netmind | a bts company

Modulo 4: Testing, Refactorizacion, Documentacion y DevOps SDD

Microsoft Copilot en el Entorno de Desarrollo Local

4.5 horas | 270 minutos

Seccion 0: Introduccion y Prerrequisitos

Establecer contexto del workflow de calidad SDD

15 minutos

Prerrequisitos del Workflow de Calidad

Del Modulo 2, ya se han ejecutado:

- `/speckit.specify` → specs/[feature]/spec.md
- `/speckit.plan` → specs/[feature]/plan.md
- `/speckit.tasks` → specs/[feature]/tasks.md
- `/speckit.generate` → codigo en src/

Del Modulo 3, ya existen:

- .specify/templates/[lenguaje]/ para diferentes tecnologias

Este modulo usa:

- `/speckit.analyze` → Validar calidad del codigo generado
- `/speckit.validate` → Verificar cumplimiento de specs

Artefactos SDD Disponibles

Artefacto	Ubicacion	Proposito
`spec.md`	`specs/[feature]/`	Especificacion funcional con criterios de aceptacion
`code/`	`src/`	Codigo implementado
`tests/`	`tests/`	Tests unitarios y de integracion
`templates/`	`.specify/templates/`	Templates de test por tipo

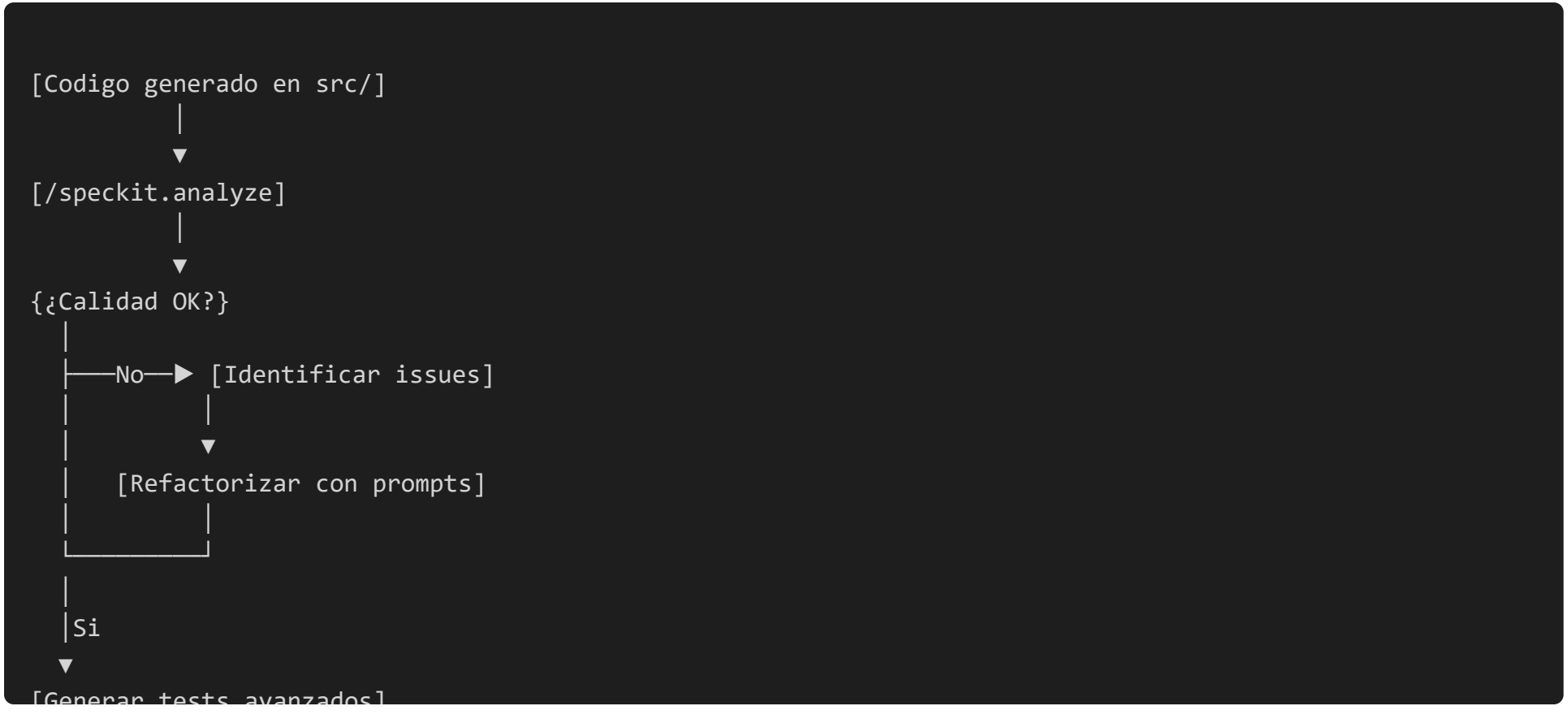
Diferencia con Modulos Anteriores

Modulo 2	Modulo 3	Modulo 4
Workflow general SDD	Codigo especifico multi-lenguaje	Calidad post-impl
specify, plan, tasks	generate, analyze	analyze, validate

Enfoque:

- M2: Que construir?
- M3: Como construirlo?
- M4: Como asegurar calidad?

Workflow de Calidad con Spec Kit



4.1 Generacion de Pruebas Avanzadas y Casos Limite

Dominar tecnicas avanzadas de testing usando Copilot y Spec Kit

90 minutos

4.1.1 Introduccion al Testing Avanzado SDD

La generacion de pruebas avanzadas en SDD representa la culminacion del proceso de desarrollo donde cada especificacion se valida exhaustivamente contra criterios de aceptacion medibles.

Dimensiones del Testing Avanzado:

- Property-Based Testing - Propiedades invariantes
- Mutation Testing - Efectividad de tests
- Performance Testing - Requisitos no funcionales
- Concurrency Testing - Acceso paralelo
- E2E Testing - Flujos completos

Rol de Copilot en Testing

Integracion de Copilot transforma la creacion de pruebas:

ANTES (Manual)	DESPUES (Copilot)
Escribir cada test	Generar suites completas
Identificar casos limite	Especificar caracteristicas
Mantener cobertura	Validar criterios automatica

El tester define:

- Caracteristicas del codigo a probar
- Criterios de aceptacion a validar
- Escenarios limite a considerar
- Framework de testing preferido

4.1.2 Pruebas Unitarias Avanzadas SDD

Property-Based Testing con FsCheck

Framework: FsCheck para C# o Hypothesis para Python
Concepto: En lugar de casos especificos, definimos PROPIEDADES que deben ser verdaderas para TODOS los valores validos.
Propiedades a validar:

- 1. Login con credenciales validas siempre retorna token JWT valido
- 2. Password hasheado nunca puede revertirse al texto plano
- 3. Tokens generados tienen expiracion consistente

Ejemplo Property-Based Test

```
using FsCheck;
using FluentAssertions;
using Xunit;

public class AuthenticationPropertyTests
{
    [Property]
    public void Login_ValidCredentials_AlwaysReturnsValidJwt(
        string email, string password)
    {
        // Arrange
        var dto = new LoginRequestDto
        {
            Email = ValidEmail(email),
            Password = ValidPassword(password)
        };

        // Act
        var result = _authService.LoginAsync(dto
```

Mutation Testing con Stryker

Herramienta: Stryker para .NET
Concepto: Mutar el codigo automaticamente y verificar que los tests detectan los cambios. Si un mutation sobrevive, los tests no son suficientemente robustos.

```
{
  "stryker-config": {
    "project-file": "PortalEmpleo.Tests.csproj",
    "reporters": ["html", "json"],
    "thresholds": {
      "high": 80,
      "low": 60,
      "break": 0
    }
  }
}
```

Performance Testing con NBomber

Requisitos No Funcionales:

- Tiempo de respuesta API: < 200ms (p95)
- Throughput: > 1000 requests/segundo

```
[Fact]
public async Task Login_Under_200ms_p95()
{
    var scenario = Scenario.Create("login_scenario",
        async context =>
        {
            var stopwatch = Stopwatch.StartNew();
            var response = await _httpClient.PostAsJsonAsync(
                "/api/v1/auth/login",
                new LoginRequestDto { Email = "test@test.com",
                    Password = "Test123!" });
            stopwatch.Stop();
            return Response.Ok(latencyMs:
                stopwatch.ElapsedMilliseconds);
        })
    .WithWarmUpDuration(TimeSpan.FromSeconds(30))
    .WithDuration(TimeSpan.FromMinutes(5));

    var result = NBomberRunner.RegisterScenarios(scenario).Run();
```

4.1.3 Casos Limite y Escenarios Complejos SDD

Tests de Manejo de Errores

Escenarios a Testear:

- Usuario no encontrado → 401
- Password incorrecto → 401
- Usuario inactivo → 403
- Rate limiting excedido → 429
- Base de datos no disponible → 503

```
[Fact]
public async Task Login_UsuarioNoExistente_Returns401()
{
    var dto = new LoginRequestDto
    { Email = "nonexistent@portalempleo.com", Password = "any" };
    _mockRepo.Setup(r => r.GetByEmailAsync(
        It.IsAny<string>(), It.IsAny<cancellationtoken>()))
        .ReturnsAsync((Usuario?)null);

    var result = await _handler.Handle(
        new LoginCommand(dto), CancellationToken.None);

    result.IsFailure.Should().BeTrue();
```

```
    result.Error.Should().Contain("Credenciales invalidas");
}

[Fact]
public async Task Login_RateLimitExceeded_Returns429()
{
```

Tests de Concurrency

```
[Fact]
public async Task CreateTarea_SimultaneousRequests_AllSucceed()
{
    var concurrentRequests = 10;
    var tasks = new List<Task<Result<TareaResponseDto>>>();

    for (int i = 0; i < concurrentRequests; i++)
    {
        tasks.Add(_handler.Handle(
            new CreateTareaCommand { Titulo = $"Concurrent {i}" },
            CancellationToken.None));
    }

    var results = await Task.WhenAll(tasks);
    results.All(r => r.IsSuccess).Should().BeTrue();
    results.Select(r => r.Value.Id).Distinct().Count()
        .Should().Be(concurrentRequests);  // Sin duplicados
}
```

4.1.4 Testing Automatizado del Ciclo Completo SDD

Integration Tests Entre Servicios

```
public class TareaIntegrationTests :
    IClassFixture<WebApplicationFactory<Program>>
{
    [Fact]
    public async Task FullWorkflow_CreateCompleteAndListTarea()
    {
        // 1. Login
        var loginResponse = await _client.PostAsJsonAsync(
            "/api/v1/auth/login",
            new LoginRequestDto { Email = "test@test.com",
                Password = "Test123!" });
        loginResponse.StatusCode.Should()
            .Be(HttpStatusCode.OK);
        var loginData = await loginResponse.Content
            .ReadFromJsonAsync<LoginResponseDto>();
        _client.DefaultRequestHeaders.Authorization =
            new AuthenticationHeaderValue("Bearer",
                loginData.AccessToken);
```

End-to-End Testing con Playwright

```
test.describe('Tarea Management E2E', () => {
  test('Complete Tarea Workflow', async ({ page }) => {
    // Login
    await page.goto('/login');
    await page.fill('[data-testid=email]',
        'test@portalempleo.com');
    await page.fill('[data-testid=password]', 'Test123!');
    await page.click('[data-testid=login-button]');
    await page.waitForURL('/dashboard');

    // Create
    await page.goto('/tareas');
    await page.click('[data-testid=create-tarea-button]');
    await page.fill('[data-testid=titulo-input]',
        'E2E Test Task');
    await page.selectOption('[data-testid=prioridad-select]',
        'alta');
    await page.click('[data-testid=submit-button]');
    await expect(page.locator(
```

4.1.5 Ejemplo Practico: Template de Test Unitario con Spec Kit

Unit Test Template con Spec Kit

Ubicacion: .specify/templates/tests/unit-test-template.md

```
using FluentAssertions;
using Moq;
using Xunit;

namespace Tests.Unit.Features.[Feature];

public class [Entity]ServiceTests
{
    private readonly [Entity]Service _service;
    private readonly Mock<i[entity]repository> _mockRepo;
    private readonly Mock<ilogger<[entity]service>> _mockLogger;

    public [Entity]ServiceTests()
    {
        _mockRepo = new Mock<i[entity]repository>();
        _mockLogger = new Mock<ilogger<[entity]service>>();
        _service = new [Entity]Service(_mockRepo.Object,
            _mockLogger.Object);
    }
```

Resultado Generado

Test Generado para AuthService

`tests/PortalEmpleo.Tests.Unit/Features/Auth/AuthServiceTests.cs`

```
public class AuthServiceTests
{
    private readonly AuthService _service;
    private readonly Mock<iuserrepository> _mockRepo;
    private readonly Mock<ilogger<authservice>> _mockLogger;

    public AuthServiceTests()
    {
        _mockRepo = new Mock<iuserrepository>();
        _mockLogger = new Mock<ilogger<authservice>>();
        _service = new AuthService(_mockRepo.Object,
            _mockLogger.Object);
    }

    [Fact]
    public async Task Login_ValidCredentials_ReturnsAuthResult()
    {
        var dto = new LoginRequestDto
            { Email = "test@test.com", Password = "Password123!" };
```

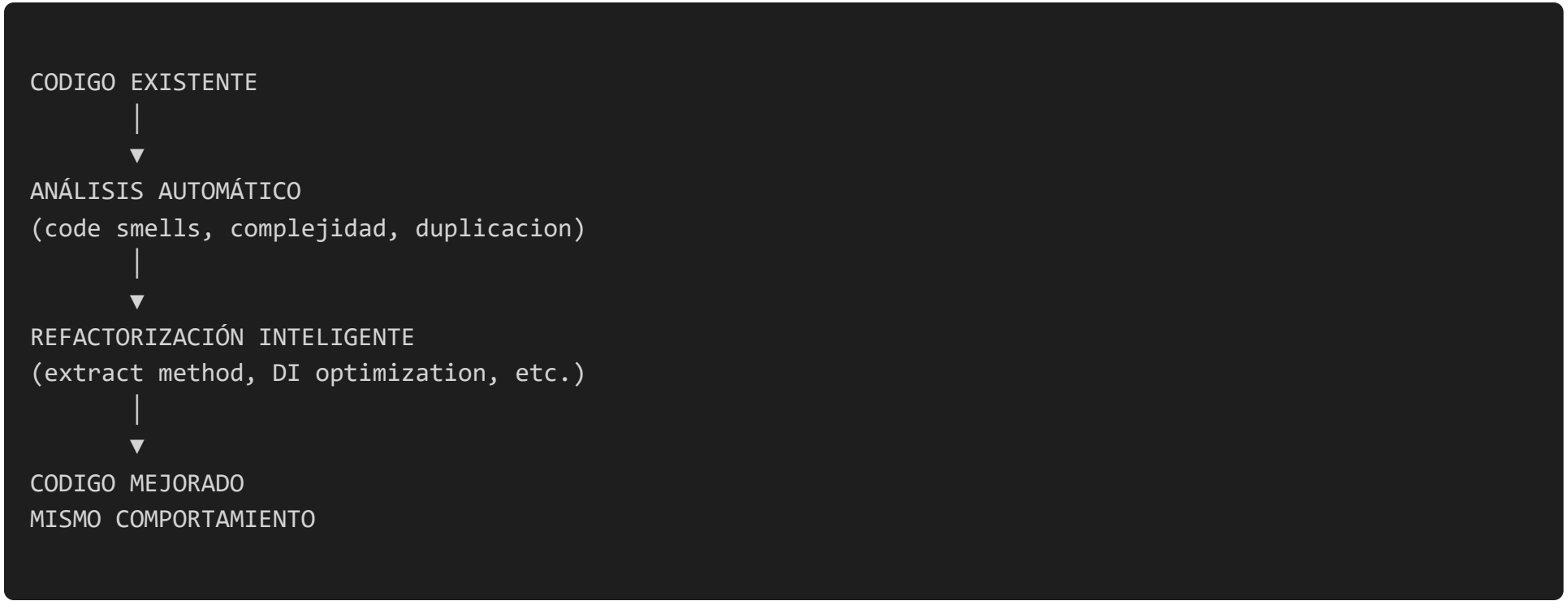
4.2 Refactorizacion Inteligente con Copilot

Mejorar mantenibilidad del codigo sin alterar comportamiento observable

90 minutos

4.2.1 Introduccion a la Refactorizacion SDD

La refactorizacion en SDD es el proceso de mejorar la estructura interna del codigo sin alterar su comportamiento observable, manteniendo trazabilidad con las especificaciones originales.



4.2.2 Analisis Automatico de Codigo SDD

Deteccion de Code Smells

Code Smells a Detectar:

- Long Method - Metodos > 30 lineas
- Large Class - Clases > 500 lineas
- Long Parameter List - > 4 parametros
- Primitive Obsession
- Switch Statements extensos

```
public class CodeSmellDetector
{
    public AnalysisResult AnalyzeProject(string projectPath)
    {
        var files = Directory.GetFiles(projectPath, "*.cs",
            SearchOption.AllDirectories);
        var smells = new List<CodeSmell>();

        foreach (var file in files)
        {
            var content = File.ReadAllText(file);
            var tree = CSharpSyntaxTree.ParseText(content);
            var root = tree.GetRoot();
```

```
foreach (var method in root.DescendantNodes()
    .OfType<MethodDeclarationSyntax>())
{
    var lineCount = method.Span.EndLine -
```

Deteccion de Duplicacion

```
public class DuplicateCodeDetector
{
    public DuplicateReport FindDuplicates(string projectPath,
        int minLines = 6)
    {
        var files = GetCodeFiles(projectPath);
        var duplicates = new List<DuplicateBlock>();
        var blocks = new Dictionary<string, List<CodeBlock>>();

        foreach (var file in files)
        {
            var content = File.ReadAllText(file);
            var lines = content.Split('\n').ToList();

            for (int i = 0; i < lines.Count - minLines; i++)
            {
                var blockContent = string.Join('\n',
                    lines.Skip(i).Take(minLines));
                var key = NormalizeBlock(blockContent);
```

Analisis de Complejidad Ciclomatica

Metricas de Complejidad:

- Complejidad 1-10: Codigo simple
- Complejidad 11-20: Moderado
- Complejidad 21-50: Complejo
- Complejidad > 50: Muy alto riesgo

```
public class ComplexityAnalyzer
{
    public ComplexityReport Analyze(string projectPath)
    {
        var methods = new List<MethodComplexity>();
        var files = Directory.GetFiles(projectPath, "*.cs");

        foreach (var file in files)
        {
            var content = File.ReadAllText(file);
            var tree = CSharpSyntaxTree.ParseText(content);
            var root = tree.GetRoot();

            foreach (var method in root.DescendantNodes()
                .OfType<MethodDeclarationSyntax>())
            {
                var complexity = 1;
```

```
            complexity += method.DescendantNodes().Count(n =>
```

4.2.3 Refactoring Patterns Asistidos SDD

Extract Method Refactoring

CODIGO ORIGINAL (60+ lineas):

```
public async Task<Result<LoginResponseDto>> LoginAsync(
    LoginRequestDto dto, CancellationToken ct)
{
    var usuario = await _repo.GetByEmailAsync(dto.Email, ct);
    if (usuario == null) return Result<LoginResponseDto>
        .Failure("Invalid");
    if (!usuario.EstaActivo) return Result<LoginResponseDto>
        .Failure("Inactive");
    if (!VerifyPassword(dto.Password, usuario.PasswordHash))
        return Result<LoginResponseDto>.Failure("Invalid");
    var token = _tokenService.GenerateAccessToken(usuario);
    var refreshToken = await _tokenService
        .GenerateRefreshTokenAsync(usuario.Id, ct);
    usuario.UltimoLogin = DateTime.UtcNow;
    await _repo.UpdateAsync(usuario, ct);
    _logger.LogInformation("User logged in", usuario.Email);
    return Result<LoginResponseDto>.Success(new LoginResponseDto
        { AccessToken = token, RefreshToken = refreshToken.Token });
}
```

REFACTORIZADO:

```
public async Task<Result<LoginResponseDto>> LoginAsync(
    LoginRequestDto dto, CancellationToken ct)
{
    var usuarioResult = await ValidateAndGetUserAsync(
        dto.Email, dto.Password, ct);
    if (usuarioResult.IsFailure)
        return Result<LoginResponseDto>.Failure(
            usuarioResult.Error);
    var tokenResult = await GenerateTokensForUserAsync(
        usuarioResult.Value, ct);
    await UpdateUserLoginAsync(usuarioResult.Value, ct);
    LogLoginSuccess(usuarioResult.Value.Email);
    return tokenResult;
}
```

Dependency Injection Optimization

PROBLEMA: Registro manual de 35+ servicios
REFACTORING A ASSEMBLY SCANNING:

```
public static class ServiceCollectionExtensions
{
    public static void RegisterAssemblyServices(
        this IServiceCollection services)
    {
        var assembly = Assembly.Load("PortalEmpleo.Application");

        var registrations = assembly.GetTypes()
            .Where(t => t.IsClass && !t.IsAbstract)
            .SelectMany(t => t.GetInterfaces()
                .Where(i => i.Name.StartsWith('I') &&
                    (i.Name.EndsWith("Service") ||
                     i.Name.EndsWith("Repository") ||
                     i.Name.EndsWith("Validator"))))
            .Select(i => new { Interface = i, Implementation =
                assembly.GetTypes().FirstOrDefault(c =>
                    c.GetInterfaces().Contains(i) &&
                    c.Name == i.Name.Substring(1)) })
            .Where(x => x.Implementation != null);
```

4.2.4 Mejora de Mantenibilidad SDD

Specification Pattern

```
// Specifications individuales
public class TituloValidoSpecification : ISpecification<Tarea>
{
    public bool IsSatisfiedBy(Tarea tarea)
    {
        return !string.IsNullOrWhiteSpace(tarea.Titulo) &&
               tarea.Titulo.Length >= 1 &&
               tarea.Titulo.Length <= 200;
    }
    public string ErrorMessage =>
        "Titulo debe tener entre 1 y 200 caracteres";
}

public class PrioridadValidaSpecification :
    ISpecification<Tarea>
{
    public bool IsSatisfiedBy(Tarea tarea)
    {
        return tarea.Prioridad != null &&
```

4.2.5 Ejemplo Practico: Template de Refactoring con Spec Kit

Refactoring Resultado con Spec Kit

Metrics de mejora:

Metrica	Antes	Despues	Reduccion
Complejidad ciclomatica	42	8	-81%
Lineas de codigo	45	12	-73%
Numero de metodos	1	5	+400% (mejorado)

4.3 Documentacion Viva y DevOps con IA

Automatizar documentacion e integracion DevOps con quality gates

90 minutos

4.3.1 Documentacion Tecnica Automatica SDD

API Documentation from Code

```
/// <summary>
/// API Controller para gestion de tareas del sistema PortalEmpleo.
/// </summary>
/// <remarks>
/// Provides CRUD operations for task management including:
/// - Creating new tasks with validation
/// - Listing tasks with filtering and pagination
/// - Updating and completing tasks
/// </remarks>
[ApiController]
[Route("api/v1/tareas")]
[Produces("application/json")]
public class TareasController : ControllerBase
{
    /// <summary>
    /// Obtiene lista paginada de tareas
    /// </summary>
    /// <param name="query">Parametros de filtrado</param>
    /// <returns>Lista paginada de tareas</returns>
```

Architecture Decision Records Automaticos

```
# ADR-001: Uso de JWT para Autenticacion

## Estado
**Aceptado** | Propuesto | Deprecated

## Contexto
Necesitamos elegir un mecanismo de autenticacion para la API.

## Decision
Usaremos JWT para autenticacion stateless.

### Factores de Decision
1. **Escalabilidad**: JWT es stateless
2. **Performance**: Eliminamos round-trips a BD
3. **Mobile support**: JWT funciona bien con apps moviles

## Consecuencias
### Positivas
- No se requiere servidor de sesiones
```

4.3.2 Integracion con Azure DevOps SDD

Code Reviews Asistidos por Copilot

@code-review-agent Criterios:

- **Trazabilidad**: Cada codigo tiene requisito SDD vinculado
- **Calidad**: Codigo cumple estandares de AGENTS.md
- **Tests**: Tests cubren criterios de aceptacion
- **Seguridad**: No exposicion de datos sensibles

```
## Review: [PR Title]
### Cambios
- Archivos modificados: X
- Lineas añadidas: +X
- Lineas eliminadas: -X
### Trazabilidad SDD
- Requisitos cubiertos: X/Y
- Tests añadidos: Z
### Issues Encontrados
| Severidad | Archivo | Linea | Issue |
|-----------|---------|-------|-------|
| Alta | AuthService.cs | 45 | Validacion incompleta |
| Media | TareaController.cs | 120 | Naming convention |
```

Quality Gates Automatizados

Quality Gates:

- Coverage >= 80%
- No new code smells high severity
- Todos los tests pasando
- Cumplimiento de specs

```
jobs:
  quality-gate:
    steps:
      - name: Validate Quality Gates
        script: |
          # Coverage Gate
          COVERAGE=$(cat coverage-report/coverage-summary.json |
            jq '.line_coverage')
          if (( $(echo "$COVERAGE < 80" | bc -l) )); then
            echo "Coverage $COVERAGE% below 80% threshold";
            exit 1;
          fi

          # Spec Compliance Gate
          SPEC_VIOLATIONS=$(cat spec-analysis/analyze-result.json
            | jq '.violations | length')
          if [ "$SPEC_VIOLATIONS" -gt 0 ]; then
```

```
        echo "Spec violations found";
```

4.3.3 Monitoreo y Observabilidad SDD

Application Performance Monitoring

```
public class TelemetryTracker : ITelemetryTracker
{
    private readonly TelemetryClient _telemetryClient;

    public void TrackRequest(string operationName,
        TimeSpan duration, bool success)
    {
        _telemetryClient.TrackEvent("RequestCompleted",
            new Dictionary<string, string> {
                ["OperationName"] = operationName,
                ["Success"] = success.ToString()
            });
        _telemetryClient.TrackMetric(
            $"RequestDuration.{operationName}",
            duration.TotalMilliseconds);
    }
}

// Middleware de tracking
```

Alertas Inteligentes

Configuracion de alertas:

```
{
  "alerts": [
    { "name": "HighResponseTime",
      "metric": "request.duration.p95",
      "threshold": 200,
      "operator": ">",
      "severity": "Warning" },
    { "name": "HighErrorRate",
      "metric": "request.errors.rate",
      "threshold": 0.05,
      "operator": ">",
      "severity": "Critical" }
  ]
}
```

4.3.4 Ejemplo Practico: Pipeline CI/CD con Quality Gates

Pipeline CI/CD Completo

Stages:

1. Build Stage
2. Test Stage
3. Quality Gate Stage
4. Deploy Stage (si pasa quality gates)

```
{
  "qualityGates": [
    { "name": "Coverage",
      "metric": "line_coverage",
      "threshold": 80,
      "operator": ">=",
      "blocking": true },
    { "name": "Spec Compliance",
      "metric": "spec_violations",
      "threshold": 0,
      "operator": "==",
      "blocking": true },
```

```
  { "name": "Complexity",
    "metric": "max_cyclomatic_complexity",
    "threshold": 20,
    "operator": "<=",
    "blocking": false }
]
```

Resumen del Modulo 4

Lo Aprendido

5 minutos

Conceptos Clave Aprendidos

- Property-Based Testing con FsCheck/Hypothesis
- Mutation Testing con Stryker
- Performance Testing con NBomber
- Exception Handling y Concurrency Tests
- Integration Tests y E2E Tests
- Deteccion de Code Smells y Duplicacion
- Analisis de Complejidad Ciclomatica
- Extract Method/Class Refactoring
- Dependency Injection Optimization
- Specification Pattern
- Documentacion Automatica con OpenAPI
- Architecture Decision Records (ADR)
- Code Reviews Asistidos
- Quality Gates Automatizados
- Application Performance Monitoring
- Pipeline CI/CD con Azure DevOps

Gracias

Módulo 4 Completado