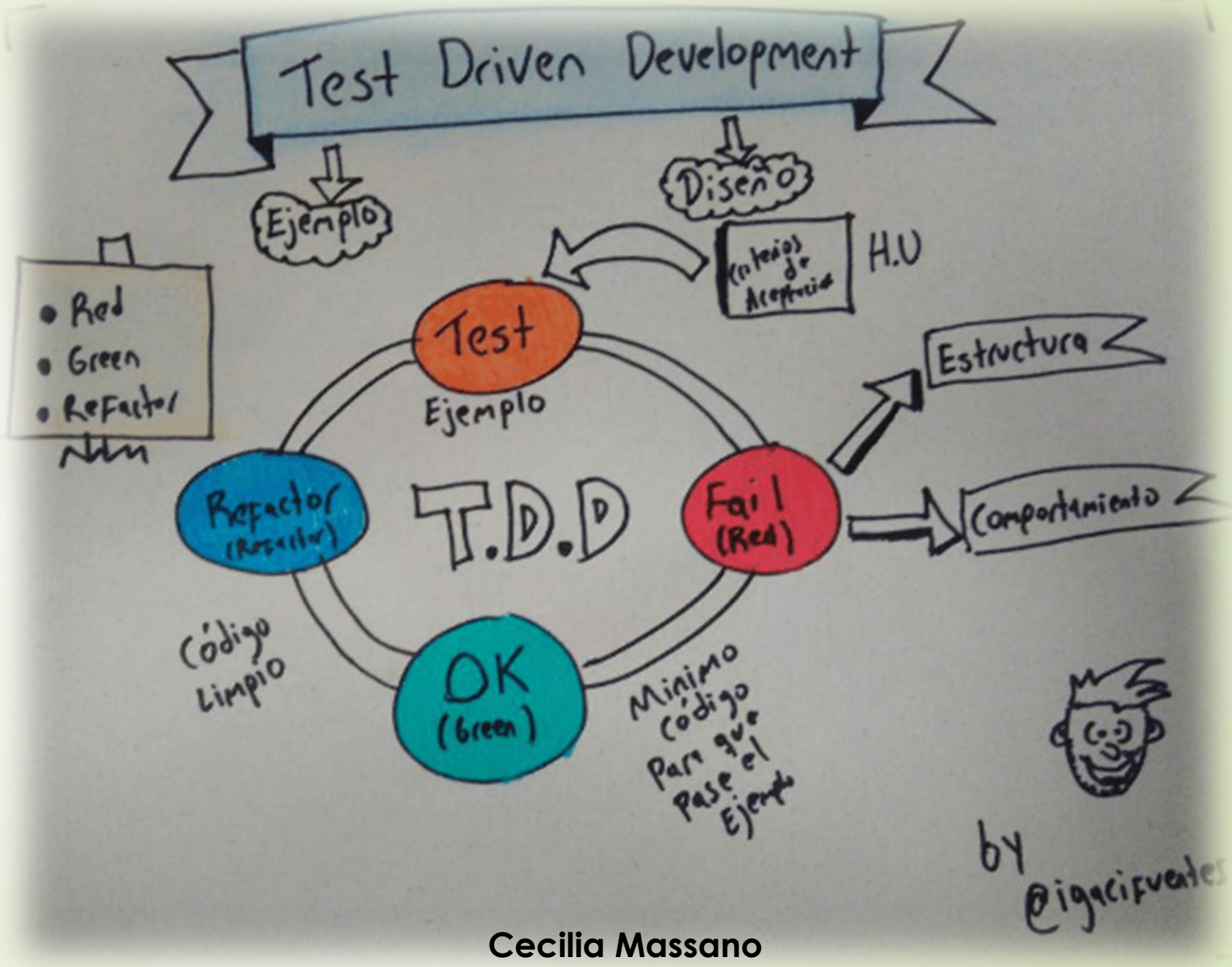
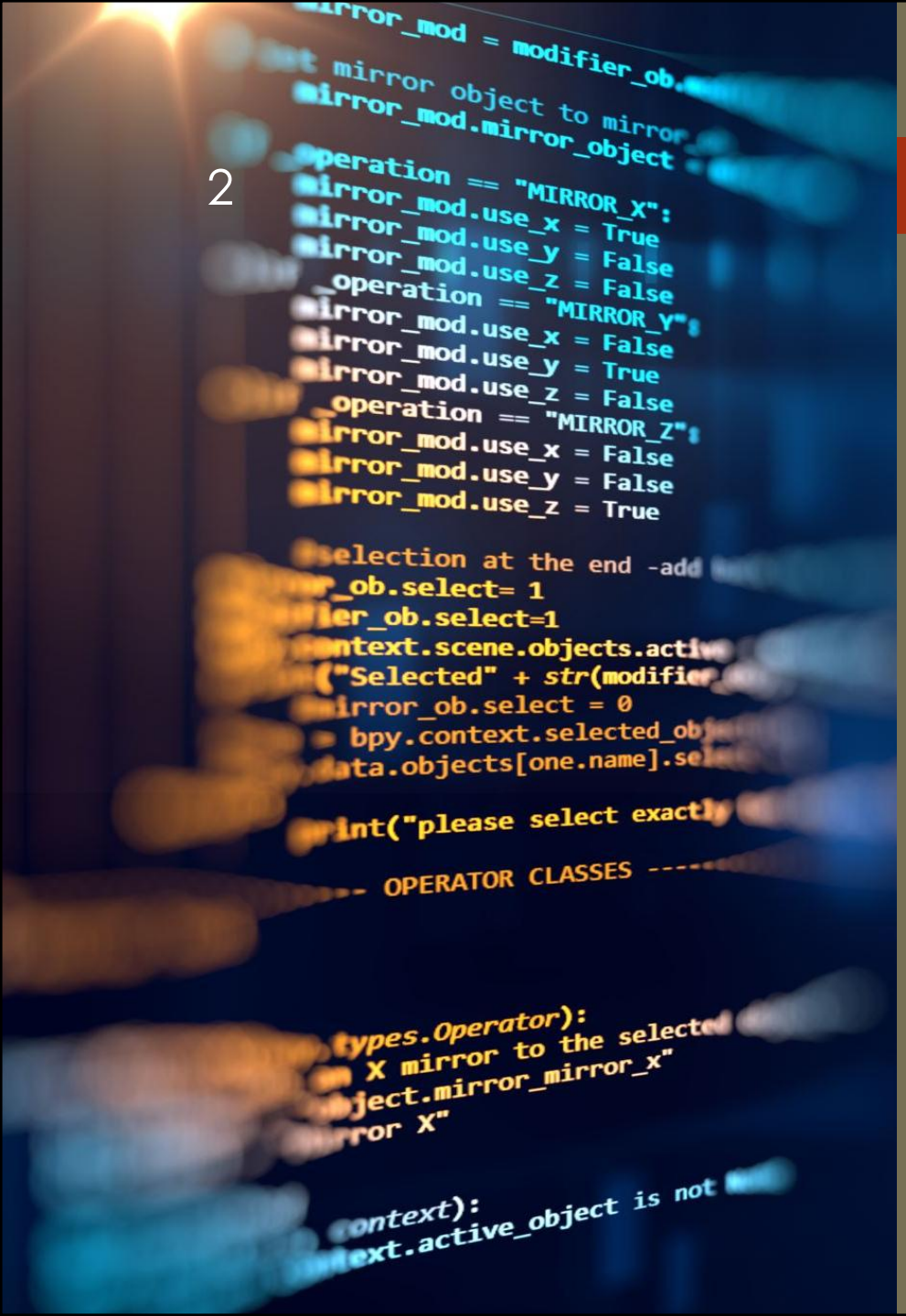
Universidad Tecnológica Nacional
Facultad Regional Córdoba
Cátedra de Ingeniería de Software
Docentes: Judith Meles & Laura Covaro

Test Driven Development (TDD)



Cecilia Massano



TDD

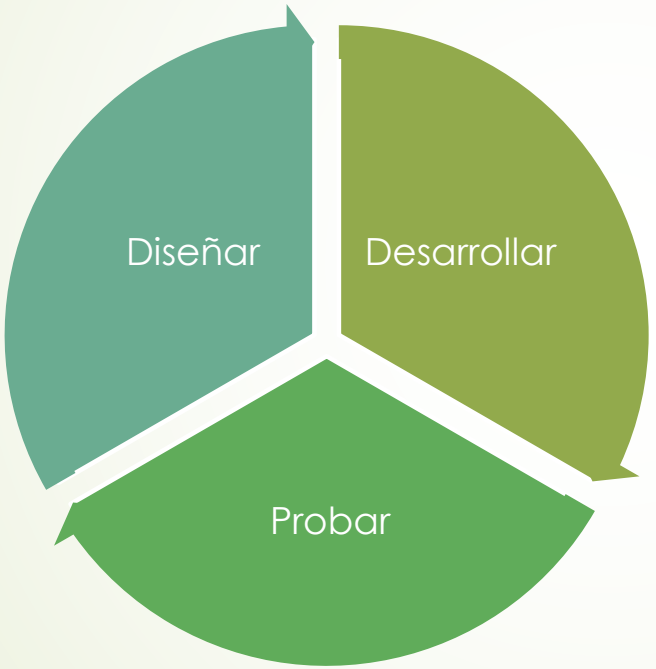
"El acto de diseñar tests es uno de los mecanismos conocidos más efectivos para prevenir errores…El proceso mental que debe desarrollarse para crear tests útiles puede descubrir y eliminar problemas en todas las etapas del desarrollo"

B. Beizer

"Test-Driven Development": Kent Beck. XP

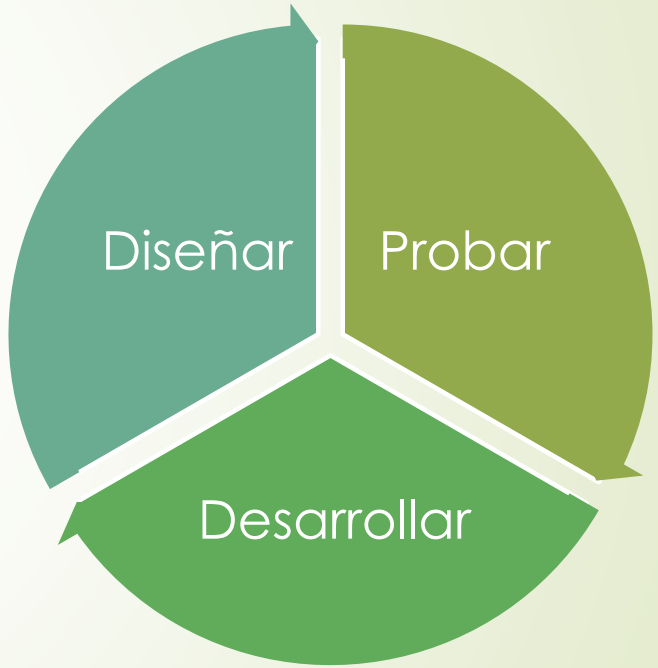
Desarrollo tradicional vs TDD

Desarrollo tradicional



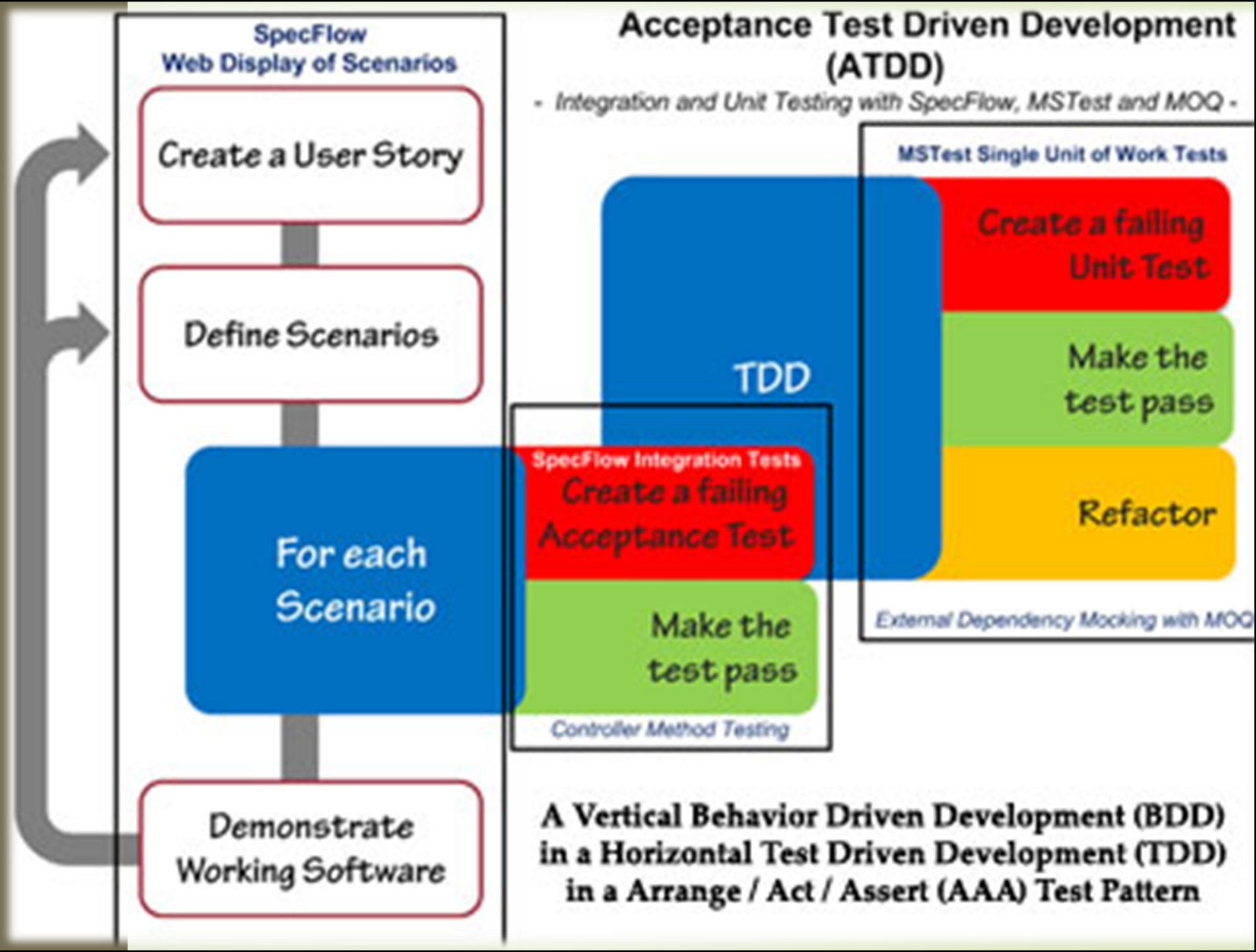
- Los desarrolladores diseñas las pruebas unitarias **después** de escribir el código
- El aseguramiento y control de calidad es más reactivo

TDD



- Los desarrolladores diseñas las pruebas unitarias **antes** de escribir el código
- El aseguramiento y control de calidad es mas proactivo

Cómo ver el **desarrollo de software ágil** con **TDD**



TDD

Desarrollo guiado por pruebas de software, o Test-driven development (TDD)

Es una técnica avanzada que involucra otras dos prácticas: *Escribir las pruebas primero (Test First Development)* y *Refactorización (Refactoring)*.

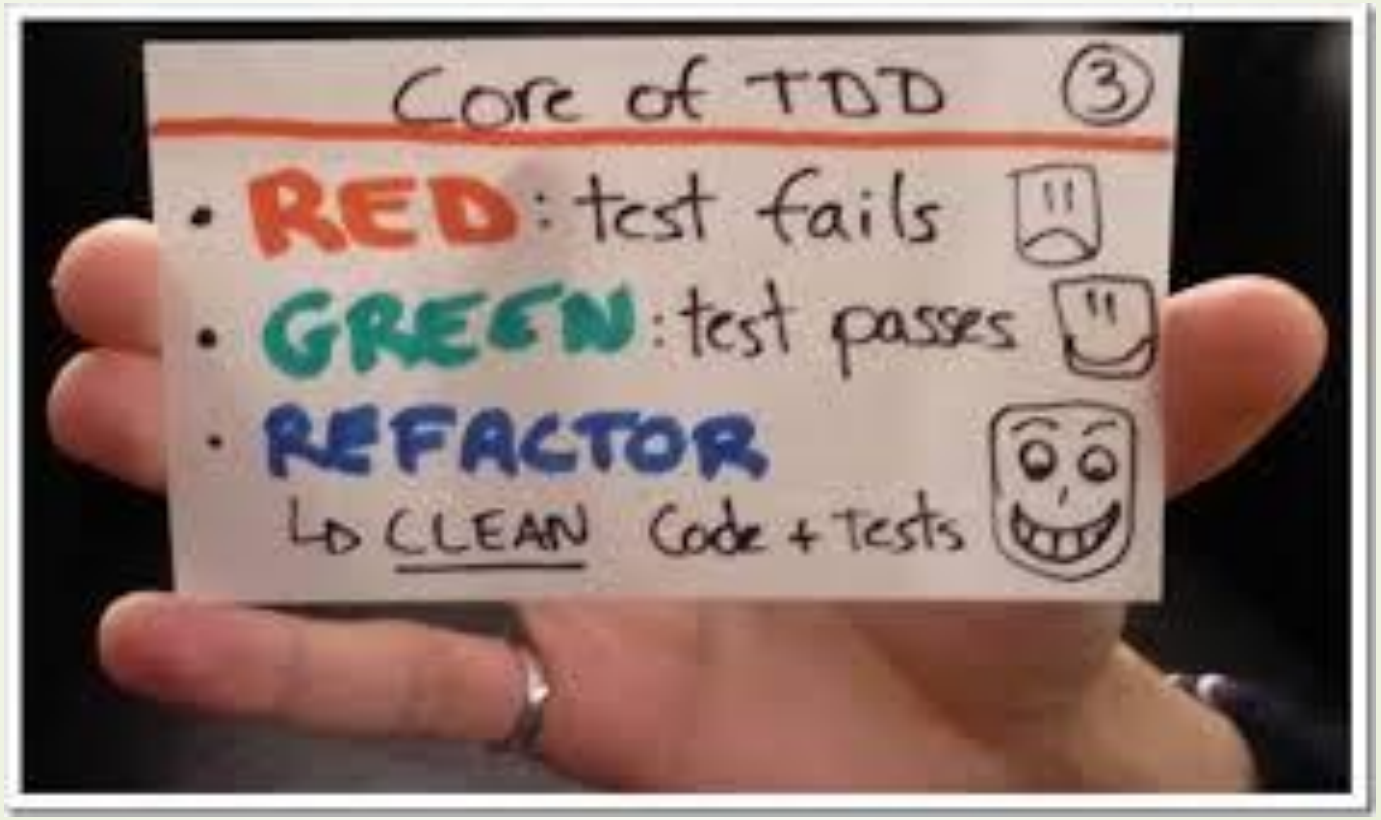
Para escribir las pruebas generalmente se utilizan las pruebas unitarias

Beneficios

- Código **robusto**
  - Código más predecible y limpio, que permite saber **cuándo el código está terminado**
  - Código más **tolerable al cambio**, al tener escritas las pruebas es más fácil de entender
  - Código más **seguro**
  - Aumenta la posibilidad de **reducir la duplicación de código** (si el test pasa el código ya existe)
- Código más **barato** de mantener, al ser más entendible
- Con la práctica **acelera** la velocidad de **desarrollo** y la posibilidad de **despliegue continuo**
- Asegura **cobertura de prueba**, lo que conlleva a código de calidad
- Obliga a pensar en cómo usar el componente, por lo tanto, el **diseño de las API**
- Promueve el desarrollo de componentes **con alta cohesión y bajo acoplamiento**

El Corazón de TDD

El ciclo Red-Green-Refactor



Las 3 leyes de TDD

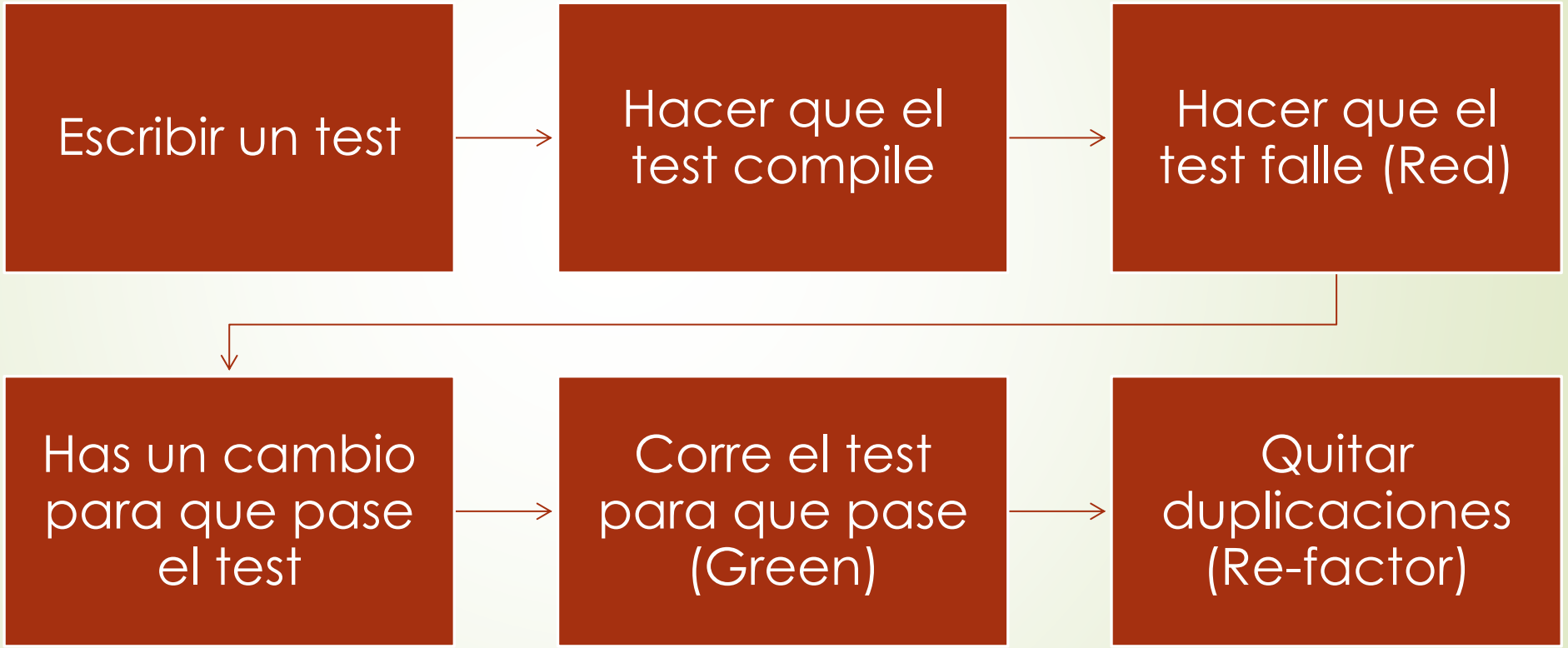
Robert C. Martin

No escribir una línea de código hasta que no hayas escrito antes un test case que falla

No es necesario escribir más test de los necesarios para que falle. Normalmente con un test que falle es suficiente.

No vas a escribir más código en Producción que el necesario para que el test pase.

Pasos del Desarrollo conducido por Testing (TDD)



Patrones de TDD

Patrones *red bar*

- One Step Test: Escribe un solo test que verifique una pieza mínima de funcionalidad.
- Starter Test: Primer test que te ayuda a explorar la API o comportamiento deseado.
- Explanation test: Escribe comentarios o preguntas para aclarar qué esperas que haga el código
- Learning test: permite conocer APIs externas sobre las que no sabemos su comportamiento.
- Another Test: Añade un nuevo test para cubrir un caso adicional o una variante del comportamiento.
- Regression Test: Agrega un test que reproduce un bug descubierto, para evitar regresiones futuras.

Patrones *green bar*

- Fake it: Devuelve un valor fijo o solución "falsa" solo para hacer pasar el test.
- Triangulación: Introduces un **segundo test con diferentes datos** para forzar una implementación más general
- Obvious implementation: Cuando la solución es trivial y obvia, puedes implementarla directamente sin "Fake It".
- One to many: Patrón para resolver test para colecciones de elementos.

Patrones de TDD

Patrones de TDD

- Test: para nombrar las pruebas con la palabra test y la clase a probar.
- Isolated test: mantener los test totalmente independientes
- Test List: primero escribe la lista de tests que vas a realizar
- Test first: escribe primero la prueba
- Assert test: primero escribir la confirmación y luego la prueba
- Test data: usa datos para el testing que sean fáciles de leer y seguir.
- Evident data: Incluye los resultados esperados y reales dentro del test, e intenta que la relación entre los distintos datos sea evidente.

Otros patrones

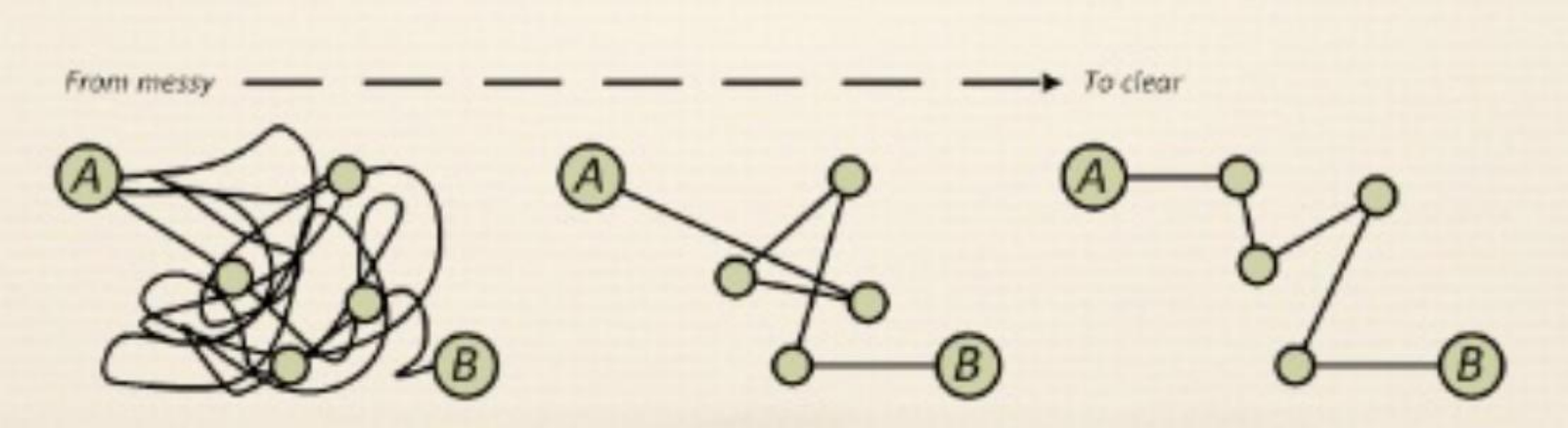
- Patrones de testing
- Patrones de diseño
- xUnit Patterns

Refactoring

- Es una forma disciplinada de introducir cambios reduciendo la posibilidad de introducir defectos.
- Es una transformación del software que:
  - Preserva la estructura externa
  - Mejora la estructura interna

Son cambios que se realizan en el sistema que:

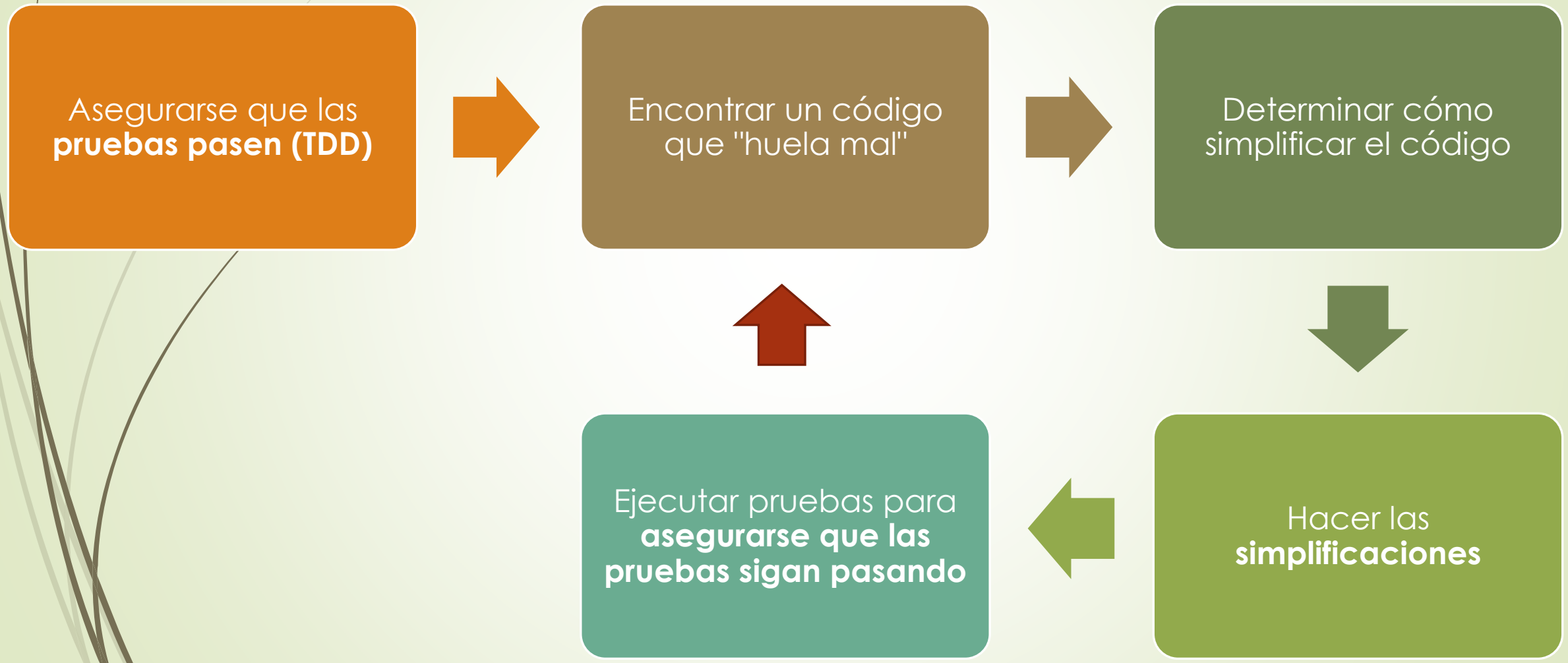
- No cambien el comportamiento observable (todas las pruebas aún pasan)
- **Eliminan la duplicación o la complejidad innecesaria**
- **Mejoran la calidad del software**
- Hacen que el código sea más simple y más fácil de entender
- Flexibiliza el código
- Hace que el código sea más fácil de cambiar



¿Por qué es necesario hacer Refactoring?

- Evitar el "deterioro del diseño"
- Limpiar desorden en el código
- Simplificar el código
- Incrementar la legibilidad y la comprensibilidad
- Encontrar errores
- Reducir el tiempo de depuración
- Incorporar el aprendizaje que hacemos sobre la aplicación
- Rehacer las cosas es fundamental en todo proceso creativo

¿Cómo hacer Refactoring?



En resumen


¿QUÉ ES TDD?
SE COMBINAN 2 METODOLOGÍAS
1. Test-first development: Escribir las pruebas primero.
2. Refactoring: Refactozación de código.
1 La prueba debe fallar: Se muestran los fallos en rojo.
2 Escribir el código mínimo para que la prueba pase
3 La prueba debe pasar: Las que pasan se muestran en verde.
4 Refactoring: Se debe mejorar el código.
TDD (Test Drive Development) es una metodología de desarrollo, cuyo objetivo es crear primero las pruebas y luego escribir el software.
Mejoran el código en cualquier momento.
La lectura del código será mucho mejor al tener ejemplos de uso (las pruebas).
Los equipos de testing, development y analyst serán más felices y eficientes.

Fuentes y Bibliografía

- Test Driven Development by example. Ken Beck
- Clean Code. Robert C. Martin
- Artículo Embracing Test Driven Development. Brad Huett

- Material introductorio para consultar: https://www.youtube.com/watch?v=1gttkO9JKtU