# La Deuda Técnica

## El regalo que nadie quiere

POSADEY

# Introducción

- Presentación
  - Manuel Rábade
  - manuel@rabade.net
- Preguntas
  - Experiencia
  - Trayectoria

- Motivación
  - No me agradaba porque..
  - No es atractivo, pero…
  - Hay información, pero…
- ¿Qué vamos a ver?
  - Teoría
  - Adquirir
  - Gestionar

# Origen del concepto

Ward Cunningham (1992):

***Mature sections** of the program have been **revised or rewritten many times** providing the consolidation that is key to understanding and continued incremental development*

***Immature code** may work fine and be completely acceptable to the customer, **excess quantities will make a program unmasterable**, leading to extreme specialization of programmers and finally an inflexible product.*

***Shipping first time code is like going into debt**. A little debt speeds development so long as it is paid back promptly with a rewrite.*

***The danger occurs when the debt is not repaid.** Every minute spent on not-quite-right code counts as interest on that debt. Entire engineering organizations can be brought to a stand-still under the debt load of an unconsolidated implementation.*

# Evolución

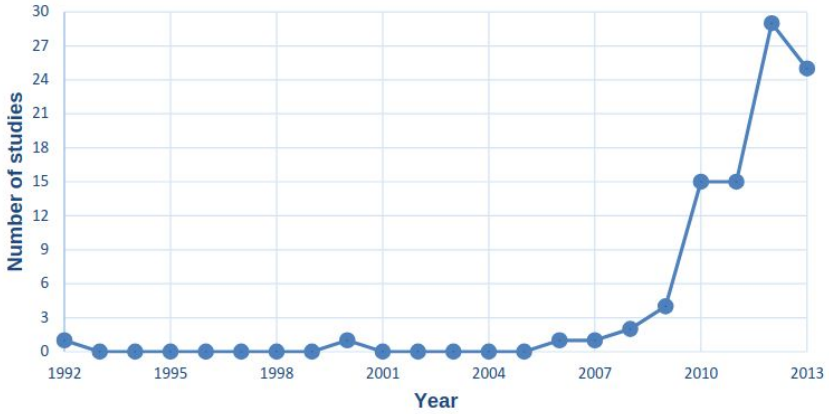
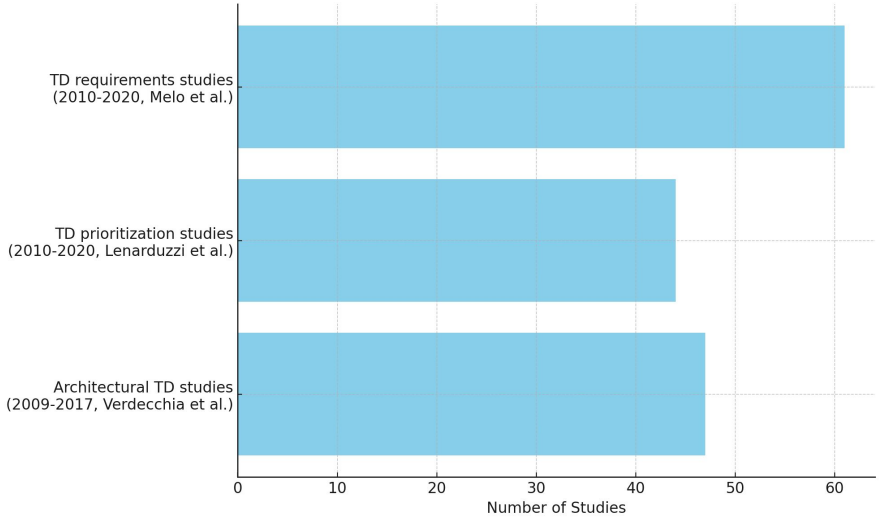



Fig. 5. Distribution of selected studies over time period.




A systematic mapping study on technical debt and its management (2015)
Technical debt in systems engineering: A systematic literature review (2023)

# Definición

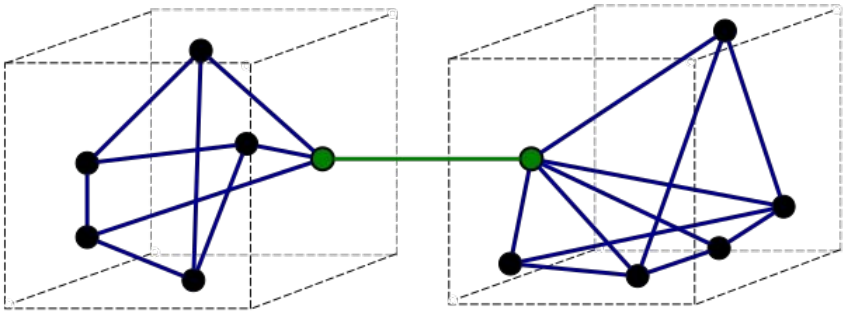
- Compromisos que optimizan el desarrollo de software en el corto plazo a expensas de la calidad o sostenibilidad del software en el largo plazo.

  Solo son deuda si bloquean los posibles resultados del negocio.

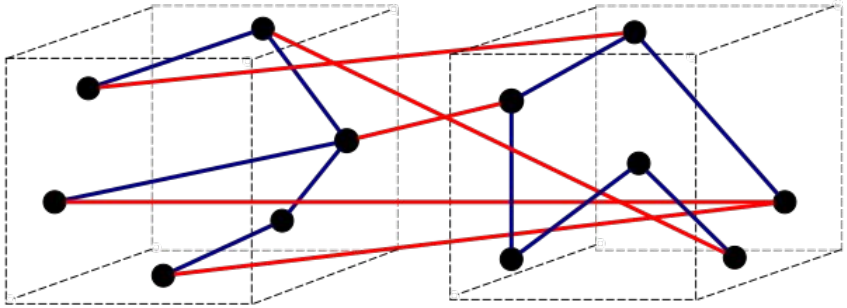
- Los intereses de la deuda son el costo adicional que se acumula con el tiempo al mantener o trabajar sobre un sistema técnicamente endeudado.
- No es algo negativo si es un compromiso consciente que permite ganar velocidad en momentos clave del desarrollo.

# Ejemplos

- Falta de pruebas automatizadas
- Dependencias desactualizadas
- Documentación incompleta
- Código altamente acoplado con baja cohesión
- Procesos manuales repetitivos
- Silos de conocimiento



a) Good (loose coupling, high cohesion)

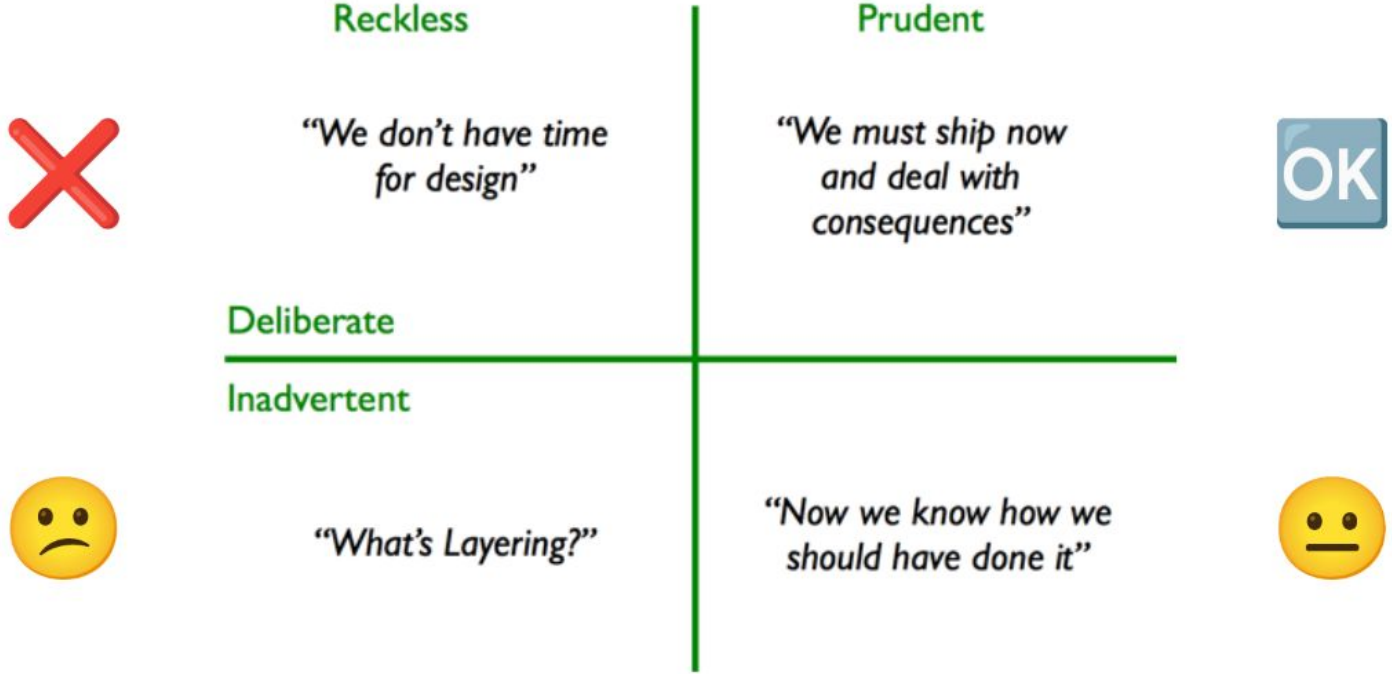


b) Bad (high coupling, low cohesion)

# Para comprender la deuda técnica

- Los requisitos de desarrollo son, muy probablemente, hipótesis.
- Escribir código es fácil, lo difícil es mantenerlo y escribirlo considerando su mantenimiento.
- La acumulación de deuda técnica por priorizar el rápido desarrollo de funcionalidad a expensas de la calidad, desempeño, pruebas, etc es algo muy común.
- Es necesario adoptar la perspectiva de los desarrolladores en lugar de imponer definiciones externas y, desde ahí, trabajar en cómo abordarla.
- Es fundamental la capacidad para concebir el estado ideal de un sistema y usarlo como referencia para evaluar su estado actual.

# Adquirir deuda técnica sabiamente




Technical Debt Quadrant (2009) de Martin Fowler

# ¿Es un problema la deuda técnica?

Señales para determinar si es un problema:

1. Tiempo para la entrega de funcionalidad
2. Impacto al usuario final
3. Satisfacción de desarrolladores
4. Capacidad para incorporar nuevos desarrolladores
5. Degradación de métricas

# Gestionar deuda técnica

1. Comprender el contexto
2. Identificar
3. Clasificar y priorizar
4. Abordar

# Comprender el contexto

- Dirección del negocio
  - Misión, visión, objetivos, KPIs
- Impacto del equipo
  - Crecimiento, expansión, sostenibilidad, satisfacción del cliente
- Etapa del negocio
  - Experimentación, tracción, crecimiento, optimización

# Identificar

- Crear un documento que liste la deuda técnica conocida
- Usar experiencia y conocimiento
- Tomar como referencia las mejores prácticas de la industria
- Metodologías cualitativas y cuantitativas
- Utilizar marcos de clasificación como el de Google o ThoughtWorks

# Marcos de clasificación

**Google**

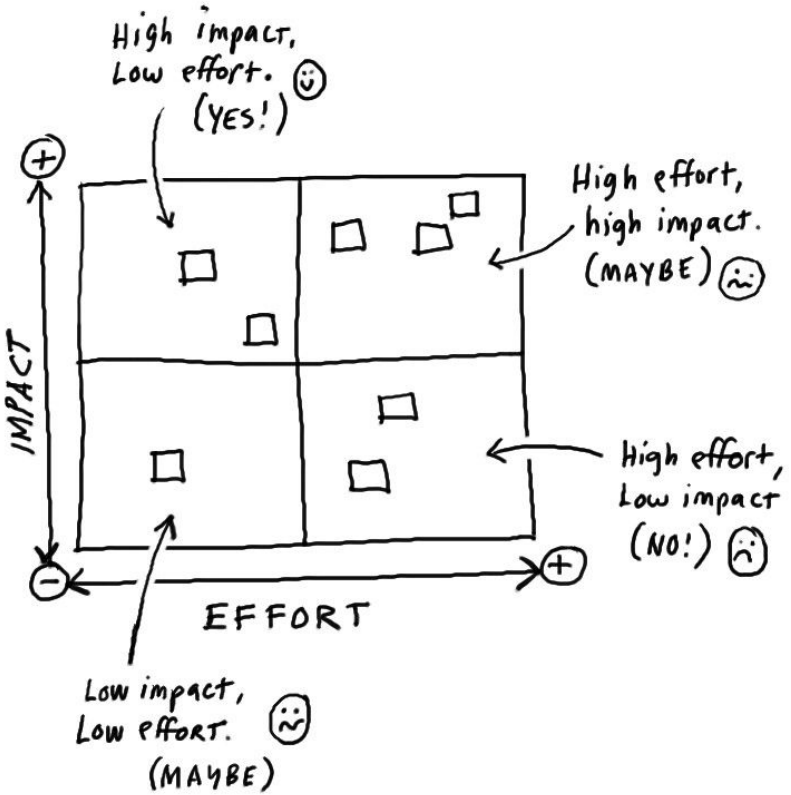
1) Se necesita una migración o está en proceso
2) Documentación del proyecto y APIs
3) Pruebas
4) Calidad del código
5) Código inactivo o abandonado
6) Degradación del código
7) El equipo carece de la experiencia necesaria
8) Dependencias
9) La migración se ejecutó de manera deficiente o se abandonó
10) Proceso de lanzamiento

**ThoughtWorks**

1) Calidad del código
2) Pruebas
3) Acoplamiento
4) Funcionalidades sin uso o de bajo valor
5) Bibliotecas o frameworks obsoletos
6) Herramientas ineficientes
7) Problemas de confiabilidad y rendimiento
8) Procesos manuales
9) Despliegues automatizados
10) Compartir conocimiento

# Clasificar y priorizar

- Clasificar cada elemento en función del impacto y el esfuerzo requerido
- Priorizar los elementos con una matriz de valor/esfuerzo
- Involucrar otros stakeholders



# Abordar

- Integrar elementos de alta prioridad en el roadmap de producto
- Comunicar el impacto en términos de negocio
  - En lugar de decir: "Necesitamos solucionar nuestra deuda técnica", explicarlo así: "Nuestro stack tecnológico actual provoca demoras en el desarrollo de nuevas funcionalidades, lo que afecta nuestra velocidad de lanzamiento y la satisfacción del cliente"
- Enfoques
  - Mejora continua
  - Proyectos a gran escala

# Niveles de madurez

**Nivel 1: Reactivo**
Sin procesos reales para gestionar la deuda técnica.

**Nivel 2: Proactivo**
Identificar y rastrear la deuda técnica. Tomar decisiones sobre su urgencia e importancia.

**Nivel 3: Estratégico**
Un enfoque proactivo más designar campeones para mejorar la planificación y la toma de decisiones en torno a la deuda técnica e identificar y abordar causas raíz.

**Nivel 4: Estructural**
Un enfoque estratégico más medidas para optimizar la gestión de la deuda técnica localmente, integrando consideraciones en el flujo de trabajo del desarrollador y estandarizando cómo se maneja en toda la organización.

Defining, Measuring, and Managing Technical Debt (2023)

# Cierre

¿Un sistema monolítico es deuda técnica?