El Libro de Lógica (Logic Book)

Gobernanza y Disciplina en la Era del Desarrollo Aumentado por IA

Prefacio al Marco de Trabajo DirGen y a la Metodología Driven Black Box Development

Prefacio: El Espectro de la Gobernanza en la Nueva Era del Software

La llegada de los Grandes Modelos de Lenguaje (LLMs) como agentes en el desarrollo de software no representa una simple mejora incremental, sino un cambio de paradigma.

Nos encontramos en una encrucijada histórica que exige repensar cómo concebimos, diseñamos y construimos sistemas.

El camino del 'Vibe Coding' —una adopción intuitiva y caótica de la IA— promete velocidad, pero suele desembocar en sistemas frágiles y deuda técnica exponencial.

El camino alternativo es la disciplina aumentada, donde la potencia de la IA se canaliza mediante principios de ingeniería rigurosos y explícitos.

1. El Dilema de la Caja Negra: Abstracción vs. Abdicación

La capacidad de la IA para generar implementaciones complejas como 'cajas negras' es, simultáneamente, su mayor fortaleza y su mayor riesgo.

Sin gobernanza, delegar la implementación equivale a abdicar de la responsabilidad de ingeniería, creando una Torre de Babel técnica.

La solución no es rechazar la caja negra, sino gobernarla mediante contratos externos explícitos e inmutables.

2. Driven Black Box Development (DBBD): La Metodología Fundamental

DBBD se basa en la separación radical entre la Lógica (el 'Qué') y la Implementación (el 'Cómo').

La Lógica de Negocio es la fuente de verdad versionada, mientras que la Implementación es una caja negra reemplazable evaluada por su cumplimiento del contrato.

Este enfoque evita la pérdida de lógica de negocio durante implementaciones y refactorizaciones autónomas.

Ideal para prototipado rápido, desarrollo de frontends y herramientas internas donde la funcionalidad inmediata prima sobre la robustez.

3. Directed Generative Software Engineering (DirGen): Implementación de Alto Rigor

DirGen amplía los principios de DBBD para entornos empresariales de misión crítica.

La caja negra no solo debe cumplir la lógica de negocio, sino también requisitos no funcionales convertidos en Quality Gates automatizados:

- Seguridad: cero vulnerabilidades críticas.

- Calidad de Código: cobertura superior al 80%.

- Rendimiento: latencia y throughput dentro de límites definidos.

- Cumplimiento Normativo: trazabilidad total de decisiones y cambios.

4. El Artefacto Central: El Libro de Lógica (Logic Book)

El Logic Book es la constitución del proyecto, vivo y adaptativo, pero regulador y vinculante.

Durante un ciclo, sus principios son inmutables y toda implementación debe obedecerlos.

Es la herramienta clave para asegurar que la visión estratégica se mantenga intacta en la generación automatizada.

5. Conclusión: Un Espectro de Gobernanza

No existe un modelo único para el desarrollo aumentado por IA. Proponemos un espectro adaptable:

- DBBD ofrece agilidad y protege la lógica de negocio.

- DirGen añade rigor, seguridad y calidad técnica.

Ambos comparten un principio común: La Lógica dirige y la Implementación obedece.

El Logic Book es el instrumento que hace posible esta separación y gobernanza.

DirGen - Logic Book

Sistema Operativo para Desarrollo de Software Aumentado por IA

Documento conceptual de la arquitectura y flujo de gobernanza de DirGen

PARTE I: El Orquestador - El Núcleo de la Gobernanza

Capítulo 1: Visión General del Sistema DirGen

0.1. Misión de la Plataforma

La plataforma DirGen es un Sistema Operativo para el Desarrollo de Software Aumentado por IA. Su misión no es reemplazar al desarrollador, sino elevar su rol de 'constructor' a 'director'. DirGen industrializa el proceso de desarrollo, transformando requerimientos de negocio en software funcional, validado y gobernable a través de una línea de ensamblaje operada por agentes de IA autónomos.

0.2. Arquitectura Conceptual: La Fábrica y sus Estaciones de Trabajo

El sistema se concibe como una fábrica digital:

1. Materia Prima (SVAD): El usuario introduce los requerimientos en un formato narrativo (documento SVAD).

2. Línea de Ensamblaje (Orquestador): Núcleo central que gestiona el flujo de trabajo de forma secuencial, asignando y supervisando tareas.

3. Estaciones de Trabajo (Fases): Cada etapa del SDLC (Análisis, Diseño, Código, etc.) es una fase distinta en la línea.

4. Trabajadores Expertos (Agentes): Cada fase tiene un agente de IA especializado (RequirementsAgent, Planner Agent, etc.) que recibe un contrato de tarea y la ejecuta autónomamente.

5. Control de Calidad (Quality Gates): Al final de cada fase, el trabajo es validado antes de pasar a la siguiente estación.

0.3. Filosofía de Interacción: El IDE Conversacional

El usuario interactúa con la plataforma mediante una interfaz que combina IDE y conversación. El Centro de Mando permite iniciar procesos, proporcionar contexto y aprobar explícitamente puntos de control.

Este Logic Book optimiza el diseño de DirGen, sirviendo como base para generar prompts que construyen la propia plataforma en un ciclo de auto-mejora continua.

Capítulo 2: La Transición de Requerimiento a Plan Aprobado

1.1. Inicio del Flujo: Recepción del SVAD

1.1.1. Disparador: El usuario sube un documento SVAD a través del Centro de Mando.

1.1.2. Acción del Orquestador: Crea un run_id, inicia la Fase 0 (Análisis de Requerimientos) e invoca al RequirementsAgent.

1.2. Lógica del RequirementsAgent (Validación del SVAD)

1.2.1. Contrato: Validar el documento SVAD contra una plantilla estándar de requerimientos.

1.2.2. Plantilla: Define secciones mínimas como 'Resumen Ejecutivo', 'Requerimientos Funcionales', 'NFRs' y 'Arquitectura Propuesta'.

1.2.3. Si falla la validación:

- Notificar task_complete con estado failed.

- Incluir explicación clara y recomendaciones en el payload.

- Orquestador establece estado FAILED y transmite la explicación a la UI.

1.2.4. Si la validación es exitosa:

- Notificar éxito y proceder a la siguiente tarea.

1.3. Segunda Tarea del RequirementsAgent: Generación del PCCE

1.3.1. Contrato: Transformar el SVAD validado en archivo pcce.yml.

1.3.2. Acción: Consulta al LLM para estructurar la información en YAML.

1.3.3. Finalización: Guardar archivo y notificar task_complete con ruta al PCCE.

1.4. Lógica del Orquestador (Transición al Quality Gate Humano)

1.4.1. Disparador: Recepción exitosa de task_complete.

1.4.2. Acciones:

- Notificar finalización de la fase.

- Invocar al Planner Agent con contrato limitado a planificación.

- Retransmitir plan a la UI y entrar en estado WAITING_FOR_DESIGN_APPROVAL.

1.5. Quality Gate Humano: Aprobación del Plan de Diseño

1.5.1. Usuario NO acepta: Orquestador transiciona a FAILED y limpia artefactos.

1.5.2. Usuario acepta: Orquestador establece estado DESIGNING e invoca al Planner Agent para ejecutar el plan.

1.6. Ejecución de la Fase de Diseño

1.6.1. Contrato: Generar artefactos listados en el plan.

1.6.2. Autocorrección: Planner Agent realiza verificación interna y corrige fallos antes de notificar.

1.6.3. Finalización exitosa: Notifica task_complete y Orquestador invoca al Validator Agent.

1.6.4. Validación: Validator Agent verifica existencia de todos los archivos.

1.6.5. Aprobación final:

- Validator reporta éxito.

- Orquestador notifica aprobación y espera confirmación humana para continuar a la fase de Código.

Capítulo 3: El Contrato de la Experiencia del Usuario (UX Logic Contract)

Este capítulo es el **"puente lógico"** entre el backend (Orquestador y Agentes) y el frontend (la UI/Desktop App). Define el comportamiento esperado de la interacción humano-máquina y elimina la ambigüedad en la comunicación entre componentes. Es la fuente de verdad para la integración.

3.1. Principio Fundamental: La Lógica de la UI es Lógica de Negocio

La forma en que un usuario interactúa con la plataforma, los estados visuales que experimenta, los mensajes de error que recibe y los flujos que sigue **no son consideraciones estéticas secundarias, sino una parte integral y crítica de la lógica de negocio**. Como tal, esta lógica debe ser definida, versionada y auditada con el mismo rigor que las reglas del backend.

3.2. El Contrato de la API como Contrato de Interacción (API Interaction Contract)

En DirGen, la API no solo define schemas de datos, define un **protocolo de comportamiento conversacional**. Por cada endpoint HTTP y cada type de mensaje WebSocket, se establece el siguiente contrato.

3.2.1. POST /v1/initiate_from_svad

- **Pre-condición de la UI:** La aplicación debe estar en un estado IDLE. El usuario debe haber adjuntado un archivo SVAD.
- **Post-condición (Éxito):**
    - **Estado del Run:** El Orquestador debe transicionar el Run al estado ANALYZING_REQUIREMENTS.
    - **Respuesta UI:** La UI debe recibir un run_id y transicionar su estado a EXECUTING.
- **Post-condición (Fallo - Archivo inválido):**
    - **Respuesta del Backend:** HTTP 400 (Bad Request).
    - **Respuesta UI:** Mostrar un mensaje de error claro al usuario.

3.2.2. Mensaje WebSocket plan_generated

- **Pre-condición:** Un agente ha completado su fase de planificación.
- **Payload (Data):** { "plan": ["Tarea 1", ...], "total_tasks": X }.
- **Post-condición (UI):**
    - El componente PlanWidget **debe** renderizar la lista de tareas recibida.
    - La UI debe mantener su estado de EXECUTING.

3.2.3. Mensaje WebSocket approval_request (Nombre hipotético para la pregunta)

- **Pre-condición:** La fase que requiere aprobación humana (ej. Análisis de Req.) ha finalizado con éxito.
- **Payload (Data):** { "message": "¿Deseas iniciar la Fase de Diseño...?" }.

- **Post-condición (UI):**
    - El Log de Eventos **debe** mostrar el mensaje de la pregunta.
    - El Centro de Mando **debe** activarse, indicando que espera una respuesta.
    - El estado general de la UI debe cambiar a WAITING_FOR_APPROVAL.

3.2.4. POST /v1/run/{run_id}/approve

- **Pre-condición del Sistema:** El Run debe estar en el estado WAITING_FOR_APPROVAL.
- **Post-condición (Éxito - Aprobado):**
    - **Estado del Run:** Transiciona a la siguiente fase (ej. DESIGNING).
    - **Respuesta UI:** La UI vuelve al estado EXECUTING y la siguiente fase se resalta en la barra de progreso.
- **Post-condición (Fallo - Estado Incorrecto):**
    - **Respuesta del Backend:** El Orquestador **debe** devolver un error HTTP 409 Conflict con un cuerpo JSON { "error": "Invalid state for this operation" }.
    - **Respuesta UI:** El frontend **debe** manejar este error y mostrar una notificación informativa al usuario (ej. "La acción ya no es válida").

3.3. Mapa de Estados de la Interfaz (UI State Machine)

El frontend **debe** ser capaz de renderizar y gestionar los siguientes estados, que se derivan del estado del Run en el backend.

- IDLE: Pantalla de bienvenida, Centro de Mando esperando archivo.
- EXECUTING: La UI muestra el progreso de un agente. Spinners y animaciones de "procesando" están activos. El Centro de Mando está deshabilitado.
- WAITING_FOR_USER_INPUT: Una fase ha terminado y se requiere aprobación. El Log de Eventos muestra una pregunta, y el Centro de Mando está habilitado.
- SUCCESS: El Run ha finalizado con éxito. Se muestra el resumen final y opciones para continuar.
- FAILED: El Run ha fallado. Se muestra un mensaje de error claro con la razón del fallo.

3.4. Flujos de Usuario Críticos (User Journey Contracts)

Esta sección define las secuencias lógicas esperadas, sirviendo como base para las pruebas E2E.

Flujo: De Inicio a Aprobación de Plan de Diseño

1. **Estado Inicial:** IDLE.
2. **Usuario:** Arrastra SVAD.md.
3. **Sistema (UI/Backend):** Ejecuta el contrato POST /v1/initiate_from_svad. El estado cambia a EXECUTING.
4. **Sistema (Backend/UI):** El RequirementsAgent trabaja y el Log de Eventos muestra su progreso.

5. **Sistema (Backend/UI):** El Orquestador envía los mensajes task_complete (del agente) y luego approval_request. El estado de la UI cambia a WAITING_FOR_USER_INPUT.
6. **Usuario:** Escribe "procede" y envía.
7. **Sistema (UI/Backend):** Ejecuta el contrato POST /v1/run/{run_id}/approve. El estado de la UI vuelve a EXECUTING.
8. **Estado Final Esperado:** La fase de "Diseño" está activa en la barra de progreso. El Planner Agent ha sido invocado.

Capítulo 4: El Toolbelt - Las Herramientas del Orquestador

2.1. Principio de Seguridad: Sandboxing y Abstracción

La filosofía fundamental del Toolbelt es la seguridad a través de la abstracción. Los agentes nunca tienen acceso directo al sistema de archivos, a la terminal del sistema anfitrión, o a la red. Operan en un sandbox lógico. Toda interacción con el entorno exterior se realiza a través de una petición explícita a una herramienta expuesta por el Orquestador como un endpoint de API.

Lógica del Orquestador:

- Debe exponer un conjunto de endpoints bajo el prefijo /v1/tools/.
- Cada endpoint debe validar los argumentos recibidos para prevenir ataques (ej. path traversal en rutas de archivos).
- Todas las operaciones de archivo deben estar restringidas a un directorio de trabajo específico del run_id actual, para aislar las ejecuciones entre sí.

2.2. Herramienta filesystem

2.2.1. POST /v1/tools/filesystem/writeFile

Contrato: Escribe o sobreescribe contenido en un archivo dentro del directorio de trabajo del Run.

Input (JSON Body):

{ "path": "design/architecture.puml", // Ruta relativa al directorio del proyecto.

"content": "@startuml\n..." // Contenido completo del archivo. }

- **Lógica del Orquestador:**
    1. Verifica que la ruta path sea segura (sin ../ o rutas absolutas).
    2. Construye la ruta completa dentro del sandbox del run_id.
    3. Crea los directorios padres si no existen.
    4. Escribe el archivo.
- **Output (JSON Body):**

{ "success": true }

// o {"success": false, "error": "Mensaje de error."}

- **2.2.2. POST /v1/tools/filesystem/readFile**
  - **Contrato:** Lee el contenido de un archivo dentro del directorio de trabajo.
  - **Input (JSON Body):** { "path": "design/api/orders.yml" }
  - **Lógica del Orquestador:** Validar la ruta y leer el archivo.
  - **Output (JSON Body):** { "success": true, "content": "openapi: 3.0..." }
- **2.2.3. POST /v1/tools/filesystem/listFiles**
  - **Contrato:** Lista el contenido (archivos y directorios) de una ruta.
  - **Input (JSON Body):** { "path": "services/orders/" }
  - **Lógica del Orquestador:** Validar la ruta y listar su contenido.
  - **Output (JSON Body):** { "success": true, "files": ["main.py", "api/"], "directories": ["api/"] }

2.3. Herramienta terminal (Funcionalidad Futura - Fase 2 en adelante)

- **Contrato:** Ejecuta un comando en un shell dentro de un entorno controlado y aislado (idealmente, un contenedor Docker temporal).
- **Input (JSON Body):** { "command": "pytest --cov", "cwd": "services/orders/" }
- **Lógica del Orquestador:**
  1. Validar y sanear el comando para prevenir inyecciones.
  2. Ejecutar el comando dentro de un subproceso o contenedor con timeouts estrictos.
- **Output (JSON Body):** { "stdout": "...", "stderr": "...", "exit_code": 0 }

2.4. Mecanismo de Extensibilidad

La arquitectura del Orquestador debe permitir añadir nuevas herramientas de forma modular sin alterar el núcleo. Cada nueva herramienta debe ser:

1. Implementada como una nueva ruta de API bajo /v1/tools/.
2. Documentada en este Libro de Lógica para que los agentes "sepan" que existe y cómo usarla.

Capítulo 5: El Servidor de Comunicaciones

3.1. API REST para Comandos e Informes de Agentes

La comunicación iniciada por el cliente o los agentes hacia el Orquestador se realiza a través de una API REST.

- POST /v1/initiate_from_svad: Inicia una nueva ejecución.

- POST /v1/run/{run_id}/approve: Permite al usuario aprobar una fase.
- POST /v1/agent/{run_id}/report: Endpoint genérico para que un agente reporte su progreso (thought, action, etc.).
- POST /v1/agent/{run_id}/task_complete: Endpoint para que un agente notifique la finalización de su contrato de tarea.

3.2. Servidor WebSocket para el Flujo de Eventos en Tiempo Real

La comunicación iniciada por el Orquestador hacia la UI del cliente se realiza a través de un canal WebSocket para proveer una experiencia en tiempo real.

- **Endpoint de Conexión:** ws://host/ws/{run_id}
- **Protocolo:** Comunicación unidireccional (Servidor -> Cliente).
- **Esquema de Mensajes (Contrato de la UI):** Todos los mensajes enviados por el WebSocket deben adherirse a la siguiente estructura JSON:

```
{
  "source": "Orchestrator" | "Requirements Agent" | "Planner Agent" | ...,
  "type": "phase_start" | "plan_generated" | "thought" | "action" | ...,
  "data": { ... } // Payload específico del tipo de mensaje.
}
```

- **Lógica de Conexión:**
    1. El cliente establece la conexión después de recibir el run_id.
    2. El Orquestador mantiene un ConnectionManager que mapea cada run_id a su conexión WebSocket activa.
    3. Toda la lógica de manager.broadcast() envía mensajes al cliente correcto basándose en el run_id del evento.

PARTE II: Los Agentes - La Fuerza de Trabajo Autónoma

Capítulo 4: El Contrato y Ciclo de Vida Universal de un Agente

Todo agente que opera dentro de la plataforma DirGen, sin importar su especialización, **debe** adherirse a un ciclo de vida y a un conjunto de protocolos de comunicación estandarizados. Esto asegura que sean intercambiables, predecibles y gobernables por el Orquestador.

4.1. El Contrato de Tarea: Inputs Obligatorios

Al ser invocado por el Orquestador, cada agente debe ser capaz de recibir y procesar los siguientes argumentos de línea de comandos:

- --run-id: El identificador único universal de la ejecución. Es obligatorio.
- --pcce-path: La ruta al archivo pcce.yml que contiene el contexto técnico del proyecto. Es obligatorio.

- --feedback: Un argumento opcional que contiene un mensaje de texto con el motivo de un fallo anterior, utilizado para los bucles de corrección.

4.2. Fase Obligatoria 1: Planificación Explícita

- **Lógica:** La **primera acción** de cualquier agente después de iniciarse y leer su contexto debe ser generar un plan de alto nivel para su propia tarea. No puede ejecutar ninguna herramienta de Toolbelt antes de haber formulado y comunicado su plan.
- **Implementación:**
    1. El agente realiza una consulta a su LLM asignado con un "prompt de planificación".
    2. El objetivo de este prompt es descomponer su objetivo de alto nivel (ej. "generar artefactos de diseño") en un checklist de tareas secuenciales y lógicas (ej. ["1. Crear diagrama C4", "2. Crear API para servicio X", ...]).
- **Reporte:** Una vez generado, el agente **debe** enviar inmediatamente el plan al Orquestador a través del endpoint de reporte, usando la estructura:

{ "source": "<nombre del agente>",

"type": "plan_generated",

"data": { "plan": ["Tarea 1", "Tarea 2", ...] }}

4.3. Fase Obligatoria 2: Ejecución Cíclica (ReAct)

- **Lógica del ciclo Pensamiento-Acción-Observación:** La ejecución del plan se realiza a través de un bucle iterativo. En cada iteración:
    1. **Pensamiento:** El agente reflexiona sobre el plan, el historial y la última observación para decidir el siguiente paso. Este pensamiento se reporta al Orquestador ("type": "thought").
    2. **Acción:** El agente elige **una única herramienta** del Toolbelt y la invoca a través de una petición HTTP al Orquestador. Esta acción se reporta ("type": "action").
    3. **Observación:** La respuesta del Orquestador (el resultado de la herramienta) se convierte en la observación, se añade al historial y se utiliza para informar el siguiente "Pensamiento".
- **Lógica de Actualización de Plan (Re-planificación):**
    - Si una observación revela un obstáculo fundamental que invalida el plan actual (ej. un requisito imposible, una herramienta faltante), el agente debe ser capaz de entrar en un estado de re-planificación.
    - Genera un nuevo plan y lo reporta al Orquestador usando el type: "plan_updated". La UI debe reflejar este cambio en la estrategia.

4.4. Fase Obligatoria 3: Verificación y Finalización Profesional

- **Lógica de Auto-verificación:** Antes de declarar su trabajo como finalizado, el agente debe realizar un ciclo final de Pensamiento enfocado en la auto-auditoría. Le preguntará a su LLM: "Basado en mi objetivo original y el historial de mis acciones, ¿he cumplido con todos los requisitos?".

- **Lógica de Generación de Resumen Ejecutivo:** Si la auto-verificación es exitosa, el agente realiza una última consulta al LLM para generar un resumen profesional de su trabajo en formato Markdown.
- **Reporte Final (task_complete):** La notificación final al Orquestador es un mensaje POST al endpoint /v1/agent/{run_id}/task_complete que debe incluir:
    - role: El rol del agente (ej. "planner").
    - status: success, failed, o impossible.
    - summary (en caso de success): El resumen ejecutivo generado.
    - reason (en caso de failed o impossible): Una explicación detallada.

Capítulo 5: Lógica de Resiliencia y Escalada de los Agentes

5.1. El Bucle de Corrección (impulsado por feedback)

- **Lógica:** Cuando un agente es invocado con el argumento --feedback, indica que es un reintento.
- **Implementación:** El contenido del --feedback **debe ser inyectado al inicio del historial de trabajo** del agente. Esto fuerza al LLM a considerar el fallo anterior como la primera pieza de contexto, guiándolo a no repetir el mismo error.

5.2. La "Memoria de Fallos" y el Criterio de Escalada

- **Lógica:** Para evitar bucles de corrección infinitos en los que el agente intenta la misma solución fallida repetidamente, el agente debe tener una "memoria" interna de sus intentos fallidos dentro de un mismo Run.
- **Implementación:**
    1. El agente debe mantener una estructura de datos simple (ej. un diccionario) que registre el error_message y la estrategia_intentada (la "Acción" que lo causó).
    2. **El Criterio de Escalada:** El agente debe determinar que una tarea es "IMPOSSIBLE" y escalar si se cumple una política de resiliencia. El criterio por defecto es:

Después de N (por defecto: 5) intentos de aplicar estrategias *diferentes* para resolver el *mismo* tipo de error sin éxito.

- **Protocolo de Escalada:**
    - El agente genera un "Pensamiento" final explicando por qué la tarea es imposible, citando las estrategias fallidas.
    - Envía la notificación task_complete con el estado impossible y la justificación.
    - **Acción del Orquestador:** Al recibir un estado impossible, el Run se detiene y transiciona a FAILED, mostrando la justificación del agente al usuario.

Capítulo 6: Perfiles y Responsabilidades de los Agentes

Cada agente en la plataforma DirGen es un experto en un dominio específico del Ciclo de Vida de Desarrollo de Software (SDLC). Aunque todos operan bajo el **Ciclo de Vida Universal de un Agente (Capítulo 4)**, su "conocimiento", sus objetivos y las herramientas que priorizan son únicos.

6.1. RequirementsAgent (El Analista de Negocio)

- **Invocado en:** Fase 0: Análisis de Requerimientos.
- **Contrato de Tarea Primario:** Transformar la intención de negocio (un SVAD.md) en una especificación técnica accionable (pcce.yml).
- **Responsabilidades Clave:**
    1. **Validar:** Leer el SVAD y compararlo con una plantilla de contrato de requerimientos. Debe ser capaz de identificar secciones faltantes o ambiguas y reportarlas como un error bloqueante.
    2. **Extraer:** Realizar una consulta a su LLM para parsear la narrativa del SVAD y extraer la información estructurada clave: requerimientos funcionales, NFRs, componentes propuestos, stack, etc.
    3. **Generar:** Construir y escribir el archivo pcce.yml, asegurándose de que su formato sea sintácticamente correcto y se adhiera a la estructura esperada por el Orquestador.
- **Resultado Final Exitoso:** Un único archivo pcce.yml.
- **Restricción Importante:** Este agente **NO** se ocupa de la planificación de la implementación. Su rol termina con la traducción exitosa del SVAD.

6.2. Planner Agent (El Arquitecto de Soluciones)

- **Invocado en:** Fase 1: Diseño.
- **Contrato de Tarea Primario:** A partir de un pcce.yml validado, generar todos los artefactos de diseño de alto nivel que servirán como planos para los agentes de código.
- **Responsabilidades Clave:**
    1. **Planificar:** Como su primera acción, debe leer la sección fases.diseno.salidas_esperadas del PCCE y generar un plan detallado (checklist) de los archivos que va a crear.
    2. **Diseñar Arquitectura:** Generar diagramas visuales (como C4 en formato PlantUML) que representen la estructura de los componentes y sus interacciones.
    3. **Definir Contratos de API:** Para cada servicio que exponga una API, debe generar una especificación OpenAPI 3.0 completa y válida.
    4. **Generar otros Documentos de Diseño:** Crear cualquier otro artefacto definido en el PCCE (ej. diagramas de secuencia, modelo de datos, etc.).
- **Resultado Final Exitoso:** Un conjunto de archivos (.puml, .yml) en el directorio design/ del espacio de trabajo.

6.3. Validator Agent (El Inspector de Calidad)

- **Invocado en:** Al final de una fase, como parte de un Quality Gate.

- **Contrato de Tarea Primario:** Auditar un conjunto de artefactos generados contra las reglas definidas en el PCCE.
- **Características Clave:**
    - **No usa IA (generalmente):** A diferencia de otros agentes, sus tareas son deterministas. Ejecuta una serie de comprobaciones lógicas.
    - **Configurable por Fase:** La lógica de validación que ejecuta depende de la fase en la que es invocado.
- **Responsabilidades Específicas:**
    - **En el Quality Gate de Diseño:**
        1. Verificar la existencia de todos los archivos listados en fases.diseno.salidas_esperadas.
        2. (Futuro) Ejecutar un linter de OpenAPI contra los archivos .yml.
        3. (Futuro) Validar la sintaxis de los archivos .puml.
    - **En el Quality Gate de Código (Futuro):**
        1. Verificar la existencia de la estructura de archivos del servicio.
        2. Ejecutar pytest y parsear el resultado para verificar la cobertura de pruebas contra la política qual-test-coverage-80.
- **Resultado Final Exitoso:** Un reporte de validación con {"success": true}. Si falla, el campo message debe contener una lista clara de las desviaciones.

6.4. CodeGen Agent (El Ingeniero de Software) *(Diseño Futuro)*

- **Invocado en:** Fase 2: Generación de Código.
- **Contrato de Tarea Primario:** Recibir un único contrato de API (un archivo OpenAPI) y generar la implementación completa y funcional de ese microservicio.
- **Responsabilidades Clave:**
    1. **Planificar:** Generar un plan para la estructura del código (crear carpetas, main.py, Dockerfile, etc.).
    2. **Generar Código:** Escribir el código fuente (Python/FastAPI) que implemente cada endpoint.
    3. **Generar Pruebas:** Escribir pruebas unitarias (pytest) que validen la lógica de negocio y los endpoints.
    4. **Generar Configuración:** Crear Dockerfile, requirements.txt, y otros archivos de configuración necesarios.
- **Auto-Validación:** El agente debe ser capaz de usar la herramienta terminal para ejecutar sus propias pruebas (pytest) dentro de su sandbox y no finalizar su tarea hasta que todas pasen.

6.5. SecAgent y QAAgent (Los Auditores Especializados) *(Diseño Futuro)*

- **Invocado en:** Quality Gate de la Fase 2 (Código).

- **SecAgent:** Su única responsabilidad es ejecutar herramientas de análisis de seguridad estático (SAST) y análisis de composición de software (SCA) contra el código generado, validando políticas como sec-no-hardcoded-secrets y sec-no-known-critical-cves.
- **QAAgent:** Su responsabilidad es ejecutar un conjunto más amplio de pruebas de calidad, como pruebas de integración entre servicios (si aplica), linters de estilo, y verificar la complejidad ciclomática.

PARTE III: La Interfaz - La Ventana del Director

Capítulo 7: Filosofía y Componentes Principales

La interfaz de usuario de DirGen no es un simple dashboard pasivo. Está diseñada bajo el paradigma del **"IDE Conversacional"**, un entorno de trabajo donde el usuario dirige, supervisa y aprueba el trabajo de un equipo de agentes de IA a través de un diálogo estructurado.

7.1. Componentes Visuales Fundamentales

La interfaz se estructura en paneles lógicos, cada uno con una responsabilidad clara:

- **7.1.1. La Barra de Progreso del SDLC:**
    - **Función:** Proporcionar al usuario un mapa visual constante de en qué etapa de la línea de ensamblaje (Análisis de Req. -> Diseño -> Código...) se encuentra la ejecución actual.
    - **Lógica:** Debe actualizarse dinámicamente para resaltar la fase activa del Run.
- **7.1.2. El Centro de Mando:**
    - **Función:** Es la **única fuente de entrada y comando del usuario**. Centraliza la interacción, desde iniciar una ejecución subiendo un archivo hasta dar respuestas conversacionales y aprobaciones.
    - **Lógica:** Debe ser un componente multimodal que acepte tanto texto como archivos adjuntos. Su estado (habilitado/deshabilitado) debe ser controlado por el Orquestador (ej. se habilita cuando se espera una respuesta del usuario).
- **7.1.3. El Plan de Ejecución:**
    - **Función:** Mostrar la estrategia de alto nivel del agente activo. Proporciona transparencia sobre la intención y los pasos que el agente planea seguir.
    - **Lógica:** Debe renderizar el checklist recibido del evento plan_generated. Debe ser interactivo, permitiendo expandir tareas para ver detalles (justificación, artefactos asociados). Se actualiza si el agente emite un plan_updated.
- **7.1.4. El Espacio de Trabajo:**
    - **Función:** Servir como el inventario de la ejecución.
    - **Lógica:** Se divide en dos pestañas:
        1. **Contexto:** Lista los archivos de entrada proporcionados por el usuario (ej. el SVAD.md).
        2. **Entregables:** Se puebla en tiempo real con una lista de los artefactos generados por los agentes (.puml, .yml, código fuente, etc.). Actúa como la "bandeja de salida" de la fábrica.

- **7.1.5. El Log de Eventos:**
    - **Función:** Es la transcripción en vivo del "piso de la fábrica". Muestra el diálogo entre el Orquestador y los Agentes, así como el razonamiento interno de los propios agentes (Pensamientos).
    - **Lógica:** Debe procesar el stream de eventos del WebSocket y renderizar cada mensaje con un formato y estilo distintivo según su source y type.

Capítulo 8: El Flujo de Interacción Lógico del Usuario

Este capítulo define el contrato de comportamiento entre el usuario y la plataforma.

8.1. Lógica de Inicio de Ejecución

1. **Acción del Usuario:** El usuario arrastra un archivo SVAD.md al Centro de Mando.
2. **Respuesta de la UI:** El archivo aparece como un "chip" adjunto.
3. **Acción del Usuario:** El usuario envía el prompt (puede ser con texto adicional o simplemente el archivo).
4. **Lógica del Frontend:** Se realiza el POST /v1/initiate_from_svad.
5. **Respuesta de la UI:** La interfaz entra en modo "Ejecución". Se recibe el run_id y se establece la conexión WebSocket. La barra de etapas resalta "Análisis de Req.".

8.2. Lógica de Aprobación de Plan

1. **Evento del Backend:** El Orquestador envía un mensaje {"type": "approval_request", "data": {...}}.
2. **Respuesta de la UI:**
    - El Log de Eventos muestra el mensaje conversacional del Orquestador (ej. "¿Procedemos con la ejecución?").
    - El Plan de Ejecución muestra el checklist de tareas generado por el agente.
    - El Centro de Mando se activa, indicando que espera una respuesta del usuario.
3. **Acción del Usuario:** Escribe una respuesta afirmativa (ej. "sí", "adelante") y la envía.
4. **Lógica del Frontend:** Realiza el POST /v1/run/{run_id}/approve.
5. **Respuesta de la UI:** El Centro de Mando se deshabilita, y el flujo de eventos de la siguiente fase (Diseño) comienza a aparecer en el Log. La barra de etapas se actualiza para resaltar "Diseño".

8.3. Lógica de Monitoreo

- **Acción del Usuario:** Es principalmente pasiva. El usuario observa el flujo de eventos en el Log, el progreso en el Plan y la aparición de archivos en el panel de Entregables.
- **Interacción (Opcional):** El usuario puede expandir tareas en el Plan o previsualizar los Entregables generados.

8.4. Lógica de Finalización de Fase/Run

1. **Evento del Backend:** El Orquestador envía un evento {"type": "phase_end", "data": {"status": "APROBADO" | "RECHAZADO"}} o {"type": "executive_summary", "data": {...}}.
2. **Respuesta de la UI:**
    - Se muestra de forma destacada el **Resumen Ejecutivo** generado por el agente.
    - La barra de etapas se actualiza para marcar la fase como completada (✅) o fallida (❌).
    - Si todo el Run ha finalizado, se muestra un estado final claro, y se habilita una opción para "Abrir Carpeta del Proyecto" o "Iniciar Nuevo Run".