Definición de Arquitectura: Marco de Desarrollo Impulsado por Especificaciones (SDD)

Orquestación de flujos de trabajo de desarrollo seguros y basados en especificaciones con LLMs

Marco de Desarrollo Estructurado Impulsado por Especificaciones (SPDD)

Presentación de Arquitectura Técnica – Mayo 2026

1. Filosofía del Sistema: SPDD frente a Alternativas

- **Agilidad y Rigor unidos**: SPDD (Desarrollo Estructurado Impulsado por Prompts) reduce la brecha entre el desarrollo basado en prompts de lenguaje natural libre y las especificaciones manuales de esquemas.
- **Verificación Primero**: Aprovechamiento de especificaciones estructuradas para orquestar ciclos autónomos en entornos aislados (sandbox), preservando la revisión del desarrollador.

Desarrollo por Prompts (PDD)

- **Núcleo**: El historial del chat es la verdad.
- **Estilo**: Chat interactivo de formato libre.
- **Riesgo**: Alta desviación de diseño y regresiones.
- **Velocidad**: Prototipado muy rápido.

Desarrollo por Espec. (SDD)

- **Núcleo**: Especs formales de API/esquemas.
- **Estilo**: Esquema formal + codegen.
- **Riesgo**: Documentación desactualizada.
- **Velocidad**: Media (sobrecarga inicial).

SPDD Estructurado

- **Núcleo**: Prompt/espec. de lienzo.
- **Estilo**: Especs estructuradas + agentes.
- **Riesgo**: Bajo (puertas de calidad automatizadas).
- **Velocidad**: Alta (con guardarraíles).

1. Filosofía: Matriz de Análisis Comparativo

Criterio	PDD (Basado en Prompts)	SDD (Basado en Espec.)	SPDD (Estructurado)
Concepto Central	Historial de chat interactivo	Espec. formal (API/MD)	Prompt/lienzo estructurado
Estilo de Trabajo	Chatbot de formato libre	Espec. inicial + codegen	Lienzo + agentes orquestados
Nivel de Control	Bajo (Depende del desarrollador)	Muy Alto (Validación formal)	Alto (Gobernanza clara)
Trazabilidad	Baja (Historial difícil de auditar)	Media/Alta (Vínculos Espec. $\leftrightarrow$ Código)	Alta (Espec. $\leftrightarrow$ Diseño $\leftrightarrow$ Código $\leftrightarrow$ Prueba)
Velocidad de Entrega	Muy Alta (Solo prototipado)	Media (Fuerte sobrecarga inicial)	Alta (Automatización protegida)
Riesgo Operativo	Alto (Regresión estructural)	Bajo/Medio (Riesgo de especs obsoletas)	Bajo (Puertas de calidad automáticas)
Gobernanza	Débil (Sin intención auditable)	Muy Buena (Esquemas explícitos)	Excelente (Pista de auditoría automática)
Casos de Uso Ideales	Tareas exploratorias, scripts	Misión crítica, APIs	Entrega adaptable a escala

1. Ejemplo de Estructura: Desarrollo por Especificaciones (SDD)

Estructura del Proyecto (SDD)

Estructura típica donde el diseño formal define el código. +-

```
specs/                                  # Especificaciones formales
| +- openapi.yaml                              # Contrato de la API
| +- schema.prisma                             # Modelo de Datos
+- src/
    +- generated/                  # Código autogenerado (No editar)
    | +- api_types.ts                          # Tipos de endpoints
    | +- db_client.ts                         # Cliente ORM generado
    +- handlers/                             # Implementación manual
        +- user.ts                  # Implementa tipos de api_types
```

Flujo de Trabajo SDD

1. El arquitecto escribe el contrato `openapi.yaml`.
2. El compilador/codegen genera el boilerplate de tipos.
3. El desarrollador escribe la lógica de negocio final.

Contrato de API Ejemplo (`openapi.yaml`)

```
paths:
  /users:
    post:
      summary:  Registrar nuevo usuario
      requestBody:
        required:  true
        content:
          application/json:
            schema:
              $ref:  '#/components/schemas/UserRequest'
```

Código Implementado (`handlers/user.ts`)

```
import { UserRequest, UserResponse } from '../generated/api_types';
import { db } from '../generated/db_client';

export async function createUser(req:  UserRequest):
Promise<UserResponse> {
  const user = await db.user.create({ data:  req });
  return { id:  user.id, status:  'success' };
}
```

1. Ejemplo de Estructura: SPDD Estructurado

Estructura del Proyecto (SPDD)

Estructura donde la IA lee especificaciones en lenguaje estructurado y ejecuta cambios transaccionales. `+- .specify/ #`

```
Contexto global para IA
| +- constitution.md                    # Reglas de estilo y calidad
+- specs/                               # Prompts y especs. dinámicas
| +- user_flow.md                       # Lienzo / Requisitos de feature
+- task.md                              # Plan de trabajo activo de la IA
+- src/                                 # Código modificado directamente
    +- handlers/
        +- user.ts                      # Editado/validado por agentes
```

Flujo de Trabajo SPDD

1. Se define la intención en `user_flow.md`.
2. La IA desglosa tareas en `task.md`.
3. La IA modifica `user.ts` y valida en sandbox.

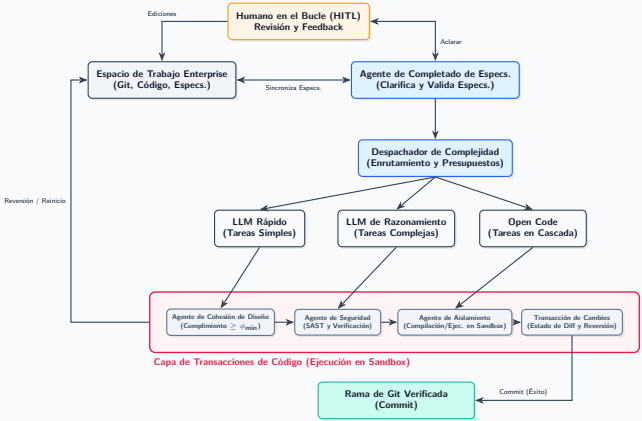
Lienzo de Especificación (`specs/user_flow.md`)

```
# Spec: Registro de Usuario
## Requisitos
El sistema debe validar el email del usuario y verificar que sea único
en la BD.
## Criterios de Aceptación
- [ ] Retornar error 400 si el email no es válido.
- [ ] Retornar 409 si el email ya existe.
```

Seguimiento de Tarea Dinámico (`task.md`)

```
# TODO
- [/] Implementar lógica de validación de email
  - [x] Agregar validación Regex en user.ts
  - [ ] Añadir consulta de unicidad en BD
- [ ] Diseñar y ejecutar suite de pruebas unitarias
```

2. Descripción General de la Arquitectura del Framework



3.1. Agente de Completado de Especs. (Integridad de Entrada)

Actúa como el guardián interactivo. Valida las especificaciones del usuario para verificar su integridad antes de iniciar la generación de código.
Modelo Matemático (Claridad de Spec.) [2, 7]:

$$S_{\text{clarity}} = \frac{T_{\text{resolved}}}{T_{\text{total}}} \times \left(1 - \frac{A_{\text{unresolved}}}{D_{\text{total}}}\right)$$

Donde:

- $T_{\text{resolved}} \in \mathbb{N}_0$, $T_{\text{total}} \in \mathbb{N}$ ($T_{\text{resolved}} \leq T_{\text{total}}$): Referencias de entidades resueltas.
- $A_{\text{unresolved}} \in \mathbb{N}_0$: Ambigüedades detectadas en la especificación.
- $D_{\text{total}} \in \mathbb{N}$: Reglas declarativas totales.

Si $S_{\text{clarity}} < \theta_{\text{threshold}}$ (donde $\theta_{\text{threshold}} \in [0, 1]$, por defecto: 0.85), el sistema solicita aclaración interactiva.

3.2. Despachador de Complejidad (El Enrutador)

Optimiza la latencia y los costos enrutando las tareas a los modelos adecuados.
Fórmula de Enrutamiento Heurístico [4]:

$$C_{\text{task}} = w_s \cdot C_s + w_i \cdot I_s + w_d \cdot D_a$$

Donde $C_s \in \mathbb{N}_0$ (contexto), $I_s \in \{0, 1, 2\}$ (impacto estructural), $D_a \in \{1, \ldots, 5\}$ (dificultad algorítmica) y los pesos $w_s, w_i, w_d \in \mathbb{R}_{\geq 0}$.
Rutas de Destino (con $\tau_1 < \tau_2 \in \mathbb{R}_{>0}$):

- $C_{\text{task}} < \tau_1$: **LLM Rápido / Local**
- $\tau_1 \leq C_{\text{task}} < \tau_2$: **LLM de Desarrollo Estándar**
- $C_{\text{task}} \geq \tau_2$: **LLM de Razonamiento / Open Code**

3.3. Agente de Síntesis de Código (El Implementador)

El motor principal que traduce especificaciones estructuradas en modificaciones exactas y limpias.

- **Responsabilidad**: Genera parches específicos en el espacio de trabajo en formato de diff unificado.
- **Entradas**: Esquemas validados, mapeos de contexto, reglas de sintaxis.
- **Salidas**: Diccionario que contiene las rutas de los parches y los cambios unificados.
- **Autocorrección**: Los errores de compilación sintáctica activan una verificación del analizador en bucle interno (máx. 3 reintentos antes de escalar).

3.4. Agente de Síntesis de Pruebas (El Validador)

Sintetiza suites de pruebas funcionales derivadas de los criterios de la especificación, evitando simulaciones (mocks) de bajo valor.

- **Responsabilidad**: Genera suites completas de pruebas unitarias, de integración y de regresión.
- **Soporte de Frameworks**: Frameworks de pruebas estándar (PyTest, Jest, Mocha).
- **Guardarraíl**: Si las pruebas sintetizadas están vacías o falla su compilación, la Capa de Transacción de Código bloquea el conjunto de cambios y activa la regeneración de la lógica de prueba.

3.5. Agente de Cohesión de Diseño (Guardián de la Arquitectura)

Garantiza que los cambios de código se adhieran a los Registros de Decisiones Arquitectónicas (ADRs) activos y a las pautas de estilo de la base de código.
Coeficiente de Cumplimiento de Diseño (C_{design}) [8]:

$$C_{\text{design}} = \frac{\sum_{i=1}^{n} P_i \cdot \text{Score}(P_i)}{\sum_{i=1}^{n} P_i} \geq \phi_{\text{min}}$$

Donde $n \in \mathbb{N}$ (reglas activas), $P_i \in \mathbb{R}_{>0}$ (peso de la regla i), $\text{Score}(P_i) \in [0, 1]$ (adherencia de la regla i) y $\phi_{\text{min}} \in [0, 1]$ (umbral mínimo, típico 0.90).

Si $C_{\text{design}} < \phi_{\text{min}}$, los cambios son rechazados con feedback estructural.

3.6. Agente de Seguridad (El Escudo de Políticas)

Escanea modificaciones en busca de vulnerabilidades de seguridad y problemas de cumplimiento.

- **Mecanismos**: Pruebas estáticas de seguridad de aplicaciones (SAST) automatizadas y auditorías de paquetes.
- **Áreas de Enfoque**: Fuga de credenciales, fallas de inyección, licencias de terceros no conformes.
- **Política**: Cualquier vulnerabilidad de riesgo **Alto** o **Crítico** aborta inmediatamente el pipeline, registra un incidente y revierte el código.

3.7. Agente de Aislamiento (Ejecución en Sandbox)

Protege la estación de trabajo del desarrollador ejecutando compilaciones y suites de pruebas en sandboxes aislados.

- Instancia entornos de ejecución en contenedores aislados (Docker/WASM) con límites de recursos.
- Genera registros de pruebas estructurales, perfiles de recursos y códigos de salida estrictos (0 para éxito).

3.8. Sincronización de Documentación y Artefactos

Evita la desviación de la documentación.

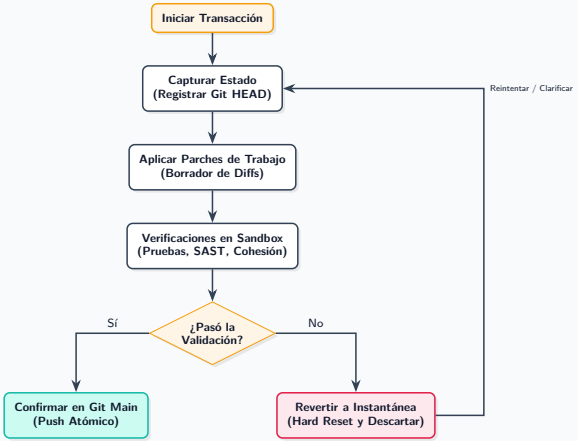
- Actualiza automáticamente especificaciones markdown, esquemas OpenAPI/Swagger, registros de cambios y ADRs correspondientes a los cambios fusionados.

3.9. Orquestador de Transacciones y Git

Garantiza operaciones ACID.

- Gestiona instantáneas de git, confirma conjuntos de cambios, actualiza el manifiesto de transacciones rastreando costos de API, sobrecarga de tokens y firmas del desarrollador.

4. Capa de Transacción de Código: Diagrama de Flujo del Ciclo de Vida



4. La Capa de Transacción de Código: Modificaciones Atómicas

4.1. Transacción de Cambios Consumidos

Cada modificación propuesta se registra en un manifiesto unificado:

- **Diffs de Estado**: Archivos de parches unificados que contienen adiciones y eliminaciones de líneas.
- **Seguimiento de Metadatos**: Tokens consumidos, costo computacional, tiempo de ejecución y métricas del sistema.
- **Auditabilidad**: Vínculos de alta fidelidad que rastrean el commit finalizado de vuelta al ID del problema de la especificación original.

4.2. Capacidad de Reversión

Si algún paso de validación (cohesión de diseño, políticas de seguridad o pruebas en sandbox) falla, la transacción es rechazada:

- **Contaminación de Estado Cero**: El espacio de trabajo se restablece inmediatamente a la instantánea previa a la transacción.
- **Registro de Transacción Fallida**: Se capturan el parche fallido, las salidas de compilación y los fallos de las pruebas en sandbox.
- **Bucle de Refinamiento**: Los registros se enrutan de vuelta al Agente de Completado de Especs. para autorefinar los prompts o consultar al desarrollador.

5. Integración en el SDLC: Fases 1 a 4

Etapa de SDLC	Agente IA / Actividad	Humano en el Bucle (HITL)	Automatización IA / Guardarraíles	Entregables Mínimos
1. Comprender Espec.	El Agente de Completado de Especs. extrae especs y objetivos.	Revisar	Análisis de brechas que compara requisitos frente a la base de código.	Resumen + Mapa de Procesos del Sistema
2. Historias y Criterios	Genera Historias de Usuario, Definición de Terminado y pruebas.	Ajustar alcance	Resolución automatizada de dependencias entre historias de usuario.	Backlog + Criterios de Validación
3. Estimación y Tamaño	El Despachador de Complejidad analiza riesgos y dimensiona tareas.	Aprobar rutas	Estimación de tamaño de tarea inicial sensible al contexto.	Matriz de Estimación + Registro de Riesgos
4. Diseño y Des.	Ejecución multiagente (GitHub Spec-Kit, Claude Code).	Aprobar código	Scaffolding de código en sandbox	Registros de Decisiones de Arquitectura + Código

5. Integración en el SDLC: Fases 5 a 7

Etapa de SDLC	Agente IA / Actividad	Humano en el Bucle (HITL)	Automatización IA / Guardarraíles	Entregables Mínimos
5. Calidad y Seguridad	El Agente de Seguridad escanea políticas. Cohesión de Diseño verifica estilo.	Revisar excepciones	Adherencia al estilo basada en AST y barreras de vulnerabilidades SAST.	Informe de Cumplimiento + Mapeo AST
6. Prueba y Lanzamiento	El Agente de Aislamiento ejecuta pruebas en sandbox. Genera datos sintéticos.	Verificar UAT	Ejecución de pruebas aislada y verificación de cobertura.	Registros Dinámicos de UAT + Informe de Cobertura
7. Despliegue y Ops	Listas de verificación de lanzamiento y runbooks de reversión.	Aprobar ventana	Compilación automatizada de notas de lanzamiento.	Changelogs + Runbooks Operativos

6. Hoja de Ruta (Cronograma 2026): Fases 1 y 2

Resumen de la Hoja de Ruta

- **Julio**: Descubrimiento y Estrategia
- **Agosto - Septiembre**: Construcción de Agentes
- **Octubre**: Integración de Herramientas
- **Noviembre - Diciembre**: Piloto Empresarial

Fase 1: Descubrimiento Estratégico y Línea Base (Julio 2026)

- **Metodología de Consultoría**: Mapear patrones actuales de desarrollo, identificando desperdicios en recopilación de requisitos y verificación manual.
- **Viabilidad**: Establecer límites para estaciones de trabajo locales frente a la orquestación en la nube.

Fase 2: Ingeniería de Agentes (Agosto - Sept 2026)

- **Agente de Completado de Espec.**: Implementar un analizador semántico para verificar requisitos contra gráficos de la base de código.
- **Cumplimiento de Políticas**: Programar reglas de estilo y restricciones de seguridad como aserciones a nivel AST.

6. Hoja de Ruta (Cronograma 2026): Fases 3 y 4

Fase 3: Sandbox e Integraciones (Octubre 2026)

- **Orquestación en Sandbox**: Desplegar perfiles de entorno de ejecución WASM/Docker en contenedores para el Agente de Aislamiento.
- **Motor de Transacciones**: Ensamblar máquinas de estado respaldadas por git-hooks que gestionen checkpoints y snapshots ACID.
- **Integraciones de Herramientas**: Construir adaptadores para GitHub Spec-Kit, Claude Code y el Método BMAD.

Fase 4: Escalamiento y Pilotos Enterprise (Nov - Dic 2026)

- **Programas de Habilitación**: Establecer módulos de capacitación para líderes de ingeniería para cambiar el enfoque de escribir código a escribir especs.
- **Pilotos de Producción**: Ejecutar proyectos piloto midiendo métricas clave: Tiempo de Ciclo, MTTR, Tasa de Escape de Defectos y Eficiencia de Costos de LLM.

7. Frameworks Externos: Integración con GitHub Spec-Kit

El Núcleo Estructural [6]

Integrado como los estándares del repositorio y el motor de prompts.
Plano de Instalación vía uv:

```
uv tool install specify-cli -from
git+https://github.com/github/spec-kit.git
```

Inicialización:

```
specify init -integration copilot
```

Mapeo de Comandos Core

- **constitution**: Pautas de límites.
- **specify / clarify**: Bucles de ambigüedad.
- **plan / tasks**: Estrategias técnicas.
- **implement**: Ejecución de cambios.

Estructura de Repositorio Aislado

```
+- .github/                                  # Prompts y Ajustes de IA
+- .specify/
    +- memory/
        +- constitution.md                          # Reglas básicas
    +- templates/                            # Plantillas de especs
    +- scripts/                             # Scripts de validación
```

7. Frameworks Externos: El Método BMad [3]

Planificación Adaptable a Escala

Usa enrutamiento dinámico para escalar la profundidad de la planificación según la complejidad de la tarea.

- Correcciones simples activan microciclos.
- Tareas complejas escalan a tableros multi-agente.

Colaboración en Modo Fiesta

Inicializa tableros de agentes cooperativos cuando $C_{\text{task}} \geq \tau_2$.

- Genera Analistas, Product Managers y Arquitectos.
- Debate y refina los cambios de código antes de las pruebas en sandbox.

Arquitecto de Pruebas (TEA)

Impulsa la verificación automatizada dentro de sandboxes aislados:

- Implementa estrategias de prueba basadas en riesgos.
- Produce datos de prueba sintéticos y simulaciones dinámicamente.

7. IDE de Desarrollo Nativo y Herramientas de Agentes

Integración de Claude Code [5, 1]

Agente principal para ejecución local de tareas por CLI.

- Opera localmente bajo las reglas definidas en `.specify/memory/constitution.md`.
- Ejecuta pruebas de forma segura, edita archivos y gestiona el flujo de trabajo de Git local.

Preajustes de OpenCode y Extensiones

Permite flexibilidad multiplataforma de IDE.

- Se integra con Cursor, Gemini CLI, Windsurf y JetBrains.
- Utiliza ajustes preestablecidos de configuración de IDE para sobrescribir los valores predeterminados de la plantilla, manteniendo la alineación del equipo.

SPDD une la intención estructurada, la validación en contenedores y la sinergia multi-agente para la escala del SDLC empresarial.

Referencias I

[1] Anthropic. Claude code: Terminal-first agentic developer tool, 2026. URL https://code.claude.com/docs/en/overview. Developer documentation and CLI reference.

[2] V. R. Basili, G. Caldiera, and H. D. Rombach. The goal question metric approach. 1994. URL https://api.semanticscholar.org/CorpusID:13884048.

[3] BMad Code Org. BMAD-METHOD: Breakthrough Method for Agile AI Driven Development. https://github.com/bmad-code-org/BMAD-METHOD, 2026. GitHub repository, accessed 2026-05-22.

[4] L. Chen, M. Zaharia, and J. Zou. Frugalgpt: How to use large language models while reducing cost and improving performance, 2023. URL https://arxiv.org/abs/2305.05176.

[5] M. S. Fayed and A. S. Fayed. Prompt-driven development with claude code: Developing a tui framework for the ring programming language. *Electronics*, 15(4), 2026. ISSN 2079-9292. doi: 10.3390/electronics15040903. URL https://www.mdpi.com/2079-9292/15/4/903.

Referencias II

[6] GitHub, Inc. Github spec-kit: Spec-driven development framework and tooling, 2026. URL https://github.com/github/spec-kit. Online repository and specification CLI.

[7] B. Meyer. Applying "design by contract". *Computer*, 25(10):40–51, Oct. 1992. ISSN 0018-9162. doi: 10.1109/2.161279. URL https://doi.org/10.1109/2.161279.

[8] L. Passos, R. Terra, M. T. Valente, R. Diniz, and N. Mendonça. Static architecture-conformance checking: An illustrative overview. *IEEE Software*, 27(5):82–89, 2010. doi: 10.1109/MS.2009.117.