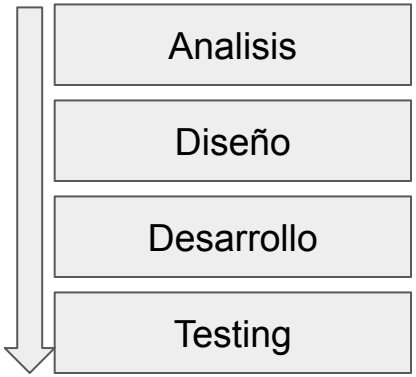
DEMOLITION FAIL FOR THE CLARION HOTEL AND CASINO

Test Driven Development

Dr. Federico Balaguer
Objetos 2 – Fac. De Informática – U.N.L.P.
federico.balaguer@lifia.info.unlp.edu.ar

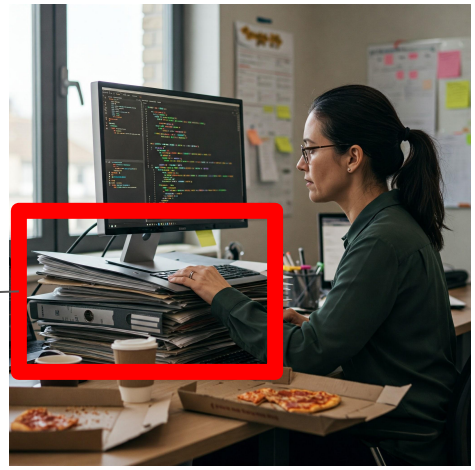


Analisis: entrevistas

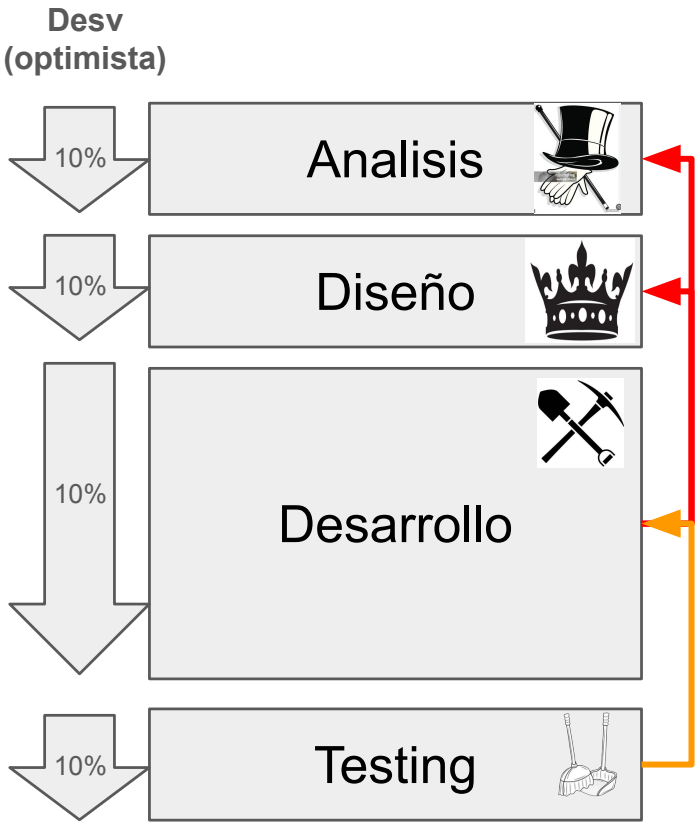
Diseño por Comité

Desarrollo

Documentos de Diseño



Testing tardío



- Costo de programación alto
- Costo de cambiar el código es muy alto
- ⇒ Pensar muy bien el desarrollo
- ⇒ Entender muy muy bien el problema

- Especialización de tareas:
  - Analisis
  - Diseño /Arquitecto
  - Desarrollador
  - Testing
- Mala comunicación (teléfono descompuesto)
  - Analistas conocen a los clientes
  - Analistas raramente hablan con Desarrolladores
- El tiempo corre y los requerimientos cambian
  - los programadores core estan disponibles?
- Testing encuentra errores…castigo para desarrolladores
- ¿Cuánto es el error acumulado?

El código sigue una arquitectura dada

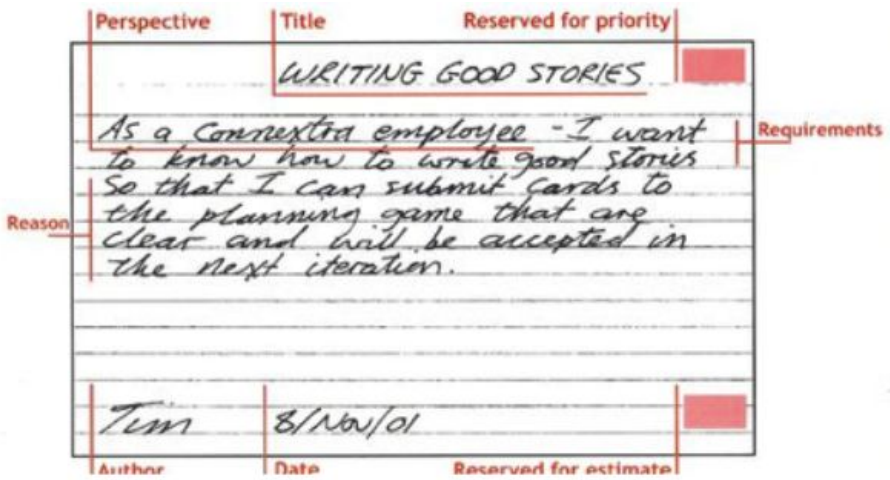
eXtreme (Programming) - Kent Beck

- ➔ Supongamos que hacer cambios tenga un costo despreciable
  - pequeños
  - incrementales
  - automáticos
  - preservan el comportamiento

⇒ **Es más fácil reparar/actualizar/evolucionar el código fuente**

eXtreme Programming

- ➔ Reducir el tiempo/proceso entre requerimiento a código fuente
- ➔ Backlog de historias
  - Historias creadas por un cliente
  - historia == tarjeta
  - Sprints de 1 o 2 semanas
- ➔ Tareas de programación
  - 1 o 2 días (máximo)
  - Pair Programing
  - TestCases como “contratos”



La Arquitectura emerge desde el código

Foco en el desarrollo (osea: el programador)

1. Uno como programador quiere
   - tener la más clara idea de lo que tiene que programar
   - algo que especifique cuánto falta para terminar y cuando terminé
   - que sea “objetivo” e “imparcial” (ejecutable)

⇒ Test cases ejecutables [1]

—

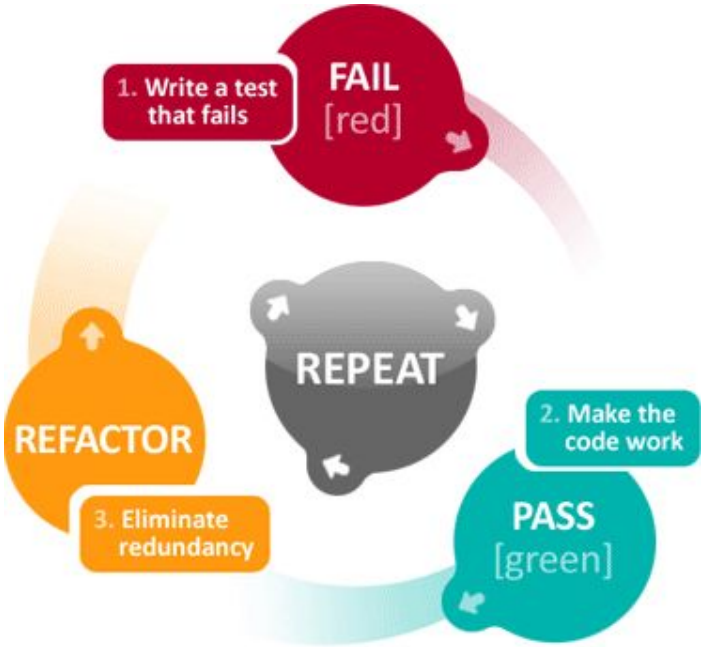
2. Entonces cada nueva historia (requerimiento)
   a. se expresa un contrato de desarrollo
      i. descripción
      ii. test cases que fallan (primero que nada)
   b. desarrollo
      i. refactoring (si es necesario)
      ii. programar código nuevo hasta que pase todos los test cases

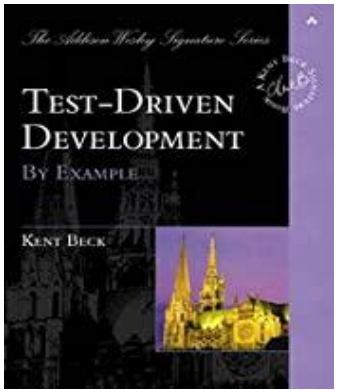
1. Kent Beck escribió el primer framework de testing de unidad SUnit en 1989

TDD = eXtreme(eXtreme(Programming))

Foco en el desarrollo (osea: el programador)

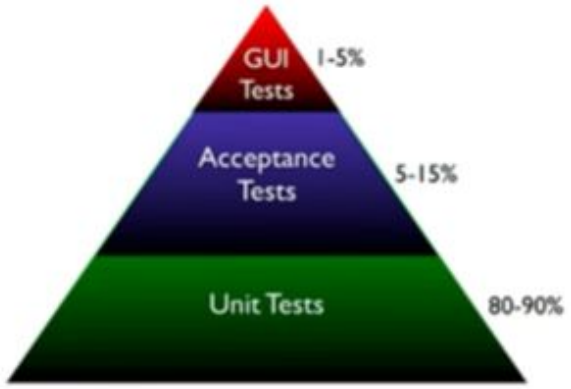
1. Red: Write a failing test for a new feature or requirement.
2. Green: Write just enough code to make the failing test pass.
3. Refactor: Improve the code while ensuring the tests still pass.
4. (Goto 1)Repeat: Continue with another new test case, repeating the cycle.





Test Driven Development (TDD)

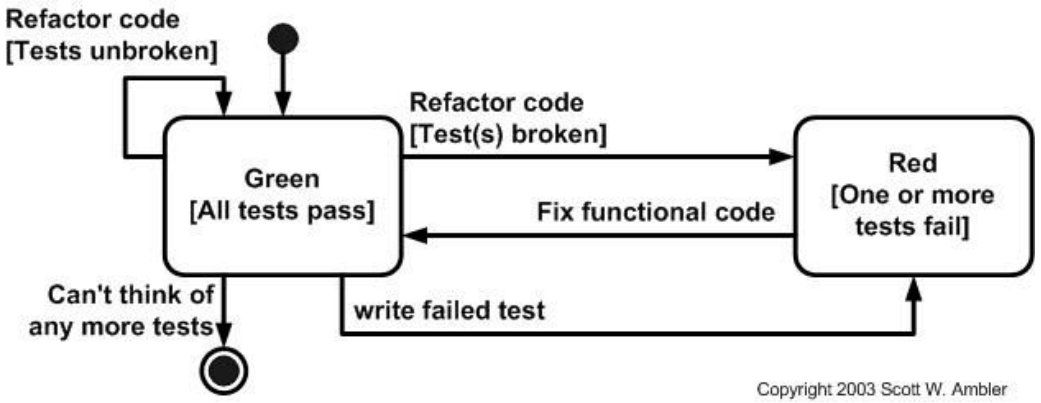
- Combina:
  - Test First Development: escribir el test antes del código que haga pasar el test
  - Refactoring
- Objetivo:
  - pensar en el diseño y qué se espera de cada requerimiento antes de escribir código
  - escribir código limpio que funcione (como técnica de programación)
- Granularidad de los Test
  - Test de unidad
    - aislar cada unidad de un programa y mostrar que funciona correctamente.
    - Escritos desde la perspectiva del programador
  - Test de aceptación
    - Por cada funcionalidad esperada.
    - Escritos desde la perspectiva del cliente



Filosofía de TDD

- Escribir 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
- 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.

Proceso de TDD



- Captura de Requerimientos
  - Historias cortas
  - Menor distancia entre Cliente y Programador
- Herramientas de testing (xUnit)
  - Testing Unidad
  - Test Regresión
  - Testing Integración (criterio de aceptación)
- Integración Continua

Cifrado: Rail Fence

Definiciones Previas

Sea **M** una cadena de caracteres (el mensaje a cifrar) de longitud L = |M|.

Sea **k** el número de rieles, donde k ∈ ℕ, k ≥ 2.

Sea **Z** la secuencia de posiciones generada por el patrón de zig-zag con k rieles.

1. Generar la secuencia de posiciones `Z`:

- Sea `P = [0, 1, 2, ..., k-2, k-1, k-2, ..., 1]` (una secuencia en zigzag de rieles)
- Repetir `P` hasta que su longitud sea al menos `L` → `Z = P * ceil(L / len(P))`
- Truncar `Z` a longitud `L`: `Z = Z[0:L]`

2. Construir los rieles:

- Crear `k` listas vacías: `R[0], R[1], ..., R[k-1]`
- Para cada índice `i ∈ [0, L-1]`:
  - Añadir `M[i]` a `R[Z[i]]`

¿Es fácil pensar una implementacion?
¿Cómo testearía algo así?

3. Concatenar los rieles:

- El mensaje cifrado `C` es:
  `C = R[0] + R[1] + ... + R[k-1]`
  (donde `+` denota concatenación de listas o cadenas)

Rail Fence (Historia)

Propósito: quiero ofuscar un mensaje para luego enviarlo por un medio público:
Quiero usar un cifrador Rail Fence parametrizable .
“WEAREDISCOVEREDFLEEATONCE” con 3 rieles:

```
W . . . E . . . C . . . R . . . L . . . T . . . E
. E . R . D . S . O . E . E . F . E . A . O . C .
. . A . . . I . . . V . . . D . . . E . . . N . .
```

“WECRLTEERDSOEEFEAOCAIVDEN”

—

Con 5 rieles:

w								c								l							e
	e						s		o						f		e					c	
		a				i				v				d				e				n		
			r		d						e		e						a		o			
				e								r								t				

“WCLEESOFECAIVDENRDEEAOERT”

PARA RESOLVER EN CLASE

Wrap up

Beneficios

- Historias de Usuario y Tareas más Pequeñas.
- Criterios de Aceptación Claros y Realizables
- Priorización Basada en el Valor de Negocio y el Riesgo
- Diseño Emergente
- Refinamiento Continuo
- Colaboración y Entendimiento Compartido

Requerimientos

- Comunicación y Colaboración Efectivas (que incluye al “cliente”)
- Backlog Refinado con Criterios de Aceptación Testeables
- Equipo con Cultura orientada al Testing
- Frameworks de Pruebas Unitarias Adecuados
- Herramientas de Cobertura de Código (Opcional Inicialmente)
- Integración Continua (CI)

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