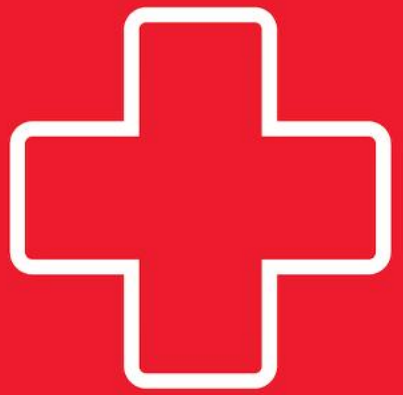
LEGACY CODE



FIRST AID KIT

14 TECHNIQUES TO QUICKLY AND SAFELY RESCUE A CODEBASE

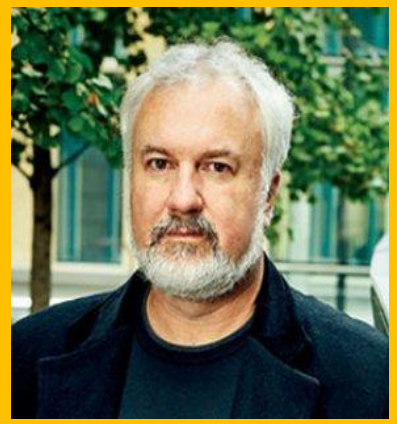
Carone Menes, Leonardo - Fernández Suarez, Ignacio
Fernández García, Alejandro

NICOLAS CARLO

- desarrollador web independiente especializado en la mejora y mantenimiento de bases de código heredado

LEGACY CODE

- Mucha gente lo define como código "viejo"
- Código sin pruebas (tests) siendo algo común encontrar código sin probar - Michael Feathers



DEFINICIÓN NICOLAS CARLO

Código valioso que te da miedo cambiar

Valioso
Si no fuera valioso, simplemente se podría desechar

Miedo a cambiar
Si fuera facilmente modificable entonces no tendria mucho enfoque

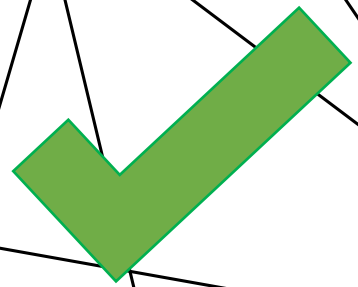
¿CÓMO TRATAMOS CON ESE CÓDIGO?

REESCRIBIR VS REFACTORIZAR

resultado de volver a implementar una gran parte de la funcionalidad existente sin reutilizar su código fuente

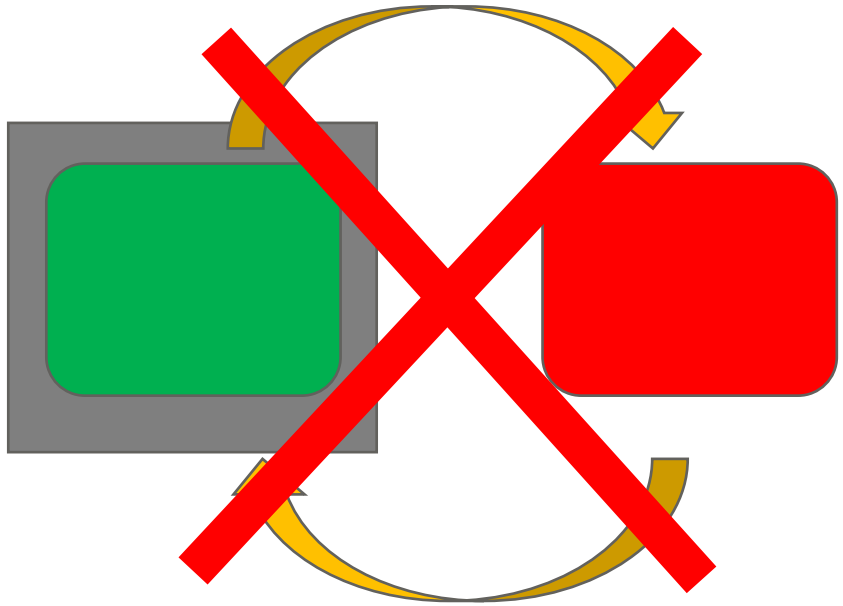
★ Salvo por proyectos similares a otros, pero no son el mismo

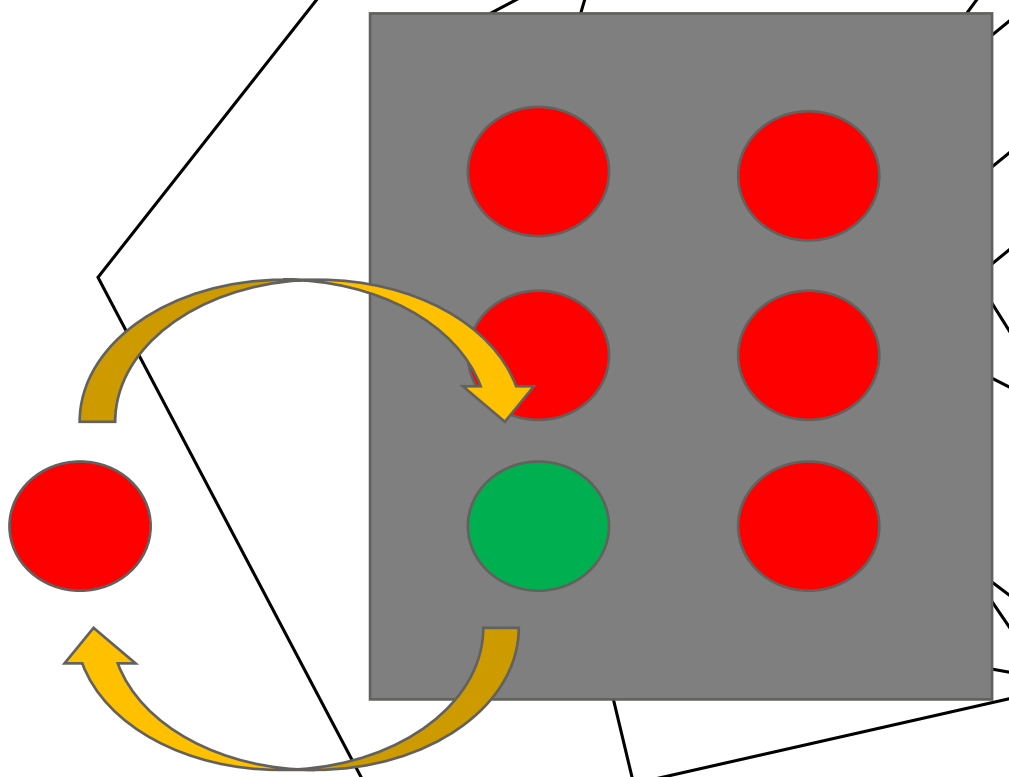
modificar el código fuente de un programa sin cambiar su funcionalidad



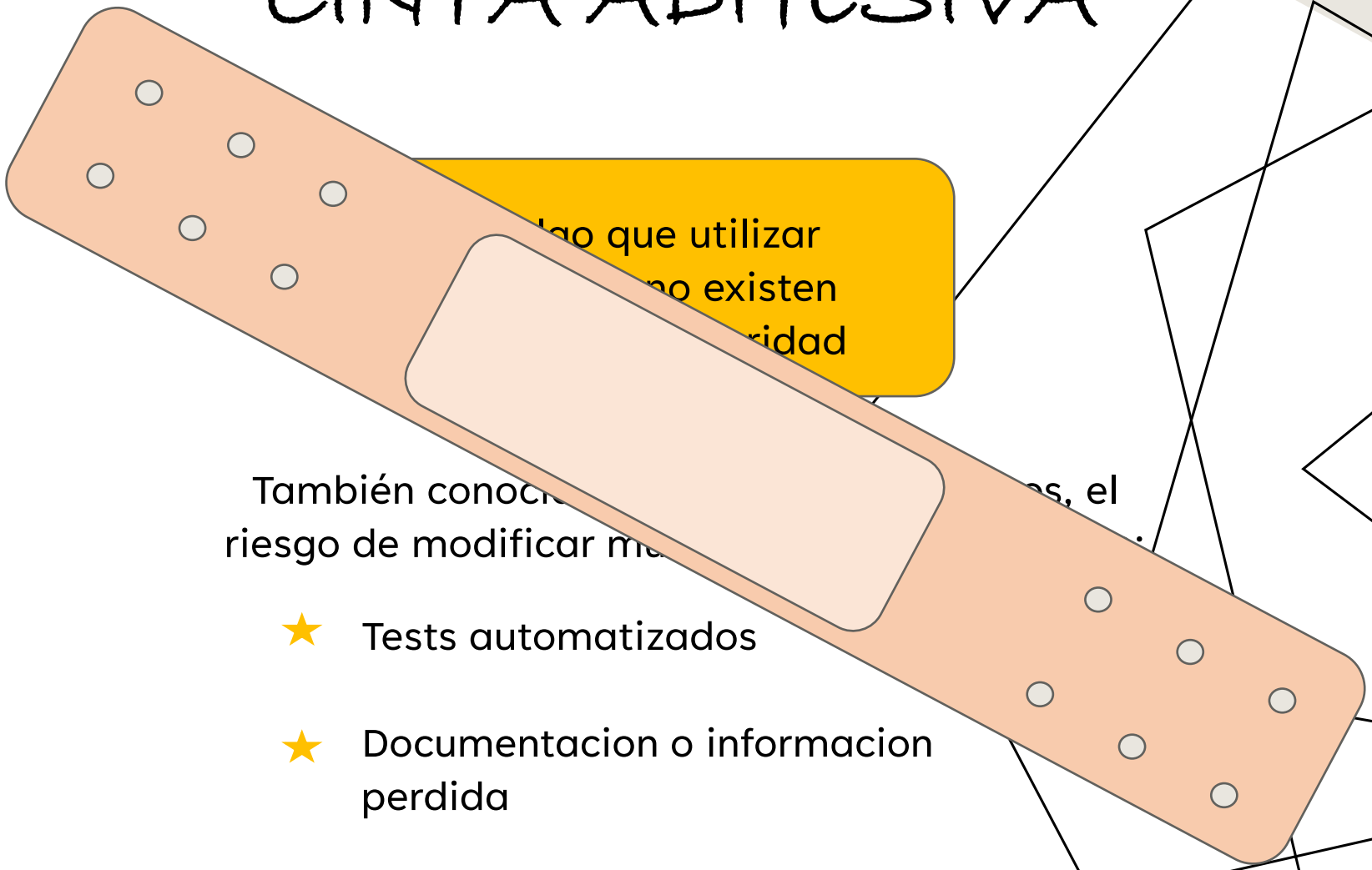
ESTRATEGIA DE REESCRIBIR

La idea principal es modificar partes monolíticas del sistema mientras que la estructura externa se mantiene.





CINTA ADHESIVA



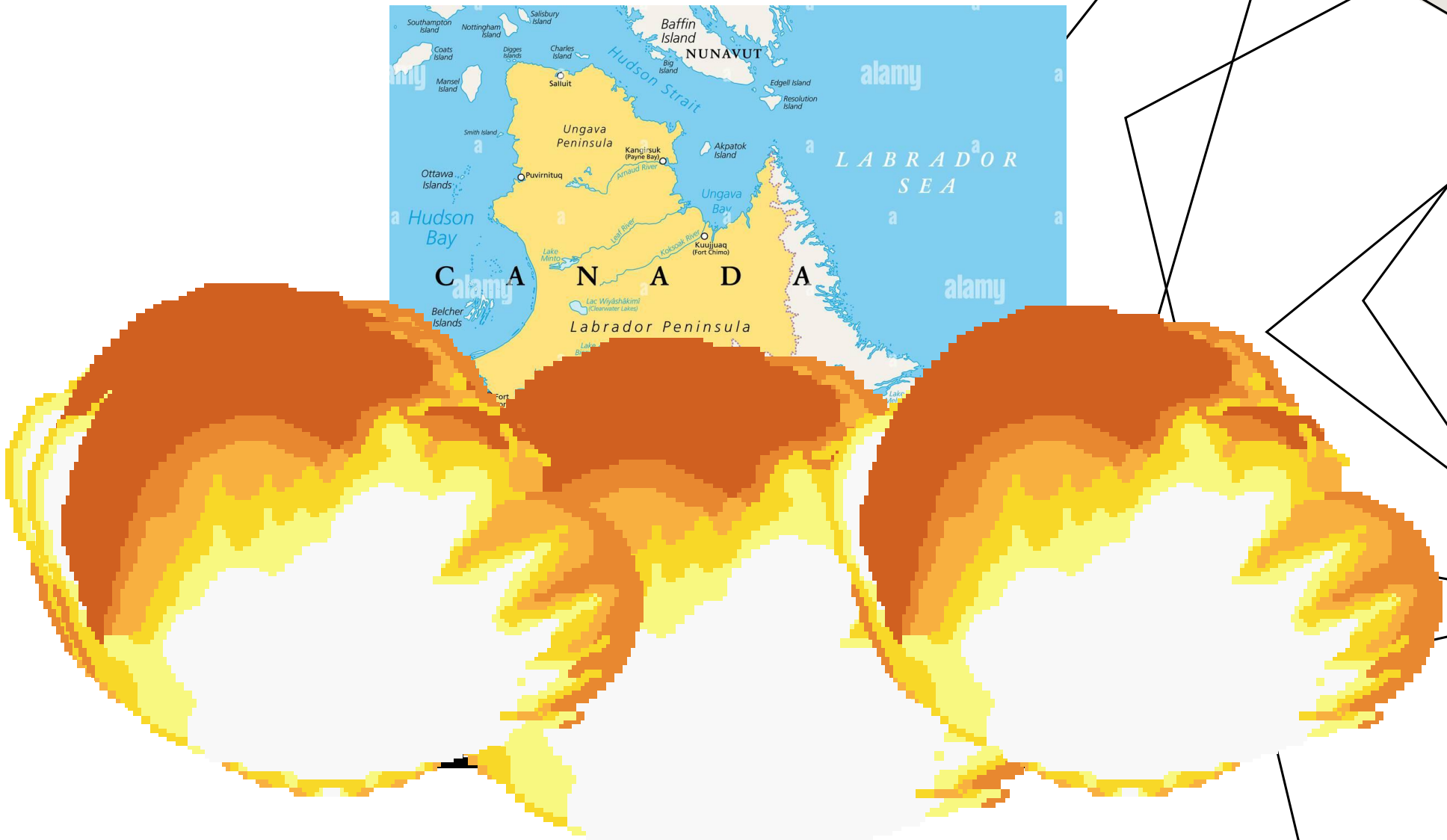
...go que utilizar
...no existen
...ridad

También conoci... ...s, el
riesgo de modificar m...

- Tests automatizados
- Documentacion o informacion perdida
- Monitorizacion / Logs

RIESGOS DE LAS REESCRITURAS

- Riesgo y Subestimación
- Problema de Validación
- Analogía con IA


Baffin Island
NUNAVUT
Hudson Strait
Ungava Peninsula
Hudson Bay
LABRADOR SEA
Ungava Bay
Labrador Peninsula
CANADA
Salluit
Puvirnituq
Kangiqsuk (Payne Bay)
Kuujjuaq (Fort Chimo)
Akpatok Island
Ottawa Islands
Belcher Islands
Resolution Island
Edgell Island
Mansel Island
Coats Island
Charles Island
Big Island
Lake Minto
Lac Wiyâshâkimî (Clearwater Lakes)
alamy

EJEMPLO DE QUEBEC

Intentaron modernizar un sistema heredado (relacionado con vehículos). tomando 3-4 días para el cambio en el cual todo lo relacionado fue cerrado.

Hubo consecuencias

Ocurrió en febrero, causó problemas hasta agosto

Costó millones de dólares adicionales

Daños reputacionales al gobierno

RESISTENCIA A INVERTIR EN REFACTORIZACIÓN

Sin ganancia no tiene importancia – **Un libro**

La gerencia suele oponer resistencia a la hora de refactorizar codigo viejo

Solo quieren nuevas funciones a vender

RESISTENCIA A INVERTIR EN REFACTORIZACIÓN

2 Problemas

Los superiores no tienen el conocimiento tecnico como para captar la importancia del mantenimiento

Problemas culturales

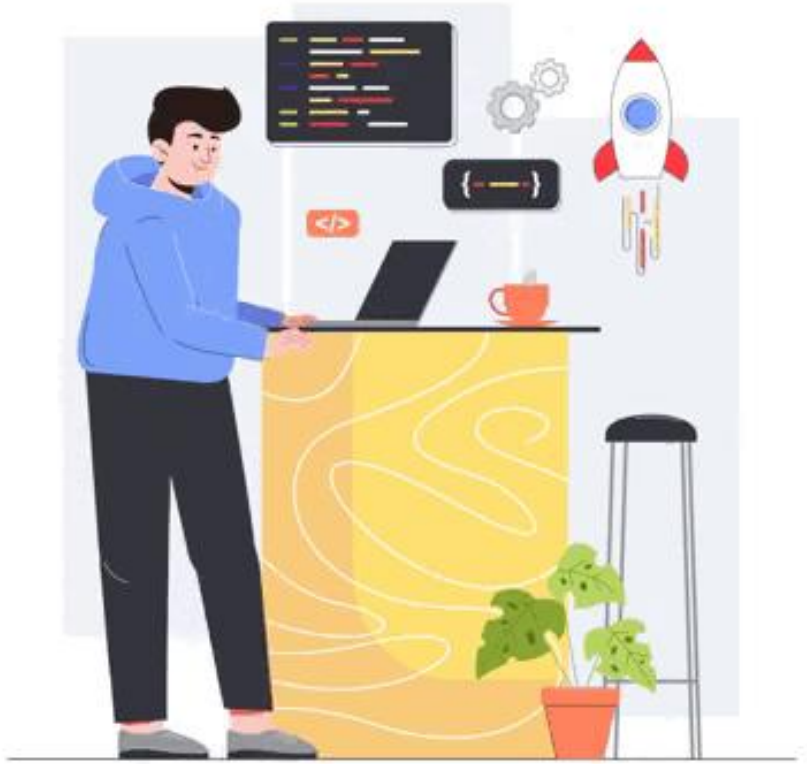
VS

2 Soluciones

Comunicarse de forma que muestre como afecta el no refactorizar ni mantener codigo a la empresa.

Dar importancia al mantenimiento como parte de la cultura trabajo

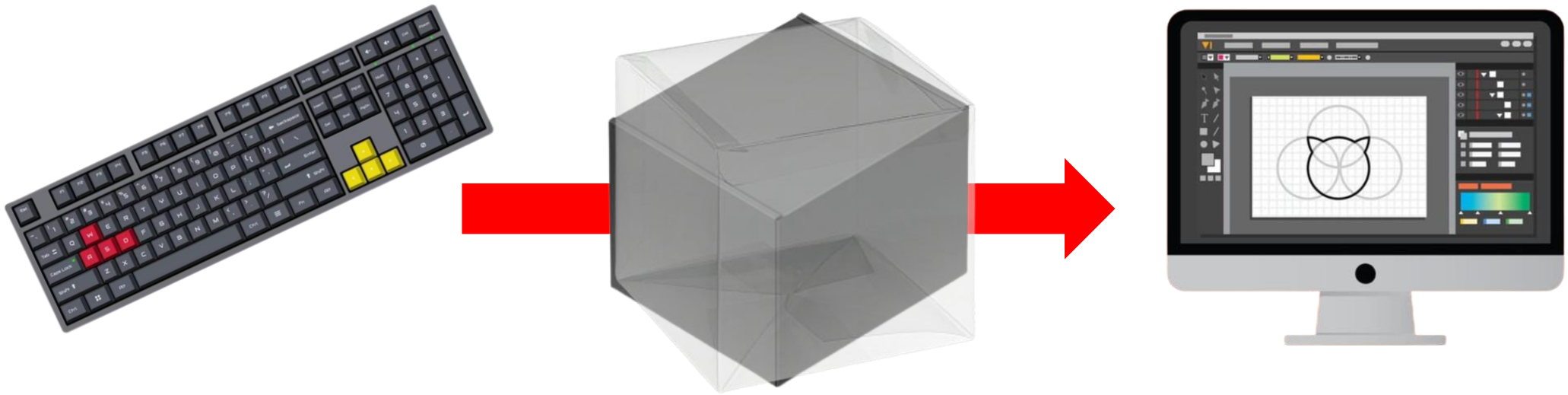
Patrones y Buenas Prácticas



Reescribir como "Desperdicio"

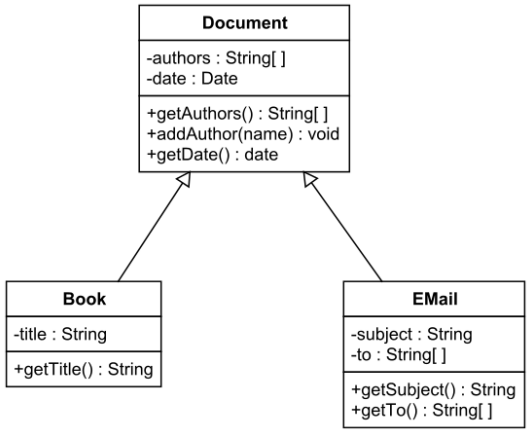
“No es un desperdicio, es una señal de éxito”

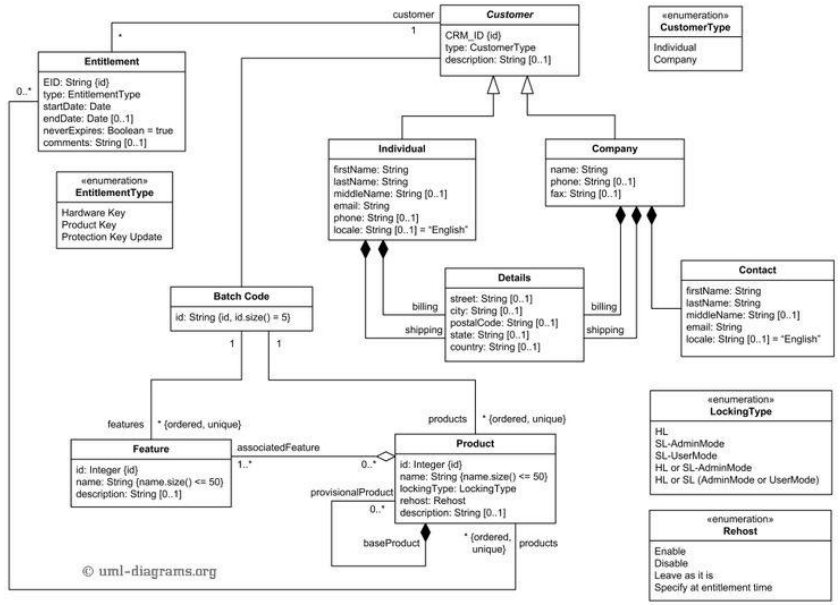
“Pasar de "0 a 1" requiere algo simple; pasar de "1 a 100" requiere escalabilidad “

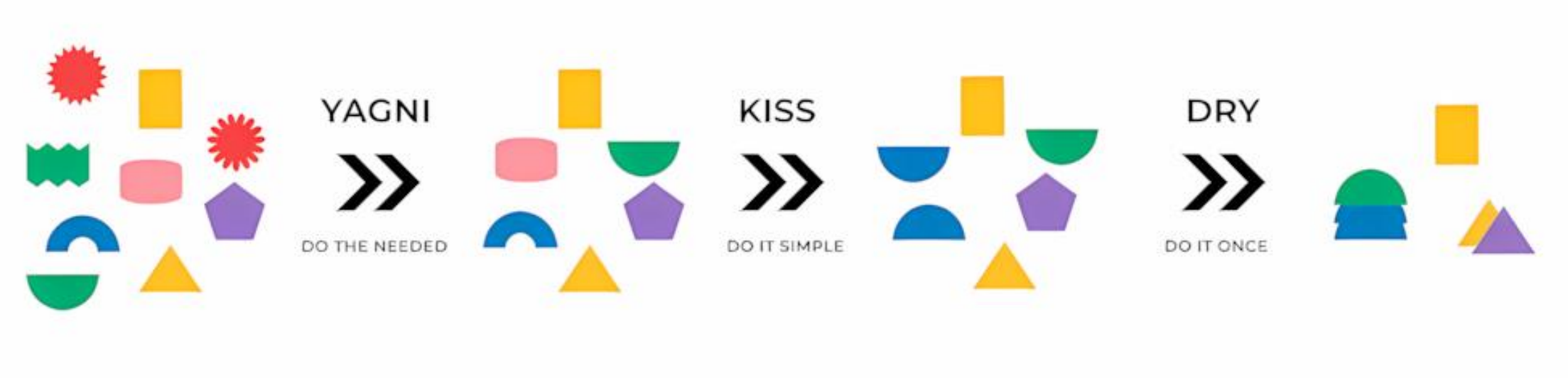


Marianne Belloti : *"Construye la cosa incorrecta de la manera correcta“*

- *Empieza simple*
- *No sobre-ingenieríes al principio*
- *Acepta que tendrás que revisitar y rehacer decisiones pasadas*







YOU AREN'T GONNA NEED IT

KEEP IT SIMPLE, STUPID

DON'T REPEAT YOURSELF

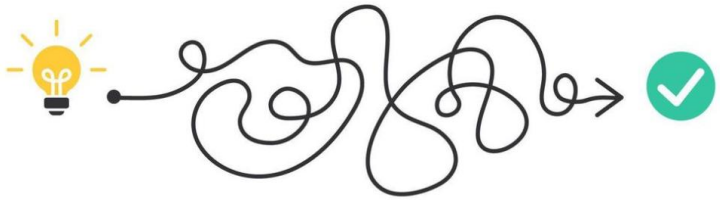
Acumular Refactorizaciones

Refactorización Continua

Ideal

- Refactorizar continuamente mientras se desarrollan nuevas features
- Hacer el cambio fácil y luego hacer el cambio fácil

Realidad



- A veces hay problemas grandes
- Requieren un esfuerzo extra
- Atarlo a un resultado de negocio

Cuando ya existe mucha Deuda Técnica

"Semana de Sostenibilidad”

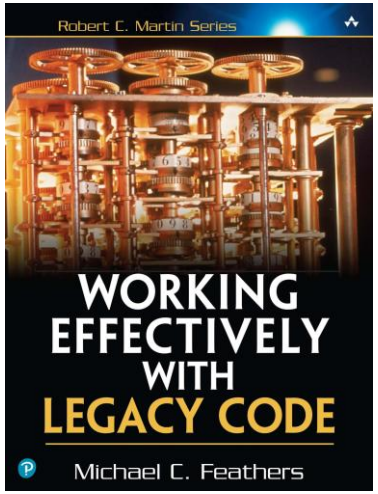
- Conciencia y Priorización Temporal
- Asignación Dedicada de Recursos
- Mejor que Nada

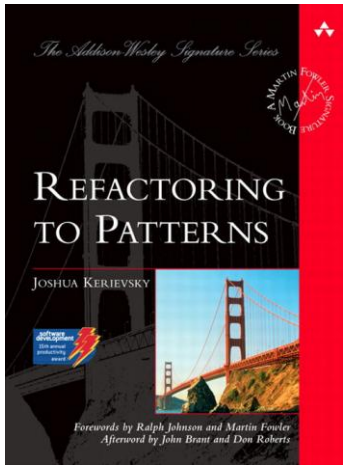
- Menos ideal que la cultura continua
- Riesgo de que se cancele o posponga por "otras prioridades“

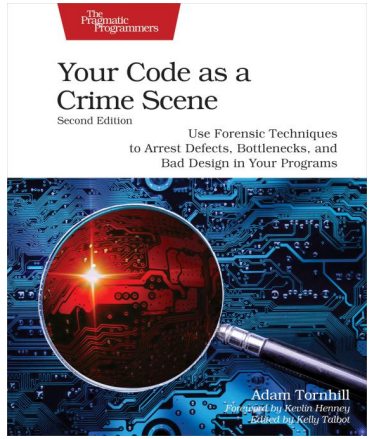
Involucrar Ingenieros en Producto

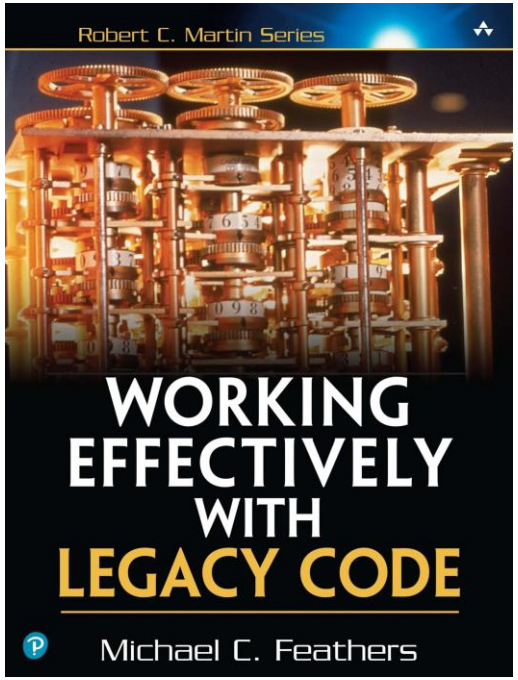
- Ingenieros que conocen el sistema
- Sobre qué construir a continuación

Comparación del Libro de Nicolas con Otros Clásicos

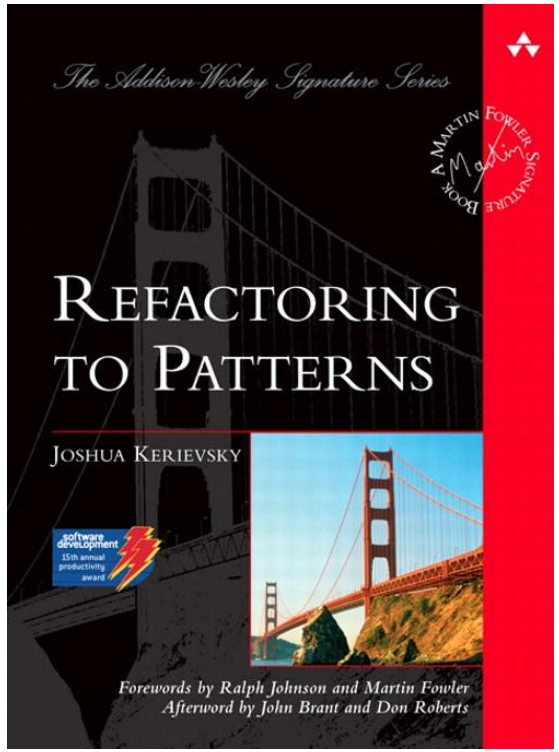




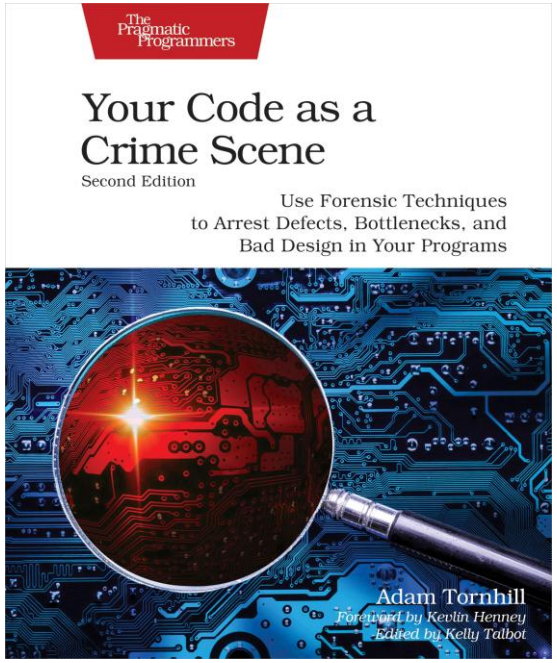




- Enfoque Principal (Técnicas vs. Mentalidad)
- Estilo (Botiquín vs. Guía)
- Nivel de Abstracción (Detallado vs. General)
- Uso Recomendado (Problemas Inmediatos vs. Entendimiento General)
- Objetivo (Solucionar vs. Comprender)



- Estilo Principal (Catálogo vs. Referencia Fundamental)
- Alcance (Legacy Code vs. Refactorización General)
- Profundidad (Solución Rápida vs. Mejora General)
- Tamaño (Conciso vs. Exhaustivo)
- Enfoque Principal (Código Heredado vs. Diseño del Código)



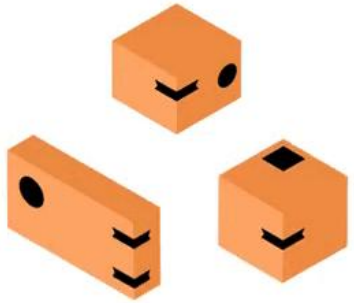
- Enfoque Principal (Resolución vs. Análisis)
- Fuente de Datos (Código vs. Historial)
- Objetivo Principal (Mantenibilidad vs. Priorización)
- Metodología (Aplicar Técnicas vs. Identificar Hotspots)
- Aplicación (Modificación Directa vs. Toma de Decisiones)

Detalles sobre las Herramientas del Libro

Análisis de Hotspots

Refactorizaciones Automatizadas

Invertir Dependencias

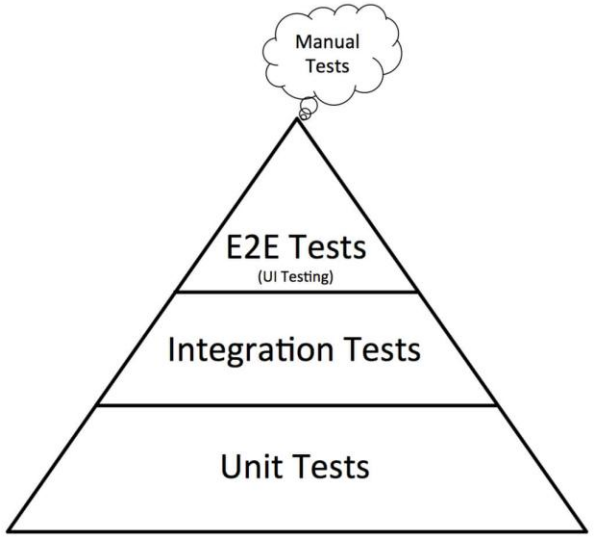


Naming como Proceso

IA y Código Heredado

- El uso de IA para trabajar con código heredado, gran tema desde 2023.
- Ayuda a entender fragmentos de código.
- ⚠️ No debemos confiar ciegamente en los resultados de la IA.

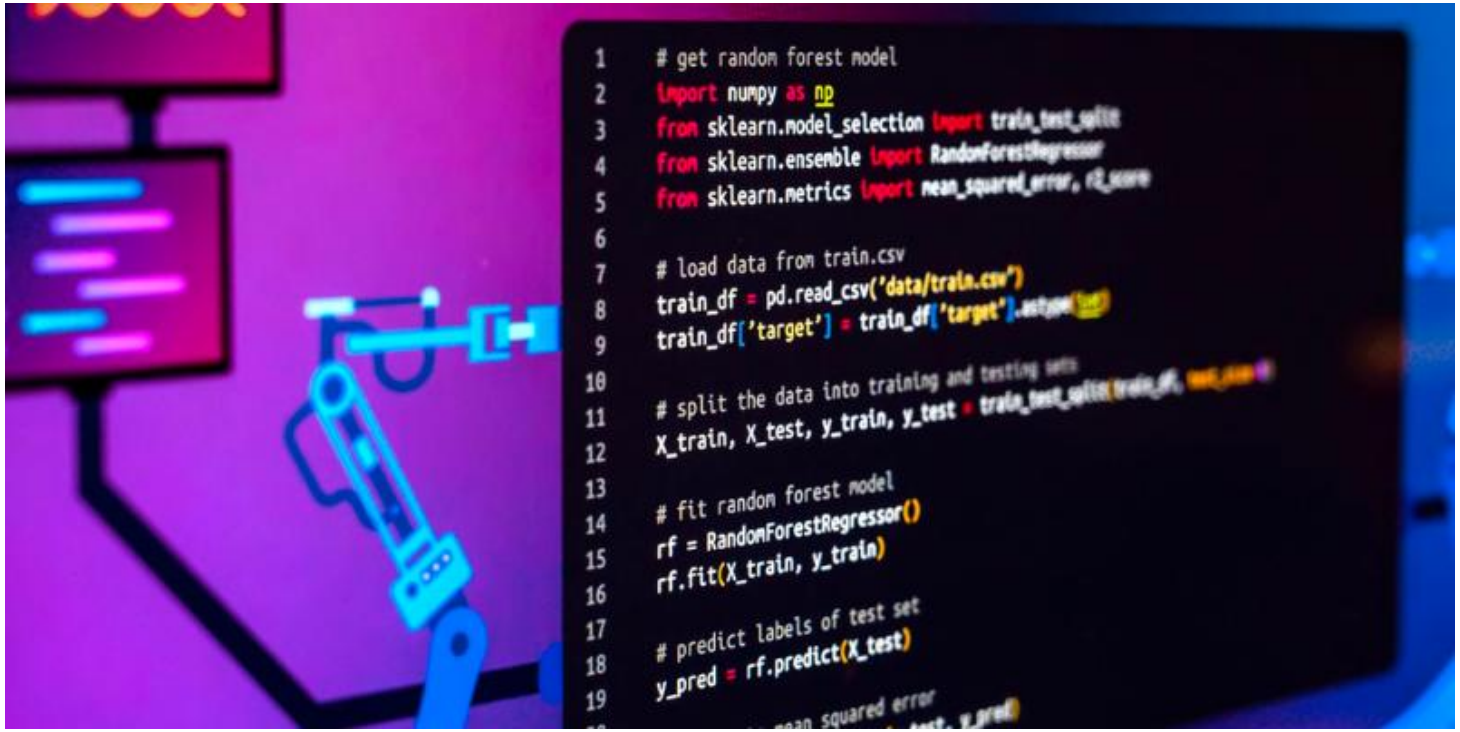
Test automáticos – Promesa y problemas



- Generación de tests automáticos.
- Dos principales problemas:
  - No se sabe si cubrirán todos los casos de uso.
  - Test demasiado rígidos lo cual dificulta la refactorización de código.
- Aún así la IA es útil para sugerir casos de uso.

Refactorización asistida por IA

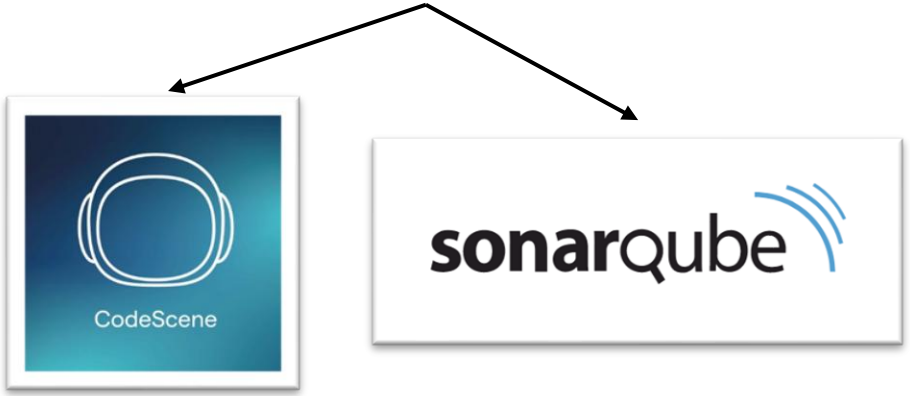
- La IA sugiere movimientos de refactorización y el editor los realiza de manera segura.



Análisis de Comportamiento

Flujo de ejecución → Dependencias entre parte de código → Puntos frágiles

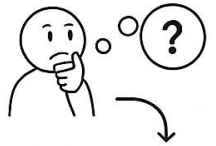
Pruebas automáticas → Herramientas de apoyo

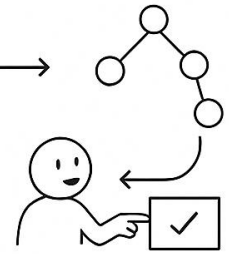


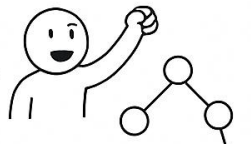
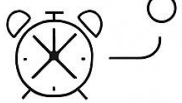
Método Mikado

Proceso

1. Define el objetivo final (ej. „Actualizar dependencia X"). Escribelo.
2. Intenta lograrlo en un timebox corto (ej. 10–15 min).
3. Falla (es lo esperado). El temporizador suena.
4. Piensa: ¿Qué me bloqueó? ¿Qué subtareas necesito hacer primero?
5. Revierte todos los cambios hechos (git reset —hard o git stash). ¡Este paso es crucial y dificil!
6. Dibuja: Añade las subtareas/ bloqueadores como nodos conectados al objetivo en un papel o mind map (el "Grafo Mikado").
7. Elige una subtarea hoja (la más pequeña/profunda).
8. Vuelve al paso 2 con esa subtarea como objetivo.
9. Eventualmente: Lograrás completar una subtarea en el timebox.
10. Haz commit de ese pequeño cambio exitoso.
11. Tacha ese nodo en el grafo.
12. Elige otra hoja y repite hasta completar el objetivo principal.

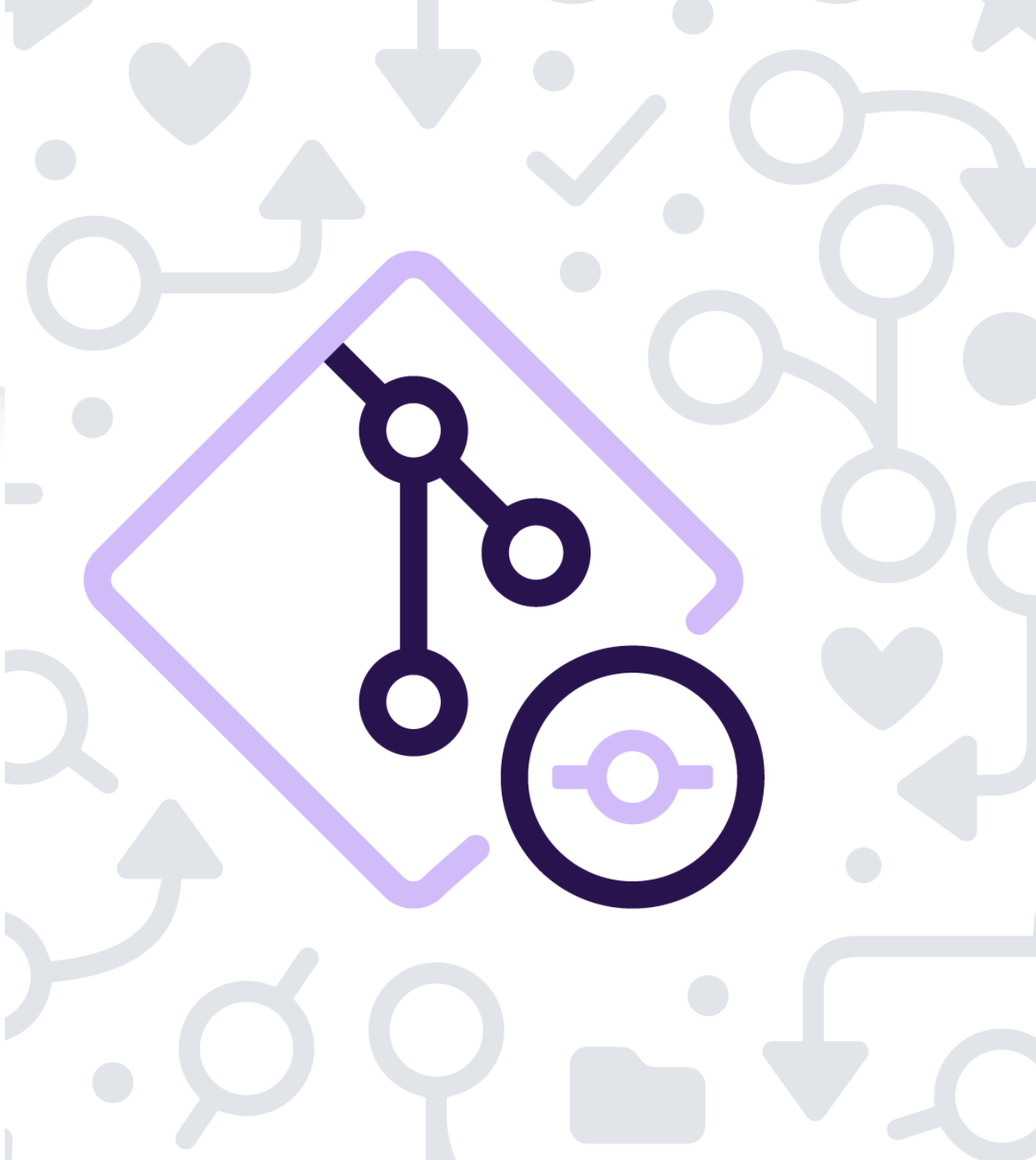






Commits pequeños y disciplina en el flujo de trabajo

- Practicar refactorings incrementales, hacer pequeñas mejoras y comprometer esos cambios antes de seguir.
- El Método Mikado ayuda a forzarlo.
- **Tip de Nicolas para entrenarse**: Usar un temporizador de gimnasio (gym timer) que vibre cada X minutos (ej. 5 min) como recordatorio constante para considerar hacer commit.



Lecciones Aprendidas desde la Escritura del Libro

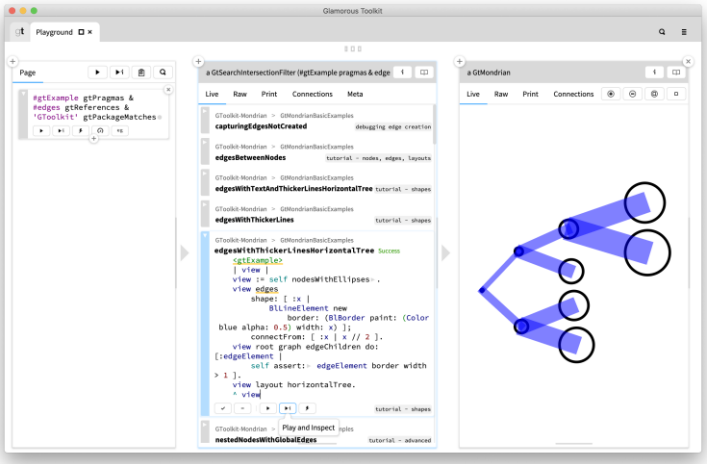
- Las 14 técnicas siguen siendo útiles y aplicables, incluso tres años después.
- La IA no estaba en el horizonte al momento de escribir el libro.
- Revisión de las técnicas a principios de año para verificar su vigencia.

Approval Testing (Pruebas de Aprobación)

- Práctica útil en código nuevo.
- Pueden mantenerse a largo plazo si la salida (printer) es útil y significativo.

Desarrollo Moldeable

- Forma distinta de programar.
- No solo se escribe código, sino que también se moldea el entorno de desarrollo.
- Herramienta principal: *Glamorous Toolkit*.



Ventajas y desafíos

Ventajas

- El editor también es una herramienta de documentación y visualización de la base de código

  “Puedes hacer que tu editor sirva como herramienta de documentación y visualización de tu base de código al mismo tiempo.”

  Nicolas Carlo

⚠ Desafíos

- Tiene una curva de entrada alta

  “No conozco a nadie que sepa usar esto y programar con ello. [..] Tiene una curva de entrada alta.”
  Nicolas Carlo

  “Puede tener mucho impacto [...] si trabajas en salud, finanzas, o algo así, puede valer la pena.”

  Nicolas Carlo

“Si quieres abrir la mente
sobre lo que es posible hacer,
te animo a mirar
Glamorous Toolkit.”

Nicolas Carlo

Gracias!

Arquitectura del Software

ALEJANDRO FERNÁNDEZ GARCÍA
IGNACIO FERNÁNDEZ SUÁREZ
LEONARDO CARONE MENES