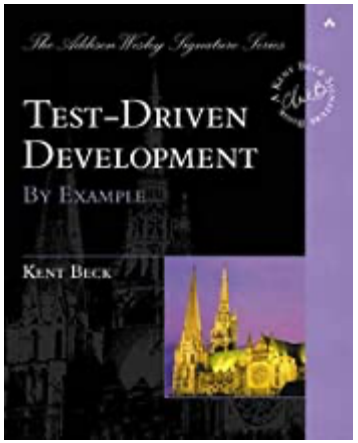
Test Driven Development

Dra. Alejandra Garrido
Objetos 2 – Fac. De Informática – U.N.L.P.
alejandra.garrido@lifia.info.unlp.edu.ar

TDD

- Método de desarrollo creado por Kent Beck en 2002



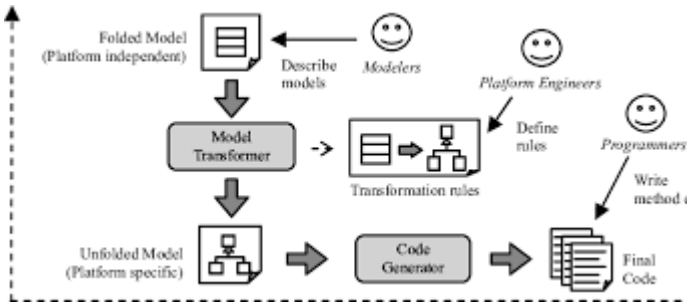
- Desarrollo dirigido por pruebas

Concepción del testing pre-TDD

- Ultimo paso del desarrollo
- Función reservada de un equipo separado de testing o QA (quality assurance)
- Equipo de QA escribe toda la documentación sobre la forma de testear cada función
- Luego de meses de testing, la lista de errores (bugs) vuelve a los programadores para ser corregidos

Qué pasaba en la IS en ese momento?

- Grupo fuerte de investigadores trabajando en MDD: Model Driven Development.
- Foco en el diseño y documentación



- Agile Manifesto firmado en 2001
- 1er libro del método Scrum publicado en 2001

Manifesto for Agile Software Development

We are uncovering better ways of developing
software by doing it and helping others do it.
Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on
the right, we value the items on the left more.

Surgen las metodologías ágiles

- Las metodologías ágiles o “lightweight” son:
  - adaptativas (como opuesto a predictivas)
  - orientadas a la gente (y no al proceso)

- Reconocen la gran diferencia entre el diseño y la construcción en la ingeniería “civil”, y el diseño y la construcción en software.

Características principales

Agile Approach
Mismo grupo de personas para todo el desarrollo que trabajan en un mismo espacio
Comunicación de calidad
Desarrollo iterativo e incremental
Producto funcionando en cada “build”
Se valora el feedback
Cambios bienvenidos “changes embraced”

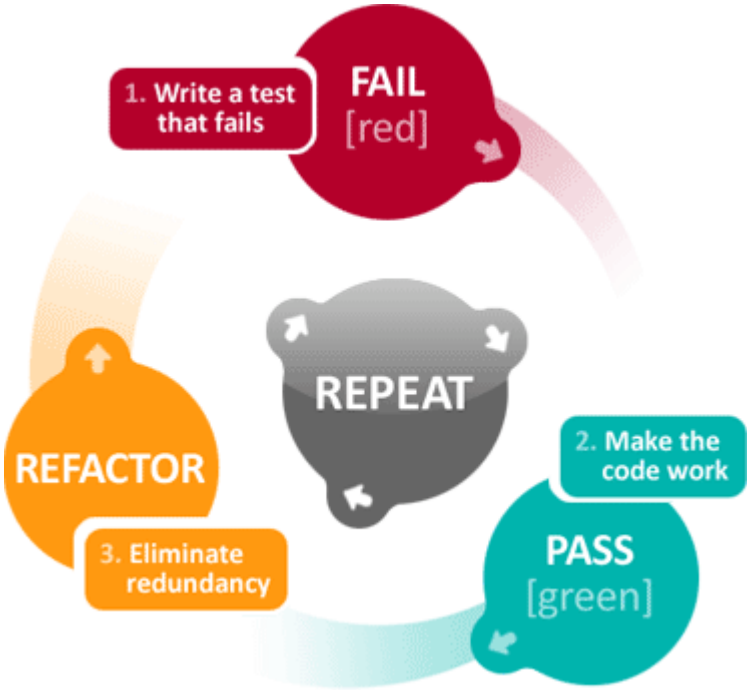
Por dónde empezar

- Si se debe ir tomando de a un requerimiento o pocos por vez para tener un producto funcionando al final de cada iteración, no cuento con un diseño completo para empezar a desarrollar

- Qué cosa guía el desarrollo? Por dónde empezar si no es de un diseño completo?

Test Driven Development

- Empezar por el test!



Test Driven Development (TDD)

- Combina:
  - *Test First Development*: escribir el test antes del código que haga pasar el test
  - *Refactoring*
- Objetivo:
  - pensar en el diseño y qué se espera de cada requerimiento antes de escribir código
  - escribir código limpio que funcione (como técnica de programación)

Granularidad

- Test de aceptación
  - Por cada funcionalidad esperada.
  - Escritos desde la perspectiva del cliente

- Test de unidad
  - aislar cada unidad de un programa y mostrar que funciona correctamente.
  - Escritos desde la perspectiva del programador

¿Por qué no dejar testing para el final?

- Para conocer cuál es el final (final feliz)
- Para mantener bajo control un proyecto con restricciones de tiempo ajustadas (permite estimar)
- Para poder refactorizar rápido y seguro
- Para darle confianza al desarrollador de que va por buen camino
- Como una medida de progreso

Filosofía de TDD

- Vuelco completo al desarrollo de software tradicional. En vez de escribir el código primero y luego los tests, se escriben los tests primero antes que el código.
- Se escriben tests funcionales para capturar use cases que se validan automáticamente
- Se escriben tests de unidad para enfocarse en pequeñas partes a la vez y aislar los errores

Filosofía de TDD (cont.)

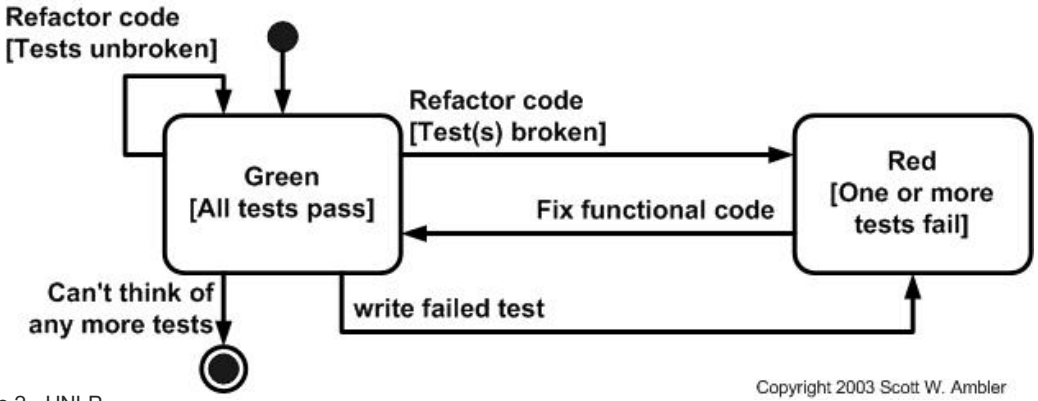
- No agregar funcionalidad hasta que no haya un test que no pasa porque esa funcionalidad no existe.
- Una vez escrito el test, se codifica lo necesario para que todo el test pase.
- Pequeños pasos: un test, un poco de código
- Una vez que los tests pasan, se refactoriza para asegurar que se mantenga una buena calidad en el código.

Algunas reglas de TDD

- Diseñar incrementalmente:
  - teniendo código que funciona como feedback para ayudar en las decisiones entre iteraciones.
- Los programadores escriben sus propios tests:
  - no es efectivo tener que esperar a otro que los escriba por ellos.
- El diseño debe consistir de componentes altamente cohesivos y desacoplados entre si:
  - mejora evolución y mantenimiento del sistema.

Automatización de TDD

- TDD asume la presencia de herramientas de testing (como las de la familia xUnit).
- Sin herramientas que automaticen el testing, TDD es prácticamente imposible.
- El ambiente de desarrollo debe proveer respuesta rápida ante cada cambio (build en 10 minutos).



¿Qué logramos con TDD?

- Diseño simple y limpio
- Desarrollar más rápido
- Saber cuando terminamos
- Confianza para el desarrollador
- Coraje para refactorizar
- Documentación práctica que evoluciona naturalmente
- Incrementar la calidad del software

Incrementar la calidad del software

- Mejorar la calidad del software, en dos aspectos:
  - que el software esté *construido correctamente*
  - que el software construido sea el *correcto*

Problemas y respuestas

- Unit testing infinito: por cada método público
- Test coupling: al estar los tests atados a la implementación

- “Test with a purpose” (Kent Beck)
- Saber por qué se testea algo y a qué nivel debe testearse.
- El objetivo de testear es encontrar bugs
- Testear tanto como sea el **riesgo** del artefacto

Reducir el riesgo del proyecto

- El riesgo puede reducirse de distintas formas:
  - reduciendo la cantidad de bugs;
  - previniendo la aparición de bugs;
  - acercando el SUT a las necesidades del usuario;
  - testeando el SUT bajo condiciones extremas

SUT: System Under Testing

Cuándo/Cómo/Por qué testear

- Se puede aplicar a cualquier artefacto del desarrollo
- Se debe testear temprano y frecuentemente
  - Es un error común olvidarse de ejecutar los tests de forma frecuente
- Un test vale más que la opinión de muchos

Bibliografia

- “Test Driven Development: by Example”. Kent Beck. Addison Wesley. 2002
- Introduction to Test Driven Development. Scott Ambler. http://agiledata.org/essays/tdd.html
- FitNesse: http://fitnesse.org/
  framework para acceptance testing a través de una wiki