Clase 7

Reflexión e IC

Clase 7 – Parte I

Reflexión

1. Reflexión

1.1. Concepto

La **reflexión** es la capacidad que tiene un programa para observar e incluso modificar su estructura de alto nivel.

1. Reflexión

1.1. Concepto

Es la habilidad de un código fuente para examinar y modificar su estructura, específicamente los valores, metadata, propiedades y funciones del programa en tiempo de ejecución.

1. Reflexión

1.1. Concepto

La reflexión es una actividad computacional que razona sobre su propia computación.

1. Reflexión

1.1. Concepto

Normalmente la reflexión es **dinámica** o en **tiempo de ejecución**, aunque algunos lenguajes de programación permiten reflexión **estática** o en **tiempo de compilación**.

1. Reflexión

1.1. Concepto

Es más común en lenguajes de programación de **alto nivel** ejecutándose sobre una máquina virtual, como **Smalltalk** o **Java**, y menos común en lenguajes como **C**.

1. Reflexión

1.1. Concepto

Cuando el código fuente de un programa se compila, normalmente se pierde la información sobre la estructura del programa al generarse el código de bajo nivel.
Pero si permite reflexión, se guarda la estructura como metadatos en el código generado.

1. Reflexión

1.2. Historia

Las primeras computadoras se programaban en sus lenguajes **ensamblador** o ***assembler*** nativos, que eran inherentemente reflexivos dado que esas arquitecturas originales podían ser programadas definiendo instrucciones como datos y usando código auto-modificable.

1. Reflexión

1.2. Historia

Cuando la programación se fue trasladando a lenguajes de alto nivel, esta capacidad reflexiva natural desapareció y no regresó hasta que aparecieron los lenguajes de programación con reflexión integrada en sus sistemas.

1. Reflexión

1.2. Historia

La noción de reflexión computacional en los lenguajes de programación fue introducida por **Brian Cantwell Smith** en la presentación de su tesis doctoral en 1982.

1. Reflexión

1.3. Implementación

Un lenguaje con reflexión proporciona un conjunto de funcionalidades disponibles en tiempo de ejecución que serían muy difícilmente realizables en un lenguaje de más bajo nivel.

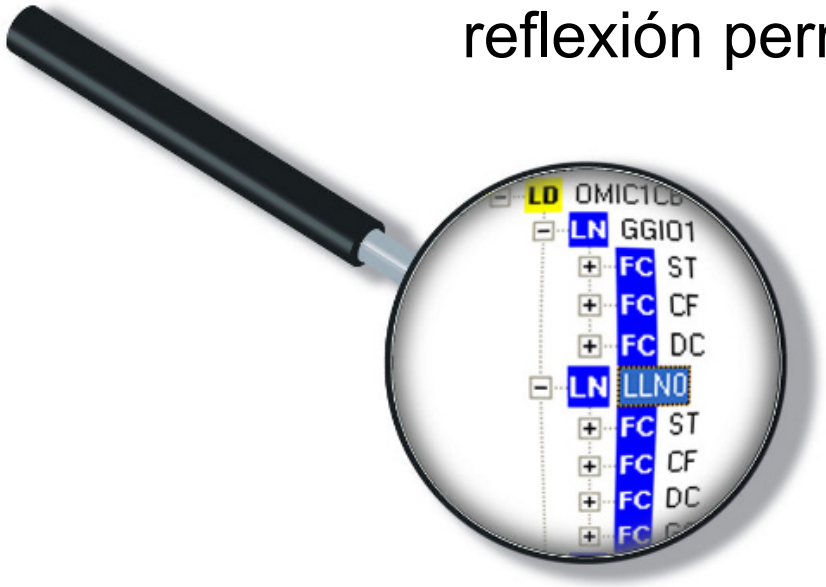
1. Reflexión

1.3. Implementación

Funcionalidades típicas:
-Descubrir y modificar construcciones de código fuente (bloques, clases, métodos, protocolos);
-Convertir una cadena que corresponde al nombre simbólico de una clase o función en una referencia o invocación a esa clase o función;
-Evaluar una cadena como si fuera una sentencia de código fuente.

1. Reflexión

1.3. Implementación



En lenguajes orientados a objetos como Java, la reflexión permite la inspección de clases, interfaces, campos y métodos en tiempo de ejecución sin conocer sus nombres en tiempo de compilación, entre otras funcionalidades.

1. Reflexión

1.3. Implementación

La reflexión también se puede usar para adaptar dinámicamente un programa a diferentes situaciones.

1. Reflexión

1.3. Implementación

Por ejemplo, en una aplicación que usa las clases A y B para realizar operaciones similares, sin reflexión el llamado a sus métodos tendría que ser hardcodeado.
Pero con reflexión, los llamados los métodos se podrían realizar sin hardcodear sus nombres.

1. Reflexión

1.4. Ejemplos

Los siguientes fragmentos de código crean una instancia **a** de la clase **A**, e invocan su método "**hola**".

```
function enEdition(){
   /* Ne rien faire mode edit + preload */
   if( encodeURIComponent(document.location).sear
turn;
   // /&preload=/

   if ( !wgPageName.match(/Discussion.*\/Traduct
   var diff = new Array();
   var status; var pecTraduction; var pecRelectu
   var avancementTraduction; var avancementRelec
/* ************ Parser ************ */
   var params = document.location.search.substr
gth).split('&');
   var i = 0;
   var tmp; var name;
   while ( i < params.length )
   {
      tmp = params[i].split('=');
      name = tmp[0];
      switch( name ) {
         case 'status':
         status = tmp[
```

1. Reflexión

1.4. Ejemplos (1)


```php
class A{
      private $ejemplo = true;
      public hola(){...}
}

// Sin reflexión
$a = new A(); $a->hola();

// Con reflexión
$reflector = new ReflectionClass('A');
$a = $reflector->newInstance();
$hola = $reflector->getMethod('hola');
$hola->invoke($a);
```

1. Reflexión

1.4. Ejemplos (2)

PHP

```
$propiedad = $reflector -> getProperty("ejemplo");
$propiedad -> setAccessible(true);

$obj = new A();
echo $propiedad -> getValue($obj); // Works
echo $obj -> myProperty; // Doesn't work (error)
```

1. Reflexión

1.4. Ejemplos

Ruby

```ruby
# Sin reflexión
obj = A.new
obj.hola

# Con reflexión
class_name = "A"
method = :hola
obj = Kernel.const_get(class_name).new
obj.send method
```

1. Reflexión

1.4. Ejemplos

Python


```
# Sin reflexión
obj = A()
obj.hola()

# Con reflexión
class_name = "A"
method = "hola"
obj = globals()[class_name]()
getattr(obj, method)()
```

1. Reflexión

1.4. Ejemplos

Java

```
// Sin reflexión
A a = new A();
a.hola();

// Con reflexión
Object a = A.class.newInstance();
Method m = a.getClass().getDeclaredMethod
                    ("hola", new Class<?>[0]);
m.invoke(a)
```

Clase 7 – Parte II

IC (Integración Continua)

2. Integración continua

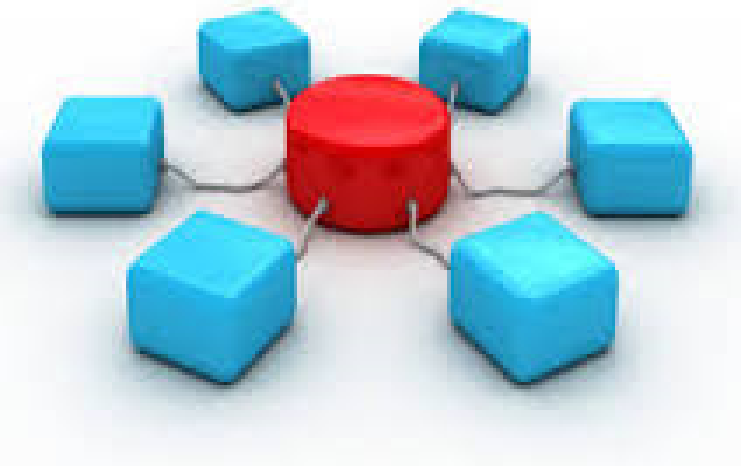
2.1. Concepto

La **integración continua** es un modelo informático que consiste en hacer la compilación y ejecución de pruebas de todo un proyecto de manera automática lo más a menudo posible, para poder detectar fallos cuanto antes.

2. Integración continua

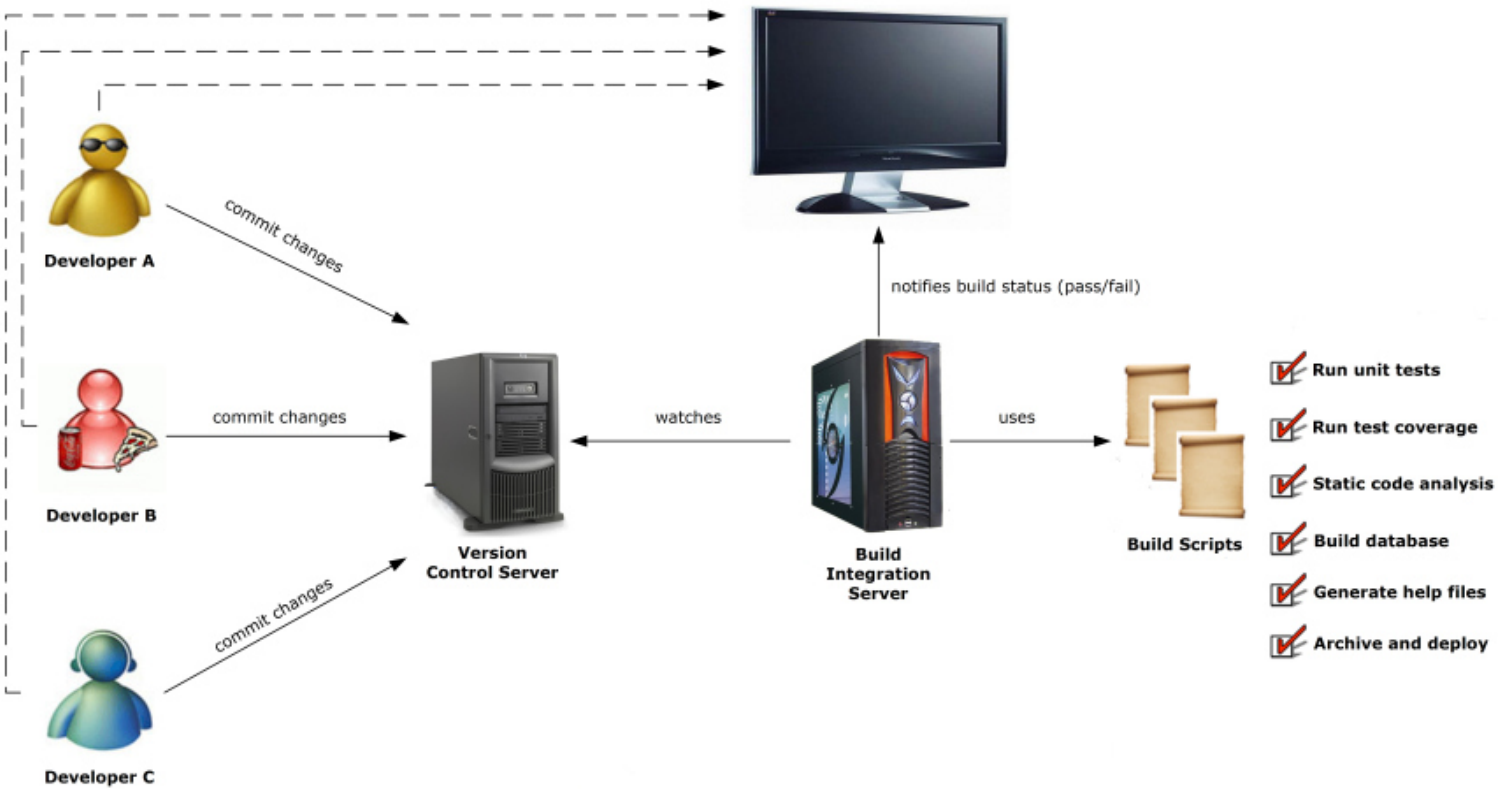
2.1. Concepto

Implica la fusión, varias veces al día, de todas las copias de los trabajos de los desarrolladores con una línea principal compartida.



2. Integración continua

2.1. Concepto



2. Integración continua

2.1. Concepto

El proceso consiste normalmente en descargar cada X cantidad de horas el código fuente desde el control de versiones, compilarlo, ejecutar las pruebas y generar informes.

2. Integración continua

2.1. Concepto

Se usan aplicaciones como **Anthill**, **Bamboo**, **Continuum**, **Hudson** o **Jenkins** para proyectos Java, y **CruiseControl.Net** o **Team Foundation Build** para .Net, que se encargan de controlar las ejecuciones y se apoyan en herramientas como **Ant** o **Maven** (para Java) y **Nant** o **MSBUILD** (para .Net) que se encargan de las compilaciones, pruebas e informes.

2. Integración continua

2.1. Concepto

La integración continua está estrechamente asociada con el **desarrollo ágil de software**, como la metodología de programación extrema.

2. Integración continua

2.2. Historia

La integración continua, conocida como CI por sus siglas en inglés, fue nombrada y propuesta por **Grady Booch** en 1991.

2. Integración continua

2.2. Historia

En 1997 fue incorporada por **Kent Beck** a su metodología **XP**.
Beck publicó un trabajo sobre integración continua en 1998, remarcando la importancia de la comunicación cara a cara por sobre el soporte tecnológico.

2. Integración continua

2.3. Ventajas

Se pueden detectar y solucionar los problemas de integración de manera continua, evitando el tradicional caos que aparece cuando se acerca la fecha de entrega.

2. Integración continua

2.3. Ventajas

Permite disponer constantemente de una versión para pruebas, demos o lanzamientos anticipados.
Con IC se pueden monitorear las métricas de calidad del proyecto prácticamente en tiempo real.