Introducción a la ingeniería de software – Clase 5

En la clase anterior ...

Diseño

Definiciones.

Conceptos de diseño

Diseño arquitectónico

Estilos.

Buenas prácticas

Conceptos para un buen diseño

Avisos

- Ingeniería de SW, Pfleeger, Shari
- TP N°3

Temas de hoy

Implementación

Pautas. Aspectos.
Documentación.

Pruebas

Objetivos de la prueba.
Tipos de defectos.
Organización de las pruebas.
Equipo.

Principios

Principios para guiar la práctica.

1

Implementación

Codificación. Documentación.

Implementación de una solución

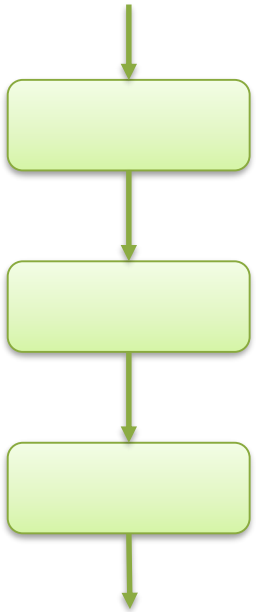
- Codificación del software
  - Múltiples lenguajes de programación
  - Varias personas involucradas en el proceso
  - Grupos de trabajo: cooperación y coordinación

- Se definen estándares de programación
  - Estilo, formato: código y documentación

Pautas para la programación

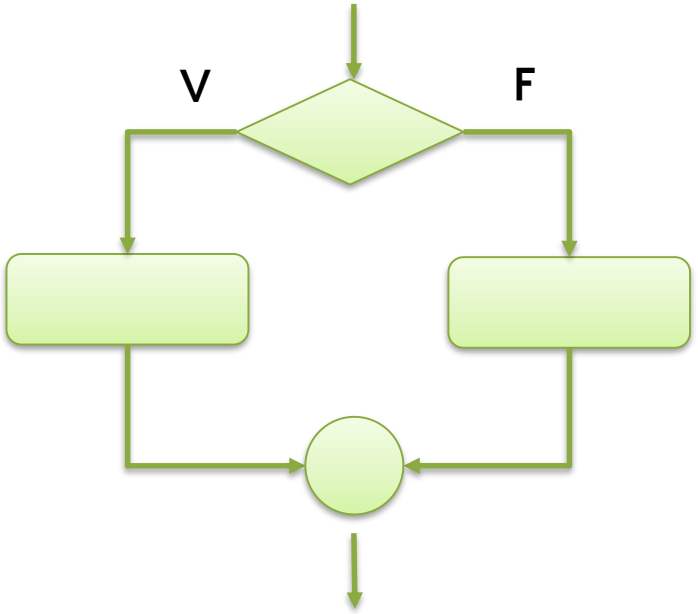
- A partir del diseño del componente el programador tiene libertad para aplicar su creatividad en la implementación.
- Cada componente de un programa implica al menos tres aspectos principales:
  - 1-Estructuras de control
  - 2-Algoritmos
  - 3-Estructuras de datos

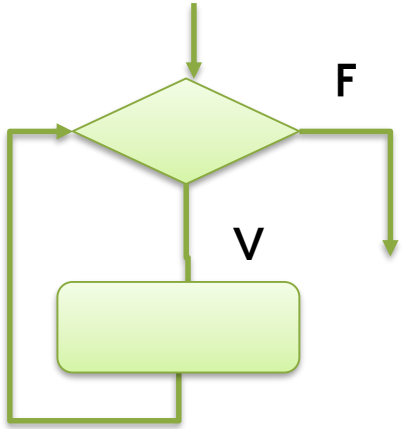
1-Estructuras de control



Secuencia

Selección





Decisión

1-Estructuras de control

- Determinadas por la estructura misma del código
- Debe poder leerse fácilmente un componente de lo general a lo particular
- **Modularidad**: también aplicarla al código
  - Facilita el mantenimiento y la reutilización
  - Generalizar todo lo posible sin afectar la performance o la facilidad de comprensión

2-Algoritmos

- Se debe tener en cuenta:
  - **Tiempo** de ejecución
  - Costo de **escritura**
  - Costo de **prueba**: en tiempo y en dificultad de contrucción de los casos de prueba
  - Costo de **modificación** cuando sea necesario
  - No sacrificar claridad y corrección por velocidad de ejecución

3-Estructuras de datos

- Buscar que el almacenamiento y la manipulación de los datos sea directa
- Que ayude a simplificar el programa
- La estructura de los datos puede determinar la estructura del programa

Documentación del código

Explica qué hace el programa y cómo lo hace

- Documentación **interna**: material descriptivo escrito directamente dentro del código

- Documentación **externa**: toda otra documentación que acompaña al programa

Documentación interna

- Identifica el programa
- Describe algoritmos, estructuras de datos y flujo de control
- Usualmente ubicada al comienzo de cada componente
  - Denominación del componente
  - Quién lo ha escrito
  - Dónde va ubicado en el diseño general del sistema
  - Cuándo fue escrito/revisado/modificado
  - Justificación del componente
  - Uso que hace de las estructuras

Documentación interna y externa

- Recomendable en la documentación interna
  - Comentarios que acompañen el código
  - Nombres de variables significativas
  - Dar formato que ayude a la comprensión
- Incluir en la documentación externa
  - Descripción del problema
  - Descripción de los algoritmos elegidos
  - Descripción de los datos

Documentación interna - estándares

Header Comment Blocks

All source code files in the PEAR repository shall contain a "page-level" docblock at t
above each class. Below are examples of such docblocks.

```
<?php

/* vim: set expandtab tabstop=4 shiftwidth=4 softtabstop=4: */

/**
 * Short description for file
 *
 * Long description for file (if any)...
 *
 * PHP version 5
 *
 * LICENSE: This source file is subject to version 3.01 of the PHP license
 * that is available through the world-wide-web at the following URI:
 * http://www.php.net/license/3_01.txt.  If you did not receive a copy of
 * the PHP License and are unable to obtain it through the web, please
 * send a note to license@php.net so we can mail you a copy immediately.
 *
 * @category   CategoryName
 * @package    PackageName
 * @author     Original Author <author@example.com>
 * @author     Another Author <another@example.com>
 * @copyright  1997-2005 The PHP Group
 * @license    http://www.php.net/license/3_01.txt  PHP License 3.01
 * @version    SVN: $Id$
 * @link       http://pear.php.net/package/PackageName
 * @see        NetOther, Net_Sample::Net_Sample()
```

```
 * @link       http://pear.php.net/package/PackageName
 * @see        NetOther, Net_Sample::Net_Sample()
 * @since      File available since Release 1.2.0
 * @deprecated File deprecated in Release 2.0.0
 */

/*
* Place includes, constant defines and $_GLOBAL settings here.
* Make sure they have appropriate docblocks to avoid phpDocumentor
* construing they are documented by the page-level docblock.
*/

/**
 * Short description for class
 *
 * Long description for class (if any)...
 *
 * @category   CategoryName
 * @package    PackageName
 * @author     Original Author <author@example.com>
 * @author     Another Author <another@example.com>
 * @copyright  1997-2005 The PHP Group
 * @license    http://www.php.net/license/3_01.txt  PHP License 3.01
 * @version    Release: @package_version@
 * @link       http://pear.php.net/package/PackageName
 * @see        NetOther, Net_Sample::Net_Sample()
 * @since      Class available since Release 1.2.0
 * @deprecated Class deprecated in Release 2.0.0
 */
class Foo_Bar
{
}
```

Generación de documentación

```
/**
 * Returns an Image object that can then be painted on the screen.
 * The url argument must specify an absolute {@link URL}. The name
 * argument is a specifier that is relative to the url argument.
 * <p>
 * This method always returns immediately, whether or not the
 * image exists. When this applet attempts to draw the image on
 * the screen, the data will be loaded. The graphics primitives
 * that draw the image will incrementally paint on the screen.
 *
 * @param  url  an absolute URL giving the base location of the image
 * @param  name the location of the image, relative to the url argument
 * @return      the image at the specified URL
 * @see         Image
 */
public Image getImage(URL url, String name) {
        try {
            return getImage(new URL(url, name));
        } catch (MalformedURLException e) {
            return null;
        }
}
```

getImage

```
public Image getImage(URL url,
                      String name)
```

Returns an `Image` object that can then be painted on the screen. The `url` argument must specify an absolute URL. The `name` argument is a specifier that is relative to the `url` argument.

This method always returns immediately, whether or not the image exists. When this applet attempts to draw the image on the screen, the data will be loaded. The graphics primitives that draw the image will incrementally paint on the screen.

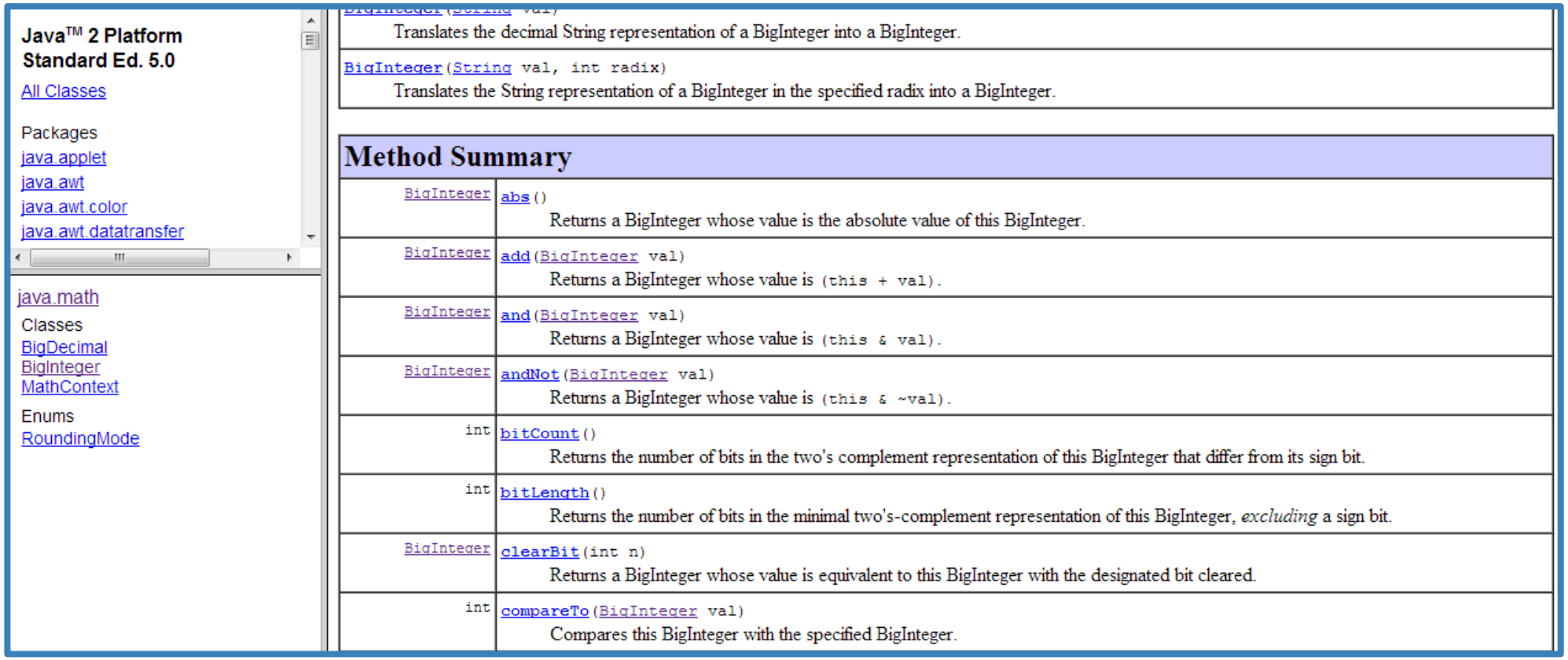
Parameters:
`url` - an absolute URL giving the base location of the image.

`name` - the location of the image, relative to the `url` argument.

Returns:
the image at the specified URL.

See Also:
`Image`

Generación de documentación



Java™ 2 Platform Standard Ed. 5.0

All Classes

Packages
java.applet
java.awt
java.awt.color
java.awt.datatransfer

java.math
Classes
BigDecimal
BigInteger
MathContext
Enums
RoundingMode

Translates the decimal String representation of a BigInteger into a BigInteger.

`BigInteger(String val, int radix)`
Translates the String representation of a BigInteger in the specified radix into a BigInteger.

Method Summary

BigInteger	`abs()` Returns a BigInteger whose value is the absolute value of this BigInteger.
BigInteger	`add(BigInteger val)` Returns a BigInteger whose value is `(this + val)`.
BigInteger	`and(BigInteger val)` Returns a BigInteger whose value is `(this & val)`.
BigInteger	`andNot(BigInteger val)` Returns a BigInteger whose value is `(this & ~val)`.
int	`bitCount()` Returns the number of bits in the two's complement representation of this BigInteger that differ from its sign bit.
int	`bitLength()` Returns the number of bits in the minimal two's-complement representation of this BigInteger, *excluding* a sign bit.
BigInteger	`clearBit(int n)` Returns a BigInteger whose value is equivalent to this BigInteger with the designated bit cleared.
int	`compareTo(BigInteger val)` Compares this BigInteger with the specified BigInteger.

2

Pruebas

Objetivos.
Defectos. Clasificación.

Prueba de los programas

Se concentra en la búsqueda de defectos en los programas

- ¿Por qué falla el software?
  - Especificación **erronea** o requerimientos **omitidos**
  - Requerimientos **imposibles** de implementar dado el HW y SW con los que se cuenta
  - El diseño del **sistema** es defectuoso
  - El diseño del **programa** es defectuoso
  - El **código** tiene **errores**

Objetivo de la prueba

- Se prueba para **demostrar la existencia de defectos**
- La prueba es destructiva, se considera exitosa solamente si encuentra defectos o si se producen fallas como consecuencia de los procedimientos de prueba
- La visión del programador es opuesta: busca demostrar que su código es correcto. Esto va en contra del espíritu de las pruebas

Tipos de defectos

- **Algorítmicos**: el algoritmo o la lógica de un componente no producen la salida apropiada para la entrada dada, por errores en los pasos del procesamiento
  - Bifurcar antes o después de tiempo
  - Probar una condición erronea
  - Olvidar inicializar variables o constantes de un bucle
  - Olvidar probar una condición particular
  - Comparar variables de tipos de datos inapropiados

Tipos de defectos

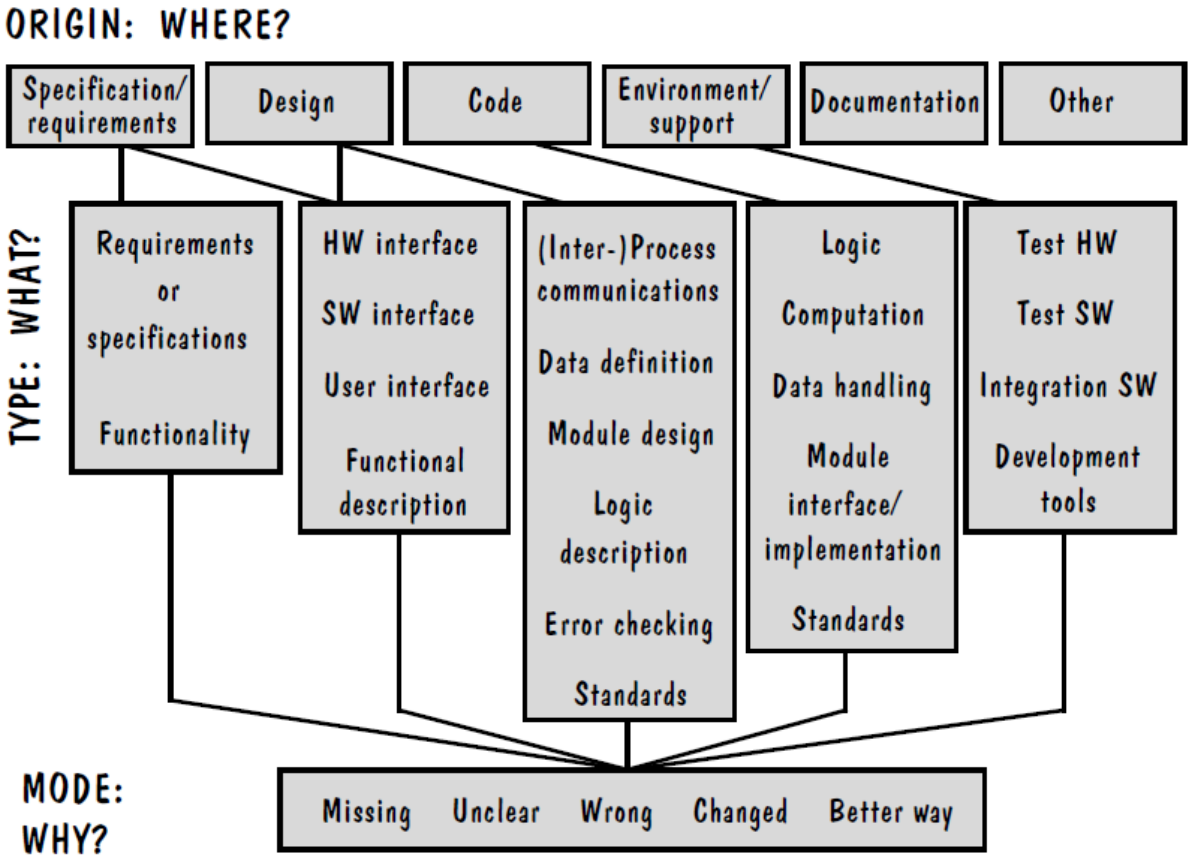
- **De sintaxis**: en las estructuras del lenguaje de programación
- **De computación y de precisión**: implementación erroneas de fórmulas o cálculos sin la exactitud necesaria
- **De documentación**: cuando ésta no se corresponde con lo que el programa verdaderamente hace
- **Por estrés o sobrecarga**: se sobrepasan las estructuras

Tipos de defectos

- **De capacidad o límites**: se deteriora el desempeño del sistema por superar sus límites especificados
- **De sincronización o coordinación**: coordinación inadecuada de procesos que deben ejecutar de forma simultánea
- **De rendimiento o desempeño**: el sistema no opera con el desempeño (velocidad, seguridad, exactitud, confiabilidad) esperado
- **De recuperación**: la respuesta a una falla no es la esperada

Clasificación de defectos

- Clasificación de Hewlett-Packard



Visiones de los objetos de prueba

- Caja cerrada o negra
  - El objeto se analiza desconociendo el contenido
  - Las pruebas consisten en alimentar la “caja negra” con entradas y observar cuáles son las salidas
- Caja abierta o blanca
  - Con la visión de caja cerrada no se puede elegir casos de prueba representativos
  - Se utiliza la estructura del objeto para probar de diversas maneras
  - Por ejemplo: todos los caminos posibles del código

Organización de las pruebas

1. De módulo, de componente, o de unidad
   - Verificar funcionamiento del componente con los tipos de entrada esperados, estructuras de los datos, lógica y condiciones de límites para los datos
2. De integración
   - Verificar el funcionamiento conjunto de los componentes del sistema
3. Funcional
   - Determinar si las funciones descriptas en la especificación son realmente ejecutadas por el sistema integrado

Organización de las pruebas

4. De rendimiento o desempeño
   - Comparar el sistema con los requerimientos de hardware y software.
5. De aceptación
   - En conjunto con el cliente se prueba el sistema contra las especificaciones de requerimientos
6. De instalación
   - Se prueba una vez más, para garantizar que una vez instalado funcione como debe hacerlo

Pruebas de unidad

- Existen diversas técnicas de prueba
- Ejemplos:
  - Ejecución simbólica
    - Ejecución simulada del código utilizando símbolos en lugar de variables de datos
  - Demostración automatizada de teoremas
- Casos de prueba: conjunto particular de casos de entrada a ser utilizado en la prueba de un programa.

Minuciosidad de la prueba

- La selección de los casos de prueba se hace para utilizar al menos un enfoque:
  - Comprobación de sentencias: ejecutar al menos una vez cada sentencia del componente
  - Pruebas de la ramificación: cada rama de cada decisión en el código se elige al menos una vez
  - Comprobación del camino: cada camino distinto formado en el código se ejecuta por lo menos una vez
  - ...

Planificación de la prueba

- Pasos
  - Determinación de los objetivos de la prueba
  - Diseño de los casos de prueba
  - Preparación escrita de los casos de prueba
  - Verificación de los casos de prueba
  - Ejecución de los casos de prueba
  - Evaluación de los resultados

Equipo de prueba

- Los programadores están demasiado familiarizados con la estructura e intención de la implementación y pueden tener dificultades reconociendo errores y diferencias con los requerimientos.

- Idealmente el equipo de pruebas es independiente del equipo de implementación.

Equipo de prueba

- Las personas encargadas de las pruebas son profesionales con experiencia y se involucran en esta tarea desde el comienzo del proyecto.
  - Organizar las pruebas
  - Diseñar las pruebas
  - Ejecutar las pruebas

3

Principios

Para guiar la práctica del ingeniero

Principios

- El ingeniero de software aplica de manera cotidiana un conjunto de métodos, herramientas y **principios**.

- Las técnicas y metodologías pueden cambiar, las herramientas pueden renovarse o modificarse, pero los principios básicos en los que basamos la profesión se conservan y dan sustento a todo lo demás

Principios

- Para guiar el proceso
- Para guiar la práctica
- De comunicación
- De planificación
- De modelado
- De construcción
- De despliegue

Analizamos principios para el ingeniero

- Leer sección correspondiente del capítulo 4 Pressman.
- Resumir cada principio:
  - Frase
  - Ejemplo
  - Imagen
  - ...

  de manera tal de poder explicarlo a sus compañeros.

Principios para guiar el proceso

1. Ser ágil
2. Centrarse en la calidad
3. Poder adaptar
4. Formar un equipo eficaz
5. Establecer mecanismos para la comunicación y coordinación
6. Administrar el cambio
7. Evaluar el riesgo
8. Crear productos que agreguen valor para otros

Principios para guiar la práctica

1. Dividir y conquistar
2. Entender y usar la abstracción
3. Buscar coherencia
4. Centrarse en la transferencia de información
5. Modularizar eficazmente
6. Buscar patrones
7. Usar perspectivas diferentes
8. Pensar en quien dará mantenimiento al software

Principios para la comunicación

Uno de los mayores desafíos de la profesión:
lograr una comunicación efectiva.

1. Escuchar
2. Prepararse previamente
3. Determinar un “facilitador” o “líder”
4. Comunicarse cara a cara
5. Tomar notas y documentar decisiones

Principios para la comunicación

6. Buscar la colaboración
7. Centrarse, “modularizar” la discusión
8. Utilizar dibujos para clarificar
9. ¡Avanzar! al acordar, si no se puede acordar, si no se puede aclarar, avanzar.
10. Negociar no es competir, lo ideal es que todos ganen.

Principios para la planificación

1. Entender el alcance del proyecto
2. Involucrar a los participantes en la planificación
3. Reconocer la necesidad de iterar
4. Estimar basandose en conocimiento previo
5. Considerar los riesgos
6. Ser realista

Principios para la planificación

7. Ajustar la granularidad
8. Definir cómo se buscará la calidad
9. Describir cómo se administrará el cambio
10. Realizar seguimiento frecuente del plan y ajustar en la medida que sea necesario

Principios para el modelado

- Dos* clases de modelos:
  1. De requerimientos
  2. De diseño

- Requerimientos: dominio de la información, de la funcionalidad y del comportamiento
- Diseño: arquitectura, interfaz de usuario y detalle a nivel de componentes

Principios para el modelado

- Modelado de los requerimientos (análisis)
  1. Representar y entender el dominio del problema
  2. Definir las funciones del software
  3. Representar el comportamiento del software
  4. Dividir los modelos: información, función, comportamiento
  5. Avanzar desde lo esencial hacia el detalle

Principios para el modelado

- Modelado del diseño
  1. Rastreable a los requerimientos
  2. Tener en cuenta la arquitectura a construir
  3. Es tan importante diseñar datos como funciones
  4. Cuidar las interfaces
  5. Resaltar facilidad de uso y necesidades del usuario
  6. Independencia funcional de los componentes
  7. Buscar buen acoplamiento de componentes
  8. Los modelos deben ser fáciles de comprender
  9. Hacerlo iterativo, buscando sencillez

Principios para el modelado

- Calidad externa: propiedades del software observables fácilmente por el usuario

- Calidad interna: propiedades que sirven al ingeniero de software

- La calidad interna conduce a la calidad externa. Se deben buscar LAS DOS CALIDADES.

Principios para la construcción

- Preparación:
  - Entender el problema
  - Comprender conceptos y principios básicos de diseño
  - Elegir el lenguaje de programación “correcto”
  - Seleccionar un ambiente de desarrollo adecuado
  - Crear pruebas unitarias para aplicar al terminar

Principios para la construcción

- Programación
  - Programación estructurada
  - Seleccionar cuidadosamente las estructuras de datos
  - Entender la arquitectura, crear acorde a ella
  - Mantener la lógica condicional sencilla
  - Cuidar el anidamiento
  - Seguir estandares
  - Usar nombres significativos
  - Escribir código “autodocumentado” y legible

Principios para la construcción

- Validación
  - Recorrer el código
  - Realizar pruebas unitarias
  - Corregir errores detectados
  - Rediseñar el código cuando resulte apropiado

Principios para la construcción

- Pruebas
  - Pensar la prueba como proceso que tiene como **objetivo encontrar un error**
  - Un caso de prueba bueno tiene probabilidades altas de encontrar errores no detectados
  - Una prueba exitosa es aquella que encontró un error no detectado

Principios para la construcción

- Pruebas
  - Rastreables a los requerimientos
  - Planeadas con anterioridad
  - Principio de Pareto: “proporciones 80-20”

> El 80% de los errores no detectados durante las pruebas, se relacionan con el 20% de los componentes del programa.

  - Comenzar en lo pequeño y avanzar a lo grande
  - Es imposible hacer pruebas exhaustivas

Principios para el despliegue

Entrega – Apoyo – Retroalimentación

1. Manejar la expectativa del cliente
2. Ensamblar y probar paquete completo
3. Establecer previamente el soporte a brindar
4. Proporcionar material de aprendizaje
5. Los defectos primero se corrigen

Principios - resumen

- Los proyectos, los métodos, los conceptos y las herramientas son diferentes, los principios se aplican en todos los casos.
- Ayudan en la aplicación de una ingeniería de software eficaz.
- Guían al equipo de software durante todo el proceso.

RESUMEN.1

- ¿Qué es la Ingeniería de Software?
- Proceso de resolución de problemas
- Participantes y roles en el proceso
- Sistemas. Software producto y herramienta
- Proceso. Flujos. Modelos.
- Ciclos de vida
- Ingeniería de Requerimientos

RESUMEN.2

- Diseño. Descomposición
- Diseño arquitectónico. Patrones.
- Modularidad. Abstracción. Acoplamiento. Cohesión.
- Implementación. Codificación.
- Buenas practicas. Documentación
- Pruebas. Tipos de pruebas. De proceso. De producto.
- Principios para el ingeniero de software

Bibliografía

- *Ingeniería de software . Teoría y Práctica* - S. L. Pfleeger

  Capítulo 7 – Escribiendo los programas.

  Capítulo 8 – Prueba de los programas.

  Capítulo 9 – La prueba del sistema.

- *Ingeniería del software. Un enfoque práctico* - R. Pressman

  Capítulo 4 – Principios que guían la práctica.