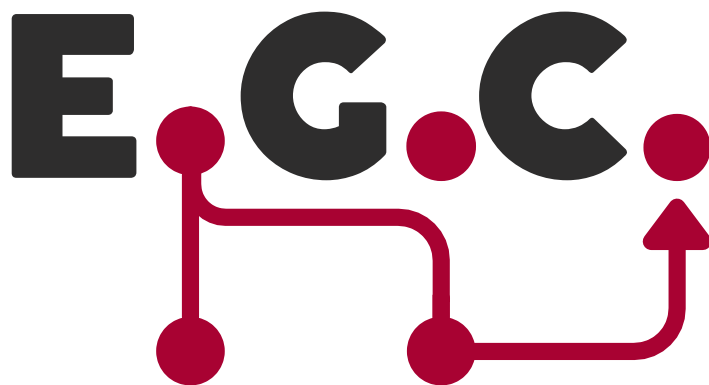




Grado en Ingeniería Informática - Ingeniería del Software

Evolución y Gestión de la Configuración



Escuela Técnica Superior de
Ingeniería Informática

Pruebas de software

1. Introducción

2. ¿Qué es una prueba software?
 3. Tipos y niveles de prueba
 4. Diseño de casos de prueba
 5. Desarrollo guiado por pruebas
 6. ¿Y cómo afecta la IA a todo esto?
 7. Reflexión final
- 

1. Introducción | ejemplo de otro dominio

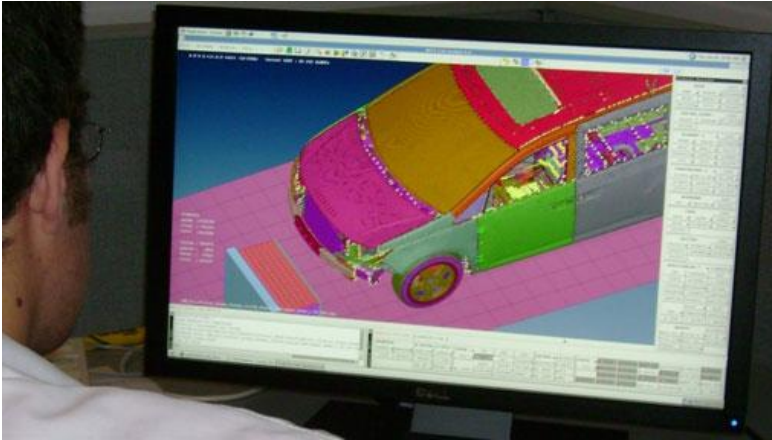
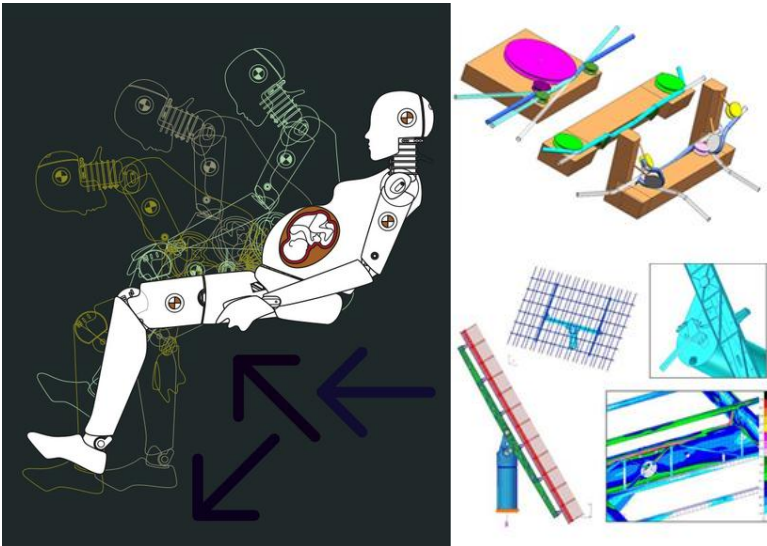


Probar: hacer examen y experimento de las cualidades de alguien o algo.



Test: *a critical examination, observation or evaluation.*

1. Introducción | ejemplo de otro dominio

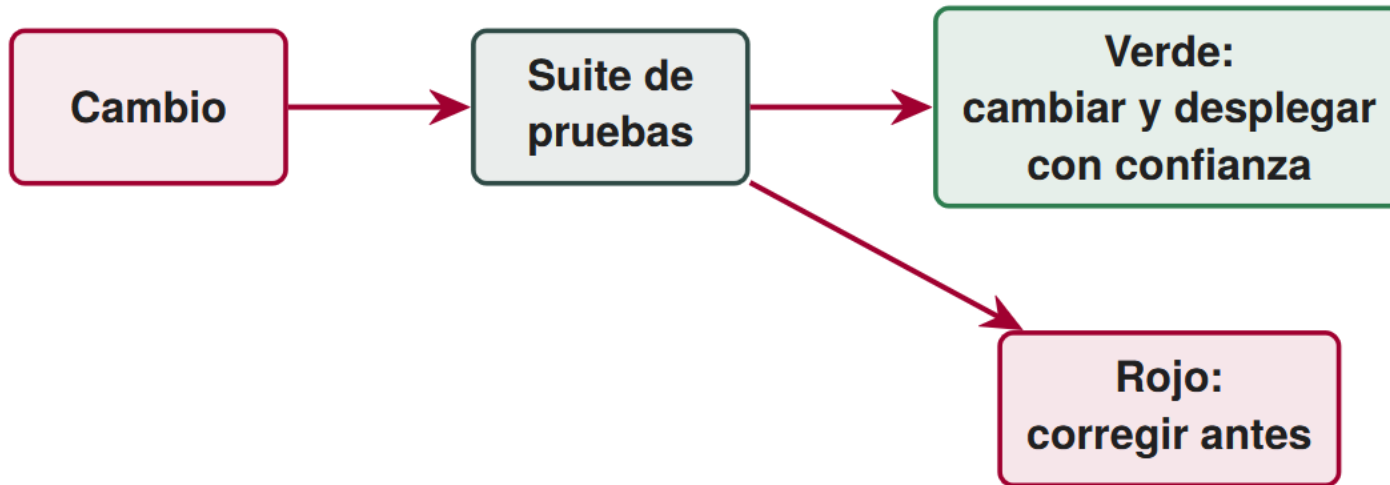


Diseño de la prueba



Ejecución de la prueba

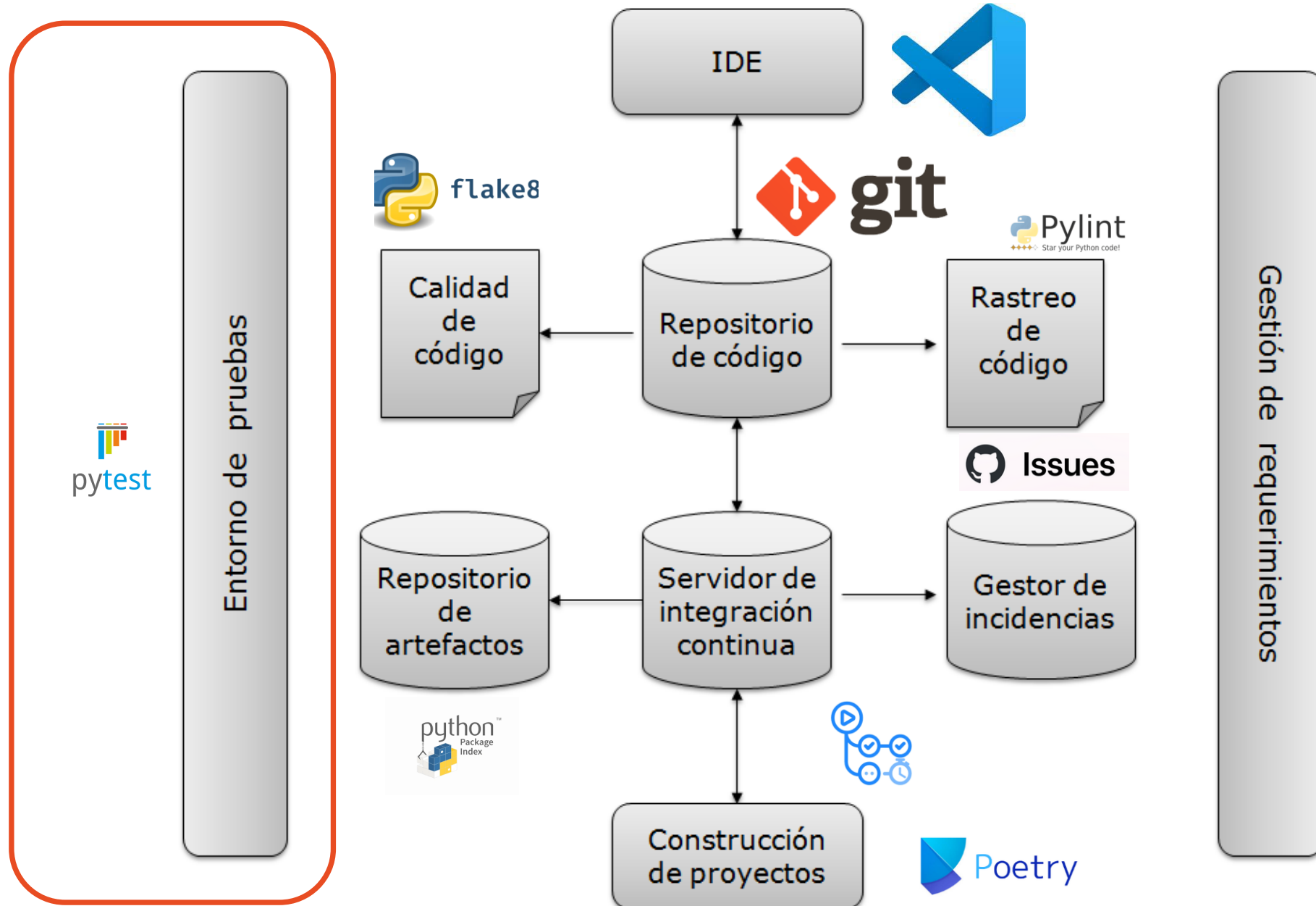
1. Introducción | detector de problemas



“Las pruebas automatizadas reducen el miedo del equipo de desarrollo a incluir nuevas funcionalidades o a refactorizar el código”

Robert C. Martin
Clean Craftsmanship

1. Conceptos básicos | ecosistemas de desarrollo



1. Introducción | ejemplos



Autenticación:

- **Crear usuario** → probar si un usuario puede ser creado exitosamente en el sistema mediante el formulario de registro
- **Inicio de sesión** → probar que el usuario puede iniciar sesión con credenciales válidas
- **Autenticación fallida** → probar que los intentos de iniciar sesión con credenciales incorrectas son rechazados

Hubfile:

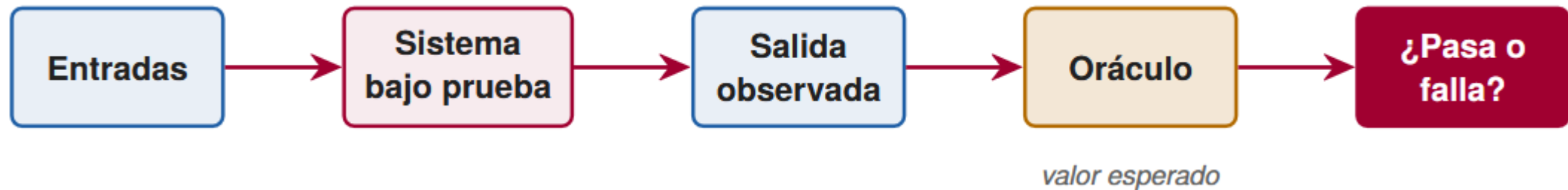
- **Transformación UVL exitosa** → probar que un archivo UVL es procesado y transformado correctamente a otro formato. Por ejemplo, JSON
- **UVL mal formado** → probar si el sistema responde como se espera (con un error controlado) cuando el archivo UVL no es válido

1. **Introducción**
 2. **¿Qué es una prueba software?**
 3. Tipos y niveles de prueba
 4. Diseño de casos de prueba
 5. Desarrollo guiado por pruebas
 6. ¿Y cómo afecta la IA a todo esto?
 7. Reflexión final
- 

2. ¿Qué es una prueba software?

- Las pruebas de software nos permiten evaluar de forma sistemática si un programa cumple sus requisitos y funciona como esperamos.
- Una prueba ejecuta una parte del sistema bajo unas condiciones concretas, observa el resultado y lo compara con lo que esperábamos que ocurriera:
 - Si el resultado observado coincide con el esperado, la prueba pasa.
 - Si no coincide, la prueba falla (y nos está indicando que hay algo que debemos investigar).

2. ¿Qué es una prueba software? | proceso general



- Un caso de prueba se suele documentar identificando los siguientes puntos (*IEEE Standard for Software Testing Documentation*):
 - **Identificador** del caso de prueba
 - **Entradas**
 - **Salidas** esperadas
 - **Dependencias** (si es necesario ejecutar antes otros casos de prueba)

2. ¿Qué es una prueba software? | ¿qué pueden demostrar las pruebas?



Demostración formal: Análisis que permite *afirmar* que la implementación cumple sus requisitos



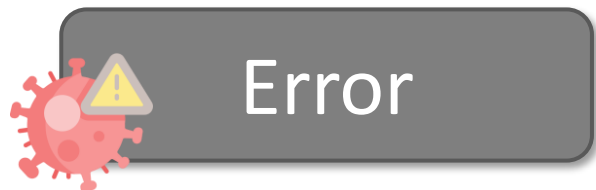
Prueba: Análisis que permite *aumentar nuestra confianza* en que la implementación cumple sus requisitos

“Testing shows the presence, not the absence of bugs”

Dijkstra

2. ¿Qué es una prueba software? | definiciones

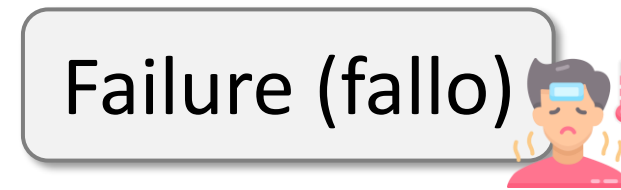
Equivocación inicial o exposición a un problema, que introduce una condición errónea en el sistema.



La causa que queda introducida en el software como consecuencia del error (y que puede provocar un fallo).



Comportamiento incorrecto de un sistema o incapacidad para realizar su función. Se podría definir como "los síntomas".



Debugging



El proceso de encontrar un bug dado un fallo.

Testing



Evaluar el software observando su ejecución. Si una prueba falla, hay un comportamiento que se debe investigar.

2. ¿Qué es una prueba software? | observar un fallo

Se necesitan **tres condiciones** para poder observar un fallo (**RIP**)

Accesibilidad
(**R**eachability)

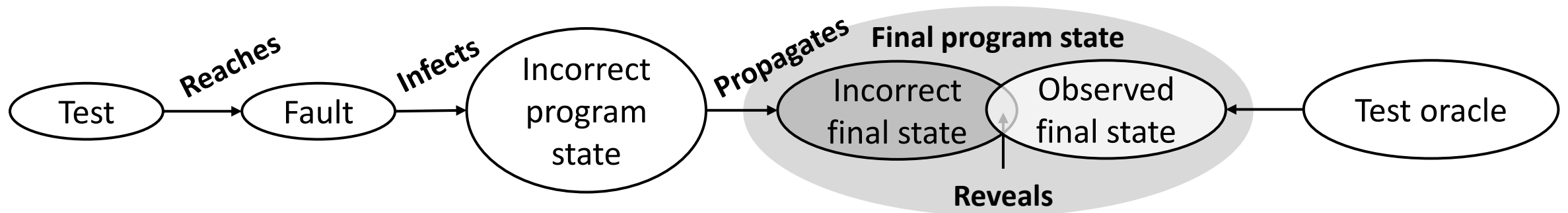
El lugar/lugares donde se encuentra el fallo deben ser accedidos.

Infección
(**I**nfection)

Una vez accedido, el estado del programa debe ser incorrecto.

Propagación
(**P**ropagation)

El estado infectado debe propagarse para poder observar su estado incorrecto



2. ¿Qué es una prueba software? | actividades

- Se pueden distinguir 4 tipos de actividades

1. Diseño



3. Ejecución



2. Automatización



4. Evaluación



- Cada actividad requiere distintas competencias, pero es frecuente que se use a la misma persona para las 4 actividades (no recomendado, especialmente para sistemas complejos y críticos).

1. **Introducción**
 2. **¿Qué es una prueba software?**
 3. **Tipos y niveles de prueba**
 4. **Diseño de casos de prueba**
 5. **Desarrollo guiado por pruebas**
 6. **¿Y cómo afecta la IA a todo esto?**
 7. **Reflexión final**
- 

3. Tipos y niveles de prueba | pruebas funcionales vs no funcionales

	Pruebas funcionales	Pruebas no funcionales
Definición	Evaluar que el sistema haga lo que se supone que debe hacer	Evaluar cómo funciona el sistema
Objetivo	Optimizar la correcta implementación de los requisitos del sistema	Optimizar y observar aspectos de calidad del sistema
Qué evalúan	Funciones específicas	Características de calidad
Beneficios	Identificación temprana de fallos y verificación de requisitos	Mejorar de la experiencia del usuario, garantía de escalabilidad y seguridad del sistema

3. Tipos y niveles de prueba | pruebas funcionales vs no funcionales

La Universidad de Sevilla colapsada el primer día de automatrícula

LUNES, 07 DE SEPTIEMBRE DE 2009 10:33 SEVILLA - CULTURA Y EDUCACIÓN



Los servidores de la Universidad de Sevilla quedaron completamente bloqueados con la apertura del plazo de automatrícula desde las 9 de esta mañana. Los universitarios que consiguieron acceder a la aplicación, tras esperar 10-15 minutos para que cargase la ventana de automatrícula, recibían una indicación de error de acceso obligándoles a repetir el proceso una y otra vez.



3. Tipos y niveles de prueba | pruebas unitarias

- Diseñadas para ejercitar una parte **pequeña y específica** de funcionalidad.
- Ayudan a **acotar** el ámbito de un fallo.
- Prueban que una pequeña parte **aislada** del software es correcta.
- No debe ir más allá de sus **límites**, aunque a veces no se pueden evitar las dependencias



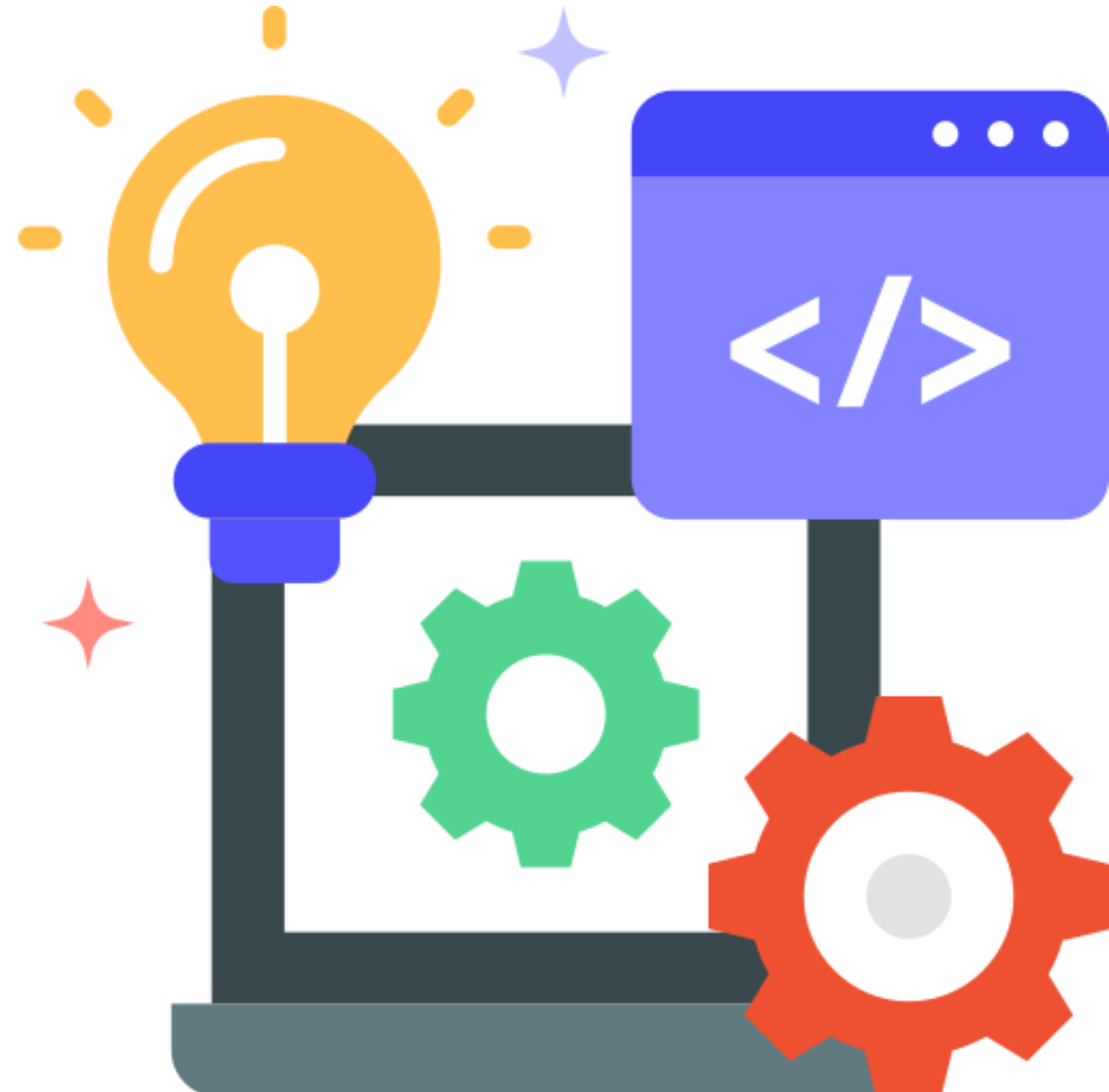
3. Tipos y niveles de prueba | pruebas de integración

- Observar si los módulos o componentes funcionan bien cuando se **combinan**.
- Interacciones **internas** (entre módulos) o **externas** (con servicios).
- Detectan fallos en la **interfaz de comunicación** entre componentes.



3. Tipos y niveles de prueba | pruebas de sistema

- Validar que el sistema completo cumple con los **requisitos** funcionales y no funcionales.
- Abarcan **todos** los componentes y su **interacción** en el entorno real.
- Detectan errores de **arquitectura** y observan si el sistema completo se comporta como se espera.

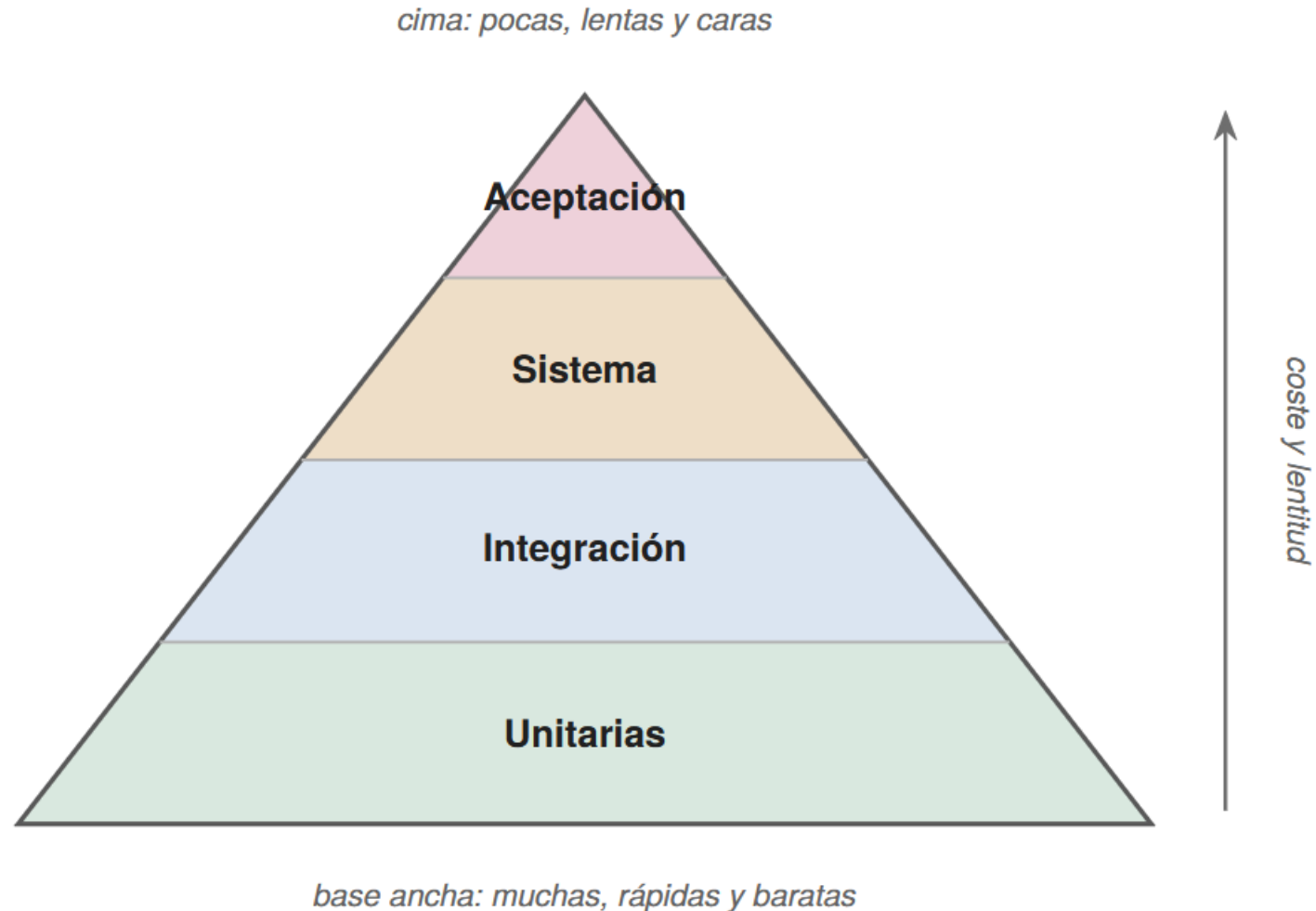


3. Tipos y niveles de prueba | pruebas de aceptación

- Confirmar que el software cumple con los **criterios de aceptación** definidos por el cliente.
- Abarcan todo el sistema desde el punto de vista del **usuario final**.
- Son el acuerdo por el que el sistema es **apto para su entrega y uso**, centrándose en la satisfacción del usuario.



3. Tipos y niveles de prueba | pirámide de pruebas



1. **Introducción**
 2. **¿Qué es una prueba software?**
 3. **Tipos y niveles de prueba**
 4. **Diseño de casos de prueba**
 5. **Desarrollo guiado por pruebas**
 6. **¿Y cómo afecta la IA a todo esto?**
 7. **Reflexión final**
- 

4. Diseño de casos de prueba

*“Automatizar pruebas sin
diseño es como hacer una
pizza sin base”*

- **profe de EGC**

4. Diseño de casos de prueba

Un plan de pruebas exhaustivo es impracticable

Las posibles entradas de un programa pueden (y suelen) ser infinitas, e.g. compilador

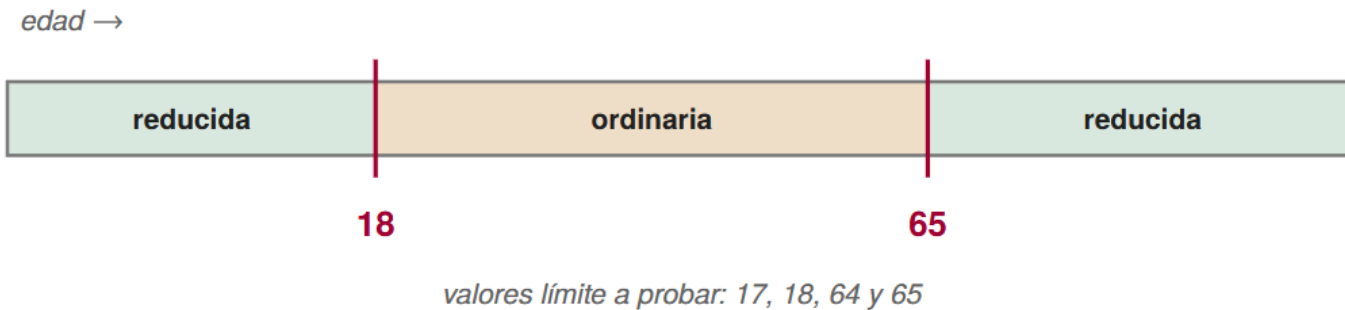
El objetivo de un buen banco de pruebas (*test set*, *test suite*): pocas entradas, muchos fallos.

¿Cómo seleccionar las entradas? Usando un criterio de cobertura (*coverage criteria*)

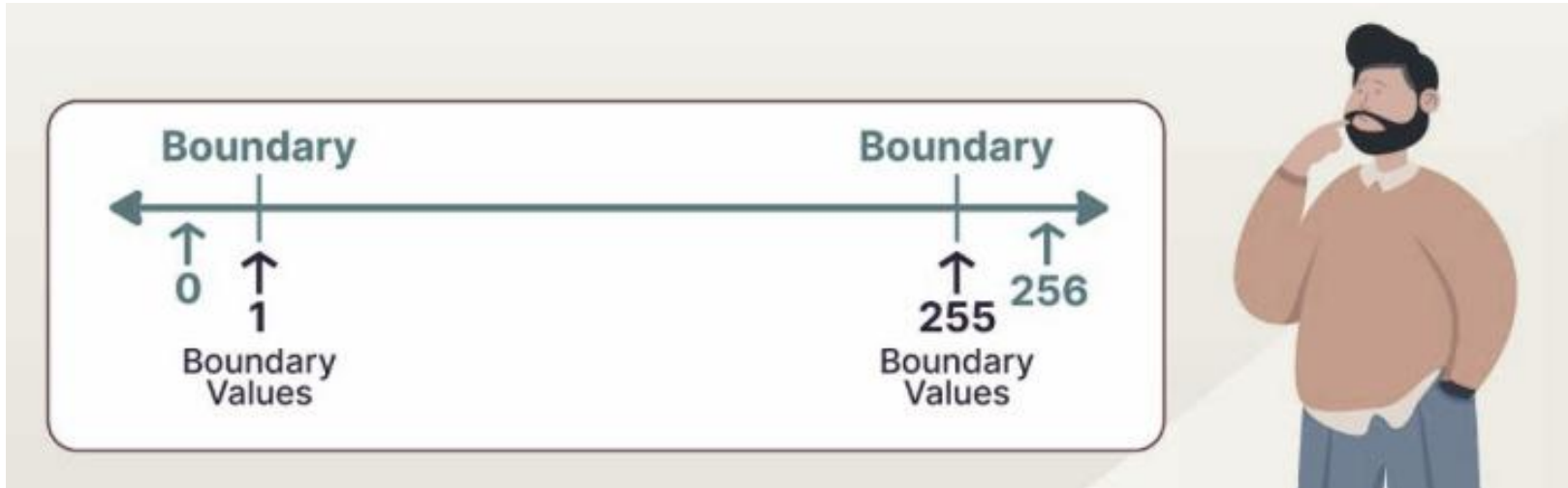
Criterio de cobertura: conjunto de reglas que imponen una serie de requisitos a un banco de pruebas (*test suite/ test set*).

4. Diseño de casos de prueba | particiones equivalentes

- Se divide el espacio de prueba en clases equivalentes
- Las particiones deben ser disjuntas
- Inconveniente: rápida explosión



4. Diseño de casos de prueba | valores límite



- Según esta técnica, los errores en los programas suelen estar en los valores límite de sus entradas.
- Se prueban los valores límite de las particiones equivalentes, en caso de haberlas.

4. Diseño de casos de prueba | particiones equivalentes y valores límite

Ejemplo

El precio de la matrícula depende de la edad del cliente. Hay tres rangos: menores de 18 años, entre 18 y 65 años y mayores de 65 años.

Entonces, hay 3 **clases equivalentes**: 0-17, 18-64, 65-*

Usamos los **valores límite**: 16,17,18,19,64,65,66

O simplificado (el límite y los adyacentes): 17,18,64,65

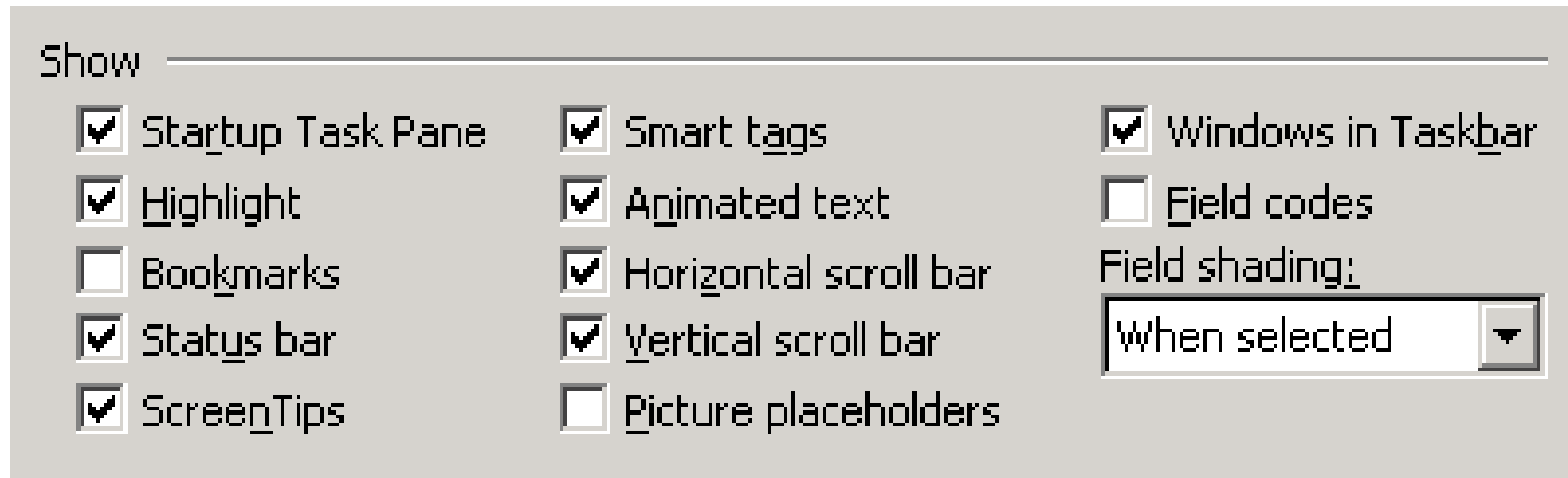
4. Diseño de casos de prueba | combinaciones pairwise, n-wise



Método **combinatorio** que propone hacer pruebas sobre todas las posibles combinaciones de dos parámetros de entrada.

Para un sistema que tiene n entradas binarias, el número total de combinaciones posibles de esas entradas es 2^n

4. Diseño de casos de prueba | combinaciones pair-wise, n-wise



$2^{12} \times 3 = 12.288$ combinaciones !!

4. Diseño de casos de prueba | combinaciones pair-wise, n-wise

Imagina X, Y, Z booleanos:

Todas las posibles pruebas

	X	Y	Z
T1	0	0	0
T2	0	0	1
T3	0	1	0
T4	0	1	1
T5	1	0	0
T6	1	0	1
T7	1	1	0
T8	1	1	1

Aplicando pair wise

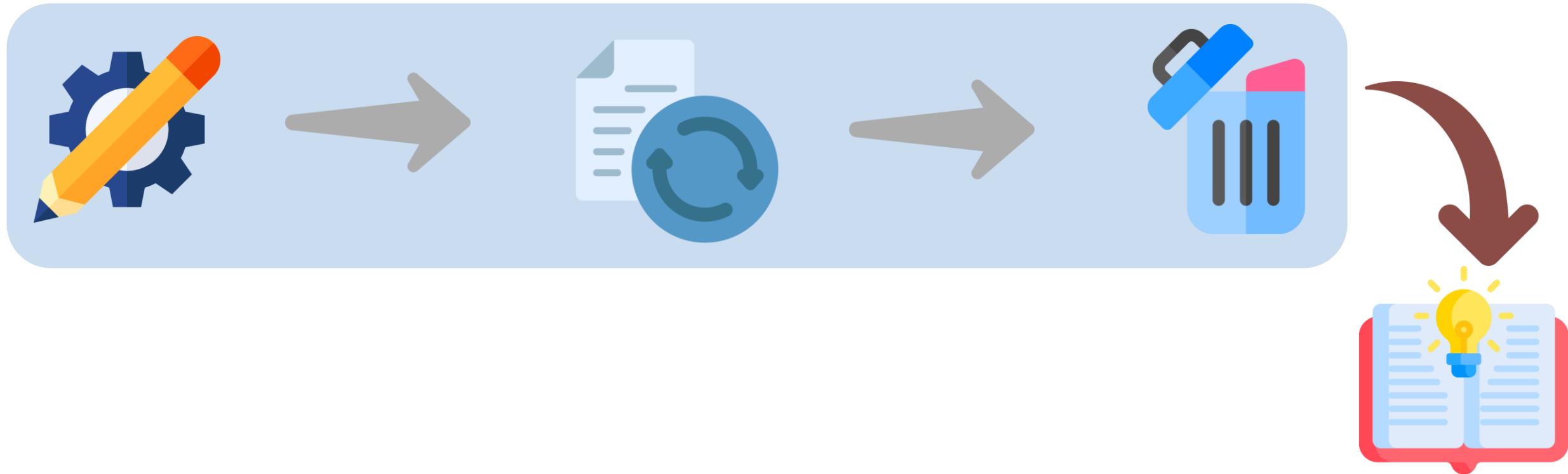
	X	Y	Z
T1	0	0	0
T4	0	1	1
T6	1	0	1
T7	1	1	0

π



4. Diseño de casos de prueba | cobertura CRUD

Para **cada elemento persistente**, se prueban todas sus **operaciones CRUD**.



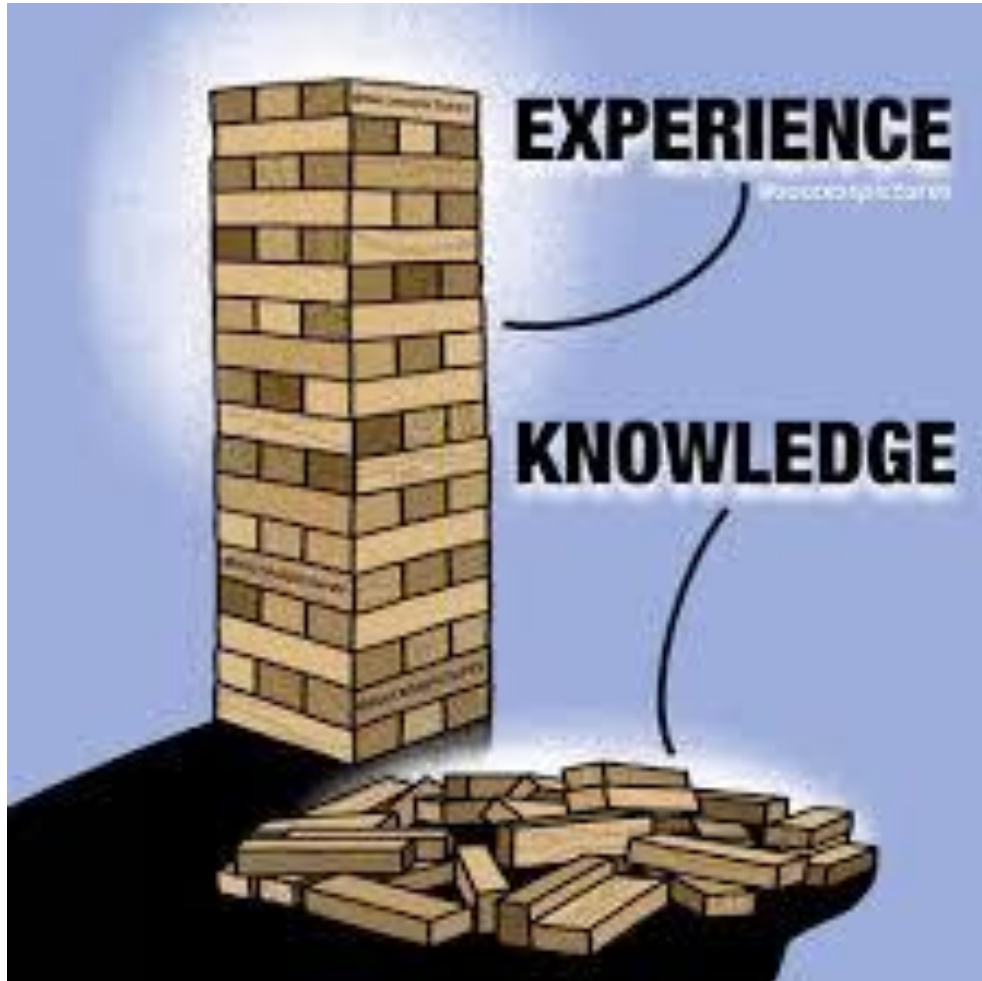
4. Diseño de casos de prueba | cobertura CRUD

- Para cada elemento persistente, se prueban todas sus operaciones *CRUD*
- Cada caso de prueba comienza por una *C*, seguida por todas las *R* y se termina por una *D*
- Tras cada *C*, *U* o *D*, se ejecuta una *R* que nos sirve de oráculo

Ejemplo:

- Crear *DataSet* (*C*)
- Verificar que se ha creado (*R*)
- Actualizar sus valores (*U*)
- Verificar que se han actualizado (*R*)
- Borrar *DataSet* (*D*)
- Verificar que se ha borrado (*R*)

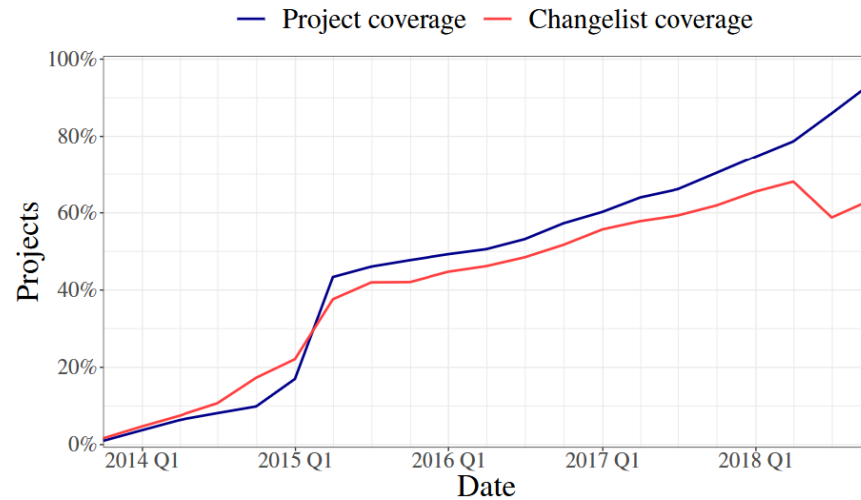
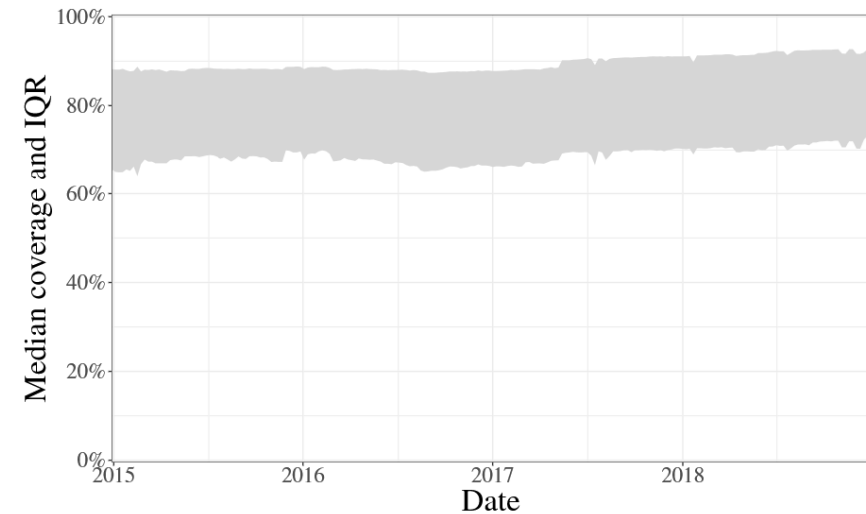
4. Diseño de casos de prueba | errores conocidos



Se basa en la **experiencia** e **intuición** del *tester* para identificar errores en el sistema.

En lugar de seguir un conjunto estructurado de casos de prueba, el *tester* se anticipa en detectar **áreas donde es probable que existan defectos**.

4. Diseño de casos de prueba | cobertura de código



SENTIMENT	RESPONSES	PERCENTAGE
Positive	173	60%
Neutral	67	24%
Negative	46	16%
Total	286	100%

Code coverage at Google

Marko Ivankovic · [Goran Petrovic](#) · René Just · Gordon Fraser ·
Proceedings of the 2019 27th ACM Joint Meeting on European
Software Engineering Conference and Symposium on the
Foundations of Software Engineering, ACM, pp. 955-963

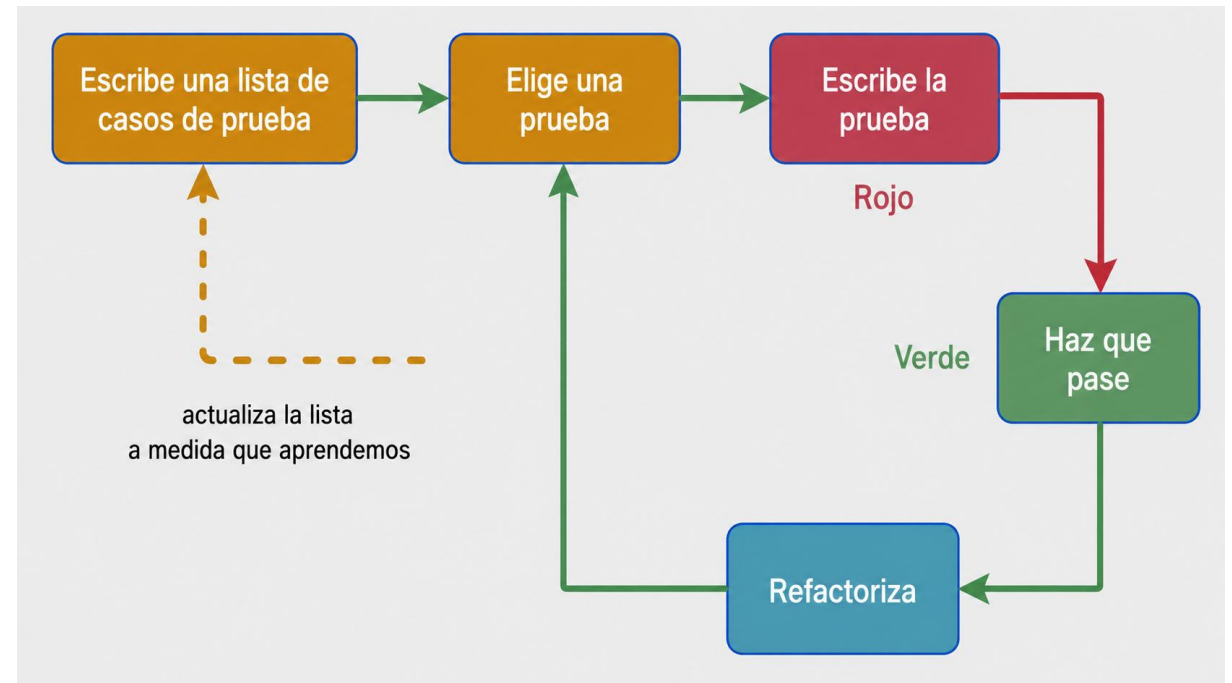
1. **Introducción**
 2. **¿Qué es una prueba software?**
 3. **Tipos y niveles de prueba**
 4. **Diseño de casos de prueba**
 5. **Desarrollo guiado por pruebas**
 6. **¿Y cómo afecta la IA a todo esto?**
 7. **Reflexión final**
- 

5. Desarrollo guiado por pruebas

TDD: Test Driven Development

Metodología de desarrollo de software en la que las pruebas unitarias se escriben antes del código de la funcionalidad que se quiere implementar. El ciclo básico de TDD sigue tres pasos:

- 1. Escribir una prueba fallida:** escribir una prueba para una funcionalidad que aún no está implementada.
- 2. Escribir el código mínimo:** escribir el código necesario para pasar la prueba.
- 3. Refactorizar:** mejorar el código manteniendo las pruebas exitosas.



<https://martinfowler.com/bliki/TestDrivenDevelopment.html>

5. Desarrollo guiado por pruebas | evidencia científica y debate abierto en la industria

What Do We (Really) Know about Test-Driven Development?

Itir Karac and Burak Turhan

JULY/AUGUST 2018 | **IEEE SOFTWARE**

Los efectos positivos que TDD puede tener sobre la calidad del software quizá no se deban solo a escribir las pruebas antes, sino a otros elementos del proceso:

- trabajar en tareas pequeñas y bien definidas
- mantener ciclos de desarrollo cortos y estables
- escribir solo el código necesario para superar la siguiente prueba
- refactorizar con frecuencia

Is TDD Dead?

A series of conversations between Kent Beck, David Heinemeier Hansson, and myself on the topic of Test-Driven Development (TDD) and its impact upon software design.

El objetivo siempre debería ser tener un código funcional, bien diseñado y acompañado de pruebas que permitan mantenerlo con confianza, independientemente de la técnica utilizada para lograrlo.

1. **Introducción**
 2. **¿Qué es una prueba software?**
 3. **Tipos y niveles de prueba**
 4. **Diseño de casos de prueba**
 5. **Desarrollo guiado por pruebas**
 6. **¿Y cómo afecta la IA a todo esto?**
 7. **Reflexión final**
- 

6. ¿Y cómo afecta la IA a todo esto?



Using LLMs to Bridge the Gaps in QA Test Plans at Firefox

JAN. 16, 2026 | GENERATIVE AI / OPEN SOURCE

The study explores using LLMs to automatically generate test plans for Firefox features, aiming to reduce the manual effort and blind spots in traditional QA planning.

Engineering at Meta

POSTED ON SEPTEMBER 30, 2025 TO AI RESEARCH, ML APPLICATIONS, SECURITY & PRIVACY

LLMs Are the Key to Mutation Testing and Better Compliance

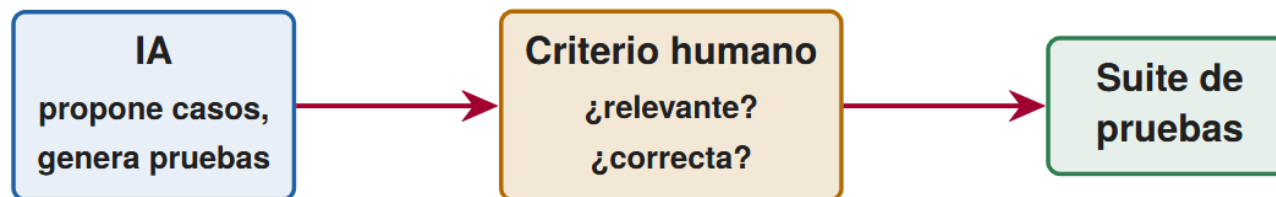
By leveraging LLMs we've been able to overcome the barriers that have prevented mutation testing from being efficiently deployed at scale. This allows us to greatly simplify risk assessments, reduce cognitive load for developers, and, ultimately, create a safer online ecosystem by enabling continuous compliance.



APRIL 28, 2026

Orchestrating AI Code Review at scale

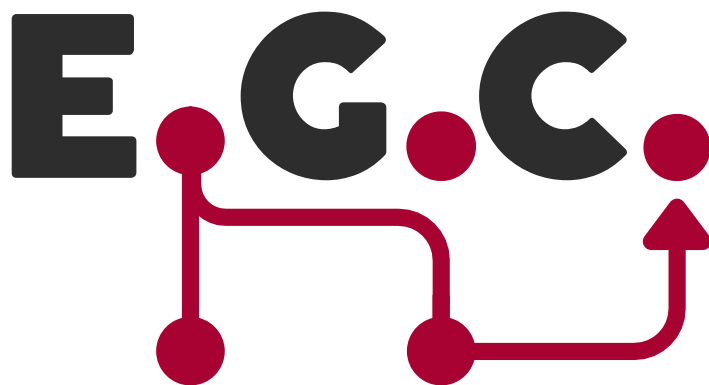
Instead of building a monolithic code review agent from scratch, we decided to build a [CI-native](#) orchestration system around [OpenCode](#), an open-source coding agent. Today, when an engineer at Cloudflare opens a merge request, it gets an initial pass from a coordinated smörgåsbord of AI agents. Rather than relying on one model with a massive, generic prompt, we launch up to seven specialised reviewers covering security, performance, code quality, documentation, release management, and compliance with our internal Engineering Codex. These specialists are managed by a coordinator agent that deduplicates their findings, judges the actual severity of the issues, and posts a single structured review comment.



1. **Introducción**
 2. **¿Qué es una prueba software?**
 3. **Tipos y niveles de prueba**
 4. **Diseño de casos de prueba**
 5. **Desarrollo guiado por pruebas**
 6. **¿Y cómo afecta la IA a todo esto?**
 7. **Reflexión final**
- 

7. Reflexión final

- **Una suite de pruebas es una herramienta de gestión del riesgo**
 - Aumentan nuestra confianza
 - Revelan fallos lo antes posible
 - Nos permiten cambiar el código sin miedo
- **Automatización y criterio humano**
 - Podemos automatizar la ejecución, la comparación de resultados y la integración con el pipeline
 - Pero el diseño sigue requiriendo criterio humano.
- **¿Qué podemos hacer en el proyecto?**
 - Identificar cómo nuestros proyectos hacen pruebas o proponer cómo se podrían hacer
 - Observar qué técnicas de casos de prueba se usan o proponer alguna
 - Observar qué herramientas se usan para pruebas tanto funcionales como no funcionales



Grado en Ingeniería Informática – Ingeniería del Software

Evolución y Gestión de la Configuración



Escuela Técnica Superior de
Ingeniería Informática

¡Gracias!