netmind | a bts company

Módulo 2

Microsoft Copilot en el Ciclo de Vida SDD

Integración de Copilot y Spec Kit en las fases de diseño y desarrollo

Prerrequisitos y Contexto

Workflow Spec Kit para el Módulo 2

Artefactos SDD Disponibles

Artefacto	Ubicación	Propósito
`constitution.md`	`.specify/memory/`	Principios, estándares y convenciones del proyecto
`spec.md`	`specs/[feature]/`	Especificación funcional con requisitos y criterios de aceptación
`plan.md`	`specs/[feature]/`	Plan técnico con decisiones de arquitectura
`tasks.md`	`specs/[feature]/`	Tareas discretas para implementación

Arquitectura de Dos Niveles para Prompts SDD

Nivel	Ubicación	Propósito
Nivel 1: Templates Globales	`.specify/templates/`	Prompts reutilizables en cualquier proyecto
Nivel 2: Prompts por Feature	`specs/[feature]/`	Adaptaciones concretas para una feature específica

Arquitectura de Dos Niveles para Prompts SDD

Nivel	Ubicación	Propósito
Nivel 1: Templates Globales	`.specify/templates/`	Prompts reutilizables en cualquier proyecto
Nivel 2: Prompts por Feature	`specs/[feature]/`	Adaptaciones concretas para una feature específica

Beneficios de Guardar Prompts

- **Trazabilidad:** Saber qué prompt generó cada artefacto
- **Reutilización:** Prompts probados en features similares
- **Colaboración:** Todo el equipo usa los mismos prompts
- **Auditoría:** Demonstrar cómo se generó el código

Sección 1

Fase de Diseño y Arquitectura con SDD

Introducción al Diseño SDD

Introducción al Diseño SDD

- Las especificaciones de alto nivel se transforman en decisiones técnicas concretas

Introducción al Diseño SDD

- Las especificaciones de alto nivel se transforman en decisiones técnicas concretas
- Vinculación estricta entre diseño y especificaciones previas

Introducción al Diseño SDD

- Las especificaciones de alto nivel se transforman en decisiones técnicas concretas
- Vinculación estricta entre diseño y especificaciones previas
- Trazabilidad bidireccional entre decisiones y requisitos

Introducción al Diseño SDD

- Las especificaciones de alto nivel se transforman en decisiones técnicas concretas
- Vinculación estricta entre diseño y especificaciones previas
- Trazabilidad bidireccional entre decisiones y requisitos
- Copilot como arquitecto asistente que genera propuestas

Introducción al Diseño SDD

- Las especificaciones de alto nivel se transforman en decisiones técnicas concretas
- Vinculación estricta entre diseño y especificaciones previas
- Trazabilidad bidireccional entre decisiones y requisitos
- Copilot como arquitecto asistente que genera propuestas
- Integración con Spec Kit para flujo hacia implementación

Introducción al Diseño SDD

- Las especificaciones de alto nivel se transforman en decisiones técnicas concretas
- Vinculación estricta entre diseño y especificaciones previas
- Trazabilidad bidireccional entre decisiones y requisitos
- Copilot como arquitecto asistente que genera propuestas
- Integración con Spec Kit para flujo hacia implementación

Dónde guardar prompts de diseño:

`specs/[feature]/design-prompts.md` o directamente en `plan.md`

Generación de Diagramas de Flujo

La generación de diagramas de procesos es el primer paso en la fase de diseño.

````
"Genera un diagrama de flujo Mermaid para el proceso de creación de tareas en PortalEmpleo:

## Especificación de Referencia
- Archivo: specs/001-task-api/spec.md
- Requisito: REQ-TASK-001 (Crear tarea)
- Criterios de aceptación: CA-TASK-001, CA-TASK-002

## Proceso de Negocio
1. Usuario autenticado accede a pantalla de creación
2. Sistema muestra formulario con campos
3. Usuario completa formulario y envía
4. Sistema valida datos
5. Sistema crea registro en base de datos
6. Sistema asigna ID único
7. Sistema retorna confirmación

## Formato de Salida
```mermaid
graph TD
````
````

Diagrama de Arquitectura C4

```
"Genera diagrama C4 de arquitectura para PortalEmpleo basado en constitution.md:

## Contexto del Sistema
- PortalEmpleo: Sistema de gestión de tareas empresariales
- Usuarios: ~1000 concurrentes
- SLA: 99.5% disponibilidad, < 200ms respuesta

## Componentes Definidos en Specs
- API Gateway: Punto de entrada
- Auth Service: Autenticación JWT
- Task Service: Gestión de tareas
- User Service: Perfiles de usuario
- SQL Server: Base de datos principal
- Redis: Cache de sesiones

## Patrón Arquitectónico
- Microservicios con API Gateway
- Comunicación síncrona REST
- Base de datos por servicio"
```

Documentación de Decisiones Técnicas (ADR)

Los ADRs capturan decisiones arquitectónicas importantes y su justificación.

```
Genera un ADR para la decisión de arquitectura de datos en PortalEmpleo:

## Decisión a Documentar
- Usar Entity Framework Core con SQL Server como ORM primario
- Implementar Repository Pattern con Unit of Work

## Contexto
- Proyecto existente con legacy code en ADO.NET
- Equipo familiarizado con Entity Framework
- Requisito de migración gradual desde sistema legacy

## Opciones Consideradas
1. Continue with ADO.NET (status quo)
2. Dapper + repositories
3. Entity Framework Core + repositories

## Decisión Seleccionada
Entity Framework Core + repositories
```

Validación de Arquitectura

La validación asegura que las decisiones cumplan con los requisitos no funcionales.

```
"Valida la siguiente propuesta de arquitectura contra constitution.md:

## Propuesta de Arquitectura
- API: REST sobre HTTPS (TLS 1.3)
- Database: SQL Server 2022 en Azure
- Auth: JWT con refresh tokens (30 min / 7 días)
- Deployment: Azure App Service con auto-scaling

## Requisitos No Funcionales de constitution.md
- APIs responden en < 200ms (p95)
- Todas las APIs requieren JWT de Microsoft Entra ID
- Datos cifrados in transit (TLS 1.2+) y at rest (AES-256)
- Audit log completo de cambios

## Matriz de Cumplimiento
| Requisito NF | Propuesta | Cumple | Gap |
|--------------|-----------|--------|-----|
| TLS 1.3 | TLS 1.3 | ✓ | - |
| JWT Entra ID | JWT custom | X | Migrar a Entra ID |
```

Esqueletos de Proyectos C#

```
"Genera estructura completa de proyecto ASP.NET Core 8 para PortalEmpleo API:

## Requisitos Técnicos
- .NET 8, C# 12
- Clean Architecture
- Entity Framework Core 8
- SQL Server 2022
- xUnit para testing
- FluentValidation

## Estructura Requerida
src/
├── PortalEmpleo.API/
│   ├── Controllers/
│   ├── Middleware/
│   ├── Filters/
│   ├── Program.cs
│   └── appsettings.json
├── PortalEmpleo.Application/
```

Program.cs Generado

```
using PortalEmpleo.Infrastructure.Data;
using PortalEmpleo.Infrastructure.Identity;
using PortalEmpleo.Application;
using PortalEmpleo.Infrastructure;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddApplication();
builder.Services.AddInfrastructure(builder.Configuration);
builder.Services.AddApiServices();

var app = builder.Build();

if (app.Environment.IsDevelopment())
{
    app.UseDeveloperExceptionPage();
}

app.UseApiServices();
```

Estructura de Notebooks Python

```
Genera estructura de proyecto Python para análisis de productividad:

## Stack Tecnológico
- Python 3.12
- Jupyter Notebooks
- Pandas, NumPy, Scikit-learn
- MLflow para tracking
- Plotly para visualización
- FastAPI para serving

## Estructura Requerida
notebooks/
├── 01-data-exploration.ipynb
├── 02-feature-engineering.ipynb
├── 03-model-training.ipynb
├── 04-model-evaluation.ipynb
├── 05-prediction-api.ipynb
└── 06-dashboard-analysis.ipynb
```

Estructura React Native SDD

```
Genera estructura de proyecto React Native con Expo para PortalEmpleo Mobile:

## Stack Tecnológico
- React Native con Expo 51
- TypeScript 5
- React Query para data fetching
- React Navigation 6
- Zustand para state management

## Estructura Requerida
src/
├── components/
│   ├── common/
│   ├── forms/
│   └── layout/
├── screens/
│   ├── auth/
│   ├── tasks/
│   └── home/
```

Generación Automática desde AGENTS.md

```
@agent Configura AGENTS.md para generación automática de esqueletos:

## Requerimiento
Cuando el agente detecte creación de nuevo feature,
deberá generar automáticamente la estructura completa.

## Patrón de Detección
- Directorio nuevo en specs/
- Nuevo spec.md con feature name
- Trigger: creación de primer archivo en src/[Feature]/

## Template a Usar
- Template: .specify/templates/feature-skeleton.md
- Ubicación: src/[FeatureName]/
- Archivos: controller, service, repository, dto, validator, tests
```

Convenciones de Nomenclatura C#

```
// Entidades: Entity suffix
public class TareaEntity { }

// DTOs: Purpose + DTO suffix
public class TareaCreateDto { }
public class TareaResponseDto { }

// Commands/Queries (MediatR)
public record CreateTareaCommand : IRequest<tarearesponsedto> { }

// Services: Interface + Implementation
public interface ITareaService { }
public class TareaService : ITareaService { }

// Métodos CRUD estándar
Tarea GetById(Guid id);
List<tarea> GetAll();
Tarea Create(TareaCreateDto dto);
Tarea Update(Guid id, TareaUpdateDto dto);
```

Patrones de Organización de Código

Patrón: Feature-First (C#)

```
src/Features/
└── Tasks/
    ├── Controllers/
    ├── Services/
    ├── Repositories/
    ├── DTOs/
    ├── Validators/
    ├── Entities/
    └── Tests/
```

Patrones de Organización de Código

Patrón: Feature-First (C#)

```
src/Features/
└── Tasks/
    ├── Controllers/
    ├── Services/
    ├── Repositories/
    ├── DTOs/
    ├── Validators/
    ├── Entities/
    └── Tests/
```

Patrón: DDD Layered

Patrones de Organización de Código

Patrón: Feature-First (C#)

```
src/Features/
└── Tasks/
    ├── Controllers/
    ├── Services/
    ├── Repositories/
    ├── DTOs/
    ├── Validators/
    ├── Entities/
    └── Tests/
```

Patrón: DDD Layered

```
src/
├── Domain/           # Enterprise logic
├── Application/      # Use cases
├── Infrastructure/   # Frameworks
└── API/              # Entry points

## Reglas de Dependencia
- Domain → No dependencias
- Application → Solo Domain
- Infrastructure → Application y Domain
- API → Application
```

Configuración de Testing SDD

```
Genera configuración de testing para PortalEmpleo:

## Frameworks
- Unit Testing: xUnit + Moq + FluentAssertions
- Integration Testing: ASP.NET Core TestHost
- E2E: Playwright (Web) / Detox (Mobile)
- Coverage: ReportGenerator + Coverlet

## Estructura de Tests
tests/
├── PortalEmpleo.Tests.Unit/
│   ├── Features/
│   │   └── Tasks/
│   └── Application/
├── PortalEmpleo.Tests.Integration/
│   ├── API/
│   └── Data/
└── PortalEmpleo.Tests.E2E/
    ├── SpecFlow/
```

Ejemplo Práctico: Diagrama C4

Caso de estudio: PortalEmpleo - Task Management API

Dónde se guarda:

- Prompt: `specs/001-task-api/design-prompts.md`
- Diagrama: `docs/diagrams/architecture-c4.md`

Ejemplo Práctico: Diagrama C4

Caso de estudio: PortalEmpleo - Task Management API

Dónde se guarda:

- Prompt: `specs/001-task-api/design-prompts.md`
- Diagrama: `docs/diagrams/architecture-c4.md`

Integración con Spec Kit:

1. Copiar prompt de `design-prompts.md`
2. Ejecutar con `@backend-dev`
3. Guardar resultado en `docs/diagrams/`
4. Validar con `/speckit.analyze`

Sección 2

Fase de Desarrollo Asistido con SDD

Introducción al Desarrollo SDD

Introducción al Desarrollo SDD

- Las especificaciones se transforman en código funcional

Introducción al Desarrollo SDD

- Las especificaciones se transforman en código funcional
- Prompts incluyen contexto completo del proyecto

Introducción al Desarrollo SDD

- Las especificaciones se transforman en código funcional
- Prompts incluyen contexto completo del proyecto
- Trazabilidad entre código y requisitos

Introducción al Desarrollo SDD

- Las especificaciones se transforman en código funcional
- Prompts incluyen contexto completo del proyecto
- Trazabilidad entre código y requisitos
- Flujo: task → spec review → implement → validate → document

Introducción al Desarrollo SDD

- Las especificaciones se transforman en código funcional
- Prompts incluyen contexto completo del proyecto
- Trazabilidad entre código y requisitos
- Flujo: task → spec review → implement → validate → document
- Integración con Spec Kit mediante `/speckit.implement`

Introducción al Desarrollo SDD

- Las especificaciones se transforman en código funcional
- Prompts incluyen contexto completo del proyecto
- Trazabilidad entre código y requisitos
- Flujo: task → spec review → implement → validate → document
- Integración con Spec Kit mediante `/speckit.implement`

Dónde guardar prompts de desarrollo:

En `tasks.md` (columna "Prompt") o en `.specify/templates/code-prompts.md`

Transformación de Specs a Código

```
Implementa el endpoint POST /api/tareas basado en specs/001-task-api/spec.md:

## Especificación de Referencia
- Sección: 2.1.1 POST /api/tareas
- Requisito: REQ-TASK-001
- Criterios de aceptación: CA-TASK-001, CA-TASK-002, CA-TASK-003

## Especificación del Endpoint
POST /api/tareas
Content-Type: application/json

Request Body:
{
  "titulo": "string (1-200 chars)",
  "descripcion": "string (0-2000 chars)",
  "fechaLimite": "ISO8601 datetime",
  "prioridad": "baja|media|alta|urgente"
}
```

DTO y Validator Generados

```
public class CreateTareaDto
{
    [Required]
    [StringLength(200, MinimumLength = 1)]
    public string Titulo { get; set; }

    [StringLength(2000)]
    public string? Descripcion { get; set; }

    public DateTime? FechaLimite { get; set; }

    [RegularExpression("baja|media|alta|urgente")]
    public string Prioridad { get; set; } = "media";

    public List<string>? Etiquetas { get; set; }
}

public class CreateTareaValidator : AbstractValidator<createtareadto>
{
```

Controller Action Generado

```
[HttpPost]
[ProducesResponseType(typeof(TareaResponseDto), StatusCodes.Status201Created)]
[ProducesResponseType(typeof(ValidationProblemDetails), StatusCodes.Status400BadRequest)]
public async Task<iactionresult> Create(
    [FromBody] CreateTareaDto dto,
    CancellationToken ct)
{
    var result = await _tareaService.CreateTareaAsync(dto, GetCurrentUserId(), ct);
    return CreatedAtAction(nameof(GetById), new { id = result.Id }, result);
}</iactionresult>
```

Repository Pattern con EF Core

```
Implementa el repositorio de tareas con Entity Framework Core:

## Interfaz Requerida
public interface ITareaRepository
{
    Task<tarea?> GetByIdAsync(Guid id, CancellationToken ct);
    Task<list<tarea>> GetAllAsync(CancellationToken ct);
    Task<list<tarea>> GetByStatusAsync(TareaStatus status, CancellationToken ct);
    Task<list<tarea>> GetByAssigneeAsync(Guid usuarioId, CancellationToken ct);
    Task<tarea> AddAsync(Tarea tarea, CancellationToken ct);
    Task<tarea> UpdateAsync(Tarea tarea, CancellationToken ct);
    Task<bool> DeleteAsync(Guid id, CancellationToken ct);
}

## Requisitos de Implementación
1. Usar AsNoTracking para queries de solo lectura
2. Incluir navegación de UsuarioAsignado en GetById
3. Soft delete pattern (campo IsDeleted)
4. Concurrency token para updates
```

TareaRepository Implementado

```
public class TareaRepository : ITareaRepository
{
    private readonly PortalEmpleoDbContext _context;

    public TareaRepository(PortalEmpleoDbContext context)
    {
        _context = context;
    }

    public async Task<tarea?> GetByIdAsync(Guid id, CancellationToken ct)
    {
        return await _context.Tareas
            .Include(t => t.UsuarioAsignado)
            .AsNoTracking()
            .FirstOrDefaultAsync(t => t.Id == id && !t.IsDeleted, ct);
    }

    public async Task<tarea> AddAsync(Tarea tarea, CancellationToken ct)
    {
```

Factory Pattern para Entidades

```
Implementa TareaFactory para creación de entidades Tarea:

## Propósito
Abstraer la lógica de creación de objetos Tarea,
validando reglas de negocio en el constructor.

## Factory Interface
public interface ITareaFactory
{
    Tarea Create(string titulo, Guid creadoPorId, string? descripcion = null);
    Tarea CreateFromDto(CreateTareaDto dto, Guid usuarioId);
}

## Reglas de Negocio en Creación
1. Titulo requerido, trimming automático
2. Estado inicial: TareaStatus.Pendiente
3. Prioridad: default "media" si no especificada
4. FechaLímite: nullable (sin límite si null)
```

TareaFactory Implementado

```
public class TareaFactory : ITareaFactory
{
    public Tarea Create(string titulo, Guid creadoPorId, string? descripcion = null)
    {
        if (string.IsNullOrWhiteSpace(titulo))
            throw new ArgumentException("El título es requerido", nameof(titulo));

        titulo = titulo.Trim();
        if (titulo.Length > 200)
            throw new ArgumentException("El título no puede exceder 200 caracteres", nameof(titulo));

        var tarea = new Tarea
        {
            Id = Guid.NewGuid(),
            Titulo = titulo,
            Descripcion = descripcion?.Trim(),
            Estado = TareaStatus.Pendiente,
            Prioridad = TareaPrioridad.Media,
            CreadoPorId = creadoPorId
```

Observer Pattern con Domain Events

```
Implementa el sistema de eventos de dominio para PortalEmpleo:

## Eventos de Dominio Requeridos
public class TareaCreatedEvent : DomainEvent
{
    public Guid TareaId { get; }
    public Guid CreadoPorId { get; }
    public DateTime CreadoEn { get; }
}

public class TareaCompletedEvent : DomainEvent
{
    public Guid TareaId { get; }
    public Guid CompletadoPorId { get; }
}

// MediatR Integration
public class TareaCreatedEvent : INotification
{
```

Domain Events Implementado

```
public abstract class DomainEvent : INotification
{
    public Guid EventId { get; } = Guid.NewGuid();
    public DateTime OccurredAt { get; } = DateTime.UtcNow;
    public string? CorrelationId { get; set; }
}

public class TareaCreatedEvent : DomainEvent
{
    public Guid TareaId { get; }
    public Guid CreadoPorId { get; }
    public DateTime CreadoEn { get; }

    public TareaCreatedEvent(Guid tareaId, Guid creadoPorId)
    {
        TareaId = tareaId;
        CreadoPorId = creadoPorId;
        CreadoEn = DateTime.UtcNow;
    }
```

Sugerencias Contextuales SDD

Sugerencias Contextuales SDD

- Aprovechan el conocimiento del proyecto en AGENTS.md

Sugerencias Contextuales SDD

- Aprovechan el conocimiento del proyecto en AGENTS.md
- Se activan durante la edición activa del código

Sugerencias Contextuales SDD

- Aprovechan el conocimiento del proyecto en AGENTS.md
- Se activan durante la edición activa del código
- Respetan las convenciones establecidas

Sugerencias Contextuales SDD

- Aprovechan el conocimiento del proyecto en AGENTS.md
- Se activan durante la edición activa del código
- Respetan las convenciones establecidas
- Incluyen contexto de specs y código existente

Sugerencias Contextuales SDD

- Aprovechan el conocimiento del proyecto en AGENTS.md
- Se activan durante la edición activa del código
- Respetan las convenciones establecidas
- Incluyen contexto de specs y código existente

Dónde guardar templates:

En `AGENTS.md` (sección Autocompletado Contextual) y `.specify/templates/completion-templates/`

Función Compleja con Filtros

```
Implementa el método de filtrado avanzado de tareas:

## Firma Requerida
public async Task<pagedresult<tarearesponsedto>> GetTareasAsync(
    TareaFiltersDto filters,
    int page = 1,
    int pageSize = 20,
    string sortBy = "CreadoEn",
    string sortOrder = "desc",
    CancellationToken ct = default)

## Filtros Soportados
public class TareaFiltersDto
{
    public string? Search { get; set; }
    public TareaStatus? Estado { get; set; }
    public TareaPrioridad? Prioridad { get; set; }
    public Guid? AsignadoAId { get; set; }
    public DateTime? FechaLimiteDesde { get; set; }
```

GetTareasAsync Implementado

```
public async Task<pagedresult<tarearesponsedto>> GetTareasAsync(
    TareaFiltersDto filters,
    int page = 1,
    int pageSize = 20,
    string sortBy = "CreadoEn",
    string sortOrder = "desc",
    CancellationToken ct = default)
{
    pageSize = Math.Min(pageSize, 100);

    var currentUserId = _currentUserService.GetCurrentUserId();

    var query = _repository.GetQueryable()
        .Where(t => !t.IsDeleted)
        .Where(t => t.CreadoPorId == currentUserId ||
                    t.AsignadoAId == currentUserId);

    if (!string.IsNullOrWhiteSpace(filters.Search))
    {
```

Implementación de Interfaz ITareaService

```
Implementa la interfaz ITareaService según specs/001-task-api/spec.md:

## Interfaz a Implementar
public interface ITareaService
{
    Task<tarearesponsedto> CreateTareaAsync(CreateTareaDto dto, Guid usuarioId, CancellationToken ct);
    Task<tarearesponsedto?> GetTareaByIdAsync(Guid id, Guid usuarioId, CancellationToken ct);
    Task<pagedresult<tarearesponsedto>> GetTareasAsync(TareaFiltersDto filters, Guid usuarioId, int page, int pageSize, CancellationToken ct);
    Task<tarearesponsedto> UpdateTareaAsync(Guid id, UpdateTareaDto dto, Guid usuarioId, CancellationToken ct);
    Task<bool> DeleteTareaAsync(Guid id, Guid usuarioId, CancellationToken ct);
}

## Reglas de Negocio
1. Solo propietario puede modificar
2. Soft delete implementado
3. Eventos de dominio publicados
4. Auditoría completa de cambios
5. Concurrency handling apropiado</bool></tarearesponsedto></pagedresult<tarearesponsedto></tarearesponsedto?></tarearesponsedto>
```

Referencias Oficiales

Documentación y recursos adicionales

Documentación de Herramientas

Documentación de Herramientas

- **GitHub Copilot Documentation**: https://docs.github.com/en/copilot

Documentación de Herramientas

- **GitHub Copilot Documentation**: https://docs.github.com/en/copilot
- **Spec Kit Repository**: https://github.com/github/spec-kit

Documentación de Herramientas

- **GitHub Copilot Documentation**: https://docs.github.com/en/copilot
- **Spec Kit Repository**: https://github.com/github/spec-kit
- **Microsoft Learn - Spec-Driven Development**: https://learn.microsoft.com/es-es/training/modules/...

Recursos de Patrones y Diagramas

Recursos de Patrones y Diagramas

- **Entity Framework Core**: https://docs.microsoft.com/en-us/ef/core/

Recursos de Patrones y Diagramas

- **Entity Framework Core**: https://docs.microsoft.com/en-us/ef/core/
- **MediatR**: https://github.com/jbogard/MediatR

Recursos de Patrones y Diagramas

- **Entity Framework Core**: https://docs.microsoft.com/en-us/ef/core/
- **MediatR**: https://github.com/jbogard/MediatR
- **Clean Architecture**: https://github.com/ardalis/CleanArchitecture

Recursos de Patrones y Diagramas

- **Entity Framework Core**: https://docs.microsoft.com/en-us/ef/core/
- **MediatR**: https://github.com/jbogard/MediatR
- **Clean Architecture**: https://github.com/ardalis/CleanArchitecture
- **Mermaid Chart Syntax**: https://mermaid.js.org/

Recursos de Patrones y Diagramas

- **Entity Framework Core**: https://docs.microsoft.com/en-us/ef/core/
- **MediatR**: https://github.com/jbogard/MediatR
- **Clean Architecture**: https://github.com/ardalis/CleanArchitecture
- **Mermaid Chart Syntax**: https://mermaid.js.org/
- **C4 Model**: https://c4model.com/

Recursos de Patrones y Diagramas

- **Entity Framework Core**: https://docs.microsoft.com/en-us/ef/core/
- **MediatR**: https://github.com/jbogard/MediatR
- **Clean Architecture**: https://github.com/ardalis/CleanArchitecture
- **Mermaid Chart Syntax**: https://mermaid.js.org/
- **C4 Model**: https://c4model.com/
- **ADR Documentation**: https://github.com/joelparkerhenderson/architecture-decision-record

Resumen del Módulo 2

Conceptos clave y herramientas SDD

Resumen: Conceptos y Herramientas

Sección	Conceptos Clave	Herramientas/Artefactos
2.1 Diseño SDD	Diagramas C4, ADRs, esqueletos de proyecto	Spec Kit design prompts, templates de arquitectura
2.2 Desarrollo SDD	Generación de código con trazabilidad	tasks.md, code-prompts, patrones (Factory, Repository, Observer)
Contexto SDD	Arquitectura de dos niveles de prompts	Templates globales vs prompts por feature

Conceptos Clave del Módulo

Conceptos Clave del Módulo

- Workflow Spec Kit completo: `specify` → `plan` → `tasks` → `implement` → `analyze`

Conceptos Clave del Módulo

- Workflow Spec Kit completo: `specify` → `plan` → `tasks` → `implement` → `analyze`
- Arquitectura de dos niveles para prompts: templates globales y específicos por feature

Conceptos Clave del Módulo

- Workflow Spec Kit completo: `specify` → `plan` → `tasks` → `implement` → `analyze`
- Arquitectura de dos niveles para prompts: templates globales y específicos por feature
- Trazabilidad completa desde specs hasta código implementado

Conceptos Clave del Módulo

- Workflow Spec Kit completo: `specify` → `plan` → `tasks` → `implement` → `analyze`
- Arquitectura de dos niveles para prompts: templates globales y específicos por feature
- Trazabilidad completa desde specs hasta código implementado
- Patrones de diseño SDD: Factory, Repository, Observer, Domain Events

Conceptos Clave del Módulo

- Workflow Spec Kit completo: `specify` → `plan` → `tasks` → `implement` → `analyze`
- Arquitectura de dos niveles para prompts: templates globales y específicos por feature
- Trazabilidad completa desde specs hasta código implementado
- Patrones de diseño SDD: Factory, Repository, Observer, Domain Events
- Integración con AGENTS.md para contexto continuo

Conceptos Clave del Módulo

- Workflow Spec Kit completo: `specify` → `plan` → `tasks` → `implement` → `analyze`
- Arquitectura de dos niveles para prompts: templates globales y específicos por feature
- Trazabilidad completa desde specs hasta código implementado
- Patrones de diseño SDD: Factory, Repository, Observer, Domain Events
- Integración con AGENTS.md para contexto continuo

Próximo paso:

El Módulo 3 profundiza en técnicas avanzadas de generación de código y desarrollo de APIs con Copilot.

Gracias

Módulo 2 Completado

Siguiente: Modulo 3 - Desarrollo de Código Potenciado por AI