Ministerio de Producción
Presidencia de la Nación

Ministerio de Educación y Deportes

Subsecretaría de Servicios Tecnológicos y Productivos

Programa 111 mil
VOS PODÉS SER UNO.

DESARROLLO DE SOFTWARE TEST-DRIVEN DEVELOPMENT

Ministerio de Educación y Deportes
Presidencia de la Nación

Ministerio de Producción
Presidencia de la Nación

TDD: Contexto

*Es una técnica de diseño e implementación de software, que está basada en **pruebas unitarias** y **refactorización***

Originalmente propuesta dentro de

- **XP – Extreme Programming**
- Está orientado a obtener la
  - satisfacción del cliente
  - Asume (y permite enfrentarse a) cambios
  - de requerimientos y tecnología
  - Enfatiza el trabajo en equipo
  - Se prueba el producto (SW) desde el primer día de trabajo

- **TDD**: cómo lo hago? Por donde empiezo? Qué es lo que hay que implementar y lo que no?

TDD – Test Driven Development

- Se centra en 3 pilares fundamentales:
  - La implementación de las **funciones justas** que el cliente necesita y no más
  - La minimización del número de defectos que llegan al software en fase de producción
  - La producción de software modular, reutilizable y preparado para el cambio

Intenta eliminar la ambigüedad
de los requerimientos en lenguaje natural
↦ traducir el caso de uso o tarea
- **a varios ejemplos (pruebas)**
- No exime de revisiones de código ni de consultas a desarrolladores expertos

Ministerio de Educación y Deportes Presidencia de la Nación
Ministerio de Producción Presidencia de la Nación

Razones para usar TDD (en XP) [K. Beck]

- La calidad del software mejora
- Se obtiene código reutilizable
- El trabajo en equipo se hace más fácil
- Permite confiar en compañeros
  - (aun con menos experiencia)
  - Mejora la comunicación entre los
  - miembros del equipo

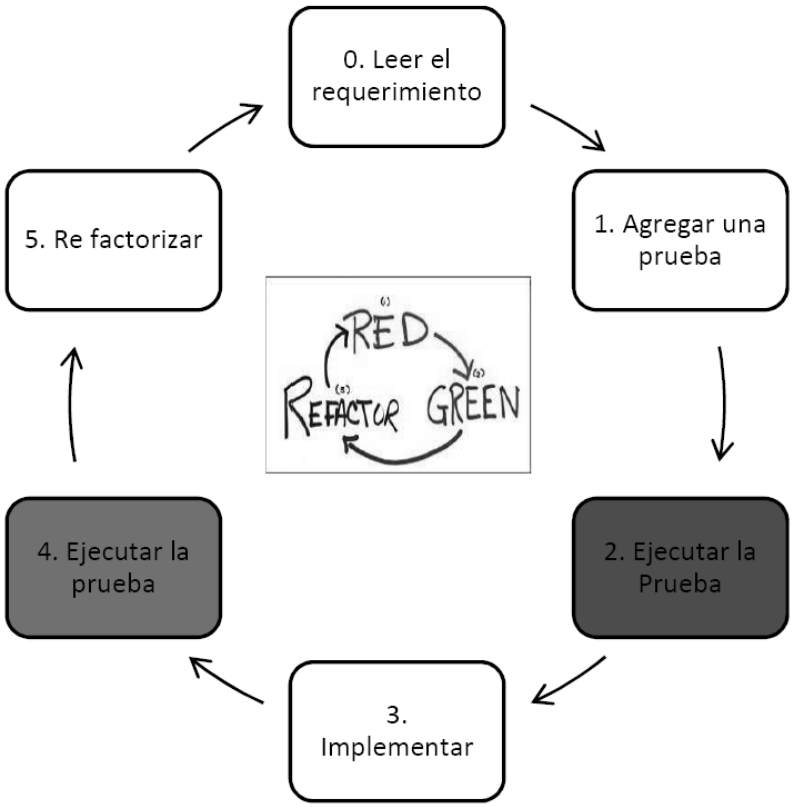
- Las personas responsables de QA adquieren un rol mas inteligente e interesante
- Escribir el ejemplo (test) antes que el código, obliga a escribir el mínimo de funcionalidad necesaria, evitando "sobre-diseñar" **↦ YAGNI**
- Incrementa la productividad
- Los tests contribuyen a la documentación técnica

Pasos de TDD

Escribir la especificación del requisito
- (ejemplo = test)

- Implementar el código según dicho ejemplo

- Re-factorizar el código, para eliminar duplicidad y hacer mejoras



Ministerio de Educación y Deportes Presidencia de la Nación
Ministerio de Producción Presidencia de la Nación

Escribir la especificación primero

El cliente pide un **requisito**, y debemos acordar
- un **criterio de aceptación**

- Partiendo de ese criterio, se derive una **prueba concreta**
  - Imaginar cómo sería el código si ya estuviera implementado y cómo comprobar que, efectivamente, hace lo que se le pide que haga
  - Escribir varias pruebas (casos)
  - Puede utilizarse algún framework como JUnit (Java)

Ejemplo de una Calculadora que suma números

```
public void testSuma() {
   assertEquals(8, Calculadora.suma(3,5));
}
```

Implementar el código para el ejemplo

- Codificar lo **mínimo necesario para hacer funcionar el ejemplo**
  - Foco en que la prueba pase (clases y métodos)
  - El código resultante puede no ser “óptimo”
  - (el diseño puede mejorarse más tarde)
  - Los flujos condicionales/alternativos/complementarios
  - pueden tratarse al abordar otros ejemplos (en otras iteraciones)

Al principio da “error” porque el código (clase Calduladora) no existe

… luego

```
public class Calculadora {
  public static int suma (int a, int b) {
     int c = a + b;
     return c;
  }
}
```

Ministerio de Educación y Deportes Presidencia de la Nación
Ministerio de Producción Presidencia de la Nación

Refactorizar

- Implica **modificar el diseño** (estructuralmente), pero **sin alterar su comportamiento** (funcionalmente)
  - Se analiza el código generado en fases previas de TDD
  - Por ejemplo: buscar líneas duplicadas, cumplimiento de buenas prácticas y principios de desarrollo, claridad del código, etc.
  - Por cada cambio, **se (re-)ejecutan todos los tests**

Código de suma “refactorizado”

```
public class Calculadora {
  public static int suma (int a, int b) {
     return a+b;
  }
}
```

Sigue pasando el test: **assertEquals(8, Calculadora.suma(3,5)**

Razones para Utilizar TDD

- La calidad del software aumenta
- Se obtiene código altamente reutilizable
- El trabajo en equipo se hace más fácil, une a las personas
- Nos permite confiar en nuestros compañeros, aunque tengan menos experiencia
- Multiplica la comunicación entre los miembros del equipo
- Las personas encargadas del aseguramiento de calidad adquieren un rol más inteligente e interesante
- Escribir el ejemplo (test) antes que el código obliga a escribir el mínimo de funcionalidad necesaria, evitando sobre-diseñar
- Los tests son la mejor documentación técnica que podemos consultar a la hora de entender qué misión cumple cada pieza de software
- Incrementa la productividad