UD 06.REFACTORIZACIÓN

v1.0 10.12.20

ceedcv
CENTRE ESPECÍFIC
D´EDUCACIÓ A DISTÀNCIA DE
LA COMUNITAT VALENCIANA

Entornos de desarrollo (ED)

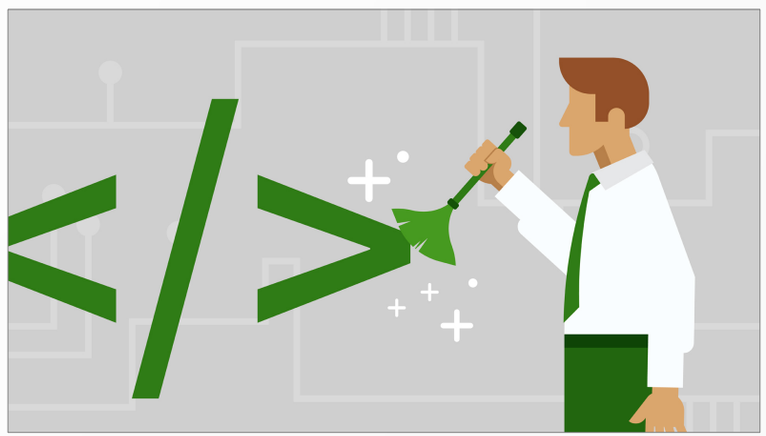
Sergio Badal
sergio.badal@ceedcv.es

Extraído de los apuntes de:
Cristina Álvarez Villanueva; Fco. Javier Valero Garzón; M.ª Carmen Safont

UD 06.REFACTORIZACIÓN

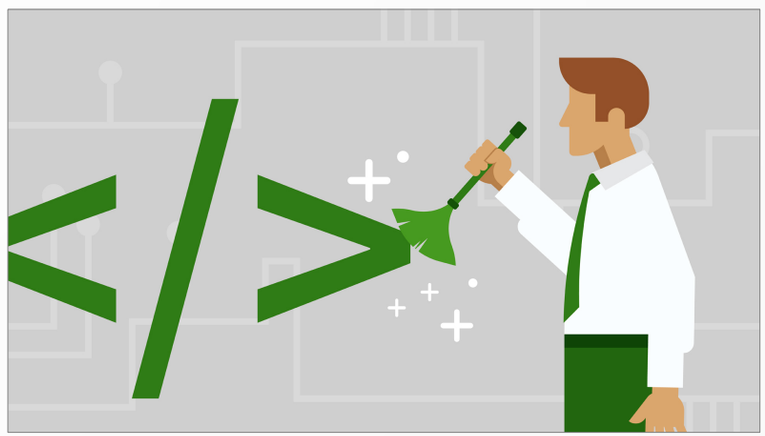
- **Pasos a seguir**

1) Lee la documentación (PDF)
2) Instala el software necesario (sigue los pasos)
3) Realiza los TESTS todas las veces que quieras
4) Acude al FORO DE LA UNIDAD
    - Para cualquier duda sobre esta unidad
5) Acude al FORO DEL MÓDULO
    - Para cualquier duda sobre el módulo



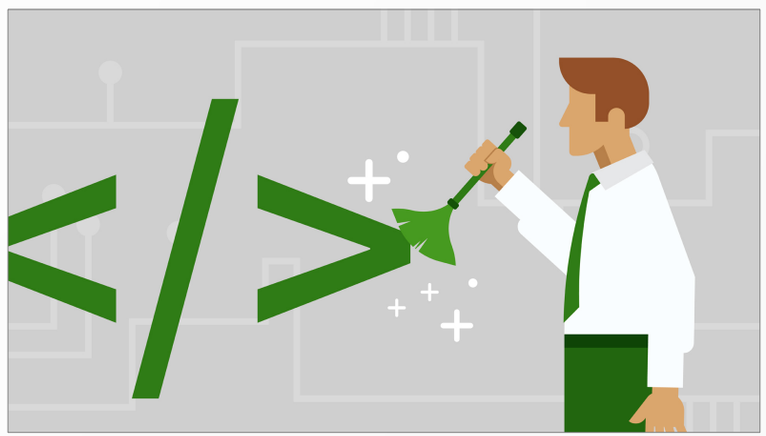
UD 06.REFACTORIZACIÓN

- **¿Qué veremos en esta UNIDAD?**
  - **SEMANA ÚNICA**
    - **REFACTORIZACIÓN**
    - **REFACTORIZACIÓN en NETBEANS/ECLIPSE**
      - NO HAY PRÁCTICA PARA ENTREGAR ESTA SEMANA



UD 06.REFACTORIZACIÓN

- **¿Qué veremos en esta UNIDAD?**
  - **SEMANA ÚNICA**
    - **REFACTORIZACIÓN**
    - **REFACTORIZACIÓN en NETBEANS/ECLIPSE**
      - NO HAY PRÁCTICA PARA ENTREGAR ESTA SEMANA



Finalmente, no habrá más prácticas evaluables pero una o varias preguntas de la parte de test del exámen de evaluación serán de esta unidad.

UD 06.REFACTORIZACIÓN

6 REFACTORIZACIÓN

6.1 ¿QUÉ ES REFACTORIZAR?

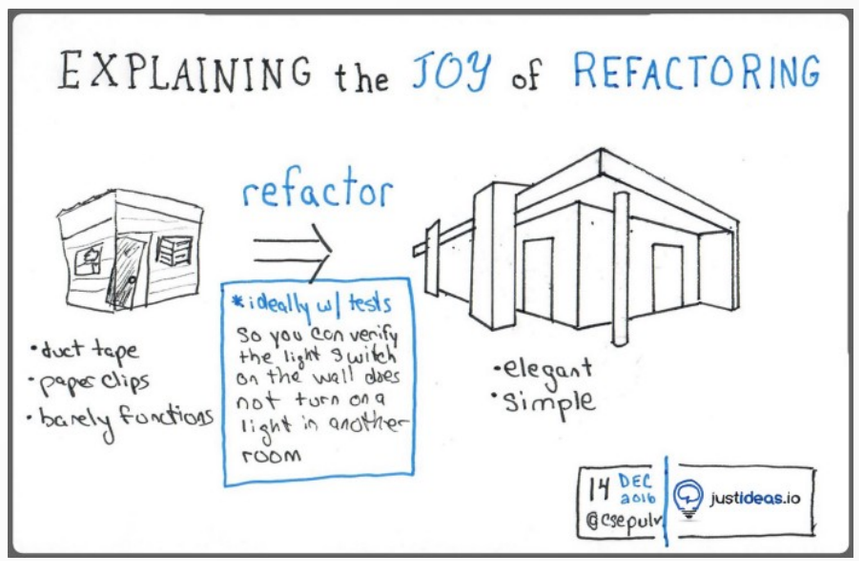
6.2 CUÁNDO REFACTORIZAR

6.3 TIPOS DE REFACTORIZACIÓN

6.4 IMPLEMENTACIÓN

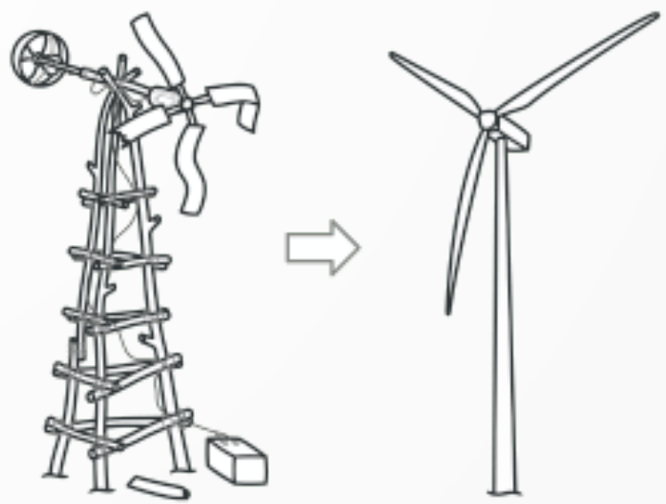
6.1 ¿QUÉ ES REFACTORIZAR?

- **Programar** código no únicamente es buscar funcionalidad, sino también **limpieza y elegancia de escritura**.
- Así, aunque debería irse programando ya con una estructura inicial y buscando la limpieza de código, una vez terminado siempre se pasa a la fase de Refactorización.
- En ella le damos “chapa y pintura” a nuestro código.
  - Es decir, cambiamos la estructura interna de nuestro software para hacerlo **más fácil de comprender, de modificar, más limpio** y sin cambiar su comportamiento observable.
- Por tanto, refactorizar es:
  - **Mejorar el código fuente sin cambiar el resultado del programa.**



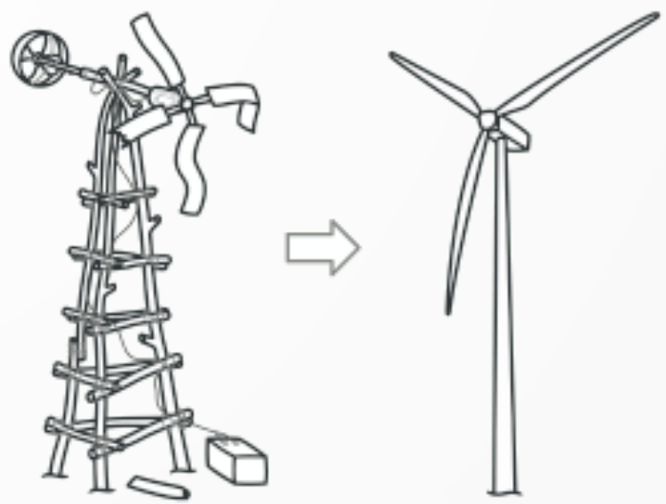
6.1 ¿QUÉ ES REFACTORIZAR?

- Refactorizamos por varios motivos:
  - Para **eliminar código duplicado**
    - Con una visisión global del código.
  - Para **hacerlo más legible/fácil de entender**
    - Renombrando identificadores, reorganizando parámetros, métodos, clases, paso de mensajes y dependencias.
  - Para **encontrar errores** al reorganizar un **programa (*)**



6.1 ¿QUÉ ES REFACTORIZAR?

- Refactorizamos por varios motivos:
  - Para **eliminar código duplicado**
    - Con una visisión global del código.
  - Para **hacerlo más legible/fácil de entender**
    - Renombrando identificadores, reorganizando parámetros, métodos, clases, paso de mensajes y dependencias.
  - Para **encontrar errores** al reorganizar un **programa (*)**



¿Recuerdas qué era un programa? ¿Y una aplicación?

6.1 ¿QUÉ ES REFACTORIZAR?

- ¿Qué se consigue?
  - mejorar el diseño del código
  - mejorar su legibilidad
  - mejora la productividad de los programadores
- En otras palabras:

1) **Calidad**. Código sencillo y bien estructurado, legible y entendible sin necesidad de haber estado integrado en el equipo de desarrollo durante varios meses.
2) **Eficiencia**. Mantener un buen diseño y un código estructurado es sin duda la forma más eficiente de desarrollar. El esfuerzo invertido en evitar la duplicación de código y en simplificar el diseño se ve compensado a la hora de las modificaciones
3) **Evitar la reescritura de código**. Refactorizar es mejor que reescribir.

(!) Refactoriza el autor del código. ¿En qué etapas?

(!) Reescribe quien da con un código no refactorizado. ¿En qué etapas?

6.1 ¿QUÉ ES REFACTORIZAR?

- ¿Qué se consigue?
  - mejorar el diseño del código
  - mejorar su legibilidad
  - mejora la productividad de los programadores
- En otras palabras:

1) **Calidad**. Código sencillo y bien estructurado, legible y entendible sin necesidad de haber estado integrado en el equipo de desarrollo durante varios meses.
2) **Eficiencia**. Mantener un buen diseño y un código estructurado es sin duda la forma más eficiente de desarrollar. El esfuerzo invertido en evitar la duplicación de código y en simplificar el diseño se ve compensado a la hora de las modificaciones
3) **Evitar la reescritura de código**. Refactorizar es mejor que reescribir.

(!) Refactoriza el autor del código. ¿En qué etapas?

=> Codificación / Pruebas / Mantenimiento (¿de qué tipo?)

(!) Reescribe quien da con un código no refactorizado. ¿En qué etapas?

=> Mantenimiento (¿de qué tipo?)

UD 06.REFACTORIZACIÓN

6 REFACTORIZACIÓN

6.1 ¿QUÉ ES REFACTORIZAR?

6.2 CUÁNDO REFACTORIZAR

6.3 TIPOS DE REFACTORIZACIÓN

6.4 IMPLEMENTACIÓN

6.2 CUÁNDO REFACTORIZAR

- ¿Recuerdas los tipos de mantenimiento?
  - Repasemos la UD1 …
    - … eran 3 …

      … y …

      … ¡acabamos de añadir uno más!
      -
      -
      -
      - **NUEVO: ¡ !**

6.2 CUÁNDO REFACTORIZAR

- ¿Recuerdas los tipos de mantenimiento?
  - Repasemos la UD1 …
    - … eran 3 …

      … y …

      … ¡acabamos de añadir uno más!
      - **EVOLUTIVO**
      - **ADAPTATIVO**
      - **CORRECTIVO**
      - **NUEVO: ¡PERCEPTIVO!**

6.2 CUÁNDO REFACTORIZAR

- ¿Recuerdas los tipos de mantenimiento?
  - Repasemos la UD1 …
    - … eran 3 …

      … y …

      … ¡acabamos de añadir uno más!
      - **EVOLUTIVO => mejoras**
      - **ADAPTATIVO => cambios legales/coyunturales**
      - **CORRECTIVO => solución de errores**
      - **NUEVO: ¡PERCEPTIVO! => refactorización a posteriori**

6.2 CUÁNDO REFACTORIZAR

- ¿Recuerdas los tipos de mantenimiento?
  - Repasemos la UD1 …
    - … eran 3 …

… y …

… ¡acabamos de añadir uno más!

- **EVOLUTIVO => mejoras**
- **ADAPTATIVO => cambios legales/coyunturales**
- **CORRECTIVO => solución de errores**
- **NUEVO: ¡PERCEPTIVO! => refactorización a posteriori**

El mantenimiento perceptivo es el único que suele realizar el autor de la fase de codificación. ¿Por qué?

-
-

6.2 CUÁNDO REFACTORIZAR

- ¿Recuerdas los tipos de mantenimiento?
  - Repasemos la UD1 …
    - … eran 3 …

      … y …

      … ¡acabamos de añadir uno más!
      - **EVOLUTIVO => mejoras**
      - **ADAPTATIVO => cambios legales/coyunturales**
      - **CORRECTIVO => solución de errores**
      - **NUEVO: ¡PERCEPTIVO! => refactorización a posteriori**

El mantenimiento perceptivo es el único que suele realizar el autor de la fase de codificación. ¿Por qué?

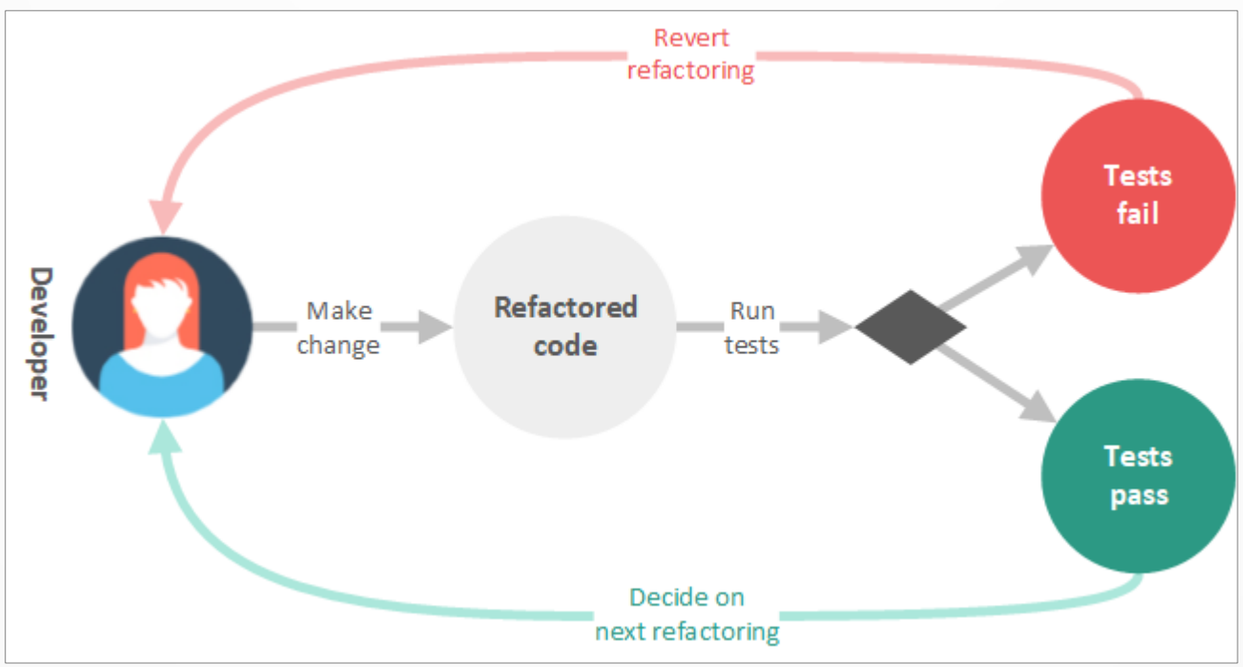
- *El propio autor del código es quien mejor lo conoce.*
- *El autor del código es quien realmente refactoriza. El resto, puede que refactorice con éxito pero gran parte del tiempo reescribirá.*

6.2 CUÁNDO REFACTORIZAR

- Pero…
  - ¡**No siempre esperamos a la fase de mantenimiento (perceptivo) para refactorizar**!
- ¿Cuándo debemos refactorizar?
  - Si vemos que el diseño se vuelve complicado y difícil de entender, y que **cada paso hacia adelante** empieza a ser muy costoso, lo mejor es parar de desarrollar/codificar y analizar si el rumbo tomado es correcto.
- Los síntomas que indican que algún código de software tiene problemas se conocen como **"Bad Smells"** y suelen ser ...
  - Código duplicado, métodos largos, clases largas, cláusulas switch innecesarias, comentarios, etc.
  - Se debe aplicar una refactorización que permita corregir ese problema
- **Refactorización continua vs a posteriori**

6.2 CUÁNDO REFACTORIZAR

- Es imprescindible que el proyecto tenga pruebas automáticas, que nos permitan saber si el desarrollo sigue cumpliendo los requisitos que implementaba.
  - **Sin ellas, refactorizar conlleva un alto riesgo.**



6.2 CUÁNDO REFACTORIZAR

- **En contra:**
  - Refactorizar no está considerado como avance del proyecto para los clientes y no suele haber tiempo para ello
    - ¿A qué etapa del ciclo de vida te recuerda esto?
- **A favor:**
  - Refactorizar es una forma de mejorar la calidad.
  - Bien realizada, rápida y segura, genera satisfacción también en el cliente **¿por qué?**
    -

6.2 CUÁNDO REFACTORIZAR

- **En contra:**
  - Refactorizar no está considerado como avance del proyecto para los clientes y no suele haber tiempo para ello
    - ¿A qué etapa del ciclo de vida te recuerda esto? documentación
- **A favor:**
  - Refactorizar es una forma de mejorar la calidad.
  - Bien realizada, rápida y segura, genera satisfacción también en el cliente **¿por qué?**
    - Mejora el rendimiento, reduce tiempos

6.2 CUÁNDO REFACTORIZAR

- **Peligro:**
  - Es un arma de doble filo, ya que puede ser eterna (siempre hay un "código mejor") => **espiral refactorizadora**
  - Es importante mantener la refactorización bajo control
  - Definir claramente los objetivos antes de comenzar a refactorizar y estimar su duración ANTES DE COMENZAR.
  - Si se excede el tiempo planificado es necesario un replanteamiento.
  - Refactorizar demasiado implica:
    - Pérdida de tiempo
    - Incremento de complejidad de diseño de tanto refactorizar
    - Sensación de no estar avanzando. Sensaciones anímicas negativas.

UD 06.REFACTORIZACIÓN

6 REFACTORIZACIÓN

6.1 ¿QUÉ ES REFACTORIZAR?

6.2 CUÁNDO REFACTORIZAR

6.3 TIPOS DE REFACTORIZACIÓN

6.4 IMPLEMENTACIÓN