Introducción a Refactoring

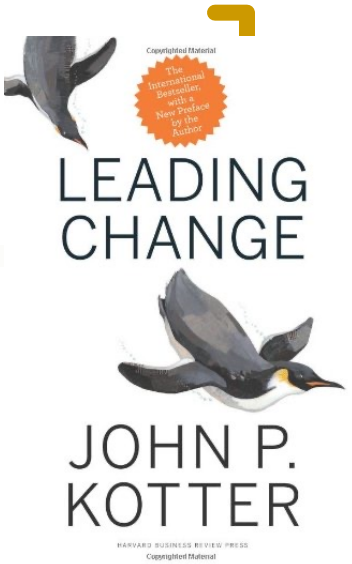
Dra. Alejandra Garrido
Objetos 2 – Fac. De Informática – U.N.L.P.
alejandra.garrido@lifia.info.unlp.edu.ar

Cambios

Leyes de Lehman

- Continuing Change (1974)
  - Los sistemas deben adaptarse continuamente o se vuelven progresivamente menos satisfactorios
- Continuing Growth (1991)
  - la funcionalidad de un sistema debe ser incrementada continuamente para mantener la satifacción del cliente
- Increasing Complexity (1974)
  - A medida que un sistema evoluciona su complejidad se incrementa a menos que se trabaje para evitarlo
- Declining Quality (1996)
  - La calidad de un sistema va a ir declinando a menos que se haga un mantenimiento riguroso

Prepararse para el cambio



- El ritmo del cambio en los negocios está creciendo **exponencialmente**
- Cambio exponencial implica tiempo de reacción **exponencialmente menor**
- La incapacidad de cambiar el software de manera rápida y segura implica que se pierden oportunidades de negocio

Costo del mantenimiento

- Mantenimiento
  - correctivo, evolutivo, adaptativo, perfectivo, preventivo.

- **Entender código existente:**
  50% del tiempo de mantenimiento

Lidiando con el código spaguetti

```
if (evt1.AbsoluteTime < evt2.AbsoluteTime) {
            return -1;
        } else if (evt1.AbsoluteTime > evt2.AbsoluteTime) {
            return 1;
        } else {
            // a iguar valor de AbsoluteTime, los channelEvent tienen prioridad
            if(evt1.MidiEvent is ChannelEvent && evt2.MidiEvent is MetaEvent) {
                return -1;
            } else if(evt1.MidiEvent is MetaEvent && evt2.MidiEvent is ChannelEvent){
                return 1;
            //      si ambos son channelEvent, dar prioridad a NoteOn == 0 sobre NoteOn > 0
            } else if(evt1.MidiEvent is ChannelEvent && evt2.MidiEvent is ChannelEvent) {

                chanEvt1 = (ChannelEvent) evt1.MidiEvent;
                chanEvt2 = (ChannelEvent) evt2.MidiEvent;

                // si ambos son NoteOn
                if(     chanEvt1.EventType == ChannelEventType.NoteOn
                        && chanEvt2.EventType == ChannelEventType.NoteOn){

                    //      chanEvt1 en NoteOn(0) y el 2 es NoteOn(>0)
                    if (chanEvt1.Arg1 == 0 && chanEvt2.Arg1 > 0) {
                        return -1;
                    //      chanEvt1 en NoteOn(0) y el 2 es NoteOn(>0)
                    } else if(chanEvt2.Arg1 == 0 && chanEvt1.Arg1 > 0) {
                        return 1;
                    } else {
                        return 0;
```

Big Balls of Mud

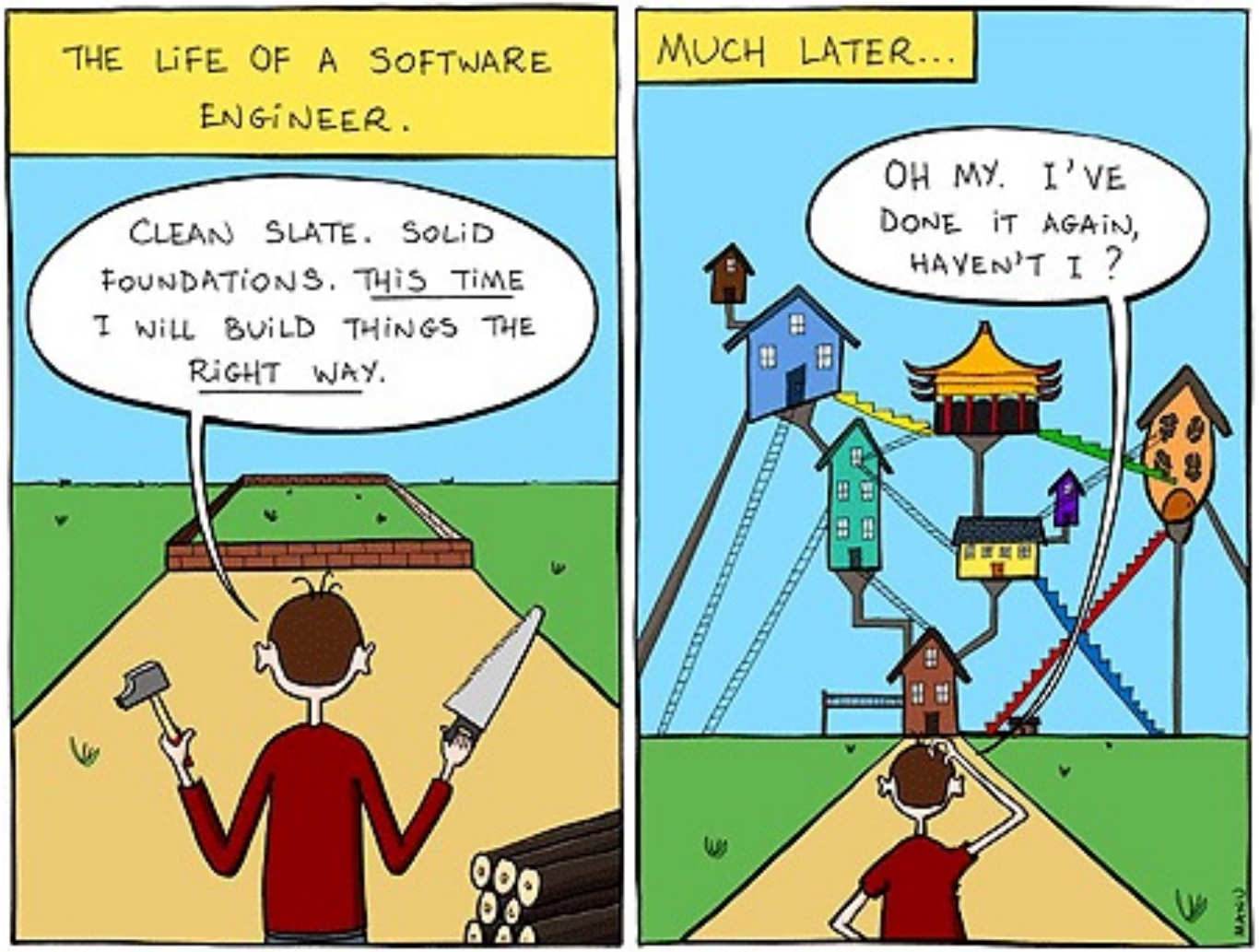
Brian Foote & Joe Yoder. 2000. http://www.laputan.org/mud/

Like a set of dominoes, the former home of the Atlanta Braves collapsed Saturday (Courtesy WXIA)

Diseñar es difícil!

- Los elementos distintivos de la arquitectura de un sistema no surgen hasta *después* de tener código que funciona
- No se trata sólo de agregar, sino de *adaptar*, *transformar, mejorar*
- Construir el sistema perfecto es imposible
- Los errores y el cambio son inevitables
- Hay que aprender del **feedback**

Somos humanos



¿Qué hacemos?

- Reconocer las fuerzas que llevan al deterioro de la arquitectura y aprender a reconocer oportunidades para mejorarla
- Darnos libertad para explorar
- Después limpiar! (pagar la deuda técnica)

La iteración es fundamental

- “Reusable software is the result of many design iterations. Some of these iterations occur after the software has been reused”

  (Bill Opdyke. 1992)

- Estudiando la evolución de grandes arquitecturas descubre que hay cambios de una iteración a la siguiente que no agregan ni alteran la funcionalidad, sino que son cambios estructurales que mejoran el diseño

Refactoring

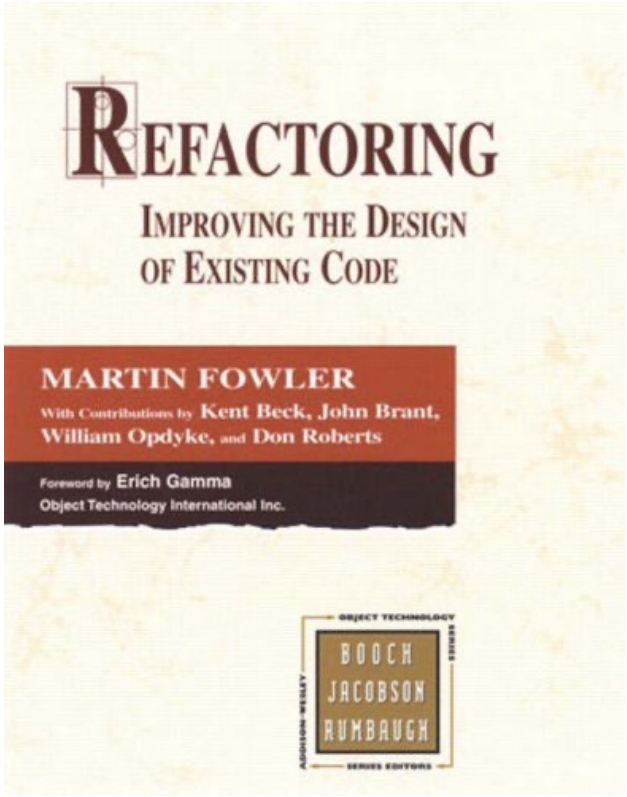
- Refactoring es una **transformación** que preserva el comportamiento, pero mejora el diseño

Bill Opdyke, PhD Thesis "Refactoring Object-Oriented Frameworks". Univ. of Illinois at Urbana-Champaign (UIUC). 1992. Director: Ralph Johnson.

Refactoring como un proceso

- Es el proceso a través del cual se cambia un sistema de software
  - para ***mejorar*** la organización, legibilidad, adaptabilidad y mantenibilidad del código luego que ha sido escrito
  - que ***NO altera*** el comportamiento externo del sistema

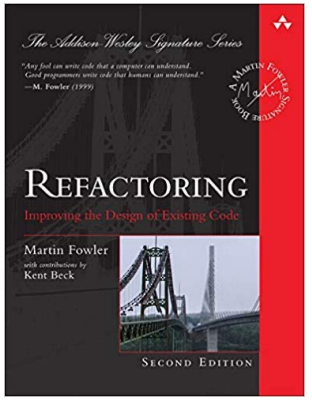
Refactoring by Martin Fowler



En qué contexto surge

- Refactoring. Fowler. 1999.
- Agile Manifesto: http://agilemanifesto.org/. 2001
- Agile Software Development with Scrum. Ken Schwaber, Mike Beedle. 2001
- Test Driven Development. Kent Beck. 2003. Basado en refactoring y testing como dos prácticas esenciales.

Refactoring by Fowler



- *Refactoring* (sustantivo): cada uno de los cambios catalogados
  - "A change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior".
  - Con un nombre específico y una **secuencia de pasos** ordenados ("mechanics")

- *Refactor* (verbo): el proceso de aplicar refactorings
  - "To restructure software by applying a series of refactorings without changing its observable behavior"

Características del Refactoring

- Implica
  - Eliminar duplicaciones
  - Simplificar lógicas complejas
  - Clarificar códigos
- Cuándo
  - Una vez que tengo código que funciona y pasa los tests
  - A medida que voy desarrollando:
    - cuando encuentro código difícil de entender (ugly code)
    - cuando tengo que hacer un cambio y necesito reorganizar primero
  - Antes de llegar a

- Testear después de cada cambio

Cómo ayuda el refactoring?

- Introduce mecanismos que solucionan problemas de diseño
- A través de cambios ***pequeños***
  - Hacer muchos cambios pequeños es más fácil y más seguro que un gran cambio
  - Cada pequeño cambio pone en evidencia otros cambios necesarios