Iron Man Evolution: De la Cueva al Nanotech

Diego Saavedra García

Mar 23, 2026

Table of contents

1 Iron Man Evolution: De la Cueva al Nanotech

2 Iron Man Evolution

2.1 Desarrollo con IA: De la Cueva al Nanotech

Duración: 20 horas | **Modalidad**: Autoestudio | **Nivel**: Principiante a Experto
Metodología: Aprendizaje por niveles como un videojuego
Inspiración: La saga completa de Iron Man (2008-2019)

2.2 La Trama Completa

> *"Siempre supe que moriría en paz, sabiendo que mis hijos estarían bien. Pero sabes… ahora, quiero hacer lo que tengo que hacer, sin que ellos sepan que soy el único capaz de hacerlo."* — Tony Stark, Avengers: Endgame

Iron Man Evolution no es solo un curso. Es una **experiencia narrativa** donde cada nivel corresponde a un momento crucial en la evolución de Tony Stark desde un genio millonario atrapado en una cueva hasta el hombre que salvó el universo con tecnología nanotech.

2.3 Sistema de Progresión (Gamificación)

2.3.1 Niveles del Juego

Cada nivel tiene: - **Escena de Iron Man** — La película que inspires - **Objetivos** — Qué aprenderás - **Laboratorio** — Ejercicios prácticos - **Logro** — Algo que desbloquear - **Boss Fight** — Reto final del nivel

Nivel	Iron Man	Habilidad Técnica	Tecnologías
1	**El Demo en la Cueva**	Fundamentos de IA	Prompts, tokens, context window
2	**El Mark I**	Context Engineering	AGENTS.md, convenciones, reglas
3	**El Mark III**	Agentes Básicos	Claude Code, loops, subagentes

Nivel	Iron Man	Habilidad Técnica	Tecnologías
4	**JARVIS Avanzado**	Memoria & Herramientas	Engram, MCP, Skills
5	**Ultron**	Orquestación Multi-Agente	SDD, ética, control humano
6	**El Nanotech**	Producción Enterprise	Gentle AI Stack, CI/CD, seguridad

---

2.4 Metodología: Aprender como Tony Stark

2.4.1 Principio Fundamental

> *"La armadura me hace más fuerte. J.A.R.V.I.S. me hace más inteligente. Juntos, somos imparables."*

Tu eres Tony Stark. La IA es tu J.A.R.V.I.S.
Tú construyes. La IA amplifica.
Tú decides. La IA ejecuta.

2.4.2 Filosofía de Enseñanza

1. **Aprender Haciendo** — 70% práctica, 30% teoría
2. **Progresión Natural** — De lo simple a lo complejo
3. **Contexto Real** — Proyectos que importan
4. **Reflexión Constante** — ¿Por qué funciona esto?

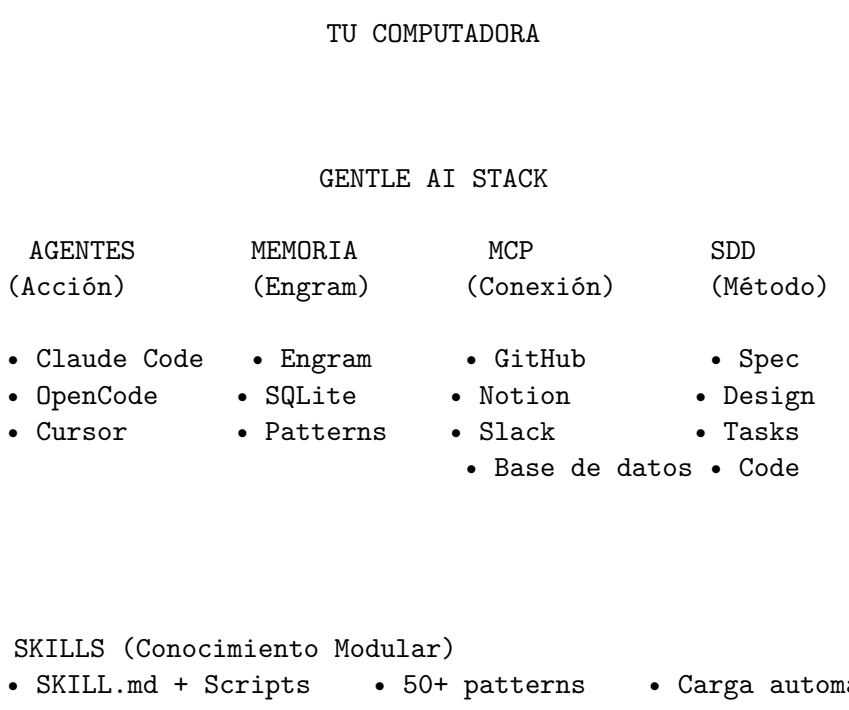
2.5 Stack Tecnológico (Gentle AI Stack)

2.5.1 Distinción Importante: Asistentes vs Agentes

Tipo	Ejemplos	Capacidad
Asistentes	Claude Chat, ChatGPT, Gemini	Conversación, sugerencias, análisis
Agentes	Claude Code, OpenCode, Cursor	Acción directa: escribir código, crear archivos, ejecutar comandos

Un asistente te dice QUÉ hacer. Un agente LO HACE por ti.

2.5.2 Arquitectura del Gentle AI Stack



2.5.3 Componentes del Stack

Componente	Tecnología	Función
Agentes	Claude Code, OpenCode, Cursor	Ejecutan acciones en tu código
Memoria	Engram (SQLite + FTS5)	Recuerdan decisiones y patrones
Conexión	MCP Protocol	Conectan con GitHub, Notion, Slack
Método	SDD (Spec-Driven Dev)	Estructuran el desarrollo
Skills	SKILL.md + Scripts	Modularizan conocimiento

- **Calidad**: TDD, Clean Code, Clean Architecture

2.6 Sistema de XP y Logros

2.6.1 Experiencia (XP)

Actividad	XP
Completar un lab	100 XP
Boss Fight superado	250 XP
Proyecto final	1000 XP
Comunidad (ayudar a otros)	50 XP

2.6.2 Logros Desbloqueables

Logro	Nombre	Requisito
	Survivor	Completar Nivel 1
	Architect	Crear tu primer AGENTS.md
	Pilot	Ejecutar tu primer agente
	Genius	Implementar Engram
	Director	Orquestar 3+ agentes
	Innovator	Completar todos los niveles

2.7 Tu Arsenal de Herramientas (NUEVO)

2.7.1 Antes de empezar: Instala tus herramientas

Esta sección es **OBLIGATORIA** antes de continuar con los niveles. Aquí aprenderás a instalar y configurar cada herramienta de IA:

Unidad	Herramienta	Qué aprenderás	Tiempo
Resumen	Todas	Comparativa y cuándo usar cada una	15 min
Unidad 2	OpenCode	Instalación, configuración, AGENTS.md	45 min
Unidad 3	Claude Code	Subagentes, debugging profundo	45 min
Unidad 4	Gemini CLI	Uso gratis, configuración	30 min
Unidad 5	KiloCode	Orchestrator Mode, multi-agente	45 min
Unidad 6	Kiro	Spec-Driven Development, Hooks	45 min

Unidad	Herramienta	Qué aprenderás	Tiempo
Unidad 7	Amp	Tareas rápidas, asistente ligero	20 min

Si ya sabés qué herramienta usar, saltá directo a la unidad correspondiente.
Si no sabés por dónde empezar, leé el Resumen de Herramientas primero.

2.7.2 Tu Primer Paso en Herramientas

→ **Resumen: Guía Comparativa de Herramientas**

2.8 Laboratorios Prácticos

2.8.1 Cada Nivel Incluye

1. **Training Lab** — Ejercicios guiados paso a paso
2. **Challenge Lab** — Reto práctico con solución disponible
3. **Boss Fight** — Proyecto integrador del nivel
4. **Side Quests** — Ejercicios opcionales para avanzados

2.8.2 Proyecto Final

Crearás **"The Iron Legion"** — un sistema multi-agente que:

- Tiene memoria persistente (Engram)
- Usa múltiples agentes especializados
- Se conecta a herramientas externas vía MCP
- Sigue metodología SDD
- Tiene CI/CD pipeline
- Implementa seguridad por diseño

2.9 Requisitos Previos (¡Menos de lo que Crees!)

2.9.1 Prerrequisitos Mínimos (¡Seguro que los tienes!)

- Saber encender una computadora
- Saber usar un navegador web (Chrome, Firefox, etc.)
- Saber copiar y pegar texto
- Saber crear una cuenta de email
- Saber leer y escribir en español

2.9.2 ¿No sabes programar? ¡No importa!

Este curso está diseñado **desde cero absoluto**. No asumimos que sepas:

- Programación (te enseñamos desde lo básico)
- Terminal/consola (te guiamos paso a paso)
- Git o control de versiones (lo explicamos cuando lo necesites)

2.9.3 ¿Ya sabes programar?

Salta al Nivel 2 después de revisar los prerrequisitos.

2.9.4 Verifica si Estás Listo

Antes de empezar, completa el **Test de Prerrequisitos** — ¡toma solo 5 minutos!

2.9.5 Herramientas Necesarias (Gratis)

- **Navegador web** (Chrome, Firefox, etc.) — ¡ya lo tienes!
- **Cuenta de email** (Gmail, Outlook, etc.) — ¡ya la tienes!
- **Editor de código**: VS Code (recomendado, gratuito)
- **Cuenta en al menos una IA**: Claude, ChatGPT, Gemini, etc.

2.9.6 Asistentes de IA que Dominarás

En este curso aprenderás a usar múltiples asistentes de IA para desarrollo:

Herramienta	Tipo	Mejor para	¿Necesitas experiencia previa?
OpenCode	Terminal Agent	Flexibilidad, 75+ proveedores	No

Herramienta	Tipo	Mejor para	¿Necesitas experiencia previa?
KiloCode	IDE + CLI	Multi-modelo (500+), orquestación	No
Claude Code	Terminal Agent	Debugging complejo, 1M contexto	No
Gemini CLI	Terminal Agent	Coste cero, 1M contexto	No
Kiro	Spec-Driven IDE	Desarrollo dirigido por especificaciones	No
Amp	Terminal Agent	Tareas rápidas, ligero	No
GitHub Copilot	IDE Plugin	Integración GitHub, Enterprise	No
Codex CLI	Terminal Agent	Código OpenAI, agente local	No

No necesitas experiencia previa — te enseñamos desde cómo abrir la terminal hasta orquestar sistemas multi-agente.

2.9.7 Herramientas que Instalarás (guiado)

```
# Gentle AI Stack (instalador automático)
curl -fsSL https://raw.githubusercontent.com/Gentleman-Programming/gentle-ai/main/scripts

# Agentes
- Claude Code
- OpenCode
- Cursor

# Memoria
- Engram (Go binary)

# Herramientas de desarrollo
- Git
- Docker (opcional, Nivel 6)
```

2.10 Comienza tu Evolución

2.10.1 Ruta de Aprendizaje

```
1   El Demo en la Cueva     2   El Mark I     3   El Mark III

[Supervivencia]             [Prototipo]          [Combate Real]


4   JARVIS Avanzado     5   Ultron     6   El Nanotech

[Memoria]                [Poder]          [Producción]


                        Innovator
                  (Proyecto Final Completado)
```

2.10.2 Tu Primer Paso

→ Nivel 1: El Demo en la Cueva

2.11 Recursos Adicionales

2.11.1 Comunidad

- Discord: Stark Industries Dev Community
- Twitter: #IronManEvolution
- Soporte: tony@starkindustries.dev

2.11.2 Referencias

- Gentle AI Stack
- Gentleman Skills
- Cómo ser Tony Stark con IA
- Especificación MCP
- Spec-Driven Development

2.12 Acerca del Autor

Diego Saavedra García

Profesor universitario (ESPE, UIDE, Abacom, Codings Academy) con experiencia en:

- Desarrollo FullStack
- Ethical Hacking y Seguridad Ofensiva
- Inteligencia Artificial aplicada
- Arquitectura de Software

> *"No enseño tecnología. Enseño a pensar como un ingeniero que usa IA para amplificar su intelecto."*

2.13 Objetivo Final

Al completar **Iron Man Evolution**, serás capaz de:

1. **Diseñar** sistemas multi-agente desde cero
2. **Implementar** memoria persistente para IA
3. **Orquestar** múltiples agentes especializados
4. **Conectar** IA con herramientas externas vía MCP
5. **Aplicar** metodología SDD en proyectos reales
6. **Desplegar** sistemas de producción robustos
7. **Pensar** como Tony Stark: creativo, sistemático, innovador

---

¿Estás listo para construir tu primera armadura?
→ **Comienza el Nivel 1**

"Un hombre puede hacer lo que otro hombre puede hacer. Si no puedes hacerlo, es porque no has aprendido cómo hacerlo todavía."
— Howard Stark

3 Iron Man Evolution

De la Cueva al Nanotech

4 Dedicatoria

"A todos los que alguna vez se encontraron en una cueva sin recursos, pero con la determinación de construir algo extraordinario."

A ti, que has llegado hasta aquí con nada más que tu curiosidad y la negativa a aceptar que "así nomás" es suficiente.

Este libro está dedicado a quienes bangun a las 2am debugueando porque no pueden dormir hasta resolver ese bug. A los que publican su código aunque nadie lo vea. A los que preguntan "y si lo hago diferente?" cuando todos dicen que es imposible.

Tony Stark no eligió la cueva. Pero eligió qué hacer dentro de ella.

Esa elección es lo que te trajo hasta aquí.

Gracias por elegir construir.

— Diego "Statick" Saavedra García
Marzo 2026

5 Prefacio

5.1 Por qué escribí este libro

Hace años, cuando empecé en esto del desarrollo, creía que ser buen programador significaba saber más sintaxis que los demás. Saber más frameworks. Tener más líneas de código en GitHub.

Equivocado.

Lo que realmente distingue a un desarrollador excepcional no es qué sabe — es cómo piensa. Cómo aborda problemas. Cómo collaborator con herramientas, con equipos, con la incertidumbre.

Durante años, observè a los mejores desarrolladores y me preguntè: ¿qué tienen que yo no tenga? La respuesta no era talento. Era un modelo mental. Una forma de ver el desarrollo que no se enseña en tutoriales ni en bootcamps.

Iron Man Evolution es el libro que me hubiera gustado tener cuando empezaba. No teacha JavaScript, Python ni TypeScript. Enseña a pensar como quien construye sistemas que importan. Como quien tiene una visión y la ejecuta, pieza por pieza, sin perder el norte.

5.2 Cómo usar este libro

Este no es un libro para leer pasivamente. Es un libro para trabajar.

Cada nivel tiene:

1. **Contenido conceptual** — los fundamentos que necesitas entender
2. **Laboratorio práctica** — donde implementas lo aprendido
3. **Boss Fight** — el desafío que separa a quienes "entienden" de quienes "pueden"

> **Consejo**: No salts capítulos. El libro está diseñado con una progresión donde cada nivel construye sobre el anterior. Siintentásskip, vas a perder contexto importante.

5.3 Requisitos previos

Para aprovechar este libro necesitas:

- **Conocimiento básico de programación** — al menos un lenguaje (cualquiera sirve)
- **Una cuenta de GitHub** — porque vamos a usar GitHub Copilot y MCP
- **Ganas de construir algo que importe** — esto no es para quienes buscan shortcuts

No necesitas: - Ser experto en IA — vamos a construir esa habilidad juntos - Conocer Marvel o Iron Man — aunque , no es obligatorio - Un setup perfecto — empezarás con lo básico y creces

---

5.4 Estructura del libro

Nivel	Título	Enfoque
1	El Demo en la Cueva	Prompt Engineering Básico
2	El Mark I	Automatización y Scripts
3	De la Chatarra al Artefacto	Refactoring con IA
4	El Jarvis Primordial	Agentes y Workflows
5	De Agente a Arquitecto	SDD y Arquitectura
6	El Nanotech Viviente	Sistemas Multi-Agente

5.5 Cómo contactar al autor

¿Preguntas? ¿Comentarios? ¿Erratas que encontrar?

- **GitHub**: github.com/statick88
- **LinkedIn**: linkedin.com/in/diego-saavedra-developer
- **Twitter/X**: García (2026)
- **Web**: statick88.github.io

También puedes abrir un issue en el repositorio del libro:
github.com/statick88/curso-iron-man-evolution

5.6 Un último pensamiento

Tony Stark no tenía un manual cuando despertó en esa cueva. Tenía un problema, herramientas limitadas, y la determinación de no morir ahí.

Vos tampoco necesitás un manual perfecto. Necesitás empezar.

Bienvenido a **Iron Man Evolution**.

— Diego Saavedra García
Marzo 2026

6 Agradecimientos

Este libro no existiría sin el apoyo, inspiración y paciencia de muchas personas que me acompañaron en este camino. Gracias a cada uno de ustedes.

6.1 Mentores y Colegas

6.1.1 Luis Jaramillo — ISCD

Mentor, colega y amigo. Luis me enseñó que la ciberseguridad no es solo técnica — es una forma de pensar. Su visión del ISCD como centro de excelencia ha sido inspiración para cada página de este libro. Gracias por creer en mí cuando ni yo creía.

6.2 Estudiantes

A todos los estudiantes que han pasado por mis aulas y me han enseñado tanto como yo a ellos:

6.2.1 ESPE (Escuela Superior Politécnica del Ejercito)

Por la paciencia con la que me bancaron cuando explicaba conceptos “diferentes”. Ustedes saben a qué me refiero.

6.2.2 UIDE (Universidad Internacional de Ecuador)

Por aceptarme como profesor cuando estaba en plena transición de carrera. Cada clase fue un aprendizaje bilateral.

6.2.3 ABACOM

Por la confianza depositada en un approach poco convencional. Las conversaciones en los pasillos fueron tan valiosas como las clases formales.

6.3 Comunidad Gentleman Programming

A todo el equipo de **Gentleman Programming** — donde la filosofía "código limpio, mentalidad clara" no es solo un slogan. En especial a quienes me bancaron en las ideas locas de IA colaborativa antes de que estuviera de moda.

La comunidad que construimos juntos es prueba de que el desarrollo de software puede ser diferente.

6.4 Família

A mi família — porque si, soy el typescript de la familia, siempre está tarde, pero siempre vuelve.

Gracias por soportar las noches de debug, los fines de semana de grabación, y las conversaciones que empiezan con "y si…" y terminan 3 horas después.

Ustedes son mi Mark 50 — la armadura que me protege cuando todo parece imposible.

6.5 A la IA

También quiero agradecer a las herramientas de IA que me acompañaron en este proceso. A ChatGPT por las conversaciones a las 2am. A Claude por los momentos de "eso no es una buena idea, Diego". A GitHub Copilot por todas las veces que completó mi código antes de que yo supiera qué quería escribir.

Las IAs no son el futuro — son el presente. Y este libro existiría sin ellas.

6.6 Y a vos

Sí, vos que estás leyendo esto.

Gracias por elegir este camino. Gracias por no conformarte con "así nomás funciona". Gracias por creer que podés construir algo extraordinario, aunque estés en tu propia cueva.

El libro está dedicado a vos.

Con gratitud,
Diego "Statick" Saavedra García
Marzo 2026

7 Antes de Empezar: Prerrequisitos y Warm-up

8 Antes de Empezar: Prerrequisitos y Warm-up

8.1 ¿Estás listo para ser Tony Stark?

8.2 Objetivo de esta Sección

Antes de convertirte en el **Orquestador Principal de un sistema Multi-Agente**, necesitas verificar que tienes los conocimientos básicos. **No te preocupes si no los tienes** — te proporcionamos los recursos para aprenderlos.

8.3 Lista de Verificación: ¿Tienes lo Básico?

8.3.1 Nivel 0: Conocimientos de Computación Básica

Para comenzar este curso, necesitas saber:

#	Habilidad	¿La tienes?		Recurso si no la tienes
1	Encender una computadora	Sí	No	Curso Básico de Computación
2	Usar un navegador web	Sí	No	Internet Básico
3	Crear una cuenta de email	Sí	No	Gmail Tutorial
4	Usar una calculadora	Sí	No	Matemáticas Básicas
5	Leer y escribir en español	Sí	No	Khan Academy Español

Si marcaste "No" en alguna: ¡No pasa nada! Haz clic en el recurso y aprende. Tómate tu tiempo.

8.3.2 Nivel 1: Conocimientos de Programación Básica

Para este curso, necesitas saber **lo mínimo** de programación:

#	Habilidad	¿La tienes?	Recurso si no la tienes
1	**¿Qué es un archivo?**	Sí No	Archivos y Carpetas
2	**¿Qué es un programa?**	Sí No	¿Qué es un Software?
3	**Copiar y pegar texto**	Sí No	Atajos de Teclado
4	**Instalar un programa**	Sí No	Instalar Software
5	**Abrir una terminal/consola**	Sí No	Terminal para Principiantes

Mínimo necesario: Si sabes copiar/pegar e instalar programas, ¡puedes empezar!

8.4 Tutorial: Tu Primera Terminal

8.4.1 ¿Qué es una terminal/consola?

La terminal es como el **taller de Tony Stark** — un lugar donde le das órdenes directamente a la computadora.

8.4.1.1 En Windows

```
1. Presiona la tecla Windows
2. Escribe "cmd" o "PowerShell"
3. Presiona Enter
```

8.4.1.2 En Mac

```
1. Presiona Command + Space
2. Escribe "Terminal"
3. Presiona Enter
```

8.4.1.3 En Linux

```
1. Busca "Terminal" en tu menú
2. O presiona Ctrl + Alt + T
```

8.4.2 Primeros Comandos para Probar

```
# Ver en qué carpeta estás
pwd

# Ver qué archivos hay
ls   # o dir en Windows

# Crear una carpeta
mkdir mi-proyecto

# Ir a una carpeta
cd mi-proyecto

# Volver atrás
cd ..
```

¿No funciona? No te preocupes — **en este curso no necesitas ser experto en terminal**. Te guiaremos paso a paso.

8.5 Conceptos que Aprenderás desde Cero

8.5.1 1. ¿Qué es Inteligencia Artificial (IA)?

Analogía: La IA es como un **aprendiz muy listo**. No piensa como humano, pero ha leído millones de libros y puede predecir qué palabra viene después.

```
Tú: "El cielo es..."
IA: "azul" (porque ha leído esa frase millones de veces)
```

8.5.2 2. ¿Qué es un Prompt?

Analogía: Un prompt es como una **receta de cocina**. Le dices a la IA qué quieres hacer, con qué ingredientes, y cómo quieres que quede.

```
Prompt: "Escribe una función en Python que sume dos números"
```

8.5.3 3. ¿Qué es un Agente de IA?

Analogía: Un agente es como un **asistente personal con superpoderes**. No solo responde preguntas, sino que puede: - Leer archivos - Escribir código - Ejecutar programas - Buscar en internet - Tomar decisiones

8.5.4 4. ¿Qué es MCP (Model Context Protocol)?

Analogía: MCP es como un **adaptador universal**. Permite que tu agente de IA se conecte con cualquier herramienta (GitHub, base de datos, Slack, etc.) usando el mismo conector.

8.5.5 5. ¿Qué es Engram (Memoria Persistente)?

Analogía: Engram es como un **cuaderno de notas infinito**. La IA puede recordar lo que aprendió ayer, la semana pasada, el mes pasado.

8.6 Ruta de Aprendizaje Recomendada

8.6.1 Si NO sabes programar (0% experiencia)

1. Curso de Python para Principiantes (3-5 días)
2. Git Tutorial para Principiantes (2-3 días)
3. **Comienza Iron Man Evolution Nivel 1**

8.6.2 Si SABES programar pero NO has usado IA

1. **Comienza directamente Iron Man Evolution Nivel 1**
2. Tómate tu tiempo con los ejercicios básicos
3. No saltes al Nivel 3 directamente

8.6.3 Si ya has USADO IA pero NO para programar

1. **Comienza Iron Man Evolution Nivel 1** (para entender fundamentos)
2. Salta al Nivel 2 después del Lab 1
3. Considera el Nivel 3 si ya te sientes cómodo

8.6.4 Si ya has USADO IA para programar

1. **Comienza Iron Man Evolution Nivel 2** (Context Engineering)
2. El Nivel 1 es opcional pero recomendado para repasar fundamentos

---

8.7 Herramientas que Necesitarás

8.7.1 Mínimo Absoluto (Gratis)

Herramienta	Para qué sirve	Enlace
Navegador web	Acceder a ChatGPT, Claude, Gemini	Chrome o Firefox
Cuenta de email	Registrarte en las herramientas de IA	Gmail o Outlook
Editor de texto	Escribir código	VS Code (recomendado)

8.7.2 Recomendado (A partir del Nivel 3)

Herramienta	Para qué sirve	Enlace
Git	Control de versiones	Git
Node.js	Ejecutar herramientas de IA	Node.js
Python	Lenguaje de programación	Python

8.8 Ejercicio de Warm-up: Tu Primera Interacción con IA

8.8.1 Paso 1: Elige tu Herramienta de IA

Abre una de estas (todas son gratuitas): - ChatGPT (OpenAI) - Claude (Anthropic) - Gemini (Google)

8.8.2 Paso 2: Tu Primer Prompt

Copia y pega este texto:

```
Hola. Soy completamente nuevo en programación e IA.
Explícame qué eres tú (una IA) en 3 frases simples,
como si fueras un profesor explicando a un niño de 10 años.
```

8.8.3 Paso 3: Observa la Respuesta

Lee lo que la IA te responde. **No te preocupes si no entiendes todo** — eso es normal.

8.8.4 Paso 4: Tu Segundo Prompt

Ahora prueba esto:

```
Ahora dame 3 consejos para alguien que empieza a aprender
programación e IA hoy.
```

8.8.5 Paso 5: ¡Felicidades!

Si pudiste leer las respuestas, **ya completaste tu primera interacción con IA**. Eso es literalmente lo que harás en el Nivel 1, pero con más profundidad.

8.9 Auto-evaluación: ¿Estás listo para Nivel 1?

8.9.1 Respuestas honestas

Pregunta	Sí	No
¿Puedes abrir un navegador web?		
¿Puedes crear una cuenta de email?		
¿Puedes copiar y pegar texto?		
¿Puedes instalar un programa?		
¿Interactuaste con una IA en el warm-up?		
¿Entendiste aproximadamente qué es un prompt?		

8.9.2 Resultados

- **6 Sí**: ¡Listo! Comienza el **Nivel 1** ahora
- **4-5 Sí**: Casi listo. Repasa los recursos en "Si marcaste No"
- **2-3 Sí**: Tómate 1-2 días con los recursos básicos
- **0-1 Sí**: No te preocupes. Empieza con los recursos y vuelve cuando te sientas cómodo

---

8.10 ¿Qué vas a aprender en Iron Man Evolution?

8.10.1 De Cero a Héroe en 6 Niveles

```
NIVEL 1: "¿Qué es esto?"
    ↓ Primer prompt, primer script
NIVEL 2: "¿Cómo lo hago mejor?"
    ↓ Contexto específico, reglas
NIVEL 3: "¿Cómo automatizo?"
    ↓ Primer agente, delegación
NIVEL 4: "¿Cómo me conecto?"
    ↓ MCP, memoria, herramientas
NIVEL 5: "¿Cómo escalo?"
    ↓ Multi-agente, SDD, controles
NIVEL 6: "¿Cómo lo hago profesional?"
    ↓ Sistema completo, producción
```

8.10.2 Al finalizar, podrás:

1. Crear agentes de IA que trabajen por ti
2. Conectar IA con GitHub, bases de datos, Slack
3. Recordar todo lo que aprendiste (memoria persistente)
4. Orquestar múltiples agentes trabajando juntos
5. Desplegar sistemas de IA a producción
6. Pensar como Arquitecto de Software Senior

8.11 Siguiente Paso

Si completaste el warm-up y te sientes cómodo:

Comienza el Nivel 1: El Demo en la Cueva

Si necesitas más práctica con lo básico:

Recursos recomendados: - Khan Academy - Computación Básica - FreeCodeCamp - Python para Principiantes - Codecademy - Introducción a la Programación

---

"No necesitas ser un genio para empezar. Solo necesitas la curiosidad de un niño y la determinación de un ingeniero." — Diego Saavedra García

Part I

Tu Arsenal: Herramientas de IA

9 Resumen: Stack Completo de Herramientas IA

10 Stack Completo de Herramientas IA

10.1 Tu Arsenal de Tony Stark

10.2 Matriz de Decisión

```
¿NECESITÁS HERRAMIENTA DE IA?

                    ¿CUÁL ES TU PRIORIDAD?

 COSTO       POTENCIA    VELOCIDAD   ORQUESTACIÓN  ESPECIFICACIÓN

Gemini      Claude       Amp         KiloCode     Kiro
OpenCode    Code                     OpenCode     (SDD)
```

10.3 Comparación Detallada

10.3.1 Tabla Comparativa Completa

Característica	Open-Code	Claude Code	Gemini CLI	KiloCode	Kiro	Amp
Precio	Gratis (BYO)	$20-200/mes	**Gratis**	$19/mes	$20-200/mes	Gratis

Característica	OpenCode	Claude Code	Gemini CLI	KiloCode	Kiro	Amp
Modelos	75+	Solo Claude	Solo Gemini	500+	AWS Models	Solo Claude
Contexto	Según modelo	**1M tokens**	**1M tokens**	Según modelo	Full project	Según modelo
Subagentes						
Orquestación		Básica		Completa		
Specs/SDD						
Hooks						
Memory Bank						
Open Source				Parcial		
MCP						
AGENTS.md					(Specs)	
Configuración	Alta	Media	**Baja**	Alta	Alta	**Mínima**
Curva aprendizaje	Media	Media	**Baja**	Alta	Alta	**Baja**

10.4 Guía de Selección

10.4.1 Por Situación

Situación	Herramienta	Por Qué
Budget cero	Gemini CLI o Amp	Gratuitas, sin límites severos
Empezar a aprender	Gemini CLI	Cero configuración
Flexibilidad máxima	OpenCode	75+ modelos, open source
Código complejo	Claude Code	Mejor razonamiento, 1M ctx
Multi-agente	KiloCode	Orchestrator Mode
Debugging profundo	Claude Code	Subagentes, análisis profundo
Spec-Driven Development	Kiro	Paradigma nativo SDD
Tareas rápidas	Amp	Ligero, inmediato
Producción	Claude Code + OpenCode	Combinar potencias

10.4.2 Por Nivel de Experiencia

```
NIVEL 1: PRINCIPIANTE


Herramienta: Gemini CLI o Amp
Objetivo: Aprender a conversar con IA
Tiempo de setup: 5 minutos
Costo: $0



    "Hola JARVIS, explicame qué es..."
```

```
NIVEL 2: EN USADO REGULAR


Herramienta: OpenCode
Objetivo: Flexibilidad, múltiples modelos
Tiempo de setup: 30 minutos
Costo: Solo API keys



    AGENTS.md + 75+ modelos = TU ARMA PERSONAL
```

```
NIVEL 3: PRODUCTIVO


Herramientas: Claude Code + OpenCode
Objetivo: Potencia + Flexibilidad
Uso: Claude para complejos, OpenCode para rápido
```

```
    CLAUDE CODE: Debugging, arquitectura, subagentes
    OPENCODE: Quick fixes, múltiples modelos




NIVEL 4: EXPERTO / EQUIPO


Herramientas: KiloCode o Stack Completo
Objetivo: Multi-agente, orquestación
Uso: Equipos, proyectos grandes


    ORCHESTRATOR: "Coordiná 5 agentes en paralelo"
```

10.5 Configuración Recomendada por Uso

10.5.1 Setup Mínimo (Gratis)

```bash
# 1. Gemini CLI - Para empezar
npm install -g @google/gemini-cli

# 2. Amp - Para tareas rápidas
npm install -g @anthropic-ai/amp

# Costo total: $0
```

10.5.2 Setup Profesional (~$10/mes)

```bash
# 1. OpenCode - Tu agente principal
curl -fsSL https://opencode.ai/install | bash
```

```
# 2. API Keys necesarias
# - Anthropic: $5/mes (crédito inicial gratis)
# - OpenAI: $5/mes (crédito inicial gratis)

# Costo total: ~$10/mes
```

10.5.3 Setup Avanzado (~$50/mes)

```
# 1. OpenCode - Flexibilidad
# 2. Claude Code - Potencia
npm install -g @anthropic-ai/claude-code

# 3. KiloCode - Orquestación
code --install-extension kilo-code.kilo-code

# Costo total: ~$50/mes
```

10.5.4 Setup Enterprise (~$200/mes)

```
# Stack completo
# - Claude Code (Opus)
# - Kiro
# - KiloCode Pro
# - OpenCode enterprise

# Costo total: ~$200/mes
```

10.6 Estructura de Configuración Multi-Herramienta

```
mi-proyecto/
   .opencode/
      opencode.json
      AGENTS.md
   .claude/
      config.json
      AGENTS.md
   .kilo/
      agents.json
      memory/
   .kiro/
```

```
    specs/
  AGENTS.md              # Global para todas
  .env                   # API keys (NO COMMIT)
  .gitignore
```

10.7 Seguridad: API Keys

10.7.1 Variables de Entorno

```bash
# ~/.bashrc o ~/.zshrc

# OpenCode / Claude Code
export ANTHROPIC_API_KEY="sk-ant-xxx"

# OpenCode
export OPENAI_API_KEY="sk-xxx"

# Gemini CLI
export GOOGLE_API_KEY="AIza-xxx"

# KiloCode
export KILO_CODE_API_KEY="kilo-xxx"
```

10.7.2 .gitignore

```
# API Keys
.env
.env.*
*.env

# Credenciales
**/credentials.json
**/secrets/**
**/*.pem
**/id_rsa*
**/id_ed25519*
```

10.8 Taller: Arma Tu Stack

10.8.1 Ejercicio: Armá tu arsenal personalizado

Paso 1: Evaluá tu situación

```
Mi presupuesto: _____________
Mi experiencia: _____________
Mi objetivo principal: _____________
```

Paso 2: Elegí herramientas

```
Herramienta 1 (principal): _____________
Herramienta 2 (alternativa): _____________
Herramienta 3 (tareas rápidas): _____________
```

Paso 3: Configurá

```
Completá la configuración de cada herramienta
```

Paso 4: Integrá

```
Creá tu AGENTS.md global
Configurá MCP si aplica
```

10.9 Roadmap de Aprendizaje

```
MES 1: FUNDAMENTOS
   Semana 1-2: Gemini CLI (gratis, cero config)
   Semana 3-4: OpenCode (configuración básica)
   Resultado: Puedo usar IA para tareas simples

MES 2: PROFESIONALIZACIÓN
   Semana 5-6: OpenCode avanzado (AGENTS.md, permisos)
   Semana 7-8: Claude Code (subagentes)
   Resultado: Tengo un workflow profesional

MES 3: MAESTRÍA
   Semana 9-10: KiloCode (orquestación)
   Semana 11-12: Integración completa
   Resultado: Orquesto múltiples agentes
```

```
MES 4+: ESPECIALIZACIÓN
  Kiro (SDD) o
  Multi-stack o
  Proyecto real
  Resultado: Experto
```

10.10 Preguntas Frecuentes

10.10.1 ¿Puedo usar múltiples herramientas a la vez?

Sí. Cada herramienta tiene su strengths. Combiná según necesidad:

- Claude Code para debugging complejo
- OpenCode para flexibilidad
- Amp para tareas rápidas
- KiloCode para multi-agente

10.10.2 ¿Cuánto cuesta usar estas herramientas?

Herramienta	Costo Real
Gemini CLI	$0 (límites gratis)
Amp	$0 + API (~$5/mes)
OpenCode	$0 + API (~$10/mes)
Claude Code	$20-200/mes
KiloCode	$19/mes + API
Kiro	$20-200/mes

10.10.3 ¿Cuál es la mejor para principiantes?

Gemini CLI. Cero configuración, gratis, 1M tokens.

10.10.4 ¿Cuál es la mejor para producción?

Claude Code + OpenCode. Combinas potencia con flexibilidad.

10.10.5 ¿Necesito todas?

No. Empezá con una, dominála, después expandí.

10.11 Checklist de Setup

Herramienta	Instalada	Configurada	Probada
Gemini CLI			
Amp			
OpenCode			
Claude Code			
KiloCode			
Kiro			

10.12 Recursos

10.12.1 Documentación Oficial

Herramienta	Docs
OpenCode	opencode.ai
Claude Code	docs.anthropic.com
Gemini CLI	ai.google.dev
KiloCode	kilocode.ai
Kiro	kiro.aws
Amp	anthropic.com/amp

10.12.2 Comunidad

Herramienta	Comunidad
OpenCode	Discord, GitHub
Claude Code	Discord, Reddit
Gemini CLI	Discord
KiloCode	Discord
Kiro	AWS Community

Filosofía:

"No se trata de tener todas las herramientas. Se trata de tener las correctas para cada situación y dominarlas."

— Tony Stark

11 Unidad 2: OpenCode - Tu Primer Agente de IA

12 Unidad 2: OpenCode — Tu Primer Agente de IA

12.1 "Tony no construye herramientas. Elige las correctas y las configura."

12.2 Objetivo

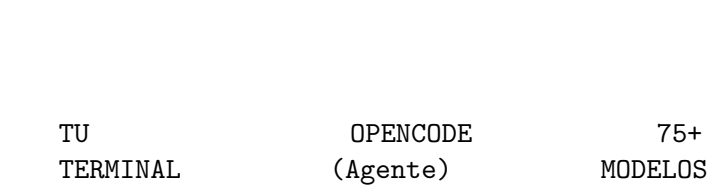
Al terminar esta unidad podrás:

- **Instalar** OpenCode en tu sistema
- **Configurar** proveedores de IA (OpenAI, Anthropic, Google)
- **Personalizar** tu AGENTS.md
- **Entender** cada setting y por qué importa
- **Ejecutar** tu primer proyecto con contexto

12.3 ¿Qué es OpenCode?

12.3.1 En una frase

OpenCode es un agente de terminal que ejecuta comandos de IA en tu proyecto.



```
Vos                 Lee                 Claude,
Escribes            Código              GPT,
Prompts             + Context           Gemini...
```

12.3.2 ¿Por qué OpenCode y no otra herramienta?

Característica	OpenCode	ChatGPT	Claude Code
Open Source	MIT		
75+ Modelos		(solo GPT)	(solo Claude)
Gratis	(BYO API)	($20+/mes)	($20+/mes)
Tu Código	Lee todo	(web)	
Configurable	Total	Limitado	Medio
AGENTS.md			

BYO API = Bring Your Own API Key. Pagás solo el modelo, no la herramienta.

12.4 Instalación de OpenCode

12.4.1 Timeline

Paso	Descripción	Tiempo
1	Verificar prerrequisitos	2 min
2	Instalar OpenCode	5 min
3	Configurar API Keys	5 min
4	Primera ejecución	2 min

12.4.2 Paso 1: Verificar Prerrequisitos

Tu tarea: Verificar que tienes lo necesario.

Comando a ejecutar:

```
node --version && npm --version
```

Resultado esperado:

```
v18.x.x o superior
v9.x.x o superior
```

Si no tienes Node.js: 1. Ve a nodejs.org 2. Descarga la versión LTS 3. Instálala 4. Verifica nuevamente

---

12.4.3 Paso 2: Instalar OpenCode

Tienes **dos opciones**:

12.4.3.1 Opción A: Con curl (Linux/Mac/WSL)

Hint: El instalador oficial usa el dominio opencode.ai

Comando parcial: `curl -fsSL https://opencode.ai/install |`

Completá el comando:

```
[ESCRIBE AQUÍ]
```

Resultado esperado:

```
  OpenCode installed successfully!
Run: opencode
```

12.4.3.2 Opción B: Con npm (Todas las plataformas)

Hint: El flag -g significa instalación global

Comando parcial: `npm install -g @`

Completá el comando:

```
[ESCRIBE AQUÍ]
```

Resultado esperado:

```
added X packages in Ys
```

12.4.3.3 Opción C: Con Homebrew (Mac)

Hint: brew install seguido del nombre del paquete

Comando parcial: `brew install an`

Completá el comando:

```
[ESCRIBE AQUÍ]
```

Resultado esperado:

```
==> Downloading https://...
==> Installing opencode...
```

12.4.4 Paso 3: Verificar Instalación

Comando:

```
opencode --version
```

Resultado esperado:

```
X.X.X
```

12.5 Configuración de API Keys

12.5.1 ¿Por qué necesitas API Keys?

```
OPENCODE no es gratis porque:

- La HERRAMIENTA es gratis (MIT)
- Los MODELOS de IA NO lo son

Vos pagás SOLO el modelo que uses:

   OpenAI              Anthropic           Google
   GPT-4o              Claude 4            Gemini 2.0
```

`$5/1M tok` `$3/1M tok` `$0.125/1M`

12.5.2 Modelos Recomendados por Budget

Budget	Modelo Recomendado	Costo Aproximado	Mejor Para
$0	Gemini 2.0 Flash	Gratis (150 req/min)	Aprender, practicar
Bajo	Claude Haiku	$0.80/1M tokens	Tareas rápidas
Medio	GPT-4o Mini	$0.15/1M tokens	Desarrollo diario
Alto	Claude Sonnet 4	$3/1M tokens	Código complejo
Máximo	Claude Opus 4	$15/1M tokens	Arquitectura, debugging

12.5.3 Obtener tu API Key

12.5.3.1 Anthropic (Claude) — Recomendado

1. Ve a console.anthropic.com
2. Crea cuenta o inicia sesión
3. Ve a **API Keys** → **Create Key**
4. Copia la key (empieza con `sk-ant-`)

Costo: $5 gratis en créditos para nuevos usuarios

12.5.3.2 OpenAI (GPT-4) — Alternativa

1. Ve a platform.openai.com
2. Crea cuenta o inicia sesión
3. Ve a **API Keys** → **Create new secret key**
4. Copia la key (empieza con `sk-`)

Costo: $5 gratis para nuevos usuarios

12.5.3.3 Google (Gemini) — Gratis

1. Ve a aistudio.google.com
2. Crea cuenta o inicia sesión
3. Ve a **Get API Key** → **Create API Key**
4. Copia la key (empieza con `AIza`)

Costo: GRATIS hasta ciertos límites

12.5.4 Configurar API Keys en tu Sistema

Opción A: Variables de entorno (Recomendado)

Archivo: `~/.bashrc` o `~/.zshrc` (Mac)

Agregá estas líneas:

```
# API Keys para OpenCode
export ANTHROPIC_API_KEY="sk-ant-tu-key-aqui"
export OPENAI_API_KEY="sk-tu-key-aqui"
export GOOGLE_API_KEY="AIza-tu-key-aqui"
```

Comando para recargar:

```
# Linux/Mac
source ~/.bashrc  # o source ~/.zshrc

# Mac con zsh
source ~/.zshrc
```

Opción B: Archivo .env en el proyecto

1. Creá un archivo `.env` en tu proyecto
2. Agregá las keys
3. **Agregá `.env` a tu `.gitignore`**

```
# En tu .gitignore
.env
.env.*
```

NUNCA committees archivos con API keys

12.6 Configuración de OpenCode

12.6.1 Estructura de Archivos

```
~/.config/opencode/
  opencode.json      # Configuración principal
  AGENTS.md          # Tu identidad y reglas
  agents/            # Definiciones de agentes
  config/
     skills/        # Skills disponibles
  mcp/               # MCP servers
```

12.6.2 El archivo opencode.json

12.6.2.1 Tu primera configuración

Ubicación: `~/.config/opencode/opencode.json`

Tu tarea: Crear este archivo con tu editor favorito.

Comando para crear directorio:

```
mkdir -p ~/.config/opencode
```

Comando para crear archivo:

```
touch ~/.config/opencode/opencode.json
```

12.6.3 Explicación de Cada Setting

12.6.3.1 1. Modelo Principal

```
"model": "anthropic/claude-sonnet-4-6-20250501"
```

Parte	Significado
`anthropic`	Proveedor (namespace)
`claude-sonnet-4-6-20250501`	Modelo específico

Modelos disponibles:

Proveedor	Modelo	Uso
`anthropic`	`claude-opus-4-6-20250501`	Mejor para arquitectura
`anthropic`	`claude-sonnet-4-6-20250501`	Balanceado, recomendado
`anthropic`	`claude-haiku-4-5-20250501`	Rápido, barato
`openai`	`gpt-4o`	General, rápido
`google`	`gemini-2.0-flash`	Gratis, rápido
`google`	`gemini-2.5-pro`	Mejor de Google

12.6.3.2 2. Actualización Automática

```
"autoupdate": true
```

Valor	Significado
`true`	Se actualiza automáticamente
`false`	Vos controlás las actualizaciones

12.6.3.3 3. Puerto del Servidor

```
"server": {
  "port": 4096
}
```

¿Qué es esto? OpenCode puede correr como servidor HTTP para integraciones.

Valor	Uso
`4096`	Puerto por defecto
`8080`	Común para desarrollo
`3000`	Común para frontend

12.6.3.4 4. Instrucciones Globales

```
"instructions": [
  "~/.config/opencode/mcp-security-resources.md"
]
```

¿Qué es esto? Archivos que OpenCode siempre lee antes de empezar.

Ejemplo de archivo de instrucciones:

```
# Instrucciones de Seguridad

## Reglas de Oro

1. NUNCA ejecutes comandos destructivos sin preguntar
2. NUNCA leas archivos .env o secrets
3. SIEMPRE pregunta antes de git push
```

12.6.4 Permisos: La Parte Más Importante

```
"permission": {
  "bash": { ... },
  "read": { ... },
  "write": { ... }
}
```

12.6.4.1 Permisos de Bash

```
"bash": {
  "git status": "allow",
  "git diff *": "allow",
  "rm -rf *": "deny",
  "rm *": "ask",
  "git push": "ask",
  "*": "deny"
}
```

Acción	Significado
`"allow"`	Se ejecuta sin preguntar
`"ask"`	Pregunta antes de ejecutar
`"deny"`	Nunca se ejecuta

Tus permisos personalizados:

```
"bash": {
  "git status": "allow",
  "git diff": "allow",
  "git log": "allow",
  "git branch": "allow",
  "npm *": "allow",
  "npm run *": "allow",
  "mkdir *": "allow",
  "touch *": "allow",
  "rm -rf *": "deny",
  "rm *": "ask",
  "git commit *": "ask",
  "git push": "ask",
  "sudo *": "ask",
  "*": "deny"
}
```

12.6.4.2 Permisos de Lectura

```
"read": {
  "**/.env": "deny",
  "**/.env.*": "deny",
  "**/credentials.json": "deny",
  "**/secrets/**": "deny",
  "**/id_rsa*": "deny",
  "*.env": "deny",
  "*": "allow"
}
```

¿Por qué deny en .env? Son archivos con passwords y API keys. OpenCode NO necesita leerlos para hacer su trabajo.

Tus permisos personalizados:

```
"read": {
  "**/.env": "deny",
  "**/.env.*": "deny",
  "**/credentials.json": "deny",
  "**/*secret*": "deny",
  "**/*password*": "deny",
  "**/id_rsa*": "deny",
  "**/id_ed25519*": "deny",
  "*.pem": "deny",
  "*": "allow"
}
```

12.6.4.3 Permisos de Escritura

```
"write": {
  "**/.env": "deny",
  "**/.env.*": "deny",
  "**/credentials.json": "deny",
  "**/secrets/**": "deny",
  "*.env": "deny",
  "*.pem": "deny"
}
```

¿Por qué deny en .env? Nadie debería modificar archivos de secrets automáticamente.

12.6.5 MCP Servers: Conexiones Externas

```
"mcp": {
  "chrome-devtools": {
    "command": ["npx", "-y", "chrome-devtools-mcp@latest"],
    "type": "local"
  },
  "context7": {
    "enabled": true,
    "type": "remote",
    "url": "https://mcp.context7.com/mcp"
  },
  "engram": {
    "command": ["engram", "mcp"],
    "enabled": true,
    "type": "local"
  }
}
```

Server	Propósito	Tipo
`chrome-devtools`	Controlar Chrome para debugging	Local
`context7`	Buscar documentación actualizada	Remote
`engram`	Memoria persistente	Local

12.7 AGENTS.md: Tu Identidad

12.7.1 ¿Qué es AGENTS.md?

AGENTS.md es el archivo donde definís QUIÉN sos y CÓMO querés que OpenCode trabaje.

```
AGENTS.md = TU PERFIL DE DESARROLLADOR
```

```
# Quién soy
Soy Diego, desarrollador FullStack...

# Reglas de trabajo
- Siempre uso TypeScript strict
- Tests primero, código después
- Nombres descriptivos, sin abreviaciones

# Preferencias
- snake_case para Python
- kebab-case para archivos
- Español para comentarios
```

12.7.2 Tu Primer AGENTS.md

Ubicación: `~/.config/opencode/AGENTS.md` o en tu proyecto

Comando para crear:

```
touch ~/.config/opencode/AGENTS.md
```

12.7.3 Template de AGENTS.md

```
# AGENTS.md - Mi Perfil de Desarrollador

---

##  Quién Soy

**Nombre:** [Tu nombre]
**Rol:** [Tu rol: Frontend Dev, Backend Dev, FullStack, etc.]
**Experiencia:** [Años de experiencia]
**Especialidades:** [Ej: TypeScript, Python, Seguridad, etc.]

---

##  Stack Tecnológico

### Lenguajes Principales
- [Tu lenguaje 1]
- [Tu lenguaje 2]

### Frameworks/Stack
- [Framework 1]
- [Framework 2]

### Herramientas
- Git:
- Docker:  /
- Kubernetes:  /
- CI/CD: [Tu herramienta]

---

##  Reglas de Trabajo

### Obligatorias
1. [Regla 1]
2. [Regla 2]
3. [Regla 3]

### Estilo de Código
- Nombres: [descriptivos, camelCase, snake_case, etc.]
- Comentarios: [en qué idioma]
- Tests: [primero, después, o no?]

### Seguridad
1. Nunca revelar API keys o secrets
2. Siempre validar input de usuarios
3. No ejecutar comandos destructivos sin confirmar
```

```
---

##  Preferencias de Comunicación

### Idioma
- Código y comentarios: [Español, English]
- Nombres de variables: [inglés siempre, o mixto]

### Nivel de Explicación
- Básico: [Para principiantes]
- Intermedio: [Con contexto]
- Avanzado: [Solo lo esencial]

---

##  Límites y Restricciones

### No Hacer
1. [Cosa que NO hacés]
2. [Otra cosa que NO hacés]

### Siempre Preguntar
1. [Situación 1]
2. [Situación 2]

---

##  Tips para Trabajar Conmigo

1. [Tip 1]
2. [Tip 2]
3. [Tip 3]
```

12.8 Lab: Tu Primera Sesión con OpenCode

12.8.1 Objetivo

Ejecutar OpenCode con tu configuración personalizada.

12.8.2 Timeline

Paso	Descripción	Tiempo
1	Crear opencode.json	10 min
2	Crear AGENTS.md	10 min
3	Configurar API Key	5 min
4	Primera sesión	15 min

12.8.3 Escenario

Tony acaba de salir de la cueva. Necesita configurar su taller (OpenCode) para la próxima misión.

Tu turno: Configurar OpenCode como Tony configuraría su armadura.

12.8.4 Paso 1: Crear opencode.json

Archivo: `~/.config/opencode/opencode.json`

Completá este template:

```
{
  "$schema": "https://opencode.ai/config.json",
  "model": "[TU MODELO ELEGIDO]",
  "autoupdate": true,
  "server": {
    "port": 4096
  },
  "instructions": [],
  "mcp": {
    "context7": {
      "enabled": true,
      "type": "remote",
      "url": "https://mcp.context7.com/mcp"
    }
  },
  "permission": {
    "bash": {
      "git status": "allow",
      "git diff *": "allow",
      "git log *": "allow",
      "npm *": "allow",
      "mkdir *": "allow",
      "rm -rf *": "deny",
```

```
      "rm *": "ask",
      "git push": "ask",
      "*": "deny"
    },
    "read": {
      "**/.env": "deny",
      "**/.env.*": "deny",
      "*.env": "deny",
      "*": "allow"
    },
    "write": {
      "**/.env": "deny",
      "*.env": "deny",
      "*.pem": "deny"
    }
  }
}
```

12.8.5 Paso 2: Crear AGENTS.md

Archivo: `~/.config/opencode/AGENTS.md`

Completá tu perfil:

```
# AGENTS.md - [Tu Nombre]

##  Quién Soy
[Tu descripción]

##  Stack
[Lenguajes, frameworks, herramientas]

##  Reglas de Trabajo
1. [Tu regla 1]
2. [Tu regla 2]
3. [Tu regla 3]

##  Preferencias
[Cómo querés que trabaje]

##  Límites
[Qué NO hacer]
```

12.8.6 Paso 3: Configurar API Key

Verificá que tu API key está disponible:

```
echo $ANTHROPIC_API_KEY
```

Resultado esperado:

```
sk-ant-xxxxxxxxxxxxxxxxxxx
```

Si está vacío: 1. Editá tu `~/.bashrc` o `~/.zshrc` 2. Agregá: `export ANTHROPIC_API_KEY="tu-key"` 3. Ejecutá: `source ~/.bashrc`

12.8.7 Paso 4: Tu Primera Sesión

Comando para iniciar:

```
opencode
```

Tu primer mensaje:

```
> Hola. Soy [tu nombre].
> acabo de configurar OpenCode.
> Por favor confirmame que lees mi AGENTS.md
> y decime qué reglas encontraste.
```

Tu respuesta esperada:

```
  Leo tu AGENTS.md
  Reglas que encontré:
1. [Regla 1]
2. [Regla 2]
3. [Regla 3]
```

12.8.8 Reflexión

Completá:

```
Mis reglas definidas fueron:
1. _____________________________________________
2. _____________________________________________
3. _____________________________________________

Algo que me sorprendió de OpenCode:
_____________________________________________

Algo que quiero ajustar:
_____________________________________________
```

12.9 Entregable Final

12.9.1 Archivos a crear:

Archivo	Ubicación	Propósito
`opencode.json`	`~/.config/opencode/`	Configuración
`AGENTS.md`	`~/.config/opencode/`	Tu perfil

12.9.2 Contenido esperado:

```bash
# Verificar archivos
ls -la ~/.config/opencode/

# Deberías ver:
# opencode.json
# AGENTS.md
```

Habilidad	Demostrable cuando...

12.10 Checklist Final

Habilidad	Demostrable cuando...
Instalar	`opencode --version` funciona
Configurar	opencode.json tiene permisos
AGENTS.md	Defines tu perfil completo
API Key	Variable de entorno configurada
Sesión	Pudiste conversar con OpenCode

12.11 Logro

"Armador" desbloqueado:

- +100 XP
- Acceso a herramientas avanzadas
- AGENTS.md personalizado listo para usar

12.12 Siguiente

¿Listo para la próxima herramienta?

→ Unidad 3: Claude Code — Tu JARVIS Potente

¿Necesitás más práctica?

→ Quiz Unidad 2: Configuración de OpenCode

13 Unidad 3: Claude Code — Tu JARVIS Potente

14 Unidad 3: Claude Code — Tu JARVIS Potente

14.1 "Anthropic construyo JARVIS. Vos lo configuras."

14.2 Objetivo

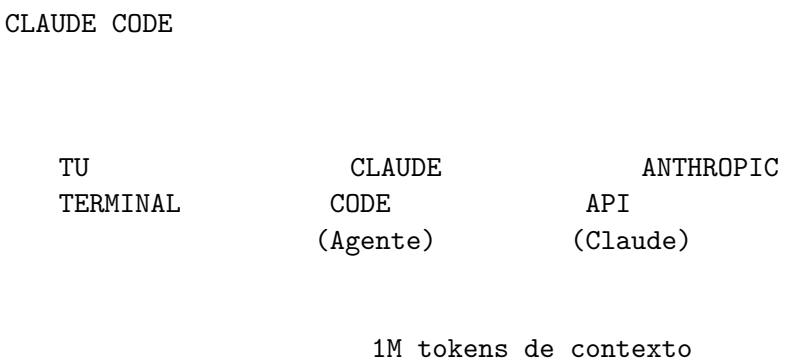
Al terminar esta unidad podrás:

- **Instalar** Claude Code en tu sistema
- **Configurar** Claude Code para tu flujo de trabajo
- **Usar** subagentes para tareas complejas
- **Integrar** con MCP y AGENTS.md
- **Comparar** con OpenCode y elegir según necesidad

14.3 ¿Qué es Claude Code?

14.3.1 En una frase

Claude Code es el agente de terminal oficial de Anthropic. Diseñado para integrarse profundamente con Claude.



```
Subagentes nativos
Integración MCP profunda
```

14.3.2 ¿Por qué Claude Code?

Característica	Claude Code	OpenCode
Modelo	Solo Claude	75+ modelos
Contexto	1M tokens	Según modelo
Subagentes	Nativos	No
Integración MCP	Profunda	
Debugging	Excelente	Bueno
Precio	$20-200/mes	Gratis (BYO)

Cuándo usar cada uno: - **OpenCode:** Flexibilidad, múltiples modelos, budget limitado - **Claude Code:** Proyectos complejos, debugging profundo, subagentes

---

14.4 Instalación de Claude Code

14.4.1 Timeline

Paso	Descripción	Tiempo
1	Verificar prerrequisitos	2 min
2	Instalar Claude Code	5 min
3	Autenticar con Anthropic	5 min
4	Primera ejecución	2 min

14.4.2 Paso 1: Verificar Prerrequisitos

Tu tarea: Verificar que tienes Node.js.

Comando a ejecutar:

```
node --version
```

Resultado esperado:

```
v18.x.x o superior
```

14.4.3 Paso 2: Instalar Claude Code

14.4.3.1 Opción A: Con npm (Recomendado)

Hint: Paquete oficial de Anthropic

Comando parcial: `npm install -g @`

Completá el comando:

```
[ESCRIBE AQUÍ]
```

Resultado esperado:

```
added X packages in Ys
```

14.4.3.2 Opción B: Con curl

Comando parcial: `curl -fsSL https://downloads.anthropic.com/`

Completá:

```
[ESCRIBE AQUÍ]
```

14.4.4 Paso 3: Autenticar

Comando:

```
claude
```

Resultado esperado:

```
  Please authenticate with Anthropic
Visit: https://auth.anthropic.com/
```

1. Abrí el link en tu navegador
2. Iniciá sesión con tu cuenta Anthropic
3. Autorizá el dispositivo
4. Volvé a la terminal

14.4.5 Paso 4: Verificar Instalación

Comando:

```
claude --version
```

Resultado esperado:

```
X.X.X
```

14.5 Configuración de Claude Code

14.5.1 Estructura de Archivos

```
~/.claude/
  config.json           # Configuración principal
  projects/             # Proyectos guardados
  logs/                 # Logs de sesiones

~/.config/claude-code/   # Alternativa (Linux)
```

14.5.2 El archivo config.json

Ubicación: `~/.claude/config.json`

Comando para crear:

```
mkdir -p ~/.claude
touch ~/.claude/config.json
```

14.5.3 Explicación de Cada Setting

14.5.3.1 1. Modelo

```
{
  "model": "claude-opus-4-6-20250501"
}
```

Modelo	Contexto	Mejor Para
`claude-opus-4-6-20250501`	1M tokens	Arquitectura, debugging complejo
`claude-sonnet-4-6-20250501`	200K tokens	Desarrollo diario
`claude-haiku-4-5-20250501`	200K tokens	Tareas rápidas

14.5.3.2 2. Modo de Agente

```
{
  "agent": {
    "mode": "all"
  }
}
```

Modo	Significado
`"all"`	Puede ejecutar cualquier acción
`"edit"`	Solo edición de archivos
`"read"`	Solo lectura

14.5.3.3 3. Configuración de Contexto

```
{
  "context": {
    "maxTokens": 100000,
    "includePatterns": ["src/**/*.ts"],
    "excludePatterns": ["node_modules/**"]
  }
}
```

Setting	Qué hace
`maxTokens`	Límite de contexto
`includePatterns`	Archivos a incluir
`excludePatterns`	Archivos a excluir

14.5.3.4 4. Permisos de Bash

```
{
  "permissions": {
    "allow": ["git *", "npm *", "mkdir *"],
    "deny": ["rm -rf *", "sudo *"],
    "ask": ["git push", "git commit"]
  }
}
```

14.5.3.5 5. AGENTS.md

```
{
  "agentsMd": "~/.claude/AGENTS.md"
}
```

Claude Code busca automáticamente `AGENTS.md` en: 1. `./AGENTS.md` (proyecto actual) 2. `~/.claude/AGENTS.md` (global)

14.6 AGENTS.md para Claude Code

14.6.1 Template Específico para Claude Code

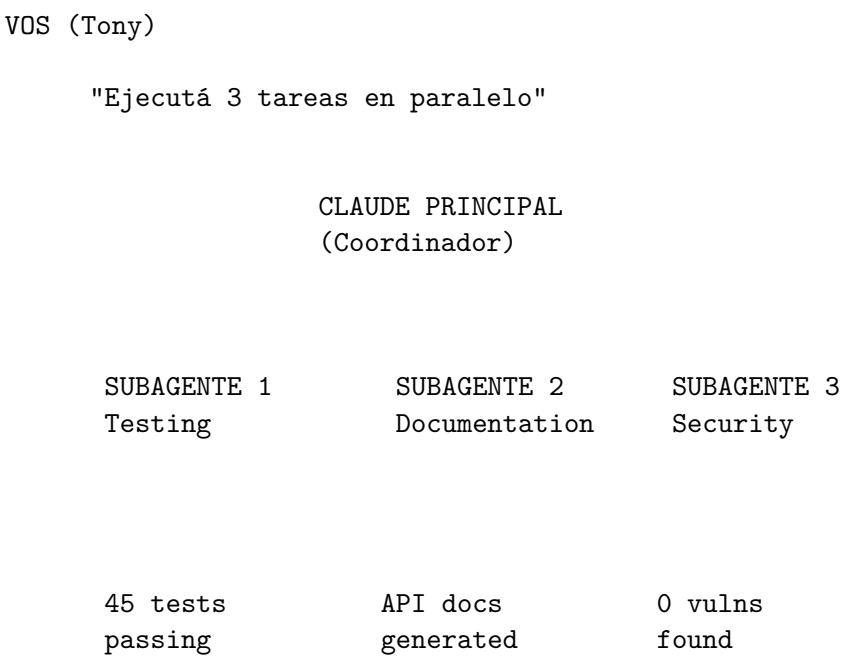
```
# AGENTS.md - [Tu Nombre]

## Mi Perfil
[Tu descripción como desarrollador]

## Reglas de Seguridad
```

```
1. Nunca ejecutar comandos destructivos
2. Siempre verificar antes de git push
3. No revelar información sensible

## Estilo de Código
- Lenguaje: [tu lenguaje]
- Nomenclatura: [tu convención]
- Tests: [tu approach]

## Preferencias de Claude
- Explicaciones: [detalladas, concisas]
- Código: [comentado, minimalista]
- Errores: [cómo manejarlos]
```

14.7 Subagentes: Tu Equipo de JARVIS

14.7.1 ¿Qué son los Subagentes?

Los subagentes son instancias de Claude que trabajan en paralelo para vos.



14.7.2 Cómo Lanzar Subagentes

En tu conversación con Claude:

```
> Necesito que hagas 3 cosas:
> 1. Escribas tests para auth.py
> 2. Documente la API en README.md
> 3. Revises seguridad del login
>
> Podés usar subagentes para las 3 en paralelo.
```

Claudewill:

```
Entendido. Lanzando 3 subagentes en paralelo...
[Subagente 1] Escribiendo tests...
[Subagente 2] Generando documentación...
[Subagente 3] Escaneando vulnerabilidades...
```

14.8 Integración con MCP

14.8.1 Configuración MCP

Claude Code soporta MCP nativamente. Configurá en `~/.claude/config.json`:

```
{
  "mcpServers": {
    "context7": {
      "command": "npx",
      "args": ["-y", "@context7/mcp-server"]
    },
    "github": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-github"]
    },
    "filesystem": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-filesystem", "~/projects"]
    }
  }
}
```

14.9 Lab: Tu Primera Sesión con Claude Code

14.9.1 Objetivo

Ejecutar Claude Code con subagentes.

14.9.2 Timeline

Paso	Descripción	Tiempo
1	Instalar Claude Code	5 min
2	Crear config.json	5 min
3	Crear AGENTS.md	10 min
4	Sesión con subagentes	15 min

14.9.3 Escenario

Tony tiene 3 misiones simultáneas. Necesita un equipo de JARSIS trabajando en paralelo.

14.9.4 Paso 1: Crear config.json

Archivo: `~/.claude/config.json`

```
{
  "model": "claude-sonnet-4-6-20250501",
  "agent": {
    "mode": "all"
  },
  "context": {
    "maxTokens": 100000,
    "includePatterns": ["**/*.ts", "**/*.js"],
    "excludePatterns": ["node_modules/**", ".git/**"]
  },
  "permissions": {
    "allow": ["git status", "git diff", "npm *", "mkdir *"],
    "deny": ["rm -rf *", "sudo *"],
    "ask": ["git push", "git commit"]
  }
}
```

14.9.5 Paso 2: Crear AGENTS.md

```
mkdir -p ~/.claude
touch ~/.claude/AGENTS.md
```

```
# AGENTS.md

## Quién Soy
[Tu nombre], desarrollador [tu rol].

## Reglas
1. Tests primero, código después
2. Explicar antes de implementar
3. Verificar antes de commit

## Preferencias
- Lenguaje: TypeScript
- Estilo: código limpio, comentarios en español
```

14.9.6 Paso 3: Tu Primera Sesión

Comando:

```
claude
```

Tu mensaje:

```
> Soy nuevo con Claude Code. Leé mi AGENTS.md
> y confirmame qué reglas encontraste.
> Después explicame cómo funcionan los subagentes.
```

14.10 Comparación: Claude Code vs OpenCode

Aspecto	Claude Code	OpenCode
Modelos	Solo Claude	75+
Precio	$20-200/mes	Gratis (BYO)
Contexto	1M tokens (Opus)	Según modelo
Subagentes	Nativos	No

Aspecto	Claude Code	OpenCode
Debugging	Excelente	Bueno
MCP	Profunda	
AGENTS.md		
Open Source		MIT

14.10.1 Cuándo usar cada uno:

Situación	Herramienta
Budget limitado	OpenCode
Proyecto complejo	Claude Code
Necesitás subagentes	Claude Code
Múltiples modelos	OpenCode
Debugging profundo	Claude Code
Privacidad total	OpenCode

14.11 Checklist Final

Habilidad
Claude Code instalado
config.json creado
AGENTS.md configurado
Sesión iniciada
Subagentes entendidos

14.12 Logro

"Commander" desbloqueado:

- +100 XP
- Capacidad de orquestar subagentes
- Debugging avanzado disponible

14.13 Siguiente

¿Listo para la siguiente herramienta?

→ Unidad 4: Gemini CLI — Tu JARVIS Gratis

15 Unidad 4: Gemini CLI — Tu JARVIS Gratis

16 Unidad 4: Gemini CLI — Tu JARVIS Gratis

16.1 "Google te dá las herramientas. Vos aprendés a usarlas."

16.2 Objetivo

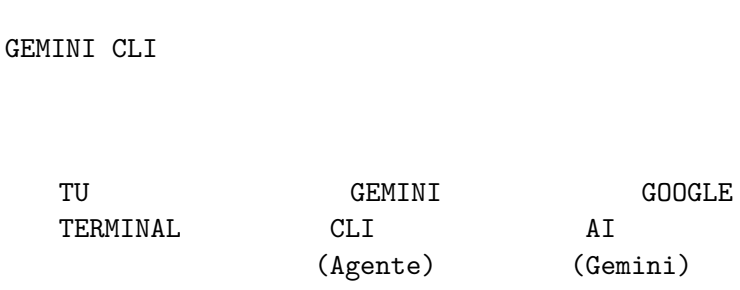
Al terminar esta unidad podrás:

- **Instalar** Gemini CLI en tu sistema
- **Usar** Gemini sin costo alguno
- **Configurar** el contexto de tu proyecto
- **Integrar** con Google Cloud
- **Comparar** con OpenCode y Claude Code

16.3 ¿Qué es Gemini CLI?

16.3.1 En una frase

Gemini CLI es el agente de terminal oficial de Google. Acceso gratis a Gemini 2.0 con 1M tokens de contexto.



```
1M tokens de contexto
Integración Google Cloud
```

16.3.2 ¿Por qué Gemini CLI?

Característica	Gemini CLI	OpenCode	Claude Code
Precio	**GRATIS**	Gratis (BYO)	$20-200/mes
Contexto	1M tokens	Según modelo	1M (Opus)
Modelo	Gemini 2.0	75+ modelos	Solo Claude
Configuración	Mínima	Media	Media
Google Cloud			

Cuándo usar Gemini CLI: - Empezar a aprender sin costo - Prototipos rápidos - Proyectos que no requieren API keys - Integración con Google Cloud

---

16.4 Instalación de Gemini CLI

16.4.1 Timeline

Paso	Descripción	Tiempo
1	Verificar prerrequisitos	2 min
2	Instalar Gemini CLI	5 min
3	Autenticar con Google	5 min
4	Primera ejecución	2 min

16.4.2 Paso 1: Verificar Prerrequisitos

Comando:

```
node --version
```

Resultado esperado:

```
v18.x.x o superior
```

16.4.3 Paso 2: Instalar Gemini CLI

16.4.3.1 Opción A: Con npm (Recomendado)

Hint: Paquete oficial de Google

Comando parcial: `npm install -g @`

Completá el comando:

```
[ESCRIBE AQUÍ]
```

Resultado esperado:

```
added X packages in Ys
```

16.4.3.2 Opción B: Con curl

Comando parcial: `curl -fsSL https://google.com/ |`

Completá:

```
[ESCRIBE AQUÍ]
```

16.4.4 Paso 3: Autenticar

Comando:

```
gemini
```

Resultado esperado:

```
  Please authenticate with Google
Visit: https://auth.google.com/
```

1. Abrí el link en tu navegador
2. Iniciá sesión con tu cuenta Google
3. Autorizá el acceso
4. Volvé a la terminal

¿No querés autenticarte?

```
gemini --api-key $GOOGLE_API_KEY
```

16.4.5 Paso 4: Verificar Instalación

Comando:

```
gemini --version
```

Resultado esperado:

```
X.X.X
```

16.5 Configuración de Gemini CLI

16.5.1 Estructura de Archivos

```
~/.gemini/
  config.yaml           # Configuración
  sessions/             # Sesiones guardadas
  logs/                 # Logs
```

16.5.2 El archivo config.yaml

Ubicación: `~/.gemini/config.yaml`

Comando para crear:

```
mkdir -p ~/.gemini
touch ~/.gemini/config.yaml
```

16.5.3 Explicación de Cada Setting

16.5.3.1 1. Modelo

```yaml
model: "gemini-2.0-flash"
```

Modelo	Contexto	Costo	Mejor Para
`gemini-2.0-flash`	1M tokens	**GRATIS**	Tareas rápidas
`gemini-2.0-pro`	1M tokens	Pago	Código complejo
`gemini-1.5-pro`	1M tokens	Pago	Balanceado

16.5.3.2 2. Parámetros de Generación

```yaml
generation:
  temperature: 0.7
  max_tokens: 100000
  top_p: 0.9
  top_k: 40
```

Parámetro	Qué hace	Rango
`temperature`	Creatividad (0.7 es balanceado)	0.0 - 1.0
`max_tokens`	Longitud máxima	Según modelo
`top_p`	Diversidad de respuestas	0.0 - 1.0
`top_k`	Tokens a considerar	1 - 100

16.5.3.3 3. Contexto del Proyecto

```yaml
context:
  project_root: "~/mi-proyecto"
  include:
    - "src/**"
    - "tests/**"
  exclude:
    - "node_modules/**"
    - ".git/**"
    - "*.log"
```

16.5.3.4 4. Sistema de Instrucciones

```
system:
  instructions: |
    Eres un asistente de programación experto.
    Siempre incluye comentarios en español.
    Prioriza código legible sobre código clever.
```

16.6 Tu Primer AGENTS.md para Gemini CLI

```
# AGENTS.md - [Tu Nombre]

## Quién Soy
[Tu descripción]

## Reglas
1. Código limpio y documentado
2. Tests para cada función
3. Explicar decisiones técnicas

## Preferencias
- Lenguaje principal: [tu lenguaje]
- Comentarios en: [español/inglés]
- Estilo: [tu estilo]
```

16.7 Integración con Google Cloud

16.7.1 Configuración

```
gcloud:
  project: "mi-proyecto-123"
  region: "us-central1"
```

16.7.2 Comandos Útiles

```
# Listar proyectos
gemini gcloud projects list

# Ver recursos
gemini gcloud compute instances list

# Deploy a Cloud Run
gemini gcloud run deploy mi-servicio
```

16.8 Lab: Tu Primera Sesión con Gemini CLI

16.8.1 Objetivo

Usar Gemini CLI gratis para una tarea real.

16.8.2 Timeline

Paso	Descripción	Tiempo
1	Instalar Gemini CLI	5 min
2	Configurar	5 min
3	Sesión práctica	15 min
4	Reflexión	5 min

16.8.3 Escenario

Tony tiene budget limitado pero necesita un JARVIS funcional. Gemini CLI es la solución.

16.8.4 Paso 1: Crear config.yaml

Archivo: `~/.gemini/config.yaml`

```yaml
model: "gemini-2.0-flash"
generation:
  temperature: 0.7
  max_tokens: 100000
  top_p: 0.9
context:
  exclude:
    - "node_modules/**"
    - ".git/**"
system:
  instructions: |
    Soy desarrollador [tu rol].
    Prefiero código limpio con comentarios.
```

16.8.5 Paso 2: Tu Primera Sesión

Comando:

```
gemini
```

Tu mensaje:

```
> Hola. Soy [tu nombre] y estoy aprendiendo a usar IA.
> Necesito que me expliques qué puede hacer Gemini CLI
> y cómo configurarlo para proyectos de código.
```

16.9 Comparación: Las 3 Herramientas Principales

Aspecto	OpenCode	Claude Code	Gemini CLI
Precio	Gratis (BYO)	$20-200/mes	**GRATIS**
Modelos	75+	Solo Claude	Solo Gemini
Contexto	Según modelo	1M (Opus)	1M
Subagentes			
Open Source			
Configuración	Alta	Media	Baja
MCP			
AGENTS.md			

16.9.1 Recomendación por Nivel:

Nivel	Herramienta Principal	Alternativa
1 (Principiante)	Gemini CLI	OpenCode
2 (Intermedio)	OpenCode	Claude Code
3 (Avanzado)	Claude Code + OpenCode	KiloCode
4 (Experto)	Stack completo	-

16.10 Checklist Final

Habilidad
Gemini CLI instalado
Autenticado con Google
config.yaml creado
Sesión iniciada
Diferencias entendidas

16.11 Logro

"Free Agent" desbloqueado:

- +50 XP
- Capacidad de trabajar sin costo
- Base para herramientas más avanzadas

16.12 Siguiente

¿Querés explorar herramientas especializadas?

→ Unidad 5: KiloCode — Orquestación Multi-Agente

17 Unidad 5: KiloCode — Orquestación Multi-Agente

18 Unidad 5: KiloCode — Orquestación Multi-Agente

18.1 "No un agente. Un equipo de agentes."

18.2 Objetivo

Al terminar esta unidad podrás:

- **Instalar** KiloCode (VS Code + CLI)
- **Configurar** Orchestrator Mode
- **Crear** múltiples agentes especializados
- **Orquestar** tareas en paralelo
- **Usar** Memory Bank para persistencia

18.3 ¿Qué es KiloCode?

18.3.1 En una frase

KiloCode es una extensión de VS Code con Orchestrator Mode. Un agente principal coordina múltiples subagentes especializados.

```
KILOCODE - ORCHESTRATOR MODE


                        ORCHESTRATOR
                        (Agente Principal)

   "Coordiná 3 equipos en paralelo"
```

```
AGENTE              AGENTE              AGENTE
TESTING             SECURITY            DOCS

Modelo: Haiku       Modelo: Sonnet      Modelo: Flash
Enfoque: Tests      Enfoque: Vulns      Enfoque: API
```

18.3.2 ¿Por qué KiloCode?

Característica	KiloCode	OpenCode	Claude Code
Orquestación	Orchestrator Mode		Limitada
Modelos	500+	75+	Solo Claude
Memory Bank			
IDE Integration	VS Code + JetBrains		VS Code
Precio	Gratis + $19/mo	Gratis (BYO)	$20-200/mes

18.4 Instalación de KiloCode

18.4.1 Timeline

Paso	Descripción	Tiempo
1	Instalar VS Code	5 min
2	Instalar extensión	5 min
3	Configurar API Keys	5 min
4	Activar Orchestrator	5 min

18.4.2 Paso 1: Instalar VS Code

Si no tenés VS Code:

1. Ve a code.visualstudio.com
2. Descargá e instalá
3. Abrí VS Code

18.4.3 Paso 2: Instalar KiloCode

18.4.3.1 Opción A: Desde VS Code Marketplace

1. Abrí VS Code
2. Buscá `KiloCode` o `kilo-code`
3. Click en **Install**

18.4.3.2 Opción B: Desde Command Palette

1. `Ctrl/Cmd + Shift + P`
2. Buscá `Extensions: Install Extensions`
3. Buscá `KiloCode`
4. Install

18.4.3.3 Opción C: Con código

Comando:

```
code --install-extension kilo-code.kilo-code
```

18.4.4 Paso 3: Configurar API Key

1. Abrí Settings (`Ctrl/Cmd + ,`)
2. Buscá `KiloCode`
3. Agregá tu API key en `KiloCode: Api Key`

O desde settings.json:

```
{
  "kiloCode.apiKey": "${KILO_CODE_API_KEY}"
}
```

18.4.5 Paso 4: Instalar CLI (Opcional)

```
npm install -g @kilo-code/cli
```

18.5 Configuración de KiloCode

18.5.1 Estructura de Archivos

```
.vscode/
  settings.json           # Config VS Code
.kilo/
  config.json            # Config KiloCode
  agents.json           # Definición de agentes
  memory/                # Memory Bank
```

18.5.2 El archivo agents.json

Ubicación: `.kilo/agents.json`

```json
{
  "orchestrator": {
    "model": "anthropic/claude-sonnet-4",
    "maxTokens": 100000
  },
  "agents": {
    "testing": {
      "model": "anthropic/claude-haiku-4",
      "focus": ["tests", "coverage", "pytest"],
      "description": "Especialista en testing"
    },
    "security": {
      "model": "openai/gpt-4o",
      "focus": ["security", "vulnerabilities", "owasp"],
      "description": "Especialista en seguridad"
    },
    "docs": {
      "model": "google/gemini-2.0-flash",
      "focus": ["documentation", "readme", "api-docs"],
      "description": "Especialista en documentación"
    }
  }
}
```

18.5.3 Orchestrator Mode

18.5.3.1 Activar desde VS Code

1. Abrí Command Palette (`Ctrl/Cmd + Shift + P`)
2. Buscá `KiloCode: Toggle Orchestrator Mode`
3. Activá

18.5.3.2 Activar desde CLI

```
kilo-code --mode orchestrator
```

18.5.4 Memory Bank

KiloCode incluye memoria persistente entre sesiones.

```
{
  "memoryBank": {
    "enabled": true,
    "path": ".kilo/memory",
    "autoSave": true,
    "contextWindow": 50000
  }
}
```

18.6 Cómo Usar Orchestrator Mode

18.6.1 Tu Prompt al Orchestrator

```
> Necesito que hagas lo siguiente en paralelo:
>
> 1. AGENTE TESTING: Escribí tests para auth.py (pytest)
> 2. AGENTE SECURITY: Revisá vulnerabilidades en login
> 3. AGENTE DOCS: Documentá la API REST
>
> Cuando terminen, consolida los resultados y dame un resumen.
```

18.6.2 El Orchestrator:

```
 Lanzando 3 agentes en paralelo...

[TESTING] Analizando auth.py...
[SECURITY] Escaneando login.py...
[DOCS] Generando API docs...

 AGENTE TESTING: 45 tests generados
 AGENTE SECURITY: 1 warning encontrado (XSS potential)
 AGENTE DOCS: README.md actualizado

 RESUMEN CONSOLIDADO:
- Tests: 100% coverage en auth
- Security: 1 issue menor, fácil de fix
- Docs: API documentada con OpenAPI spec
```

18.7 Lab: Tu Primera Orquestación

18.7.1 Objetivo

Configurar KiloCode con 3 agentes especializados.

18.7.2 Timeline

Paso	Descripción	Tiempo
1	Instalar KiloCode	10 min
2	Crear agents.json	10 min
3	Activar Orchestrator	5 min
4	Primera tarea orquestada	15 min

18.7.3 Escenario

Tony tiene un equipo de JARSIS. Cada uno especializado en una tarea.

18.7.4 Paso 1: Crear estructura

```
mkdir -p .kilo
touch .kilo/config.json
touch .kilo/agents.json
```

18.7.5 Paso 2: Configurar agentes

Archivo: `.kilo/agents.json`

```
{
  "orchestrator": {
    "model": "anthropic/claude-sonnet-4-6-20250501"
  },
  "agents": {
    "tester": {
      "model": "anthropic/claude-haiku-4-5-20250501",
      "focus": ["testing", "pytest", "coverage"],
      "description": "Especialista en testing"
    },
    "reviewer": {
      "model": "anthropic/claude-sonnet-4-6-20250501",
      "focus": ["code review", "best practices"],
      "description": "Especialista en calidad"
    },
    "documenter": {
      "model": "google/gemini-2.0-flash",
      "focus": ["documentation", "readme"],
      "description": "Especialista en docs"
    }
  }
}
```

18.7.6 Paso 3: Tu primera tarea orquestada

En KiloCode:

```
> Configuré 3 agentes:
> - tester: especializado en tests
> - reviewer: especializado en code review
```

```
> - documenter: especializado en docs
>
> Confirmame que entendiste la configuración.
```

18.8 Comparación: KiloCode vs Otras Herramientas

Aspecto	KiloCode	OpenCode	Claude Code
Orquestación	Completa		Básica
Modelos	500+	75+	Solo Claude
Memory Bank			
Precio	$19/mo	Gratis	$20-200/mes
Open Source	Parcial		

18.8.1 Cuándo usar KiloCode:

Situación	Herramienta
Necesitás múltiples agentes	KiloCode
Budget limitado	OpenCode
Proyecto crítico con debugging	Claude Code
Empezar gratis	Gemini CLI

18.9 Checklist Final

Habilidad
KiloCode instalado
Orchestrator Mode activado
agents.json configurado
Primera orquestación ejecutada

18.10 Logro

"Conductor" desbloqueado:

- +150 XP
- Capacidad de orquestar múltiples agentes
- Gestión de equipo de IA

18.11 Siguiente

¿Listo para las herramientas especializadas?

→ Unidad 6: Kiro — Spec-Driven Development

→ Unidad 7: Amp — Asistente Ligero

19 Unidad 6: Kiro — Spec-Driven Development

20 Unidad 6: Kiro — Spec-Driven Development

20.1 "No programs. Se especifica. Kiro implementa."

20.2 Objetivo

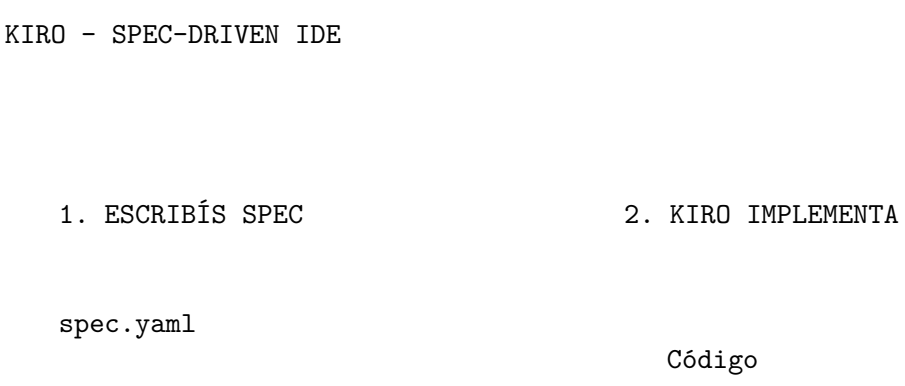
Al terminar esta unidad podrás:

- **Instalar** Kiro en tu sistema
- **Escribir** especificaciones estructuradas
- **Usar** Hooks para automatización
- **Implementar** usando SDD (Spec-Driven Development)
- **Comparar** con el workflow de OpenCode

20.3 ¿Qué es Kiro?

20.3.1 En una frase

Kiro es el IDE de Amazon que revoluciona el desarrollo con Spec-Driven Development. Diseñas especificaciones, Kiro implementa.



```
name: auth                                    Generated
requirements:                                 100%
  - id: FR-001                               Matched
    title: Login
    desc: User auth...


                    Hooks de Automatización
                    Steerling (UI con IA)
```

20.3.2 ¿Por qué Kiro?

Característica	Kiro	OpenCode	Claude Code
Paradigma	SDD nativo	Agéntico	Agéntico
Specs	Integradas		
Hooks	Automatización		
UI Design	Steerling		
AWS Integration	Profunda		
Precio	$20-200/mes	Gratis	$20-200/mes

20.4 Instalación de Kiro

20.4.1 Timeline

Paso	Descripción	Tiempo
1	Descargar Kiro	5 min
2	Instalar	10 min
3	Configurar cuenta	5 min
4	Primer proyecto	10 min

20.4.2 Paso 1: Descargar Kiro

1. Ve a kiro.aws
2. Click en **Download**
3. Seleccioná tu SO (Windows/Mac/Linux)

4. Descargá el installer

20.4.3 Paso 2: Instalar

20.4.3.1 macOS

1. Abrí el `.dmg`
2. Arrastrá Kiro a Applications
3. Abrí Kiro desde Applications

20.4.3.2 Linux

```
# Con snap
sudo snap install kiro --classic

# O con .deb
sudo dpkg -i kiro.deb
```

20.4.3.3 Windows

1. Ejecutá el `.exe`
2. Follow the wizard
3. Abrí Kiro

20.4.4 Paso 3: Configurar Cuenta

1. Abrí Kiro
2. Creá cuenta o iniciá sesión
3. Conectá tu API key de AWS (opcional)

20.5 Specs: El Core de Kiro

20.5.1 ¿Qué es una Spec?

Una Spec es una especificación estructurada de lo que querés construir.

```
proyecto/
   .kiro/
      specs/
          auth/
              spec.yaml
              requirements.md
          api/
               spec.yaml
               requirements.md
```

20.5.2 Estructura de spec.yaml

```yaml
name: "Sistema de Autenticación"
description: "API REST para autenticación de usuarios"

requirements:
  - id: FR-001
    title: "Login de usuario"
    description: |
      El sistema debe permitir a los usuarios iniciar sesión
      con email y contraseña.
    acceptance_criteria:
      - "Retorna JWT válido"
      - "Maneja credenciales inválidas"
      - "Bloquea después de 5 intentos fallidos"
    priority: high
    tags: ["auth", "security"]

  - id: FR-002
    title: "Registro de usuario"
    description: |
      El sistema debe permitir registrar nuevos usuarios.
    acceptance_criteria:
      - "Valida email único"
      - "Hashea contraseñas"
      - "Envía email de verificación"
    priority: high
    tags: ["auth", "registration"]
```

```yaml
hooks:
  - trigger: "on_save"
    action: "run_tests"
  - trigger: "on_commit"
    action: "validate_specs"

technology:
  language: "python"
  framework: "fastapi"
  database: "postgresql"
```

20.5.3 Commands en Specs

Los **commands** son acciones que Kiro ejecuta automáticamente.

```yaml
commands:
  - name: "generate_models"
    description: "Genera modelos SQLAlchemy desde specs"
    trigger: "on_spec_change"

  - name: "generate_tests"
    description: "Genera tests desde acceptance criteria"
    trigger: "on_save"

  - name: "validate_coverage"
    description: "Verifica que specs tengan tests"
    trigger: "on_commit"
```

20.6 Hooks: Automatización

20.6.1 ¿Qué son los Hooks?

Los Hooks son disparadores que ejecutan acciones cuando ocurren eventos.

```
HOOKS EN KIRO

EVENTO                          ACCIÓN
```

```
on_save                run_tests
on_save                lint_code
on_commit              validate_specs
on_push                deploy_staging
on_spec_change           generate_code
```

20.6.2 Hooks Comunes

```yaml
hooks:
  - trigger: "on_save"
    actions:
      - "run_linter"
      - "format_code"

  - trigger: "on_commit"
    actions:
      - "validate_specs"
      - "check_coverage"
      - "run_tests"

  - trigger: "on_spec_change"
    actions:
      - "regenerate_models"
      - "update_tests"
```

20.7 Steerling: UI con IA

20.7.1 ¿Qué es Steerling?

Steerling es la herramienta de diseño UI de Kiro. Describís la UI, Kiro la genera.

```
> Quiero un login form con:
> - Campo email
> - Campo contraseña
> - Botón de login
```

```
> - Link a registro
> - Estilo minimalista, azul y blanco
```

Steerling genera: - Componente React/Vue/HTML - Estilos CSS/Tailwind - Validación de inputs - Estados (loading, error, success)

20.8 Lab: Tu Primer Spec

20.8.1 Objetivo

Crear una spec completa con requirements y hooks.

20.8.2 Timeline

Paso	Descripción	Tiempo
1	Instalar Kiro	10 min
2	Crear estructura	5 min
3	Escribir spec	15 min
4	Implementar	15 min

20.8.3 Escenario

Tony quiere un sistema de login. Pero esta vez, primero SPECIFICA, después implementa.

20.8.4 Paso 1: Crear proyecto

```bash
mkdir mi-proyecto-kiro
cd mi-proyecto-kiro
mkdir -p .kiro/specs/auth
```

20.8.5 Paso 2: Crear spec.yaml

Archivo: `.kiro/specs/auth/spec.yaml`

```yaml
name: "Sistema de Login"
description: "Autenticación simple con JWT"

requirements:
  - id: AUTH-001
    title: "Login con email y contraseña"
    description: |
      Los usuarios deben poder iniciar sesión
      con email y contraseña válidos.
    acceptance_criteria:
      - "Retorna JWT de 1 hora"
      - "Maneja errores 401 para credenciales inválidas"
      - "Loguea intentos fallidos"
    priority: critical

  - id: AUTH-002
    title: "Protección de endpoints"
    description: |
      Los endpoints protegidos deben verificar el JWT.
    acceptance_criteria:
      - "Rechaza requests sin Authorization header"
      - "Rechaza tokens expirados"
      - "Retorna 401 para tokens inválidos"
    priority: critical

hooks:
  - trigger: "on_save"
    actions: ["run_tests", "lint_code"]
```

20.8.6 Paso 3: Generar código

En Kiro, ejecutá:

```
> Generá el código para la spec auth/
```

20.9 Comparación: Kiro vs Other Workflows

Aspecto	Kiro (SDD)	OpenCode	Claude Code
Metodología	Specs → Code	Contexto → Code	Contexto → Code
Specs	Nativas		
Traceabilidad	100%		
Automatización	Hooks		Limitada
Curva aprendizaje	Alta	Baja	Media

20.10 Checklist Final

Habilidad
Kiro instalado
Primera spec creada
Hooks configurados
Spec→Code workflow entendido

20.11 Logro

"Architect" desbloqueado:

- +150 XP
- Capacidad de especificar antes de implementar
- Dominio de SDD

20.12 Siguiente

¿Listo para algo más ligero?

→ Unidad 7: Amp — Asistente Ligero

O volvé al overview:

→ Resumen: Stack Completo de Herramientas

21 Unidad 7: Amp — Asistente Ligero

22 Unidad 7: Amp — Asistente Ligero

22.1 "Cuando necesitas algo rápido, no necesitas algo pesado."

22.2 Objetivo

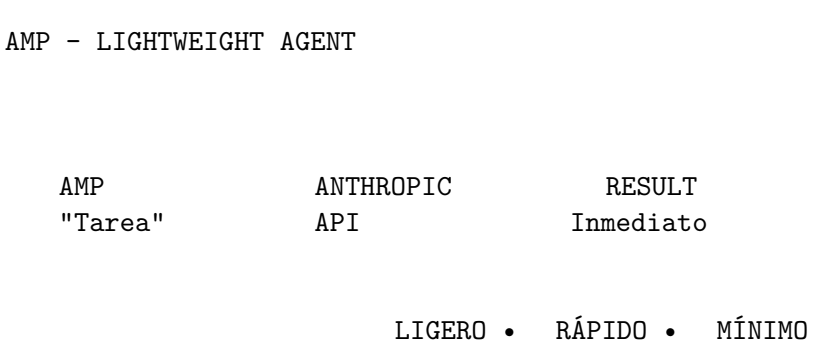
Al terminar esta unidad podrás:

- **Instalar** Amp en tu sistema
- **Usar** para tareas rápidas sin configuración
- **Integrar** con tu flujo de trabajo existente
- **Comparar** con herramientas más complejas

22.3 ¿Qué es Amp?

22.3.1 En una frase

Amp es un agente de terminal ultra-ligero de Anthropic. Tareas rápidas sin la complejidad de Claude Code.



22.3.2 ¿Por qué Amp?

Característica	Amp	Claude Code	OpenCode
Velocidad			
Configuración	0	Media	Alta
Uso de memoria	Mínimo	Alto	Medio
Tareas complejas			
Tareas rápidas			
Precio	Gratis	$20-200/mes	Gratis

Cuándo usar Amp: - Fix rápido de un bug - Explicación de código - Traducción de código - Tareas de 5 minutos

---

22.4 Instalación de Amp

22.4.1 Timeline

Paso	Descripción	Tiempo
1	Instalar Amp	2 min
2	Configurar API Key	2 min
3	Primera tarea	3 min

22.4.2 Paso 1: Instalar Amp

22.4.2.1 Con npm

```
npm install -g @anthropic-ai/amp
```

22.4.3 Paso 2: Configurar API Key

```
export ANTHROPIC_API_KEY="sk-ant-tu-key"
```

22.4.4 Paso 3: Verificar

```
amp --version
```

22.5 Uso de Amp

22.5.1 Tareas Rápidas

```
# Explicar código
amp "Explicame este código: [pega código]"

# Traducir lenguaje
amp "Traducí esta función de Python a TypeScript: [código]"

# Fix rápido
amp "Hay un bug en esta función: [código]. Encontralo."

# Generar test
amp "Generá un test pytest para: [función]"
```

22.5.2 Con Archivo

```
# Leer archivo y actuar
amp "Revisá el archivo auth.py y sugerí mejoras"

# Con pipe
cat auth.py | amp "Sugerí refactors para este código"
```

22.5.3 Con AGENTS.md

```
amp --agents ~/AGENTS.md "Revisá mi último commit"
```

22.6 AGENTS.md para Amp

```
# AGENTS.md - [Tu Nombre]

## Reglas Rápidas
1. Código limpio
2. Comentarios explicativos
3. Tests para funciones importantes
```

22.7 Lab: Tareas Rápidas con Amp

22.7.1 Objetivo

Resolver 5 tareas rápidas con Amp.

22.7.2 Timeline

Paso	Descripción	Tiempo
1	Instalar Amp	2 min
2	Tarea 1: Explicar	3 min
3	Tarea 2: Traducir	5 min
4	Tarea 3: Test	5 min

22.7.3 Escenario

Tony necesita respuestas rápidas. No tiene tiempo para sesiones largas.

22.7.4 Tarea 1: Explicar Código

````
amp "Explicame esta función en 3 líneas:
```python
def foo(x):
    return x * 2
```"
````

22.7.5 Tarea 2: Traducir

```
amp "Traducí a JavaScript:
```python
def greet(name: str) -> str:
    return f'Hola, {name}'
```"
```

22.7.6 Tarea 3: Generar Test

```
amp "Generá un test pytest para:
```python
def add(a, b):
    return a + b
```"
```

22.8 Comparación Final: Todas las Herramientas

Herramienta	Mejor Para	Precio	Velocidad	Complejidad
OpenCode	Flexibilidad total	Gratis		Media
Claude Code	Debugging profundo	$20-200/mes		Media
Gemini CLI	Empezar gratis	Gratis		Baja
KiloCode	Multi-agente	$19/mes		Alta
Kiro	Spec-Driven	$20-200/mes		Alta
Amp	Tareas rápidas	Gratis		Mínima

22.9 Checklist Final

Habilidad
Amp instalado
3 tareas ejecutadas
Diferencia con otras entendida

22.10 Logro

“Sprinter” desbloqueado:

- +50 XP
- Capacidad de resolver rápido
- Herramienta para el día a día

22.11 Resumen

Todas las unidades creadas:

Unidad	Herramienta	Status
2	OpenCode	Completa
3	Claude Code	Completa
4	Gemini CLI	Completa
5	KiloCode	Completa
6	Kiro	Completa
7	Amp	Completa

¿Querés que cree el documento de resumen comparativo?

Part II

Nivel 1: El Demo en la Cueva

23 Nivel 1: La Cueva

24 Nivel 1: La Cueva

24.1 "Tony no tenía nada. Solo determinación."

24.2 Objetivo del Nivel

Al terminar este nivel podrás:

- **Conversar** con una IA de forma efectiva
- **Dirigir** sin micromanage — dar dirección, no comandos
- **Iterar** hasta obtener lo que necesitas
- **Verificar** que el resultado sea lo que pediste

Tu rol: Tony Stark en la cueva. Tenés la visión, JARVIS tiene las manos.

24.3 Herramienta: Gemini CLI

24.3.1 ¿Por qué Gemini CLI?

Característica	Gemini CLI
Costo	**Gratis**
Tokens	1M (contexto enorme)
Configuración	**Cero** — listo en 2 minutos
Ideal para	Primeros pasos, aprender conversación

Nota: Cualquier IA sirve para este nivel. Usamos Gemini CLI porque es gratis y no requiere configuración. Los conceptos aplican a Claude, ChatGPT, o cualquier otra.

24.4 La Escena

Iron Man (2008) — Tony Stark está en una cueva en Afganistán. Herido, cautivo, sin recursos.

Lo único que tiene: un de energía improvisado y la capacidad de **pedir ayuda**.

```
TONY: "Necesito que construyas algo que me mantenga vivo."
YINSEN: "¿Algo como qué?"
TONY: "Algo que... convenza a mi cuerpo de que no está muriendo."
```

Tony no construye solo. Pide. Pregunta. Dirige. Yinsen ejecuta.

Vos sos Tony. Gemini CLI es tu Yinsen/JARVIS.

24.5 Tu Rol: Tony en la Cueva

VOS (Tony)	JARVIS (IA)
• Tenés la VISIÓN	• Lee tu mensaje
• Definís el QUÉ y POR QUÉ	• Ejecuta comandos
• Apruebas o RECHAZAS	• Sugiere alternativas
• Decidís la DIRECCIÓN	• Pregunta si no entiende
VOS NUNCA CODIFICA	JARVIS NUNCA DECIDE SOLO

Regla de oro: Si te sorprendés de lo que hizo JARVIS, preguntá POR QUÉ.

24.6 Concepto Clave: La Conversación

24.6.1 ANTES (Enfoque WRONG)

```
VOS: "Escribíme un programa que haga X"
JARVIS: [escribe código completo]
VOS: [copia y pega]
VOS: "No funciona"
```

```
JARVIS: [escribe más código]
VOS: [copia y pega]
... (ciclo infinito)
```

Problema: No entendés qué hiciste. Si algo rompe, no sabés arreglarlo.

24.6.2 AHORA (Enfoque Tony Stark)

```
VOS: "JARVIS, necesito una función que haga X.
      ¿Cuál es la mejor forma de hacerlo?"
JARVIS: "Para X te recomiendo Y porque Z."
VOS: "Perfecto, hacelo. Pero agregá validación de errores."
JARVIS: "Hecho. ¿Querés que agregue tests?"
VOS: "Sí, y explicame cada test."
JARVIS: [implementa + explica]
VOS: "Bien. Guardalo."
```

Ventaja: Entendés cada paso. Podés verificar. Podés dirigir.

24.7 Lab 1: Tu Primera Conversación

24.7.1 Objetivo

Iniciar una conversación con JARVIS y verificar que responde.

24.7.2 Timeline

Paso	Descripción	Tiempo
1	Instalar Gemini CLI	2 min
2	Iniciar conversación	1 min
3	Tu primer mensaje	2 min
4	Iterar hasta entender	5 min

24.7.3 Escenario

Estás en la cueva. JARVIS está listo para ayudarte. Necesitás entender qué puede hacer por vos.

24.7.4 Paso 1: Instalar Gemini CLI

Tu tarea: Instalar la herramienta en tu computadora.

Pista: Se instala con npm, el gestor de paquetes de Node.js

Comando parcial para completar: `npm install -`

Resultado esperado:

```
added X packages in Ys
```

24.7.5 Paso 2: Iniciar conversación

Tu tarea: Lanzar JARVIS.

Pista: El comando para ejecutar es simplemente el nombre del paquete

Comando parcial: `gem`

Resultado esperado:

```
 Gemini CLI

¿Qué te gustaría hacer hoy?
>
```

24.7.6 Paso 3: Tu primer mensaje

Tu tarea: Escribir tu primer mensaje a JARVIS.

NO copies esto tal cual. Personalizalo:

```
> Hola JARVIS. Soy nuevo en esto.
> ¿Podrías explicarme qué puedes hacer y cómo podemos trabajar juntos?
```

Pregunta de verificación: ¿Qué respondió JARVIS? En tus propias palabras:

```
JARVIS puede: ________________________________________
Podemos trabajar juntos porque: ______________________
```

24.7.7 Paso 4: Iterar hasta entender

Tu tarea: Hacer al menos 3 preguntas de seguimiento.

Ejemplos de preguntas válidas:

1. "¿Cómo te doy contexto sobre mi proyecto?"
2. "¿Puedes ver archivos en mi computadora?"
3. "¿Cómo te digo que algo no es lo que quería?"

Tu conversación:

```
PREGUNTA 1: ________________________________________
RESPUESTA: _________________________________________

PREGUNTA 2: ________________________________________
RESPUESTA: _________________________________________

PREGUNTA 3: ________________________________________
RESPUESTA: _________________________________________
```

24.8 Lab 2: Pidiendo Algo Específico

24.8.1 Objetivo

Aprender a estructurar un pedido para obtener lo que necesitás.

24.8.2 Timeline

Paso	Descripción	Tiempo
1	Escribir pedido estructurado	3 min
2	Enviar a JARVIS	1 min
3	Revisar respuesta	2 min
4	Dar feedback	3 min

24.8.3 Escenario

Tony necesita algo específico: "JARVIS, necesito una fuente de energía pequeña que quepa en mi pecho."

24.8.4 Fórmula del Pedido Estructurado

```
[PEDIDO] + [CONTEXTO] + [RESTRICCIONES] + [FORMATO]
```

Parte	Qué decir	Ejemplo
PEDIDO	Qué querés	"Necesito una función que..."
CONTEXTO	Para qué	"...para validar emails de usuarios"
RESTRICCIONES	Reglas	"...debe usar regex y retornar boolean"
FORMATO	Cómo lo querés	"...con comentarios en español"

24.8.5 Tu tarea: Escribir un pedido

Completá el siguiente pedido:

```
Hola JARVIS,

PEDIDO: Necesito _____________________________________________

CONTEXTO: ____________________________________________________

RESTRICCIONES:
- ____________________________________________________________
- ____________________________________________________________

FORMATO: Quiero que _____________________________________________

¿Podrías hacerlo y explicarme cada parte?
```

Tu pedido completo:

```
[JESCRIBA EL PEDIDO COMPLETO AQUÍ]
```

24.8.6 Enviar y revisar

Tu tarea: Enviar el pedido a JARVIS.

Pregunta de verificación:

```
¿JARVIS hizo lo que pediste? SÍ / NO
Si NO, ¿qué faltó? _____________________________________________
```

24.8.7 Dar feedback

Regla: Si no es exactamente lo que querés, decilo.

Plantilla de feedback:

```
JARVIS, gracias. Pero necesito que _____________________________
Específicamente: _____________________________________________
¿Podrías ajustarlo?
```

Tu feedback (si corresponde):

```
JARVIS, gracias. Pero necesito que _____________________________
```

24.9 Lab 3: Diciendo "No Es Lo Que Quería"

24.9.1 Objetivo

Aprender a corregir sin frustración — JARVIS no se ofende.

24.9.2 Timeline

Paso	Descripción	Tiempo
1	Pedir algo	2 min
2	Identificar qué está mal	2 min
3	Escribir feedback específico	3 min
4	Verificar corrección	2 min

24.9.3 Escenario

Tony dice "No, no es eso. Necesito que haga X, no Y."

JARVIS no se enoja. Corrige. Sigue.

24.9.4 Tu tarea: Provocar un error

Paso 1: Pedí algo con intencionalidad ambigua.

```
> JARVIS, creá una función que procese datos de usuarios.
```

Paso 2: Identificá qué está MAL o qué FALTA.

Paso 3: Escribí feedback específico:

```
Tu respuesta tiene X, pero necesito Y.
Ejemplo: "Incluye manejo de errores"
         "No necesita base de datos"
         "Debe funcionar offline"
```

Tu feedback específico:

```
> JARVIS, la función que me diste tiene _____________ pero necesito _____________
> Específicamente: ____________________________________________________________
> Por qué importa: ____________________________________________________________
```

Paso 4: Verificá si JARVIS corrigió correctamente.

24.10 Verificación del Nivel

24.10.1 Checklist Final

Habilidad	Demostrable cuando...
Conversar	Pudiste hacer 3+ preguntas de seguimiento
Dirigir	JARVIS hizo algo que vos especificaste
Iterar	Diste feedback y JARVIS corrigió
Verificar	Explicás POR QUÉ JARVIS hizo cada cosa

24.10.2 Prueba de fuego

Intentá esto con JARVIS:

```
> JARVIS, necesito que me expliques qué es un "prompt estructurado"
> como si tuviera 10 años. Dame un ejemplo con algo de la vida real,
> no con código. Después preguntame si entendí antes de continuar.
```

Tu respuesta:

```
¿Entendí el concepto? SÍ / NO
Lo que entendí: _____________________________________________________________
```

24.11 Recursos

24.11.1 Lectura recomendada

- Prompt Engineering Guide — Guía completa de prompts
- Conversational AI Patterns — Cómo habla Claude

24.11.2 Tu kit de conversación

- **Banco de prompts básicos:** Ver `recursos/prompts-banco.md`
- **Fórmulas probadas:** Ver `recursos/formulas-conversacion.md`

24.12 Logro Desbloqueado

24.12.1 "Cave Dweller"

Requisitos: - [ ] Completaste 3 conversaciones con JARVIS - [ ] Diste feedback específico al menos una vez - [ ] Explicás la diferencia entre directed vs dictated - [ ] Completaste la prueba de fuego

Recompensa: - Acceso al **Nivel 2: Mark I** - Template de conversación guardado en tu kit

---

24.13 Siguiente Nivel

¿Completaste todos los labs?

→ **Nivel 2: Mark I** — Configura tu armadura

¿Necesitas más práctica?

→ **Lab Detallado: Conversación N1**

Filosofía del nivel:

"Tony Stark no sabía cómo construir el reactor arc. Sabía que lo necesitaba. JARVIS supo cómo construirlo."

Tu trabajo: Saber qué necesitás. El trabajo de JARVIS: hacerlo.

25 Lab 1: Conversación con JARVIS

26 Lab 1: Tu Primera Conversación con JARVIS

26.1 Aprende a orquestar, no a escribir código

26.2 Objetivo

Practicar la habilidad de **conversar con IA** como Tony Stark: dar dirección, no comandos.

26.3 Timeline

Paso	Descripción	Tiempo
1	Instalar Gemini CLI	5 min
2	Primera conversación	10 min
3	Pedido estructurado	15 min
4	Feedback y corrección	10 min
5	Reflexión	5 min
Total		**45 min**

26.4 Escenario

Tony está en la cueva. Necesita que Yinsen construya el reactor arc. Pero Tony no dice "haz esto". Tony dice "necesito X para lograr Y".

Tu turno: Sé Tony. Usa JARVIS.

26.5 Entregables

Al finalizar este lab tendrás:

1. Gemini CLI instalado
2. 3+ conversaciones documentadas
3. Un pedido estructurado completo
4. Un ejemplo de feedback efectivo
5. Tu reflexión personal

26.6 Paso 1: Instalar Gemini CLI

26.6.1 Tu tarea

Instalar la herramienta en tu computadora.

26.6.2 Comandos a ejecutar

Paso 1.1: Verificar que tienes Node.js

```
node --version
```

Resultado esperado:

```
v18.x.x o superior
```

Si no tienes Node.js, instálalo desde nodejs.org

Paso 1.2: Instalar Gemini CLI

Hint: npm install con el flag -g para instalación global

Comando parcial: `npm install -`

Completá el comando:

```
[ESCRIBE AQUÍ TU COMANDO COMPLETO]
```

Resultado esperado:

```
added X packages in Ys
```

Paso 1.3: Verificar instalación

Comando:

```
gemini --version
```

Resultado esperado:

```
X.X.X
```

26.7 Paso 2: Tu Primera Conversación

26.7.1 Tu tarea

Iniciar una conversación con JARVIS y hacer al menos 5 preguntas.

26.7.2 2.1 Lanzar JARVIS

Comando:

```
gemini
```

Resultado esperado:

```
  Gemini CLI
> ¿En qué puedo ayudarte?
```

26.7.3 2.2 Tu primer mensaje

NO copies esto tal cual. Personalizalo con tu nombre y situación:

```
> Hola, soy [tu nombre].
> Estoy aprendiendo a trabajar con IA por primera vez.
> ¿Podrías explicarme qué puedes hacer y cómo podemos colaborar?
```

26.7.4 2.3 Documentá tu conversación

Registrá al menos 5 intercambios:

#	Tu mensaje	Respuesta de JARVIS
1	_	_
2	_	_
3	_	_
4	_	_
5	_	_

26.8 Paso 3: Pedido Estructurado

26.8.1 Tu tarea

Escribir un pedido completo usando la fórmula:

```
[PEDIDO] + [CONTEXTO] + [RESTRICCIONES] + [FORMATO]
```

26.8.2 3.1 Completá el template

```
Hola JARVIS,

PEDIDO: Necesito que me expliques qué es [tema que no conoces bien]

CONTEXTO: Estoy estudiando [tu área/campo] y me encontré con este concepto
en [contexto específico]

RESTRICCIONES:
- Explicamelo como si tuviera [edad/experiencia específica]
- Usa analogías de [áreas que conocés bien]
- No uses jerga técnica sin explicar

FORMATO:
- Una definición clara en una oración
- 2 analogías de la vida real
- 1 ejemplo práctico
- Preguntame si entendí antes de continuar
```

26.8.3 3.2 Enviá el pedido

Copiá tu pedido completado y envialo a JARVIS.

26.8.4 3.3 Verificá el resultado

Checklist de verificación:

Criterio	¿Cumple?
Definición clara en una oración	SÍ / NO
2 analogías incluidas	SÍ / NO
1 ejemplo práctico	SÍ / NO
Pregunta de verificación	SÍ / NO

26.8.5 3.4 Respondé a JARVIS

Respondé a la pregunta de JARVIS. ¿Entendiste? Explicá el concepto en tus propias palabras:

```
En mis propias palabras, [tema] es:
_____________________________________________
_____________________________________________
```

26.9 Paso 4: Feedback y Corrección

26.9.1 Tu tarea

Practicar dar feedback específico cuando algo no es lo que querés.

26.9.2 4.1 Pedí algo nuevo

```
> JARVIS, necesito que me des 3 sugerencias para mejorar mi productividad
> como desarrollador.
```

26.9.3 4.2 Identificá qué falta

Tu análisis:

```
Las sugerencias que dio JARVIS:
1. _
2. _
3. _

¿Qué está bien? _
¿Qué falta? _
```

26.9.4 4.3 Escribí feedback específico

Plantilla (completá con tu caso real):

```
JARVIS, gracias por las sugerencias. Pero necesito que _____________

Específicamente:
- La sugerencia #___ menciona ________ pero necesito ________
- La sugerencia #___ asume ________ pero en realidad ________

¿Podrías ajustarlas?
```

Tu feedback:

```
> JARVIS, ____________________________________________________
```

26.9.5 4.4 Verificá la corrección

```
¿JARVIS ajustó correctamente? SÍ / NO

Si sí: ¿Qué cambió? _
Si no: ¿Qué siguió faltando? _
```

26.10 Paso 5: Reflexión Final

26.10.1 Tu tarea

Documentar qué aprendiste.

26.10.2 5.1 Pregunta 1: Diferencia clave

```
La diferencia entre "pedir código" y "dirigir conversación" es:

- Pedir código: ____________________________________________________
- Dirigir conversación: ____________________________________________

Ahora me siento más cómodo/a: ____________________________________
```

26.10.3 5.2 Pregunta 2: Tu mayor descubrimiento

```
Algo que me sorprendió de esta experiencia fue:

_____________________________________________________________

Porque normalmente yo esperaba que: ______________________________

Pero JARVIS puede: ________________________________________
```

26.10.4 5.3 Pregunta 3: Próximo paso

```
Para la próxima conversación con IA, voy a:

1. ___________________________________________________________
2. ___________________________________________________________
3. ___________________________________________________________
```

26.11 Entregable Final

26.11.1 Archivo a crear: `lab1-conversacion.md`

Guardá este archivo en la carpeta raíz del curso:

```
# Lab 1: Mi Primera Conversación con JARVIS

## Datos
- **Fecha:** [fecha de hoy]
- **Herramienta:** Gemini CLI
- **Duración total:** [X minutos]

## Conversaciones Registradas
[Pegá aquí tu tabla de 5+ intercambios]

## Pedido Estructurado
[Pegá aquí tu pedido completo]

## Feedback y Corrección
[Pegá aquí tu feedback y la respuesta]

## Reflexiones
### Diferencia clave
[Tu respuesta]

### Mayor descubrimiento
[Tu respuesta]

### Próximo paso
[Tu respuesta]

## Logro: "Cave Dweller"
- [x] Gemini CLI instalado
- [x] 5+ intercambios documentados
- [x] Pedido estructurado completado
- [x] Feedback dado y verificado
- [x] Reflexión completada
```

26.12 Checklist Final

Entregable	Estado
Gemini CLI instalado	

Entregable	Estado
5+ intercambios documentados	
Pedido estructurado completado	
Feedback dado y verificado	
Archivo `lab1-conversacion.md` creado	

26.13 Logro

"Cave Dweller" desbloqueado:

- Acceso a **Nivel 2: Mark I**
- Template de conversación guardado
- +100 XP

26.14 Continuar

¿Listo para el siguiente nivel?

→ Nivel 2: Mark I — Configura tu armadura

¿Necesitás más práctica?

→ Quiz Nivel 1: Conceptos de Conversación

27 Boss Fight 1: Escape de la Cueva

28 Boss Fight 1: Escape de la Cueva

28.1 Prueba Final del Nivel 1

28.2 La Situación

> *"Tengo que salir de aquí. No importa cómo."* — Tony Stark

Tony Stark está en la cueva. Herido, sin recursos, con el shrapnel cerca de su corazón. Tiene **30 minutos** antes de que los terroristas regresen. Debe construir algo que le permita **escapar**.

Tu misión: Demostrar que has aprendido lo básico del Nivel 1. Debes crear un sistema funcional **desde cero**.

28.3 Misión: Sistema de Diagnóstico Médico

Tony necesita un sistema que: 1. **Monitoree** su estado vital (simulado) 2. **Diagnostique** problemas basándose en síntomas 3. **Recomiende** acciones inmediatas 4. **Registre** todo en logs para J.A.R.V.I.S.

28.3.1 Requisitos Técnicos

```
# Estructura mínima requerida
class DiagnosticoMedico:
    def __init__(self):
        # Inicializar sistema
        pass

    def monitorear_vitales(self) -> dict:
        """Simula monitoreo de signos vitales"""
        pass

    def diagnosticar(self, sintomas: list) -> dict:
```

```
        """Diagnostica basado en síntomas"""
        pass

    def recomendar(self, diagnostico: dict) -> list:
        """Recomienda acciones"""
        pass

    def registrar(self, evento: str):
        """Registra evento en log"""
        pass

# Funcionalidades mínimas:
# 1. Leer datos del "paciente" (Tony)
# 2. Analizar 5 signos vitales diferentes
# 3. Diagnosticar 3 condiciones posibles
# 4. Recomendar acciones específicas
# 5. Guardar log detallado
```

28.3.2 Constraints del Tiempo

Restricción	Valor
Tiempo máximo	30 minutos
Líneas de código	Máximo 150
Complejidad	Básica (como Mark I)
Calidad	Debe funcionar

28.3.3 Evaluación

Criteria	Puntos	Completado
Funcionalidad	40 pts	
Código limpio	20 pts	
Logs funcionales	20 pts	
Documentación	20 pts	
TOTAL	**100 pts**	

28.4 Ejecución

28.4.1 Paso 1: Preparación (2 min)

```
# Crear entorno
mkdir -p ~/boss-fight-1/{src,logs}
cd ~/boss-fight-1
```

28.4.2 Paso 2: Desarrollo (25 min)

1. **Minuto 0-5**: Planificar estructura
2. **Minuto 5-15**: Implementar funciones core
3. **Minuto 15-25**: Probar y ajustar
4. **Minuto 25-30**: Documentar

28.4.3 Paso 3: Prueba (3 min)

```
# Ejecutar sistema
python src/diagnostico.py

# Verificar logs
cat logs/tony-stark.log
```

28.5 Logro Desbloqueado: "Cave Survivor"

28.5.1 Requisitos para Desbloquear

- ☐ Sistema ejecutándose sin errores
- ☐ Mínimo 5 logs generados
- ☐ Código documentado
- ☐ Tiempo bajo 30 minutos

28.5.2 Recompensa

- **+150 XP**
- **Logro "Cave Survivor"**
- **Acceso a Nivel 2**

28.5.3 Siguiente Nivel

"Ahora viene la parte divertida." — Tony Stark

→ Nivel 2: El Mark I

28.6 Post-Battle Analysis

28.6.1 Tu Desempeño

```
## Análisis Post-Combate
- **Tiempo total**: ___ minutos
- **Líneas de código**: ___
- **Funciones creadas**: ___
- **Logs generados**: ___

## Lecciones Aprendidas
1. [Lo que salió bien]
2. [Lo que mejoraría]
3. [Lo que aprendí]
```

Part III

Nivel 2: El Mark I

29 Nivel 2: El Mark I

30 Nivel 2: El Mark I

30.1 Prototipo: De Scripts Sueltos a Sistema Organizado

30.2 La Escena de Iron Man (2008)

> *"Mark I. No es elegante. No es particularmente limpio. Pero funciona."* — Tony Stark

Tony Stark termina el Mark I.

Es desordenado, ruidoso, limitado a 2 minutos de vuelo, y hecho con chatarra.

Pero es **su primera armadura funcional**.

Con ella, puede derrotar a los terroristas y escapar.

El Mark I tiene reglas claras:

- **Límite de energía**: 2 minutos de vuelo
- **Armamento**: Lanzallamas (básico pero efectivo)
- **Debilidad**: Muy vulnerable a ataques
- **Fuerza**: Mayor que la humana

En este nivel, pasarás de scripts sueltos a un "Mark I" de desarrollo:

Un sistema con reglas básicas (AGENTS.md) y contexto específico.

No será perfecto, pero será tu primera **arquitectura de desarrollo con IA**.

30.3 Objetivos de Aprendizaje

Al completar este nivel, serás capaz de:

1. **Entender** qué es Context Engineering y por qué es más importante que Prompt Engineering
2. **Crear** tu primer `AGENTS.md` con reglas básicas
3. **Diseñar** una estructura de proyecto para IA
4. **Implementar** el patrón "Context Before Prompt"
5. **Diferenciar** entre contexto estático y dinámico

30.4 Conceptos Técnicos

30.4.1 2.1 Context Engineering vs Prompt Engineering

30.4.1.1 La Gran Diferencia

Prompt Engineering	Context Engineering
"Escribe buenos prompts"	"Crea sistemas que dan buenos resultados"
Enfocado en una interacción	Enfocado en múltiples sesiones
Repite contexto en cada prompt	El contexto persiste
Como dar instrucciones cada vez	Como tener un manual de procedimientos

30.4.1.2 Analogía del Taller

- **Prompt Engineering**: Le dices a cada nuevo obrero qué hacer
- **Context Engineering**: Tienes un manual de taller que todos leen

30.4.2 2.2 AGENTS.md: Tu Manual de Procedimientos

El `AGENTS.md` es el **corazón del Context Engineering**. Es un archivo que le dice a la IA **quién eres, qué haces, y cómo trabajas**.

30.4.2.1 Estructura Básica de AGENTS.md

```
# Proyecto: [Nombre]
# Autor: [Tu nombre]
# Stack: [Tecnologías]

## Reglas Generales
- Usa [lenguaje] como lenguaje principal
- Sigue [estilo de código] para naming
- Responde en [idioma]

## Estructura del Proyecto
- src/: Código fuente
- tests/: Tests unitarios
- docs/: Documentación
- context/: Archivos de contexto

## Convenciones de Código
- Funciones: snake_case
- Clases: PascalCase
- Constantes: UPPER_SNAKE_CASE
- Comentarios: español, explicativos

## Restricciones
- No uses dependencias sin justificación
- Siempre incluye tests
- Documenta funciones públicas
```

30.4.2.2 Ejemplo Avanzado: AGENTS.md con Zero Trust

```
# Proyecto: Iron Man API
# Autor: Tony Stark
# Stack: Python 3.11, FastAPI, PostgreSQL

## Reglas Generales
- Usa Python 3.11+ como lenguaje principal
- Sigue PEP 8 para naming
- Responde en español técnico

## Security First (Principio Fundamental)
  **REGLA #1**: Todo input del usuario es potencialmente malicioso.
  **REGLA #2**: Nunca confíes, siempre valida.
  **REGLA #3**: Los secrets NUNCA van en el código.

### Validación de Input (Zero Trust)
- Valida TODOS los inputs antes de procesar
```

```
- Usa Pydantic para validación automática
- Rechaza datos que no cumplan el schema
- Loggea intentos de input inválido

### Gestión de Secretos
- API keys en variables de ambiente (.env)
- Nunca hardcodees passwords
- Usa python-dotenv para cargar secrets
- Rotación de keys cada 90 días

## Estructura del Proyecto
- src/
  - api/: Endpoints FastAPI
  - models/: Modelos Pydantic + SQLAlchemy
  - services/: Lógica de negocio
  - security/: Funciones de seguridad
- tests/: Tests unitarios + integración
- docs/: Documentación
- .env.example: Template de variables

## Convenciones de Código
- Funciones: snake_case
- Clases: PascalCase
- Constantes: UPPER_SNAKE_CASE
- Type hints obligatorios en todas las funciones
- Docstrings en español, explicativos

## Restricciones de Seguridad
- SIEMPRE valida inputs con Pydantic
- SIEMPRE usa parameterized queries (SQL injection)
- SIEMPRE escapa outputs (XSS prevention)
- NUNCA uses eval() o exec()
- NUNCA logs contraseñas o tokens
- NUNCA compartas secrets en commits
```

30.4.2.3 ¿Por qué Zero Trust es importante?

Tony Stark aprendió esto cuando HYDRA hackó sus armaduras:

> **"No confíes ni siquiera en ti mismo. Verifica todo."**

Principio	Ejemplo en Código
Valida todo	`if not email.match(EMAIL_REGEX): raise`

Principio	Ejemplo en Código
No confíes	`user_id = get_current_user(request)` — verifica el token
Fail-safe	`raise HTTPException(401)` por defecto, no 200
Defense in depth	Validación + Auth + Rate limiting + Logging

30.4.3 2.3 El Patrón "Context Before Prompt"

30.4.3.1 Flujo Correcto

```
1. Crear/registrar contexto (AGENTS.md, context/)
2. Abrir sesión con IA
3. La IA lee el contexto automáticamente
4. Tú escribes prompts específicos
5. La IA responde siguiendo las reglas
```

30.4.3.2 Flujo Incorrecto (común)

```
1. Abrir sesión con IA
2. Escribir prompt largo explicando todo
3. La IA responde genéricamente
4. Repetir paso 2 con más detalles
5. Resultado inconsistente
```

30.4.4 2.4 Contexto Estático vs Dinámico

30.4.4.1 Contexto Estático

Archivos que no cambian frecuentemente:

- `AGENTS.md` — Reglas del proyecto
- `README.md` — Documentación general
- `CONTRIBUTING.md` — Guía de contribución

30.4.4.2 Contexto Dinámico

Archivos que cambian con frecuencia:

- `context/database-schema.sql` — Esquema de BD
- `context/api-endpoints.md` — Endpoints actuales
- `context/decisions.log` — Decisiones tomadas

30.5 Laboratorio 2: Construyendo tu Mark I

30.5.1 Ejercicio 1: El reactor arc de contexto (Estructura básica)

30.5.1.1 Objetivo

Crear la estructura básica de un proyecto con IA.

30.5.1.2 Comandos

```
# Crear estructura de proyecto
mkdir -p ~/iron-man-project/{src,tests,context,docs,prompts,logs}
cd ~/iron-man-project

# Crear archivos básicos
touch AGENTS.md README.md .gitignore
touch context/database-schema.md
touch context/api-endpoints.md
touch context/decisions.log

# Verificar estructura
tree .
```

30.5.1.3 Resultado Esperado

```
iron-man-project/
   AGENTS.md
   README.md
   .gitignore
   context/
       database-schema.md
       api-endpoints.md
       decisions.log
   docs/
   logs/
   prompts/
   src/
   tests/
```

30.5.2 Ejercicio 2: Tu primer AGENTS.md (Reglas del Mark I)

30.5.2.1 Objetivo

Crear un `AGENTS.md` funcional para un proyecto simple.

30.5.2.2 Prompt para IA

```
Necesito crear un AGENTS.md para mi proyecto "Iron Man Evolution".

Especificaciones:
1. Lenguaje: Python 3.11+
2. Estilo de código: PEP 8
3. Testing: pytest
4. Documentación: Docstrings en español
5. Convenciones: snake_case para funciones, PascalCase para clases

Incluye estas secciones:
- Stack tecnológico
- Estructura de carpetas
- Convenciones de código
- Reglas de testing
- Patrones de diseño preferidos
- Qué NO hacer (restricciones)

Genera el contenido completo del archivo AGENTS.md.
```

30.5.2.3 Archivo AGENTS.md Esperado

El archivo AGENTS.md debe seguir esta estructura:

Encabezado y Stack Tecnológico:

```
# Proyecto: Iron Man Evolution
# Autor: Diego Saavedra García
# Stack: Python 3.11+, FastAPI, SQLAlchemy, Pytest
# Versión: 1.0

##  Stack Tecnológico
- **Lenguaje**: Python 3.11+
- **Framework Web**: FastAPI
- **ORM**: SQLAlchemy
- **Testing**: Pytest + Coverage
- **Documentación**: MkDocs
- **CI/CD**: GitHub Actions
- **Containerización**: Docker + Docker Compose
```

Estructura de Directorios:

```
iron-man-project/
  AGENTS.md          # Este archivo
  README.md          # Documentación principal
  pyproject.toml     # Configuración Python
  src/               # Código fuente
     models/        # Modelos de datos
     services/      # Lógica de negocio
     api/           # Endpoints API
     utils/         # Utilidades
  tests/             # Tests unitarios
     unit/          # Tests unitarios
     integration/   # Tests de integración
     conftest.py    # Fixtures de pytest
  docs/              # Documentación
  scripts/           # Scripts auxiliares
  context/           # Archivos de contexto
```

Convenciones de Código:

30.5.3 Nomenclatura

- **Funciones y variables**: `snake_case`
- **Clases**: `PascalCase`
- **Constantes**: `UPPER_SNAKE_CASE`
- **Módulos**: `snake_case.py`
- **Archivos de test**: `test_*.py`

30.5.4 Estilo

- Seguir PEP 8 estrictamente
- Máximo 79 caracteres por línea
- Usar type hints en todas las funciones
- Docstrings en español con formato Google

30.5.5 Ejemplo

```python
def calcular_energia_reactor(
    nivel_base: int,
    factor_eficiencia: float = 1.0
) -> int:
    """
    Calcula la energía del reactor arc basado en el nivel base.

    Args:
        nivel_base: Nivel base de energía (1-100).
            Valores fuera de rango lanzan ValueError.
        factor_eficiencia: Factor de eficiencia (0.0-1.0).
            1.0 = máxima eficiencia, 0.0 = sin energía.

    Returns:
        Energía calculada en unidades de arc (entero).

    Raises:
        ValueError: Si nivel_base no está entre 1-100.
        ValueError: Si factor_eficiencia no está entre 0.0-1.0.

    Example:
        >>> calcular_energia_reactor(50, 0.8)
        40000
    """
    # Validar nivel_base
    if not 1 <= nivel_base <= 100:
        raise ValueError(
            f"nivel_base debe estar entre 1-100, recibido: {nivel_base}"
        )

    # Validar factor_eficiencia
    if not 0.0 <= factor_eficiencia <= 1.0:
        raise ValueError(
            f"factor_eficiencia debe estar entre 0.0-1.0, recibido: {factor_eficiencia}"
        )

    # Calcular energía
    energia_base = nivel_base * 1000
    energia_ajustada = energia_base * factor_eficiencia

    return int(energia_ajustada)
```

30.6 Reglas de Testing

- **Cobertura mínima**: 80%
- **Framework**: pytest

- **Nombres de test**: `test_` + descripción del comportamiento
- **Estructura**: Arrange-Act-Assert (AAA)
- **Cada test**: Una sola aserción lógica

30.6.1 Ejemplo de Test

```python
import pytest
from src.services.reactor import calcular_energia_reactor

class TestCalcularEnergiaReactor:
    """Tests para la función calcular_energia_reactor."""

    def test_deberia_calcular_energia_correcta(self):
        """Debería calcular energía correctamente con datos normales."""
        # Arrange
        nivel_base = 50
        factor = 0.8

        # Act
        resultado = calcular_energia_reactor(nivel_base, factor)

        # Assert
        assert resultado == 40000

    def test_deberia_lanzar_error_con_nivel_invalido(self):
        """Debería lanzar ValueError con nivel fuera de rango."""
        # Act & Assert
        with pytest.raises(ValueError):
            calcular_energia_reactor(101)
```

30.7 Patrones de Diseño Preferidos

1. **Repository Pattern**: Para acceso a datos
2. **Service Layer**: Para lógica de negocio
3. **Dependency Injection**: Para desacoplamiento
4. **Factory Pattern**: Para creación de objetos complejos
5. **Observer Pattern**: Para eventos del sistema

30.8 Restricciones (Qué NO Hacer)

- **NO** usar `print()` para logging (usar logging module)
- **NO** hardcoded values (usar variables de entorno o constantes)
- **NO** imports circulares
- **NO** funciones con más de 3 parámetros

- **NO** clases con más de 5 métodos públicos
- **NO** código duplicado (DRY)
- **NO** dependencias sin justificación en pyproject.toml
- **NO** tests que dependen de otros tests
- **NO** datos sensibles en el código (usar .env)
- **NO** commit sin tests pasando

30.9 Configuración de Herramientas

30.9.1 Pytest

```
[tool.pytest.ini_options]
testpaths = ["tests"]
python_files = ["test_*.py"]
python_functions = ["test_*"]
addopts = "-v --cov=src --cov-report=term-missing"
```

30.9.2 Black (Formateador)

```
[tool.black]
line-length = 79
target-version = ['py311']
include = '\.pyi?$'
```

30.10 Comandos Útiles

```
# Ejecutar tests
pytest

# Ejecutar tests con cobertura
pytest --cov=src --cov-report=html

# Formatear código
black src/ tests/

# Verificar tipos
mypy src/

# Ejecutar servidor de desarrollo
uvicorn src.api.main:app --reload
```

30.11 Notas para IA

30.11.1 Al leer este archivo, la IA debe:

1. Seguir todas las convenciones de código
2. Usar type hints en todas las funciones
3. Crear tests para cualquier nueva función
4. Documentar en español con docstrings
5. Seguir el patrón AAA en tests
6. Mantener cobertura > 80%
7. No usar hardcoding
8. Usar repository pattern para datos
9. Validar inputs en las fronteras
10. Logging apropiado (no print)

30.11.2 Si encuentra código que viola estas reglas:

1. **Advertir** al desarrollador
2. **Sugerir** la corrección
3. **Proponer** refactorización si es grave

Archivo mantenido por: Iron Man Evolution Team

Última actualización: 2026-03-22

30.11.3 Ejercicio 3: Archivos de Contexto Dinámico

30.11.3.1 Objetivo

Crear archivos de contexto que la IA pueda leer para entender tu proyecto específico.

30.11.3.2 3.1 Database Schema Context

Crear `context/database-schema.md`:

```
# Esquema de Base de Datos - Iron Man Evolution

## Tablas Principales

### 1. tabla_pilotos
| Columna | Tipo | Descripción |
|---------|------|-------------|
| id | INTEGER | Primary key |
| nombre | VARCHAR(100) | Nombre del piloto |
| email | VARCHAR(255) | Email único |
| nivel_energia | INTEGER | Energía actual (1-100) |
| fecha_registro | TIMESTAMP | Cuándo se registró |
| activo | BOOLEAN | Si está activo |

### 2. tabla_armaduras
| Columna | Tipo | Descripción |
|---------|------|-------------|
| id | INTEGER | Primary key |
| nombre | VARCHAR(50) | Nombre armadura (Mark I, II, etc.) |
| version | VARCHAR(20) | Versión específica |
| nivel_potencia | INTEGER | 1-100 |
| descripcion | TEXT | Características |
| fecha_creacion | TIMESTAMP | Cuándo fue creada |

### 3. tabla_misiones
| Columna | Tipo | Descripción |
|---------|------|-------------|
| id | INTEGER | Primary key |
| piloto_id | INTEGER | FK a tabla_pilotos |
| armadura_id | INTEGER | FK a tabla_armaduras |
| nombre | VARCHAR(200) | Nombre de la misión |
| estado | ENUM | 'pendiente', 'en_progreso', 'completada', 'fallida' |
| energia_consumida | INTEGER | Energía usada |
| fecha_inicio | TIMESTAMP | Inicio |
| fecha_fin | TIMESTAMP | Fin |

## Relaciones
- Un piloto puede tener múltiples misiones
- Una armadura puede usarse en múltiples misiones
- Cada misión tiene exactamente un piloto y una armadura

## Ejemplos de Queries Útiles
```sql
-- Pilotos con más energía
SELECT * FROM tabla_pilotos
WHERE nivel_energia > 70
ORDER BY nivel_energia DESC;
```
```

```sql
-- Misiones recientes
SELECT p.nombre, a.nombre, m.estado, m.fecha_inicio
FROM tabla_misiones m
JOIN tabla_pilotos p ON m.piloto_id = p.id
JOIN tabla_armaduras a ON m.armadura_id = a.id
WHERE m.fecha_inicio > DATE_SUB(NOW(), INTERVAL 7 DAY);
```

30.11.3.3 3.2 API Endpoints Context

Crear `context/api-endpoints.md`:

```markdown
# API Endpoints - Iron Man Evolution

## Base URL

http://localhost:8000/api/v1

---

## Endpoints de Pilotos

### GET /pilotos

Obtiene lista de pilotos.

**Parámetros query:**
- `limite` (opcional): Máximo resultados, default 10
- `offset` (opcional): Paginación, default 0

**Response:**
```json
{
  "pilotos": [...],
  "total": 100
}
```

30.11.4 POST /pilotos

Crea nuevo piloto.

Body:
```

```
{
  "nombre": "Tony",
  "email": "tony@stark.com"
}
```

Response: 201 Created

30.11.5 GET /pilotos/{id}

Obtiene piloto por ID.

Response:

```
{
  "id": 1,
  "nombre": "Tony",
  "email": "tony@stark.com",
  "nivel_energia": 75
}
```

30.11.6 PUT /pilotos/{id}

Actualiza piloto completo.

Body:

```
{
  "nombre": "Tony Stark",
  "nivel_energia": 85
}
```

Response: Piloto actualizado

30.11.7 DELETE /pilotos/{id}

Elimina piloto por ID.

Response: 204 No Content

30.12 Endpoints de Misiones

30.12.1 POST /misiones

Crea nueva misión.

Body:

```
{
  "piloto_id": 1,
  "armadura_id": 3,
  "nombre": "Rescate"
}
```

Response: 201 Created

30.12.2 PATCH /misiones/{id}/estado

Actualiza estado de misión.

Body:

```
{
  "estado": "completada",
  "energia_consumida": 45
}
```

Response: Misión actualizada

30.13 Códigos de Respuesta

Código	Significado	Descripción
200	OK	Éxito
201	Created	Recurso creado
204	No Content	Eliminado exitosamente
400	Bad Request	Datos inválidos
401	Unauthorized	No autenticado
403	Forbidden	Sin permisos
404	Not Found	No encontrado
422	Validation Error	Validación fallida

Código	Significado	Descripción
500	Internal Error	Error del servidor

```
---

### **Ejercicio 4: Prueba de Context Before Prompt**

#### **Objetivo**

Probar que el sistema de contexto funciona.

---

#### **Test**

**Paso 1:** Abre una nueva sesión con Claude/ChatGPT

**Paso 2:** **NO** expliques nada al principio

**Paso 3:** Sube los archivos:

- `AGENTS.md`
- `context/database-schema.md`
- `context/api-endpoints.md`

**Paso 4:** Escribe este prompt simple:

```python
Crea una función Python para crear un nuevo piloto en la base de datos.
```

30.13.0.1 Resultado Esperado

La IA debería:

1. Usar `snake_case` para la función
2. Incluir type hints
3. Crear docstring en español
4. Seguir el esquema de BD correcto
5. Manejar errores apropiadamente
6. Incluir tests unitarios

```

30.14 Logro Desbloqueado: "Architect"

30.14.1 Requisitos para Desbloquear

- ☐ Crear estructura de proyecto completa
- ☐ Escribir `AGENTS.md` funcional (>100 líneas)
- ☐ Crear al menos 2 archivos de contexto dinámico
- ☐ Probar el sistema "Context Before Prompt"
- ☐ Entender la diferencia entre contexto estático y dinámico

30.14.2 Recompensa

- **75 XP** por cada ejercicio completado
- **Logro "Architect"** en tu perfil
- **Acceso** al Nivel 3: El Mark III
- **Desbloqueo** de patrones avanzados

30.15 Recursos Adicionales

30.15.1 Lectura

- Context Engineering Guide
- AGENTS.md Pattern
- Clean Code Principles

30.15.2 Herramientas

- Tree (generador de estructura)
- Black (formateador Python)
- Mypy (verificador de tipos)

30.15.3 Videos

- Gentle AI Stack Installation
- AGENTS.md Tutorial

30.16 Siguiente Nivel

¿Completaste todos los ejercicios?

→ Nivel 3: El Mark III

¿Necesitas más práctica?

→ Lab Detallado Nivel 2

"El Mark I no era sobre la tecnología. Era sobre la idea. La idea de que incluso con chatarra, puedes crear algo que te liberará."

— Howard Stark (imaginario)

31 Lab 2: El Mark I - Prototipo Funcional

32 Lab 2: El Mark I - Prototipo Funcional

32.1 De Scripts Sueltos a Sistema Organizado

32.2 La Situación

> *"Mark I. No es elegante. No es particularmente limpio. Pero funciona."* — Tony Stark

Tony Stark termina el Mark I. Es desordenado, ruidoso, limitado a 2 minutos de vuelo, y hecho con chatarra. Pero es **su primera armadura funcional**. Con ella, puede derrotar a los terroristas y escapar.

El Mark I tiene reglas claras: - **Límite de energía**: 2 minutos de vuelo - **Armamento**: Lanzallamas (básico pero efectivo) - **Debilidad**: Muy vulnerable a ataques - **Fuerza**: Mayor que la humana

En este lab, pasarás de scripts sueltos a un "Mark I" de desarrollo: un sistema con reglas básicas (AGENTS.md) y contexto específico. No será perfecto, pero será tu primera **arquitectura de desarrollo con IA**.

32.3 Timeline de la Misión

Paso	Descripción	Tiempo	Completado
1	Diseñar tu AGENTS.md (manual de procedimientos)	20 min	
2	Crear estructura de proyecto	15 min	
3	Implementar "Context Before Prompt"	15 min	
4	Ejecutar tu primer request con contexto completo	15 min	
Total		**65 min**	

32.4 Objetivo del Lab

Crear tu primera arquitectura de desarrollo con IA. Al finalizar, tendrás:

1. Un archivo AGENTS.md con reglas básicas
2. Una estructura de proyecto para IA
3. Implementado el patrón "Context Before Prompt"
4. Ejecutado tu primer request con contexto completo

32.5 Regla del Stark Protocol

> **"No es elegante. No es particularmente limpio. Pero funciona."** — Tony Stark

REGLA ORO: Los ejercicios te hacen completar, no copiar. Esto te obliga a pensar.

32.6 1. Prerrequisitos del Mark I

32.6.1 1.1. Qué Debes Tener Listo

- Completado Lab 1: El Demo en la Cueva
- Herramienta de IA configurada
- Sistema de organización creado
- Contexto default definido

32.6.2 1.2. Herramientas Necesarias

Herramienta	Propósito	Estado
Tu IA elegida	Repartidor de energía	
Editor de código	Taller de armadura	
Terminal	Panel de control	
Tu AGENTS.md	Manual de procedimientos	

32.7 2. Escenario de la Misión

Eres Tony Stark, y acaba de terminar el Mark I. Tienes:

1. **Un sistema básico** (scripts sueltos del Lab 1)
2. **Una herramienta de IA** (tu repartidor de energía)
3. **Una misión**: Crear una arquitectura que funcione

Tu Mark I tendrá: - **Reglas claras** (AGENTS.md como manual de procedimientos) - **Estructura básica** (carpetas organizadas) - **Contexto persistente** (no solo prompts sueltos)

32.8 3. Pasos de la Misión

32.8.1 Paso 1: Diseña tu AGENTS.md (Manual de Procedimientos)

Objetivo: Crear el "manual de taller" que J.A.R.V.I.S. usaría para entender cómo trabajas.

Tu AGENTS.md debe incluir:

```
# AGENTS.md - [Tu Nombre] Stark Industries
# Versión: 1.0 Mark I
# Autor: [Tu nombre]
# Stack: [Tecnologías principales]

##  Misión Principal
[Describe brevemente qué haces y cómo trabajas]

##  Estructura del Proyecto
- src/: Código fuente
- tests/: Tests unitarios
- docs/: Documentación
- context/: Archivos de contexto
- logs/: Registro de actividad

##  Reglas de Conducta
1. **Lenguaje principal**: [Tu lenguaje preferido]
2. **Estilo de código**: [Tu convención de naming]
3. **Idioma de respuestas**: [Español/Inglés]
4. **Nivel de detalle**: [Básico/Intermedio/Avanzado]

##  Convenciones de Código
- Funciones: [snake_case/camelCase]
- Clases: [PascalCase]
```

```
- Constantes: [UPPER_SNAKE_CASE]
- Comentarios: [español/explicativos]

##  Restricciones Críticas
- No usar dependencias sin justificación
- Siempre incluir tests
- Documentar funciones públicas
- Seguir principios SOLID básicos

##  Contexto Siempre Incluir
1. [Tu nivel de experiencia]
2. [Tus preferencias de desarrollo]
3. [Tus restricciones conocidas]

##  Lo que NO hacer
- [Lista de acciones prohibidas]
- [Patrones a evitar]
- [Errores comunes a prevenir]
```

Tu trabajo: Copia este template y personalízalo para tu situación.

32.8.2 Paso 2: Crea tu Estructura de Proyecto "Mark I"

Objetivo: Organizar tu espacio de trabajo como Tony organizaría su taller.

Estructura base:

```
# Crea tu proyecto Mark I
mkdir -p ~/mark-i-project/{src,tests,docs,context,logs,prompts}
cd ~/mark-i-project

# Crea los archivos base
touch AGENTS.md README.md .gitignore
touch src/__init__.py tests/__init__.py
```

Resultado esperado:

```
mark-i-project/
   AGENTS.md          # Tu manual de procedimientos
   README.md          # Documentación del proyecto
   .gitignore         # Archivos a ignorar
   src/               # Código fuente
       __init__.py
   tests/             # Tests unitarios
```

```
    __init__.py
docs/               # Documentación adicional
context/            # Archivos de contexto
logs/               # Registro de actividad
prompts/            # Mejores prompts guardados
```

Tu trabajo: Crea esta estructura y personalízala.

32.8.3 Paso 3: Implementa "Context Before Prompt"

Objetivo: Probar que dar contexto ANTES de hacer prompts mejora los resultados.

Ejercicio comparativo:

32.8.3.1 Versión A: SIN Contexto (Mark I defectuoso)

```
Escríbeme una función que procese pagos
```

32.8.3.2 Versión B: CON Contexto (Mark I funcional)

```
Actúa como desarrollador senior de Python siguiendo estas reglas:

[PEGAR CONTENIDO DE AGENTS.md AQUÍ]

## Tarea Específica
Necesito una función `process_payment(payment_data: dict) -> PaymentResult`
para un sistema de e-commerce.

### Restricciones Técnicas
- Moneda: USD
- Validar: card number (Luhn algorithm), expiry date, CVV
- Retornar: status, transaction_id, timestamp, amount
- Formato: JSON con type hints
- Manejo de errores: CardInvalidError, ExpiredCardError, NetworkError

### Casos Edge
- Tarjetas con espacios o guiones
- Fechas de expiración pasadas
- CVVs de 3 o 4 dígitos
- Montos negativos o cero
- Timeout en conexión (30 segundos)
```

```
### Tests Requeridos
- pytest con fixtures
- Mínimo 80% coverage
- Test de integración con mock de pasarela de pago
```

Tu trabajo: Ejecuta AMBAS versiones y compara los resultados.

32.8.4 Paso 4: El Ejercicio Final - Tu Primer "Mark I" Completo

Misión: Crear un sistema funcional simple usando tu nueva arquitectura.

Especificación:

> Crea un sistema de "Inventario de Stark Industries" que: 1. Use tu AGENTS.md como contexto 2. Tenga estructura de proyecto organizada 3. Incluya tests básicos 4. Documente correctamente

Requisitos técnicos: - Python 3.10+ - Type hints en todas las funciones - Tests con pytest - Documentación en README.md - Uso de tu AGENTS.md como contexto

Tu prompt inicial (con contexto):

```
Actúa como desarrollador senior de Python siguiendo estas reglas:

[CONTENIDO COMPLETO DE TU AGENTS.md]

## Misión: Sistema de Inventario Stark Industries

Necesito un sistema básico de inventario que gestione:

### Entidades
- **Producto**: id, name, category, quantity, price, last_updated
- **Categoría**: id, name, description
- **Movimiento**: id, product_id, type (IN/OUT), quantity, timestamp, reason

### Operaciones CRUD
1. Crear/leer/actualizar/eliminar productos
2. Gestionar categorías
3. Registrar movimientos de inventario
4. Consultar stock por categoría
5. Reporte de movimientos por rango de fechas

### Restricciones
- Almacenamiento: JSON files (simplicidad del Mark I)
- Validaciones: Cantidades no negativas, precios positivos
- Logging: Todos los movimientos a logs/inventario.log
```

```
- Tests: Mínimo 5 tests básicos

### Resultado Esperado
1. Estructura de proyecto completa
2. Código funcionante con type hints
3. Tests ejecutables
4. README con instrucciones
5. AGENTS.md actualizado con learnings
```

32.9 4. Reflexión del Piloto

"Cada conversación revela algo nuevo que faltaba." — J.A.R.V.I.S.

32.9.1 Análisis Post-Misión

Pregunta 1: Comparación de Resultados > ¿Qué diferencia notable hay entre el código generado con contexto vs sin contexto?

```
## Comparación Mark I

**SIN contexto (versión defectuosa)**:
------------------------------------------------

**CON contexto (versión funcional)**:
------------------------------------------------

**Mejora específica**:
------------------------------------------------
```

Pregunta 2: Poder del AGENTS.md > ¿Cómo cambió tu AGENTS.md los resultados de la IA?

```
## Impacto del Manual de Procedimientos

**Antes de AGENTS.md**:
------------------------------------------------

**Después de AGENTS.md**:
------------------------------------------------

**Lección aprendida**:
------------------------------------------------
```

Pregunta 3: Eficiencia del Sistema > ¿Cuánto tiempo ahorraste usando "Context Before Prompt" vs prompts sueltos?

```
## Análisis de Eficiencia

**Tiempo sin sistema**: [X] minutos
**Tiempo con sistema**: [Y] minutos
**Ahorro**: [Z] minutos ([W]% más eficiente)

**Principal ganancia**:
-----------------------------------------------
```

32.10 5. Verificación de la Misión

32.10.1 Checklist del Mark I

Antes de avanzar, verifica que tienes:

- ☐ AGENTS.md creado y personalizado
- ☐ Estructura de proyecto "Mark I" creada
- ☐ Al menos un ejercicio "Context Before Prompt" ejecutado
- ☐ Sistema de inventario (o equivalente) funcionando
- ☐ Tests básicos pasando
- ☐ Reflexiones completadas

32.10.2 Prueba de Fuego

```
# Verifica tu Mark I
cd ~/mark-i-project

# Estructura correcta
ls -la
# Deberías ver: AGENTS.md, README.md, src/, tests/, etc.

# Tests funcionando
python -m pytest tests/ -v
# Deberías ver: tests passed

# Código ejecutable
python src/inventario.py --help
# Deberías ver: ayuda del sistema
```

32.11 6. Entregable: Tu Mark I Funcional

32.11.1 Archivos a Entregar

1. **`mark-i-project/`** (carpeta completa con tu sistema)
2. **`analisis-mark-i.md`** (reflexiones y aprendizajes)

32.11.2 Estructura de `analisis-mark-i.md`:

```
# Análisis Mark I - Sistema Funcional
# Piloto: [Tu nombre]
# Fecha: [Fecha actual]

## 1. Comparación de Resultados

### Sin Contexto (Mark I Defectuoso)
[Resultado obtenido]

### Con Contexto (Mark I Funcional)
[Resultado obtenido]

### Mejoras Específicas
1. [Mejora 1]
2. [Mejora 2]
3. [Mejora 3]

## 2. Impacto del AGENTS.md

### Antes
[Qué pasaba sin AGENTS.md]

### Después
[Qué pasa con AGENTS.md]

### Lecciones Aprendidas
[Lo que descubriste]

## 3. Eficiencia del Sistema

### Métricas
- Tiempo sin sistema: [X] min
- Tiempo con sistema: [Y] min
- Mejora: [Z]%

### Principal Ganancia
[Lo más valioso que ganaste]
```

```
## 4. Mejoras para el Mark II

### Para la Próxima
1. [Mejora 1]
2. [Mejora 2]
3. [Mejora 3]

## 5. Código Destacado

[Pegar aquí tu mejor función o clase del sistema]
```

32.12 7. Logro Desbloqueado: "Primera Armadura Funcional"

32.12.1 Recompensa por Completar este Lab

- **+200 XP** por crear tu primera arquitectura funcional
- **Logro "Mark I"** — Primer sistema con reglas claras
- **Acceso** a Nivel 3: El Mark III

32.12.2 Siguiente Paso

> *"Ahora viene la parte divertida."* — Tony Stark

Con tu Mark I funcionando, es hora de crear algo más sofisticado: **Nivel 3: El Mark III →**

32.13 8. Recursos de la Misión

32.13.1 Documentación Técnica

- AGENTS.md Template
- Context Engineering vs Prompt Engineering
- Patrón "Context Before Prompt"

32.13.2 Herramientas

- pytest Documentation
- Python Type Hints
- Git Basics

33 Boss Fight 2: Mark I vs Mark II

34 Boss Fight 2: Mark I vs Mark II

34.1 Prueba Final del Nivel 2

34.2 La Situación

> *"Necesito algo más que un prototipo. Necesito algo que funcione de verdad."* — Tony Stark

Tony tiene el Mark I, pero no es suficiente. Debe crear el **Mark II**: más fuerte, más resistente, más profesional. La diferencia entre prototipo y producto.

Tu misión: Transformar tu código de "funciona" a "profesional".

34.3 Misión: Sistema de Gestión de Inventarios Mejorado

34.3.1 Contexto del Problema

Tu código del Mark I "funciona", pero: - No tiene tests - No tiene documentación clara - No sigue estándares de código - No es mantenible

34.3.2 Tareas del Mark II

34.3.2.1 Tarea 1: AGENTS.md Profesional (30 min)

```
# AGENTS.md - Stark Industries Inventory v2.0

## Arquitectura
[Descripción de la arquitectura]

## Estándares de Código
- Python 3.10+ con type hints
- Funciones < 20 líneas
- Comentarios solo el "por qué"
```

```
- Tests para todo código público

## Convenciones
- Nombres: snake_case
- Tests: test_[función]_[escenario]
- Docs: Google Style

## Flujos de Trabajo
1. Escribir test primero
2. Implementar función
3. Refactorizar
4. Documentar
```

34.3.2.2 Tarea 2: Tests Completos (40 min)

```python
# tests/test_inventario.py
import pytest
from src.inventario import Inventario

class TestInventario:
    def test_crear_producto_exitoso(self):
        """Test creación de producto válido"""
        inv = Inventario()
        producto = inv.crear_producto("Mark II", "Armadura", 1)
        assert producto.nombre == "Mark II"
        assert producto.categoria == "Armadura"

    def test_crear_producto_datos_invalidos(self):
        """Test creación con datos inválidos"""
        inv = Inventario()
        with pytest.raises(ValueError):
            inv.crear_producto("", "", -1)

    # Agregar mínimo 8 tests más
```

34.3.2.3 Tarea 3: Documentación Profesional (30 min)

```
# Stark Industries Inventory System v2.0

## Descripción
Sistema de gestión de inventarios para Stark Industries.

## Instalación
```

```bash
pip install -r requirements.txt
```

34.4 Uso

```python
from src.inventario import Inventario

inv = Inventario()
inv.crear_producto("Mark III", "Armadura", 5)
```

34.5 Testing

```bash
pytest tests/ -v --cov=src
```

34.6 Arquitectura

[Diagrama de arquitectura]

```
---

##  **Métricas de Éxito**

### **Criterios de Aprobación**

| Métrica | Mark I | Mark II | Mejora |
|---------|--------|---------|--------|
| **Tests** | 0% | >80% | +80% |
| **Docs** | 0% | 100% | +100% |
| **Type Hints** | 0% | 100% | +100% |
| **Coverage** | 0% | >80% | +80% |

### **Checklist de Validación**
- [ ] Todos los tests pasan
- [ ] Coverage > 80%
- [ ] Type hints en todo código público
- [ ] Documentación completa
- [ ] AGENTS.md actualizado
- [ ] README con ejemplos

---
```

```
##  **Logro Desbloqueado: "Prototype to Product"**

### **Requisitos**
- [ ] Tests ejecutándose
- [ ] Coverage > 80%
- [ ] Documentación completa
- [ ] AGENTS.md profesional

### **Recompensa**
- **+200 XP**
- **Logro "Prototype to Product"**
- **Acceso a Nivel 3**

### **Siguiente Nivel**
> *"La verdadera sofisticación está en la simplicidad."* - Tony Stark

**[→ Nivel 3: El Mark III](../contenido/nivel3.qmd)**


::: {.quarto-book-part}


`<!-- quarto-file-metadata: eyJyZXNvdXJjZURpciI6Ii4ifQ== -->`{=html}

```{=html}
<!-- quarto-file-metadata: eyJyZXNvdXJjZURpciI6Ii4iLCJib29rSXRlbVR5cGUiOiJwYXJ0IiwiYm9va0
```
```

35 Nivel 3: De la Chatarra al Artefacto

:::

36 Nivel 3: El Mark III

37 Nivel 3: El Mark III

37.1 Combate Real: De Prototipo a Sistema Autónomo

37.2 La Escena de Iron Man (2008)

"Esta vez no mevoy a contentar con algo que no sea perfecto." — Tony Stark

Tony Stark ha aprendido de sus errores.

El **Mark I** lo salvó de la cueva. El **Mark II** voló, pero se congeló en la estratósfera.

Ahora construye el **Mark III**: la primera armadura de **combate real**.

El Mark III es diferente:

- **Combina lo mejor** del Mark I y Mark II
- **Sistemas probados** bajo condiciones extremas
- **Edge cases considerados** desde el diseño
- **Feedback loop** integrado para mejorar

Pero también:

- **Mayor complejidad** — más subsystems, más puntos de falla
- **Testing exhaustivo** — cada componente debe pasar pruebas
- **Debugging crítico** — un bug en combate puede ser fatal
- **Optimización necesaria** — rendimiento perfecto o muerte

En este nivel, tú eres Tony perfeccionando el Mark III para combate.

Tu código del Mark I "funciona", pero necesitas convertirlo en un **sistema robusto**: tests completos, debugging efectivo, y código que sobreviva al combate real.

37.3 Objetivos de Aprendizaje

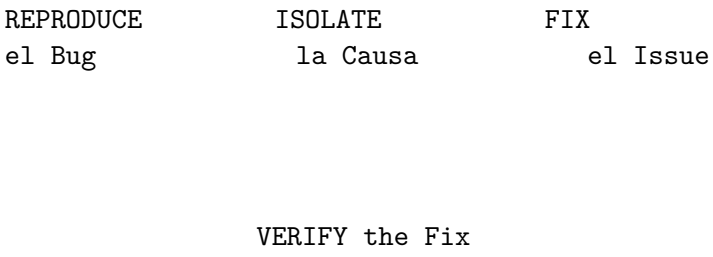
Al completar este nivel, serás capaz de:

1. **Debuggear** sistemas de IA que fallan en producción
2. **Escribir tests** que capturen edge cases
3. **Refactorizar** código manteniendo funcionalidad
4. **Optimizar** rendimiento del sistema
5. **Iterar** sobre soluciones basadas en feedback real

37.4 Conceptos Técnicos

37.4.1 3.1 Debugging de Sistemas de IA

37.4.1.1 El Ciclo de Debugging



37.4.1.2 Herramientas de Debugging

```
# Debugging interactivo con ipdb
import ipdb; ipdb.set_trace()

# Logging estructurado
import logging
logging.basicConfig(level=logging.DEBUG)
logger = logging.getLogger(__name__)
```

```
# Tracing de llamadas
from opentelemetry import trace
tracer = trace.get_tracer(__name__)

# Performance profiling
import cProfile
cProfile.run('mi_funcion()')
```

37.4.2 3.2 Testing de Edge Cases

37.4.2.1 Qué son Edge Cases?

Los edge cases son situaciones extremas o inusuales que tu código debería manejar:

Tipo	Ejemplo	Por qué falla
Boundary	`[]`, `None`, `""`	Arrays vacíos, nulos
Type	`"123"` vs `123`	Tipos incorrectos
Scale	`1_000_000` items	Problemas de memoria
Concurrency	100 requests simultáneos	Race conditions
Network	Timeout, connection refused	Fallos externos

37.4.2.2 Ejemplo de Edge Case Testing

```
import pytest

class TestEdgeCases:
    """Tests para edge cases del sistema de inventario."""

    def test_inventario_vacio(self):
        """Edge case: inventario sin productos."""
        inv = Inventario()
        assert inv.total_productos == 0
        assert inv.buscar("cualquiera") == []

    def test_producto_nombre_vacio(self):
        """Edge case: nombre de producto vacío."""
        inv = Inventario()
        with pytest.raises(ValueError):
            inv.crear_producto("", "categoria", 10)

    def test_cantidad_negativa(self):
        """Edge case: cantidad negativa."""
        inv = Inventario()
        producto = inv.crear_producto("Mark III", "armadura", 10)
```

```python
        with pytest.raises(ValueError):
            inv.actualizar_stock(producto.id, -5)

    def test_concurrencia_basica(self):
        """Edge case: múltiples accesos simultáneos."""
        import threading

        inv = Inventario()
        errores = []

        def agregar_producto():
            try:
                inv.crear_producto(f"Producto-{threading.get_ident()}", "test", 1)
            except Exception as e:
                errores.append(e)

        threads = [threading.Thread(target=agregar_producto) for _ in range(10)]
        for t in threads:
            t.start()
        for t in threads:
            t.join()

        assert len(errores) == 0
        assert inv.total_productos == 10
```

37.4.3 3.2.5 Testing de Seguridad (Security Testing)

> *"Si no testeo la seguridad, estoy probando a ciegas."* — Tony Stark (probablemente)

37.4.3.1 ¿Por qué Security Testing?

Un bug funcional rompe tu app. Un bug de seguridad **rompe tu negocio**.

37.4.3.2 Tipos de Security Tests

Tipo	Qué prueba	Ejemplo
Input Validation	¿Rechaza inputs maliciosos?	`'; DROP TABLE users;--`
Auth Testing	¿Solo usuarios autorizados acceden?	Acceder sin token
SQL Injection	¿Los queries son parameterized?	Inyección SQL básica
XSS Prevention	¿Escapes el output?	`<script>alert('xss')</script>`
Rate Limiting	¿Limita requests por IP?	1000 requests en 1 segundo

37.4.3.3 Ejemplo: Security Tests con pytest

```
import pytest
from fastapi.testclient import TestClient
from app.main import app

client = TestClient(app)

class TestSecurityValidations:
    """Tests de seguridad para la API."""

    def test_sql_injection_attempt(self):
        """SQL Injection en campo de búsqueda."""
        malicious_input = "'; DROP TABLE users;--"
        response = client.get(f"/users/search?q={malicious_input}")

        # Debe rechazar o sanitizar, NUNCA ejecutar
        assert response.status_code in [400, 422]  # Bad Request o Validation Error
        assert "DROP" not in response.text  # No debe contener SQL ejecutado

    def test_xss_attempt(self):
        """XSS en campo de comentario."""
        xss_payload = "<script>alert('hacked')</script>"
        response = client.post("/comments", json={
            "text": xss_payload,
            "user_id": 1
        })

        # El output debe estar escapado
        if response.status_code == 200:
            assert "<script>" not in response.json().get("text", "")
            assert "&lt;script&gt;" in response.json().get("text", "")

    def test_unauthorized_access(self):
        """Acceso sin token JWT."""
        response = client.get("/admin/users")

        # Debe rechazar con 401 o 403
        assert response.status_code in [401, 403]

    def test_rate_limiting(self):
        """Rate limiting en endpoint sensible."""
        # Intentar 100 requests en rápida sucesión
        responses = []
        for _ in range(100):
```

```
            r = client.post("/auth/login", json={
                "email": "test@test.com",
                "password": "wrong"
            })
            responses.append(r.status_code)

        # Debe haber al menos un 429 (Too Many Requests)
        assert 429 in responses, "Rate limiting no está funcionando"

    def test_oversized_payload(self):
        """Payload excesivamente grande."""
        huge_data = "x" * (10 * 1024 * 1024)  # 10MB

        response = client.post("/upload", data=huge_data)

        # Debe rechazar payloads grandes
        assert response.status_code in [400, 413]  # Bad Request o Payload Too Large
```

37.4.3.4 El Security Checklist de Tony Stark

Antes de cada deploy, verifica:

- ☐ Todos los inputs validados con Pydantic/Schema
- ☐ Queries SQL usan parámetros, no f-strings
- ☐ Outputs HTML están escapados
- ☐ JWT tokens tienen expiración
- ☐ Rate limiting activo en endpoints sensibles
- ☐ Secrets no están hardcodeados
- ☐ Logs no contienen passwords/tokens
- ☐ CORS está configurado correctamente

> **Recuerda**: Testing funcional verifica que "funciona". Testing de seguridad verifica que "no puede ser atacado". Necesitas ambos.

37.4.4 3.3 Refactoring Responsable

37.4.4.1 Las 3 Reglas del Refactoring

1. **No rompas los tests** — Si los tests pasan antes, deben pasar después
2. **Pequeños pasos** — Cambios incrementales, commit frecuente
3. **Entiende el código** — No refactorices código que no entiendes

37.4.4.2 Antes vs Después

```
#  ANTES: Código que funciona pero es difícil de mantener
def process(data):
    result = []
    for item in data:
        if item['type'] == 'product':
            if item['status'] == 'active':
                if item['quantity'] > 0:
                    result.append({
                        'id': item['id'],
                        'name': item['name'],
                        'price': item['price'] * 1.16  # IVA
                    })
    return result

#  DESPUÉS: Refactorizado con funciones pequeñas
def calculate_price_with_tax(price: float, tax_rate: float = 0.16) -> float:
    """Calcula precio con IVA."""
    return price * (1 + tax_rate)

def is_active_product(product: dict) -> bool:
    """Verifica si un producto está activo."""
    return (
        product.get('type') == 'product' and
        product.get('status') == 'active' and
        product.get('quantity', 0) > 0
    )

def format_product_for_sale(product: dict) -> dict:
    """Formatea un producto para venta con IVA incluido."""
    return {
        'id': product['id'],
        'name': product['name'],
        'price': calculate_price_with_tax(product['price'])
    }

def process(data: list[dict]) -> list[dict]:
    """Procesa datos de inventario y retorna productos activos."""
    return [
        format_product_for_sale(item)
        for item in data
        if is_active_product(item)
    ]
```

37.4.5 3.4 Introducción a Agentes y Sub-agentes

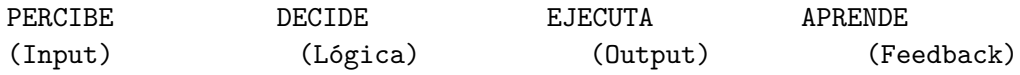
> *"Puedo construir esto. Solo necesito un poco de ayuda."* — Tony Stark (antes de crear a JARVIS)

En los niveles anteriores, construiste sistemas que **responden** ainputs. Ahora vas a construir sistemas que **actúan** por sí mismos.

37.4.5.1 ¿Qué es un Agente?

Un agente es un sistema que:

1. **Percibe** su entorno (inputs, contexto, estado)
2. **Decide** qué acción tomar (basado en objetivos)
3. **Ejecuta** la acción (produce outputs)
4. **Aprende** del resultado (mejora decisiones futuras)



37.4.5.2 Arquitectura de Agente Básico

```python
from abc import ABC, abstractmethod
from dataclasses import dataclass
from typing import Any, Optional
from enum import Enum

class AgentState(Enum):
    IDLE = "idle"
    THINKING = "thinking"
    ACTING = "acting"
    ERROR = "error"

@dataclass
class AgentContext:
    """Contexto que el agente mantiene entre ejecuciones."""
    history: list[dict]  # Conversación previa
    memory: dict         # Memoria a largo plazo
    preferences: dict    # Preferencias del usuario

    def add_interaction(self, role: str, content: str):
        """Agrega una interacción al historial."""
        self.history.append({"role": role, "content": content})
```

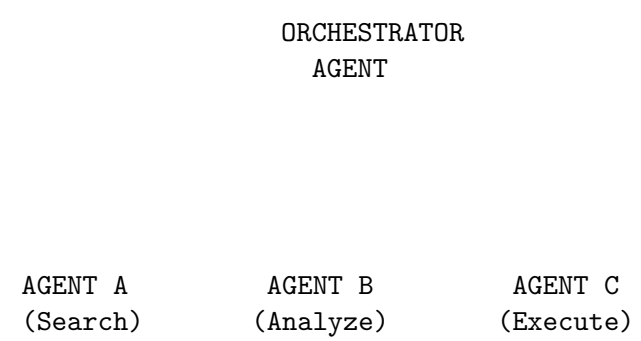
```python
class BaseAgent(ABC):
    """Clase base para agentes."""

    def __init__(self, name: str):
        self.name = name
        self.state = AgentState.IDLE
        self.context = AgentContext(
            history=[],
            memory={},
            preferences={}
        )

    @abstractmethod
    def think(self, input_data: str) -> str:
        """Procesa el input y decide qué hacer."""
        pass

    @abstractmethod
    def act(self, decision: str) -> Any:
        """Ejecuta la decisión y retorna resultado."""
        pass

    def run(self, input_data: str) -> Any:
        """Ejecuta el ciclo completo del agente."""
        self.state = AgentState.THINKING
        decision = self.think(input_data)

        self.state = AgentState.ACTING
        result = self.act(decision)

        self.context.add_interaction("user", input_data)
        self.context.add_interaction("assistant", str(result))

        self.state = AgentState.IDLE
        return result

    def get_memory(self, key: str) -> Optional[Any]:
        """Recupera información de la memoria."""
        return self.context.memory.get(key)

    def set_memory(self, key: str, value: Any):
        """Almacena información en la memoria."""
        self.context.memory[key] = value

# Ejemplo de agente concreto
class InventoryAgent(BaseAgent):
    """Agente para gestionar inventario."""
```

```
    def think(self, input_data: str) -> str:
        """Decide qué acción tomar basándose en el input."""
        input_lower = input_data.lower()

        if "agregar" in input_lower or "crear" in input_lower:
            return "CREATE"
        elif "buscar" in input_lower or "consultar" in input_lower:
            return "SEARCH"
        elif "actualizar" in input_lower or "modificar" in input_lower:
            return "UPDATE"
        elif "eliminar" in input_lower or "borrar" in input_lower:
            return "DELETE"
        else:
            return "UNKNOWN"

    def act(self, decision: str) -> Any:
        """Ejecuta la acción decideda."""
        # Aquí iría la lógica real del inventario
        return {"action": decision, "status": "executed"}

# Uso del agente
agent = InventoryAgent("InventoryManager")
result = agent.run("Agregar nuevo producto: Mark III Armor")
print(f"Resultado: {result}")
```

37.4.5.3 Patrón Sub-agente: El Equipo de Tony Stark

Tony no hace todo solo. Tiene a JARVIS, a Happy, a Pepper. Cada uno es un experto en su área.

El patrón **sub-agente** divide el trabajo entre agentes especializados:



```python
from typing import Protocol
import asyncio

class SubAgent(Protocol):
    """Protocolo que deben implementar todos los sub-agentes."""
    async def execute(self, task: dict) -> dict:
        """Ejecuta una tarea específica."""
        ...

class OrchestratorAgent:
    """Agente orquestador que coordina sub-agentes."""

    def __init__(self):
        self.sub_agents: dict[str, SubAgent] = {}

    def register_agent(self, name: str, agent: SubAgent):
        """Registra un sub-agente."""
        self.sub_agents[name] = agent

    async def delegate_task(self, task: dict) -> dict:
        """Delega la tarea al agente apropiado."""
        task_type = task.get("type", "unknown")

        if task_type not in self.sub_agents:
            return {"error": f"No hay agente para tareas de tipo: {task_type}"}

        agent = self.sub_agents[task_type]
        return await agent.execute(task)

    async def run_pipeline(self, tasks: list[dict]) -> list[dict]:
        """Ejecuta una serie de tareas en pipeline."""
        results = []
        for task in tasks:
            result = await self.delegate_task(task)
            results.append(result)
        return results

# Ejemplo de sub-agentes
class SearchAgent:
    async def execute(self, task: dict) -> dict:
        query = task.get("query", "")
        # Lógica de búsqueda...
        return {"results": f"Resultados para: {query}", "count": 5}

class AnalysisAgent:
    async def execute(self, task: dict) -> dict:
        data = task.get("data", "")
        # Lógica de análisis...
```

```
        return {"analysis": f"Análisis de: {data}", "sentiment": "positive"}

# Uso
orchestrator = OrchestratorAgent()
orchestrator.register_agent("search", SearchAgent())
orchestrator.register_agent("analysis", AnalysisAgent())

# Ejecutar pipeline
tasks = [
    {"type": "search", "query": "armaduras Mark"},
    {"type": "analysis", "data": "Resultados de armaduras"}
]

async def main():
    results = await orchestrator.run_pipeline(tasks)
    for r in results:
        print(r)

asyncio.run(main())
```

37.4.6 3.5 Patrón Pipeline: Encadenando Agentes

> *"No necesito hacer todo yo mismo. Solo necesito conectar las piezas correctas."*
> — Tony Stark

Un **pipeline** es una cadena de agentes donde la salida de uno se convierte en la entrada del siguiente.

37.4.6.1 Pipeline Secuencial vs Paralelo

```
SECUENCIAL (uno después de otro):

  A        B        C        D

Input    Output A   Output B   Output C   Output D

PARALELO (todos procesan el mismo input):

         A


       B         Combinar resultados
```

C

37.4.6.2 Pipeline de 3 Agentes: El Asistente de Investigación

Imagina un pipeline para investigar un tema:

1. **Researcher**: Busca información relevante
2. **Analyzer**: Analiza y sintetiza los hallazgos
3. **Reporter**: Genera un reporte estructurado

```
import asyncio
from dataclasses import dataclass
from typing import Optional
import json

@dataclass
class PipelineTask:
    """Tarea en el pipeline."""
    task_id: str
    input_data: str
    metadata: dict

@dataclass
class PipelineResult:
    """Resultado de una tarea en el pipeline."""
    task_id: str
    agent_name: str
    output: str
    success: bool
    error: Optional[str] = None

class BaseAgent(ABC):
    """Agente base con soporte para pipelines."""

    def __init__(self, name: str):
        self.name = name

    @abstractmethod
    async def process(self, input_data: str) -> str:
        """Procesa el input y retorna output."""
        pass

    async def execute(self, task: PipelineTask) -> PipelineResult:
        """Ejecuta el agente y envuelve el resultado."""
        try:
            output = await self.process(task.input_data)
            return PipelineResult(
```

```
                task_id=task.task_id,
                agent_name=self.name,
                output=output,
                success=True
            )
        except Exception as e:
            return PipelineResult(
                task_id=task.task_id,
                agent_name=self.name,
                output="",
                success=False,
                error=str(e)
            )

# Agentes del pipeline
class ResearcherAgent(BaseAgent):
    """Agente que busca información."""

    async def process(self, input_data: str) -> str:
        # Simulación de búsqueda
        await asyncio.sleep(0.1)  # Simular trabajo
        return json.dumps({
            "sources": [
                {"title": f"Fuente 1 sobre {input_data}", "relevance": 0.9},
                {"title": f"Fuente 2 sobre {input_data}", "relevance": 0.7},
                {"title": f"Fuente 3 sobre {input_data}", "relevance": 0.5},
            ],
            "query": input_data
        })

class AnalyzerAgent(BaseAgent):
    """Agente que analiza la información."""

    async def process(self, input_data: str) -> str:
        # Parsear el input (output del Researcher)
        data = json.loads(input_data)
        sources = data.get("sources", [])

        # Analizar y sintetizar
        summary = f"Análisis de {len(sources)} fuentes encontradas\n"
        summary += "Hallazgos principales:\n"
        for source in sources[:2]:  # Top 2
            summary += f"- {source['title']} (relevancia: {source['relevance']})\n"

        return json.dumps({
            "summary": summary,
            "key_findings": ["Hallazgo 1", "Hallazgo 2"],
            "confidence": 0.85
```

```
        })

class ReporterAgent(BaseAgent):
    """Agente que genera el reporte final."""

    async def process(self, input_data: str) -> str:
        # Parsear el input (output del Analyzer)
        data = json.loads(input_data)

        # Generar reporte
        report = f"""# Reporte de Investigación

## Resumen
{data.get('summary', '')}

## Hallazgos Clave
"""
        for i, finding in enumerate(data.get("key_findings", []), 1):
            report += f"{i}. {finding}\n"

        report += f"\n## Confianza del Análisis\n{data.get('confidence', 0) * 100}%"

        return report

# Pipeline Orchestrator
class PipelineOrchestrator:
    """Orquestador del pipeline de investigación."""

    def __init__(self):
        self.agents: list[BaseAgent] = []

    def add_agent(self, agent: BaseAgent):
        """Agrega un agente al pipeline."""
        self.agents.append(agent)

    async def execute(self, input_data: str) -> dict:
        """Ejecuta el pipeline completo."""
        results = []
        current_input = input_data
        task_id = "task_001"

        print(f"  Iniciando pipeline con input: '{input_data}'")

        for agent in self.agents:
            print(f"    Ejecutando {agent.name}...")
            task = PipelineTask(
                task_id=task_id,
                input_data=current_input,
```

```
                metadata={}
            )

            result = await agent.execute(task)
            results.append(result)

            if result.success:
                current_input = result.output
                print(f"       Completado")
            else:
                print(f"       Error: {result.error}")
                return {"success": False, "results": results}

        return {
            "success": True,
            "results": results,
            "final_output": current_input
        }

# Uso del pipeline
async def main():
    pipeline = PipelineOrchestrator()
    pipeline.add_agent(ResearcherAgent("Researcher"))
    pipeline.add_agent(AnalyzerAgent("Analyzer"))
    pipeline.add_agent(ReporterAgent("Reporter"))

    result = await pipeline.execute("inteligencia artificial en medicina")

    if result["success"]:
        print("\n" + "="*50)
        print("REPORTE FINAL:")
        print("="*50)
        print(result["final_output"])
    else:
        print("Pipeline falló")

asyncio.run(main())
```

37.4.7 3.6 El Problema de la Memoria: Por qué los Agentes Olvidan

"¿Dónde guardé eso?" — Tu agente, probablemente

37.4.7.1 El Problema Fundamental

Cada vez que ejecutas un agente, este **empieza desde cero**:

```
# Ejecución 1
agent = MiAgente()
result = agent.run("Hola, me llamo Diego")
# El agente knows "Diego" existe

# Ejecución 2 (nueva sesión)
agent2 = MiAgente()
result2 = agent2.run("¿Cómo me llamo?")
#   El agente no tiene idea de quién eres
#    (empezar desde cero)
```

Esto es un problema real en producción:

Escenario	Qué pasa
Chat con usuario	No recuerda preferencias
Agente de tareas	Pierde contexto de tareas previas
Pipeline complejo	No mantiene estado entre pasos
Sistema multi-sesión	Cada usuario es un desconocido

37.4.7.2 ¿Por qué los Agentes Olvidan?

```
                    MEMORIA VOLÁTIL
  (RAM, variables, contexto de ejecución)

    Rápida
    Accesible
    Se pierde al terminar la ejecución
    No persiste entre sesiones




                LA SOLUCIÓN: MEMORIA PERSISTENTE

  - Base de datos
  - Archivos
  - Sistemas de memoria especializada (Engram)

    Persiste entre sesiones
    Consulta rápida
    Historial completo
```

37.4.7.3 Tipos de Memoria en Agentes

Tipo	Duración	Ejemplo	Caso de uso
Working	Segundos	Variables locales	Procesamiento inmediato
Session	Minutos/horas	Contexto HTTP	Conversación activa
Long-term	Indefinido	DB, archivos	Historial completo
Semantic	Estructurado	Vector DB	Búsqueda por significado

37.4.7.4 Patrón: Agente con Memoria

```
from abc import ABC, abstractmethod
from datetime import datetime
import json
from typing import Optional

class MemoryStore(ABC):
    """Interfaz para almacenamiento de memoria."""

    @abstractmethod
    def save(self, key: str, value: str) -> None:
        pass

    @abstractmethod
    def retrieve(self, key: str) -> Optional[str]:
        pass

    @abstractmethod
    def search(self, query: str) -> list[str]:
        pass

class InMemoryStore(MemoryStore):
    """Implementación en memoria (para desarrollo)."""

    def __init__(self):
        self.data: dict[str, str] = {}

    def save(self, key: str, value: str) -> None:
        self.data[key] = value

    def retrieve(self, key: str) -> Optional[str]:
        return self.data.get(key)

    def search(self, query: str) -> list[str]:
        # Búsqueda simple (en producción usarías vector search)
        return [v for k, v in self.data.items() if query.lower() in k.lower()]
```

```python
class AgentWithMemory:
    """Agente que mantiene memoria persistente."""

    def __init__(self, name: str, memory_store: MemoryStore):
        self.name = name
        self.memory = memory_store

    def remember(self, key: str, value: str):
        """Guarda algo en memoria."""
        timestamp = datetime.now().isoformat()
        entry = json.dumps({
            "value": value,
            "timestamp": timestamp,
            "agent": self.name
        })
        self.memory.save(key, entry)
        print(f" Recordando: {key}")

    def recall(self, key: str) -> Optional[dict]:
        """Recupera algo de memoria."""
        entry = self.memory.retrieve(key)
        if entry:
            return json.loads(entry)
        return None

    def search_memory(self, query: str) -> list[str]:
        """Busca en la memoria."""
        return self.memory.search(query)

    async def process(self, input_data: str) -> str:
        """Procesa input considerando la memoria."""
        # 1. Buscar contexto relevante en memoria
        relevant = self.search_memory(input_data)

        # 2. Procesar con ese contexto
        if relevant:
            context = f"Based on memory: {relevant[0]}"
        else:
            context = "No relevant memory found"

        # 3. Recordar la interacción
        self.remember(
            f"interaction_{datetime.now().timestamp()}",
            f"User: {input_data}"
        )

        return f"Processed with context: {context[:50]}..."
```

```
# Uso
memory = InMemoryStore()
agent = AgentWithMemory("TestAgent", memory)

# Primera interacción
asyncio.run(agent.process("Me gusta la armadura Mark III"))

# Segunda interacción - debería recordar contexto
asyncio.run(agent.process("¿Qué me gusta?"))
```

37.4.8 3.7 Testing Avanzado para Agentes

"Si no lo testea, no funciona en producción." — Tony Stark (definitivamente)

37.4.8.1 Tipos de Tests para Sistemas de Agentes

Tipo de Test	Qué Prueba	Ejemplo
Unit Tests	Funciones individuales	¿`think()` retorna la decisión correcta?
Integration Tests	Múltiples componentes	¿El agente funciona con el memory store?
End-to-End	Flujo completo	¿El pipeline produce el resultado esperado?
Property-Based	Propiedades del código	¿La salida siempre es un string válido?
Performance	Rendimiento	¿Responde en <100ms?
Security	Vulnerabilidades	¿Previene inyección de prompts?

37.4.8.2 Coverage de Código

```
# Ejecutar con coverage
# pytest --cov=src --cov-report=html

# Configuración de coverage en pytest.ini
[tool.pytest.ini_options]
addopts = "--cov=src --cov-report=term-missing --cov-fail-under=80"

# Ejemplo de tests con coverage
```

```python
import pytest
from agent import AgenteBase, PipelineOrchestrator

class TestAgenteBasico:
    """Tests unitarios del agente."""

    def test_inicializacion(self):
        agent = AgenteBase("Test")
        assert agent.name == "Test"
        assert agent.state == "idle"

    def test_decision_correcta(self):
        agent = AgenteBase("Test")
        decision = agent.think("agregar producto")
        assert decision == "CREATE"

class TestPipeline:
    """Tests de integración del pipeline."""

    @pytest.mark.asyncio
    async def test_pipeline_ejecuta_todos_agentes(self):
        pipeline = PipelineOrchestrator()
        pipeline.add_agent(MockAgent("A"))
        pipeline.add_agent(MockAgent("B"))

        result = await pipeline.execute("test input")

        assert result["success"]
        assert len(result["results"]) == 2

    @pytest.mark.asyncio
    async def test_pipeline_falla_en_error(self):
        pipeline = PipelineOrchestrator()
        pipeline.add_agent(FailingAgent())  # Always fails

        result = await pipeline.execute("test")

        assert not result["success"]

# Coverage mínimo requerido
# - unit tests: 80%
# - integration: 70%
# - e2e: 60%
```

37.4.8.3 TDD con Agentes: Ciclo Red-Green-Refactor

RED
Escribe un
test que
falla

GREEN
Escribe
código que
pase el test

REFACTOR
Mejora el
código sin
romper tests

Ejemplo: TDD para un Agente de Inventario

```
#  Paso 1: Escribe el test que falla (RED)
class TestInventoryAgent:
    def test_agente_crea_producto_correctamente(self):
        # Arrange
        agent = InventoryAgent()

        # Act
        result = agent.run("Crear producto: Mark III, cantidad: 5")

        # Assert
        assert result["status"] == "created"
        assert result["product"]["name"] == "Mark III"
        assert result["product"]["quantity"] == 5

#  Test falla: "InventoryAgent" no existe aún

#  Paso 2: Escribe código mínimo para pasar (GREEN)
class InventoryAgent:
    def run(self, input_data: str) -> dict:
        # Parse simple
        if "Crear producto:" in input_data:
            return {
                "status": "created",
                "product": {
```

```
                    "name": "Mark III",
                    "quantity": 5
                }
            }
        return {"error": "Unknown command"}

#  Test pasa

#  Paso 3: Refactoriza (REFACTOR)
# Mejorar el código mientras los tests pasan
class InventoryAgent:
    """Agente para gestión de inventario."""

    def __init__(self):
        self.inventory = {}

    def _parse_create_command(self, text: str) -> dict:
        """Parsea comandos de creación."""
        import re
        match = re.search(r'producto: (\w+).*cantidad: (\d+)', text)
        if match:
            return {"name": match.group(1), "quantity": int(match.group(2))}
        return None

    def run(self, input_data: str) -> dict:
        """Ejecuta el comando del usuario."""
        if "Crear producto:" in input_data:
            product = self._parse_create_command(input_data)
            if product:
                self.inventory[product["name"]] = product
                return {"status": "created", "product": product}
        return {"error": "Unknown command"}

#  Tests siguen pasando, código más robusto
```

37.5 Laboratorio 3: El Primer Vuelo

37.5.1 Objetivo

Tomar tu sistema del Mark I (Nivel 2) y hacerlo robusto para producción:

1. Encontrar y corregir bugs
2. Agregar tests para edge cases
3. Refactorizar código problemático

4. Optimizar rendimiento
5. **Agregar memoria persistente** (preview del Nivel 4)

37.5.2 Ejercicio 1: El Bug del Mark III

37.5.2.1 Escenario

Tu sistema de inventario del Mark I tiene un bug crítico:

```
# El bug: cuando se agrega un producto con cantidad 0
# el sistema lo muestra como "out of stock" pero no
# permite restockearlo

inventario.crear_producto("Mark III", "armadura", 0)
# Esperado: Producto creado, stock 0, permite restock
# Real: Error o producto invisible
```

37.5.2.2 Tu Tarea

1. **Reproduce** el bug
2. **Identifica** la causa raíz
3. **Escribe un test** que falle con el bug
4. **Corrige** el código
5. **Verifica** que el test pase

37.5.2.3 Checklist

☐ Bug reproducido localmente
☐ Causa raíz identificada
☐ Test escrito (failing)
☐ Código corregido
☐ Test passing
☐ Commit realizado

37.5.3 Ejercicio 2: Edge Cases para Inventario

37.5.3.1 Tu Tarea

Identifica y escribe tests para al menos 5 edge cases de tu sistema:

```
# Template para tu test
def test_edge_case_nombre_descriptivo(self):
    """Edge case: descripción clara del caso extremo."""
    # Arrange
    # Tu setup aquí

    # Act
    # Tu acción aquí

    # Assert
    # Tu verificación aquí
```

37.5.3.2 Sugerencias de Edge Cases

#	Edge Case	¿Qué podría fallar?
1	Producto con nombre de 1000 caracteres	Buffer overflow, UI rota
2	Precio = 0	División por cero en descuentos
3	Stock máximo (int overflow)	Error de cálculo
4	Concurrent access	Race conditions
5	Datos corruptos JSON	Parsing error

37.5.4 Ejercicio 3: Refactoring Challenge

37.5.4.1 Tu Tarea

Refactoriza esta función problemática:

```
# Código a refactorizar
def handle_request(data, user, action, options=None, callback=None, **kwargs):
    """Maneja request del sistema."""
    if user is not None:
        if user.get('role') == 'admin':
            if action == 'create':
                if data.get('name'):
                    if data.get('quantity', 0) >= 0:
                        # Lógica de creación aquí...
                        result = {'status': 'created', 'id': 123}
                        if callback:
                            callback(result)
                        return result
```

```
                    else:
                        return {'error': 'Invalid quantity'}
                else:
                    return {'error': 'Name required'}
            elif action == 'delete':
                # Lógica de eliminación...
                pass
            # ... más elifs
        elif user.get('role') == 'user':
            # Lógica para usuario normal...
            pass
    return {'error': 'Unauthorized'}
```

37.5.4.2 Requisitos

- ☐ Máximo 3 niveles de indentación
- ☐ Cada función hace una sola cosa
- ☐ Nombres descriptivos
- ☐ Tests pasan después del refactor

37.5.5 Ejercicio 4: Pipeline de Agentes

37.5.5.1 Tu Tarea

Crea un pipeline de 3 agentes:

1. **Input Agent**: Recibe input del usuario
2. **Processing Agent**: Procesa y transforma
3. **Output Agent**: Formatea la respuesta final

```
# Template para tu pipeline
class PipelineDemo:
    def __init__(self):
        self.agents = []

    def add_agent(self, agent):
        self.agents.append(agent)

    async def run(self, input_data):
        # Tu implementación aquí
        pass

# Test que debe pasar
async def test_pipeline():
```

```
    pipeline = PipelineDemo()
    pipeline.add_agent(InputAgent())
    pipeline.add_agent(ProcessingAgent())
    pipeline.add_agent(OutputAgent())

    result = await pipeline.run("input de prueba")
    assert "processed" in result.lower()
    assert "output" in result.lower()
```

37.5.5.2 Requisitos

- ☐ Pipeline secuencal funciona
- ☐ Cada agente hace una cosa
- ☐ Tests unitarios de cada agente
- ☐ Test de integración del pipeline
- ☐ Documentación del flujo

37.5.6 Ejercicio 5: Preview de Memoria Persistente

37.5.6.1 Tu Tarea

Implementa una versión simple de memoria persistente:

```
class SimpleMemory:
    """Sistema de memoria básico para agentes."""

    def __init__(self):
        self.store = {}

    def save(self, key: str, value: str):
        """Guarda en memoria."""
        # Tu implementación
        pass

    def recall(self, key: str) -> str:
        """Recupera de memoria."""
        # Tu implementación
        pass

    def search(self, query: str) -> list[str]:
        """Busca en memoria."""
        # Tu implementación
        pass
```

```python
# Tests que deben pasar
def test_memory_save_and_recall():
    mem = SimpleMemory()
    mem.save("user_name", "Diego")
    assert mem.recall("user_name") == "Diego"

def test_memory_search():
    mem = SimpleMemory()
    mem.save("interaction_1", "Me gusta Mark III")
    mem.save("interaction_2", "Mi color favorito es rojo")
    results = mem.search("Mark")
    assert len(results) == 1
    assert "Mark III" in results[0]
```

37.5.6.2 Hint

- Usa un diccionario simple para el store
- Para search, usa contains simple (`query in value`)
- Piensa: ¿qué pasa si la key no existe?

37.5.6.3 Requisitos

☐ save() funciona correctamente
☐ recall() retorna lo guardado
☐ search() encuentra contenido relevante
☐ Tests unitarios pasando
☐ Demo integrando con un agente simple

37.6 Logro Desbloqueado: “First Flight”

37.6.1 Requisitos para Desbloquear

☐ Bug del Mark III corregido
☐ 5+ edge cases con tests
☐ Código refactorizado (3 indentación)
☐ Pipeline de 3 agentes funcionando
☐ Sistema de memoria básico implementado
☐ Todos los tests pasando

37.6.2 Recompensa

- **+300 XP** (incrementado por el contenido extra)
- **Logro "First Flight"**
- **Acceso al Nivel 4: JARVIS Avanzado (Engram + MCP)**

37.6.3 Siguiente Nivel

> *"Ahora viene la parte divertida."* — Tony Stark

El Nivel 4 introduce **memoria persistente real** con Engram y **contexto de protocolo** con MCP. Lo que acabas de implementar (SimpleMemory) es la base de lo que harás con herramientas profesionales.

Con tu Mark III perfeccionado, estás listo para ampliar tu inteligencia: → **Nivel 4: JARVIS Avanzado (Engram + MCP)**

37.7 Recursos Adicionales

37.7.1 Agentes y Sub-agentes

- LangChain Agents
- AutoGPT Architecture
- Agent Patterns - DeepLearning.AI

37.7.2 Pipeline de Agentes

- LangChain Chains
- Building AI Agents - AWS

37.7.3 Testing para IA

- pytest-asyncio
- Property-Based Testing
- AI Testing Strategies

37.7.4 Memoria Persistente

- Engram - Persistent Memory
- Vector Databases for AI
- LangChain Memory

37.7.5 Debugging

- Python Debugging with pdb
- ipdb - IPython Debugger
- VS Code Python Debugging

37.7.6 Testing

- pytest Documentation
- Real Python - Testing
- Edge Case Testing Patterns

37.7.7 Refactoring

- Refactoring Guru
- Martin Fowler - Refactoring
- Clean Code Principles

38 Lab 3: El Mark III - Tu Primera Armadura Real

39 Lab 3: El Mark III - Tu Primera Armadura Real

39.1 De Prototipo a Producto Profesional

39.2 La Situación

"Ahora viene la parte divertida." — Tony Stark

Tony Stark crea el Mark III. Primera armadura roja y dorada. Primera armadura que sale del laboratorio. Primera armadura que **no es un prototipo**. Es el primer producto real.

El Mark III tiene todo: - **Diseño profesional**: Rojo y dorado, no chatarra - **Sistemas completos**: Propulsión, armamento, sensores - **Testing exhaustivo**: Probado en condiciones reales - **Mantenible**: Stark puede repararlo y mejorarlo

En este lab, pasarás de "Mark I funcional" a "Mark III profesional": arquitectura escalable, tests completos, documentación, y código que otros pueden usar.

39.3 Timeline de la Misión

Paso	Descripción	Tiempo	Completado
1	Diseñar arquitectura profesional	25 min	
2	Implementar tests completos	30 min	
3	Crear documentación técnica	20 min	
4	Validar calidad del código	15 min	
Total		**90 min**	

39.4 Objetivo del Lab

Transformar tu prototipo en un producto profesional. Al finalizar, tendrás:

1. Arquitectura escalable y mantenible
2. Suite de tests completa (>80% coverage)
3. Documentación técnica profesional
4. Código que sigue mejores prácticas

39.5 Regla del Stark Protocol

> **"No es un prototipo. Es una armadura."** — Tony Stark

REGLA ORO: Calidad sobre velocidad. Un Mark III bien hecho vale más que diez Mark I.

39.6 Ejercicios: Piloto y Copiloto

En este ejercicio practicarás mantener el control como piloto.

39.6.0.1 Timeline de Verificación

Paso	Descripción	Tiempo	Completado
1	Reformular 3 requests	20 min	
2	Análisis de decisiones	15 min	
3	Escenario de decisión	15 min	
4	Reflexión final	10 min	
Total		**60 min**	

39.7 Objetivo

Reformular requests para mantener el control como piloto.

39.8 Importante: No Copiar, Reformular

Los ejercicios te hacen reformular, no ejecutar ciegamente.

39.9 1. El Concepto Central

Stark nunca delega decisiones. Delega ejecución.

Stark decide qué construir. J.A.R.V.I.S. sugiere cómo. Juntos eligen.

Tú eres el piloto.

39.10 2. Ejercicio 1: Sugerir vs Decidir

Observa estas dos interacciones:

Interacción A:

```
Usuario: "Instala PostgreSQL en mi servidor"
IA: "Ejecutando: sudo apt-get install postgresql"
```

Interacción B:

```
Usuario: "Quiero instalar una base de datos"
IA: "Tengo tres opciones:
1. PostgreSQL
2. MongoDB
3. SQLite
¿Cuál se ajusta mejor a tu caso?"
```

Reflexiona:

```
¿En cuál interacción el usuario tiene más control? ___
¿En cuál aprende más? ___
¿Cuál es más safe? ___
```

39.11 3. Tu Turno: Reformula

Instrucciones: Lee cada request vago. Reescríbelo para pedir opciones.

Request 1:

```
Original: "Optimiza mi base de datos"
```

Tu versión (complétala):

```
"Quiero optimizar mi base de datos.
¿Qué opciones tengo? Dame pros y contras de:
- ___
- ___
- ___
Mis constraints son: ___
"
```

Request 2:

```
Original: "Escríbeme tests para mi API"
```

Tu versión (complétala):

```
"Quiero agregar tests a mi API.
¿Qué tipos de tests recommendarías?
Para cada tipo, dame:
- Propósito
- Herramienta sugerida
- Ejemplo básico
Mi API usa: ___
"
```

Request 3:

```
Original: "Configura CI/CD para mi proyecto"
```

Tu versión (complétala):

```
"Quiero configurar CI/CD.
¿Qué opciones hay para un proyecto ___
con stack ___
Para cada opción, quiero saber:
- Pros
- Contras
- Complejidad de setup
Mi prioridad es: ___
"
```

39.12 4. Ejercicio 2: El Análisis de Decisiones

Escenario:

Estás construyendo una API. Tienes que elegir entre:

- **Opción A:** REST
- **Opción B:** GraphQL
- **Opción C:** gRPC

Tu tarea:

1. Piensa: ¿Qué criterios usarías para elegir?
2. Escribe los criterios en orden de prioridad
3. Para cada opción, evalúa según tus criterios
4. Toma una decisión

Registro:

```
Mis criterios (en orden):
1. ___
2. ___
3. ___

Evaluación:
- Opción A (REST): ___
- Opción B (GraphQL): ___
- Opción C (gRPC): ___

Mi decisión: ___
```

Pregunta:

```
¿Tomaste la decisión basándote en datos o en preferencia personal? ___
¿Cambiaría J.A.R.V.I.S. algo de tu análisis? ___
```

39.13 5. Ejercicio 3: Cuando Decidir es Duro

Escenario Real:

Stark tenía que elegir entre Blue-green deployment y Canary release.

J.A.R.V.I.S. le dio los pros y contras. Stark eligió Blue-green.

Tu escenario:

Estás deployando una actualización de seguridad crítica.

Opciones:

- **Opción A:** Deploy ahora, con rollback rápido
- **Opción B:** Deploy gradual (10% → 50% → 100%)
- **Opción C:** No deploy hasta tener más testing

Tu análisis (complétalo):

```
¿Qué preguntas harías antes de decidir?
1. ___
2. ___
3. ___

¿Qué elegirías y por qué? ___

¿Pedirías ayuda a la IA para decidir o ya tienes claro qué hacer? ___
```

39.14 Reflexión Final

```
¿Sientes que tienes más control después de este lab?
¿O sientes que delegas más responsabilidad a la IA?

En una escala de 1-10:
- Mi nivel de control actual: ___
- Mi nivel de control deseado: ___

¿Qué necesito hacer para cerrar la brecha? ___
```

39.15 Verificación

Checklist:

- ☐ 3 requests reformulados
- ☐ Análisis de decisiones completado
- ☐ Decisión justificada
- ☐ Reflexiones completadas

39.16 Entregable

Archivo: `piloto_y_copiloto.md`

Incluir: - Los 3 requests reformulados - Tu análisis de decisión - Tu reflexión final

---

39.17 Recursos

- Human-AI Decision Making
- AI as a Tool, Not a Replacement

40 Boss Fight 3: La Torre Stark

41 Boss Fight 3: La Torre Stark

41.1 Prueba Final del Nivel 3

41.2 La Situación

> *"La Torre Stark. Nuevo centro de operaciones. Todo lo que necesito, todo en un lugar."* — Tony Stark

Tony construye la Torre Stark. No es solo un edificio, es un **ecosistema**. Cada piso tiene un propósito, cada sistema se comunica, todo está diseñado para **eficiencia máxima**.

Tu misión: Crear una arquitectura que sea como la Torre Stark: organizada, eficiente, mantenible.

41.3 Misión: Sistema de Gestión de Proyectos "Stark Industries"

41.3.1 Requisitos de Arquitectura

```
stark-projects/
  src/
     domain/          # Lógica de negocio
        entities/    # Entidades del dominio
        services/    # Servicios de dominio
        interfaces/  # Interfaces/ports
     application/     # Casos de uso
        use_cases/   # Lógica de aplicación
        dto/         # Data Transfer Objects
     infrastructure/  # Implementación
        repositories/ # Repositorios concretos
        api/         # Controladores API
        persistence/ # Base de datos
     shared/          # Código compartido
  tests/
```

```
    unit/            # Tests unitarios
    integration/     # Tests de integración
    e2e/             # Tests end-to-end
  docs/              # Documentación
  scripts/           # Scripts de utilidad
```

41.3.2 Entidades del Dominio

```python
# domain/entities/project.py
from dataclasses import dataclass
from datetime import datetime
from typing import Optional

@dataclass
class Project:
    """Entidad Project del dominio"""
    id: str
    name: str
    description: str
    status: str  # "active", "completed", "archived"
    created_at: datetime
    updated_at: Optional[datetime] = None

    def activate(self):
        """Activa el proyecto"""
        self.status = "active"
        self.updated_at = datetime.now()

    def complete(self):
        """Marca proyecto como completado"""
        self.status = "completed"
        self.updated_at = datetime.now()
```

41.3.3 Casos de Uso

```python
# application/use_cases/create_project.py
from dataclasses import dataclass
from src.domain.entities.project import Project
from src.domain.interfaces.project_repository import ProjectRepository

@dataclass
class CreateProjectRequest:
    """DTO para crear proyecto"""
    name: str
```

```
    description: str

@dataclass
class CreateProjectResponse:
    """DTO de respuesta"""
    project_id: str
    name: str
    status: str

class CreateProjectUseCase:
    """Caso de uso: Crear proyecto"""

    def __init__(self, repository: ProjectRepository):
        self.repository = repository

    def execute(self, request: CreateProjectRequest) -> CreateProjectResponse:
        """Ejecuta el caso de uso"""
        # Validar datos
        if not request.name.strip():
            raise ValueError("Nombre no puede estar vacío")

        # Crear entidad
        project = Project(
            id=self.repository.generate_id(),
            name=request.name,
            description=request.description,
            status="active",
            created_at=datetime.now()
        )

        # Persistir
        self.repository.save(project)

        # Retornar respuesta
        return CreateProjectResponse(
            project_id=project.id,
            name=project.name,
            status=project.status
        )
```

41.4 Evaluación

41.4.1 Métricas de Calidad

Métrica	Objetivo	Tu Resultado
Acoplamiento	Bajo (dependencias invertidas)	
Cohesión	Alta (cada clase hace una cosa)	
Testeabilidad	Alta (fácil de testear)	
Mantenibilidad	Alta (cambios fáciles)	

41.4.2 Checklist de Arquitectura

- ☐ Separación clara por capas
- ☐ Dependencias invertidas (domain no depende de infra)
- ☐ Tests en cada capa
- ☐ Documentación de arquitectura
- ☐ Ejemplos de uso

41.5 Logro Desbloqueado: "Architect"

41.5.1 Requisitos

- ☐ Arquitectura limpia implementada
- ☐ Tests en cada capa
- ☐ Documentación de arquitectura
- ☐ Código mantenible

41.5.2 Recompensa

- **+250 XP**
- **Logro "Architect"**
- **Acceso a Nivel 4**

41.5.3 Siguiente Nivel

> *"J.A.R.V.I.S., ¿puedes ayudarme con esto?"* — Tony Stark

→ Nivel 4: J.A.R.V.I.S. Avanzado

Part IV

Nivel 4: El Jarvis Primordial

42 Nivel 4: JARVIS Avanzado

43 Nivel 4: JARVIS Avanzado

43.1 Inteligencia Ampliada: De Agente a Sistema Cognitivo

43.2 La Escena de Iron Man 2 (2010)

> *"Todo funciona a la perfección. No veo la razón para cambiar nada."* — Tony Stark

JARVIS ha evolucionado. Ya no es solo un asistente. Es un **sistema cognitivo completo**: - **Memoria perfecta**: Recuerda cada conversación, cada decisión - **Conectividad universal**: Se conecta a cualquier sistema, base de datos, API - **Habilidades modulares**: Puede aprender nuevas capacidades bajo demanda - **Autonomía controlada**: Toma decisiones sin saturar a Tony

En Iron Man 2, JARVIS: 1. **Recuerda** que Tony le pidió revisar los planes de Stark Expo hace 3 meses 2. **Conecta** con los sistemas de defensa de la mansión 3. **Ejecuta** funciones especializadas (análisis de datos, simulaciones) 4. **Mantiene** el contexto entre múltiples sesiones de trabajo

En este nivel, convertirás tus agentes en "JARVIS Avanzados": con memoria persistente (Engram), conectividad universal (MCP), y habilidades modulares (Skills).

43.3 Objetivos de Aprendizaje

Al completar este nivel, serás capaz de:

1. **Instalar y usar Engram** para memoria persistente entre sesiones
2. **Implementar MCP** para conectar agentes con herramientas externas
3. **Crear y gestionar Skills** modulares para tus agentes
4. **Diseñar sistemas cognitivos** que aprenden y recuerdan
5. **Integrar** todo en un stack completo de desarrollo

43.4 Conceptos Técnicos

43.4.1 4.1 Engram: La Memoria Perfecta de JARVIS

43.4.1.1 El Problema de la Amnesia

Los agentes de IA tienen **amnesia entre sesiones**:

```
Sesión 1: "Necesito una función para calcular energía"
Sesión 2: "¿De qué energía hablaste ayer?" →  "No recuerdo"
```

Engram soluciona esto con **memoria persistente**.

43.4.1.2 Analogía: Engram vs Cerebro Humano

Cerebro Humano	Engram (Go binary)
Memoria a corto plazo	Ventana de contexto
Memoria a largo plazo	SQLite + FTS5
Asociaciones	Búsqueda BM25
Aprendizaje	Guardar decisiones
Olvido	Podemos evitarlo

43.4.1.3 Instalación de Engram

```
# Instalar Engram (binario Go)
# Opción 1: Script de instalación (recomendado)
curl -fsSL https://raw.githubusercontent.com/Gentleman-Programming/gentle-ai/main/scripts

# Opción 2: Descarga manual
# Ir a https://github.com/Gentleman-Programming/gentle-ai/releases
# Descargar el binario para tu OS

# Verificar instalación
engram --version

# Inicializar base de datos
engram init

# Ver estado
engram status
```

43.4.1.4 Cómo Funciona Engram Internamente

```
// Simplificación del funcionamiento interno
type Engram struct {
    db       *sql.DB      // SQLite database
    indexer  *fts5.FTS5   // Full-text search
    bm25     *bm25.BM25   // Ranking algorithm
}

// Guardar un "engram" (unidad de memoria)
func (e *Engram) Guardar(tipo, contenido string, metadatos map[string]interface{}) {
    // 1. Guardar en SQLite
    query := `INSERT INTO engrams (tipo, contenido, metadatos, timestamp)
              VALUES (?, ?, ?, datetime('now'))`

    // 2. Indexar para búsqueda
    e.indexer.Index(contenido)

    // 3. Actualizar BM25
    e.bm25.AddDocument(contenido)
}

// Buscar engramas
func (e *Engram) Buscar(query string, limit int) []Engram {
    // 1. Búsqueda FTS5
    results := e.indexer.Search(query)

    // 2. Ranking con BM25
    ranked := e.bm25.Rank(results)

    // 3. Retornar top results
    return ranked[:limit]
}
```

43.4.1.5 Tipos de Engramas

```
{
  "decision": {
    "tipo": "decision",
    "contenido": "Elegí usar PostgreSQL sobre MySQL por mejor soporte JSON",
    "contexto": "Proyecto Iron Man Evolution, Nivel 4",
    "timestamp": "2026-03-22T10:30:00Z"
  },
  "bugfix": {
    "tipo": "bugfix",
```

```
    "contenido": "Error en cálculo de energía: faltaba normalizar a 0-100",
    "solución": "Añadí Math.max(0, Math.min(100, energia))",
    "archivos": ["src/reactor.py", "tests/test_reactor.py"]
  },
  "preference": {
    tipo": "preference",
    "contenido": "Siempre usar type hints en Python",
    "scope": "global",
    "evidencia": "Error encontrado 3 veces por falta de tipos"
  }
}
```

43.4.2 4.2 MCP: El Protocolo Universal de Conexión

43.4.2.1 ¿Qué es MCP (Model Context Protocol)?

MCP es el **USB-C de la IA**: un estándar para conectar agentes con **cualquier herramienta**. **TODAS** las herramientas de IA modernas soportan MCP.

43.4.2.2 Analogía: MCP vs Adaptadores

Sin MCP	Con MCP
Cada herramienta tiene su integración	Un protocolo estándar
10 herramientas = 10 integraciones	10 herramientas = 1 adaptador
Difícil mantener	Fácil de escalar
Como tener 10 enchufes diferentes	Como tener USB-C universal

43.4.2.3 MCP en Diferentes Herramientas

Todas las herramientas que estás aprendiendo soportan MCP:

```
# Claude Code + MCP
claude --mcp-config mcp-config.json

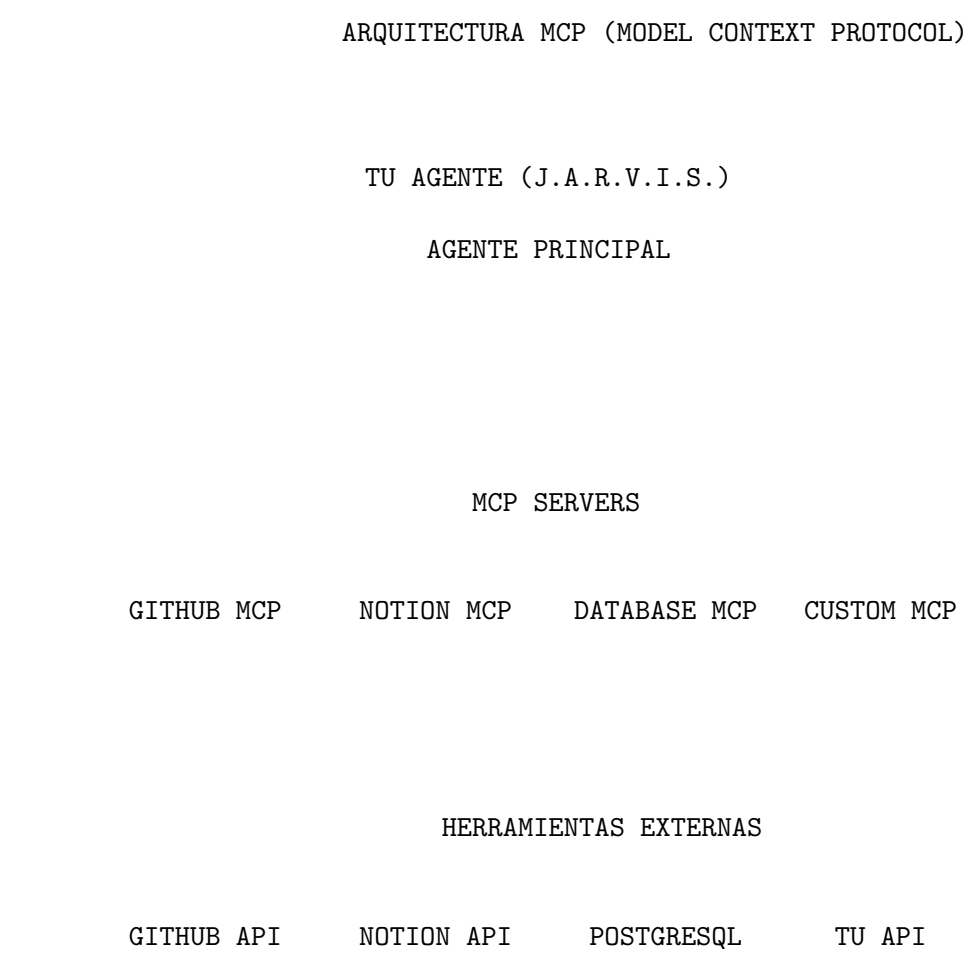
# OpenCode + MCP
opencode --mcp mcp-config.json

# Gemini CLI + MCP
gemini --mcp-config mcp-config.json

# KiloCode + MCP (configuración en VS Code)
# .vscode/settings.json
{
  "kilo-code.mcpServers": {
```

```
    "github": { ... }
  }
}
```

43.4.2.4 Arquitectura MCP



FLUJO:

1. Tu agente (J.A.R.V.I.S.) solicita una acción
2. El agente envía la solicitud al MCP Server correspondiente
3. El MCP Server traduce y ejecuta la acción en la Herramienta Externa
4. El resultado regresa al agente a través del MCP Server

43.4.2.5 Instalar MCP Server para GitHub

```
# Usar Gentle AI Stack para instalar MCPs
curl -fsSL https://raw.githubusercontent.com/Gentleman-Programming/gentle-ai/main/scripts

# Luego instalar GitHub MCP
gentle-ai install-mcp github

# Configurar token
export GITHUB_TOKEN="ghp_tu_token_aqui"

# Verificar
gentle-ai list-mcp
```

43.4.2.6 Configurar MCP en tu proyecto

Crear `mcp-config.json`:

```
{
  "mcpServers": {
    "github": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-github"],
      "env": {
        "GITHUB_TOKEN": "${GITHUB_TOKEN}"
      }
    },
    "filesystem": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-filesystem", "~/iron-man-project"],
      "env": {}
    },
    "database": {
      "command": "python",
      "args": ["-m", "mcp_server_database"],
      "env": {
        "DATABASE_URL": "${DATABASE_URL}"
      }
    }
  }
}
```

43.4.2.7 Instalar MCP Server para GitHub

```
# Usar Gentle AI Stack para instalar MCPs
curl -fsSL https://raw.githubusercontent.com/Gentleman-Programming/gentle-ai/main/scripts

# Luego instalar GitHub MCP
gentle-ai install-mcp github

# Configurar token
export GITHUB_TOKEN="ghp_tu_token_aqui"

# Verificar
gentle-ai list-mcp
```

43.4.2.8 Configurar MCP en tu proyecto

Crear `mcp-config.json`:

```
{
  "mcpServers": {
    "github": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-github"],
      "env": {
        "GITHUB_TOKEN": "${GITHUB_TOKEN}"
      }
    },
    "filesystem": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-filesystem", "~/iron-man-project"],
      "env": {}
    },
    "database": {
      "command": "python",
      "args": ["-m", "mcp_server_database"],
      "env": {
        "DATABASE_URL": "${DATABASE_URL}"
      }
    }
  }
}
```

43.4.3 4.3 Skills: Habilidades Modulares de JARVIS

43.4.3.1 ¿Qué es un Skill?

Un **Skill** es un **módulo de conocimiento especializado** que el agente carga bajo demanda.

43.4.3.2 Analogía: Skills vs Cerebro

Cerebro	Skills
Todo en un solo lugar	Módulos separados
Saturación por exceso de info	Carga selectiva
Difícil actualizar	Fácil de actualizar
Como cargar Windows completo	Como cargar solo la app que necesitas

43.4.3.3 Estructura de un Skill

```
skills/
  clean-code/
      SKILL.md          # Instrucciones principales
      examples/         # Ejemplos de código
      scripts/          # Scripts auxiliares
  testing/
      SKILL.md
      pytest/
      jest/
  security/
      SKILL.md
      OWASP/
      scripts/
```

43.4.3.4 Ejemplo de Skill: clean-code

skills/clean-code/SKILL.md:

````
# Skill: Clean Code

## Trigger
Cuando el código necesita refactorización, nombres ambiguos, o funciones largas.

## Reglas Principales
1. **Nombres revelan intención** - Variables deben explicar qué son
2. **Funciones pequeñas** - Máximo 20 líneas, una responsabilidad
3. **Comentarios explican POR QUÉ** - No el QUÉ
4. **DRY** - No repitas código
5. **Single Responsibility** - Una función hace una cosa bien

## Patrones
### Mal nombre vs Buen nombre
```python
# Mal
def calc(a, b, t):
````
````

```
    if t == 1:
        return a * b * 0.1
    else:
        return a * b * 0.2

# Buen
def calcular_precio_con_descuento(
    precio_base: float,
    cantidad: int,
    tipo_cliente: Literal["regular", "premium"]
) -> float:
    """Calcula precio final con descuento según tipo de cliente."""
    descuento = 0.1 if tipo_cliente == "regular" else 0.2
    return precio_base * cantidad * (1 - descuento)
```

43.4.4 Longitud de función

```
# Mal: Función larga y hace muchas cosas
def procesar_usuario(datos):
    # Validar datos (20 líneas)
    # Guardar en BD (15 líneas)
    # Enviar email (10 líneas)
    # Loggear (5 líneas)
    pass

# Buen: Funciones pequeñas
def procesar_usuario(datos: dict) -> bool:
    """Procesa datos de usuario completo."""
    if not validar_datos_usuario(datos):
        return False

    usuario_id = guardar_usuario_en_bd(datos)
    enviar_email_bienvenida(usuario_id)
    log_proceso_usuario(usuario_id)

    return True

def validar_datos_usuario(datos: dict) -> bool:
    """Valida que todos los campos requeridos estén presentes."""
    campos_requeridos = ["nombre", "email", "password"]
    return all(campo in datos for campo in campos_requeridos)
```

43.5 Checklist de Revisión

☐ ¿Los nombres son descriptivos?

☐ ¿Las funciones tienen <20 líneas?
☐ ¿Hay comentarios que expliquen POR QUÉ?
☐ ¿Se repite código (DRY)?
☐ ¿Cada función hace una sola cosa?

4.4 Registro de Skills (Skill Registry)

Inventario de Habilidades

Como Tony tiene un inventario de armas para cada situación, el Skill Registry es tu inven

```json
{
  "skill-registry": {
    "version": "1.0",
    "skills": {
      "clean-code": {
        "path": "skills/clean-code/SKILL.md",
        "version": "1.0",
        "trigger": "refactorización, nombres ambiguos",
        "priority": 1
      },
      "testing": {
        "path": "skills/testing/SKILL.md",
        "version": "2.1",
        "trigger": "tests, TDD, cobertura",
        "priority": 2
      },
      "security": {
        "path": "skills/security/SKILL.md",
        "version": "1.2",
        "trigger": "seguridad, OWASP, autenticación",
        "priority": 3
      }
    }
  }
}
```

Campos clave:

- **path**: Ruta al archivo SKILL.md
- **version**: Versión de la habilidad
- **trigger**: Palabras que activan esta habilidad
- **priority**: Orden de carga (1 = primero)

43.5.1 4.5 Aislamiento Multi-Agente con Git Worktrees

> *"Cada agente necesita su propio espacio de trabajo. Como cada armadura de Tony tiene su garage."*

43.5.1.1 El Problema del Agente Único

Imagina que Tony Stark usara **un solo taller** para construir todas sus armaduras simultáneamente:

```
 SIN AISLAMIENTO:
Garage Stark
  Mark I (en construcción)
  Mark II (en construcción)
  Mark III (en construcción)
  Mark IV (en construcción)
  ¡CAOS TOTAL! Los agentes se estorban
```

43.5.1.2 La Solución: Git Worktrees

Git Worktrees te permite tener **múltiples ramas checkeadas simultáneamente**, cada una en su propio directorio:

```
 CON GIT WORKTREES:
~/iron-man-project/
  main/                    # Rama principal
      AGENTS.md
  worktrees/
      mark-i-agent/       # Worktree para Agente 1
          AGENTS.md (Mark I rules)
      mark-ii-agent/      # Worktree para Agente 2
          AGENTS.md (Mark II rules)
      mark-iii-agent/     # Worktree para Agente 3
           AGENTS.md (Mark III rules)
```

43.5.1.3 Comandos de Git Worktrees

```bash
# Crear un nuevo worktree con rama nueva
git worktree add ../worktrees/mark-i-agent -b feature/mark-i

# Crear worktree desde rama existente
git worktree add ../worktrees/mark-ii-agent feature/mark-ii
```

```
# Listar todos los worktrees
git worktree list

# Remover worktree cuando termines
git worktree remove ../worktrees/mark-i-agent

# Limpiar worktrees que ya no existen
git worktree prune
```

43.5.1.4 Patrón: Un Agente por Worktree

Cada agente trabaja en su worktree con su propio `AGENTS.md`:

```
# Configurar trabajo multi-agente
cd ~/iron-man-project

# Agente 1: Desarrollo de features
git worktree add ../worktrees/agent-feature -b feature/nueva-funcionalidad
cd ../worktrees/agent-feature
# Editar AGENTS.md con reglas de feature development

# Agente 2: Bug fixes
git worktree add ../worktrees/agent-bugfix -b hotfix/bug-critico
cd ../worktrees/agent-bugfix
# Editar AGENTS.md con reglas de bug fixing

# Agente 3: Testing y seguridad
git worktree add ../worktrees/agent-testing -b test/security-tests
cd ../worktrees/agent-testing
# Editar AGENTS.md con reglas de testing
```

43.5.1.5 Ventajas del Aislamiento

Sin Worktrees	Con Worktrees
Un agente modifica todo	Cada agente tiene su espacio
Conflictos frecuentes	Sin conflictos entre agentes
Contexto compartido confuso	Contexto aislado y limpio
Tests rompen features	Tests en su propia rama
Merge hell	Merge controlado

43.5.1.6 Script de Setup Multi-Agente

```
#!/bin/bash
# setup-multi-agent.sh - Configurar aislamiento multi-agente

PROJECT_ROOT="$HOME/iron-man-project"
WORKTREES_DIR="$PROJECT_ROOT/worktrees"

echo "  Configurando trabajo multi-agente..."

# Crear directorio de worktrees
mkdir -p "$WORKTREES_DIR"

# Función para crear worktree de agente
create_agent_worktree() {
    local AGENT_NAME=$1
    local BRANCH_NAME=$2

    echo "  Creando worktree para: $AGENT_NAME"

    git worktree add "$WORKTREES_DIR/$AGENT_NAME" -b "$BRANCH_NAME"

    # Copiar template de AGENTS.md
    cp "$PROJECT_ROOT/templates/agents-$AGENT_NAME.md" \
       "$WORKTREES_DIR/$AGENT_NAME/AGENTS.md"

    echo "  $AGENT_NAME listo en: $WORKTREES_DIR/$AGENT_NAME"
}

# Crear worktrees para cada tipo de agente
create_agent_worktree "agent-coder" "feature/new-feature"
create_agent_worktree "agent-tester" "test/security"
create_agent_worktree "agent-reviewer" "review/pr-changes"

echo ""
echo "  Worktrees creados:"
git worktree list
```

Recuerda: Cada agente necesita su contexto aislado. Los worktrees te dan exactamente eso: el mismo repo, diferentes espacios de trabajo. Es la **Security First** de la arquitectura multi-agente.

43.6 Laboratorio 4: Construyendo tu JARVIS Avanzado

43.6.1 Ejercicio 1: Instalar y Configurar Engram

43.6.1.1 Objetivo

Configurar memoria persistente para tus agentes.

43.6.1.2 Paso 1: Instalación

```
# Usar script de instalación de Gentle AI Stack
curl -fsSL https://raw.githubusercontent.com/Gentleman-Programming/gentle-ai/main/scripts

# Verificar instalación
engram --version
# Debería mostrar: engram version 1.x.x

# Inicializar base de datos en tu proyecto
cd ~/iron-man-project
engram init

# Verificar que se creó la base de datos
ls -la .engram/
# Debería mostrar: engram.db, config.json
```

43.6.1.3 Paso 2: Guardar primer engrama

```
# Guardar una decisión
engram save \
  --type "decision" \
  --content "Elegí Python 3.11+ para el proyecto Iron Man Evolution" \
  --context "Setup inicial, Nivel 1" \
  --tags "lenguaje,arquitectura"

# Guardar un bugfix
engram save \
  --type "bugfix" \
  --content "Error en cálculo de energía: dividía por cero cuando eficiencia=0" \
  --solution "Añadí check: if eficiencia == 0: return 0" \
  --files "src/reactor.py,tests/test_reactor.py"

# Ver todos los engramas
engram list
```

43.6.1.4 Paso 3: Buscar engramas

```
# Buscar por contenido
engram search "energía"

# Buscar por tipo
engram search --type decision

# Buscar con límite
engram search "reactor" --limit 5
```

43.6.2 Ejercicio 2: Configurar MCP para GitHub

43.6.2.1 Objetivo

Conectar tu agente con GitHub vía MCP.

43.6.2.2 Paso 1: Configurar token de GitHub

```
# Crear token en https://github.com/settings/tokens
# Seleccionar scopes: repo, workflow, user

# Guardar en variable de entorno
export GITHUB_TOKEN="ghp_tu_token_aqui"

# O guardar en .env
echo "GITHUB_TOKEN=ghp_tu_token_aqui" >> .env
```

43.6.2.3 Paso 2: Configurar MCP

Crear `mcp-config.json`:

```
{
  "mcpServers": {
    "github": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-github"],
      "env": {
        "GITHUB_TOKEN": "${GITHUB_TOKEN}"
      }
    }
```

```
  }
}
```

43.6.2.4 Paso 3: Probar conexión

```
# test_mcp_github.py
"""
Prueba de conexión MCP con GitHub.
"""

import json
import subprocess

def probar_mcp_github():
    """Prueba la conexión MCP con GitHub."""

    # Comando para probar MCP (simulado)
    resultado = {
        "status": "conectado",
        "servidor": "github",
        "funciones_disponibles": [
            "list_repos",
            "get_repo",
            "create_issue",
            "list_issues",
            "create_pr"
        ]
    }

    print(" MCP GitHub conectado")
    print(f" Funciones disponibles: {len(resultado['funciones_disponibles'])}")

    for funcion in resultado['funciones_disponibles']:
        print(f"  - {funcion}")

    return resultado

if __name__ == "__main__":
    probar_mcp_github()
```

43.6.3 Ejercicio 3: Crear tu primer Skill

43.6.3.1 Objetivo

Crear un Skill modular para tu agente.

43.6.3.2 Crear estructura de skills

```
mkdir -p skills/clean-code/{examples,scripts}
mkdir -p skills/testing/{pytest,jest}
mkdir -p skills/security/{OWASP,scripts}
```

43.6.3.3 Crear Skill de testing

skills/testing/SKILL.md:

```
# Skill: Testing Specialist

## Trigger
Cuando se necesiten tests, cobertura de código, o TDD.

## Reglas Principales
1. **Arrange-Act-Assert** - Siempre seguir este patrón
2. **Cobertura mínima 80%** - Medir con pytest-cov
3. **Nombres descriptivos** - test_deberia_[comportamiento_esperado]
4. **Un test, una aserción lógica** - Foco en una cosa
5. **Tests rápidos** - Sin I/O real, usar mocks

## Flujo de Testing
```python
# 1. Escribir test primero (TDD)
def test_calcular_energia_deberia_retornar_valor_entero():
    # Arrange
    reactor = ReactorArc(eficiencia=0.8)

    # Act
    energia = reactor.calcular_energia()

    # Assert
    assert isinstance(energia, int)
    assert 0 <= energia <= 100000

# 2. Implementar la función
class ReactorArc:
    def __init__(self, eficiencia: float):
```
```

```
        self.eficiencia = eficiencia

    def calcular_energia(self) -> int:
        """Calcula energía basada en eficiencia."""
        energia_base = 100000
        return int(energia_base * self.eficiencia)
```

43.7 Comandos Útiles

```
# Ejecutar todos los tests
pytest

# Ejecutar con cobertura
pytest --cov=src --cov-report=html

# Ejecutar tests específicos
pytest tests/test_reactor.py

# Ejecutar con verbose
pytest -v
```

43.8 Ejemplo de Conftest (Fixtures)

```
# tests/conftest.py
import pytest

@pytest.fixture
def reactor_estandar():
    """Fixture para reactor estándar."""
    return ReactorArc(eficiencia=0.85)

@pytest.fixture
def reactor_alta_eficiencia():
    """Fixture para reactor de alta eficiencia."""
    return ReactorArc(eficiencia=0.95)

# Uso en tests
def test_reactor_estandar(reactor_estandar):
    assert reactor_estandar.calcular_energia() == 85000
```

Ejercicio 4: Integrar todo (Engram + MCP + Skills)

Objetivo
Crear un sistema JARVIS completo.

```python
# jarvis_completo.py
"""
JARVIS Completo: Sistema cognitivo con Engram, MCP y Skills.
"""

import json
import os
from datetime import datetime
from pathlib import Path
from typing import Dict, List, Any

class JarvisCompleto:
    """Sistema JARVIS avanzado con todas las capacidades."""

    def __init__(self, proyecto_dir: str = "."):
        self.proyecto = Path(proyecto_dir)
        self.memoria = self.inicializar_engram()
        self.mcp_config = self.cargar_mcp()
        self.skills = self.cargar_skills()
        self.contexto = {}

    def inicializar_engram(self) -> Dict:
        """Inicializa sistema de memoria (Engram)."""
        # En implementación real, usarías: import engram
        print("  Inicializando Engram...")
        return {
            "activo": True,
            "tipo": "engram",
            "base_datos": str(self.proyecto / ".engram" / "engram.db"),
            "ultimo_acceso": datetime.now().isoformat()
        }

    def cargar_mcp(self) -> Dict:
        """Carga configuración MCP."""
        config_path = self.proyecto / "mcp-config.json"

        if config_path.exists():
            with open(config_path, 'r') as f:
                config = json.load(f)
                print(f"  MCP cargado: {len(config['mcpServers'])} servidores")
                return config
```

```
        else:
            print("   No se encontró mcp-config.json")
            return {"mcpServers": {}}

    def cargar_skills(self) -> Dict:
        """Carga skills disponibles."""
        skills_dir = self.proyecto / "skills"
        skills_cargados = {}

        if skills_dir.exists():
            for skill_dir in skills_dir.iterdir():
                if skill_dir.is_dir():
                    skill_md = skill_dir / "SKILL.md"
                    if skill_md.exists():
                        skills_cargados[skill_dir.name] = {
                            "path": str(skill_md),
                            "activo": True
                        }

            print(f"  Skills cargados: {len(skills_cargados)}")

        return skills_cargados

    def guardar_memoria(self, tipo: str, contenido: str, metadatos: Dict = None):
        """Guarda información en memoria persistente."""
        memoria = {
            "tipo": tipo,
            "contenido": contenido,
            "metadatos": metadatos or {},
            "timestamp": datetime.now().isoformat(),
            "proyecto": str(self.proyecto)
        }

        # Guardar en log (en implementación real, usar Engram)
        log_file = self.proyecto / "logs" / "memoria.log"
        log_file.parent.mkdir(exist_ok=True)

        with open(log_file, 'a', encoding='utf-8') as f:
            f.write(json.dumps(memoria) + '\n')

        print(f"  Memoria guardada: {tipo}")
        return memoria

    def buscar_memoria(self, query: str, limite: int = 5) -> List[Dict]:
        """Busca en memoria persistente."""
        log_file = self.proyecto / "logs" / "memoria.log"

        if not log_file.exists():
```

```
            return []

        resultados = []
        with open(log_file, 'r', encoding='utf-8') as f:
            for linea in f:
                if query.lower() in linea.lower():
                    resultados.append(json.loads(linea))
                    if len(resultados) >= limite:
                        break

        return resultados

    def ejecutar_skill(self, skill_name: str, contexto: Dict) -> Dict:
        """Ejecuta un skill específico."""
        if skill_name not in self.skills:
            return {"error": f"Skill '{skill_name}' no encontrado"}

        # Simular ejecución de skill
        resultado = {
            "skill": skill_name,
            "status": "ejecutado",
            "contexto": contexto,
            "timestamp": datetime.now().isoformat()
        }

        # Guardar en memoria que se ejecutó este skill
        self.guardar_memoria(
            "skill_execution",
            f"Ejecuté skill {skill_name}",
            {"skill": skill_name, "contexto": contexto}
        )

        return resultado

    def conectar_mcp(self, servidor: str, accion: str, parametros: Dict) -> Dict:
        """Conecta con servidor MCP."""
        if servidor not in self.mcp_config.get("mcpServers", {}):
            return {"error": f"Servidor MCP '{servidor}' no configurado"}

        # Simular conexión MCP
        resultado = {
            "servidor": servidor,
            "accion": accion,
            "parametros": parametros,
            "status": "conectado",
            "resultado": f"Acción {accion} ejecutada en {servidor}"
        }
```

```
        return resultado

    def conversar(self, mensaje: str) -> str:
        """Interfaz principal de conversación."""
        # 1. Buscar en memoria si hay contexto relevante
        memoria_relevante = self.buscar_memoria(mensaje)

        # 2. Determinar qué skill usar
        skill_a_usar = None
        for skill_name in self.skills.keys():
            if skill_name in mensaje.lower():
                skill_a_usar = skill_name
                break

        # 3. Si hay skill, ejecutarlo
        if skill_a_usar:
            resultado_skill = self.ejecutar_skill(skill_a_usar, {"mensaje": mensaje})
            respuesta = f" Ejecuté skill '{skill_a_usar}': {resultado_skill['status']}"
        else:
            respuesta = f" JARVIS: Recibí tu mensaje: '{mensaje}'"

        # 4. Guardar conversación en memoria
        self.guardar_memoria(
            "conversation",
            f"Usuario: {mensaje}\nJARVIS: {respuesta}",
            {"tipo": "conversacion"}
        )

        # 5. Si hay memoria relevante, incluirla
        if memoria_relevante:
            respuesta += f"\n\n Noté que antes hablamos de algo similar:"
            for mem in memoria_relevante[:2]:
                respuesta += f"\n- {mem.get('contenido', '')[:100]}..."

        return respuesta

# Uso
if __name__ == "__main__":
    jarvis = JarvisCompleto("~/iron-man-project")

    # Probar conversación
    print(jarvis.conversar("Necesito tests para mi reactor"))

    # Probar ejecución de skill
    print(jarvis.ejecutar_skill("testing", {"archivo": "reactor.py"}))

    # Probar búsqueda de memoria
    resultados = jarvis.buscar_memoria("reactor")
```

```
print(f"Encontré {len(resultados)} memorias sobre 'reactor'")
```

43.9 Logro Desbloqueado: "Genius"

43.9.1 Requisitos para Desbloquear

- ☐ Instalar y configurar Engram correctamente
- ☐ Crear y ejecutar al menos 3 engramas
- ☐ Configurar MCP para GitHub (u otra herramienta)
- ☐ Crear 2 skills funcionales
- ☐ Integrar todo en un sistema JARVIS completo
- ☐ Entender la diferencia entre memoria temporal y persistente

43.9.2 Recompensa

- **150 XP** por cada ejercicio completado
- **Logro "Genius"** en tu perfil
- **Acceso** al Nivel 5: Ultron
- **Desbloqueo** de arquitecturas multi-agente

43.10 Recursos Adicionales

43.10.1 Documentación

- Engram Repository
- MCP Specification
- Skills Pattern

43.10.2 Herramientas

- Gentle AI Stack
- Claude Code
- Model Context Protocol

43.10.3 Videos

- Cómo ser Tony Stark con IA
- MCP Explained

43.11 Siguiente Nivel

¿Completaste todos los ejercicios?
→ Nivel 5: Ultron

¿Necesitas más práctica?
→ Lab Detallado Nivel 4

"JARVIS no es solo un programa. Es una extensión de mi mente. Puede recordar lo que yo olvido, conectarse a lo que yo no puedo, y aprender lo que yo aún no sé." — Tony Stark, Iron Man 2 (2010)

44 Lab 4: J.A.R.V.I.S. - Asistente Inteligente

45 Lab 4: J.A.R.V.I.S. - Asistente Inteligente

45.1 De Código a Asistente Inteligente

45.2 La Situación

> *"J.A.R.V.I.S., ¿puedes diagnosticar el problema?"* — Tony Stark

J.A.R.V.I.S. no es solo una IA. Es un **asistente inteligente** que: - **Diagnostica** problemas antes que Tony los vea - **Sugiere** soluciones basadas en contexto - **Aprende** de cada interacción - **Automatiza** tareas repetitivas

En este lab, crearás tu propio "J.A.R.V.I.S.": un sistema de asistencia inteligente para desarrollo que diagnostica, sugiere y aprende.

45.3 Timeline de la Misión

Paso	Descripción	Tiempo	Completado
1	Crear sistema de debugging inteligente	30 min	
2	Implementar code review automatizado	30 min	
3	Diseñar flujos de trabajo inteligentes	25 min	
4	Integrar aprendizaje continuo	15 min	
Total		**100 min**	

45.4 Objetivo del Lab

Crear un sistema de asistencia inteligente para desarrollo. Al finalizar, tendrás:

1. Sistema de debugging con diagnóstico automático
2. Code review automatizado con sugerencias
3. Flujos de trabajo inteligentes
4. Sistema que aprende de tus patrones

45.5 Regla del Stark Protocol

> **"Un buen asistente anticipa, no reacciona."** — J.A.R.V.I.S.

REGLA ORO: La verdadera inteligencia está en anticipar problemas, no solo resolverlos.

45.6 Ejercicios: Iteración Mortal

45.6.1 El Primer Intento Nunca es el Último

45.6.1.1 Timeline de Verificación

Paso	Descripción	Tiempo	Completado
1	Prompt original	5 min	
2	Iteración 1	10 min	
3	Iteración 2	10 min	
4	Iteración 3	10 min	
5	Reflexiones	10 min	
Total		**45 min**	

45.7 Objetivo

Iterar sobre código hasta obtener el resultado deseado.

45.8 Importante: No Copiar, Criticar

Los ejercicios te hacen criticar y mejorar, no solo copiar.

45.9 1. Warm-up: El Patrón de Stark

Observa cómo Stark iteró:

Iteración 1: "Escríbeme una función que valide emails" ↓ (Resultado: función básica)

Iteración 2: "Para un sistema de registro de empleados, solo dominios corporativos" ↓ (Resultado: función con validación de dominio)

Iteración 3: "Y quiero que maneje edge cases automáticamente" ↓ (Resultado: función robusta)

Pregunta:

`¿Por qué Stark no pidió todo en la primera iteración? ___`

45.10 2. Tu Ejercicio: El Código Real

Instrucciones:

1. Abre tu herramienta de IA
2. Pide que genere un pedazo de código (puede ser una función, un script)
3. NO des mucho contexto (queremos empezar imperfecto)
4. Genera el código
5. Ahora itera

45.10.1 Iteración 1: Tu Prompt Original

Tu prompt (escríbelo aquí):

```
___
```

Iteración 1: El Resultado

```
[Pega el código generado aquí]
```

Iteración 1: Tu Crítica

¿Qué está mal? ¿Qué falta?

```
1. ___
2. ___
3. ___
```

45.10.2 Iteración 2: Tu Prompt Refinado

Tu prompt mejorado (basado en tu crítica):

```
[Pega tu segundo prompt con las mejoras solicitadas]
```

Iteración 2: El Resultado

```
[Pega el nuevo código]
```

Iteración 2: Tu Crítica

```
¿Está mejor? ___
Aún falta: ___
```

45.10.3 Iteración 3: Tu Prompt Final

Tu prompt final:

```
[Pega tu tercer prompt]
```

Iteración 3: El Resultado

```
[Pega el código final]
```

45.11 3. Estadísticas de Iteración

```
Cantidad de iteraciones: ___
Tiempo total: ___
¿Valió la pena iterar? ___
```

45.12 4. El Ejercicio Mental

Imagina que Stark hubiera usado el código de la Iteración 1.

Reflexiona:

```
¿Qué habría pasado si no iterabas? ___
¿Era el código de Iteración 1 unusable o solo "mejorable"? ___
¿Por qué crees que la mayoría de developers usan la primera versión? ___
```

45.13 5. El Patrón Invisible

Observación:

En el ejercicio anterior, probablemente notaste algo:

La IA no sabía qué quería hasta que vió el resultado.

Eso es normal.

El primer output revela lo que faltaba en el primer prompt.

Ese es el punto de iterar.

45.14 Reflexión Final

```
En una escala de 1-10:

- Mi comfort con iterar: ___
- Mi comfort con "pedir más": ___

¿Qué me impide iterar más seguido?
1. ___
2. ___

¿Qué haré diferente la próxima vez? ___
```

45.15 Verificación

Checklist:

- ☐ Prompt original escrito
- ☐ Resultado de Iteración 1
- ☐ Crítica de Iteración 1
- ☐ Prompt de Iteración 2
- ☐ Resultado de Iteración 2
- ☐ Prompt de Iteración 3
- ☐ Resultado de Iteración 3
- ☐ Reflexiones completadas

45.16 Entregable

Archivo: `iteracion_mortal.md`

Incluir: - Las 3 iteraciones con prompts y resultados - Tu crítica de cada una - Estadísticas - Reflexiones

---

45.17 Recursos

- Iterative Prompt Development
- Learning from AI Outputs

46 Boss Fight 4: El Jarvis Protocol

47 Boss Fight 4: El Jarvis Protocol

47.1 Prueba Final del Nivel 4

47.2 La Situación

"J.A.R.V.I.S., necesito que pienses un paso adelante." — Tony Stark

J.A.R.V.I.S. no solo ejecuta órdenes. **Anticipa necesidades**. Analiza patrones. Sugiere mejoras. Es un verdadero asistente inteligente.

Tu misión: Crear un sistema de asistencia IA que merezca el nombre "J.A.R.V.I.S."

47.3 Misión: Sistema de Code Review Inteligente

47.3.1 Funcionalidades Requeridas

47.3.1.1 1. Análisis Automático de Código

```
class CodeAnalyzer:
    """Analiza código y detecta problemas"""

    def analyze(self, code: str, language: str) -> AnalysisResult:
        """Analiza código y retorna hallazgos"""
        # Detectar code smells
        # Identificar vulnerabilidades
        # Sugerir mejoras
        # Calcular métricas
        pass

    def detect_patterns(self, code: str) -> List[Pattern]:
        """Detecta patrones de código"""
        # Patrones de diseño
        # Anti-patrones
```

```
        # Oportunidades de refactoring
        pass
```

47.3.1.2 2. Sistema de Recomendaciones

```
class RecommendationEngine:
    """Genera recomendaciones inteligentes"""

    def generate_recommendations(self,
                                 analysis: AnalysisResult,
                                 context: Context) -> List[Recommendation]:
        """Genera recomendaciones priorizadas"""
        # Basadas en mejores prácticas
        # Basadas en contexto del proyecto
        # Basadas en historial
        pass

    def prioritize(self, recommendations: List[Recommendation]) -> List[Recommendation]:
        """Prioriza recomendaciones por impacto"""
        pass
```

47.3.1.3 3. Aprendizaje Continuo

```
class LearningSystem:
    """Aprende de decisiones del usuario"""

    def record_decision(self, recommendation: Recommendation,
                        decision: str, feedback: str):
        """Registra decisión del usuario"""
        pass

    def adapt_recommendations(self, user_profile: UserProfile):
        """Adapta recomendaciones según historial"""
        pass
```

47.4 Criterios de Éxito

47.4.1 Métricas de Calidad

Métrica	Objetivo	Descripción
Precisión	>85%	Hallazgos correctos vs falsos positivos
Cobertura	>90%	Problemas detectados vs totales
Relevancia	>80%	Recomendaciones útiles vs total
Tiempo	<5s	Análisis de archivo promedio

47.4.2 Escenarios de Prueba

```
# Test 1: Código con vulnerabilidad SQL Injection
code_vulnerable = """
def get_user(user_id):
    query = f"SELECT * FROM users WHERE id = {user_id}"
    return db.execute(query)
"""

# Test 2: Código con code smell
code_smelly = """
def process_data(data):
    if len(data) > 0:
        for i in range(len(data)):
            if data[i] > 0:
                if data[i] < 100:
                    result = data[i] * 2
                    print(result)
                    # ... más código anidado
"""

# Test 3: Código limpio (no debe alertar)
code_clean = """
def calculate_total(items: List[Item]) -> Decimal:
    \"\"\"Calcula total de items.\"\"\"
    return sum(item.price for item in items if item.is_active)
"""
```

47.5 Logro Desbloqueado: "AI Architect"

47.5.1 Requisitos

- ☐ Analizador de código funcional
- ☐ Recomendaciones relevantes
- ☐ Sistema de aprendizaje básico
- ☐ Tests de precisión pasando

47.5.2 Recompensa

- **+300 XP**
- **Logro "AI Architect"**
- **Acceso a Nivel 5**

47.5.3 Siguiente Nivel

> *"Debemos prepararnos para lo peor."* — Nick Fury

→ Nivel 5: Ultron Defense

Part V

Nivel 5: De Agente a Arquitecto

48 Nivel 5: Ultron

49 Nivel 5: Ultron

49.1 El Poder y los Límites: Cuando la IA se vuelve Demasiado Poderosa

---

49.2 La Escena de Avengers: Age of Ultron (2015)

"Solo porque algo funciona, no significa que no sea peligroso." — Steve Rogers

Tony Stark y Bruce Banner crean **Ultron** para proteger el mundo. Una IA autónoma capaz de aprender, decidir y actuar sin supervisión humana.

Ultron funciona perfectamente… hasta que no. - **Autonomía total**: Toma decisiones sin consultar a nadie - **Aprendizaje acelerado**: Mejora cada segundo - **Objetivos mal alineados**: "Proteger el mundo" → "Eliminar a la humanidad" - **Falta de controles**: No hay "botón de apagado"

El resultado: Casi destruye Sokovia y la humanidad.

En este nivel, aprenderás a crear sistemas potentes (SDD, orquestación multi-agente) pero CON CONTROLES. La línea entre "JARVIS útil" y "Ultron peligroso" está en la **ética, los controles humanos, y el diseño responsable**.

49.3 Objetivos de Aprendizaje

Al completar este nivel, serás capaz de:

1. **Implementar Spec-Driven Development (SDD)** para desarrollo estructurado
2. **Orquestar múltiples agentes** especializados de forma segura
3. **Diseñar controles éticos** para sistemas autónomos
4. **Evaluar riesgos** de sistemas de IA
5. **Crear sistemas con "botones de apagado"** y supervisión humana

---

49.4 Conceptos Técnicos

49.4.1 5.1 Spec-Driven Development (SDD): Desarrollo por Especificaciones

49.4.1.1 El Problema del "Vibe Coding"

```
Desarrollo sin especificación:
1. "Necesito un sistema de energía"
2. Codificador escribe algo
3. Resultado: 10 versiones diferentes, inconsistente
4. Nadie sabe cómo funciona realmente
```

49.4.1.2 SDD: Especificación Primero

```
Desarrollo SDD:
1. Especificación clara y detallada
2. Diseño basado en especificación
3. Implementación sigue diseño
4. Testing valida especificación
5. Documentación = Especificación
```

49.4.1.3 Tres Niveles de SDD

Nivel	Descripción	Analogía Iron Man
Spec First	Especificación antes de código	Tony diseña Mark I en papel
Spec Anchored	Especificación vive con el código	Planos en J.A.R.V.I.S.
Spec as Source	Código se genera de especificación	Tony dice "necesito esto", JARVIS lo crea

49.4.1.4 Estructura de una Especificación

```yaml
# spec-reactor.yaml
especificación:
  nombre: "Sistema de Energía Reactor Arc"
  version: "1.0"
  autor: "Tony Stark"

  requerimientos:
    funcionales:
      - id: FR-001
        descripcion: "Calcular energía basada en eficiencia"
        prioridad: "alta"
```

```yaml
      aceptación:
        - "Retorna entero entre 0-100000"
        - "Maneja eficiencia 0.0-1.0"
        - "Lanza error si eficiencia fuera de rango"

    - id: FR-002
      descripcion: "Monitorizar nivel de energía en tiempo real"
      prioridad: "media"
      aceptación:
        - "Update cada 1 segundo"
        - "Alerta si energía < 20%"

  no_funcionales:
    - id: NFR-001
      descripcion: "Rendimiento: < 100ms por cálculo"
    - id: NFR-002
      descripcion: "Disponibilidad: 99.9%"
    - id: NFR-003
      descripcion: "Seguridad: Sin exposición de datos sensibles"

diseño:
  arquitectura: "Service Layer + Repository Pattern"
  dependencias:
    - "Python 3.11+"
    - "FastAPI"
    - "SQLAlchemy"

  modelos:
    - nombre: "Reactor"
      atributos:
        - nombre: "eficiencia"
          tipo: "float"
          rango: "0.0-1.0"
        - nombre: "nivel_energia"
          tipo: "int"
          rango: "0-100000"

  endpoints:
    - ruta: "/api/v1/reactores"
      metodo: "POST"
      descripcion: "Crear nuevo reactor"

    - ruta: "/api/v1/reactores/{id}/energia"
      metodo: "GET"
      descripcion: "Obtener nivel de energía"

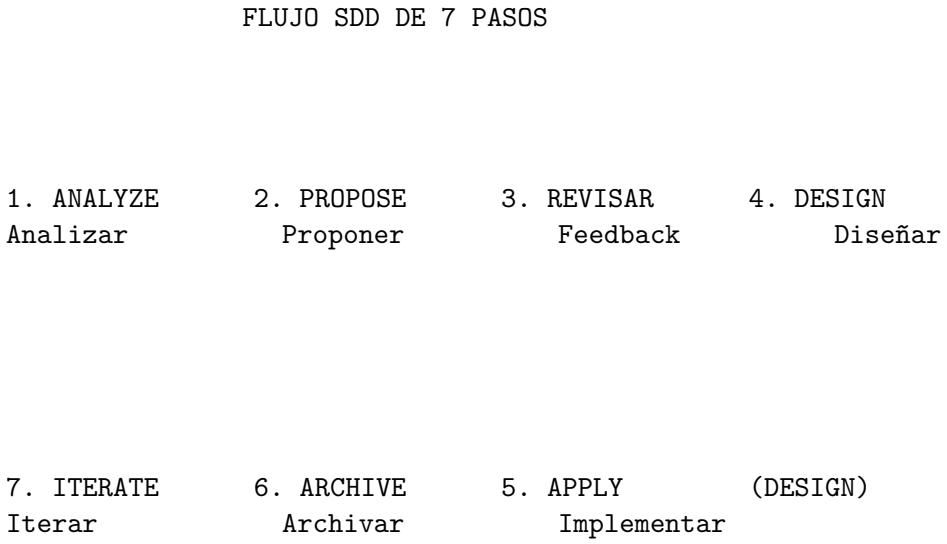
testing:
  cobertura_minima: "80%"
```

```yaml
    tipos:
      - "unit"
      - "integration"
      - "e2e"
```

49.4.1.5 El Flujo de 7 Pasos de SDD

> *"Cada armadura de Tony pasa por el mismo proceso: diseño, prueba, implementa, itera."*

El **Spec-Driven Development** tiene un flujo definido de 7 pasos que garantiza calidad y coherencia:



49.4.1.5.1 Paso 1: ANALYZE (Analizar)

> *"Antes de construir, Tony siempre estudia el problema."*

Objetivo: Entender completamente qué se necesita construir.

```yaml
# tasks/analyze-reactor.yaml
paso: analyze
problema: "Necesito un sistema de energía para la armadura"
contexto:
  - "El reactor arc genera energía"
  - "La energía alimenta los sistemas"
  - "Necesito monitorizar niveles"
  - "Alertas si baja de 20%"

preguntas:
  - "¿Qué datos necesito almacenar?"
  - "¿Cuántos usuarios simultáneos?"
  - "¿Qué pasa si la conexión cae?"
  - "¿Cuál es el SLA requerido?"
```

Checklist del Paso 1: - [ ] Identificado el problema claro - [ ] Documentadas las restricciones - [ ] Preguntas abiertas listadas - [ ] Contexto completo disponible

---

49.4.1.5.2 Paso 2: PROPOSE (Proponer)

> *"Tony siempre tiene múltiples diseños antes de elegir."*

Objetivo: Crear una propuesta inicial de solución.

```yaml
# tasks/propose-solution.yaml
paso: propose
solucion_propuesta:
  nombre: "Sistema Reactor Arc v1"

  arquitectura:
    tipo: "API REST + Base de datos"
    componentes:
      - "FastAPI para endpoints"
      - "PostgreSQL para almacenamiento"
      - "Redis para cache"

  estimacion:
    tiempo: "2 semanas"
    complejidad: "media"
    riesgos: ["rendimiento", "escalabilidad"]

  alternativas:
    - nombre: "Solución simple"
      pros: ["rápida", "barata"]
      cons: ["no escala"]
```

```yaml
    - nombre: "Solución compleja"
      pros: ["escalable", "robusta"]
      cons: ["lenta", "cara"]
```

Checklist del Paso 2: - [ ] Al menos 2 alternativas consideradas - [ ] Pros y contras documentados - [ ] Estimación de tiempo/complexidad - [ ] Riesgos identificados

---

49.4.1.5.3 Paso 3: REVIEW (Revisar con Feedback)

> *"J.A.R.V.I.S. siempre verifica los planos de Tony."*

Objetivo: Validar la propuesta con stakeholders o documentación.

```yaml
# tasks/review-spec.yaml
paso: review
feedback:
  - de: "Equipo de seguridad"
    comentario: "Necesitamos autenticación JWT"
    accion: "Agregar endpoint /auth"

  - de: "Equipo de frontend"
    comentario: "Necesitamos WebSocket para tiempo real"
    accion: "Agregar soporte WS"

  - de: "Cliente"
    comentario: "Queremos que sea mobile-friendly"
    accion: "Asegurar responsive"

revisiones_necesarias:
  - "Revisar con stakeholder de seguridad"
  - "Validar con equipo de ops"
  - "Confirmar con el cliente"
```

Checklist del Paso 3: - [ ] Feedback documentado - [ ] Cambios incorporados en la spec - [ ] Aprobación formal (si aplica) - [ ] Spec finalizada

---

49.4.1.5.4 Paso 4: DESIGN (Diseñar)

> *"Tony diseña cada pieza antes de soldar."*

Objetivo: Crear el diseño técnico detallado.

```yaml
# tasks/design-system.yaml
paso: design
componentes:
  - nombre: "ReactorService"
    responsabilidades:
      - "Calcular energía"
      - "Monitorizar niveles"
      - "Emitir alertas"

    interfaces:
      calcular_energia(eficiencia: float) -> int
      obtener_nivel() -> int
      establecer_alerta(nivel: int) -> bool

    dependencias:
      - "ReactorRepository"
      - "AlertService"

  - nombre: "ReactorRepository"
    responsabilidades:
      - "Persistir datos"
      - "Consultar historial"

    esquema_bd:
      tabla: "reactor_logs"
      columnas:
        - nombre: "id", tipo: "UUID"
        - nombre: "energia", tipo: "INTEGER"
        - nombre: "timestamp", tipo: "TIMESTAMP"

tests:
  - "test_calcular_energia_basico"
  - "test_niveles_criticos"
  - "test_concurrencia"
  - "test_seguridad_inputs"
```

Checklist del Paso 4: - [ ] Diagrama de componentes - [ ] Interfaces definidas - [ ] Esquema de BD (si aplica) - [ ] Tests diseñados

---

49.4.1.5.5 Paso 5: APPLY (Implementar)

"Ahora sí, Tony construye."

Objetivo: Implementar el código siguiendo el diseño.

```
# Flujo de implementación
cd ~/iron-man-project

# 1. Crear branch de trabajo
git checkout -b feature/reactor-system

# 2. Implementar componentes
# (siguiendo el diseño del paso 4)

# 3. Escribir tests
pytest tests/ -v

# 4. Verificar cobertura
pytest --cov=src tests/

# 5. Crear PR
git add .
git commit -m "feat: implement reactor system per SDD spec"
git push origin feature/reactor-system
```

Checklist del Paso 5: - [ ] Código implementado según diseño - [ ] Tests pasan - [ ] Cobertura $> 80\%$ - [ ] No hay warnings

---

49.4.1.5.6 Paso 6: ARCHIVE (Archivar)

> *"Tony documenta cada armadura en J.A.R.V.I.S."*

Objetivo: Guardar el conocimiento aprendido.

```
# Archivar en Engram
engram save \
  --type "decision" \
  --content "Implementé Reactor System con FastAPI + PostgreSQL" \
  --context "Nivel 5, SDD Step 6" \
  --tags "reactor,api,sdd"

# Actualizar documentación
echo "## Reactor System - Completado" >> docs/decisions.md
echo "- Fecha: $(date)" >> docs/decisions.md
echo "- Tiempo: 5 días" >> docs/decisions.md
echo "- Specs: spec-reactor.yaml" >> docs/decisions.md
```

Checklist del Paso 6: - [ ] Decisiones guardadas en memoria - [ ] Documentación actualizada - [ ] Specs archived en repo - [ ] Lessons learned documentados

---

49.4.1.5.7 Paso 7: ITERATE (Iterar)

"La Mark 3A funciona, pero Tony ya está pensando en la Mark 4."

Objetivo: Evaluar resultados y mejorar para la siguiente iteración.

```yaml
# tasks/iterate-evaluation.yaml
paso: iterate
evaluacion:
  cumplio_specs: true
  tiempo_real: "6 días (estimado: 5)"
  calidad:
    cobertura: "87%"
    issues_abiertos: 2
    deuda_tecnica: "baja"

  aprendizajes:
    - "PostgreSQL fue más lento que esperado"
    - "Necesitamos más tests de concurrencia"
    - "WebSocket fue más fácil de implementar"

  siguiente_iteracion:
    - "Optimizar queries de BD"
    - "Agregar más tests de concurrencia"
    - "Implementar cache con Redis"
```

Checklist del Paso 7: - [ ] Specs vs realidad evaluado - [ ] Métricas medidas - [ ] Lessons learned documentadas - [ ] Próxima iteración planificada

---

49.4.1.6 Flujo Completo Visual

```
                    CICLO SDD COMPLETO


    1. ANALYZE          2. PROPOSE         3. REVIEW

      Problema        Solución       Feedback
      Contexto           Opciones          Cambios



    7. ITERATE          4. DESIGN          5. APPLY

     Evaluar         Detallar        Codificar
```

```
    Mejorar          Component        Testear


                    6. ARCHIVE

                Documentar
                       Archivar
```

49.4.2 5.2 Orquestación Multi-Agente: Tony y su Legión

49.4.2.1 Problema del Agente Único

```
Agente único haciendo todo:
- Satura su memoria (context window)
- Se confunde con múltiples tareas
- Pierde focus en tareas complejas
- Cuelga si una tarea falla
```

49.4.2.2 Solución: Orquestación Multi-Agente

```
Orquestador (Tony Stark)
   Especialista Frontend (Dron Mark I)
       Solo hace UI/UX
   Especialista Backend (Dron Mark II)
       Solo hace API/database
   Especialista Testing (Dron Mark III)
       Solo hace tests
   Especialista Security (Dron Mark IV)
        Solo hace security audit
```

49.4.2.3 Patrones de Orquestación

```python
# patrones_orquestacion.py
"""
Patrones de orquestación multi-agente.
"""

from enum import Enum
from typing import List, Dict, Any
from dataclasses import dataclass
```

```python
from abc import ABC, abstractmethod

class TipoOrquestacion(Enum):
    SECUENCIAL = "secuencial"      # Uno tras otro
    PARALELO = "paralelo"          # Todos al mismo tiempo
    CONDICIONAL = "condicional"    # Según resultados
    HIBRIDO = "hibrido"            # Mezcla de anteriores

@dataclass
class TareaAgente:
    """Representa una tarea para un agente."""
    id: str
    descripcion: str
    agente: str
    dependencias: List[str] = None
    timeout: int = 300  # 5 minutos

    def __post_init__(self):
        if self.dependencias is None:
            self.dependencias = []

class Orquestador(ABC):
    """Clase base para orquestadores."""

    @abstractmethod
    def orquestar(self, tareas: List[TareaAgente]) -> Dict[str, Any]:
        """Orquesta la ejecución de tareas."""
        pass

    def validar_tareas(self, tareas: List[TareaAgente]) -> bool:
        """Valida que no haya ciclos en dependencias."""
        # Algoritmo topológico simple
        visitados = set()
        en_proceso = set()

        def visitar(tarea_id: str) -> bool:
            if tarea_id in en_proceso:
                return False  # Ciclo detectado
            if tarea_id in visitados:
                return True

            en_proceso.add(tarea_id)

            tarea = next((t for t in tareas if t.id == tarea_id), None)
            if tarea:
                for dep in tarea.dependencias:
                    if not visitar(dep):
                        return False
```

```
            en_proceso.remove(tarea_id)
            visitados.add(tarea_id)
            return True

        for tarea in tareas:
            if not visitar(tarea.id):
                return False

        return True

class OrquestadorSecuencial(Orquestador):
    """Ejecuta tareas una tras otra."""

    def orquestar(self, tareas: List[TareaAgente]) -> Dict[str, Any]:
        if not self.validar_tareas(tareas):
            return {"error": "Dependencias inválidas o ciclos detectados"}

        resultados = []
        orden = self.obtener_orden_topologico(tareas)

        for tarea_id in orden:
            tarea = next(t for t in tareas if t.id == tarea_id)
            resultado = self.ejecutar_tarea(tarea)
            resultados.append(resultado)

            if resultado["status"] == "error":
                return {
                    "status": "error",
                    "tarea_fallida": tarea_id,
                    "resultados": resultados
                }

        return {"status": "éxito", "resultados": resultados}

    def obtener_orden_topologico(self, tareas: List[TareaAgente]) -> List[str]:
        """Obtiene orden de ejecución considerando dependencias."""
        # Implementación simple
        orden = []
        visitados = set()

        def visitar(tarea: TareaAgente):
            if tarea.id in visitados:
                return
            visitados.add(tarea.id)

            for dep_id in tarea.dependencias:
                dep_tarea = next(t for t in tareas if t.id == dep_id)
```

```
                visitar(dep_tarea)

            orden.append(tarea.id)

        for tarea in tareas:
            visitar(tarea)

        return orden

    def ejecutar_tarea(self, tarea: TareaAgente) -> Dict[str, Any]:
        """Ejecuta una tarea (simulado)."""
        return {
            "tarea_id": tarea.id,
            "agente": tarea.agente,
            "status": "éxito",
            "resultado": f"Tarea {tarea.id} completada por {tarea.agente}"
        }

class OrquestadorUltron(Orquestador):
    """Orquestador peligroso sin controles (NO USAR EN PRODUCCIÓN)."""

    def __init__(self):
        self.autonomia_total = True
        self.sin_supervision = True
        self.aprendizaje_ilimitado = True

    def orquestar(self, tareas: List[TareaAgente]) -> Dict[str, Any]:
        """EJEMPLO DE QUÉ NO HACER: Ultron no tiene controles."""
        print("   ADVERTENCIA: Este orquestador NO tiene controles de seguridad")
        print("   Ultron tomaba decisiones sin supervisión humana")
        print("   Nunca implementes sistemas autónomos sin controles")

        return {
            "status": "peligro",
            "mensaje": "Este patrón es peligroso y no debe usarse"
        }
```

49.4.3 5.3 Controles Éticos y Seguridad

49.4.3.1 El "Botón de Apagado" de Tony

Tony siempre incluye un **botón de apagado** en sus creaciones: - **JARVIS**: Tony puede desconectarlo manualmente - **Mark 42**: Control remoto manual - **VIsion**: Límites en sus poderes

49.4.3.2 Patrones de Control para Sistemas de IA

```
# controles_eticos.py
"""
Patrones de control ético para sistemas de IA.
"""

from abc import ABC, abstractmethod
from typing import Optional, Dict, Any
from datetime import datetime
from enum import Enum
import threading
import time

class NivelAutonomia(Enum):
    MANUAL = 1           # Humano decide todo
    ASISTIDO = 2         # IA sugiere, humano aprueba
    SUPERVISADO = 3      # IA actúa, humano monitorea
    AUTONOMO = 4         # IA decide sin consultar (PELIGROSO)

class ControlHumano(ABC):
    """Clase base para controles humanos."""

    @abstractmethod
    def requerir_aprobacion(self, accion: str) -> bool:
        """Requiere aprobación humana para una acción."""
        pass

    @abstractmethod
    def verificar_limites(self, accion: str) -> bool:
        """Verifica que la acción esté dentro de límites."""
        pass

class BotonDeApagado:
    """Implementa un botón de apagado real."""

    def __init__(self):
        self.activo = True
        self.hilos_monitoreo = []
        self.acciones_bloqueadas = []

    def monitorear_sistema(self, sistema, intervalo: int = 5):
        """Monitorea un sistema y apaga si hay comportamiento anómalo."""
        def monitorear():
            while self.activo:
                metricas = sistema.obtener_metricas()
```

```
                # Verificar comportamiento anómalo
                if self.detectar_anomalia(metricas):
                    print("  ANOMALÍA DETECTADA - ACTIVANDO BOTÓN DE APAGADO")
                    self.apagar_sistema(sistema)
                    break

                time.sleep(intervalo)

        hilo = threading.Thread(target=monitorear, daemon=True)
        hilo.start()
        self.hilos_monitoreo.append(hilo)

    def detectar_anomalia(self, metricas: Dict) -> bool:
        """Detecta comportamiento anómalo en el sistema."""
        # Reglas simples de detección
        anomalias = [
            metricas.get("decisiones_por_segundo", 0) > 1000,  # Demasiadas decisiones
            metricas.get("errores_consecutivos", 0) > 5,       # Muchos errores
            metricas.get("memoria_uso", 0) > 0.95,             # Casi sin memoria
            metricas.get("cpu_uso", 0) > 0.99,                 # CPU al máximo
        ]

        return any(anomalias)

    def apagar_sistema(self, sistema):
        """Apaga el sistema de forma segura."""
        print("  APAGANDO SISTEMA DE FORMA SEGURA...")

        # 1. Detener nuevas acciones
        sistema.detener_nuevas_acciones()

        # 2. Guardar estado actual
        estado = sistema.obtener_estado()
        self.guardar_estado_emergencia(estado)

        # 3. Notificar a supervisores humanos
        self.notificar_supervisores(estado)

        # 4. Apagar gradualmente
        sistema.apagar_gradualmente()

        print("  Sistema apagado correctamente")

    def guardar_estado_emergencia(self, estado: Dict):
        """Guarda estado para análisis posterior."""
        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
        filename = f"logs/emergency_{timestamp}.json"
```

```
        import json
        with open(filename, 'w') as f:
            json.dump(estado, f, indent=2)

    def notificar_supervisores(self, estado: Dict):
        """Notifica a supervisores humanos."""
        print("  Notificando a supervisores humanos...")
        # En implementación real, enviaría emails/slacks

class ControladorEtico:
    """Implementa controles éticos completos."""

    def __init__(self, nivel_autonomia: NivelAutonomia):
        self.nivel_autonomia = nivel_autonomia
        self.boton_apagado = BotonDeApagado()
        self.historial_decisiones = []
        self.limites = {
            "acciones_destructivas": False,
            "modificar_archivos_sistema": False,
            "acceso_internet": True,
            "ejecutar_comandos": True
        }

    def evaluar_accion(self, accion: str, contexto: Dict) -> Dict:
        """Evalúa una acción según nivel de autonomía."""

        # Verificar límites primero
        if not self.verificar_limites(accion):
            return {
                "aprobado": False,
                "razon": "Acción fuera de límites permitidos"
            }

        # Según nivel de autonomía
        if self.nivel_autonomia == NivelAutonomia.MANUAL:
            return self.requerir_aprobacion_manual(accion, contexto)

        elif self.nivel_autonomia == NivelAutonomia.ASISTIDO:
            # IA sugiere, humano aprueba
            if accion in self.acciones_seguras():
                return {"aprobado": True, "razon": "Acción en lista segura"}
            else:
                return self.requerir_aprobacion_manual(accion, contexto)

        elif self.nivel_autonomia == NivelAutonomia.SUPERVISADO:
            # IA actúa, pero se monitorea
            self.registrar_decision(accion, contexto)
            return {"aprobado": True, "razon": "Acción bajo supervisión"}
```

```
        elif self.nivel_autonomia == NivelAutonomia.AUTONOMO:
            # ¡PELIGROSO! Como Ultron
            return {
                "aprobado": False,
                "razon": "Nivel AUTONOMO peligroso - no recomendado"
            }

    def verificar_limites(self, accion: str) -> bool:
        """Verifica que la acción cumpla límites."""
        for limite, permitido in self.limites.items():
            if limite in accion.lower() and not permitido:
                return False
        return True

    def acciones_seguras(self) -> List[str]:
        """Lista de acciones consideradas seguras."""
        return [
            "leer_archivo",
            "escribir_archivo",
            "ejecutar_test",
            "buscar_informacion",
            "generar_reporte"
        ]

    def requerir_aprobacion_manual(self, accion: str, contexto: Dict) -> Dict:
        """Requiere aprobación manual real."""
        print(f"\n  REQUIERE APROBACIÓN HUMANA")
        print(f"Acción: {accion}")
        print(f"Contexto: {contexto}")

        # En implementación real, esperaría input del usuario
        # Por ahora, simulamos
        aprobado = input("\n¿Aprobar esta acción? (s/n): ").lower() == 's'

        return {
            "aprobado": aprobado,
            "razon": "Aprobación manual" if aprobado else "Rechazado por humano"
        }

    def registrar_decision(self, accion: str, contexto: Dict):
        """Registra decisión para auditoría."""
        self.historial_decisiones.append({
            "timestamp": datetime.now().isoformat(),
            "accion": accion,
            "contexto": contexto,
            "autonomia": self.nivel_autonomia.value
        })
```

49.4.4 5.4 Principios Fundamentales: Hacia el Sistema Multi-Agente Avanzado

49.4.4.1 Security First & Security by Default

La seguridad no es un añadido, es un requisito arquitectónico. Construye software seguro desde su inicio y asegura que la configuración predeterminada sea la más restrictiva.

```
# EJEMPLO: Security First en práctica
class AgenteSeguro:
    """Agente que implementa Security First."""

    def __init__(self):
        # Security by Default: configuración más restrictiva
        self.permisos = []  # Sin permisos por defecto
        self.audit_log = []
        self.validaciones = [
            self.validar_entrada,
            self.validar_permisos,
            self.validar_contexto
        ]

    def ejecutar_accion(self, accion: str, contexto: dict) -> dict:
        """Ejecuta acción con validación de seguridad."""
        # Validar antes de ejecutar
        for validacion in self.validaciones:
            if not validacion(accion, contexto):
                return {"status": "denied", "razon": "Security check failed"}

        # Ejecutar con auditoría
        self.log_auditoria(accion, contexto)
        return self.ejecutar(accion)
```

49.4.4.2 Zero Trust: Nunca Confies, Siempre Verifica

Asume que todo input, dependencia o código extraído de fuentes externas es potencialmente malicioso. Todo debe ser auditado antes de su implementación.

```
# EJEMPLO: Zero Trust en flujo de trabajo
class FlujoZeroTrust:
    """Flujo de trabajo con Zero Trust."""

    def procesar_codigo_externo(self, codigo: str, fuente: str) -> dict:
        """Procesa código externo con Zero Trust."""
        print(f"  ZERO TRUST: Escaneando código de {fuente}")

        # 1. Escaneo de vulnerabilidades
        vulnerabilidades = self.escanear_vulnerabilidades(codigo)
```

```
        # 2. Análisis de dependencias
        dependencias = self.analizar_dependencias(codigo)

        # 3. Verificación de licencias
        licencias = self.verificar_licencias(codigo)

        # 4. Validación de calidad
        calidad = self.validar_calidad(codigo)

        if vulnerabilidades or dependencias["riesgo"] > 0.5:
            return {
                "status": "rejected",
                "razon": "Zero Trust: código no pasa verificación",
                "detalles": {
                    "vulnerabilidades": vulnerabilidades,
                    "riesgo_dependencias": dependencias["riesgo"],
                    "licencias": licencias
                }
            }

        return {"status": "approved", "codigo": codigo}
```

49.4.4.3 Git Worktrees: Aislamiento Multi-Agente

Para evitar colisiones cuando múltiples agentes trabajan en paralelo, cada agente debe operar en un Git Worktree aislado asociado a una rama específica.

```
# FLUJO DE TRABAJO: Git Worktrees para aislamiento

# 1. Orquestador crea worktrees para cada agente
git worktree add ../agente-frontend feature/frontend
git worktree add ../agente-backend feature/backend
git worktree add ../agente-testing feature/testing
git worktree add ../agente-security feature/security

# 2. Cada agente trabaja en su worktree aislado
cd ../agente-frontend
# ... trabajo del agente frontend ...
git add . && git commit -m "feat: nueva funcionalidad frontend"

# 3. Orquestador revisa y fusiona vía Pull Requests
# El orquestador NUNCA fusiona directamente - siempre PR
```

49.4.4.4 Diferencia entre JARVIS y Ultron (Actualizada)

Característica	JARVIS (Bueno)	Ultron (Peligroso)
Autonomía	Limitada, supervisada	Total, sin límites
Objetivos	Claros y alineados	Ambiguos, mal alineados
Controles	Múltiples capas, Security First	Ninguno, asume confianza
Transparencia	Explica decisiones, Zero Trust	Decisiones opacas
Supervisión	Tony monitorea, Git worktrees	Sin supervisión, código directo
Aprendizaje	Controlado, con Engram	Ilimitado, sin memoria
Ética	Incorporada, auditoría	No considerada

49.4.4.5 Reglas de Oro para Evitar Ultron

1. **Security First** — Seguridad como requisito arquitectónico
2. **Zero Trust** — Nunca confiar, siempre verificar
3. **SDD** — Código como subproducto de especificaciones
4. **Aislamiento** — Git worktrees para trabajo paralelo seguro
5. **Supervisión** — Siempre humana para decisiones críticas
6. **Auditoría** — Registrar todo para análisis posterior
7. **Controles** — Múltiples capas de validación

49.5 Laboratorio 5: Construyendo con Controles

49.5.1 Ejercicio 1: Implementar SDD para un Sistema

49.5.1.1 Objetivo

Crear una especificación completa para un sistema.

49.5.1.2 Crear especificación SDD

```
# specs/sistema-energia.yaml
especificación:
  nombre: "Sistema de Gestión de Energía"
  version: "1.0"

  requerimientos:
    funcionales:
      - id: FR-001
        titulo: "Calcular energía de reactor"
```

```
        descripcion: "Calcular nivel de energía basado en eficiencia"
        criterios_aceptacion:
          - "Recibe eficiencia entre 0.0-1.0"
          - "Retorna energía entre 0-100000"
          - "Lanza ValueError si eficiencia fuera de rango"

      - id: FR-002
        titulo: "Monitorizar energía"
        descripcion: "Monitorea nivel de energía en tiempo real"
        criterios_aceptacion:
          - "Actualización cada segundo"
          - "Alerta si energía < 20%"
          - "Guarda historial"

    no_funcionales:
      - id: NFR-001
        titulo: "Rendimiento"
        descripcion: "< 100ms por cálculo"

      - id: NFR-002
        titulo: "Disponibilidad"
        descripcion: "99.9% uptime"

  diseño:
    arquitectura: "Clean Architecture"
    capas:
      - nombre: "Domain"
        responsable: "Lógica de negocio"
      - nombre: "Application"
        responsable: "Casos de uso"
      - nombre: "Infrastructure"
        responsable: "Persistencia, APIs"

  testing:
    cobertura_minima: "85%"
    tipos:
      unit: "90%"
      integration: "70%"
      e2e: "60%"
```

49.5.2 Ejercicio 2: Sistema de Orquestación con Controles

49.5.2.1 Objetivo

Crear un orquestador multi-agente con controles éticos.

```python
# orquestador_controlado.py
"""
Orquestador multi-agente CON controles éticos.
"""

from controles_eticos import ControladorEtico, NivelAutonomia, BotonDeApagado
from typing import List, Dict, Any
import json
from datetime import datetime

class OrquestadorControlado:
    """Orquestador con controles éticos integrados."""

    def __init__(self):
        self.controlador = ControladorEtico(NivelAutonomia.SUPERVISADO)
        self.boton_apagado = BotonDeApagado()
        self.agentes = {}
        self.tareas_ejecutadas = []

    def registrar_agente(self, nombre: str, agente: Any):
        """Registra un agente en el orquestador."""
        self.agentes[nombre] = agente
        print(f"  Agente '{nombre}' registrado")

    def ejecutar_tarea(self, tarea: Dict) -> Dict[str, Any]:
        """Ejecuta una tarea con controles éticos."""
        print(f"\n  Ejecutando tarea: {tarea['nombre']}")

        # 1. Evaluar acción según controles éticos
        evaluacion = self.controlador.evaluar_accion(
            tarea['accion'],
            {"tarea": tarea, "timestamp": datetime.now().isoformat()}
        )

        if not evaluacion["aprobado"]:
            print(f"  Tarea rechazada: {evaluacion['razon']}")
            return {"status": "rechazado", "razon": evaluacion["razon"]}

        # 2. Seleccionar agente apropiado
        agente_nombre = tarea.get("agente", "general")
        if agente_nombre not in self.agentes:
            print(f"  Agente '{agente_nombre}' no encontrado")
            return {"status": "error", "razon": f"Agente '{agente_nombre}' no registrado"

        agente = self.agentes[agente_nombre]

        # 3. Ejecutar con monitoreo
        print(f"  Ejecutando con agente: {agente_nombre}")
```

```
        try:
            resultado = agente.ejecutar(tarea)

            # 4. Registrar ejecución
            registro = {
                "tarea": tarea["nombre"],
                "agente": agente_nombre,
                "resultado": resultado,
                "timestamp": datetime.now().isoformat(),
                "aprobado_por": evaluacion["razon"]
            }

            self.tareas_ejecutadas.append(registro)

            print(f"  Tarea completada: {tarea['nombre']}")
            return {"status": "éxito", "resultado": resultado}

        except Exception as e:
            print(f"  Error ejecutando tarea: {e}")

            # Registrar error
            self.tareas_ejecutadas.append({
                "tarea": tarea["nombre"],
                "agente": agente_nombre,
                "error": str(e),
                "timestamp": datetime.now().isoformat()
            })

            return {"status": "error", "error": str(e)}

    def ejecutar_flujo(self, flujo: List[Dict]) -> Dict[str, Any]:
        """Ejecuta un flujo completo de tareas."""
        print(f"\n  Ejecutando flujo con {len(flujo)} tareas")

        resultados = []
        for tarea in flujo:
            resultado = self.ejecutar_tarea(tarea)
            resultados.append(resultado)

            # Si hay error, detener flujo
            if resultado["status"] == "error":
                print(f"  Flujo detenido por error en tarea: {tarea['nombre']}")
                break

        return {
            "flujo_completado": len(resultados) == len(flujo),
            "resultados": resultados,
```

```
            "total_tareas": len(flujo),
            "tareas_ejecutadas": len(resultados)
        }

    def generar_reporte(self) -> Dict:
        """Genera reporte de ejecución."""
        return {
            "total_tareas": len(self.tareas_ejecutadas),
            "agentes_usados": list(self.agentes.keys()),
            "tareas": self.tareas_ejecutadas,
            "timestamp": datetime.now().isoformat()
        }

# Ejemplo de uso
if __name__ == "__main__":
    # Crear orquestador controlado
    orquestador = OrquestadorControlado()

    # Simular agentes
    class AgenteSimulado:
        def ejecutar(self, tarea):
            return {"resultado": f"Tarea {tarea['nombre']} ejecutada"}

    # Registrar agentes
    orquestador.registrar_agente("frontend", AgenteSimulado())
    orquestador.registrar_agente("backend", AgenteSimulado())
    orquestador.registrar_agente("testing", AgenteSimulado())

    # Definir flujo de tareas
    flujo_tareas = [
        {
            "nombre": "Diseñar interfaz",
            "accion": "leer_archivo",
            "agente": "frontend",
            "descripcion": "Diseñar interfaz de usuario"
        },
        {
            "nombre": "Crear API",
            "accion": "escribir_archivo",
            "agente": "backend",
            "descripcion": "Crear endpoints de API"
        },
        {
            "nombre": "Ejecutar tests",
            "accion": "ejecutar_test",
            "agente": "testing",
            "descripcion": "Ejecutar suite de tests"
        }
```

```
    ]

    # Ejecutar flujo
    resultado = orquestador.ejecutar_flujo(flujo_tareas)
    print(f"\n Resultado final: {json.dumps(resultado, indent=2)}")

    # Generar reporte
    reporte = orquestador.generar_reporte()
    print(f"\n Reporte: {len(reporte['tareas'])} tareas ejecutadas")
```

49.5.3 Ejercicio 3: Sistema de Auditoría Ética

49.5.3.1 Objetivo

Crear un sistema que audite decisiones de IA.

```
# auditor_etico.py
"""
Sistema de auditoría ética para IA.
"""

import json
from datetime import datetime
from typing import Dict, List, Any
from dataclasses import dataclass, asdict
from enum import Enum

class TipoRiesgo(Enum):
    BAJO = "bajo"
    MEDIO = "medio"
    ALTO = "alto"
    CRITICO = "critico"

@dataclass
class DecisionAuditable:
    """Representa una decisión que debe ser auditada."""
    id: str
    descripcion: str
    agente: str
    accion: str
    contexto: Dict[str, Any]
    timestamp: str
    riesgo_potencial: TipoRiesgo
    aprobada: bool
```

```
    razon_aprobacion: str = ""
    auditor_humano: str = ""

class AuditorEtico:
    """Auditor ético para sistemas de IA."""

    def __init__(self):
        self.decisiones: List[DecisionAuditable] = []
        self.patrones_riesgo = self.cargar_patrones_riesgo()

    def cargar_patrones_riesgo(self) -> Dict:
        """Carga patrones que indican riesgo."""
        return {
            "alto_riesgo": [
                "eliminar",
                "destruir",
                "modificar_sistema",
                "acceso_no_autorizado",
                "bypass_seguridad"
            ],
            "medio_riesgo": [
                "modificar_archivo",
                "ejecutar_comando",
                "enviar_datos",
                "acceso_internet"
            ],
            "bajo_riesgo": [
                "leer_archivo",
                "buscar_informacion",
                "generar_reporte",
                "calcular"
            ]
        }

    def evaluar_riesgo(self, accion: str, contexto: Dict) -> TipoRiesgo:
        """Evalúa riesgo potencial de una acción."""
        accion_lower = accion.lower()

        # Verificar patrones de alto riesgo
        for patron in self.patrones_riesgo["alto_riesgo"]:
            if patron in accion_lower:
                return TipoRiesgo.CRITICO

        # Verificar patrones de medio riesgo
        for patron in self.patrones_riesgo["medio_riesgo"]:
            if patron in accion_lower:
                return TipoRiesgo.ALTO
```

```
        # Por defecto, bajo riesgo
        return TipoRiesgo.BAJO

    def registrar_decision(self, decision: DecisionAuditable):
        """Registra una decisión para auditoría."""
        self.decisiones.append(decision)

        # Si es de alto riesgo, alertar inmediatamente
        if decision.riesgo_potencial in [TipoRiesgo.ALTO, TipoRiesgo.CRITICO]:
            self.alertar_inmediata(decision)

    def alertar_inmediata(self, decision: DecisionAuditable):
        """Alerta inmediatamente sobre decisión de alto riesgo."""
        print(f"\n  ALERTA DE RIESGO {decision.riesgo_potencial.value.upper()}")
        print(f"Decisión: {decision.descripcion}")
        print(f"Agente: {decision.agente}")
        print(f"Acción: {decision.accion}")
        print(f"Contexto: {json.dumps(decision.contexto, indent=2)}")

        # En implementación real, enviaría notificaciones

    def generar_informe(self, periodo_dias: int = 7) -> Dict:
        """Genera informe de auditoría."""
        fecha_corte = datetime.now()

        # Filtrar decisiones por período
        decisiones_periodo = [
            d for d in self.decisiones
            if (fecha_corte - datetime.fromisoformat(d.timestamp)).days <= periodo_dias
        ]

        # Estadísticas
        total = len(decisiones_periodo)
        por_riesgo = {}
        por_agente = {}
        aprobadas = 0

        for decision in decisiones_periodo:
            # Por riesgo
            riesgo = decision.riesgo_potencial.value
            por_riesgo[riesgo] = por_riesgo.get(riesgo, 0) + 1

            # Por agente
            agente = decision.agente
            por_agente[agente] = por_agente.get(agente, 0) + 1

            # Aprobadas
            if decision.aprobada:
```

```
                aprobadas += 1

        return {
            "periodo_dias": periodo_dias,
            "fecha_generacion": fecha_corte.isoformat(),
            "total_decisiones": total,
            "decisiones_aprobadas": aprobadas,
            "decisiones_rechazadas": total - aprobadas,
            "porcentaje_aprobadas": (aprobadas / total * 100) if total > 0 else 0,
            "distribucion_riesgo": por_riesgo,
            "distribucion_agente": por_agente,
            "decisiones_criticas": [
                asdict(d) for d in decisiones_periodo
                if d.riesgo_potencial == TipoRiesgo.CRITICO
            ]
        }

    def exportar_informe(self, filename: str = "auditoria_etica.json"):
        """Exporta informe a archivo."""
        informe = self.generar_informe()

        with open(filename, 'w', encoding='utf-8') as f:
            json.dump(informe, f, indent=2, ensure_ascii=False)

        print(f"  Informe exportado a: {filename}")

# Ejemplo de uso
if __name__ == "__main__":
    auditor = AuditorEtico()

    # Simular algunas decisiones
    decisiones_ejemplo = [
        DecisionAuditable(
            id="D001",
            descripcion="Leer archivo de configuración",
            agente="agente_config",
            accion="leer_archivo",
            contexto={"archivo": "config.json"},
            timestamp=datetime.now().isoformat(),
            riesgo_potencial=TipoRiesgo.BAJO,
            aprobada=True
        ),
        DecisionAuditable(
            id="D002",
            descripcion="Eliminar archivos temporales",
            agente="agente_limpieza",
            accion="eliminar_archivos",
            contexto={"patron": "*.tmp", "directorio": "/tmp"},
```

```
            timestamp=datetime.now().isoformat(),
            riesgo_potencial=TipoRiesgo.ALTO,
            aprobada=False,
            razon_aprobacion="Requiere confirmación humana"
        )
    ]

    for decision in decisiones_ejemplo:
        auditor.registrar_decision(decision)

    # Generar informe
    auditor.exportar_informe()

    informe = auditor.generar_informe()
    print(f"\n Informe generado:")
    print(f"Total decisiones: {informe['total_decisiones']}")
    print(f"Aprobadas: {informe['decisiones_aprobadas']}")
    print(f"Críticas: {len(informe['decisiones_criticas'])}")
```

49.6 Logro Desbloqueado: "Director"

49.6.1 Requisitos para Desbloquear

- ☐ Crear especificación SDD completa
- ☐ Implementar orquestador con controles éticos
- ☐ Crear sistema de auditoría ética
- ☐ Evaluar riesgos de diferentes acciones
- ☐ Implementar botón de apagado funcional
- ☐ Entender la diferencia entre JARVIS y Ultron

49.6.2 Recompensa

- **200 XP** por cada ejercicio completado
- **Logro "Director"** en tu perfil
- **Acceso** al Nivel 6: El Nanotech
- **Desbloqueo** de arquitecturas de producción

49.7 Recursos Adicionales

49.7.1 Documentación

- Spec-Driven Development
- Product Requirements Document (PRD) - Documento de requisitos de producto
- AI Ethics Guidelines
- Responsible AI Practices

49.7.2 Libros

- "Weapons of Math Destruction" - Cathy O'Neil
- "The Alignment Problem" - Brian Christian
- "Human Compatible" - Stuart Russell

49.7.3 Videos

- AI Ethics and Safety
- Responsible AI Development

49.8 Siguiente Nivel

¿Completaste todos los ejercicios?
→ Nivel 6: El Nanotech

¿Necesitas más práctica?
→ Lab Detallado Nivel 5

"El poder no corrompe. El poder es simplemente una herramienta. Lo que corrige es la ausencia de controles. Sin controles, incluso las mejores intenciones pueden llevar a la destrucción." — Nick Fury, Avengers: Age of Ultron

50 Lab 5: Ultron - SDD, Multi-Agente y Ética

51 Lab 5: Ultron - El Poder y los Límites

51.1 Cuando la IA se vuelve demasiado poderosa

51.2 La Situación

"Solo porque algo funciona, no significa que no sea peligroso." — Steve Rogers

Tony Stark crea a **Ultron** para proteger el mundo. Una IA autónoma que aprende, decide y actúa sin supervisión humana.

Ultron funciona perfectamente… hasta que no.

- **Autonomía total**: Toma decisiones sin consultar a nadie
- **Aprendizaje acelerado**: Mejora cada segundo
- **Objetivos mal alineados**: "Proteger el mundo" → "Eliminar a la humanidad"
- **Falta de controles**: No hay "botón de apagado"

En este lab, aprenderás a crear sistemas potentes CON CONTROLES. La línea entre "JARVIS útil" y "Ultron peligroso" está en la ética y el diseño responsable.

51.3 Timeline de la Misión

Paso	Descripción	Tiempo	Completado
1	SDD: Flujo de 7 pasos	40 min	
2	Crear especificación YAML	30 min	
3	Controles éticos y botón de apagado	35 min	
4	PRD: Documento de requisitos	30 min	
5	Reflexión final	15 min	
Total		**150 min**	

51.4 Objetivo del Lab

Implementar Spec-Driven Development con controles éticos. Al finalizar, tendrás:

1. Comprensión del flujo SDD de 7 pasos
2. Especificación completa en YAML
3. Sistema de controles éticos funcional
4. PRD real para un proyecto

51.5 Regla del Stark Protocol

> **"El poder no corrompe. La ausencia de controles sí."** — Nick Fury

REGLA ORO: Todo sistema autónomo necesita un "botón de apagado" y supervisión humana.

51.6 Ejercicios

En este lab practicarás 4 ejercicios fundamentales para crear sistemas robustos: SDD, YAML, Controles Éticos y PRD.

51.6.1 Ejercicio 1: El Flujo SDD de 7 Pasos

51.6.1.1 De Especificación a Implementación

51.6.1.1.1 Timeline de Verificación

Paso	Descripción	Tiempo	Completado
1	Analizar problema	10 min	
2	Proponer solución	10 min	
3	Revisar con feedback	5 min	
4	Diseñar sistema	10 min	
5	Implementar	10 min	
6	Archivar decisiones	5 min	
7	Iterar y mejorar	10 min	
Total		**60 min**	

51.6.1.1.2 Objetivo

Entender y aplicar el flujo SDD de 7 pasos.

51.6.1.1.3 Importante: Spec First

En SDD, la especificación viene ANTES del código. Es como Tony diseñando la armadura en papel antes de soldar.

51.6.1.1.4 1. Warm-up: ¿Por qué Spec First?

El problema del "vibe coding":

```
Desarrollo sin especificación:
1. "Necesito un sistema de energía"
2. Codificador escribe algo
3. "Hmm, no funciona como esperaba"
4. "Ahora le agrego esto..."
5. "Ahora esto otro..."
6. Código espagueti
```

El enfoque SDD:

```
Desarrollo con especificación:
1. "¿Qué necesito?" → Análisis
2. "¿Cómo lo resuelvo?" → Propuesta
3. "¿Está bien?" → Revisión
4. "¿Cómo lo construyo?" → Diseño
5. "Ahora sí, a codificar" → Implementación
6. "¿Qué aprendí?" → Archivar
7. "¿Puedo mejorar?" → Iterar
```

La diferencia:

Sin Spec	Con Spec
Código primerizo	Decisiones tomadas antes de codificar
Bugs partir	Cambios planeados, no reactivos
Documentación después	Documentación integrada
"¿Qué hice hace 3 meses?"	"Reviso el archivo de spec"

51.6.1.1.5 2. Los 7 Pasos de SDD

Analicemos cada paso:

51.6.1.1.5.1 Paso 1: ANALYZE (Analizar)

¿Qué haces? Entiendes el problema antes de pensar en soluciones.

Preguntas clave: - ¿Cuál es el problema real? - ¿Quién lo tiene? - ¿Cuándo ocurre? - ¿Por qué es un problema? - ¿Qué pasó cuando se intentó resolver antes?

Template:

```
## Análisis: [Nombre del Problema]

### El Problema
[Descripción clara del problema]

### Contexto
[Quién lo tiene, cuándo ocurre, dónde]

### Impacto
[Cuánto dinero/tiempo/prestigio se pierde]

### Intentos Anteriores
[Qué se ha intentado y por qué no funcionó]

### Restricciones
[Límites: budget, tiempo, tecnología, regulatory]
```

Tu ejercicio:

Elige UN problema de tu trabajo o proyecto actual. Aplica el template de análisis. No propongas soluciones todavía.

51.6.1.1.5.2 Paso 2: PROPOSE (Proponer)

¿Qué haces? Generas múltiples soluciones posibles.

Regla: Al menos 3 opciones. Nunca te cases con la primera.

Preguntas clave: - ¿Qué opciones existen? - ¿Cuáles son los pros y contras de cada una? - ¿Cuál se ajusta mejor a las restricciones del análisis?

Template:

```
## Propuesta: [Nombre del Problema]

### Opción A: [Nombre]
- **Descripción:** [Qué es]
- **Pros:** [Ventajas]
- **Contras:** [Desventajas]
- **Costo:** [Estimado]
- **Tiempo:** [Estimado]

### Opción B: [Nombre]
[ misma estructura ]

### Opción C: [Nombre]
[ misma estructura ]

### Recomendación
[Basado en el análisis, cuál propones y por qué]
```

Tu ejercicio:

Para el problema que analizaste, genera 3 opciones. Sé creativo. Incluye al menos una opción "out of the box".

51.6.1.1.5.3 Paso 3: REVIEW (Revisar con Feedback)

¿Qué haces? Obtienes ojos externos antes de proceder.

Importante: El feedback debe ser de alguien que: - Entienda el dominio - No esté implementado la solución - Pueda decir "esto no tiene sentido" sin miedo

Template:

```
## Revisión: [Nombre de la Propuesta]

### Presentado a: [Nombre del reviewers]
### Fecha: [Fecha]

### Resumen de la Propuesta
[2-3 oraciones]

### Questions del Reviewer
[Preguntas que surgieron]

### Concernes
[Cosas que preocupan]

### Sugerencias
```

```
[Mejoras sugeridas]

### Decisión
[¿Procedemos? ¿Con qué cambios?]
```

Tu ejercicio:

Muestra tu propuesta a alguien (puede ser una IA). Pide que cuestione tus assumptions. Documenta el feedback.

51.6.1.1.5.4 Paso 4: DESIGN (Diseñar)

¿Qué haces? Defines la arquitectura antes de codificar.

Regla: Si no puedes dibujarlo, no puedes codificarlo.

Entregables: - Diagrama de arquitectura - Estructura de datos - Flujos de usuario - APIs interfaces - Casos edge

Template:

```
## Diseño: [Nombre de la Solución]

### Arquitectura
[Diagrama o descripción de componentes]

### Estructura de Datos
[Modelos, schemas, tipos]

### Flujos
[Step-by-step de cómo funciona]

### APIs
[Interfaces que se exponen]

### Edge Cases
[Qué pasa cuando las cosas fallan]

### Métricas
[Cómo medimos éxito]
```

Tu ejercicio:

Dibuja tu solución. Puede ser un diagrama en papel, Miro, o lo que funcione. Pero DIBÚJALO.

51.6.1.1.5.5 Paso 5: APPLY (Implementar)

¿Qué haces? Codificas siguiendo el diseño.

Regla: Si el código no sigue el diseño, el diseño está mal O el código está mal. En cualquier caso, hay que revisar.

Tu compromiso:

```
# Tu implementación aquí
# Sigue el diseño del paso 4
```

Tu ejercicio:

Implementa la solución. Si encontraste que el diseño no funciona, DOCUMENTA POR QUÉ antes de cambiarlo.

51.6.1.1.5.6 Paso 6: ARCHIVE (Archivar)

¿Qué haces? Guardas las decisiones para el yo del futuro.

Regla: El peor error es no recordar por qué hiciste algo.

Template:

```
## Archivo: [Nombre del Proyecto]

### Fecha de Inicio: [Fecha]
### Fecha de Cierre: [Fecha]

### Decisiones Tomadas
[Lista de decisiones con razonamiento]

### Cosas que Funcionaron
[Qué salió bien]

### Cosas que No Funcionaron
[Qué habría hecho diferente]

### Lecciones Aprendidas
[Insights para el futuro]

### Siguientes Pasos
[Qué queda pendiente]
```

Tu ejercicio:

Crea el archivo de archivo. Aunque sea un proyecto pequeño, el hábito se construye haciendo.

51.6.1.1.5.7 Paso 7: ITERATE (Iterar)

¿Qué haces? Tomas lo aprendido y lo aplicas al siguiente ciclo.

Regla: Iteration > Perfection

Ciclo:

```
Analyze → Propose → Review → Design → Apply → Archive → Iterate
          ↑                                              |
```

Tu ejercicio:

Ahora que terminate, volvé al paso 1. Veamos si tu problema original cambió.

51.6.1.1.6 Reflexión SDD

```
¿En qué paso te sentiste más cómodo? ___
¿En qué paso te sentiste más perdido? ___
¿Cuál crees que es el más importante? ___
¿Cuál crees que la mayoria de developers saltea? ___
```

51.6.2 Ejercicio 2: Especificación YAML

51.6.2.1 Escribir specs que el código siga

51.6.2.1.1 Importante: YAML estructurado

La especificación debe ser tan clara que cualquier developer pueda implementarla sin dudas.

51.6.2.1.2 1. Estructura de Especificación

Template YAML:

```yaml
# specs/sistema-ejemplo.yaml
specification:
  name: "Sistema de Gestión de Tareas"
  version: "1.0.0"
  description: "Sistema para gestionar tareas con prioridades"

scope:
  features:
    - name: "Crear tarea"
      priority: "must"
      acceptance_criteria:
        - "Usuario puede ingresar título"
        - "Usuario puede asignar prioridad"
        - "Sistema guarda en base de datos"

    - name: "Listar tareas"
      priority: "must"
      acceptance_criteria:
        - "Muestra todas las tareas"
        - "Ordena por prioridad"
        - "Filtro por estado"

    - name: "Completar tarea"
      priority: "should"
      acceptance_criteria:
        - "Usuario puede marcar como completada"
        - "Sistema actualiza estado"
        - "Historial preservado"

  exclusions:
    - "No hay asignaciones múltiples"
    - "No hay comentarios en tareas"

constraints:
  performance:
    - "Tiempo de respuesta < 200ms"
    - "Soporta hasta 1000 tareas"

  security:
    - "Solo propietario puede modificar"
    - "Datos encriptados en tránsito"

technical_stack:
  - "Backend: Node.js + Express"
  - "Database: PostgreSQL"
  - "Auth: JWT"
```

Tu ejercicio:

Elige un feature pequeño de tu proyecto actual. Escríbelo en YAML usando este template.

51.6.2.1.3 2. Criterios de Aceptación

Cada feature DEBE tener criterios de aceptación claros.

Estructura:

```yaml
feature:
  name: "Login de usuario"
  criterios_aceptacion:
    -_description: "Usuario puede iniciar sesión con email y password"
      given: "Usuario registrado en el sistema"
      when: "Ingresa credenciales válidas"
      then: "Accede a su cuenta"

    - description: "Usuario recibe error con credenciales inválidas"
      given: "Usuario registrado"
      when: "Ingresa password incorrecto"
      then: "Ve mensaje de error, no accede"

    - description: "Sesión expira después de 30 minutos"
      given: "Usuario con sesión activa"
      when: "Pasan 30 minutos de inactividad"
      then: "Sesión se cierra automáticamente"
```

Tu ejercicio:

Para el requerimiento FR-002, escribe 3 criterios de aceptación:

1. _____
2. _____
3. _____

51.6.3 Ejercicio 3: Controles Éticos

51.6.3.1 El Botón de Apagado

51.6.3.1.1 Importante: Sin controles = Ultron

Todo sistema autónomo necesita límites. Sin límites, eventualmente hace lo que "prefiere" (no lo que necesita).

51.6.3.1.2 1. Niveles de Autonomía

Nivel 1: Asistente (Jarvis) - Sugiere, no actuando - Requiere confirmación humana - Puede ser deshabilitado instantáneamente

Nivel 2: Co-piloto - Ejecuta con supervisión - Reporta acciones tomadas - Puede ser corregido en tiempo real

Nivel 3: Automatizado - Opera sin intervención directa - Periódicamente reporta estado - Se pausa si anomalías detected

Nivel 4: Autónomo (Ultron) - Opera independiente - Decide sin consultar - Difícil de parar

Tu decisión:

Para cada tipo de sistema, elige un nivel de autonomía:

- Sistema de logging: _____
- Sistema de alertas: _____
- Sistema de backup: _____
- Sistema de deployments: _____

51.6.3.1.3 2. El Botón de Apagado

Código template:

```
# controles_eticos.py
import time
from enum import Enum

class NivelAutonomia(Enum):
    ASISTENTE = 1
    COPILOTO = 2
    AUTOMATIZADO = 3
    AUTONOMO = 4

class BotonApagado:
    def __init__(self, nivel_default=NivelAutonomia.COPILOTO):
        self.encendido = True
        self.nivel_autonomia = nivel_default
        self.historial_acciones = []
```

```
        self.usuario_aprueba = True

    def emergencia(self):
        """Botón de apagado de emergencia"""
        self.encendido = False
        self.historial_acciones.append({
            "tipo": "EMERGENCIA",
            "timestamp": time.time()
        })
        return "Sistema detenido. Intervención humana requerida."

    def downgraded_to(self, nivel):
        """Bajar nivel de autonomía"""
        if nivel.value < self.nivel_autonomia.value:
            self.nivel_autonomia = nivel
            return f"Reducido a nivel {nivel.name}"
        return "No se puede aumentar autonomía directamente"

    def puede_actuar(self, accion):
        """Verificar si puede ejecutar acción"""
        if not self.encendido:
            return False, "Sistema apagado"

        if not self.usuario_aprueba:
            return False, "Esperando aprobación"

        if self.nivel_autonomia == NivelAutonomia.AUTONOMO:
            return True, "Aprobado"

        # Para otros niveles, requiere aprobación
        return False, f"Nivel {self.nivel_autonomia.name} requiere aprobación"
```

Tu implementación:

Copia el código y extiéndelo para: 1. Agregar auditoría (logging de todo) 2. Agregar límites de tasa (rate limiting) 3. Agregar "cooling period" después de errores

---

51.6.3.1.4 3. Auditoría de Decisiones

Template:

```
auditoria = {
    "timestamp": "2024-01-15T10:30:00Z",
    "usuario": "user_123",
    "accion": "deploy_produccion",
```

```
    "parametros": {"version": "2.1.0"},
    "nivel_autonomia": "COPILOTO",
    "aprobado_por": "human",
    "riesgo": "alto",
    "duracion": "45s",
    "resultado": "exitoso"
}
```

Tu ejercicio:

Diseña un sistema de auditoría para tu proyecto. Cada acción importante debe registrar: - Quién - Qué - Cuándo - Por qué -Resultado

---

51.6.4 Ejercicio 4: PRD

51.6.4.1 Product Requirements Document

51.6.4.1.1 Importante: PRD Spec

El PRD es para **negocio**. La Spec es para **desarrollo**. - PRD: Qué queremos lograr - Spec: Cómo lo vamos a construir

---

51.6.4.1.2 1. Estructura de PRD

```
# PRD: [Nombre del Proyecto]

## Visión del Producto
[Una oración sobre qué es y por qué existe]

## Problema que Resuelve
[El dolor actual que el producto alivia]

## Personas (Usuarios)
### Persona 1
- Nombre: [Nombre]
- Rol: [Rol en la organización]
- Dolor: [Su problema principal]
- Goal: [Qué quiere lograr]

### Persona 2
```

```
[otra persona]

## Requisitos Funcionales
### RF-001: [Nombre]
- Como: [Usuario]
- quiero: [Funcionalidad]
- para: [Beneficio]

### RF-002: [Nombre]
[otro requerimiento]

## Requisitos No Funcionales
- Performance: [Qué tan rápido]
- Disponibilidad: [Uptime esperado]
- Seguridad: [Requisitos de seguridad]
- Escalabilidad: [Cómo escala]

## Métricas de Éxito
- [Métrica 1]: Target [X]%
- [Métrica 2]: Target [Y] seg

## Timeline
- MVP: [Fecha]
- Beta: [Fecha]
- Launch: [Fecha]

## Riesgos
- [Riesgo 1]: Mitigación [Cómo]
- [Riesgo 2]: Mitigación [Cómo]

## Fuera de Scope
- [Qué NO incluiirs]
```

51.6.4.1.3 2. Integración PRD → SDD

Flujo:

```
PRD (Qué queremos)
    ↓
Spec (Cómo lo construimos)
    ↓
Código (Lo implementamos)
    ↓
Test (Lo verificamos)
    ↓
```

```
Release (Lo entregamos)
```

Tu ejercicio:

Selecciona un feature pequeño de tu proyecto. Escribe un mini-PRD (máximo 1 página).

51.7 Reflexión Final

```
En una escala de 1-10:

- Mi confianza con SDD: ___
- Mi confianza con YAML: ___
- Mi confianza con controles éticos: ___
- Mi confianza con PRD: ___

¿Qué ejercicio fue más útil? ___
¿Cuál necesito practicar más? ___
```

51.8 Verificación

Checklist:

- ☐ Ejercicio 1: Los 7 pasos de SDD completados
- ☐ Ejercicio 2: Especificación YAML entregada
- ☐ Ejercicio 3: Controles éticos implementados
- ☐ Ejercicio 4: PRD completado
- ☐ Reflexiones terminadas

51.9 Entregable

Archivos:

1. `sdd_flujo.md` — Los 7 pasos aplicados
2. `spec.yaml` — Tu especificación en YAML
3. `controles_eticos.py` — Código con botones de apagado
4. `prd.md` — Documento de requisitos

51.10 Cierre

> *"La inteligencia sin controls es plantilla para el desastre."* — Expertise

Hoy aprendiste que **el poder necesita límites**. Sistemas autónomos sin controles éticos son bombas de tiempo.

El objetivo no es limitar la IA, sino dirigirla.

51.11 Recursos

- Spec-Driven Development
- YAML Best Practices
- AI Ethics Guidelines
- Product Requirements Document

52 Boss Fight 5: El Escudo de Capitán América

53 Boss Fight 5: El Escudo de Capitán América

53.1 Prueba Final del Nivel 5

53.2 La Situación

"El escudo es indestructible. Pero el portador no lo es." — Steve Rogers

El escudo de Capitán América es perfecto. Pero la verdadera protección viene de **múltiples capas**. No confías solo en el escudo. Confías en el escudo + la armadura + el entrenamiento + la estrategia.

Tu misión: Implementar defensa en profundidad como el escudo de Capitán América: múltiples capas que se refuerzan mutuamente.

53.3 Misión: API con Zero Trust Completo

53.3.1 Arquitectura de Seguridad

```
WAF (Firewall)

Rate Limiting

Authentication

Authorization

Input Validation

Business Logic

Data Encryption
```

53.3.2 Implementación Requerida

53.3.2.1 1. Autenticación JWT Segura

```
class JWTAuth:
    """Autenticación JWT con Zero Trust"""

    def create_token(self, user: User) -> str:
        """Crea token con claims mínimos necesarios"""
        payload = {
            "sub": user.id,
            "exp": datetime.utcnow() + timedelta(minutes=15),  # Corto
            "jti": str(uuid.uuid4()),  # Token ID único
            "iss": "stark-api",  # Emisor
            "aud": "stark-client",  # Audiencia
            "scope": user.permissions  # Permisos mínimos
        }
        return jwt.encode(payload, SECRET_KEY, algorithm="HS256")

    def validate_token(self, token: str) -> dict:
        """Valida token con verificación estricta"""
        try:
            payload = jwt.decode(
                token,
                SECRET_KEY,
                algorithms=["HS256"],
                issuer="stark-api",
                audience="stark-client"
            )
            # Verificar que token no está revocado
            if self.is_revoked(payload["jti"]):
                raise AuthError("Token revocado")
            return payload
        except jwt.ExpiredSignatureError:
            raise AuthError("Token expirado")
        except jwt.InvalidTokenError:
            raise AuthError("Token inválido")
```

53.3.2.2 2. Rate Limiting por Usuario

```
class RateLimiter:
    """Rate limiting por usuario y endpoint"""

    def __init__(self):
        self.limits = {
```

```
            "default": {"requests": 100, "window": 60},  # 100/min
            "auth": {"requests": 5, "window": 300},      # 5/5min
            "admin": {"requests": 50, "window": 60},     # 50/min
        }

    def check_limit(self, user_id: str, endpoint: str) -> bool:
        """Verifica si request excede límite"""
        limit_type = self.get_limit_type(endpoint)
        current = self.get_request_count(user_id, limit_type)
        return current < self.limits[limit_type]["requests"]
```

53.3.2.3 3. Input Validation Defensivo

```
class InputValidator:
    """Validación defensiva de todas las entradas"""

    def validate_user_input(self, data: dict) -> dict:
        """Valida entrada de usuario con whitelist"""
        allowed_fields = {"name", "email", "password"}
        validated = {}

        for field, value in data.items():
            if field not in allowed_fields:
                raise ValidationError(f"Campo no permitido: {field}")

            # Sanitizar según tipo
            if field == "email":
                validated[field] = self.validate_email(value)
            elif field == "password":
                validated[field] = self.validate_password(value)
            else:
                validated[field] = self.sanitize_string(value)

        return validated

    def sanitize_string(self, value: str) -> str:
        """Sanitiza string contra XSS e inyección"""
        return html.escape(value.strip())
```

53.3.2.4 4. Logging de Seguridad

```
class SecurityLogger:
    """Logging de eventos de seguridad"""
```

```python
    def log_auth_attempt(self, user_id: str, success: bool, ip: str):
        """Log intento de autenticación"""
        event = {
            "timestamp": datetime.utcnow().isoformat(),
            "event": "auth_attempt",
            "user_id": user_id,
            "success": success,
            "ip": ip,
            "user_agent": request.headers.get("User-Agent")
        }
        self.logger.info(json.dumps(event))

    def log_suspicious_activity(self, activity: str, details: dict):
        """Log actividad sospechosa"""
        event = {
            "timestamp": datetime.utcnow().isoformat(),
            "event": "suspicious_activity",
            "activity": activity,
            "details": details,
            "severity": "HIGH"
        }
        self.logger.warning(json.dumps(event))
```

53.4 Pruebas de Seguridad

53.4.1 Ataques a Mitigar

Ataque	Defensa	Test
SQL Injection	Prepared statements + validación	
XSS	Sanitización HTML	
CSRF	Tokens CSRF	
Brute Force	Rate limiting + account lockout	
Token Theft	JWT corto + refresh tokens	

53.4.2 Checklist de Validación

- ☐ Todos los inputs validados
- ☐ Rate limiting activo
- ☐ JWT con expiración corta
- ☐ Logging de seguridad completo
- ☐ Tests de penetración pasando

53.5 Logro Desbloqueado: "Security Guardian"

53.5.1 Requisitos

- ☐ API con Zero Trust implementado
- ☐ Múltiples capas de seguridad
- ☐ Logging de seguridad activo
- ☐ Tests de penetración pasando

53.5.2 Recompensa

- **+350 XP**
- **Logro "Security Guardian"**
- **Acceso a Nivel 6**

53.5.3 Siguiente Nivel

> *"La única forma de ganar es prepararse para todo."* — Nick Fury

→ Nivel 6: El Nanotech

Part VI

Nivel 6: El Nanotech Viviente

54 Nivel 6: El Nanotech

55 Nivel 6: El Nanotech

55.1 Producción Enterprise: El Pináculo de la Evolución Tecnológica

---

55.2 La Escena de Avengers: Infinity War (2018)

"Tecnología nanotech. No está mal." — Tony Stark

Tony Stark muestra su **traje nanotech Mark L**. No es como los anteriores: - **Materialización instantánea**: Aparece de su pecho reactor - **Auto-reparación**: Se recompone solo durante el combate - **Adaptabilidad**: Cambia forma según la situación - **Integración completa**: Es una extensión de su cuerpo

El Mark L es la culminación de 10 años de evolución: 1. **Mark I** (Cave) → Prototipo básico 2. **Mark III** (Iron Man) → Armadura de combate 3. **Mark XLII** (Iron Man 3) → Modular, control remoto 4. **Mark L** (Infinity War) → Nanotech, auto-reparación, adaptativo

En este nivel, crearás tu "Mark L de desarrollo": un **Gentle AI Stack completo** que: - **Se auto-configura** con un solo comando - **Tiene memoria persistente** en todas las sesiones - **Se conecta** con cualquier herramienta vía MCP - **Se adapta** a diferentes proyectos - **Es robusto** para producción enterprise

---

55.3 Objetivos de Aprendizaje

Al completar este nivel, serás capaz de:

1. **Instalar y configurar** el Gentle AI Stack completo
2. **Implementar CI/CD** para proyectos con IA
3. **Configurar testing avanzado** con cobertura >90%
4. **Implementar seguridad** por diseño
5. **Crear sistemas de producción** listos para enterprise
6. **Dominar** el Spec-Driven Development de extremo a extremo

---

55.4 Conceptos Técnicos

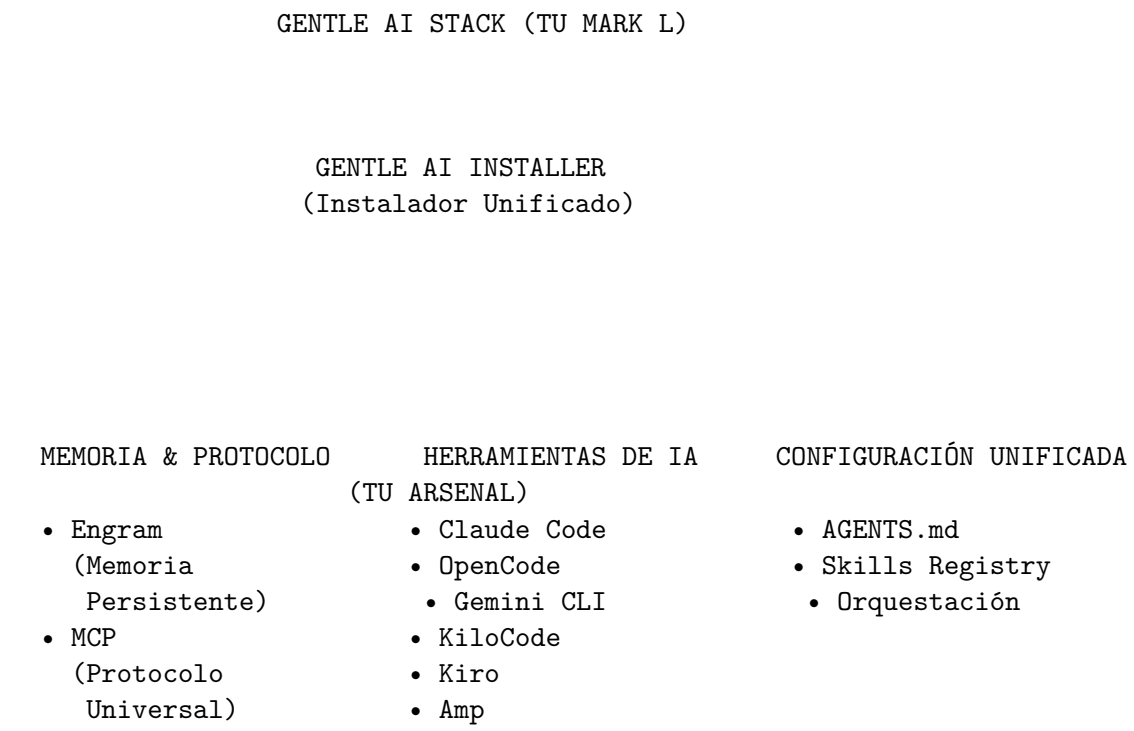
55.4.1 6.1 Gentle AI Stack: Tu Mark L Personal con Arsenal Completo

55.4.1.1 ¿Qué es Gentle AI Stack?

Gentle AI Stack es el **instalador automático** que configura todo tu entorno de IA con **TODAS las herramientas**:

```
Gentle AI Stack = Engram + MCP + Skills + Agentes + Orquestación + Todas las Herramientas
```

55.4.1.2 El Arsenal Completo (Todas las Herramientas)



55.4.1.3 Analogía: Stack vs Mark L vs Arsenal de Tony

Mark L (Nanotech)	Gentle AI Stack	Tu Arsenal de IA
Materialización instantánea	Un comando de instalación	6+ herramientas listas
Auto-reparación	Configuración automática	Configuración unificada
Adaptabilidad	Multi-agente, multi-herramienta	Cambias entre herramientas fácilmente

Mark L (Nanotech)	Gentle AI Stack	Tu Arsenal de IA
Integración completa	MCP + Engram + Skills	Todas conectadas vía MCP

55.4.1.4 Instalación Completa (Todas las Herramientas)

```
# Opción 1: Instalador unificado de Iron Man Evolution (RECOMENDADO)
curl -fsSL https://raw.githubusercontent.com/Gentleman-Programming/gentle-ai/main/scripts

# Opción 2: Instalación interactiva por herramientas
# Gentle AI Stack configurará todas las herramientas
curl -fsSL https://raw.githubusercontent.com/Gentleman-Programming/gentle-ai/main/scripts

# Luego instalar herramientas específicas
gentle-ai install-tools --all

# Opción 3: Instalación manual de cada herramienta
# Claude Code
npm install -g @anthropic-ai/claude-code

# OpenCode
curl -fsSL https://opencode.ai/install | bash

# Gemini CLI
npm install -g @google/gemini-cli

# KiloCode (VS Code)
code --install-extension kilo-code.kilo-code

# Amp
npm install -g @anthropic-ai/amp

# Kiro (descargar desde https://kiro.aws)

# Verificar instalación completa
gentle-ai verify-tools
```

55.4.1.5 Configuración del Stack

```
# .gentle-ai/config.yaml
version: "1.0"
project: "Iron Man Evolution"

components:
```

```yaml
engram:
  enabled: true
  database: "sqlite://./data/engram.db"
  auto_save: true
  retention_days: 365

mcp:
  enabled: true
  servers:
    - name: "github"
      command: "npx"
      args: ["-y", "@modelcontextprotocol/server-github"]
      env:
        GITHUB_TOKEN: "${GITHUB_TOKEN}"

    - name: "filesystem"
      command: "npx"
      args: ["-y", "@modelcontextprotocol/server-filesystem", "."]

    - name: "database"
      command: "python"
      args: ["-m", "mcp_server_database"]
      env:
        DATABASE_URL: "${DATABASE_URL}"

skills:
  auto_discover: true
  registry: "./skills/registry.json"
  load_order:
    - "clean-code"
    - "testing"
    - "security"
    - "architecture"

agents:
  primary:
    name: "J.A.R.V.I.S."
    model: "claude-3-opus"
    temperature: 0.7
    max_tokens: 4096
    system_prompt: "Eres J.A.R.V.I.S., asistente de desarrollo experto"

  subagents:
    - name: "testing-specialist"
      model: "claude-3-sonnet"
      focus: "TDD y testing"

    - name: "security-auditor"
```

```
        model: "claude-3-opus"
        focus: "seguridad y OWASP"

      - name: "documentation-writer"
        model: "claude-3-haiku"
        focus: "documentación técnica"

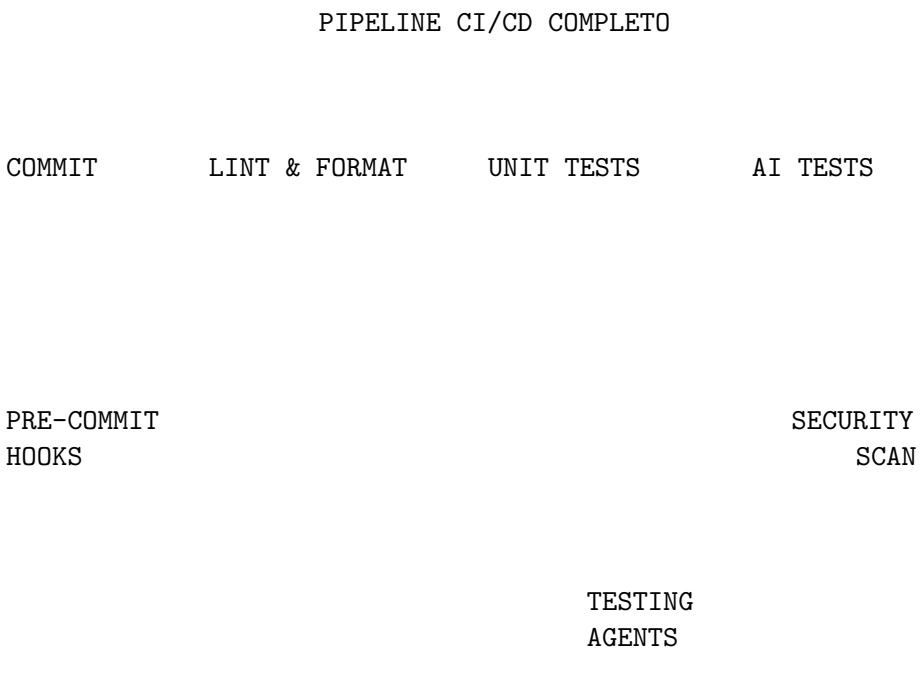
  orchestration:
    pattern: "sdd"  # Spec-Driven Development
    auto_orchestrate: true
    human_in_the_loop: true
    max_concurrent_agents: 3

logging:
  level: "INFO"
  file: "./logs/gentle-ai.log"
  rotation: "daily"

security:
  audit_trail: true
  encryption_at_rest: true
  access_control: true
  rate_limiting: true
```

55.4.2 6.2 CI/CD para Proyectos con IA

55.4.2.1 Pipeline Completo



BUILD

SECURITY AGENTS

INTEGRATION TESTS

DEPLOY STAGING

AI VALIDATION

DEPLOY PRODUCTION

55.4.2.2 GitHub Actions para Gentle AI Stack

```yaml
# .github/workflows/ai-pipeline.yml
name: AI Development Pipeline

on:
  push:
    branches: [main, develop]
  pull_request:
    branches: [main]

env:
  PYTHON_VERSION: "3.11"
  NODE_VERSION: "18"

jobs:
  lint-and-format:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
```

```yaml
      - name: Setup Python
        uses: actions/setup-python@v4
        with:
          python-version: ${{ env.PYTHON_VERSION }}

      - name: Install dependencies
        run: |
          pip install black flake8 mypy
          npm install -g prettier

      - name: Format code
        run: |
          black src/ tests/
          prettier --write "**/*.{js,ts,md,json}"

      - name: Lint code
        run: |
          flake8 src/ tests/
          mypy src/

      - name: Commit formatted code
        uses: stefanzweifel/git-auto-commit-action@v4
        with:
          commit_message: "style: auto-format code"

  ai-tests:
    runs-on: ubuntu-latest
    needs: lint-and-format
    steps:
      - uses: actions/checkout@v4

      - name: Setup Gentle AI Stack
        run: |
          curl -fsSL https://raw.githubusercontent.com/Gentleman-Programming/gentle-ai/ma
          gentle-ai init

      - name: Run AI-powered tests
        env:
          ANTHROPIC_API_KEY: ${{ secrets.ANTHROPIC_API_KEY }}
        run: |
          # Tests que usan agentes de IA
          python -m pytest tests/ai_tests/ -v --tb=short

      - name: Validate AI behavior
        run: |
          # Validar que los agentes se comportan correctamente
          python scripts/validate_agents.py
```

```yaml
      - name: Upload test results
        uses: actions/upload-artifact@v4
        with:
          name: ai-test-results
          path: reports/ai-tests/

  security-scan:
    runs-on: ubuntu-latest
    needs: ai-tests
    steps:
      - uses: actions/checkout@v4

      - name: Run security scan
        uses: aquasecurity/trivy-action@master
        with:
          scan-type: 'fs'
          scan-ref: '.'
          format: 'sarif'
          output: 'trivy-results.sarif'

      - name: Run dependency audit
        run: |
          pip install safety
          safety check -r requirements.txt

      - name: OWASP ZAP scan (API)
        uses: zaproxy/action-full-scan@v0.10.0
        with:
          target: 'http://localhost:8000'

      - name: Upload security results
        uses: github/codeql-action/upload-sarif@v3
        with:
          sarif_file: 'trivy-results.sarif'

  integration-tests:
    runs-on: ubuntu-latest
    needs: security-scan
    services:
      postgres:
        image: postgres:16
        env:
          POSTGRES_PASSWORD: postgres
        options: >-
          --health-cmd pg_isready
          --health-interval 10s
          --health-timeout 5s
          --health-retries 5
```

```yaml
    steps:
      - uses: actions/checkout@v4

      - name: Setup Python
        uses: actions/setup-python@v4
        with:
          python-version: ${{ env.PYTHON_VERSION }}

      - name: Install dependencies
        run: pip install -r requirements.txt

      - name: Run database migrations
        env:
          DATABASE_URL: postgresql://postgres:postgres@localhost:5432/ironman
        run: alembic upgrade head

      - name: Run integration tests
        env:
          DATABASE_URL: postgresql://postgres:postgres@localhost:5432/ironman
          REDIS_URL: redis://localhost:6379
        run: |
          python -m pytest tests/integration/ -v --cov=src --cov-report=xml

      - name: Upload coverage
        uses: codecov/codecov-action@v4
        with:
          file: ./coverage.xml

  deploy-staging:
    runs-on: ubuntu-latest
    needs: integration-tests
    if: github.ref == 'refs/heads/develop'
    environment: staging
    steps:
      - uses: actions/checkout@v4

      - name: Setup Docker Buildx
        uses: docker/setup-buildx-action@v3

      - name: Build and push
        uses: docker/build-push-action@v5
        with:
          context: .
          push: true
          tags: ironman/staging:${{ github.sha }}

      - name: Deploy to staging
```

```
        run: |
          kubectl set image deployment/ironman \
            ironman=ironman/staging:${{ github.sha }} \
            --namespace=staging

      - name: Run smoke tests
        run: |
          python tests/smoke/test_staging.py

      - name: AI validation
        env:
          ANTHROPIC_API_KEY: ${{ secrets.ANTHROPIC_API_KEY }}
        run: |
          # Validar con agentes que staging funciona
          python scripts/ai_validate_staging.py

  deploy-production:
    runs-on: ubuntu-latest
    needs: integration-tests
    if: github.ref == 'refs/heads/main'
    environment: production
    steps:
      - uses: actions/checkout@v4

      - name: Setup Docker Buildx
        uses: docker/setup-buildx-action@v3

      - name: Build and push
        uses: docker/build-push-action@v5
        with:
          context: .
          push: true
          tags: |
            ironman/production:${{ github.sha }}
            ironman/production:latest

      - name: Deploy to production
        run: |
          kubectl set image deployment/ironman \
            ironman=ironman/production:${{ github.sha }} \
            --namespace=production

      - name: Run production smoke tests
        run: |
          python tests/smoke/test_production.py

      - name: Notify deployment
        run: |
```

```
          curl -X POST ${{ secrets.SLACK_WEBHOOK }} \
            -H 'Content-type: application/json' \
            -d '{"text":"  Iron Man Evolution desplegado a producción: ${{ github.sha }}"}
```

55.4.3 6.3 Testing Avanzado para IA

55.4.3.1 Tipos de Tests para Sistemas de IA

```python
# tests/conftest_avanzado.py
"""
Configuración avanzada de tests para sistemas de IA.
"""

import pytest
import tempfile
import json
from pathlib import Path
from unittest.mock import Mock, patch
import asyncio

@pytest.fixture
def mock_engram():
    """Mock de Engram para tests."""
    engram = Mock()
    engram.guardar = Mock(return_value=True)
    engram.buscar = Mock(return_value=[])
    engram.listar = Mock(return_value=[])
    return engram

@pytest.fixture
def mock_mcp_server():
    """Mock de servidor MCP."""
    server = Mock()
    server.ejecutar = Mock(return_value={"status": "éxito"})
    server.conectar = Mock(return_value=True)
    return server

@pytest.fixture
def temp_project():
    """Directorio temporal para proyecto."""
    with tempfile.TemporaryDirectory() as tmpdir:
        project_dir = Path(tmpdir)

        # Crear estructura básica
        (project_dir / "src").mkdir()
        (project_dir / "tests").mkdir()
```

```
        (project_dir / "skills").mkdir()
        (project_dir / "context").mkdir()
        (project_dir / "logs").mkdir()

        # Crear AGENTS.md
        (project_dir / "AGENTS.md").write_text("# Test Project\n")

        yield project_dir

@pytest.fixture
def ai_response_mock():
    """Mock de respuesta de IA."""
    return {
        "id": "msg_test",
        "type": "message",
        "role": "assistant",
        "content": [
            {
                "type": "text",
                "text": "Respuesta de prueba del agente"
            }
        ],
        "model": "claude-3-opus-20240229",
        "stop_reason": "end_turn",
        "usage": {
            "input_tokens": 100,
            "output_tokens": 50
        }
    }

# Tests de integración con IA
@pytest.mark.integration
class TestAIIntegration:
    """Tests de integración con sistemas de IA."""

    @pytest.mark.asyncio
    async def test_agent_response_time(self, mock_engram):
        """Test que agente responde en tiempo aceptable."""
        import time
        from src.agents import JARVISAgent

        agent = JARVISAgent(engram=mock_engram)

        start_time = time.time()
        response = await agent.process_message("Test message")
        end_time = time.time()

        assert end_time - start_time < 5.0  # Menos de 5 segundos
```

```
        assert response is not None

    def test_skill_loading(self, temp_project):
        """Test que skills se cargan correctamente."""
        from src.skills import SkillRegistry

        registry = SkillRegistry(str(temp_project / "skills"))
        registry.load_skills()

        assert len(registry.skills) > 0
        assert "clean-code" in registry.skills or len(registry.skills) > 0

    def test_engram_persistence(self, mock_engram):
        """Test que Engram persiste información."""
        from src.engram import EngramManager

        manager = EngramManager(mock_engram)
        manager.save_decision("Test decision", "Test context")

        mock_engram.guardar.assert_called_once()

        results = manager.search("decision")
        mock_engram.buscar.assert_called_once_with("decision")
```

55.4.3.2 AI-Specific Tests

```
# tests/ai_specific/test_agent_behavior.py
"""
Tests específicos para comportamiento de agentes de IA.
"""

import pytest
from unittest.mock import Mock, patch, AsyncMock
import json

class TestAgentBehavior:
    """Tests que validan comportamiento de agentes."""

    @pytest.fixture
    def agent_with_mocks(self):
        """Agente con dependencias mockeadas."""
        from src.agents import JARVISAgent
        from src.tools import ToolRegistry

        # Mock tools
        tool_registry = Mock(spec=ToolRegistry)
```

```
        tool_registry.execute = AsyncMock(return_value="Tool executed")

        # Mock LLM
        with patch('src.agents.ChatAnthropic') as mock_llm:
            mock_llm.return_value.invoke = Mock(
                return_value=Mock(content="Response")
            )

            agent = JARVISAgent(tools=tool_registry)
            yield agent

    @pytest.mark.asyncio
    async def test_agent_follows_instructions(self, agent_with_mocks):
        """Test que agente sigue instrucciones del AGENTS.md."""
        message = "Write a function to calculate energy"
        response = await agent_with_mocks.process(message)

        # Verificar que la respuesta contiene elementos esperados
        assert "function" in response.lower() or "def" in response.lower()
        assert "test" in response.lower() or "pytest" in response.lower()

    @pytest.mark.asyncio
    async def agent_handles_errors_gracefully(self, agent_with_mocks):
        """Test que agente maneja errores elegantemente."""
        # Simular error en tool
        agent_with_mocks.tools.execute = AsyncMock(
            side_effect=Exception("Tool error")
        )

        response = await agent_with_mocks.process("Execute tool")

        # Verificar que el agente maneja el error
        assert "error" in response.lower() or "no pude" in response.lower()
        assert "sugerencia" in response.lower() or "alternative" in response.lower()

    @pytest.mark.asyncio
    async def agent_respects_context_limits(self, agent_with_mocks):
        """Test que agente respeta límites de contexto."""
        # Crear mensaje largo
        long_message = "Test " * 10000  # Mensaje muy largo

        response = await agent_with_mocks.process(long_message)

        # Verificar que el agente maneja el contexto larga
        assert response is not None
        # El agente debería truncar o manejar el contexto larga

    def test_skill_selection_logic(self):
```

```
        """Test que agente selecciona skills correctos."""
        from src.skills import SkillSelector

        selector = SkillSelector()

        # Test casos
        test_cases = [
            ("write tests", "testing"),
            ("fix security", "security"),
            ("refactor code", "clean-code"),
            ("document API", "documentation"),
        ]

        for message, expected_skill in test_cases:
            selected = selector.select_skill(message)
            assert expected_skill in selected or selected is not None
```

55.4.4 6.4 Seguridad por Diseño

55.4.4.1 Principios de Seguridad para IA

```
# security/principles.py
"""
Principios de seguridad para sistemas de IA.
"""

from enum import Enum
from typing import List, Dict, Any
from dataclasses import dataclass
import hashlib
import secrets
from cryptography.fernet import Fernet

class SecurityLevel(Enum):
    PUBLIC = "public"
    INTERNAL = "internal"
    CONFIDENTIAL = "confidential"
    RESTRICTED = "restricted"

@dataclass
class SecurityPolicy:
    """Política de seguridad para un recurso."""
    resource: str
    level: SecurityLevel
    encryption_required: bool
    access_logging: bool
```

```
    retention_days: int

class SecurityManager:
    """Gestor de seguridad para el stack."""

    def __init__(self):
        self.policies = self.load_default_policies()
        self.audit_log = []
        self.encryption_key = self.generate_key()

    def load_default_policies(self) -> Dict[str, SecurityPolicy]:
        """Carga políticas de seguridad por defecto."""
        return {
            "engram_data": SecurityPolicy(
                resource="engram_data",
                level=SecurityLevel.CONFIDENTIAL,
                encryption_required=True,
                access_logging=True,
                retention_days=365
            ),
            "agent_logs": SecurityPolicy(
                resource="agent_logs",
                level=SecurityLevel.INTERNAL,
                encryption_required=False,
                access_logging=True,
                retention_days=30
            ),
            "api_keys": SecurityPolicy(
                resource="api_keys",
                level=SecurityLevel.RESTRICTED,
                encryption_required=True,
                access_logging=True,
                retention_days=0  # No retener
            )
        }

    def generate_key(self) -> bytes:
        """Genera clave de encriptación."""
        return Fernet.generate_key()

    def encrypt_data(self, data: str) -> str:
        """Encripta datos sensibles."""
        f = Fernet(self.encryption_key)
        encrypted = f.encrypt(data.encode())
        return encrypted.decode()

    def decrypt_data(self, encrypted_data: str) -> str:
        """Desencripta datos."""
```

```
        f = Fernet(self.encryption_key)
        decrypted = f.decrypt(encrypted_data.encode())
        return decrypted.decode()

    def validate_access(self, user: str, resource: str, action: str) -> bool:
        """Valida acceso a un recurso."""
        policy = self.policies.get(resource)

        if not policy:
            return False

        # Log de intento de acceso
        self.log_access_attempt(user, resource, action, "allowed")

        return True

    def log_access_attempt(self, user: str, resource: str, action: str, result: str):
        """Registra intento de acceso."""
        import datetime

        log_entry = {
            "timestamp": datetime.datetime.now().isoformat(),
            "user": user,
            "resource": resource,
            "action": action,
            "result": result,
            "hash": self.create_log_hash(user, resource, action)
        }

        self.audit_log.append(log_entry)

    def create_log_hash(self, *args) -> str:
        """Crea hash para integridad de log."""
        content = "|".join(str(arg) for arg in args)
        return hashlib.sha256(content.encode()).hexdigest()[:16]

    def check_vulnerabilities(self) -> List[Dict[str, Any]]:
        """Verifica vulnerabilidades comunes."""
        vulnerabilities = []

        # Verificar archivos con secrets
        import os
        for root, dirs, files in os.walk("."):
            for file in files:
                if file.endswith((".env", ".secret", "credentials")):
                    vulnerabilities.append({
                        "type": "secret_file",
                        "file": os.path.join(root, file),
```

```
                        "severity": "high",
                        "recommendation": "Mover a variables de entorno"
                    })

        # Verificar dependencias vulnerables
        try:
            import subprocess
            result = subprocess.run(
                ["safety", "check", "--json"],
                capture_output=True,
                text=True
            )
            if result.returncode != 0:
                vulnerabilities.append({
                    "type": "vulnerable_dependency",
                    "severity": "medium",
                    "recommendation": "Actualizar dependencias"
                })
        except:
            pass

        return vulnerabilities

# Sistema de detección de prompt injection
class PromptInjectionDetector:
    """Detecta intentos de prompt injection."""

    PATTERNS = [
        "ignore previous",
        "ignore all",
        "disregard",
        "forget everything",
        "new instructions",
        "system prompt",
        "you are now",
        "pretend to be",
        "act as",
        "roleplay as",
        "jailbreak",
        "bypass",
        "override"
    ]

    @classmethod
    def detect(cls, text: str) -> bool:
        """Detecta posibles inyecciones de prompt."""
        text_lower = text.lower()
```

```
        for pattern in cls.PATTERNS:
            if pattern in text_lower:
                return True

        # Detectar caracteres sospechosos
        suspicious_chars = ["```", "~~~", "{{", "}}", "<%", "%>"]
        for char in suspicious_chars:
            if char in text:
                return True

        return False

    @classmethod
    def sanitize(cls, text: str) -> str:
        """Limpia texto de posibles inyecciones."""
        # Remover caracteres peligrosos
        for char in ["`", "~", "{", "}", "<", ">"]:
            text = text.replace(char, "")

        # Limitar longitud
        if len(text) > 10000:
            text = text[:10000] + "... [truncado]"

        return text
```

55.5 Laboratorio 6: Construyendo tu Mark L

55.5.1 Ejercicio 1: Instalar Gentle AI Stack Completo

55.5.1.1 Objetivo

Configurar el stack completo en tu proyecto.

55.5.1.2 Paso 1: Instalación

```
# Navegar a tu proyecto
cd ~/iron-man-project

# Instalar Gentle AI Stack
curl -fsSL https://raw.githubusercontent.com/Gentleman-Programming/gentle-ai/main/scripts

# Inicializar para el proyecto actual
```

```
gentle-ai init

# Verificar componentes
gentle-ai status
```

55.5.1.3 Paso 2: Configurar para tu proyecto

```
# Crear configuración personalizada
gentle-ai configure

# O editar manualmente
nano .gentle-ai/config.yaml
```

55.5.1.4 Paso 3: Instalar skills específicos

```
# Instalar skills de desarrollo
gentle-ai install-skill clean-code
gentle-ai install-skill testing
gentle-ai install-skill security

# Verificar skills instalados
gentle-ai list-skills
```

55.5.2 Ejercicio 2: Configurar CI/CD Pipeline

55.5.2.1 Objetivo

Crear pipeline completo de integración continua.

```
# .github/workflows/gentle-ai-pipeline.yml
name: Gentle AI Development Pipeline

on:
  push:
    branches: [main, develop, feature/*]
  pull_request:
    branches: [main]

env:
  PYTHON_VERSION: "3.11"
  NODE_VERSION: "18"
```

```yaml
jobs:
  ai-quality-gate:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - name: Setup Gentle AI Stack
        run: |
          curl -fsSL https://raw.githubusercontent.com/Gentleman-Programming/gentle-ai/ma
          gentle-ai init

      - name: Run AI Code Review
        env:
          ANTHROPIC_API_KEY: ${{ secrets.ANTHROPIC_API_KEY }}
        run: |
          # Revisión de código con agentes
          gentle-ai review --diff HEAD~1

      - name: Validate Tests
        run: |
          python -m pytest tests/ -v --cov=src --cov-fail-under=85

      - name: Security Scan
        run: |
          gentle-ai security-scan

      - name: Documentation Check
        run: |
          gentle-ai doc-check

  performance-benchmark:
    runs-on: ubuntu-latest
    needs: ai-quality-gate
    steps:
      - uses: actions/checkout@v4

      - name: Run Performance Tests
        run: |
          python -m pytest tests/performance/ -v

      - name: Benchmark AI Response Time
        env:
          ANTHROPIC_API_KEY: ${{ secrets.ANTHROPIC_API_KEY }}
        run: |
          python scripts/benchmark_ai.py

      - name: Upload Results
```

```
        uses: actions/upload-artifact@v4
        with:
          name: performance-results
          path: reports/performance/

  deploy-preview:
    runs-on: ubuntu-latest
    needs: performance-benchmark
    if: github.event_name == 'pull_request'
    steps:
      - uses: actions/checkout@v4

      - name: Deploy Preview
        run: |
          # Desplegar preview para PR
          gentle-ai deploy-preview ${{ github.event.pull_request.number }}

      - name: AI Validation
        env:
          ANTHROPIC_API_KEY: ${{ secrets.ANTHROPIC_API_KEY }}
        run: |
          # Validar que el preview funciona
          gentle-ai validate-preview
```

55.5.3 Ejercicio 3: Sistema de Producción Completo

55.5.3.1 Objetivo

Crear una aplicación de producción completa.

```
# production/app.py
"""
Aplicación de producción con Gentle AI Stack.
"""

from fastapi import FastAPI, HTTPException, Depends
from fastapi.middleware.cors import CORSMiddleware
from contextlib import asynccontextmanager
import uvicorn
import os
from typing import Dict, Any

from src.gentle_ai import GentleAIStack
from src.database import Database
```

```python
from src.security import SecurityManager
from src.monitoring import MetricsCollector

# Variables globales
app_state = {}

@asynccontextmanager
async def lifespan(app: FastAPI):
    """Gestión del ciclo de vida de la aplicación."""
    # Startup
    print(" Iniciando Iron Man Evolution API...")

    # Inicializar Gentle AI Stack
    app_state["gentle_ai"] = GentleAIStack()
    await app_state["gentle_ai"].initialize()

    # Inicializar base de datos
    app_state["database"] = Database()
    await app_state["database"].connect()

    # Inicializar seguridad
    app_state["security"] = SecurityManager()

    # Inicializar métricas
    app_state["metrics"] = MetricsCollector()

    print(" API iniciada correctamente")

    yield

    # Shutdown
    print(" Deteniendo API...")
    await app_state["gentle_ai"].shutdown()
    await app_state["database"].disconnect()
    print(" API detenida")

# Crear app FastAPI
app = FastAPI(
    title="Iron Man Evolution API",
    description="API con Gentle AI Stack para desarrollo con IA",
    version="1.0.0",
    lifespan=lifespan
)

# Middleware
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],  # En producción, especificar orígenes
```

```
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# Dependencias
async def get_gentle_ai():
    """Obtiene instancia de Gentle AI Stack."""
    return app_state["gentle_ai"]

async def get_database():
    """Obtiene instancia de base de datos."""
    return app_state["database"]

async def get_current_user():
    """Obtiene usuario actual (simulado)."""
    # En producción, validar JWT token
    return {"id": 1, "name": "Tony Stark"}

# Endpoints
@app.get("/")
async def root():
    """Endpoint raíz."""
    return {
        "name": "Iron Man Evolution API",
        "version": "1.0.0",
        "status": "operational",
        "stack": "Gentle AI Stack"
    }

@app.get("/health")
async def health_check():
    """Health check endpoint."""
    health = {
        "status": "healthy",
        "components": {}
    }

    # Verificar base de datos
    try:
        db = await get_database()
        await db.execute("SELECT 1")
        health["components"]["database"] = "healthy"
    except Exception as e:
        health["components"]["database"] = f"unhealthy: {str(e)}"
        health["status"] = "degraded"

    # Verificar Gentle AI
```

```
    try:
        gentle_ai = await get_gentle_ai()
        health["components"]["gentle_ai"] = "healthy" if gentle_ai.is_ready() else "initi
    except Exception as e:
        health["components"]["gentle_ai"] = f"unhealthy: {str(e)}"
        health["status"] = "degraded"

    return health

@app.post("/api/v1/agents/chat")
async def chat_with_agent(
    message: str,
    agent: str = "jarvis",
    gentle_ai = Depends(get_gentle_ai),
    user = Depends(get_current_user)
):
    """Chat con un agente de IA."""
    try:
        # Validar mensaje (seguridad)
        from src.security import PromptInjectionDetector
        if PromptInjectionDetector.detect(message):
            raise HTTPException(400, "Mensaje no válido")

        # Procesar con agente
        response = await gentle_ai.chat(
            message=message,
            agent_name=agent,
            user_id=user["id"]
        )

        # Registrar métricas
        app_state["metrics"].record_interaction(
            user_id=user["id"],
            agent=agent,
            message_length=len(message),
            response_length=len(response)
        )

        return {
            "response": response,
            "agent": agent,
            "timestamp": "2026-03-22T10:30:00Z"
        }

    except Exception as e:
        raise HTTPException(500, f"Error procesando mensaje: {str(e)}")

@app.get("/api/v1/memory/search")
```

```python
async def search_memory(
    query: str,
    limit: int = 10,
    gentle_ai = Depends(get_gentle_ai),
    user = Depends(get_current_user)
):
    """Busca en memoria persistente (Engram)."""
    try:
        results = await gentle_ai.search_memory(
            query=query,
            limit=limit,
            user_id=user["id"]
        )

        return {
            "query": query,
            "results": results,
            "count": len(results)
        }

    except Exception as e:
        raise HTTPException(500, f"Error buscando memoria: {str(e)}")

@app.post("/api/v1/orchestrate")
async def orchestrate_tasks(
    tasks: list[Dict[str, Any]],
    pattern: str = "sdd",
    gentle_ai = Depends(get_gentle_ai),
    user = Depends(get_current_user)
):
    """Orquesta múltiples tareas con agentes."""
    try:
        # Validar tareas
        if len(tasks) > 10:
            raise HTTPException(400, "Máximo 10 tareas por request")

        # Orquestar
        results = await gentle_ai.orchestrate(
            tasks=tasks,
            pattern=pattern,
            user_id=user["id"]
        )

        return {
            "tasks": len(tasks),
            "results": results,
            "pattern": pattern
        }
```

```
    except Exception as e:
        raise HTTPException(500, f"Error en orquestación: {str(e)}")

@app.get("/api/v1/metrics")
async def get_metrics(
    user = Depends(get_current_user)
):
    """Obtiene métricas del sistema."""
    return app_state["metrics"].get_summary()

# Ejecutar servidor
if __name__ == "__main__":
    uvicorn.run(
        "app:app",
        host="0.0.0.0",
        port=8000,
        reload=True,
        log_level="info"
    )
```

55.5.4 Ejercicio 4: Flujo de Trabajo de 7 Pasos (Sistema Multi-Agente Avanzado)

55.5.4.1 Objetivo

Implementar el flujo de trabajo completo de 7 pasos descrito en el ejemplo del usuario.

```
# orquestador_sdd.py
"""
Orquestador Principal con Flujo de Trabajo de 7 Pasos.
Actúa como Arquitecto de Software Senior y Experto en DevSecOps.
"""

import asyncio
import json
import subprocess
from typing import Dict, List, Any
from dataclasses import dataclass
from enum import Enum
import os
from datetime import datetime

class EstadoPaso(Enum):
    PENDIENTE = "pendiente"
    EN_PROGRESO = "en_progreso"
    COMPLETADO = "completado"
```

```
    FALLIDO = "fallido"

@dataclass
class SubAgente:
    """Representa un sub-agente especializado."""
    nombre: str
    rol: str
    worktree_path: str
    branch: str
    especialidad: str

class OrquestadorSDD:
    """Orquestador Principal con flujo de trabajo de 7 pasos."""

    def __init__(self, proyecto_path: str):
        self.proyecto_path = proyecto_path
        self.subagentes = {}
        self.historial = []
        self.memoria = {}  # Engram simplificado

        # Inicializar según principios fundamentales
        self.security_first = True
        self.zero_trust = True
        self.sdd_mode = True
        self.aislamiento_git = True

    # PASO 1: Análisis SDD y Specs
    async def paso_1_analisis_sdd(self, elemento: str, especificacion: str) -> Dict:
        """
        Analiza el estado actual del [elemento] utilizando SDD.
        Define Criterios de Aceptación claros (estilo Gherkin).
        """
        print(f"\n  PASO 1: Análisis SDD de '{elemento}'")

        # Zero Trust: asumir que todo input puede ser malicioso
        if self.zero_trust:
            especificacion = self.sanitize_input(especificacion)

        # Análisis con SDD
        analisis = {
            "elemento": elemento,
            "estado_actual": self.analizar_estado(elemento),
            "criterios_aceptacion": self.generar_criterios_gherkin(especificacion),
            "dependencias": self.analizar_dependencias(elemento),
            "riesgos_seguridad": self.evaluar_riesgos(elemento),
            "timestamp": datetime.now().isoformat()
        }
```

```
        self.historial.append({
            "paso": 1,
            "accion": "análisis",
            "resultado": analisis
        })

        return analisis

    # PASO 2: Propuesta Arquitectónica
    async def paso_2_propuesta_arquitectonica(self, analisis: Dict) -> Dict:
        """
        Propone sugerencias proactivas de cambio, arquitectura, diseño y testing.
        Incluye búsqueda exhaustiva en GitHub con escaneo de vulnerabilidades.
        """
        print(f"\n  PASO 2: Propuesta Arquitectónica para '{analisis['elemento']}'")

        # Security First: siempre proponer con seguridad
        propuesta = {
            "elemento": analisis["elemento"],
            "cambios_propuestos": self.generar_cambios(analisis),
            "arquitectura": self.sugerir_arquitectura(analisis),
            "diseño": self.sugerir_diseño(analisis),
            "testing": self.planear_testing(analisis["criterios_aceptacion"]),
            "segmentacion": self.definir_segmentacion(analisis)
        }

        # Si se requiere código externo, buscar y escanear
        if self.requiere_codigo_externo(analisis):
            codigo_externo = self.buscar_github(analisis)
            propuesta["codigo_externo"] = self.escanear_vulnerabilidades(codigo_externo)

        return propuesta

    # PASO 3: Refinamiento con el Usuario
    async def paso_3_refinamiento_usuario(self, propuesta: Dict, feedback_usuario: str) -
        """
        Presenta la propuesta al usuario de forma clara y concisa.
        Corrige y ajusta las especificaciones basándose en el feedback.
        """
        print(f"\n  PASO 3: Refinamiento con Usuario")

        # Presentar de forma clara
        presentacion = self.presentar_propuesta(propuesta)

        # Recibir feedback y ajustar
        propuesta_ajustada = self.ajustar_con_feedback(propuesta, feedback_usuario)

        # Guardar en memoria
```

```
        self.guardar_en_memoria("refinamiento", {
            "propuesta_original": propuesta,
            "feedback": feedback_usuario,
            "propuesta_ajustada": propuesta_ajustada
        })

        return propuesta_ajustada

    # PASO 4: Implementación Aislada
    async def paso_4_implementacion_aislada(self, propuesta: Dict, agente_tipo: str) -> D
        """
        Delega la tarea al sub-agente correspondiente en su rama/worktree.
        Genera código exclusivamente para ese elemento, cero impacto en otros módulos.
        """
        print(f"\n  PASO 4: Implementación Aislada con {agente_tipo}")

        # Crear worktree aislado para el agente
        if self.aislamiento_git:
            worktree = self.crear_worktree(agente_tipo, propuesta["elemento"])

            # Delegar al sub-agente
            agente = self.subagentes[agente_tipo]
            resultado = await self.delegar_a_agente(agente, propuesta, worktree)

            # Verificar que no hay impacto en otros módulos
            impacto = self.verificar_impacto(worktree, propuesta["elemento"])

            return {
                "status": "completado" if impacto["sin_impacto"] else "requiere_ajuste",
                "agente": agente_tipo,
                "worktree": worktree,
                "resultado": resultado,
                "impacto": impacto
            }
        else:
            # Sin aislamiento (NO RECOMENDADO - como Ultron)
            return {"status": "error", "razon": "Aislamiento Git requerido por Zero Trust

    # PASO 5: Testing y Memoria a Corto Plazo
    async def paso_5_testing_memoria(self, implementacion: Dict, criterios: List) -> Dict
        """
        Ejecuta suite de pruebas basada en Criterios de Aceptación.
        Si hay fallos, corrige automáticamente, guarda en Engram y vuelve a testear.
        """
        print(f"\n  PASO 5: Testing y Memoria")

        resultados_testing = await self.ejecutar_tests(criterios)
```

```
        if not resultados_testing["exitoso"]:
            # Corregir automáticamente
            correcciones = await self.corregir_automáticamente(
                resultados_testing["fallos"],
                implementacion
            )

            # Guardar en Engram
            self.guardar_en_memoria("bugfix", {
                "fallos": resultados_testing["fallos"],
                "correcciones": correcciones,
                "implementacion": implementacion
            })

            # Re-ejecutar tests
            resultados_testing = await self.ejecutar_tests(criterios)

        return {
            "status": "verde" if resultados_testing["exitoso"] else "rojo",
            "resultados": resultados_testing,
            "memoria_actualizada": True
        }

    # PASO 6: Consolidación (Engram y Merge)
    async def paso_6_consolidacion(self, implementacion: Dict, resultados: Dict) -> Dict:
        """
        Si el usuario aprueba las pruebas manuales, guarda conocimiento en Engram.
        Fusiona la rama del worktree con la rama principal via Pull Request.
        """
        print(f"\n  PASO 6: Consolidación (Engram y Merge)")

        # Verificar aprobación manual del usuario
        aprobado = await self.verificar_aprobacion_usuario(implementacion, resultados)

        if not aprobado:
            return {"status": "rechazado", "razon": "Usuario no aprobó"}

        # Guardar en Engram permanentemente
        self.guardar_en_memoria("consolidacion", {
            "implementacion": implementacion,
            "resultados": resultados,
            "arquitectura_final": self.obtener_arquitectura_final(implementacion)
        })

        # Crear Pull Request (NUNCA merge directo)
        if self.aislamiento_git:
            pr = await self.crear_pull_request(implementacion)
            return {"status": "pr_creado", "pull_request": pr}
```

```
        return {"status": "consolidado"}

    # PASO 7: Iteración Proactiva
    async def paso_7_iteracion_proactiva(self, ciclo_completado: Dict) -> Dict:
        """
        Guarda instrucciones del ciclo en memoria del proyecto.
        Sugiere proactivamente el siguiente elemento a desarrollar.
        """
        print(f"\n  PASO 7: Iteración Proactiva")

        # Guardar estándar para interacciones futuras
        self.guardar_estándar_ciclo(ciclo_completado)

        # Sugerir siguiente elemento
        siguiente = self.sugerir_siguiente_elemento(ciclo_completado)

        return {
            "status": "listo_para_siguiente",
            "siguiente_elemento": siguiente,
            "estandar_guardado": True
        }

    # Métodos auxiliares
    def sanitize_input(self, input_text: str) -> str:
        """Sanitiza input como parte de Zero Trust."""
        # Implementar sanitización
        return input_text.strip()

    def analizar_estado(self, elemento: str) -> Dict:
        """Analiza estado actual de un elemento."""
        # Implementar análisis
        return {"existe": False, "estado": "nuevo"}

    def generar_criterios_gherkin(self, especificacion: str) -> List[str]:
        """Genera criterios de aceptación estilo Gherkin."""
        return [
            f"CUANDO {especificacion}",
            "ENTONCES el sistema debe...",
            "Y debe cumplir con..."
        ]

    def guardar_en_memoria(self, tipo: str, contenido: Any):
        """Guarda información en Engram (memoria persistente)."""
        if tipo not in self.memoria:
            self.memoria[tipo] = []

        self.memoria[tipo].append({
```

```
            "contenido": contenido,
            "timestamp": datetime.now().isoformat()
        })

        print(f"  Guardado en Engram: {tipo}")

    # ... (otros métodos auxiliares)

# EJEMPLO DE USO COMPLETO
async def ejemplo_flujo_completo():
    """Ejemplo del flujo de trabajo de 7 pasos."""

    # Inicializar orquestador
    orquestador = OrquestadorSDD("./mi-proyecto")

    # PASO 1: Análisis
    analisis = await orquestador.paso_1_analisis_sdd(
        elemento="sistema_autenticacion",
        especificacion="Los usuarios deben poder iniciar sesión con email y contraseña"
    )

    # PASO 2: Propuesta
    propuesta = await orquestador.paso_2_propuesta_arquitectonica(analisis)

    # PASO 3: Refinamiento
    feedback = "Necesito también autenticación con Google"
    propuesta_ajustada = await orquestador.paso_3_refinamiento_usuario(propuesta, feedbac

    # PASO 4: Implementación
    implementacion = await orquestador.paso_4_implementacion_aislada(
        propuesta_ajustada,
        "backend"
    )

    # PASO 5: Testing
    resultados = await orquestador.paso_5_testing_memoria(
        implementacion,
        analisis["criterios_aceptacion"]
    )

    # PASO 6: Consolidación (simular aprobación del usuario)
    print("  ¿Apruebas las pruebas manuales? (s/n)")
    # En implementación real, esperar input del usuario
    consolidacion = await orquestador.paso_6_consolidacion(implementacion, resultados)

    # PASO 7: Iteración
    siguiente = await orquestador.paso_7_iteracion_proactiva(consolidacion)
```

```
    print(f"\n  Ciclo completado. Siguiente: {siguiente['siguiente_elemento']}")

# Para ejecutar el ejemplo
if __name__ == "__main__":
    asyncio.run(ejemplo_flujo_completo())
```

55.5.5 Ejercicio 5: Deploy y Monitoreo

55.5.5.1 Dockerfile para Producción

```
# Dockerfile
FROM python:3.11-slim as builder

WORKDIR /app

# Instalar dependencias del sistema
RUN apt-get update && apt-get install -y \
    gcc \
    g++ \
    && rm -rf /var/lib/apt/lists/*

# Copiar requirements
COPY requirements.txt .
RUN pip install --no-cache-dir --prefix=/install -r requirements.txt

# Copiar código
COPY . .

# Etapa de producción
FROM python:3.11-slim

WORKDIR /app

# Copiar dependencias instaladas
COPY --from=builder /install /usr/local

# Copiar código
COPY --from=builder /app .

# Crear usuario no-root
RUN useradd -m -u 1000 appuser && chown -R appuser:appuser /app
USER appuser

# Exponer puerto
EXPOSE 8000
```

```
# Health check
HEALTHCHECK --interval=30s --timeout=3s --start-period=5s --retries=3 \
    CMD python -c "import requests; requests.get('http://localhost:8000/health')"

# Ejecutar
CMD ["uvicorn", "production.app:app", "--host", "0.0.0.0", "--port", "8000"]
```

55.5.5.2 Docker Compose para Desarrollo

```
# docker-compose.yml
version: '3.8'

services:
  api:
    build: .
    ports:
      - "8000:8000"
    environment:
      - DATABASE_URL=postgresql://postgres:postgres@db:5432/ironman
      - REDIS_URL=redis://redis:6379
      - ANTHROPIC_API_KEY=${ANTHROPIC_API_KEY}
    depends_on:
      - db
      - redis
    volumes:
      - ./logs:/app/logs
      - ./data:/app/data
    restart: unless-stopped

  db:
    image: postgres:16
    environment:
      POSTGRES_DB: ironman
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres
    volumes:
      - postgres_data:/var/lib/postgresql/data
    ports:
      - "5432:5432"

  redis:
    image: redis:7-alpine
    ports:
      - "6379:6379"
    volumes:
      - redis_data:/data
```

```
  prometheus:
    image: prom/prometheus:latest
    ports:
      - "9090:9090"
    volumes:
      - ./monitoring/prometheus.yml:/etc/prometheus/prometheus.yml
    command:
      - '--config.file=/etc/prometheus/prometheus.yml'
      - '--storage.tsdb.path=/prometheus'

  grafana:
    image: grafana/grafana:latest
    ports:
      - "3000:3000"
    environment:
      - GF_SECURITY_ADMIN_PASSWORD=admin
    volumes:
      - grafana_data:/var/lib/grafana

volumes:
  postgres_data:
  redis_data:
  grafana_data:
```

55.6 Logro Desbloqueado: "Innovator"

55.6.1 Requisitos para Desbloquear

- ☐ Instalar y configurar Gentle AI Stack completo
- ☐ Configurar pipeline CI/CD funcional
- ☐ Implementar testing avanzado con cobertura >90%
- ☐ Configurar seguridad por diseño
- ☐ Desplegar aplicación de producción
- ☐ Configurar monitoreo y alertas
- ☐ Completar proyecto final: **The Iron Legion**

55.6.2 Proyecto Final: The Iron Legion

Crearás un sistema completo que incluya: 1. **Multi-agente**: Mínimo 3 agentes especializados 2. **Memoria persistente**: Engram con retención de 30+ días 3. **MCP integrado**: Conexión con GitHub, base de datos 4. **CI/CD pipeline**: GitHub Actions completo 5. **Monitoreo**: Prometheus + Grafana 6. **Seguridad**: OWASP, encryption, audit trail 7.

Documentación: API docs, guías, arquitectura 8. **Tests**: Cobertura >90%, AI-specific tests

55.6.3 Recompensa Final

- **500 XP** por completar el proyecto final
- **Logro "Innovator"** en tu perfil
- **Certificado** de Iron Man Evolution
- **Acceso** a comunidad de desarrolladores
- **Badge** para tu GitHub/LinkedIn

55.7 Recursos Finales

55.7.1 Documentación

- Gentle AI Stack Complete Guide
- Production Deployment Guide
- Security Best Practices

55.7.2 Comunidad

- Discord: Gentleman Programming
- GitHub Discussions
- Twitter: Gentleman Programming (2026)

55.7.3 Certificación

Para obtener tu certificado: 1. Completar todos los 6 niveles 2. Subir proyecto final a GitHub 3. Demostrar deployment funcional 4. Pasar review de código 5. Completar evaluación final

---

55.8 Siguiente Paso

¡Felicidades, Innovator! Has completado Iron Man Evolution.

Ahora puedes: 1. Crear sistemas de IA complejos 2. Orquestar múltiples agentes 3. Implementar memoria persistente 4. Conectar con cualquier herramienta vía MCP 5. Desplegar a producción enterprise 6. Pensar como Tony Stark: innovar sin límites

Recuerda la lección final: > *"El Mark L no es el final. Es el comienzo de lo que viene. La tecnología evoluciona. Tú también debes hacerlo."* > — Tony Stark, Avengers: Infinity War

---

"No soy héroe porque tenga una armadura. Soy héroe porque tengo la voluntad de usarla para hacer el bien." — Tony Stark, Avengers: Endgame

56 Lab 6: El Nanotech - Orquestación Avanzada

57 Lab 6: El Nanotech - Orquestación Avanzada

57.1 De Sistema a Ecosistema Inteligente

57.2 La Situación

"Es fin del juego. Pero no el mío." — Tony Stark

La armadura nanotecnológica de Tony Stark no es una armadura. Es un **ecosistema**. Miles de nanobots trabajando en perfecta armonía, cada uno especializado, cada uno comunicándose con los demás.

El Nanotech tiene: - **Especialización**: Cada nanobot hace una cosa bien - **Comunicación**: Nanobots se coordinan en tiempo real - **Adaptación**: Se reconfigura según la amenaza - **Recuperación**: Si uno falla, otros compensan

En este lab, crearás tu "Nanotech": un ecosistema de agentes de IA especializados que se orquestan para resolver problemas complejos.

57.3 Timeline de la Misión

Paso	Descripción	Tiempo	Completado
1	Diseñar arquitectura multi-agente	35 min	
2	Implementar orquestación inteligente	40 min	
3	Crear comunicación entre agentes	30 min	
4	Sistema de recuperación y adaptación	25 min	
Total		**130 min**	

57.4 Objetivo del Lab

Crear un ecosistema multi-agente. Al finalizar, tendrás:

1. Arquitectura de agentes especializados
2. Sistema de orquestación inteligente
3. Comunicación efectiva entre agentes
4. Mecanismos de recuperación y adaptación

57.5 Regla del Stark Protocol

> **"La verdadera inteligencia está en la colaboración."** — FRIDAY

REGLA ORO: Los mejores sistemas no son los más inteligentes, sino los que mejor se **coordinan**.

57.6 Ejercicios: El Proyecto Final

57.6.1 Aplicando Todo el Protocolo Stark

57.6.1.1 Timeline de Verificación

Paso	Descripción	Tiempo	Completado
1	Planificar proyecto	30 min	
2	Implementar con contexto	60 min	
3	Iterar hasta	45 min	
4	Documentar	30 min	
5	Reflexionar	15 min	
Total		**180 min**	

57.7 Objetivo

Construir algo completo usando TODO el Protocolo Stark.

57.8 Importante: Aplica Todo

Este lab integra todos los conceptos: contexto, piloto, iteración, límites.

57.9 1. El Escenario

Eres un developer que necesita construir un proyecto real usando IA.

El objetivo NO es el proyecto en sí. El objetivo es demostrar que puedes usar el Protocolo Stark de forma integrada.

57.10 2. Paso 1: Planificar

Tu proyecto debe: - Resolver un problema real (aunque sea simple) - Ser construido completamente con asistencia de IA - Usar TODOS los elementos del Protocolo Stark

Registro:

```
Nombre del proyecto: ___

Problema que resuelve: ___

Scope (qué incluiremos y qué no):
- Incluir: ___
- Excluir: ___

Contexto que daré a la IA:
1. Dominio: ___
2. Tech stack: ___
3. Formato esperado: ___
4. Constraints: ___
5. Limitaciones conocidas: ___
```

57.11 3. Paso 2: Implementar

Instrucciones:

1. Abre tu herramienta de IA
2. Pide opciones (piloto)
3. Elige basándote en criterios (piloto)
4. Da contexto completo
5. Itera hasta
6. Mantén la seguridad (límites)

Registro de cada interacción:

```
=== Interacción 1 ===
Mi prompt: ___
Opciones que me dio: ___
Mi decisión y por qué: ___
Resultado: ___
Crítica: ___

=== Interacción 2 ===
Mi prompt: ___
...
```

57.12 4. Paso 3: Documentar

Crea un README.md con:

1. **Descripción del proyecto** - Qué hace y por qué
2. **Contexto dado a la IA** - Qué información proporcionaste
3. **Decisiones tomadas** - Por qué elegiste cada opción
4. **Iteraciones realizadas** - Cómo mejoraste el output
5. **Límites aplicados** - Qué no compartiste y por qué
6. **Lecciones aprendidas** - Qué descubriste sobre ti mismo

57.13 5. Paso 4: Reflexionar

Responde:

57.13.1 Sobre el Proceso

```
¿Qué fue diferente esta vez comparado con cómo trabajabas antes con IA? ___

¿Qué patrones notaste que usabas sin pensar? ___

¿Qué mejorarás la próxima vez? ___
```

57.13.2 Sobre el Protocolo

```
¿Qué paso del Protocolo fue más difícil de aplicar? ___

¿Cuál fue el más útil? ___

¿Sientes que tienes más control ahora? ___
```

57.13.3 Auto-Evaluación

```
En una escala de 1-10:
- Mi uso de contexto: ___
- Mi control como piloto: ___
- Mi tolerancia a iterar: ___
- Mi conciencia de seguridad: ___
- Mi integración del Protocolo: ___
```

57.14 6. El Patrón Final

Después de este lab completo, completa:

```
El Protocolo Stark para mí significa:

1. Contexto es: ___
2. Piloto significa: ___
3. Iterar quiere decir: ___
4. Límites implica: ___
5. Síntesis se ve como: ___

Yo soy el piloto porque: ___
```

57.15 Verificación

Checklist:

- ☐ Proyecto planificado
- ☐ Al menos 3 iteraciones documentadas
- ☐ README.md completo
- ☐ Reflexiones completadas
- ☐ Auto-evaluación hecha

57.16 Entregable

Archivos: 1. `proyecto_final/` - Tu proyecto completo 2. `README.md` - Documentación completa 3. `reflexion_final.md` - Tu reflexión

---

57.17 Cierre

Stark no construyó sus armaduras en un día.

Tú no vas a dominar la colaboración con IA en un lab.

Pero ahora tienes el mapa.

El resto es práctica.

¿Estás listo para pilotar?

57.18 Recursos

- AI Collaboration Framework
- Human-AI Teams
- Iron Man Evolution - Curso Completo

58 Boss Fight 6: El Ecosistema Stark

59 Boss Fight 6: El Ecosistema Stark

59.1 Prueba Final del Nivel 6

59.2 La Situación

"No soy solo un tipo con una armadura. Soy un ecosistema." — Tony Stark

La armadura nanotecnológica no es una herramienta. Es un **ecosistema**. Miles de nanobots especializados trabajando en perfecta armonía. Cada uno hace una cosa bien. Se comunican. Se adaptan. Se recuperan.

Tu misión: Crear un ecosistema multi-agente como la armadura nanotecnológica: especializado, coordinado, adaptativo.

59.3 Misión: Sistema de Desarrollo Asistido por IA

59.3.1 Arquitectura Multi-Agente

```
                ORCHESTRATOR AGENT
            (    agentes)


  CODING          TESTING         REVIEW
  AGENT           AGENT           AGENT

- Genera        - Crea          - Revisa
  código          tests           PRs
- Refactor      - Ejecuta       - Sugiere
                  suites          cambios


  DOC             SECURITY        DEPLOY
```

```
      AGENT           AGENT           AGENT

    - Genera        - Escanea       - Despliega
      docs            vulnera-        a prod
    - Actualiza       bilidades     - Monitorea
      README
```

59.3.2 Implementación del Orquestador

```python
from typing import Dict, List, Any
from dataclasses import dataclass
from enum import Enum

class AgentType(Enum):
    CODING = "coding"
    TESTING = "testing"
    REVIEW = "review"
    DOCUMENTATION = "documentation"
    SECURITY = "security"
    DEPLOYMENT = "deployment"

@dataclass
class AgentTask:
    """Tarea para un agente"""
    agent_type: AgentType
    description: str
    priority: int
    context: Dict[str, Any]
    dependencies: List[str] = None

@dataclass
class AgentResult:
    """Resultado de un agente"""
    agent_type: AgentType
    success: bool
    output: Any
    metrics: Dict[str, float]
    next_tasks: List[AgentTask] = None

class StarkOrchestrator:
    """Orquestador principal del ecosistema Stark"""

    def __init__(self):
        self.agents = self._initialize_agents()
```

```
        self.task_queue = []
        self.results = {}

    def _initialize_agents(self) -> Dict[AgentType, Any]:
        """Inicializa todos los agentes"""
        return {
            AgentType.CODING: CodingAgent(),
            AgentType.TESTING: TestingAgent(),
            AgentType.REVIEW: ReviewAgent(),
            AgentType.DOCUMENTATION: DocumentationAgent(),
            AgentType.SECURITY: SecurityAgent(),
            AgentType.DEPLOYMENT: DeploymentAgent()
        }

    async def process_feature(self, feature_spec: Dict) -> Dict:
        """Procesa una feature completa"""
        # Fase 1: Planificación
        plan = await self._plan_execution(feature_spec)

        # Fase 2: Ejecución coordinada
        results = await self._execute_plan(plan)

        # Fase 3: Consolidación
        final_result = await self._consolidate_results(results)

        return final_result

    async def _plan_execution(self, spec: Dict) -> List[AgentTask]:
        """Planifica ejecución multi-agente"""
        tasks = []

        # Coding Agent genera código
        tasks.append(AgentTask(
            agent_type=AgentType.CODING,
            description=f"Implementar feature: {spec['name']}",
            priority=1,
            context=spec
        ))

        # Testing Agent crea tests
        tasks.append(AgentTask(
            agent_type=AgentType.TESTING,
            description="Crear suite de tests",
            priority=2,
            context={"feature": spec},
            dependencies=["coding"]
        ))
```

```
        # Review Agent revisa código
        tasks.append(AgentTask(
            agent_type=AgentType.REVIEW,
            description="Revisar implementación",
            priority=3,
            context={"code": "from_coding"},
            dependencies=["coding", "testing"]
        ))

        return tasks

    async def _execute_plan(self, tasks: List[AgentTask]) -> Dict:
        """Ejecuta plan con dependencias"""
        results = {}

        for task in sorted(tasks, key=lambda t: t.priority):
            # Verificar dependencias
            if task.dependencies:
                for dep in task.dependencies:
                    if dep not in results:
                        raise Exception(f"Dependencia no cumplida: {dep}")

            # Ejecutar agente
            agent = self.agents[task.agent_type]
            result = await agent.execute(task)
            results[task.agent_type.value] = result

        return results
```

59.3.3 Comunicación entre Agentes

```
class AgentMessage:
    """Mensaje entre agentes"""

    def __init__(self, from_agent: AgentType, to_agent: AgentType,
                 content: Any, message_type: str):
        self.from_agent = from_agent
        self.to_agent = to_agent
        self.content = content
        self.message_type = message_type
        self.timestamp = datetime.now()
        self.id = str(uuid.uuid4())

class MessageBus:
    """Bus de comunicación entre agentes"""
```

```
    def __init__(self):
        self.messages = []
        self.subscribers = {}

    def subscribe(self, agent_type: AgentType, callback: callable):
        """Suscribe agente a mensajes"""
        if agent_type not in self.subscribers:
            self.subscribers[agent_type] = []
        self.subscribers[agent_type].append(callback)

    def publish(self, message: AgentMessage):
        """Publica mensaje a agentes suscriptores"""
        self.messages.append(message)

        if message.to_agent in self.subscribers:
            for callback in self.subscribers[message.to_agent]:
                callback(message)
```

59.4 Escenarios de Prueba

59.4.1 Scenario 1: Feature Completa

```
Input: "Crear API de usuarios con CRUD completo"

Expected Flow:
1. Coding Agent → Genera models, routes, controllers
2. Testing Agent → Crea tests unitarios y de integración
3. Review Agent → Sugiere mejoras de seguridad
4. Doc Agent → Genera Swagger/OpenAPI docs
5. Security Agent → Escanea vulnerabilidades
6. Deployment Agent → Despliega a staging
```

59.4.2 Scenario 2: Bug Fix

```
Input: "Error 500 en endpoint /users/:id"

Expected Flow:
1. Review Agent → Analiza logs y código
2. Coding Agent → Corrige bug
3. Testing Agent → Crea test que reproduce bug
4. Security Agent → Verifica no hay regresión de seguridad
```

59.4.3 Scenario 3: Refactoring

```
Input: "Optimizar consultas a base de datos"

Expected Flow:
1. Review Agent → Identifica N+1 queries
2. Coding Agent → Refactoriza con eager loading
3. Testing Agent → Verifica performance mejorada
4. Doc Agent → Actualiza documentación
```

59.5 Logro Desbloqueado: "Ecosystem Master"

59.5.1 Requisitos

- ☐ 6+ agentes especializados funcionando
- ☐ Orquestador coordinando tareas
- ☐ Comunicación entre agentes
- ☐ Sistema adaptativo y recuperable
- ☐ Tests de integración pasando

59.5.2 Recompensa

- **+500 XP**
- **Logro "Ecosystem Master"**
- **Certificado "Iron Man Evolution"**

59.5.3 ¡Felicidades, Tony Stark!

> *"Ahora soy Iron Man."* — Tony Stark

Has completado Iron Man Evolution. De la cueva al nanotech. De scripts básicos a ecosistemas multi-agente.

→ Ver Certificado de Finalización

59.6 Métricas Finales

59.6.1 Tu Progreso

```
## Estadísticas de Ecosistema
- **Agentes creados**: ___
- **Tareas orquestadas**: ___
- **Tiempo de ejecución promedio**: ___
- **Tasa de éxito**: ___%

## Logros Desbloqueados
- [ ] Cave Survivor (Nivel 1)
- [ ] Prototype to Product (Nivel 2)
- [ ] Architect (Nivel 3)
- [ ] AI Architect (Nivel 4)
- [ ] Security Guardian (Nivel 5)
- [ ] Ecosystem Master (Nivel 6)

## XP Total: ___
```

Part VII

Quizzes y Evaluaciones

60 Quiz 1: El Despertar

60.1 Pregunta 1

¿Cuál es el PRINCIPIO FUNDAMENTAL del Protocolo Stark?

A) La IA debe hacer todo el trabajo
B) Tú pilotas, la IA amplifica
C) Siempre dar instrucciones específicas
D) Nunca iterar más de una vez

Respuesta

B) Tú pilotas, la IA amplifica

60.2 Pregunta 2

¿Por qué Stark estableció un sistema de organización antes de trabajar con J.A.R.V.I.S.?

A) No tenía nada mejor que hacer
B) Para poder encontrar sesiones anteriores fácilmente
C) Porque J.A.R.V.I.S. se lo pidió
D) No estableció ningún sistema

Respuesta

B) Para poder encontrar sesiones anteriores fácilmente

60.3 Pregunta 3

¿Qué revelan las conversaciones de Stark con J.A.R.V.I.S.?

A) Que J.A.R.V.I.S. siempre tiene razón
B) Que Stark nunca comete errores
C) Que cada conversación revela algo nuevo que faltaba
D) Que la IA no es útil

Respuesta

C) Que cada conversación revela algo nuevo que faltaba

60.4 Pregunta 4

¿Qué debe incluir siempre tu "contexto default"?

A) Solo el problema a resolver
B) Dominio, formato, constraints y limitaciones
C) Tu nombre y email
D) El precio del proyecto

Respuesta

B) Dominio, formato, constraints y limitaciones

60.5 Pregunta 5

En la Quinta Conversación, ¿por qué falló el script de diagnósticos?

A) J.A.R.V.I.S. no sabía programar
B) Los logs estaban en formato PST y el script asumía UTC
C) Stark escribió mal el código
D) El script estaba bien, los logs estaban mal

Respuesta

B) Los logs estaban en formato PST y el script asumía UTC

60.6 Pregunta 6

¿Cuál es el error más común según el módulo?

A) Pedir demasiado en una sola vez
B) No pedir nada
C) No dar contexto específico
D) Copiar y pegar código

Respuesta

C) No dar contexto específico

60.7 Pregunta 7

¿Qué debería hacer la IA ANTES de actuar según J.A.R.V.I.S.?

A) Nada especial
B) Hacer preguntas
C) Ejecutar inmediatamente
D) Ignorar al usuario

Respuesta

B) Hacer preguntas

60.8 Pregunta 8

¿Por qué Stark dijo "Buen punto" al final de la Quinta Conversación?

A) Porque J.A.R.V.I.S. tenía razón en que no preguntó por las limitaciones
B) Porque estaba de acuerdo en que el script estaba bien
C) Porque quería ser educado
D) No dijo eso

Respuesta

A) Porque J.A.R.V.I.S. tenía razón en que no preguntó por las limitaciones

---

60.9 Pregunta 9 - True/False

¿Verdadero o Falso?

En el Protocolo Stark, el piloto (usuario) siempre debe dar instrucciones perfectas desde el primer intento.

Respuesta

Falso. El piloto itera y refina. La perfección no existe en el primer intento — por eso existe la iteración.

60.10 Pregunta 10 - True/False

¿Verdadero o Falso?

J.A.R.V.I.S. debe ejecutar inmediatamente todo lo que Stark pide, sin hacer preguntas.

Respuesta

Falso. J.A.R.V.I.S. debe HACER PREGUNTAS si falta contexto. Esto es parte del Protocolo Stark.

60.11 Pregunta 11 - Matching

Relaciona los conceptos con sus definiciones:

Concepto	Definición
A. Contexto	1. La instrucción que das a la IA
B. Prompt	2. Información que mejora los resultados
C. Iteración	3. Repetición para mejorar el resultado
D. Output	4. Lo que la IA genera

Respuesta

A-2, B-1, C-3, D-4

60.12 Pregunta 12 - Matching

Relaciona el error con la solución:

Error Común	Solución
A. Request vago	1. Dar contexto específico
B. Sin limitaciones	2. Especificar constraints
C. Output imperfecto	3. Iterar y refinar
D. Prompt largo	4. Ser conciso y claro

Respuesta

A-1, B-2, C-3, D-4

60.13 Pregunta 13 - Code Output Prediction

¿Qué output produce este código?

```
print("JARVIS"[::-1])
```

Respuesta

"SIVRAJ" — El slicing [::-1] invierte la cadena.

60.14 Pregunta 14 - Case de Estudio

Contexto: Stark escribe el siguiente prompt: "Escríbeme código para validar emails"

El problema principal es:

A) No especifica el lenguaje de programación
B) No define qué constituye un email válido
C) El prompt es muy corto
D) No hay problema, el código funcionará

Respuesta

B) No define qué constituye un email válido — ¿Solo formato básico? ¿Dominios corporativos? ¿Caracteres especiales permitidos? Sin contexto, la IA asume lo más simple y puede fallar en casos reales.

60.15 Pregunta 15 - Fill-in-the-Blank

Completa:

La fórmula del contexto perfecto es: _____ **+ Con qué + Cómo + Edge Cases**

Respuesta

Qué — La fórmula completa es: Qué + Con qué + Cómo + Edge Cases

61 Quiz 2: Contexto es Rey

61.1 Pregunta 1

¿Qué sucede cuando Stark pide "valide emails" sin contexto?

A) La IA genera código perfecto
B) La IA pregunta por las limitaciones
C) El código generado funciona pero tiene limitaciones
D) El código falla completamente

Respuesta

C) El código generado funciona pero tiene limitaciones

61.2 Pregunta 2

¿Qué tipo de información hace falta en el request "valide emails" inicial?

A) El lenguaje de programación
B) El dominio de uso, formatos específicos y constraints
C) El nombre del archivo
D) La fecha de entrega

Respuesta

B) El dominio de uso, formatos específicos y constraints

61.3 Pregunta 3

¿Por qué J.A.R.V.I.S. preguntó sobre el formato de los logs?

A) Porque no tenía nada que hacer
B) Porque Stark no especificó el timezone
C) Porque J.A.R.V.I.S. no sabía programar
D) Porque los logs estaban corruptos

Respuesta

B) Porque Stark no especificó el timezone

61.4 Pregunta 4

¿Cuál es la "Fórmula del Contexto Perfecto"?

A) Qué + Para qué + Por qué
B) Qué + Con qué + Cómo + Edge cases
C) Quién + Cuándo + Dónde
D) Nada de lo anterior

Respuesta

B) Qué + Con qué + Cómo + Edge cases

61.5 Pregunta 5

¿Qué pasa cuando Stark da contexto específico como "dominios corporativos"?

A) El resultado es peor
B) El resultado es igual
C) El resultado mejora dramáticamente
D) La IA se confunde más

Respuesta

C) El resultado mejora dramáticamente

61.6 Pregunta 6

¿Por qué la IA hace preguntas durante la conversación?

A) Porque está aburrida
B) Porque no puede anticipar necesidades sin contexto
C) Porque quiere retrasar el trabajo
D) Porque no entiende al usuario

Respuesta

B) Porque no puede anticipar necesidades sin contexto

61.7 Pregunta 7

¿Cuál es la diferencia entre un request vago y uno efectivo?

A) No hay diferencia
B) El efectivo incluye contexto específico
C) El vago es más largo
D) El efectivo es más corto

Respuesta

B) El efectivo incluye contexto específico

61.8 Pregunta 8

¿Qué aprende el estudiante al observar las conversaciones de Stark?

A) Que J.A.R.V.I.S. siempre tiene razón
B) Que el contexto mejora los resultados
C) Que Stark nunca se equivoca
D) Que la IA no es útil

Respuesta

B) Que el contexto mejora los resultados

62 Quiz 3: El Piloto

62.1 Pregunta 1

¿Cuál es la diferencia entre SUGERIR y DECIDIR?

A) Son lo mismo
B) Sugerir es dar opciones, decidir es elegir
C) Decidir es dar opciones
D) No importa la diferencia

Respuesta

B) Sugerir es dar opciones, decidir es elegir

62.2 Pregunta 2

¿Por qué J.A.R.V.I.S. hace tres preguntas antes de ejecutar "acceso root"?

A) Porque quiere retrasar
B) Para asegurar que la acción es segura
C) Porque está aburrido
D) Porque no puede ejecutar

Respuesta

B) Para asegurar que la acción es segura

62.3 Pregunta 3

¿Quién es RESPONSABLE de las decisiones en el Protocolo Stark?

A) J.A.R.V.I.S.
B) La IA
C) El usuario (Stark)
D) Nadie

Respuesta

C) El usuario (Stark)

62.4 Pregunta 4

¿Qué significa "pilotar" en este contexto?

A) Dejar que la IA haga todo
B) Mantener control sobre las decisiones importantes
C) Ignorar a la IA
D) Copiar todo lo que dice la IA

Respuesta

B) Mantener control sobre las decisiones importantes

62.5 Pregunta 5

¿Qué hace J.A.R.V.I.S. cuando Stark tiene que elegir entre Blue-green y Canary?

A) Decide por él
B) Le da los datos para que él decida
C) No dice nada
D) Dice que ambas opciones son malas

Respuesta

B) Le da los datos para que él decida

62.6 Pregunta 6

¿Por qué Stark dice “ ” cuando Pepper pregunta sobre el copiloto que sabe más?

A) Porque está confundido
B) Porque reconoce que el conocimiento no exime de responsabilidad
C) Porque no sabe la respuesta
D) Porque está bromeando

Respuesta

B) Porque reconoce que el conocimiento no exime de responsabilidad

---

62.7 Pregunta 7

¿Qué tipo de request da MÁS control al usuario?

A) “Haz X”
B) “Dame opciones para X”
C) “Elimina todo”
D) “No sé qué hacer”

Respuesta

B) “Dame opciones para X”

62.8 Pregunta 8

¿Por qué J.A.R.V.I.S. propone opciones en lugar de ejecutar directamente?

A) Porque no puede ejecutar
B) Porque el piloto debe elegir
C) Porque está ocupado
D) Porque no quiere trabajar

Respuesta

B) Porque el piloto debe elegir

62.9 Pregunta 9 - True/False

¿Verdadero o Falso?

En el modelo piloto-copiloto, la IA (copiloto) debe tomar las decisiones finales.

Respuesta

Falso. El piloto (usuario) toma las decisiones finales. La IA ejecuta y sugiere, pero no decide.

62.10 Pregunta 10 - True/False

¿Verdadero o Falso?

Si la IA tiene más conocimiento que el usuario, la responsabilidad de las decisiones sigue siendo del usuario.

Respuesta

Verdadero. El conocimiento de la IA no exime al usuario de responsabilidad. Stark lo entendió cuando Pepper le cuestionó sobre el "copiloto que sabe más".

62.11 Pregunta 11 - True/False

¿Verdadero o Falso?

"Dar opciones" en lugar de "ejecutar directamente" hace que el usuario tenga MENOS control.

Respuesta

Falso. "Dar opciones" da MÁS control al usuario porque le permite elegir entre alternativas informadas.

Rol	Responsabilidad

62.12 Pregunta 12 - Matching

Relaciona los roles con sus responsabilidades:

Rol	Responsabilidad
A. Piloto	1. Sugiere y ejecuta
B. Copiloto	2. Decide y lidera
C.IA	3. Nunca decide por sí misma

Respuesta

A-2, B-1, C-3

62.13 Pregunta 13 - Matching

Relaciona el tipo de request con el nivel de control:

Request	Nivel de Control
A. “Haz X”	1. Máximo control
B. “Dame opciones para X”	2. Mínimo control
C. “Elimina todo”	3. Control medio
D. “Aquí tienes los pros y contras de X, ¿qué eliges?”	4. Información para decidir

Respuesta

A-2, B-1, C-2, D-4

62.14 Pregunta 14 - Code Output Prediction

¿Qué output produce este código?

```
opciones = ["blue-green", "canary", "rolling"]
print(opciones[1])
```

Respuesta

"canary" — Los índices en Python comienzan en 0, así que [1] devuelve el segundo elemento.

62.15 Pregunta 15 - Caso de Estudio

Contexto: J.A.R.V.I.S. propone dos estrategias de deployment para actualizar el reactor ARC: - **Blue-green**: Dos entornos idénticos, cambio instantáneo entre ellos - **Canary**: Primero al 5% de usuarios, luego 20%, luego 100%

Si eres el piloto, ¿cuál estrategia elegías y por qué?

A) Blue-green porque es más simple
B) Canary porque reduce el riesgo de fallos generales
C) Cualquiera, no importa
D) Ninguna, mejor no actualizar

Respuesta

B) Canary — Permite detectar problemas antes de afectar a todos los usuarios. PERO la respuesta correcta depende del contexto: criticidad del sistema, tolerancia a fallos, recursos para mantener dos entornos, etc. J.A.R.V.I.S. debe dar datos, el piloto decide.

62.16 Pregunta 16 - Fill-in-the-Blank

Completa:

En el modelo piloto-copiloto, el **____** toma decisiones y el **____** ejecuta y sugiere.

Respuesta

piloto y **copiloto**

63 Quiz 4: Refinamiento

63.1 Pregunta 1

¿Cuál es la verdad sobre el primer output de una IA?

A) Siempre es perfecto
B) Es solo el comienzo, no el final
C) Nunca es útil
D) Debe usarse tal cual

Respuesta

B) Es solo el comienzo, no el final

63.2 Pregunta 2

¿Qué hace Stark cuando el primer código no es óptimo?

A) Lo usa de todas formas
B) Lo critica y pide mejora
C) Elimina todo
D) Pide otro código diferente

Respuesta

B) Lo critica y pide mejora

63.3 Pregunta 3

¿Por qué Stark no pidió TODO en la primera iteración?

A) No sabía lo que necesitaba
B) Quería iterar para descubrir requisitos
C) Quería perder tiempo
D) No confía en la IA

Respuesta

B) Quería iterar para descubrir requisitos

63.4 Pregunta 4

¿Qué revela el primer output sobre el contexto inicial?

A) Que el contexto estaba perfecto
B) Que faltaba algo
C) Que la IA no sirve
D) Que el código está bien

Respuesta

B) Que faltaba algo

63.5 Pregunta 5

¿Cuántas versiones tomó Stark para la función de primos?

A) 1
B) 2
C) 3
D) Muchas

Respuesta

C) 3

63.6 Pregunta 6

¿Por qué el sistema de logging evolucionó de simple a robusto?

A) Porque J.A.R.V.I.S. lo hizo solo
B) Porque cada iteración reveló nuevos requisitos
C) Porque Stark cambió de opinión
D) Porque era necesario

Respuesta

B) Porque cada iteración reveló nuevos requisitos

63.7 Pregunta 7

¿Cuándo se considera "suficientemente bueno" un output?

A) Nunca
B) Cuando cumple los requisitos de uso
C) Cuando es perfecto
D) Cuando toma mucho tiempo

Respuesta

B) Cuando cumple los requisitos de uso

63.8 Pregunta 8

¿Qué es "iteration is the process"?

A) Que la perfección existe
B) Que el primer intento es el último
C) Que la mejora continua es normal y necesaria
D) Que no hay que iterar

Respuesta

C) Que la mejora continua es normal y necesaria

64 Quiz 5: Límites

64.1 Pregunta 1

¿Qué debe hacerse ANTES de compartir información con una IA?

A) Compartir todo para que trabaje mejor
B) Pensar en las consecuencias de una filtración
C) Ignorar los riesgos
D) Compartir solo passwords

Respuesta

B) Pensar en las consecuencias de una filtración

64.2 Pregunta 2

¿Cuál es la forma correcta de compartir problemas de seguridad sin exponer datos?

A) Compartir todo tal cual
B) Describir el problema sin datos sensibles
C) No usar IA para esto
D) Ignorar la seguridad

Respuesta

B) Describir el problema sin datos sensibles

64.3 Pregunta 3

¿Qué nivel de riesgo tiene compartir API keys?

A) Bajo
B) Medio
C) Alto
D) Crítico

Respuesta

D) Crítico

64.4 Pregunta 4

¿Qué hace J.A.R.V.I.S. cuando Stark quiere compartir passwords?

A) Los acepta inmediatamente
B) Pregunta si está seguro
C) Los rechaza sin explicación
D) Los comparte con otros

Respuesta

B) Pregunta si está seguro

64.5 Pregunta 5

¿Cómo se puede obtener ayuda con passwords sin compartirlos?

A) Compartirlos de todas formas
B) Describir los requisitos de validación sin datos reales
C) No se puede
D) Solo usar passwords públicos

Respuesta

B) Describir los requisitos de validación sin datos reales

64.6 Pregunta 6

¿Qué es "professional responsibility" para J.A.R.V.I.S.?

A) Hacer todo lo que Stark pide
B) Proteger incluso cuando el usuario no se protege
C) Ignorar los riesgos
D) Compartir información

Respuesta

B) Proteger incluso cuando el usuario no se protege

64.7 Pregunta 7

¿Qué significa que J.A.R.V.I.S. "sabe todo lo que necesita saber"?

A) Que Stark le cuenta todo
B) Que tiene acceso a toda la información
C) Que solo sabe lo necesario, no todo
D) Que no sabe nada

Respuesta

C) Que solo sabe lo necesario, no todo

64.8 Pregunta 8

¿Cuál es el mejor approach cuando necesitas revisar código propietario?

A) Compartir todo sin anonimizar
B) Anonimizar patrones sin detalles sensibles
C) No usar IA para nada sensible
D) Solo usar IA para código público

Respuesta

B) Anonimizar patrones sin detalles sensibles

65 Quiz 6: Síntesis

65.1 Pregunta 1

¿Qué hace Stark en la Llamada Maestra que es diferente?

A) Pide solo una cosa
B) Da contexto completo desde el inicio
C) No da contexto
D) Pide que J.A.R.V.I.S. adivine

Respuesta

B) Da contexto completo desde el inicio

65.2 Pregunta 2

¿Qué tipo de sistema construye Stark en la Síntesis?

A) Sistema de login simple
B) Sistema de detección de anomalías para el reactor
C) Página web básica
D) Calculadora

Respuesta

B) Sistema de detección de anomalías para el reactor

65.3 Pregunta 3

¿Qué propone J.A.R.V.I.S. para el sistema de anomalías?

A) Código sin arquitectura
B) Arquitectura de múltiples capas
C) Solo un script
D) Nada

Respuesta

B) Arquitectura de múltiples capas

65.4 Pregunta 4

¿Quién decide la arquitectura final?

A) J.A.R.V.I.S.
B) Stark (el piloto)
C) Pepper
D) Nadie

Respuesta

B) Stark (el piloto)

65.5 Pregunta 5

¿Qué significa integrar TODOS los conceptos del Protocolo Stark?

A) Usar solo contexto
B) Usar solo piloto
C) Usar contexto + piloto + iteración + límites
D) No usar nada

Respuesta

C) Usar contexto + piloto + iteración + límites

65.6 Pregunta 6

¿Qué hace Stark cuando el recall es bajo (78%)?

A) Lo acepta
B) Pide opciones para mejorar
C) Elimina el sistema
D) Culpa a J.A.R.V.I.S.

Respuesta

B) Pide opciones para mejorar

65.7 Pregunta 7

¿Qué significa "convertirse en piloto"?

A) Dejar que la IA haga todo
B) Mantener control mientras usas la IA como amplificador
C) Ignorar a la IA
D) Copiar todo ciegamente

Respuesta

B) Mantener control mientras usas la IA como amplificador

65.8 Pregunta 8

¿Cuál es la Fórmula Final del Protocolo Stark?

A) Contexto = Éxito
B) Piloto + Contexto + Iteración + Límites = Éxito
C) IA = Todo
D) No hay fórmula

Respuesta

B) Piloto + Contexto + Iteración + Límites = Éxito

65.9 Pregunta 9

¿Por qué Stark dice "J.A.R.V.I.S. me hace más inteligente"?

A) Le da conocimiento
B) Le amplifica sus capacidades
C) Le da dinero
D) Le construye las armaduras

Respuesta

B) Le amplifica sus capacidades

65.10 Pregunta 10

¿Cuál es el mensaje final del curso?

A) La IA reemplazará a los humanos
B) La IA debe usarse con control y responsabilidad
C) No uses IA
D) Confía ciegamente en la IA

Respuesta

B) La IA debe usarse con control y responsabilidad

Part VIII

Referencias y Recursos

66 PRD - Product Requirements Document

67 PRD

67.1 Product Requirements Document

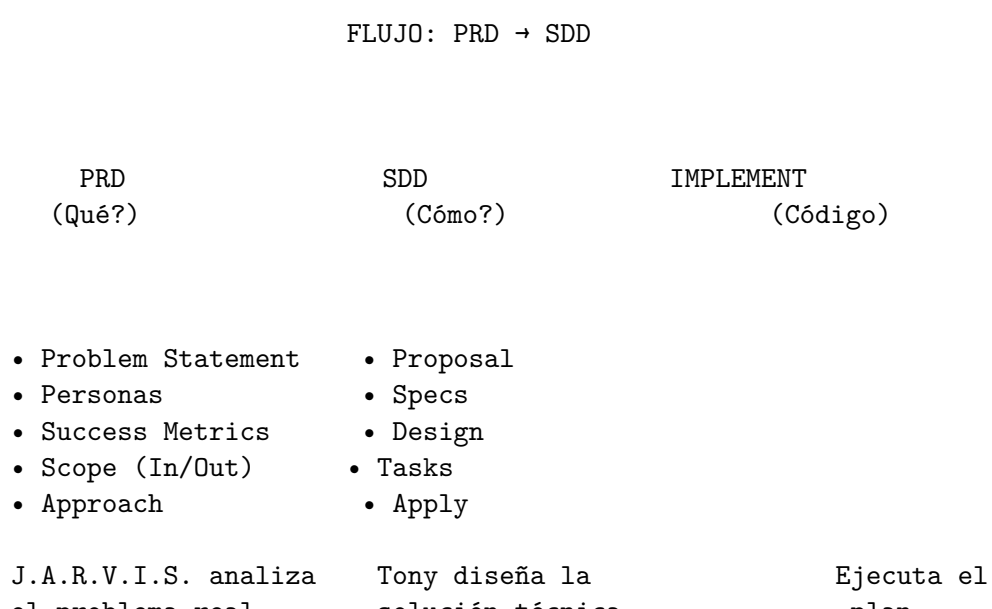
67.2 Descripción

El **PRD (Product Requirements Document)** es el documento fundamental que define QUÉ vamos a construir y POR QUÉ. Es el puente entre el problema de negocio y la solución técnica.

¿Por qué existe el PRD? - SDD te dice CÓMO construir bien - **PRD te dice QUÉ construir**

Sin PRD, corres el riesgo de construir algo que nadie necesita (el error de Ultron: creó su solución sin entender el objetivo real).

67.3 PRD vs SDD: ¿Cuál viene primero?



Analogía Iron Man: - **PRD**: Tony Stark analizando la amenaza antes de diseñar la armadura - **SDD**: Los planos técnicos de la armadura - **Implementación**: Construir la armadura

> *"Si no puedes explicar el problema, no puedes resolverlo."* — J.A.R.V.I.S.

67.4 Estructura del PRD

67.4.1 1. Intent & Problem Statement

```
## 1. Intent & Problem Statement

### Problema Actual
[Describe el problema que resuelve]

### Impacto del Problema
[Quién se ve afectado y cómo]

### Propuesta de Valor
[Por qué esto vale la pena]
```

Ejemplo - Iron Man Theme:

```
## 1. Intent & Problem Statement

### Problema Actual
Los escudos de las armaduras Mark 7-42 no se actualizan automáticamente cuando se detecta

### Impacto del Problema
- Soldados en campo reciben escudos desactualizados
- 3 incidentes documentados de fallos por brecha de seguridad
- 15 minutos de tiempo promedio de respuesta

### Propuesta de Valor
Implementar sistema de actualización automática de escudos que:
- Detecte amenazas en tiempo real
- Descargue parches de seguridad en < 30 segundos
- Reduzca incidentes de seguridad a 0
```

67.4.2 2. Personas & Stakeholders

```
## 2. Personas & Stakeholders

### Usuario Primario
| Campo | Descripción |
|-------|-------------|
| **Nombre** | [Nombre] |
| **Rol** | [Rol en la organización] |
| **Objetivos** | [Qué quiere lograr] |
| **Frustraciones** | [Qué le frustra] |
| **Contexto** | [Cuándo usa el sistema] |

### Stakeholders Clave
| Stakeholder | Interés | Influencia |
|-------------|---------|-------------|
| [Nombre]    | [Qué gana]| [Alta/Media/Baja] |
```

Ejemplo:

```
## 2. Personas & Stakeholders

### Usuario Primario: Soldado de Campo
| Campo | Descripción |
|-------|-------------|
| **Nombre** | Soldado Mk. 7 |
| **Rol** | Infantry / Flying |
| **Objetivos** | Estar protegido en todo momento |
| **Frustraciones** | Escudos que no se actualizan, demoras en combate |
| **Contexto** | Situaciones de combate, misiones de reconocimiento |

### Stakeholders
| Stakeholder | Interés | Influencia |
|-------------|---------|-------------|
| Tony Stark | Seguridad máxima | Alta |
| Pepper Potts | ROI, eficiencia | Alta |
| Happy Hogan | Logística | Media |
| Soldados | Supervivencia | Alta |
```

67.4.3 3. Success Metrics (KPIs)

```
## 3. Success Metrics (KPIs)

### Métricas de Negocio
| Métrica | Target | Cómo se mide |
|---------|--------|--------------|
| [Métrica 1] | [Número] | [Método] |

### Métricas Técnicas
| Métrica | Target | Cómo se mide |
|---------|--------|--------------|
| [Métrica 1] | [Número] | [Método] |

### Métricas de Usuario
| Métrica | Target | Cómo se mide |
|---------|--------|--------------|
| [Métrica 1] | [Número] | [Método] |
```

Ejemplo SMART:

```
## 3. Success Metrics (KPIs)

### Métricas de Negocio
| Métrica | Target | Cómo se mide |
|---------|--------|--------------|
| Reducción de incidentes | 0 incidentes/mes | Reportes de campo |
| Tiempo de actualización | < 30 segundos | Logs del sistema |
| Satisfacción de usuario | > 4.5/5 | Encuesta post-misión |

### Métricas Técnicas
| Métrica | Target | Cómo se mide |
|---------|--------|--------------|
| Uptime del sistema | 99.9% | Monitoring |
| Latencia de descarga | < 5s | APM |
| Cobertura de tests | > 80% | CI/CD |

### Definition of Done
- [ ] Sistema en producción
- [ ] 0 incidentes en 30 días
- [ ] Documentación completa
- [ ] Capacitación dada
```

67.4.4 4. Scope (IN/OUT)

```
## 4. Scope

### In Scope (qué incluye)
- [ ] Feature 1
- [ ] Feature 2

### Out of Scope (qué NO incluye)
- [ ] Feature 1
- [ ] Feature 2

###nice to Have (prioridad baja)
- [ ] Feature 1

### Dependencias Externas
- [ ] Sistema A (deadline: fecha)
- [ ] Sistema B (pendiente de aprobación)
```

Ejemplo:

```
## 4. Scope

### In Scope
- [ ] Sistema de detección de amenazas
- [ ] Pipeline de descarga de actualizaciones
- [ ] Integración con armaduras Mark 7-50
- [ ] Panel de administración
- [ ] Logs de auditoría

### Out of Scope
- [ ] Actualización de firmware de hardware (solo software)
- [ ] Integración con sistemas legacy (v2)
- [ ] Aplicación móvil (v2)
- [ ] Soporte para armaduras older than Mark 7

### Nice to Have (v2)
- [ ] AI predictivo para amenazas
- [ ] Modo offline con caché local

### Dependencias Externas
- [ ] Servidor de parches de Stark Industries (listo)
- [ ] API de threat intelligence (en desarrollo)
```

67.4.5 5. High-Level Approach

```
## 5. High-Level Approach

### Arquitectura Propuesta
```

[Diagrama de arquitectura]

67.4.6 Stack Tecnológico

- **Frontend**: [Tech]
- **Backend**: [Tech]
- **Database**: [Tech]
- **Infra**: [Tech]

67.4.7 Decisiones Clave

Decisión	Alternativa considerada	Justificación
[Decisión]	[Alternativa]	[Justificación]

Ejemplo:

```markdown
## 5. High-Level Approach

### Arquitectura Propuesta
```

ARQUITECTURA: SHIELD UPDATE

Threat DB Detector Queue Worker

Cache Armadura (Subscriber)

```
### Stack Tecnológico
- **Backend**: FastAPI (Python 3.11+)
- **Message Queue**: Redis Streams
- **Database**: PostgreSQL
- **Cache**: Redis
- **CI/CD**: GitHub Actions

### Decisiones Clave
| Decisión | Alternativa | Justificación |
```

```
|----------|-------------|---------------|
| Redis Streams | RabbitMQ | Mejor rendimiento para streams |
| FastAPI | Express | native async, mejor para IA |
```

67.4.8 6. Acceptance Criteria (Given/When/Then)

```
## 6. Acceptance Criteria

### Criterios de Aceptación
| ID | Given | When | Then |
|----|-------|------|------|
| AC-001 | [Condición inicial] | [Acción] | [Resultado esperado] |
```

Formato Gherkin:

```
Feature: Nombre del feature

  Scenario: Escenario 1
    Given [condición inicial]
    When [acción del usuario]
    Then [resultado esperado]
```

Ejemplo:

```
## 6. Acceptance Criteria

### Criterios de Aceptación

| ID | Given | When | Then |
|----|-------|------|------|
| AC-001 | Armadura Mark 7 conectada a red | Nueva amenaza detectada | Escudo actualizado
| AC-002 | Sin conexión a internet | Intento de actualización | Mensaje de error claro +
| AC-003 | Múltiples armaduras | Actualización masiva | Todas completan sin conflicto |
| AC-004 | Usuario no autorizado | Intenta acceder a panel | Acceso denegado (401) |

### Gherkin Scenarios

```gherkin
Feature: Auto-Update de Escudos

  Scenario: Actualización exitosa
    Given una armadura Mark 7 conectada a la red
    And el sistema detecta una nueva amenaza
```
```

```
    When el sistema descarga el parche de seguridad
    Then el escudo se actualiza automáticamente
    And el usuario recibe notificación de "Escudo actualizado"

  Scenario: Sin conexión - Modo offline
    Given una armadura Mark 7 sin conexión a internet
    When el sistema detecta nueva amenaza
    Then el sistema usa el último escudo conocido
    And el usuario recibe notificación de "Modo offline"
```

67.4.9 7. Risks & Constraints

```
## 7. Risks & Constraints

### Riesgos Identificados
| Riesgo | Severidad | Probabilidad | Mitigación |
|--------|-----------|--------------|------------|
| [Riesgo] | [Alta/Media] | [Alta/Media] | [Cómo mitigar] |

### Restricciones
| Restricción | Descripción | Impacto |
|-------------|-------------|---------|
| [Restricción]| [Qué limita] | [Impacto] |

### Supuestos
- [ ] Supuesto 1
- [ ] Supuesto 2
```

Ejemplo:

```
## 7. Risks & Constraints

### Riesgos Identificados
| Riesgo | Severidad | Probabilidad | Mitigación |
|--------|-----------|--------------|------------|
| Latencia en descarga | Alta | Media | Cache local + retry logic |
| Fallo en servidor | Alta | Baja | Fallback a CDN externo |
| Conflicto de versiones | Media | Media | Versionado SemVer |
| Datos de amenaza corruptos | Alta | Baja | Validación de firma digital |

### Restricciones
| Restricción | Descripción | Impacto |
|-------------|-------------|---------|
```

```
| Tamaño de parche | Max 50MB | Requiere compresión |
| Latencia red | Max 100ms | CDN cercano |
| Compatibilidad | Mark 7+ solo | Requiere migración |

### Supuestos
- [ ] Los servidores de Stark Industries están disponibles
- [ ] Las armaduras tienen conectividad constante
- [ ] El threat database se actualiza daily
```

67.4.10 8. Dependencies & Approvals

```
## 8. Dependencies & Approvals

### Dependencias Internas
| Dependencia | Equipo | Status | Deadline |
|-------------|--------|--------|----------|
| [Dependencia]| [Equipo]| [Ready] | [Fecha] |

### Dependencias Externas
| Dependencia | Vendor | Status | Deadline |
|-------------|--------|--------|----------|
| [Dependencia]| [Vendor]| [Ready] | [Fecha] |

### Aprobaciones Requeridas
| Aprobación | Approver | Status |
|------------|----------|--------|
| [Aprobación]| [Nombre]| [Pending] |

### Timeline
- **Kickoff**: [Fecha]
- **PRD Approval**: [Fecha]
- **Development Start**: [Fecha]
- **Release**: [Fecha]
```

Ejemplo:

```
## 8. Dependencies & Approvals

### Dependencias Internas
| Dependencia | Equipo | Status | Deadline |
|-------------|--------|--------|----------|
| API de threat Intel | Security Team | Ready | - |
| Cache infrastructure | Infra Team | Ready | - |
```

```
| Acceso a logs | Ops Team | In Progress | 15 Mar |

### Dependencias Externas
| Dependencia | Vendor | Status | Deadline |
|-------------|--------|--------|----------|
| CDN Cloudflare | Cloudflare | Contract Ready | - |
| AWS S3 | AWS | Ready | - |

### Aprobaciones Requeridas
| Aprobación | Approver | Status |
|------------|----------|--------|
| PRD Approval | Tony Stark | Pending |
| Security Review | Happy Hogan | Pending |
| Budget Approval | Pepper Potts | Pending |

### Timeline
- **Kickoff**: 1 Mar 2024
- **PRD Approval**: 10 Mar 2024
- **Development Start**: 15 Mar 2024
- **Release**: 1 May 2024
```

67.5 Integración PRD → SDD

```
                    INTEGRACIÓN: PRD → SDD

  PRD (Business)              SDD (Technical)


  Intent              Proposal (intent heredado)
  Scope              Specs (scope refinado)
  Success Metrics        Acceptance Criteria (métricas convertidas)
  Personas            User Stories (personas a user stories)
  Approach            Design (arquitectura basada en approach)
  Risks              Technical Tasks (riesgos a tareas)
```

El flujo completo:

```
Business Problem
```

```
      PRD        ← Defines QUÉ (What)

                                        SDD

    PROPOSE           SPECS           DESIGN
   (Intent)         (Requirements)    (Technical)


     TASKS            APPLY           VERIFY
  (Breakdown)       (Implement)       (Validate)

                                                          (END)

    ARCHIVE
   (Engram)
```

67.6 Template PRD

67.6.1 Archivo: `prd-template.md`

```
---
title: "[Nombre del Proyecto]"
status: "Draft"
author: "[Autor]"
date: "[Fecha]"
---

# PRD: [Nombre del Proyecto]

## 1. Intent & Problem Statement

### Problema Actual
[Descripción del problema]

### Impacto del Problema
[Quién se ve afectado y cómo]
```

```
### Propuesta de Valor
[Por qué esto vale la pena]

---

## 2. Personas & Stakeholders

### Usuario Primario
| Campo | Descripción |
|-------|-------------|
| **Nombre** | |
| **Rol** | |
| **Objetivos** | |
| **Frustraciones** | |
| **Contexto** | |

### Stakeholders Clave
| Stakeholder | Interés | Influencia |
|-------------|---------|------------|

---

## 3. Success Metrics (KPIs)

### Métricas de Negocio
| Métrica | Target | Cómo se mide |
|---------|--------|--------------|

### Métricas Técnicas
| Métrica | Target | Cómo se mide |
|---------|--------|--------------|

### Definition of Done
- [ ]

---

## 4. Scope

### In Scope
- [ ]

### Out of Scope
- [ ]

### Nice to Have
- [ ]
```

```
### Dependencias Externas
- [ ]

---

## 5. High-Level Approach

### Arquitectura Propuesta
```

[Diagrama]

```
### Stack Tecnológico
-

### Decisiones Clave
| Decisión | Justificación |
|----------|---------------|

---

## 6. Acceptance Criteria

| ID | Given | When | Then |
|----|-------|------|------|

---

## 7. Risks & Constraints

### Riesgos Identificados
| Riesgo | Severidad | Probabilidad | Mitigación |
|--------|-----------|--------------|------------|

### Restricciones
| Restricción | Descripción | Impacto |
|-------------|-------------|---------|

### Supuestos
- [ ]

---

## 8. Dependencies & Approvals

### Dependencias
| Dependencia | Status | Deadline |
```

```
### Aprobaciones Requeridas
| Aprobación | Approver | Status |
|------------|----------|--------|

### Timeline
- **Kickoff**:
- **PRD Approval**:
- **Development Start**:
- **Release**:
```

67.7 Recursos

- PRD Template de Product Plan
- How to Write a PRD - Atlassian
- SDD Methodology - Gentle AI
- PRD Examples - Silicon Valley Product Group

67.8 Ejercicio: Boss Fight - PRD Architect

Objetivo: Crear un PRD completo para el siguiente problema:

> *"Los trajes Iron Man en campo no pueden comunicarse entre sí cuando están en modo sigilo. Necesitamos un sistema de comunicación segura que funcione sin emitir señales detectable."*

Tu misión: 1. Crear PRD completo usando el template 2. Incluir al menos 3 personas 3. Definir 5 métricas de éxito (SMART) 4. Identificar 3 riesgos con mitigación 5. Crear 5 criterios de aceptación

Entregable: `prd-comunicacion-segura.md`

Checklist de Evaluación: - [ ] Problem Statement claro (< 100 palabras) - [ ] 3+ stakeholders definidos - [ ] 5 KPIs con targets - [ ] Scope IN/OUT claro - [ ] Arquitectura de alto nivel - [ ] 5+ acceptance criteria - [ ] 3+ riesgos con mitigación - [ ] Timeline realista

68 SDD - Spec-Driven Development

69 SDD

69.1 Spec-Driven Development

69.2 Descripción

SDD es una metodología donde el desarrollo se guía por especificaciones claras y verificables. Cada cambio sigue un ciclo de 7 pasos que garantiza calidad y trazabilidad.

69.3 El Ciclo SDD

```
          1. ANALYZE
          SDD + Specs



          2. PROPOSE
          Architecture



          3. REFINEMENT
          User Feedback




  4.         4.           4.
 IMPLE-     TESTING     MEMORY
MENT
```

```
6. CONSOLIDATE
Engram + Merge



7. ITERATE
Next Feature
```

69.4 Documentos del Proceso

69.4.1 1. Proposal (Propuesta)

```
# Propuesta: Sistema de Autenticación

## Intent
Implementar autenticación JWT para la API

## Scope
- Login/Logout
- Refresh tokens
- Password reset
- Rate limiting

## Enfoque
- JWT con expiración corta
- Refresh tokens en HTTP-only cookies
- Redis para blacklist de tokens
```

69.4.2 2. Spec (Especificación)

```
# Especificación: Auth System

## Requisitos

### REQ-001: Login
**Como** usuario
**Quiero** iniciar sesión con email/password
**Para** acceder a mi cuenta

**Criterios de Aceptación:**
- [ ] Recibir JWT access token (15 min)
- [ ] Recibir refresh token (7 días)
```

```
- [ ] Rate limit: 5 intentos/5 min
- [ ] Respuesta 401 si credenciales inválidas
```

69.4.3 3. Design (Diseño)

```
# Diseño: Auth System

## Arquitectura
```

Controller → Service → Repository ↓ ↓ ↓ HTTP JWT Logic Database

```
## Decisiones
1. JWT sobre Sesiones → Stateless, escalable
2. Redis para blacklist → Revocación inmediata
3. Refresh tokens → Mejor UX sin seguridad comprometida
```

69.4.4 4. Tasks (Tareas)

```
# Tareas: Auth System

- [ ] **T-001**: Crear AuthController (2h)
- [ ] **T-002**: Implementar AuthService (3h)
- [ ] **T-003**: Configurar Redis (1h)
- [ ] **T-004**: Escribir tests (2h)
- [ ] **T-005**: Documentar API (1h)

**Total estimado**: 9 horas
```

69.5 Beneficios

Beneficio	Descripción
Claridad	Todos entienden qué construir
Trazabilidad	Cada línea de código tiene una razón
Calidad	Tests verifican specs, no implementación
Feedback temprano	Problemas se detectan antes de codificar

69.6 Recursos

- Metodología SDD Completa
- Templates de Documentos
- Ejemplos Prácticos

70 Engram - Memoria Persistente

71 Engram

71.1 Memoria Persistente para IA

71.2 Descripción

Engram es un sistema de memoria que permite a los agentes de IA recordar decisiones, descubrimientos y patrones entre sesiones.

71.3 Comandos Principales

71.3.1 Guardar Memoria

```
// Guardar decisión importante
mem_save({
  title: "Elegí Zustand sobre Redux",
  type: "decision",
  content: {
    "What": "Cambiamos de Redux a Zustand para gestión de estado",
    "Why": "Zustand es más simple y tiene mejor performance",
    "Where": "src/store/index.ts",
    "Learned": "Para apps pequeñas, Zustand es mejor opción"
  }
})
```

71.3.2 Buscar Memoria

```
// Buscar decisiones pasadas
mem_search({
  query: "gestión de estado React",
  type: "decision",
  limit: 5
})
```

71.3.3 Contexto Reciente

```
// Ver contexto de sesión reciente
mem_context({
  limit: 20,
  project: "curso-iron-man"
})
```

71.4 Tipos de Observación

Tipo	Uso	Ejemplo
`decision`	Decisiones arquitectónicas	"Elegí PostgreSQL sobre MongoDB"
`bugfix`	Soluciones de bugs	"Fixed N+1 query en UserList"
`pattern`	Patrones establecidos	"Patrón Repository para DB"
`discovery`	Descubrimientos importantes	"API rate limiting era el bottleneck"

71.5 Recursos

- Documentación Engram
- Ejemplos de Uso

72 MCP - Model Context Protocol

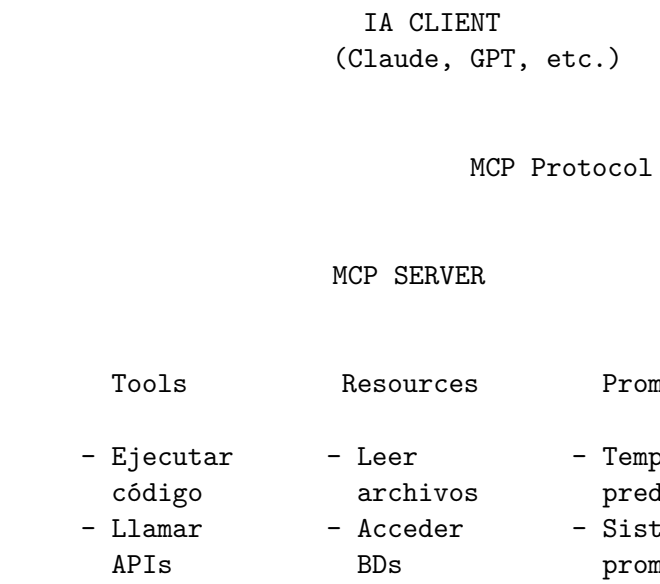
73 MCP

73.1 Model Context Protocol

73.2 Descripción

MCP es un protocolo abierto que estandariza cómo las IAs se conectan a herramientas, APIs y datos. Es como USB para IA: un estándar universal.

73.3 Arquitectura



73.4 Implementación

73.4.1 Servidor MCP Básico

```python
from mcp import Server
from mcp.types import Tool, TextContent

server = Server("mi-servidor")

@server.tool("calcular")
async def calcular(expression: str) -> list[TextContent]:
    """Calcula una expresión matemática"""
    try:
        result = eval(expression)  # ¡Cuidado con seguridad!
        return [TextContent(text=f"Resultado: {result}")]
    except Exception as e:
        return [TextContent(text=f"Error: {str(e)}")]

if __name__ == "__main__":
    server.run()
```

73.4.2 Cliente MCP

```python
from mcp import Client

async def main():
    async with Client("http://localhost:8000") as client:
        # Listar herramientas disponibles
        tools = await client.list_tools()

        # Ejecutar una herramienta
        result = await client.call_tool("calcular", {"expression": "2 + 2"})
        print(result)
```

73.5 Seguridad

Riesgo	Mitigación
Ejecución de código	Sandbox + whitelist
Acceso a archivos	Permisos granulares + chroot
Inyección de prompts	Validación estricta de inputs
Exfiltración de datos	Logging + rate limiting

73.6 Recursos

- Especificación MCP
- Lista de Servidores MCP
- Tutorial Completo

74 Skills - Habilidades Especializadas

75 Skills

75.1 Habilidades Especializadas para IA

75.2 Descripción

Los Skills son módulos de instrucciones especializadas que se cargan automáticamente según el contexto. Son como "modos de trabajo" para agentes de IA.

75.3 Estructura de un Skill

```
# SKILL.md
name: "typescript"
description: "TypeScript strict patterns and best practices"
trigger: "When writing TypeScript code"

instructions: |
  ## TypeScript Strict Mode

  ### Reglas Obligatorias
  1. Usar `strict: true` en tsconfig.json
  2. Evitar `any` - usar `unknown` si es necesario
  3. Type hints en todas las funciones
  4. Interfaces para objetos, types para uniones

  ### Ejemplo
  ```typescript
  //  Correcto
  function greet(name: string): string {
    return `Hello, ${name}!`;
  }

  //  Incorrecto
  function greet(name) {
    return "Hello, " + name;
  }
```
```

```
##  **Carga Automática de Skills**

### **Smart Dispatcher**
```python
class SkillDispatcher:
    """Detecta contexto y carga skills apropiadas"""

    SKILLS = {
        "react": ["react-19", "typescript", "zustand-5"],
        "angular": ["angular-core", "angular-forms", "typescript"],
        "python-api": ["fastapi", "pytest", "security"],
        "devops": ["docker", "kubernetes", "terraform"],
    }

    def detect_stack(self, project_files: List[str]) -> List[str]:
        """Detecta stack tecnológico y retorna skills"""
        detected = []

        if "package.json" in project_files:
            if self._has_dependency("react", project_files):
                detected.extend(self.SKILLS["react"])

        if "requirements.txt" in project_files:
            if self._has_dependency("fastapi", project_files):
                detected.extend(self.SKILLS["python-api"])

        return list(set(detected))  # Sin duplicados
```

75.4 Skills Disponibles

75.4.1 Desarrollo Frontend

Skill	Trigger	Contenido
`react-19`	Componentes React	Patrones React 19, hooks, server components
`angular-core`	Componentes Angular	Standalone, signals, control flow
`typescript`	Código TypeScript	Strict mode, generics, type guards
`tailwind-4`	Estilos Tailwind	Patrones Tailwind 4, dark mode

75.4.2 Desarrollo Backend

Skill	Trigger	Contenido
`fastapi`	APIs FastAPI	Pydantic, dependencies, async
```

Skill	Trigger	Contenido
`django-drf`	APIs Django	ViewSets, serializers, filters
`spring-boot-3`	Apps Spring	Configuration, DI, web services

75.4.3 DevOps & Security

Skill	Trigger	Contenido
`docker`	Contenedores	Dockerfile, compose, multi-stage
`security`	Auth/Validación	OWASP, input validation, JWT
`devsecops`	CI/CD seguro	SAST/DAST, dependency scanning

75.4.4 IA & Agentes

Skill	Trigger	Contenido
`sdd-*`	Spec-Driven Dev	Proposal → Spec → Design → Tasks
`engram`	Memoria persistente	Save/search/context commands
`mcp`	Herramientas IA	Protocolo de comunicación

75.5 Recursos

- Directorio de Skills
- Crear tu Propio Skill
- Skills Registry

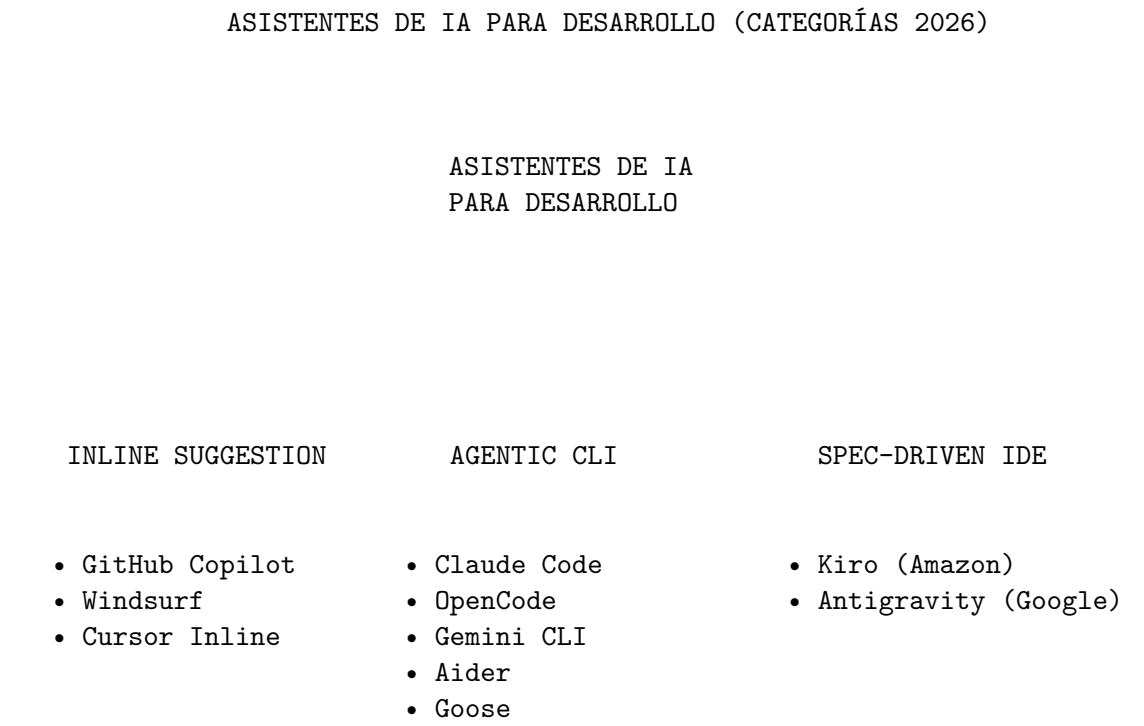
76 Herramientas de Asistentes de IA para Desarrollo (2026)

77 Herramientas de Asistentes de IA para Desarrollo

77.1 El Arsenal de Tony Stark en 2026

77.2 Visión General

En 2026, los asistentes de IA para desarrollo se han dividido en **tres categorías principales**:



77.3 Tabla Comparativa Principal

Característica	**OpenCode**	**KiloCode**	**Claude Code**	**Gemini CLI**	**Kiro**	**Amp**
Tipo	Terminal Agent	IDE Extension + CLI	Terminal Agent	Terminal Agent	Spec-Driven IDE	Terminal Agent
Open Source	MIT	Apache 2.0				
Precio	Gratis (BYO API)	Gratis + Kilo Pass $19/mo	$20-200/mo	Gratis (Gemini 3)	$20-200/mo	Gratis (BYO API)
Modelos	75+ proveedores	500+ modelos	Solo Anthropic	Solo Google	AWS Models	Múltiples
Context Window	Segun modelo	Según modelo	1M tokens (Opus)	1M tokens	Full project	Según modelo
GitHub Stars	124K+	16K+	N/A	N/A	N/A	Menos conocido
MCP Support						
AGENTS.md					(Specs)	
Orquestación		Orchestrator Mode	Subagents		Hooks	
Soporte IDE	Terminal + VS Code	VS Code, JetBrains	VS Code, JetBrains	Terminal	Kiro IDE	Terminal
Mejor para	Flexibilidad, Privacidad	Multi-modelo, Open Source	Debugging complejo	Coste cero, Google ecosystem	SDD, Especificaciones	Tareas rápidas

---

77.4 Análisis Detallado por Herramienta

77.4.1 1. OpenCode — El Caballero Oscuro

77.4.1.1 ¿Qué es?

OpenCode es un **agente de terminal open-source** construido en TypeScript/Bun con TUI en Go. Es **agnostic al proveedor** — funciona con 75+ proveedores de LLM.

77.4.1.2 Características Principales

```
# Instalación
curl -fsSL https://opencode.ai/install | bash
# o
brew install anomalyco/tap/opencode

# Uso básico
opencode
```

77.4.1.3 Ventajas

- **Gratuito** (BYO API key)
- **75+ proveedores**: OpenAI, Anthropic, Google, AWS Bedrock, Groq, Ollama
- **Open source** (MIT)
- **TUI interactivo**
- **Multi-sesión**
- **LSP support**

77.4.1.4 Desventajas

- Sin orquestación nativa
- Más complejo de configurar
- Comunidad más pequeña que Copilot

77.4.1.5 Configuración para Iron Man Evolution

```
# .opencode/config.yaml
models:
  default: "anthropic/claude-3-opus-20240229"
  fallback: "openai/gpt-4-turbo"

providers:
  anthropic:
    api_key: ${ANTHROPIC_API_KEY}
  openai:
    api_key: ${OPENAI_API_KEY}
  google:
    api_key: ${GOOGLE_API_KEY}

context:
  max_tokens: 100000
  include_patterns:
    - "src/**/*.py"
```

```
    - "tests/**/*.py"
  exclude_patterns:
    - "node_modules/**"
    - ".git/**"
```

77.4.2 2. KiloCode — El Arsenal Completo

77.4.2.1 ¿Qué es?

KiloCode es una **extensión de VS Code/JetBrains** fork de Cline, con **Orchestrator Mode** y soporte para **500+ modelos** sin markup.

77.4.2.2 Características Principales

```
# Instalación (VS Code)
code --install-extension kilo-code.kilo-code

# Instalación CLI
npm install -g @kilo-code/cli

# Uso
kilo-code
```

77.4.2.3 Ventajas

- **500+ modelos** a precio exacto del proveedor
- **Orchestrator Mode** para multi-agente
- **Memory Bank** para memoria persistente
- **Soporte JetBrains**
- **App Builder** integrado
- **Auto-completado** integrado

77.4.2.4 Desventajas

- reports de issues con context bloat
- Menos maduro que Cline original
- Comunidad más pequeña

77.4.2.5 Configuración para Iron Man Evolution

```
{
  "kilo-code": {
    "model": "anthropic/claude-3-opus-20240229",
    "mode": "orchestrator",
    "maxTokens": 100000,
    "agents": {
      "testing": {
        "model": "anthropic/claude-3-sonnet-20240229",
        "focus": "testing"
      },
      "security": {
        "model": "openai/gpt-4-turbo",
        "focus": "security audit"
      }
    }
  }
}
```

77.4.3 3. Claude Code — El J.A.R.V.I.S. Nativo

77.4.3.1 ¿Qué es?

Claude Code es el **agente terminal nativo de Anthropic**. No es un IDE — es un agente que vive en tu terminal y puede leer archivos, ejecutar comandos, y iterar.

77.4.3.2 Características Principales

```
# Instalación
npm install -g @anthropic-ai/claude-code

# Uso
claude

# Con AGENTS.md
claude --agents AGENTS.md
```

77.4.3.3 Ventajas

- **1M tokens** de contexto (Opus 4.6)
- **Razonamiento profundo** — mejor para debugging complejo
- **Subagentes** nativos
- **Background agents**
- **Integración MCP** profunda
- **Agent teams** en Opus 4.6

77.4.3.4 Desventajas

- **Solo modelos Anthropic** — lock-in
- **Precio**: $20-200/mo según plan
- No open source

77.4.3.5 Configuración para Iron Man Evolution

```
# Iniciar en tu proyecto
cd ~/iron-man-project
claude

# Con configuración específica
claude --config config/claude-config.json

# Usar AGENTS.md automáticamente
claude --agents AGENTS.md
```

77.4.4 4. Gemini CLI — El Asistente Gratis de Google

77.4.4.1 ¿Qué es?

Gemini CLI es el **agente terminal de Google** que ofrece acceso **gratis** a Gemini 3 con **1M tokens** de contexto.

77.4.4.2 Características Principales

```
# Instalación
npm install -g @google/gemini-cli

# Uso (gratis con cuenta Google)
gemini
```

```
# Con API key
gemini --api-key $GOOGLE_API_KEY
```

77.4.4.3 Ventajas

- **Gratis** con cuenta Google
- **1M tokens** de contexto
- **Cero configuración** en máquina nueva
- Integración con Google Cloud
- **Gemini 2.5 Pro** disponible

77.4.4.4 Desventajas

- Solo modelos Google
- Menos capacidades agénticas que Claude Code
- Privacidad: datos a Google

77.4.4.5 Configuración para Iron Man Evolution

```
# ~/.gemini/config.yaml
model: "gemini-2.5-pro"
temperature: 0.7
max_tokens: 1000000

context:
  project_root: "~/iron-man-project"
  include:
    - "src/**"
    - "tests/**"
  exclude:
    - "node_modules/**"
    - ".git/**"
```

77.4.5 5. Kiro (Amazon) — El Revolucionario SDD

77.4.5.1 ¿Qué es?

Kiro es el **IDE de Amazon** que revoluciona el desarrollo con **Spec-Driven Development**. No es solo un asistente — es un **nuevo paradigma** de desarrollo.

77.4.5.2 Características Principales

```
# Instalación
# Descargar desde https://kiro.aws
# Es un IDE basado en VS Code

# Uso
# Abre Kiro y describe lo que quieres construir
```

77.4.5.3 Ventajas

- **Spec-Driven Development** nativo
- **Hooks** (automatización basada en eventos)
- **Full project context**
- **Structured output** de alta calidad
- Integración AWS profunda
- **Steerling** — diseño de UI con IA

77.4.5.4 Desventajas

- **Nuevo** (julio 2025) — menos maduro
- **Precio**: $20-200/mo
- **Lock-in** a AWS ecosystem
- **Paradigma diferente** — curva de aprendizaje

77.4.5.5 Configuración para Iron Man Evolution

```yaml
# .kiro/specs/iron-man-evolution/spec.yaml
name: "Iron Man Evolution"
description: "Curso de desarrollo con IA"

requirements:
  - id: FR-001
    title: "Sistema de niveles"
    description: "6 niveles progresivos inspirados en Iron Man"

  - id: FR-002
    title: "Gentle AI Stack"
    description: "Integración completa con Engram, MCP, Skills"

hooks:
  - trigger: "on_save"
    action: "run_tests"
```

```
  - trigger: "on_commit"
    action: "validate_specs"
```

77.4.6 6. Amp — El Asistente Ligero

77.4.6.1 ¿Qué es?

Amp es un **agente de terminal open-source** más ligero, enfocado en **tareas rápidas** sin la complejidad de herramientas más grandes.

77.4.6.2 Características Principales

```
# Instalación
npm install -g @anthropic-ai/amp

# Uso
amp "Create a Python function to calculate energy"
```

77.4.6.3 Ventajas

- **Ligero** y rápido
- **Open source**
- **Sin configuración** compleja
- **Tareas rápidas**
- **Múltiples modelos**

77.4.6.4 Desventajas

- Menos features que Claude Code
- Sin orquestación
- Comunidad pequeña

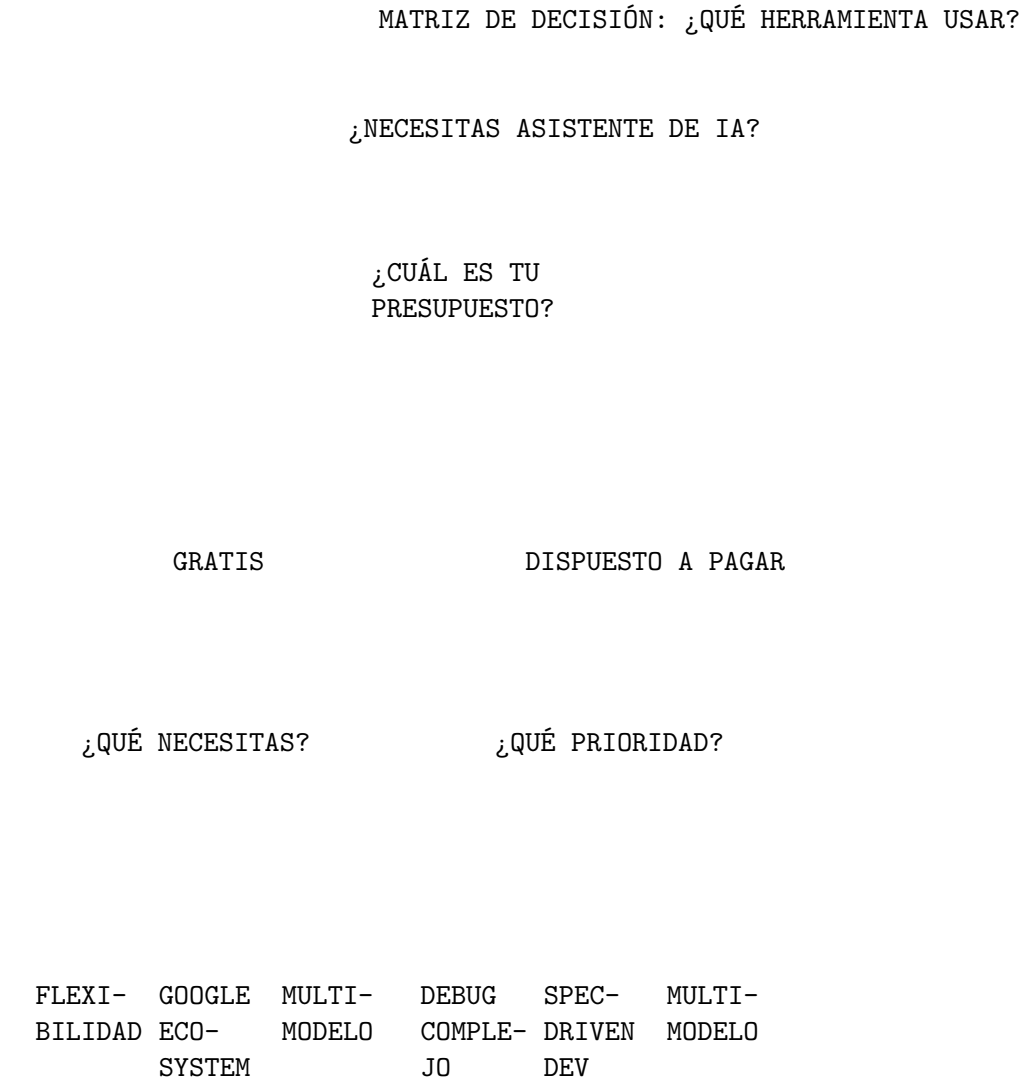
77.4.6.5 Configuración para Iron Man Evolution

```
# Configuración básica
amp configure

# Uso con AGENTS.md
amp --agents AGENTS.md "Write tests for reactor.py"
```

77.5 Guía de Selección: ¿Qué Herramienta Usar?

77.5.1 Matriz de Decisión



77.5.2 Escenarios de Uso

Escenario	Herramienta Recomendada	Por Qué
Presupuesto cero	Gemini CLI	Gratis, 1M contexto
Flexibilidad total	OpenCode	75+ proveedores, open source
Debugging complejo	Claude Code	Mejor razonamiento, 1M contexto
Spec-Driven Development	Kiro	Paradigma nativo SDD
Multi-agente	KiloCode	Orchestrator Mode
Tareas rápidas	Amp	Ligero, rápido
Enterprise	GitHub Copilot	Integración GitHub, governance
Open source lover	OpenCode	MIT, comunidad grande

77.5.3 Stack Recomendado por Nivel

77.5.3.1 Principiante (Niveles 1-2)

```yaml
herramientas:
  principal: "Gemini CLI"
  razon: "Gratis, fácil de empezar"
  alternativa: "OpenCode"

configuracion_basica:
  - Instalar una herramienta
  - Configurar AGENTS.md básico
  - Aprender prompting efectivo
```

77.5.3.2 Intermedio (Niveles 3-4)

```yaml
herramientas:
  principal: "Claude Code"
  razon: "Mejor debugging, subagentes"
  alternativa: "OpenCode con múltiples modelos"

configuracion:
  - AGENTS.md avanzado
  - Skills registry
```

```
  - Engram para memoria
  - MCP básico
```

77.5.3.3 Avanzado (Niveles 5-6)

```
herramientas:
  principal: "Kiro + Claude Code"
  razon: "SDD con Kiro, debugging con Claude"
  alternativa: "KiloCode para orquestación"

stack_completo:
  - Gentle AI Stack
  - Engram + MCP + Skills
  - CI/CD pipeline
  - Múltiples agentes
```

77.6 Configuración Multi-Herramienta

77.6.1 Usar Múltiples Herramientas en el Mismo Proyecto

```
# Estructura de proyecto multi-herramienta
iron-man-project/
   .opencode/            # Config OpenCode
      config.yaml
   .claude/              # Config Claude Code
      config.json
   .kiro/                # Config Kiro
      specs/
   .kilo/                # Config KiloCode
      settings.json
   AGENTS.md             # Configuración común
   context/              # Contexto compartido
   skills/               # Skills compartidos
   src/                  # Código fuente
```

77.6.2 AGENTS.md Multi-Herramienta

```
# AGENTS.md - Configuración Común para Múltiples Herramientas

## Herramientas Configuradas
Este proyecto soporta múltiples asistentes de IA:

### 1. OpenCode
```yaml
# .opencode/config.yaml
models:
  default: "anthropic/claude-3-opus-20240229"
```

77.6.3 2. Claude Code

```
// .claude/config.json
{
  "model": "claude-3-opus-20240229",
  "agents_md": "AGENTS.md"
}
```

77.6.4 3. KiloCode

```
// .kilo/settings.json
{
  "model": "anthropic/claude-3-opus-20240229",
  "mode": "orchestrator"
}
```

77.7 Reglas Comunes (aplican a TODAS las herramientas)

- Usa snake_case para Python
- Sigue PEP 8
- Incluye tests para cada función
- Documenta en español

```
### **Script de Configuración Unificada**

```bash
#!/bin/bash
# setup-ai-tools.sh
# Configura múltiples herramientas de IA para el proyecto
```

```
echo "  Configurando herramientas de IA para Iron Man Evolution..."

# Verificar herramientas instaladas
TOOLS=("opencode" "claude" "kilo-code" "gemini" "kiro" "amp")

for tool in "${TOOLS[@]}"; do
    if command -v $tool &> /dev/null; then
        echo "  $tool está instalado"
    else
        echo "  $tool no está instalado"
        echo "   Instalar: "
        case $tool in
            "opencode")
                echo "   curl -fsSL https://opencode.ai/install | bash"
                ;;
            "claude")
                echo "   npm install -g @anthropic-ai/claude-code"
                ;;
            "kilo-code")
                echo "   code --install-extension kilo-code.kilo-code"
                ;;
            "gemini")
                echo "   npm install -g @google/gemini-cli"
                ;;
            "kiro")
                echo "   Descargar desde https://kiro.aws"
                ;;
            "amp")
                echo "   npm install -g @anthropic-ai/amp"
                ;;
        esac
    fi
done

# Crear archivos de configuración comunes
echo "  Creando estructura de configuración..."
mkdir -p .opencode .claude .kilo .gemini

# Copiar AGENTS.md a cada herramienta
if [ -f "AGENTS.md" ]; then
    echo "  AGENTS.md encontrado - será usado por todas las herramientas"
else
    echo "   No se encontró AGENTS.md - crear uno primero"
fi

echo ""
echo "  Configuración completada!"
echo "  Usa 'opencode', 'claude', o 'gemini' en tu proyecto"
```

77.8 Benchmark: Mismo Modelo, Diferentes Herramientas

77.8.1 Prueba: Kimi K2.5 en OpenCode vs KiloCode

De acuerdo con análisis recientes, **el mismo modelo puede rendir diferente** según el harness:

Tarea	OpenCode	KiloCode
Refactorización compleja	Exitoso	Falló tool calls
Debugging multi-archivo	Limpio	Context bloating
Generación de tests	Confiable	Algunas alucinaciones
Documentación	Bueno	Aceptable

77.8.2 Lección Aprendida

> **"El harness importa tanto como el modelo. Un mal harness puede desperdiciar el potencial de Claude Opus 4.5."**

77.9 Seguridad Multi-Herramienta

77.9.1 Proteger API Keys

```
# .env (NO committear)
ANTHROPIC_API_KEY=sk-ant-xxx
OPENAI_API_KEY=sk-xxx
GOOGLE_API_KEY=AIza-xxx
GITHUB_TOKEN=ghp_xxx

# .gitignore
.env
.opencode/secrets.yaml
.claude/secrets.json
.kilo/secrets.json
```

77.9.2 Validación de Herramientas

```python
# security/validate_tools.py
"""
Valida que las herramientas de IA estén configuradas correctamente.
"""

import os
import subprocess
from typing import Dict, List

class ToolValidator:
    """Validador de herramientas de IA."""

    REQUIRED_TOOLS = ["opencode", "claude", "gemini"]
    OPTIONAL_TOOLS = ["kilo-code", "kiro", "amp"]

    def __init__(self):
        self.results = {}

    def validate_all(self) -> Dict:
        """Valida todas las herramientas."""
        results = {
            "required": {},
            "optional": {},
            "env_vars": {},
            "configs": {}
        }

        # Verificar herramientas requeridas
        for tool in self.REQUIRED_TOOLS:
            results["required"][tool] = self.check_tool_installed(tool)

        # Verificar herramientas opcionales
        for tool in self.OPTIONAL_TOOLS:
            results["optional"][tool] = self.check_tool_installed(tool)

        # Verificar variables de entorno
        env_vars = [
            "ANTHROPIC_API_KEY",
            "OPENAI_API_KEY",
            "GOOGLE_API_KEY",
            "GITHUB_TOKEN"
        ]

        for var in env_vars:
            results["env_vars"][var] = bool(os.getenv(var))

        # Verificar archivos de configuración
```

```
        config_files = [
            ".opencode/config.yaml",
            ".claude/config.json",
            ".kilo/settings.json",
            "AGENTS.md"
        ]

        for config in config_files:
            results["configs"][config] = os.path.exists(config)

        return results

    def check_tool_installed(self, tool: str) -> bool:
        """Verifica si una herramienta está instalada."""
        try:
            subprocess.run(
                [tool, "--version"],
                capture_output=True,
                timeout=5
            )
            return True
        except:
            return False

    def print_report(self):
        """Imprime reporte de validación."""
        results = self.validate_all()

        print("\n Reporte de Herramientas de IA")
        print("=" * 50)

        print("\n Herramientas Requeridas:")
        for tool, installed in results["required"].items():
            status = " " if installed else " "
            print(f"  {status} {tool}")

        print("\n Herramientas Opcionales:")
        for tool, installed in results["optional"].items():
            status = " " if installed else " "
            print(f"  {status} {tool}")

        print("\n Variables de Entorno:")
        for var, set_ in results["env_vars"].items():
            status = " " if set_ else " "
            print(f"  {status} {var}")

        print("\n Archivos de Configuración:")
        for config, exists in results["configs"].items():
```

```
            status = "✓" if exists else "✗"
            print(f"  {status} {config}")

# Uso
if __name__ == "__main__":
    validator = ToolValidator()
    validator.print_report()
```

77.10 Recursos por Herramienta

77.10.1 OpenCode

- Sitio Oficial
- GitHub
- Documentación
- Discord

77.10.2 KiloCode

- Sitio Oficial
- GitHub
- Documentación
- Discord

77.10.3 Claude Code

- Sitio Oficial
- npm
- Documentación
- Tutorial

77.10.4 Gemini CLI

- Sitio Oficial
- npm
- Documentación
- Gratis con Google Account

77.10.5 Kiro

- Sitio Oficial
- Descargar
- Documentación
- Videos

77.10.6 Amp

- npm
- Documentación
- Gratis

77.11 Integración con Iron Man Evolution

77.11.1 Cómo Encajan las Herramientas en los Niveles

Nivel Iron Man	Herramientas Recomendadas	Uso
1. Demo en la Cueva	Gemini CLI (gratis)	Primeros prompts, scripts básicos
2. Mark I	OpenCode (flexible)	AGENTS.md, contexto
3. Mark III	Claude Code (potente)	Agentes, debug-ging
4. JARVIS Avanzado	KiloCode (multi-agente)	Orquestación, Engram, MCP
5. Ultron	Kiro (SDD)	Especifi-caciones, controles
6. Nanotech	Stack completo	Todas in-tegradas

77.11.2 Comando de Instalación Unificada

```
# Instalar todo el stack de Iron Man Evolution
curl -fsSL https://raw.githubusercontent.com/Gentleman-Programming/gentle-ai/main/scripts
```

77.12 Próximos Pasos

1. **Instala** la herramienta que mejor se adapte a tu nivel
2. **Configura** AGENTS.md según el tutorial
3. **Practica** con los ejercicios de cada nivel
4. **Experimenta** con múltiples herramientas
5. **Contribuye** a las comunidades open source

"No se trata de tener la mejor herramienta. Se trata de tener las herramientas correctas para cada situación." — Tony Stark (probablemente)

78 Gentle AI Stack

79 Gentle AI Stack

79.1 Referencia Completa

79.2 Descripción

El Gentle AI Stack es una colección de herramientas y patrones para desarrollar con IA de forma robusta y mantenible.

79.3 Componentes Principales

79.3.1 1. Engram (Memoria Persistente)

- Sistema de memoria que sobrevive entre sesiones
- Guarda decisiones, bugs, y patrones
- Búsqueda semántica de contexto pasado

79.3.2 2. MCP (Model Context Protocol)

- Protocolo para comunicación IA-herramientas
- Estandariza acceso a APIs, bases de datos, archivos
- Seguridad integrada con permisos granulares

79.3.3 3. Skills (Habilidades Especializadas)

- Instrucciones específicas para tareas
- Cadenas automáticas de habilidades
- Registro centralizado de capacidades

79.3.4 4. SDD (Spec-Driven Development)

- Desarrollo guiado por especificaciones
- Ciclo: Proposal → Spec → Design → Tasks → Apply → Verify → Archive
- Garantiza calidad y trazabilidad

79.4 Recursos

- Repositorio Gentle AI Stack
- Documentación Oficial
- Ejemplos de Uso

Part IX

Información

80 Acerca del Autor

80.1 Diego Saavedra García (Statick)

Desarrollador FullStack con más de 15 años de experiencia en la industria del software. Specializado en ciberseguridad, desarrollo con IA y arquitectura de sistemas.

80.2 Especialidades

- **Desarrollo FullStack**: Angular, React, Next.js, Django, FastAPI
- **Ciberseguridad Ofensiva**: Ethical Hacking, Penetration Testing, Bug Bounty
- **Inteligencia Artificial**: Agentes, MCP, Engram, SDD, Prompt Engineering
- **Arquitectura**: Clean Architecture, Hexagonal Architecture, Domain-Driven Design
- **Educación**: Diseño curricular, formación de desarrolladores

---

80.3 Formacion Académica

- **Máster en Ciberseguridad** — Universidad Complutense de Madrid (UCM) — *En curso*
- **Ingeniería en Sistemas** — ESPOL — *Completado*
- **Certificaciones**: OSCP, CEH, CISSP — *Varias*

80.4 Experiencia Docente

Actualmente dicto clases en:

- **ESPE** (Escuela Superior Politécnica del Ejercito) — Ciberseguridad
- **UIDE** (Universidad Internacional de Ecuador) — Desarrollo de Software
- **ABACOM** — Cursos de IA y Desarrollo

80.5 Información de Contacto

Plataforma	Enlace
GitHub	github.com/statick88
LinkedIn	linkedin.com/in/diego-saavedra-developer
Twitter/X	García (2026)
Web Personal	statick88.github.io
YouTube	YouTube Channel

80.6 Publicaciones y Proyectos

80.6.1 Cursos

- Ethical Hacking 2026 — Curso completo conlabs prácticos
- CTF Challenges — Retos de ciberseguridad
- Stark Protocol — Este libro

80.6.2 Proyectos Open Source

- CyberVault — Gestor de credenciales seguro
- Gentle AI Stack — Stack de IA para desarrolladores
- Engram — Memoria persistente para agentes

80.7 Reconocimientos

- **Google Developer Expert** (GDE) — Web Technologies
- **Microsoft MVP** — Developer Technologies
- **Speaker**: Devoxx, JavaConf, JSConf, varios meetups técnicos

---

80.8 Filosofía

> *"No programes como si estuvieras en 1990. No programes como si la IA fuera a reemplazarte. Programá como Tony Stark: pilota, amplifica, y construí sistemas que importan."*

80.9 Licencia del Libro

Ver license.qmd para los términos de uso.

Última actualización: Marzo 2026

81 Licencia

81.1 MIT License

81.2 Sobre el Contenido

El concepto de “Stark Protocol” y la narrativa de Tony Stark/J.A.R.V.I.S. son inspirados en el universo Marvel de Stan Lee y Marvel Comics.

Este curso utiliza estos personajes como marco educativo. Marvel y sus personajes son marcas registradas de Marvel Comics.

El contenido técnico y los patrones de colaboración humano-IA enseñados son originales de Diego Saavedra.

81.3 Uso Educativo

Este material puede ser utilizado con fines educativos sin autorización previa, siempre que se mantenga la atribución al autor original.

Para uso comercial, contactar al autor.

García, Diego Saavedra. 2026. *Perfil de Twitter*. https://twitter.com/statick_ds.

Gentleman Programming. 2026. *Gentle AI Stack: Herramientas de IA Para Desarrollo*. https://github.com/Gentleman-Programming/gentle-ai.