Universidad Simón Bolívar
Departamento de Computación y Tecnología de la información
CI-2691- Laboratorio de algoritmos I

Pre-Laboratorio 8

El objetivo de este laboratorio es presentar una introducción a los principios de desarrollo ágil, práctica de programación en parejas, ritmo sostenido y desarrollo de casos de prueba.

Principios de los procesos de desarrollo ágil

El desarrollo ágil es muy utilizado actualmente en el área de computación para construir sistemas de software. En éste participan pequeños equipos que trabajan en colaboración organizada y disciplinada. La idea es desarrollar software de corte pequeño y mediano en lapsos cortos de tiempo. Se basa en ciertos principios, entre los cuales destacan:

- El software que funciona es la medida principal de progreso.
- Los procesos ágiles promueven un desarrollo sostenido, es decir, el equipo debe mantener un ritmo constante de forma indefinida.
- La atención continua a la calidad técnica y al buen diseño mejora la agilidad.
- La simplicidad es esencial.

El desarrollo ágil promueve ciertas prácticas que son esenciales para obtener un producto de software de calidad. El mayor beneficio de prácticas que se describen a continuación, se consigue con su aplicación conjunta y equilibrada:

- ***Diseño simple:*** Se debe diseñar la solución más simple que pueda funcionar y ser implementada en un momento determinado del proyecto.
- ***Refactorización (Refactoring):*** Es una actividad constante de reestructuración o modificación del código con el propósito de remover su repetición o duplicación, mejorar su legibilidad, simplificarlo y hacerlo más flexible; facilitando de esta manera aprendizaje, reutilización y cambios futuros. Se mejora la estructura interna del código sin alterar su comportamiento externo.
- ***Programación en parejas:*** Toda la producción de código debe realizarse con trabajo en parejas de programadores. Esto conlleva a las siguientes ventajas: menor tasa de errores, mejor diseño, mayor satisfacción de los programadores, entre otras.
- ***Propiedad colectiva del código:*** Cualquier miembro del equipo puede cambiar cualquier parte del código en cualquier momento.
- ***Estándares de programación:*** Se enfatiza que se debe establecer directrices o lineamientos para que exista un lenguaje común entre los programadores con el propósito de mantener el código legible.

Principios de buena programación

Para el desarrollo de sistemas de software bajo las prácticas de la programación ágil, es imprescindible seguir los principios que rigen una buena programación. A continuación se listan los principios que pueden ser aprovechados en una primera experiencia de programación con este método de desarrollo:

- ***Evitar la repetición de código o documentación y en su lugar reutilizar.***
- ***Principio de abstracción***: “Cada pieza de funcionalidad significativa en un programa debe ser implementada en un sólo lugar del código fuente”
- ***La mejor solución a un problema es la más simple***: Ejemplo:

```
# EJEMPLO DE UNA SOLUCIÓN POCO SIMPLE
i=0
j=100
while ( i < j ):
      # Este ciclo tiene 50 iteraciones.
      i = i+1
      j= j-1

# EJEMPLO DE UNA SOLUCIÓN SIMPLE
i = 0
while ( i < 50 ):
   # Este ciclo también tiene 50 iteraciones.
   i = i + 1
```

- *Hacer pruebas unitarias*.
- *Usar identificadores descriptivos*: Los nombres de las funciones, variables, etc. deben señalar claramente lo que hacen: Ejemplos:

```
# EJEMPLO DE NOMBRES POCO DESCRIPTIVOS
      dia0 = int( input( "Dia de nacimiento: " ) )
      mes0 = int( input( "Mes de nacimiento: " ) )
      año = int( input( "Año de nacimiento: " ) )
      dia1 = int( input( "Dia actual: " ) )
      mes1 = int( input( "Mes actual: " ) )
      año1 = int( input( "Año actual: " ) )

# EJEMPLO DE NOMBRES DESCRIPTIVOS
      diaDeNacimiento = int( input( "Dia de nacimiento: " ) )
      mesDeNacimiento = int( input( "Mes de nacimiento: " ) )
      añoDeNacimiento = int( input( "Año de nacimiento: " ) )
      diaActual = int( input( "Dia actual: " ) )
      mesActual = int( input( "Mes actual: " ) )
      añoActual = int( input( "Año actual: " ) )
```

- ***Principio del menor asombro***: El nombre de un subprograma debería corresponder con lo que hace.
- ***Principio de responsabilidad única***: Un componente de código (subprograma) debe ejecutar una única y bien definida tarea.
- ***Minimizar el acoplamiento:*** Cada componente (bloque de código, subprograma, etc.) debe minimizar las dependencias de otros componentes.

- ***Maximizar cohesión:*** Evitar implementar en un componente dos funcionalidades que no están relacionadas, cumpliendo tareas que no tienen relación.
- ***La reutilización de código es buena.***
- **No usar más parámetros de entrada de los que se necesitan:** Asegurarse de que los parámetros recibidos son los que se esperan.

Programación por Pares o en Pareja

La Programación por pares es considerada como uno de los pilares de una metodología ágil. Es una técnica relativamente sencilla de aplicar en la cual dos programadores trabajan codo con codo, en un único computador, para desarrollar el código de un programa. En este contexto, la persona que codifica recibe el nombre de *conductor*, mientras que la persona que revisa el código recibe el nombre de *observador o navegante*. Estos dos roles, conductor y observador (o navegante), en ningún momento, deben ser considerados como roles estáticos.

Los miembros de la pareja deben intercambiar sus roles de manera frecuente durante toda la actividad de desarrollo. Si no se cumple este sencillo requisito, no se podrá hablar de una verdadera Programación por Pares. El objetivo de este método de programación es mejorar la calidad del código. Las buenas prácticas en la programación por pares son:

- ***Todo se comparte***: Los miembros de la pareja, sin importar el rol que estén desempeñando en un determinado momento, son los autores del código. Ambos tendrán que aprender a compartir sus éxitos y sus fracasos, expresándose mediante frases como "cometimos un error en esta parte del código" o, mucho mejor, "hemos pasado todas las pruebas sin errores".
- ***Observador activo***: No debe caerse en el error de confundir el rol de observador con disponer de un periodo de descanso, realizar una llamada o contestar a un mensaje de texto recibido en el teléfono. Por el contrario el observador activo ocupa su tiempo realizando un proceso continuo de análisis, diseño y verificación del código desarrollado por el conductor, tratando en todo momento de que la pareja sea capaz de alcanzar la excelencia del código.
- ***Deslizar el teclado, no cambiar las sillas:*** Esta regla indica que el espacio de trabajo debe estar dispuesto de modo que no sea necesario intercambiar el lugar frente al teclado (cambiar las sillas) para poder intercambiar los roles. Con el simple movimiento de deslizar el teclado para que éste cambie de manos la pareja debe ser capaz de intercambiar los roles de conductor a observador/navegante y viceversa.
- ***La importancia del descanso:*** Deben realizarse descansos de manera periódica, que ayudarán a la pareja a retomar el desarrollo con la energía necesaria, por lo que es imprescindible que, durante los mismos, los dos desarrolladores mantengan sus manos y su mente alejados de la tarea que estén desarrollando en pareja.

- ***Roles:***
    - *Conductor*: codifica las funcionalidades, refactoriza y realiza el testing.
    - *Navegante*: guía al conductor en el desarrollo de las estrategias que se deben aplicar para codificar las funcionalidades y resolver conflictos en el código a través del aporte de ideas.

Programación Sostenida

Es importante llevar un ritmo sostenido de trabajo. El concepto que se desea establecer con esta práctica es el de planificar el trabajo para mantener un ritmo constante y razonable, sin sobrecargar al equipo. Cuando un proyecto se retrasa, trabajar tiempo extra puede ser más perjudicial que beneficioso. El trabajo extra desmotiva inmediatamente al grupo e impacta en la calidad del producto.

Programación dirigida por las pruebas o Test-Driven Development (TDD)

En las metodologías tradicionales, la fase de pruebas, incluyendo la definición de los test, es usualmente realizada sobre el final del proyecto, o sobre el final del desarrollo de cada módulo. Las metodologías ágiles proponen un modelo inverso, en el que, lo primero que se escribe son las pruebas que el sistema debe pasar. Luego, el desarrollo debe ser el mínimo necesario

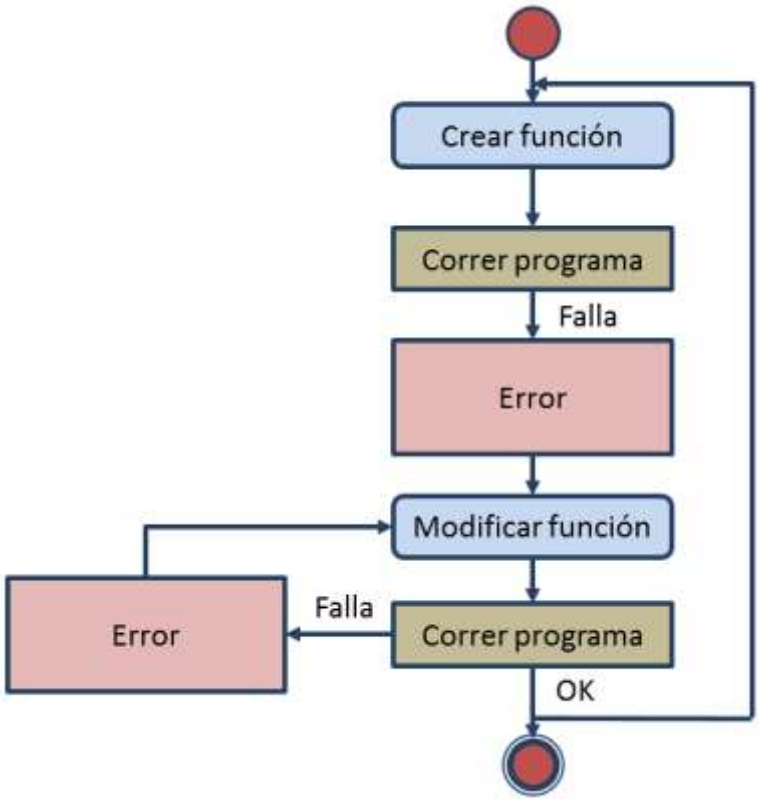
para pasar las pruebas previamente definidas. Estas pruebas, llamadas pruebas unitarias, corresponden a probar cada componente de software (o subprograma) de manera individual. Para ello, hay que intentar que los casos de prueba elegidos cubran la posibilidad de encontrar el mayor número de errores posibles, ya que los errores permanecerán ocultos hasta que se ejecute la porción de código que los provoca y para que dicha porción se ejecute es necesario que el programador realice las acciones necesarias para provocarlo.

Algunas recomendaciones para hacer pruebas: verificar el tipo de datos de entrada (**precondiciones)** y los resultados esperados (**postcondiciones**). En resumen se deben generar los "escenarios de pruebas", es decir, las diferentes condiciones en las que el programa deberá trabajar.

Ejercicios a entregar:

Para cada uno de los ejercicios que se plantea a continuación, ingrese cada uno de los valores por consola y escriba precondiciones, postcondiciones, invariantes, cotas y el manejo de las excepciones correspondientes. Así mismo, utilice técnicas de descomposición funcional (análisis descendente) dadas en laboratorios anteriores.

Método de trabajo que será aplicado: Los programadores deben cambiar el rol de observador y navegante cada vez que corra el programa y su resultado sea el esperado, considerando la figura que se muestra a continuación:



1. **(Ejercicio1PreLab8.py): Desarrollar un programa empleando el método de Programación por Pares, que tome dos argumentos y realice las funciones que siguen a continuación:**
   a.- entero + entero,
   b.- string + string,
   c.- lista + lista,
   d.- entero + string,
   e.- entero + lista
   f.- string + lista;

   Nota: Se debe convertir un entero a una cadena antes de realizar la concatenación. Si el caso es un string o entero+lista, añade el entero a la lista (independientemente de si es el primero o segundo argumento).

2. **(Ejercicio2PreLab8.py): Escriba un programa en Python utilizando el método de Programación por Pares y genere las funciones correspondientes para cada uno de los casos que se le presentan a continuación:** Dados dos números enteros m y n, devuelva, m/n, si m=10; m*n si m=5; m+n si m=3; m^n si m=2 y en cualquier otro devuelva m.

Condiciones de la entrega

Cree un archivo comprimido del tipo "tgz" llamado **PreLab8-X.tgz**, donde X es su número de carné, que contenga los archivos: "**Ejercicio1PreLab8.py**" y "**Ejercicio2PreLab8.py**". Debe subir el archivo en el aula virtual, en la sección del Laboratorio 8, el martes 08 de noviembre del 2016 antes de las 8:00 a.m.

Referencias

[1] José H. Canós, Patricio Letelier y Mary Carmen Penadés, "Metodologías Ágiles en el Desarrollo de Software", DSIC Universidad Politécnica de Valencia. Disponible en la web. http://www.willydev.net/descargas/prev/TodoAgil.Pdf

[2] Scrum Manager®, Programación por Parejas, Disponible en la web: http://www.scrummanager.net/bok

[3] Christopher Diggins, "The Principles of Good Programming", July 24, 2011. Disponible en la web: http://www.artima.com/weblogs/viewpost.jsp?thread=331531