Estrategias para desacoplar funcionalidades y facilitar el desarrollo de Software en Instituciones Públicas: El caso de la Universidad Nacional de Río Negro

Patricio Nicolas Castro, Mauro Germán Cambarieri

Universidad Nacional de Rio Negro. Sede Atlántica
Laboratorio de Informática Aplicada
{pcastro, mcambarieri}@unrn.edu.ar

RESUMEN

Este proyecto tiene como propósito explorar y adoptar la arquitectura Vertical Slice Architecture (VSA), cuyo enfoque prioriza la organización del código en "slices" o segmentos verticales orientados a funcionalidades específicas. La VSA se presenta como una alternativa para instituciones públicas, al ofrecer un diseño centrado en el usuario y en los casos de uso en la que mejora significativamente la claridad, mantenibilidad y escalabilidad de los sistemas. A diferencia de las arquitecturas tradicionales basadas en capas horizontales, este modelo permite que cada funcionalidad se implemente de manera integral dentro de cada segmento, ya que abarca desde la interfaz de usuario hasta la capa de persistencia de datos. Este enfoque asegura una alineación directa con los procesos institucionales y la misión estratégica de la organización.

La VSA se caracteriza por su capacidad para encapsular todos los componentes necesarios (lógica de negocio, interfaces y acceso a datos) dentro de cada slice vertical. Esta modularidad facilita el aislamiento funcional, lo que permite realizar refactorizaciones, pruebas unitarias y de integración y reducir el riesgo de interferir con sistemas legados u otras funcionalidades del sistema. Tal cualidad resulta particularmente relevante en entornos públicos, donde la coexistencia con tecnologías heredadas y procesos burocráticos rígidos exige soluciones que minimicen el riesgo operativo durante actualizaciones o migraciones. Adicionalmente, la arquitectura admite un diseño escalable: funcionalidades simples se resuelven con estructuras simples, mientras que casos de uso complejos incorporan capas de abstracción progresivas, esto aumenta su flexibilidad y favorece la adaptabilidad ante cambios normativos o técnicos, recurrentes en el sector público.

Un contraste crítico con las arquitecturas multicapa tradicionales radica en la reducción de dependencias transversales. En modelos horizontales, componentes como bases de datos o servicios compartidos suelen generar acoplamientos que dificultan puntualmente las modificaciones. La VSA, en cambio, mitiga este riesgo al limitar el alcance de los cambios a slices específicos.

El trabajo también explora patrones arquitectónicos complementarios para optimizar la VSA, que incluyen Transaction Script, CQRS y Mediator. Estos modelos refuerzan la cohesión dentro de cada slice. También, se realiza una comparativa con la Arquitectura Hexagonal, se discuten ventajas, desventajas, inclusive la curva de aprendizaje en cada caso. Al finalizar se valida mediante un caso de estudio referido a un sistema de información de la Universidad Nacional de Río Negro con código de para ilustrar su implementación.

Palabras clave: Arquitectura de software, Transformación digital, Vertical Slice Architecture, Arquitectura Hexagonal, Mediator, CQRS.

CONTEXTO

En el contexto del Gobierno Digital, se emplean lenguajes de modelado específicos del dominio para representar el conocimiento del dominio y las reglas de negocio de las soluciones de software. Esto permite una mayor precisión y comprensión del dominio gubernamental y facilita el desarrollo de código (OMG, 2014) [1]. Esto brinda beneficios como la detección temprana de errores, la capacidad de realizar análisis y simulaciones y la mejora de la comunicación entre expertos en el dominio y desarrolladores de software.

La Arquitectura de software conforma la columna vertebral de cualquier sistema y constituye uno de sus principales atributos de calidad [2]. La Arquitectura de Software, es definida como “la

organización fundamental de un sistema, encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución" [3]. Distintos ejemplos como la Arquitectura Hexagonal - también conocida como Puertos y Adaptadores-, desarrollada por Alistair Cockburn [4], luego la Arquitectura de Datos, Contexto e Interacción, (DCI por sus siglas en inglés Data, Context and Interaction) de James Coplien y Trygve Reenskaug [5] son muy similares, con el mismo objetivo, centrarse en el dominio específico y, dividir el software en capas. Esto facilita la mantenibilidad e independencia de los frameworks, es decir, que la arquitectura no depende de la existencia de una librería o biblioteca de software. Testeable - en este caso, la Arquitectura puede probar las reglas del negocio sin la interfaz de usuario, base de datos, servidor web o, cualquier otro elemento externo. Independiente de la Interfaz de Usuario - el componente de la Arquitectura puede cambiar algo fácilmente, sin cambiar el resto del sistema. Una interfaz de usuario web podría ser reemplazada por una consola, por ejemplo, sin cambiar las reglas del negocio. Independiente de la base de datos - dado que las reglas de negocio no están ligadas a la base de datos, es posible cambiar el motor de base de datos. Independiente de cualquier agente externo - las reglas de negocio no conocen en absoluto de las interfaces con el mundo exterior.

En este escenario, los recursos tecnológicos se presentan tanto como facilitadores esenciales y herramientas de innovación para mejorar la calidad, el desarrollo de software en el ámbito público.

El presente trabajo se encuentra en el marco del Proyecto de Investigación 40-C-875 "Herramientas Informáticas de Dominio Específico para el Desarrollo de Servicios Digitales Innovadores para Comunidades Urbanas y Rurales en el Marco de Ciudades y Regiones Inteligentes" desarrollado en el Laboratorio de Informática Aplicada, Sede Atlántica, y la dirección de Sistemas de Rectorado de la Universidad Nacional del Río Negro (UNRN).

1. INTRODUCCIÓN

La Vertical Slice Architecture (VSA) es un enfoque de diseño que prioriza la organización del código en función de las características del negocio, en lugar de las capas técnicas tradicionales. Cada "slice" contiene todos los componentes necesarios para implementar una funcionalidad específica, desde la interfaz de usuario hasta el acceso a datos [6].

Esta arquitectura facilita la organización del proceso de desarrollo, ya que abarca tanto la codificación como las pruebas. La VSA implica implementar una porción de funcionalidad tras otra, de manera independiente y donde cada una abarca todas las capas del stack, desde la interfaz de usuario y la lógica de la aplicación hasta el almacenamiento de datos [7].

VSA se enfoca en los casos de uso y la interacción del usuario con el sistema [8]. Cada módulo implementa al menos un caso de uso, pero el enfoque no define detalladamente cómo realizar los cortes verticales, lo que permite adaptabilidad según las necesidades del proyecto y la experiencia del equipo Esto también permite elegir un diseño arquitectónico diferente por corte [8]. Como sugiere Bogard (2020) mientras que los casos de uso simples y directos podrían no requerir un diseño interno complejo, los casos de uso más grandes y complicados pueden estructurarse de una manera más completa. No solo permite la adaptación a las circunstancias actuales, sino también una refactorización más fácil después de, por ejemplo, añadir más casos de uso a un corte y cuando su diseño requiere más consideración.

A continuación se muestra un diagrama que muestra un corte vertical, Figura 1.

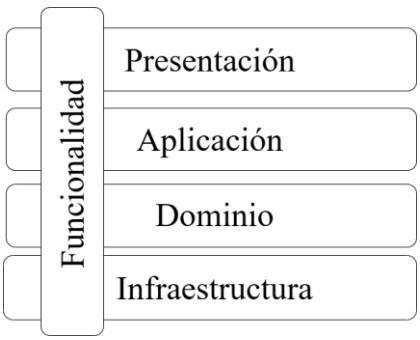

Figura 1. Vertical Slice Architecture. Adaptada de Vertical Slice Architecture [6].

Como ya se mencionó VSA propone dividir por funcionalidad, en lugar de por cuestiones técnicas. Uno de los beneficios de esta división es permitir que cada Request tenga un caso de uso distinto.

Esto nos lleva a aplicar el patrón CQRS [9], este es un patrón de diseño de software que separa la lógica de las aplicaciones entre las operaciones de escritura y las de lectura. O dicho de otra forma, las operaciones que producen cambios en el sistema y las operaciones que hacen consultas.

Otro patrón que se utiliza en esta arquitectura es Mediator, un patrón de diseño de comportamiento que se encuentra en el catálogo de GoF [10] y se encarga de controlar cómo interactúan los objetos entre sí. Aunque VSA no indique el uso de CQRS o Mediator, estos patrones fluyen bien juntos [11]. El objetivo de usar mediator es desacoplar las piezas entre sí. Mediator actúa como intermediario entre para la comunicación de los objetos se con la ayuda del mediator y ayuda a romper ayuda a romper el vínculo estrecho entre los colaboradores [11].

En el contexto de VSA, es utilizado Transaction Script, que organiza la lógica empresarial por procedimientos que maneja una única solicitud de la presentación. Esto implica obtener una entrada de datos, realizar algún procesamiento y una respuesta o persistir un cambio en el sistema. Por lo tanto, es un patrón simple que facilita el desarrollo inicial de una característica y permite que la arquitectura evolucione dada las necesidades del dominio.

El patrón Transaction Script organiza la lógica de un caso de uso como un único procedimiento que realiza llamadas directamente a la base de datos o a través de una capa delgada. Este patrón es básico y se utiliza cuando el comportamiento es simple, como obtener o guardar datos sin necesidad de un gran modelo de dominio [12]. Aplica cuando no necesitamos una gran complejidad y alcanza con asegurar que obtenemos los datos deseados para generar la respuesta.

Esto nos da la posibilidad de agregar complejidad solo cuando es necesario, cuando debemos establecer reglas y límites para proteger las invariantes y asegurar la consistencia de los cambios. Las validaciones de datos pueden ocurrir en el handler, y las validaciones que corresponden con reglas de negocio e invariantes tendrían que ocurrir en el handler. Lo que nos lleva a modelar el dominio, y representar reglas de comportamiento que se deben cumplir [13].

VSA se contrapone a arquitecturas como la arquitectura hexagonal, que busca independizar el núcleo de la aplicación de las dependencias externas mediante el uso de puertos y adaptadores. Mientras que la arquitectura hexagonal se enfoca en la independencia tecnológica y la testabilidad, VSA además busca la cohesión y la facilidad de desarrollo específico.

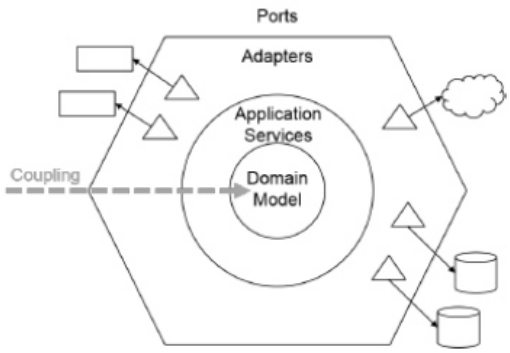

Fig. 2. Arquitectura Hexagonal de A. Cockburn [14]

La Arquitectura VSA se caracteriza por organizar el código por funcionalidad, donde cada slice contiene los componentes necesarios para implementar una funcionalidad específica. Esto resulta en un bajo acoplamiento entre slices y una alta cohesión dentro de cada uno. Por otro lado, la Arquitectura Hexagonal separa la aplicación entre la lógica de negocio y los puertos que definen las interacciones con el mundo exterior, lo que logra un fuerte desacoplamiento del núcleo de la aplicación respecto a tecnologías concretas.

En términos de complejidad, la VSA es generalmente más simple de entender e implementar, mientras que la Hexagonal puede ser más abstracta y compleja, requiere una comprensión clara del diseño del sistema. En cuanto a la abstracción, la VSA evita abstracciones prematuras, favorece la simplicidad y la evolución del código, mientras que la Hexagonal introduce abstracciones a través de puertos que definen interfaces para los adaptadores.

La flexibilidad en la VSA se manifiesta en el diseño de cada slice, adaptándose a las necesidades específicas de cada funcionalidad. En cambio, la Hexagonal ofrece alta flexibilidad para cambiar tecnologías externas, aunque puede ser menos adaptable a requisitos específicos de cada caso de uso.

En cuanto a la refactorización, la VSA requiere un esfuerzo continuo para evitar la duplicación de código, mientras que la Hexagonal facilita este proceso al permitir el intercambio de adaptadores sin afectar al núcleo. Finalmente, para los aspectos transversales como autenticación y logging, la VSA requiere consideraciones adicionales, mientras que la Hexagonal los maneja a través de adaptadores, lo que permite una implementación centralizada.

Finalmente, en VSA la curva de aprendizaje inicial puede ser más suave, ya que el enfoque en funcionalidades concretas facilita la comprensión del código, lo que requiere de equipos menos expertos. Sin embargo, a medida que el proyecto crece, es crucial tener un buen conocimiento de refactoring para evitar la duplicación de código y mantener la cohesión.

En cambio, en la arquitectura hexagonal, la curva de aprendizaje inicial puede ser más pronunciada, ya que requiere comprender los conceptos de puertos y adaptadores, patrones de diseño y la separación entre la aplicación y el mundo exterior. Sin embargo, una vez dominados estos conceptos, la arquitectura hexagonal proporciona una base sólida para construir aplicaciones testeables y mantenibles. Además, requiere de equipos relativamente expertos. A continuación, se mencionan las ventajas y desventajas de cada una.

En VSA se identifican las siguientes ventajas: (1) Facilidad de desarrollo y mantenimiento: Los cambios se limitan a un slice, reduce los side effects (secundarios o colaterales) y reduce el "context switching". (2) Testabilidad: Los slices aislados permiten pruebas unitarias y de integración sencillas. (3) Curva de aprendizaje más rápida: Los desarrolladores pueden ser productivos más rápidamente al centrarse en una sola característica.

En VSA se identifican las siguientes desventajas: (1) Duplicación de código: Si no se refactoriza adecuadamente, riesgo de código

duplicado en diferentes slices. (2) Complejidad general: A medida que crecen las features, puede volverse compleja si no se gestiona correctamente.

En la Arquitectura Hexagonal se identifican las siguientes ventajas: (1) Independencia tecnológica: Permite cambiar fácilmente las tecnologías externas (bases de datos, frameworks) sin afectar el núcleo de la aplicación. (2) Testabilidad: Facilita las pruebas unitarias al poder simular los adaptadores externos. (3) Mantenibilidad: Promueve un código limpio y modular, que facilita el mantenimiento a largo plazo.

En la Arquitectura Hexagonal se identifican las siguientes desventajas: (1) Complejidad inicial: Requiere una inversión inicial en diseño y comprensión de los conceptos de puertos y adaptadores. (2) Sobrediseño: Puede ser excesivamente abstracto si no se aplica con criterio.

El siguiente código de ejemplo está basado en un proyecto referido a un sistema de información de la UNRN. El mismo es un caso de uso para agregar un usuario, que muestra cómo implementar la arquitectura VSA con Spring Boot y Java, y aplica el patrón mediator.

```
@RestController
@RequestMapping("/command-users")
class UserAPICommand {

    JMediator mediator;

    public UserAPICommand(JMediator mediator) {
        this.mediator = mediator;
    }

    @PostMapping
    @ResponseStatus(CREATED)
    @Operation(summary = "Crear una cuenta", security
    void add(@RequestBody CreateUserCommand command)
        mediator.send(command);
    }

}
```

Fig. 3. Endpoint para agregar un usuario

```
public record CreateUserCommand(
        UUID id,
        String name,
        String lastname,
        String email,
        String username,
        String password

        ) implements IRequest<Void> {
}
```

Fig. 4. Objeto DTO del usuario

```
@Handler
public class CreateUserHandler {
    private final UsersResource usersResource;

    public CreateUserHandler(UsersResource usersResource) {
        this.usersResource = usersResource;
    }

    void create(CreateUserCommand command) {
        var representation = new UserRepresentation();
        representation.setEmail(command.email());
        representation.setUsername(command.email());
        representation.setFirstName(command.name());
        representation.setLastName(command.lastname());
        representation.setAttributes(Map.of("password", List.of(cc
        representation.setRequiredActions(List.of(UPDATE_PASSWORD.
        representation.setEnabled(true);
        try (var res = usersResource.create(representation)) {
            if (!Objects.equals(res.getStatus(), 200)) {
                throw new EmptyException("Error creating user");
            }
        } catch (Exception e) {
            throw new EmptyException("Error creating user");
        }
    }
}
```

Fig. 5. Lógica del caso de uso agregar usuario

```
@Validator
class createUserValidator {
    void validate(CreateUserCommand command) {
        if (!usersResource.searchByEmail(command.email(), true).isEmpty()) {
            throw new EmailAlreadyExistsException();
        }
        if (!usersResource.searchByUsername(command.username(), true).isEmpty()) {
            throw new UsernameAlreadyExistsException();
        }
    }
}
```

Fig. 6. Validaciones de datos

2. LÍNEAS DE INVESTIGACIÓN Y DESARROLLO

El objetivo principal de la línea de investigación aquí presentada es:

Diseñar e implementar estrategias basadas en una arquitectura de vertical slice para facilitar el desarrollo de software de calidad en instituciones públicas, promover la modularidad, el desacoplamiento de funcionalidades y la escalabilidad de las soluciones tecnológicas.

Sus ejes específicos son: (a) Estudiar el estado del arte de enfoques modernos de desarrollo de software, que incluye la arquitectura de vertical slice, el testing automatizado (TDD - Desarrollo Guiado por Pruebas) y las prácticas de refactoring continuo, para identificar mejores prácticas aplicables al contexto de instituciones públicas, (b) Evaluar marcos de trabajo y herramientas que combinen la arquitectura de vertical slice con técnicas de testing automatizado y refactoring, con el fin mejorar la calidad del software y facilitar su mantenimiento y evolución en el contexto de instituciones públicas, (c) Diseñar estrategias de testing automatizado y prácticas de refactoring continuo dentro del ciclo de vida del desarrollo que se integren con la arquitectura de vertical slice, con el objetivo de mejorar la calidad del código, reducir

la deuda técnica y mantener la modularidad y el desacoplamiento de las funcionalidades, (d) Desarrollar un caso de estudio en el que se apliquen técnicas de testing automatizado (como TDD) y refactoring en un proyecto basado en vertical slice, con el fin de validar su efectividad en el contexto de instituciones públicas, se estudiará en el contexto de la Universidad Nacional de Río Negro, (e) Desarrollar un caso de estudio en un proyecto basado en vertical slice, que aplique técnicas de testing y refactoring, con el fin de validar su efectividad en el contexto de instituciones públicas, (f) Diseñar una arquitectura de referencia que permita su instanciación para la aplicación de los enfoques propuestos y promueva la reutilización sistemática de componentes y la evolución constante del software, (g) Desarrollar y validar prototipos de software, pasibles de ser implementados y transferidos al medio.

3. RESULTADOS OBTENIDOS/ ESPERADOS

Como resultado de este proyecto, se espera identificar arquitecturas de software, metodologías y modelos conceptuales para el desarrollo de software en dominios complejos, en busca de priorizar la modularidad y la evolución de los sistemas mediante el enfoque de Vertical Slice Architecture (VSA). Desde el punto de vista arquitectónico, el diseño basado en VSA posibilitará la creación de regiones o capas desacopladas, lo que favorecerá su evolución de manera independiente y sin estar restringidas por tecnologías específicas. Este modelo contribuirá significativamente a la modernización de sistemas legados y ofrecerá una alternativa flexible para la transformación digital de instituciones públicas.

Se espera que el software cumpla con lo siguiente:

- Alta flexibilidad y adaptación a cambios en reglas de negocio, ya que cada funcionalidad podrá evolucionar de manera aislada.
- Mayor mantenibilidad y modularidad, al estructurar el código en el que abarcan toda el "stack" tecnológico de cada funcionalidad,.
- Independencia tecnológica, permitir integrar diferentes tecnologías dentro de cada "slice".
- Capacidad de reutilización de componentes.
- Aumento en la calidad del software debido a la mayor facilidad en la prueba e implementación de nuevas funcionalidades
- Reducir la deuda técnica y mejorar la eficiencia en el desarrollo de software en el contexto de las instituciones públicas.
- Promover la confianza en las soluciones tecnológicas desarrolladas para instituciones públicas.

4. FORMACIÓN DE RECURSOS HUMANOS

El grupo de trabajo se encuentra integrado por un investigador formado, un investigador en formación y dos alumnos avanzados de la carrera Licenciatura en Sistemas. En su marco se desarrollarán prácticas profesionales supervisadas, becas de formación práctica y se producirán dos trabajos finales de carrera de grado y la elaboración de trabajo de tesis de la carrera de Maestría en Ingeniería de Software de la UNLP.

5. BIBLIOGRAFÍA

[1] OMG. (2014). OMG Systems Modeling Language (OMG SysML™) Version 1.4. Object Management Group

[2] Clements, P., et al, "Software Architecture in Practice", Pearson Education, (2003).

[3] IEEE Standards Association, "1471-2000 - IEEE Rec ommended Practice for Architectural Description for Software-Intensive Systems", available at: http://standards.ieee.org/findstds/standard/1471-2000.html

[4] Alistair Cockburn. Alistair.Cockburn.us, Hexagonal architecture. https://alistair.cockburn.us/hexagonal-architecture/ (2023)

[5] James O Coplien, Trygve Mikkjel Heyerdahl Reenskaug: "The data, context and interaction paradigm. Proceedings of the 3rd annual conference on Systems, programming, and applications: software for humanity." October 2012 Pages 227–228 https://doi.org/10.1145/2384716.2384782

[6] Bogard, J. (2018). Vertical slice architecture. Jimmy Bogard. http://jimmybogard.com/vertical-slice-architecture

[7] Esposito, D. (2024). Clean architecture with .NET. Addison Wesley.

[8] J. Bogard, NDC Porto Software Developers Conference. "Vertical Slice Architecture." Porto, 24.04.2020. [Online]. Available: Vertical Slice Architecture - Jimmy Bogard https://www.youtube.com/watch?v=5kOzZz2vj2o

[9] M. Fowler. CQRS. http://martinfowler.com/bliki/CQRS.html, 2011.

[10] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Reading, MA: Addison-Wesley, 1995.

[11] Marcotte, C. H., & Zebdi, A. (2022). An Atypical ASP. NET Core 6 Design Patterns Guide: A SOLID adventure into architectural principles and design patterns using. NET 6 and C# 10. Packt Publishing Ltd.

[12] M. Fowler. Transaction Script. http://martinfowler.com/eaaCatalog/transactionScript.html, 2003

[13] Cambarieri, M., Difabio, F., & García Martínez, N. (2020). Implementación de una Arquitectura de Software guiada por el Dominio. In XXI Simposio Argentino de Ingeniería de Software (ASSE 2020)-JAIIO 49 (Modalidad virtual).

[14]. A. Cockburn, "The Pattern: Ports and Adapters," 2005: disponible en: https://alistair.cockburn.us/hexagonal-architecture/ [accedido: 18/08/2020]