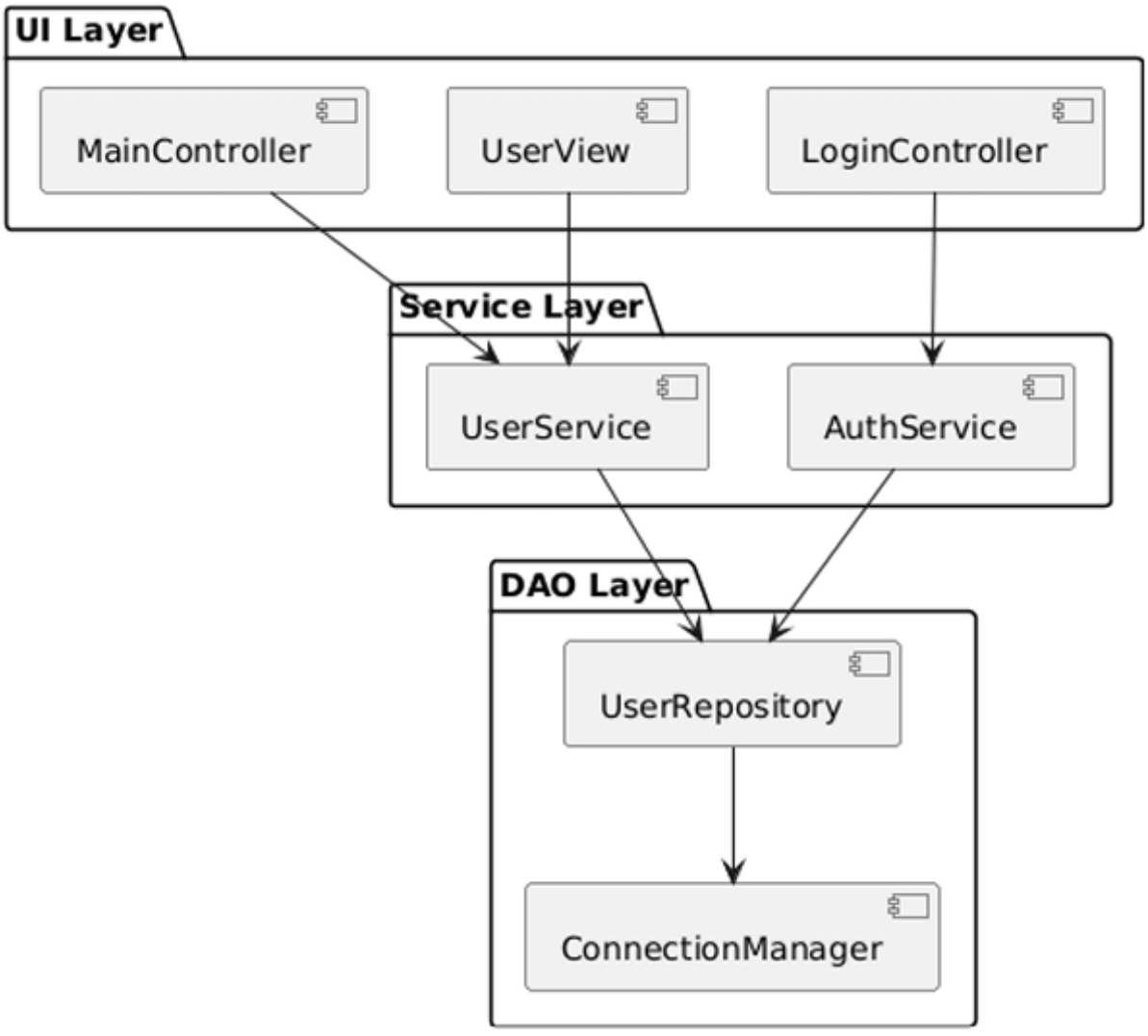
Construcción de una herramienta automática de análisis estático que permita verificar si un sistema Java, basada en el principio de capas, cumple con la arquitectura planeada.

La arquitectura planeada se conforma de 3 capas principales

- Capa de Presentación (UI): contiene controladores que interactúan con los usuarios.
- Capa de Servicios (Service): define la lógica del negocio.
- Capa de Persistencia (DAO): se encarga del acceso a datos.

Las reglas de dependencia son las siguientes:

1. Las clases en UI sólo pueden acceder a clases en Service.
2. Las clases en Service sólo pueden acceder a clases en DAO.
3. No se permiten dependencias directas entre UI y DAO.
4. No se permiten dependencias cruzadas entre clases dentro de una misma capa.



Setup y consideraciones de arquitectura

Para garantizar el cumplimiento de las reglas de dependencia y mantener una arquitectura limpia y desacoplada, se han aplicado una serie de estrategias que se explican a continuación.

Estructura del Proyecto

El código fuente se organiza bajo el paquete raíz layers, subdividido en tres capas principales:

```
java
├──ui       (capa de presentación)
├──service  (capa de lógica de negocio)
└──dao      (capa de acceso a datos)
```

Cada clase concreta está ubicada en su propio subpaquete de su capa, por ejemplo:

```
java.ui.LoginController
java.service.AuthService
java.dao.UserRepository
```

Para el apartado de testing y validación de dependencias el código se organiza bajo el directorio test, el cual contiene dos clases:

```
test
├──CrossDependencyDetector.java
└──LayersRules.java
```

- **CrossDependencyDetector**: Define las reglas necesarias para que no existan dependencias cruzadas entre paquetes y clases.
- **LayersRules**: Define las reglas necesarias para cumplir la arquitectura en capas UI → Service → DAO.

Validación de dependencias y ciclos

Para garantizar una correcta separación de responsabilidades y una arquitectura en capas limpia, se aplicaron reglas de validación sobre las dependencias entre capas y también dentro de cada capa individual. Esto se logró utilizando la librería **ArchUnit** y **Java Parser**, que permite definir y verificar reglas arquitectónicas sobre el código fuente.

Reglas de acceso entre capas

Se aplicaron reglas arquitectónicas con ArchUnit para validar el cumplimiento estricto de una arquitectura en capas. Las reglas implementadas son las siguientes:

UI solo puede acceder a si mismo y a Service (además de clases del JDK):

```
classes().that().resideInAPackage("..ui..")
        .should().onlyAccessClassesThat()
        .resideInAnyPackage("..ui..", "..service..", "java..", "javax..");
```

Service solo puede acceder a si mismo y a DAO (además de clases del JDK):

```
classes().that().resideInAPackage("..service..")
        .should().onlyAccessClassesThat()
        .resideInAnyPackage("..service..", "..dao..", "java..", "javax..");
```

UI no debe acceder directamente a DAO:

```
noClasses().that().resideInAPackage("..ui..")
        .should().accessClassesThat().resideInAPackage("..dao..");
```

DAO no debe acceder a Service:

```
noClasses().that().resideInAPackage("..dao..")
        .should().accessClassesThat().resideInAPackage("..service..");
```

DAO no debe acceder a UI:

```
noClasses().that().resideInAPackage("..dao..")
       .should().accessClassesThat().resideInAPackage("..ui..");
```

Estas reglas aseguran que:

- La comunicación fluye solo en un sentido **descendente** (UI → Service → DAO).
- No existen dependencias indebidas entre capas (UI ← Service ← DAO, UI ← DAO o UI → DAO).
- Se mantiene una separación clara de responsabilidades.

Validación de ciclos entre capas

Se definió una regla general que analiza la estructura completa de los paquetes del sistema dentro de java.(*)... Esta regla permite detectar ciclos de dependencia entre las tres capas del sistema: ui, service y dao.

```
@ArchTest
static final ArchRule no_cross_dependencies_in_layers =
       slices().matching("(dao|service|ui)..")
               .should().beFreeOfCycles()
               .because("No debe haber dependencias cíclicas entre las capas DAO,
Service y UI");
```

Con esta regla se asegura que:

- **UI no dependa de DAO directamente.**
- **Service no acceda a clases de UI o DAO de forma circular.**
- **Cada capa respete las restricciones de acceso de la arquitectura.**

Esta regla técnicamente es redundante, pues las reglas de acceso entre capas detectarían estas dependencias no permitidas. Sin embargo, es bueno como un paso de verificación doble.

Validación de ciclos dentro de una misma capa

Aunque se permite que clases dentro de una misma capa se comuniquen entre sí, no se deben formar ciclos de dependencia entre ellas. Esto es fundamental para mantener un diseño modular y fácil de mantener. Un ciclo ocurre cuando una clase depende (directa o indirectamente) de otra que eventualmente vuelve a depender de la primera.

El validador detecta estos ciclos, los normaliza para evitar repeticiones, y reporta:

- **Las clases involucradas en el ciclo.**
- **La línea exacta y tipo de uso en que ocurre cada dependencia.**

Este funciona de la siguiente manera. Para cada capa:

- Se recorre el directorio correspondiente (src/main/java/ui, por ejemplo).
- Se construye un grafo de dependencias donde:
  - Cada nodo representa una clase.
  - Cada arista representa una dependencia entre clases (variable, parameter, objectcreationexpr, field).
  - Se ejecuta una búsqueda en profundidad (DFS) para encontrar ciclos.

Por ejemplo, dentro de la capa UI, se podría detectar un ciclo similar a este:

```
java.ui.MainController → java.ui.UserView → java.ui.LoginController →
java.ui.MainController
```

Esta forma de detectar ciclos de dependencias tiene una particularidad; cada ciclo será detectado una vez por cada clase que forma parte de este. Por esto es necesario realizar una normalización de los ciclos detectados, para evitar reportar el mismo ciclo varias veces. Para esto se rota el ciclo dejando siempre al inicio la clase cuyo nombre es lexicográficamente menor (orden alfabético). Este criterio de orden es arbitrario, pero cumple su función.

Continuando el ejemplo anterior:

```
Ciclo original: MainController → UserView → LoginController (→ MainController)
Ciclo normalizado: LoginController → MainController → UserView (→ LoginController)
```

De esta forma podemos verificar si el ciclo ya fue registrado previamente y evitar información innecesaria en el reporte final.

Resultados del análisis en el proyecto de prueba tras añadir violaciones intencionadas.

Caso 1

```
package ui;
public class UserView {
  private final service.UserService service = new service.UserService();
  // 1. Capa en UI accediendo direactmante en la capa dao
  private final dao.UserRepository repository = new dao.UserRepository();
  public void showUser(String name) {...}
}
```

Clase UserView de la capa UI accediendo directamente a la clase UserService de la capa DAO.

Resultado de test LayersRules.java

```
java.lang.AssertionError: Architecture Violation [Priority: MEDIUM] - Rule 'no
classes that reside in a package 'ui' should access classes that reside in a
package 'dao', because La capa UI no debe depender directamente de la capa DAO'
was violated (1 times):
Constructor <ui.UserView.<init>()> calls constructor <dao.UserRepository.<init>()>
in (UserView.java:5)

    at
com.tngtech.archunit.lang.ArchRule$Assertions.assertNoViolation(ArchRule.java:94)
    at com.tngtech.archunit.lang.ArchRule$Assertions.check(ArchRule.java:86)
    at
com.tngtech.archunit.lang.ArchRule$Factory$SimpleArchRule.check(ArchRule.java:165)
    at
com.tngtech.archunit.junit.internal.ArchUnitTestDescriptor$ArchUnitRuleDescriptor.
execute(ArchUnitTestDescriptor.java:167)
```

```
    at
com.tngtech.archunit.junit.internal.ArchUnitTestDescriptor$ArchUnitRuleDescriptor.
execute(ArchUnitTestDescriptor.java:150)
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1597)
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1597)
```

Caso 2

```
package ui;
public class UserView {
  private final service.UserService service = new service.UserService();
  // 2. Clase UserView accediendo a la clase ConnectionManager
  dao.ConnectionManager connection = new dao.ConnectionManager();
  public void showUser(String name) {...}
}
```

Clase UserView de la capa **UI** accede **directamente** a la clase ConnectionManager de la capa **DAO**.

Resultado de test LayersRules.java

```
java.lang.AssertionError: Architecture Violation [Priority: MEDIUM] - Rule
'classes that reside in a package 'ui' should only access classes that reside in
any package ['ui', 'service', 'java..', 'javax..'], because La capa UI sólo debe
acceder a la capa Service y a clases propias del JDK' was violated (1 times):
Constructor <ui.UserView.<init>()> calls constructor <dao.ConnectionManager.<init>
()> in (UserView.java:5)

    at
com.tngtech.archunit.lang.ArchRule$Assertions.assertNoViolation(ArchRule.java:94)
    at com.tngtech.archunit.lang.ArchRule$Assertions.check(ArchRule.java:86)
    at
com.tngtech.archunit.lang.ArchRule$Factory$SimpleArchRule.check(ArchRule.java:165)
    at
com.tngtech.archunit.junit.internal.ArchUnitTestDescriptor$ArchUnitRuleDescriptor.
execute(ArchUnitTestDescriptor.java:167)
    at
com.tngtech.archunit.junit.internal.ArchUnitTestDescriptor$ArchUnitRuleDescriptor.
execute(ArchUnitTestDescriptor.java:150)
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1597)
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1597)

java.lang.AssertionError: Architecture Violation [Priority: MEDIUM] - Rule 'no
classes that reside in a package 'ui' should access classes that reside in a
package 'dao', because La capa UI no debe depender directamente de la capa DAO'
was violated (1 times):
Constructor <ui.UserView.<init>()> calls constructor <dao.ConnectionManager.<init>
()> in (UserView.java:5)

    at
```

```
com.tngtech.archunit.lang.ArchRule$Assertions.assertNoViolation(ArchRule.java:94)
    at com.tngtech.archunit.lang.ArchRule$Assertions.check(ArchRule.java:86)
    at
com.tngtech.archunit.lang.ArchRule$Factory$SimpleArchRule.check(ArchRule.java:165)
    at
com.tngtech.archunit.junit.internal.ArchUnitTestDescriptor$ArchUnitRuleDescriptor.
execute(ArchUnitTestDescriptor.java:167)
    at
com.tngtech.archunit.junit.internal.ArchUnitTestDescriptor$ArchUnitRuleDescriptor.
execute(ArchUnitTestDescriptor.java:150)
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1597)
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1597)
```

Caso 3

```
package service;
public class UserService {
  private final UserRepository repository = new UserRepository();
  //3. UserService accediendo a la clase LoginController
  ui.LoginController loginController = new ui.LoginController();
  public void registerUser(String name) {...}
  public void getUserInfo(String name) {...}
}
```

Clase UserService de la capa Service accede directamente a la clase LoginController de la capa UI. Se genera además un **ciclo de dependencia entre capas** debido a que originalmente LoginController ya dependía de la clase AuthService del paquete Service.

Resultado de test LayersRules.java

```
java.lang.AssertionError: Architecture Violation [Priority: MEDIUM] - Rule
'classes that reside in a package 'service' should only access classes that reside
in any package ['service', 'dao', 'java..', 'javax..'], because La capa Service
sólo debe acceder a la capa DAO y a clases propias del JDK' was violated (1
times):
Constructor <service.UserService.<init>()> calls constructor <ui.LoginController.
<init>()> in (UserService.java:6)

    at
com.tngtech.archunit.lang.ArchRule$Assertions.assertNoViolation(ArchRule.java:94)
    at com.tngtech.archunit.lang.ArchRule$Assertions.check(ArchRule.java:86)
    at
com.tngtech.archunit.lang.ArchRule$Factory$SimpleArchRule.check(ArchRule.java:165)
    at
com.tngtech.archunit.junit.internal.ArchUnitTestDescriptor$ArchUnitRuleDescriptor.
execute(ArchUnitTestDescriptor.java:167)
    at
com.tngtech.archunit.junit.internal.ArchUnitTestDescriptor$ArchUnitRuleDescriptor.
```

```
execute(ArchUnitTestDescriptor.java:150)
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1597)
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1597)

java.lang.AssertionError: Architecture Violation [Priority: MEDIUM] - Rule 'slices
matching '(dao|service|ui)..' should be free of cycles, because No debe haber
dependencias cíclicas entre las capas DAO, Service y UI' was violated (1 times):
Cycle detected: Slice service ->
                Slice ui ->
                Slice service
  1. Dependencies of Slice service
    - Field <service.UserService.loginController> has type <ui.LoginController> in
(UserService.java:0)
    - Constructor <service.UserService.<init>()> calls constructor
<ui.LoginController.<init>()> in (UserService.java:6)
  2. Dependencies of Slice ui
    - Field <ui.LoginController.authService> has type <service.AuthService> in
(LoginController.java:0)
    - Field <ui.MainController.userService> has type <service.UserService> in
(MainController.java:0)
    - Field <ui.UserView.service> has type <service.UserService> in
(UserView.java:0)
    - Constructor <ui.UserView.<init>()> calls constructor <service.UserService.
<init>()> in (UserView.java:3)
    - Constructor <ui.LoginController.<init>()> calls constructor
<service.AuthService.<init>()> in (LoginController.java:6)
    - Constructor <ui.MainController.<init>()> calls constructor
<service.UserService.<init>()> in (MainController.java:6)
    - Method <ui.MainController.run()> calls method
<service.UserService.registerUser(java.lang.String)> in (MainController.java:9)
    - Method <ui.UserView.showUser(java.lang.String)> calls method
<service.UserService.getUserInfo(java.lang.String)> in (UserView.java:10)
    - Method <ui.LoginController.login(java.lang.String, java.lang.String)> calls
method <service.AuthService.authenticate(java.lang.String, java.lang.String)> in
(LoginController.java:11)

    at
com.tngtech.archunit.lang.ArchRule$Assertions.assertNoViolation(ArchRule.java:94)
    at com.tngtech.archunit.lang.ArchRule$Assertions.check(ArchRule.java:86)
    at
com.tngtech.archunit.lang.ArchRule$Factory$SimpleArchRule.check(ArchRule.java:165)
    at
com.tngtech.archunit.library.dependencies.SliceRule.check(SliceRule.java:81)
    at
com.tngtech.archunit.junit.internal.ArchUnitTestDescriptor$ArchUnitRuleDescriptor.
execute(ArchUnitTestDescriptor.java:167)
    at
com.tngtech.archunit.junit.internal.ArchUnitTestDescriptor$ArchUnitRuleDescriptor.
execute(ArchUnitTestDescriptor.java:150)
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1597)
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1597)
```

Caso 4

```
package ui;
public class UserView {
  private final service.UserService service = new service.UserService();
  //3. Dependencia cruzada dentro de la capa ui accediendo a la clase
MainController
  ui.MainController mainController = new ui.MainController();
  public void showUser(String name) {...}
}
```

```
package ui;
import service.UserService;
public class MainController {
  private final UserService userService = new UserService();
  //3. Dependencia cruzada en capa ui accediendo a la clase LoginController
  ui.LoginController loginController = new ui.LoginController();
  public void run() {...}
}
```

```
package ui;
import service.AuthService;
public class LoginController {
  private final AuthService authService = new AuthService();
  //3. Dependencia cruzada en capa ui accediendo a la clase UserView
  ui.UserView userView = new ui.UserView();
  public boolean login(String username, String password) {...}
}
```

Dentro de la capa UI se detecta un ciclo de dependencias entre las clases MainController, UserView y LoginController.

Resultado de test CrossDependenciesTest.java

```
🔍 Buscando ciclos en capa: DAO
☑ Sin ciclos detectados en la capa dao.

🔍 Buscando ciclos en capa: SERVICE
☑ Sin ciclos detectados en la capa service.

🔍 Buscando ciclos en capa: UI
⚠  Ciclos encontrados en ui:
  - java.ui.LoginController -> java.ui.UserView -> java.ui.MainController ->
java.ui.LoginController
     ❌ Uso de java.ui.UserView en LoginController.java:7 (variabledeclarator)
     ❌ Uso de java.ui.UserView en LoginController.java:7 (objectcreationexpr)
     ❌ Uso de java.ui.MainController en UserView.java:6 (variabledeclarator)
     ❌ Uso de java.ui.MainController en UserView.java:6 (objectcreationexpr)
     ❌ Uso de java.ui.LoginController en MainController.java:7
(variabledeclarator)
     ❌ Uso de java.ui.LoginController en MainController.java:7
(objectcreationexpr)
```

Conclusión

La validación con **ArchUnit** y **Java Parser** permitió automatizar la verificación de las reglas arquitectónicas, lo que ayuda a prevenir violaciones futuras conforme el sistema **evolucione**.