Buenas prácticas

para la construcción de software

Introducción

Índice de contenidos

1. Definición.
2. ¿Porqué se desarrolla mal?
3. ¿Porqué debo seguir buenas prácticas?
4. Principios de la buena programación.
5. Buenas prácticas en código.
6. Consejos generales.
7. Ejemplos basados en Java, pero aplicables a “cualquier” lenguaje de programación.

Introducción

Definición

Por ***buenas prácticas*** se entiende un conjunto coherente de acciones que han rendido buen o incluso excelente servicio en un determinado contexto y que se espera que, en contextos similares, rindan similares resultados.

Éstas dependen de las épocas, de las modas y hasta de la empresa consultora o del autor que las preconiza. No es de extrañar que algunas sean incluso contradictorias entre ellas.

(Fuente: wikipedia)

Introducción

¿Porqué se desarrolla mal?

- Falta de tiempo.
- Falta de conocimiento.
- Falta de motivación.
- Acudir a las fuentes equivocadas.
- Fallos en las etapas iniciales de desarrollo de software (análisis, requisitos, etc.).
- Etc.

Introducción

¿Porqué buenas prácticas?

- Establece reglas y convenios.
- Aporta higiene al código.
- Estandariza el desarrollo.
- Fácil lectura = Fácil mantenimiento.
- Facilita la escalabilidad del código.
- Facilita la reutilización y la integración de manera homogénea.

Principios de buena programación

The Principles of Good Programming

by Christopher Diggins

Principios de buena programación

Listado de los principios

- DRY - Don’t repeat yourself .
- Abstraction Principle.
- KISS (Keep it simple, stupid!) .
- Avoid Creating a YAGNI (You aren’t going to need it).
- Do the simplest thing that could possibly work.
- Don’t make me think.
- Open/Closed Principle.

Principios de buena programación

Listado de los principios

- Write Code for the Maintainer.
- Principle of least astonishment.
- Single Responsibility Principle.
- Minimize Coupling.
- Maximize Cohesion.
- Hide Implementation Details.
- Law of Demeter.

Principios de buena programación

Listado de los principios

- Avoid Premature Optimization.
- Code Reuse is Good.
- Separation of Concerns.
- Embrace Change.

Principios de buena programación

DRY (Don't repeat yourself)

- **No repetirse a sí mismo** → evitar la repetición.

Principios de buena programación

DRY (Don't repeat yourself)

- Evitar la repetición en todas sus posibilidades:
  - No repetir código: funciones, métodos, clases, etc. → **Reutilizar**.
  - No repetir librerías.
  - No repetir documentación.
- En general: **NO REPETIR CONOCIMIENTO**.

Principios de buena programación

Abstraction Principle

- **Principio de abstracción**. “Cada pieza de funcionalidad significativa en un programa debe ser implementada en un sólo lugar del código fuente”

Principios de buena programación

Abstraction Principle

```
class Airplane implements Aircraft {
    public void takeoff() {
      // algorithm taking off
    }
    public void fly() {
      // algorithm flying an airplane
    }
    public void land() {
      // algorithm landing an airplane
    }
}
```

```
class Helicopter implements Aircraft {
    public void takeoff() {
      // algorithm taking off a helicopter
    }
    public void fly() {
      // algorithm flying a helicopter
    }
    public void land() {
      // algorithm landing a helicopter
    }
}
```

Implementa

Implementa

```
interface Aircraft {
    void takeoff();
    void fly();
    void land();
}
```

Un piloto puede pilotar tanto **Airplane** como **Helicopter**

```
class Pilot {
   private Aircraft myShip;
   public void setMyShip(Aircraft ship) {
      myShip = ship;
   }
   public void flyMyShip() {
      myShip.takeoff();
      myShip.fly();
      myShip.land();
   }
}
```

Principios de buena programación

KISS (Keep it simple, stupid!)

- **Manténgalo simple, ¡estúpido!.** La mejor solución a un problema es la más simple.
  Menos código → menos bugs.

Principios de buena programación

KISS (Keep it simple, stupid!)

- Comentar código:

```
for( i = 0, j = 100; i < j; i++, j--)
  System.out.println(i);
```

→

```
// count to 49
for( i = 0, j = 100; i < j; i++, j--)
  System.out.println(i);
```

- Hacer test unitarios.
- Nombres (variables, métodos, etc.) descriptivos.

```
Date date1;
Date date2;
Integer myInteger;
```

→

```
Date birthday;
Date currentDate;
Integer difference;
```

Principios de buena programación

Avoid Creating a YAGNI

- **YAGNI** (You aren't going to need it) – Evitar crear algo que no vamos a necesitar. Es común tratar de ver el futuro y comenzar a crear abstracciones que todavía no estamos usando.

Principios de buena programación

Avoid Creating a YAGNI

- Se pierde tiempo al añadir, probar y documentar la funcionalidad no necesaria.
- Si la nueva característica no está bien definida, puede que no funcione correctamente, aunque con el tiempo sea necesaria.
- El software se hace más grande y más complicado innecesariamente.

Principios de buena programación

Avoid Creating a YAGNI

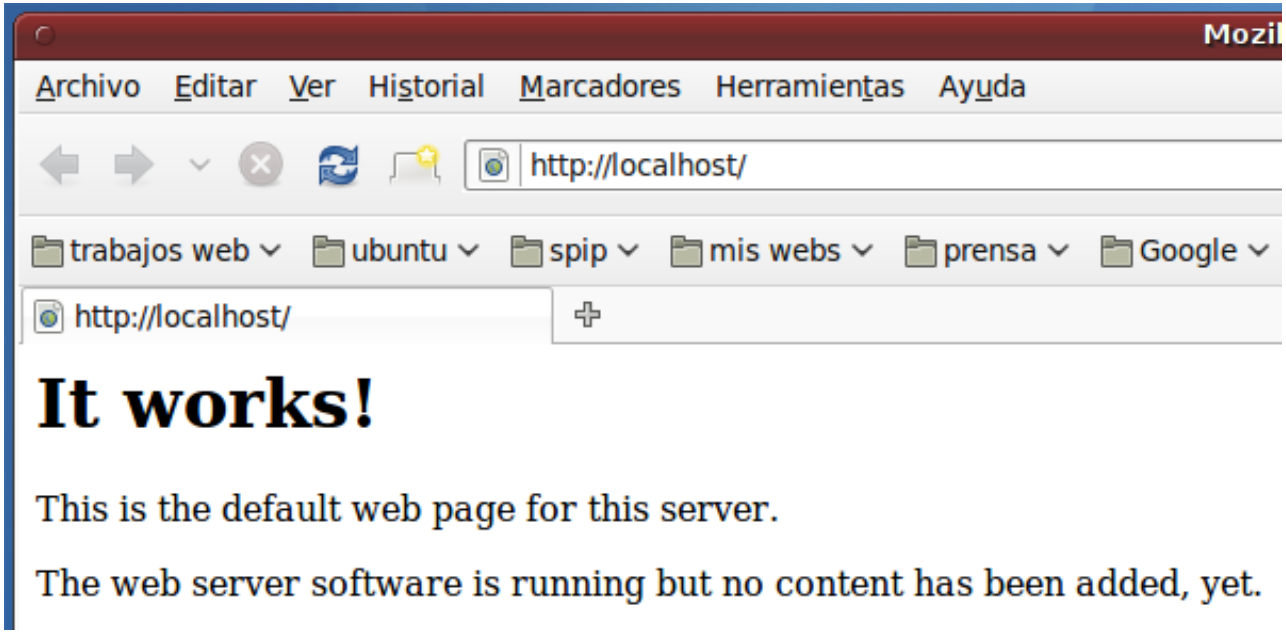
- A menos que existan especificaciones y control de revisiones, la funcionalidad no debería ser conocida por los programadores ya que podrían hacer uso de ella.
- Añadir una nueva característica innecesaria puede sugerir añadir otras nuevas características innecesarias.

Principios de buena programación

Do the simplest thing that ...

- **Do the simplest thing that could possibly work.**

Haga lo más simple que funcione.



Principios de buena programación

Don't make me think

- **No me haga pensar**. Evitar la pregunta ¿y ahora, cómo lo hago? Los nombres de las funciones, variables, etc. deben declarar claramente lo que hacen.

Principios de buena programación

Don't make me think

```
public void print(String str1, Date date1, List myList) {

        // Algorithm ...
}
```



```
public void printBirthdayPartyInfo(String userName,
    Date birthday, List<Guest> guestsList) {

        // Algorithm...
}
```

Principios de buena programación

Open/Closed Principle

- **Principio Abierto/Cerrado**. Clases, módulos, funciones, etc. deben estar abiertas a la extensión. Escribir clases, no para que otros las modifiquen, sino para que se usen y extiendan.

Principios de buena programación

Open/Closed Principle

- Beneficios del principio Open/Closed:
  - **Robustez** → No se modifican las clases ya probadas.
  - **Flexibilidad** → Se facilita la implementación de nuevos requisitos o cambios en ellos.
  - **Facilita las pruebas** → menos propenso a errores.

Principios de buena programación

Open/Closed Principle

```
class Student {
    String name;
    double percentage;
    int uid;
    [...]
```

```
class StudentBatch {
    private List<Student> studentList;
    [...]
    public void getSutdentMarkMap(HashMap<Integer, Double> studentMarkMap) {
        for (Student student : studentList) {
            studentMarkMap.put(student.uid, student.percentage);
        }
    }
    [...]
```

```
public void getSutdentMarkMap(Map<Integer, Double> studentMarkMap) {
    for (Student student : studentList) {
        studentMarkMap.put(student.uid, student.percentage);
    }
}
```

Principios de buena programación

Write Code for the Maintainer

- **Escriba código pensando en el que va a mantenerlo**. "*Escriba el código como si el que tuviera que mantenerlo fuera un psicópata asesino que conoce donde vives*".

Principios de buena programación

Principle of least astonishment

- **Principio del menor asombro**. Ejemplo: el nombre de una función debería corresponder con lo que hace.

Principios de buena programación

Principle of least astonishment

```
public void addInstance(String className) {
    try {
        Class clazz = Class.forName(className);
        objectSet.add(clazz.newInstance());
    }catch (Exception ex) {
        throw new RuntimeException(ex);
    }
}
```

```
public void createAndAddInstance(String className) {
    try {
        Class clazz = Class.forName(className);
        objectSet.add(clazz.newInstance());
    }catch (Exception ex) {
        throw new RuntimeException(ex);
    }
}
```

Este método hace más de lo que debería. Además de añadir una instancia, también la crea.

Se debería **cambiar de nombre** o **crear dos métodos**.

Principios de buena programación

Single Responsibility Principle

- **Principio de responsabilidad única**. Un componente de código debe ejecutar una única y bien definida tarea.

Principios de buena programación

Single Responsibility Principle

```
public void insertUser(User user){ … }
public void deleteUser(User user){ … }
public void updateUser(User user){ … }
```

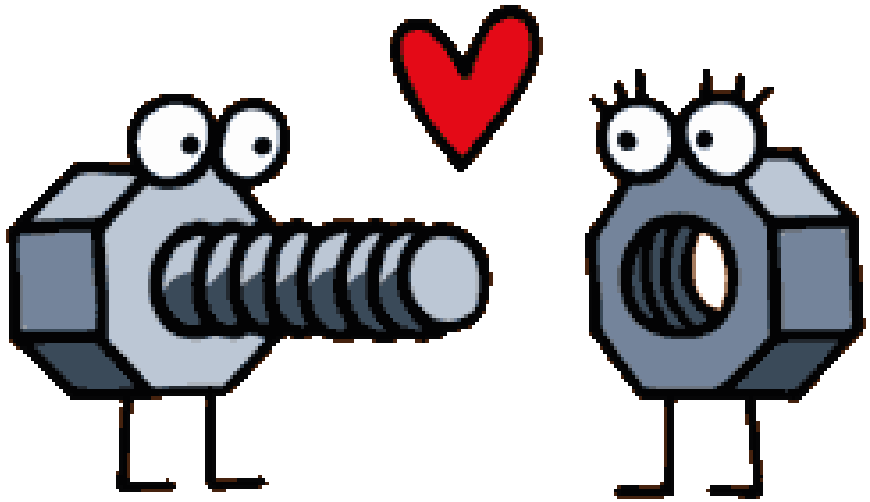
¿ QUÉ HACEN ESTAS LLAMADAS AQUÍ ?

```
public void updateUser(User user) {
    //Se actualiza el propio usuario
    userManager.update();

    //Se actualizan estadisticas para el usuario
    updateStatistics(user);

    //Se eliminan los ficheros asociados al usuario
    removeFiles(user);
}
```

Principios de buena programación

Minimize Coupling

- **Minimizar el acoplamiento**. Cada componente (bloque de código, clase, función, etc.) debe minimizar las dependencias de otros componentes.



Principios de buena programación

Maximize Cohesion

- **Maximizar cohesión**. Evitar implementar en un componente dos funcionalidades que no están relacionadas, cumpliendo tareas que no tienen relación.

Principios de buena programación

Maximize Cohesion

```
public class CoffeeCup {
  private int innerCoffee;
  public void add(int amount) {
    innerCoffee += amount;
  }
  public int remove(boolean all, int amount){
      int returnValue = 0;
      if (all) {
          int allCoffee = innerCoffee;
          innerCoffee = 0;
          returnValue = allCoffee;
      } else {
          int sip = amount;
          if (innerCoffee < amount) {
              sip = innerCoffee;
      }
          innerCoffee -= sip;
          returnValue = sip;
      }
      return returnValue;
  }
}
```

Low cohesion

```
Public class CoffeeCup {
  private int innerCoffee;
  public void add(int amount) {
      innerCoffee += amount;
  }
  public int releaseOneSip(int sipSize){
      int sip = sipSize;
      if (innerCoffee < sipSize) {
          sip = innerCoffee;
      }
      innerCoffee -= sip;
      return sip;
  }
  public int spillEntireContents() {
      int all = innerCoffee;
      innerCoffee = 0;
      return all;
  }
}
```

Hi cohesion

Principios de buena programación

Hide Implementation Details

- **Ocultar detalles de implementación**. En C# o Java → **Interfaces**. Usar correctamente **modificadores de acceso**: public, private y protected.

Principios de buena programación

Hide Implementation Details

```
ArrayList<String> users = new ArrayList<String>();
```

<u>Usar interfaces en vez de implementaciones</u>.
Se ocultan detalles de implementación y se deja libertad para elegir (ArrayList, LinkedList, etc.)

```
List<String> users = new ArrayList<String>();
```

Principios de buena programación

Avoid Premature Optimization

- **Evitar la optimización prematura**. Evitar pensar optimizar código, si apenas lo estamos armando. Ir armando el código de tal forma que podamos refactorizarlo.

Principios de buena programación

Code Reuse is Good

- **La reutilización de código es buena**. En Java → Apache Commons.

Principios de buena programación

Code Reuse is Good

Necesito un método que abrevie un string en función de un tamaño determinado...

Apache Commons nos lo proporciona

```
public static String abbreviate(String str,
int maxWidth)
```

//Ejemplo:

```
StringUtils.abbreviate("abcd", 2) = "ab..."
```

Principios de buena programación

Embrace Change

- **Abraza el cambio**. El cambio es inevitable en el desarrollo de software. No hay que luchar contra el cambio, sino trabajar para estar preparado para él → Programación ágil, integración continua, Scrum, etc.

Buenas prácticas en código

Code conventions ¿para qué?

- Alto % del coste del software → mantenimiento.
- Casi ningún software se mantiene durante toda su vida por el autor original.
- Mejora la legibilidad del software, permitiendo a los ingenieros a entender el nuevo código con mayor rapidez y en profundidad.
- Código fuente es igual que cualquier otro producto → asegurarse de que está tan bien empaquetado y limpio como cualquier otro producto.

Buenas prácticas en código

Java Code conventions

- Convenciones estándar de Sun (Oracle) para seguir y recomendar que otros sigan.
- Cubre los nombres de archivo, organización de archivos, sangría, comentarios, declaraciones, sentencias, espacios en blanco e incluye un ejemplo de código.

codeconventions-150003.pdf

Ejemplos Code conventions

Indentación

```
//DON'T USE THIS INDENTATION
if ((condition1 && condition2)
            || (condition3 && condition4)
            ||!(condition5 && condition6)) { //BAD WRAPS
            doSomethingAboutIt(); //MAKE THIS LINE EASY TO MISS
}
//USE THIS INDENTATION INSTEAD
if ((condition1 && condition2)
                        || (condition3 && condition4)
                        ||!(condition5 && condition6)) {
            doSomethingAboutIt();
}
//OR USE THIS
if ((condition1 && condition2) || (condition3 && condition4)
                        ||!(condition5 && condition6)) {
            doSomethingAboutIt();
}
```

Ejemplos Code conventions

Comentarios

- Cuatro tipos de comentarios:
  - Block
  - Single-line
  - Trailing
  - End-of-line

Ejemplos Code conventions

Clases e Interfaces

- Para programar Clases e Interfaces en Java:
  - Sin espacios entre el nombre de un método y el paréntesis “(“.
  - La llave “{” debe aparecer al final de la línea y en la misma línea que la declaración.
  - La llave “}” empieza una línea por sí misma, indentada con la línea que contiene la llave “{”.

Ejemplos Code conventions

Clases e Interfaces

```
class Sample extends Object {
        int ivar1;
        int ivar2;

        Sample(int i, int j) {
                ivar1 = i;
                ivar2 = j;
        }

        int emptyMethod() {}

        ...
}
```

Ejemplos Code conventions

Excepciones

- No dejar bloques “catch” vacíos.
- Los bloques de excepciones seguirán el siguiente formato:

```
try {
        statements;
} catch (ExceptionClass e) {
        statements;
}
```

Buenas prácticas en código

Herramientas Java

- **Checkstyle**: Herramienta para ayudar a escribir código que siga estándares. Plugin para Eclipse.
- **PMD:** Analiza y busca potenciales problemas en el código. Plugin para Eclipse.
  - Sentencias vacías try/catch, etc.
  - Variables, parámetros, etc. no usados
  - Mal uso de String/StringBuffer
  - Sentencias "if" innecesarias, for → while, etc.
  - Código copiado y pegado.

Buenas prácticas

Recursos Java

- Libro **“Effective Java”** (Joshua Bloch)
- Java Best Practices: http://www.javapractices.com/
- J2EE Best Practices: http://www.precisejava.com/

Buenas prácticas

Programación: Errores comunes

- **Fiabilidad de los parámetros de entrada**: No asegurarse de que los parámetros recibidos son los que esperamos.
- **Fiabilidad de los valores**: No comprobar si un elemento tiene valor antes de acceder a él.
- **Elementos en colecciones**: No asegurar se de que un array o colección tenga valor y contenga elementos antes de acceder a ellos.
- **No comentar**. (Sin comentarios...).

Buenas prácticas

Programación: Errores comunes

- **Acceso a colecciones**: No comprobar si existe elemento para un índice antes de acceder a él en una colección indexada.
- No liberar recursos adecuadamente.
- **Recursividad**: El método recursivo infinito.
- **Malas prácticas**: Utilizar incorrectamente técnicas conocidas.

Programación: Errores comunes

Fiabilidad de parámetros de entrada

- Los parámetros recibidos por un método «público», es decir *expuesto al mundo exterior*, nunca son confiables.

```
public static double metodoExpuesto(int nuncaMenorQue4, int nunca0){
    return nuncaMenorQue4 / nunca0 ;
}
public static void main(String... args){
    StaticDemo.metodoExpuesto(2, 0);
    //El metodo fallara despues de ejecutarse
}
```

```
public void publicarNoticia(Noticia noticia, int importancia){
    if (noticia== null){
        throw new ArgumentNullException("Parametro noticia no puede ser null");
    }
    //A partir de aqui, ira el codigo para publicar la noticia...
}
```

Programación: Errores comunes

Fiabilidad de los valores

- Acceder a los elementos sin antes comprobar si tienen un valor.

```
Bar foo = getFooBar();
foo.ToString();

//Acceder a metodos de "foo" sin comprobar, puede fallar si getFooBar
devuelve null
```



```
Bar foo = getFooBar();
if(foo != null) {
    foo.ToString();
}
```

Programación: Errores comunes

Elementos en colecciones

- No asegurarse de que un array o colección tenga valor y contenga elementos antes de acceder a ellos.

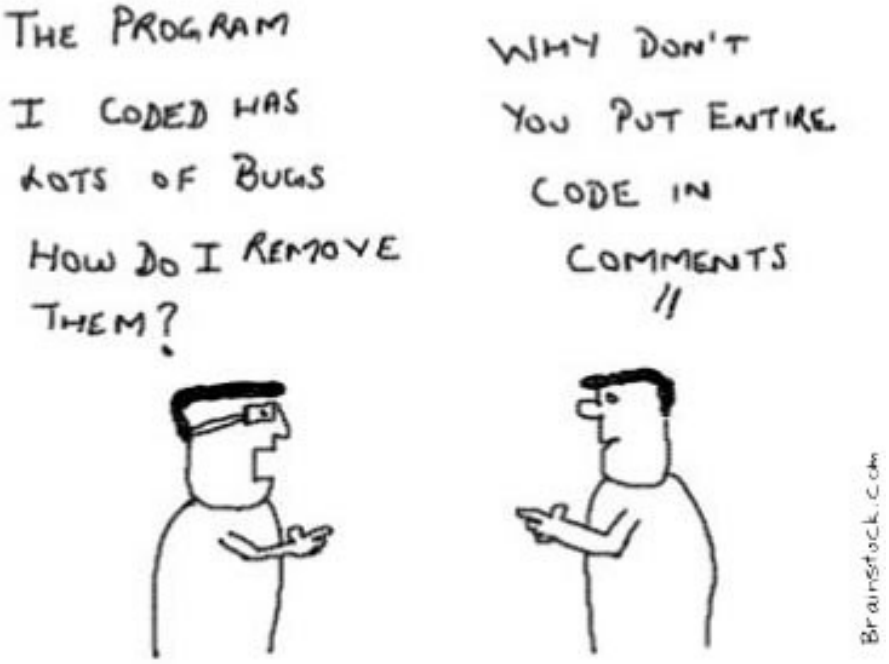
```
List<MiNumero> numeros= obtenerNumeros();
txtNumero1.text = numeros[0].toString();
txtNumero2.text = numeros[1].toString();

//Si numeros es null o numeros no tiene elementos este codigo fallara
```

```
List<MiNumero> numeros= obtenerNumeros();
if (numeros != null && numeros.size() > 0) {
    txtNumero1.text = numeros.get(0).toString();
    txtNumero2.text = numeros.get(1).toString();
}
```

Programación: Errores comunes

No comentar

- Comentar el código también es programar,pero se debe comentar con criterio.



Programación: Errores comunes

Acceso a colecciones

- Antes de acceder a un elemento de una colección indexada se debe comprobar si la colección tiene al menos el número de elementos que necesitamos.

```
for (int i = 0; i < cargarEmpresas().size(); i++) {

    if (cargarEmpresas().get(i) == ""){
        gridView.rows[i].cells[7].visible = false;
    }
}
```

¡HORROR!

```
for (int i = 0; i < empresasSize; i++) {
    if (gridView.rows.count > i && "".equals(empresas.get(i))){
        gridView.rows[i].cells[7].visible = false;
    }
}
```

Programación: Errores comunes

No liberar recursos

- Grave error no liberar recursos adecuadamente.

```
public void saveData() {
    Connection cnn;
    try{
        cnn = new SqlConnection(cnnString);
        cnn.open();
    } catch (Exception e) {
        System.out.println("Se ha producido un erro de base de datos");
        e.printStackTrace();
    }
    finally {
        if (cnn != null) {
            cnn.close();
         }
    }
}
```

Programación: Errores comunes

Recursividad

- El típico fallo al utilizar recursividad es no poner un límite a las llamadas recursivas.

```
static long LastFibonacciNumber(long anterior1, long anterior2) {

    long numero = anterior1 + anterior2;
    Console.write(numero + " ");
    numero = lastFibonacciNumber(anterior2, numero);
    return numero;
}
```

- Para evitar el conocido error de "stack overflow" → utilizar líneas de control que permitan dar fin al anidamiento de llamadas recursivas.

Programación: Errores comunes

Malas prácticas

- Utilizar incorrectamente técnicas conocidas.
- Habitualmente se implementa mal una técnica o patrón conocidos.
- **Avoid basic style errors** → javapractices.
- **Common Java Mistakes** → javaprogrammingforums
- **Java Anti-Patterns** → www.odi.ch

Buenas prácticas

Ejemplos

Buenas prácticas

Consejos generales

- Seguir estándares.
- **Seguir patrones de diseño**. 
- Reutilizar código → No reinventar la rueda.
- Usar librerías “reconocidas”, documentadas, etc. Ejemplo: Apache Commons
- Test unitarios.
- Conocer bien nuestro entorno de desarrollo (IDE): shortcuts, plugins, etc.

Buenas prácticas

Consejos generales

- **Refactorización**: Técnica de ingeniería de software para reestructurar código fuente, alterando su estructura interna sin cambiar su comportamiento externo.
- **Comentar** de forma completa pero no excesiva.
- Seguir “buenas prácticas” del Framework elegido.
- Nombrar “con sentido” las clases, variables, etc.

Buenas prácticas

Consejos generales

- Leer código ajeno. Que otros lean tú código.
- Pautas para leer código de una manera efectiva:

Declarar- Inicializar - Utilizar	Definir Funciones - Métodos	Datos Externos Tratar - Convertir
-¿Dónde esta declarada? -¿Cuándo se inicializa? -¿Es posible que no tenga valor? -¿Utiliza algún recurso externo?	-¿Qué parámetros recibo? -¿Qué restricciones quiero aplicar? -¿Cómo quiero informar del uso incorrecto de mi método?	-¿De qué tipo son? -¿Tengo que realizar alguna conversión? -¿Cuál es la forma más segura de realizar la conversión?

Buenas prácticas

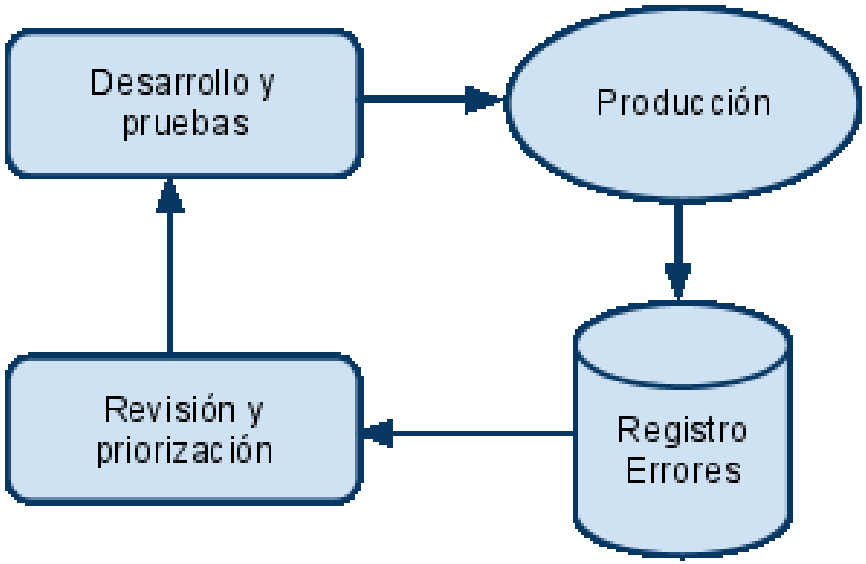
Consejos generales

- **Tener en cuenta el rendimiento**: El rendimiento de nuestras aplicaciones es tan importante como otros criterios de aceptación. Criterios a tener en cuenta para medir el rendimiento:
  - Recursos hardware del cliente.
  - Realizar pruebas de rendimiento en un entorno que simule el hardware mínimo aceptado.
  - Hacer pruebas REALES.

Buenas prácticas

Consejos generales

Buena gestión de logs: Los logs son las trazas o rastros que deja una aplicación. Nos brindarán información valiosa sobre qué está ocurriendo en nuestro sistema y dónde aplicar las mejoras más inmediatas.



Buenas prácticas

Consejos generales

- Organiza el desarrollo:
  - Define una **sintaxis de programación común** para tu equipo de trabajo.
  - Define una **estructura de directorios** basada en la separación de conceptos:
    - Código fuente.
    - Documentación.
    - Recursos.
    - Productos finales.
    - Divide proyectos en librerías.

Buenas prácticas

Consejos generales

- Usa **sistemas de control de versiones** para compartir el trabajo con el equipo. Usándolos se aprende a usarlos. 
  Libro gratuito → http://www.ericsink.com/vcbe/
- Usa herramientas o prepara scripts que te permitan automatizar procesos productivos. → ANT, Maven etc.
- Nightly builds → Generar productos finales "frescos" todos los días.

Buenas prácticas

Consejos generales

- **Comparte conocimiento**. Wiki, blogs, reuniones, charlas, etc.

Buenas prácticas

Consejos generales

- Cobertura TDD: Test Driven Development →
http://cobertura.sourceforge.net/
- Integración continua →
http://continuum.apache.org/
- Software para la gestión de tareas/bugs →
http://www.redmine.org/, http://www.mantisbt.org/, http://trac.edgewall.org/, etc.

Material extra

Buenas prácticas en el uso de repositorios

Buenas prácticas

Referencias

- Principios de buena programación: The Principles of Good Programming, Christopher Diggins.
- Java Code Conventions: http://www.oracle.com/technetwork/java/codeconventions-150003.pdf
- Effective Java. Joshua Bloch.
- MasterDISS_DesignPatterns, Iker Jamardo.

Buenas prácticas

Referencias

- Imágenes
  - **DRY**: http://www.dreamstime.com/royalty-free-stock-photo-don%27t-repeat-yourself-acronym-image20103645
  - **Abstraction Principle**: http://www.danpontefract.com/give-piece-a-chance/
  - **Open/Closed Principle**: http://www.vitalygorn.com/blog/image.axd?picture=Open-Closed.jpg
  - **YAGNI: http://blog.thescrumster.com/?tag=yagni**
  - **Don't make me think:** http://riyajs.files.wordpress.com/2011/03/usability_testing.jpg
  - **KISS:** http://milldesk.com.br/en/k-i-s-s-keep-it-simple-stupid/
  - **Write Code for the Maintainer:** http://notio.com.ar/adjuntos/imagenes/000/102/0000102304.jpg

Buenas prácticas

Referencias

- Imágenes
  - **YAGNI**: http://blog.thescrumster.com/?tag=yagni
  - **Don't make me think**: http://riyajs.files.wordpress.com/2011/03/usability_testing.jpg
  - **KISS**: http://milldesk.com.br/en/k-i-s-s-keep-it-simple-stupid/
  - **Write Code for the Maintainer**: http://notio.com.ar/adjuntos/imagenes/000/102/0000102304.jpg
  - **Single Responsibility Principle:** http://www.colourbox.com/preview/1995285-118643-3d-illustration-of-one-selected-target-with-three-arrows-right-target-concept.jpg

Buenas prácticas

Referencias

- Imágenes
  - **Maximize Cohesion**: http://vision2030.bwd.public-i.tv/viewfinder_core/public/images/uploaded/l_1259326852_cohesion.png
  - **Hide Implementation Details**: http://geek.pe /wp-content/uploads/2010/04/como-ocultar-tu-ip-al-enviar-un-correo-gmail-hotmail.jpg
  - **Avoid Premature Optimization:** http://rlv.zcache.es/la_optimizacion_prematura_es_la_raiz_de_todo_el_ma_im%C3%A1n-p147403910183181037en878_210.jpg
  - **Code Reuse is Good:** http://www.alexanderinteractive.com/blog/reuse-code.jpg

Buenas prácticas

Referencias

- Imágenes
  - **Embrace Change:** http://static.eharmony.com/dating-advice/wp-content/uploads/images/are-you-caught-in-the-comfort-zone-trap-learning-to-embrace-change-large.jpg
  - **Minimize Coupling:** http://lh3.ggpht.com/_huul_BKMb4M/S2wKiTapibI/AAAAAAAAGek/h9Qs-AKcGa8/tuerca.gi**f**

Buenas prácticas
para la construcción de software

FIN

Iker Canarias
iker.canarias (gmail)