Programación Orientada a Objetos: Proyecto

Profesor: Gerardo M. Sarria M.

Marzo 8 de 2018

1. Introducción

Los programas de computadora siempre están cambiando. Hay que solucionar errores, añadir mejoras y llevar a cabo optimizaciones. No sólo hay que cambiar la versión actual, sino también la versión del año pasado (que todavía está siendo usada) y la versión del año que viene (que ya casi funciona). Además de los problemas que requieren la realización de cambios para solucionarlos, el mismo hecho de cambio lleva problemas adicionales.

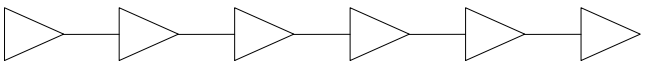
En la labor de desarrollar software, una función muy importante y normalmente descuidada es la de controlar las versiones de cada programa fuente.

El objetivo de este proyecto es desarrollar un sistema orientado a objetos que provea apoyo en la administración de cambios de forma eficiente. Este sistema debe mantener un registro de las versiones de un programa fuente o un documento del sistema con el conjunto de cambios realizados sobre ellos.

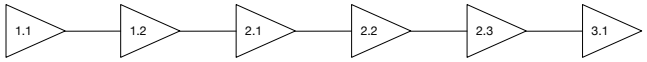
Los servicios que debe prestar son: el almacenamiento de información por cada versión construída y la obtención de cualquier versión ya generada de un archivo.

2. El Problema del Manejo de Versiones

El desarrollo típico de un archivo es la generación de nuevas versiones en forma lineal. En una gráfica de la evolución de un archivo, cada nodo representa una nueva versión. Entonces el desarrollo de un archivo se vería así:



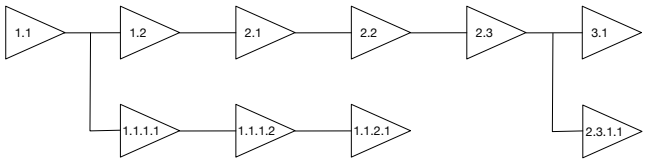
La nomenclatura de los nodos utiliza dos números en la forma $R.N$, donde R se denomina *release*, y N nivel. El nivel se aumenta por defecto, cada vez que se genera una nueva versión. Cuando ocurre una modificación mayor, el usuario puede incrementar el *release*. Así, el desarrollo lineal se podría general de la forma como lo muestra la siguiente figura.



Hay ocasiones en que se está construyendo una nueva versión, pero es necesario corregir urgentemente errores de una versión anterior, ya liberada. Esto da lugar a la generación de un nuevo "delta" que se desprende de dicha versión anterior, pero que no es parte del tronco principal, llamada rama.

Para diferenciar las ramas de las versiones del tronco principal, aparecen dos números más $R.N.B.S$: la rama y la secuencia. Si se producen más versiones sobre la ramificación, la secuencia se aumenta por defecto, a menos que el usuario indique otra acción (ver adelante).

De esta forma, se puede generar un producto con la evolución mostrada en el siguiente diagrama.



Sólo se pueden desprender ramas de troncos de versiones ya constituídas. Por ejemplo, es imposible generar la versión 1.1.1.1, sin haber generado la versión 1.2.

3. Definición Formal del Problema

Se necesita entonces que cree un sistema orientado a objetos que resuelva eficientemente el problema de controlar las versiones de un archivo fuente.

La base del sistema es mantener para cada programa, **un archivo de texto único** que contiene la información de control sobre las versiones del archivo fuente, y que guarda sólamente las diferencias entre las versiones con el fin de reducir el espacio de almacenamiento.

Las diferencias entre las versiones se registran como líneas que se insertan o se borran. Por ejemplo, suponiendo que

como versión 1.1 se tiene el siguiente programa (algoritmo 1):

```
1   from string import *
2   from math import *
3   from sys import *
4
5   def main():
6      # Imprime un saludo #
7      print "Hola Mundo!\n"
8      return 0
```

y en la versión 1.2 el programa queda así (algoritmo 2):

```
1   from sys import *
2   import net
3   from string import *
4
5   def main():
6      # Imprime los numeros de 0 a 10 #
7      i = 0
8      for i in range(10):
9         print i
10     return 0
```

En la administración de versiones sólo se debe registrar las líneas 1, 2, 6 y 7 del algoritmo 1 como eliminadas, y las líneas 2, 3, 6, 7, 8 y 9 del algoritmo 2 como insertadas, en el archivo de control del programa, cuando genera la versión 1.2.

El sistema debe proveer el siguiente conjunto de operaciones:

- **Crear:**

  Cuando un programa ya esta completo en su primera versión, se entrega a la administración del sistema por medio de la operación crear.

  ```
  crear(<nombre archivo actual>,
        <nombre archivo de control>)
  ```

  `<nombre archivo control>` es el nombre del programa, antecedido por el prefijo 's_'. Por ejemplo, dados los archivos prog1.py y prog2.py, completos en su primera versión, se entregan al administrador de versiones de la siguiente manera:

  ```
  crear(prog1.py, s_prog1.py)
  crear(prog2.py, s_prog2.py)
  ```

- **Obtener:**

  Cuando es necesario visualizar o hacer modificaciones sobre los archivos administrados, se le debe pedir una copia de la versión a trabajar, hacerle las modificaciones si es pertinente y generar el correspondiente "delta". La operación tiene varias opciones que se explican a continuación. En todo caso, siempre se graba la versión obtenida con el nombre original del archivo, sin el prefijo ('s_').

  ```
  obtener(<archivo de control>, <version>)
  ```

  Obtiene del archivo de control, una copia de la versión especificada del archivo fuente para modificarla y generar una nueva versión. Si ninguna versión es especificada se obtiene la última. Además esta operación alista el número de versión que se va a generar en el "delta".

  El número del nuevo "delta" depende del argumento R, $R.N$, $R.N.B$, ó $R.N.B.S$, como sigue:

  - $R.N + 1$, si no existe esa versión, y la última del tronco principal es $R.N$. Por ejemplo, si la última versión del tronco principal es la 1.2, y el argumento se especifica la 1.2, se genera la 1.3.

  - $R.N.1.1$, si ya existe la versión $R.N + 1$, y la versión obtenida es $R.N$. Por ejemplo, si la última versión del tronco principal es la 1.3, y se especifica en el argumento la 1.2, se genera la 1.2.1.1.

  - $R.N.B.S + 1$, si la versión obtenida es $R.N.B.S$. Por ejemplo, si existe una ramificación cuya última versión es la 1.1.1.3 y se especifica esa misma en el argumento, se genera la 1.1.1.4.

  - $R + 1.1$, si el argumento R es un número mayor al último número de *release* del tronco principal. Por ejemplo, si la última versión del tronco principal es la 1.3 y se especifica en el argumento la 2, se genera la 2.1.

  - $R.N.B + 1.1$, si en el argumento $R.N.B$, B es mayor al B de la última versión de la rama $R.N$. Por ejemplo, si existe una ramificación cuya última versión es la 1.1.1.2 y en el argumento se especifica la 1.1.2, se genera la 1.1.2.1.

- **Modificar:**

  Agrega al archivo de control, las modificaciones que se hicieron en el archivo fuente, generando una nueva versión cuyo número dependerá de la operación

obtener anterior.

```
modificar(<archivo de control>)
```

Con el fin de ahorrar espacio, en el archivo de control únicamente se almacenan las diferencias entre la versión como estaba almacenada y la nueva versión que se está almacenando.

El sistema debe informar al usuario la versión creada, el número de líneas que se insertaron, el número de líneas que se borraron, y el número de líneas que quedaron iguales, respecto a la versión inmediatamente anterior.

- **Diferencia:**

  Despliega la diferencia entre dos versiones de un mismo archivo, especificando las líneas insertadas y eliminadas, en términos de los cambios que se le deberían hacer a la primera versión para convertirla en la segunda versión.

  ```
  dif(<version 1>, <version 2>,
      <archivo de control>)
  ```

  Los cambios se especifican siguiendo el formato:

  <rango archivo 1> comando <rango archivo 2>

  donde los rangos pueden ser un número de línea o un intervalo de líneas que están implicadas en la operación y el comando correspondiente a la letra `i` para insertar ó la letra `e` para eliminar, como se explica a continuación:

  - Nc`i`Nf, donde Nc es el número de línea (o intervalo) anterior a la inserción del rango en la primera versión, es decir, la línea anterior a partir de donde se encontró la diferencia (si es la primera del archivo se usará el 0). Nf es la línea (o intervalo) insertada en la segunda versión. A continuación se deben mostrar la línea insertada .
  - Nc`e`Nf, donde Nc es el número de línea (o intervalo) eliminada en la primera versión, y Nf es el número de línea (o intervalo) de la segunda versión en donde los archivos nuevamente coinciden. Se dice que dos líneas coinciden cuando son exactamente iguales. Si los archivos no vuelven a coincidir, no hay Nf. A continuación se deben mostrar la línea eliminada.

  Las líneas que están implicadas en el cambio se anteceden con `<` si corresponden a la primera versión, ó `>` si corresponden a la segunda versión. Por ejemplo, la salida de comparación de las dos versiones anteriormente expuestas es:

  ```
  1-2e1
  <from string import *
  <from math import *
  3i2-3
  >import net
  >from string import *
  6-7e10
  <   # Imprime un saludo #
  <   print "Hola Mundo!\n"
  5i6-9
  >   # Imprime los numeros de 0 a 10 #
  >   i = 0
  >   for i in range(10):
  >       print i
  ```

- **Historial:**

  Muestra todo el historial de versiones del archivo.

  ```
  historial(<archivo de control>)
  ```

  Se debe visualizar claramente toda la secuencia de versiones (como en la tercera figura de la página 1).

4. Notas Aclaratorias

- El proyecto es parejas.
- La solución al problema debe implementarse en el lenguaje de programación C++.
- Se deberán implementar dos interfaz de entrada/salida, una texto y una gráfica. La entrada se leerá de archivos texto; la interfaz de salida debe permitir ver claramente que la solución sí satisface el pedido de la entrada.
- El desarrollo del proyecto debe quedar documentado, utilizando las herramientas adecuadas (diagramas de clases, casos de uso, diagrama de secuencia, documento de pruebas, etc.).
- Debe usarse todo el potencial de la programación orientada a objetos (clases, iteradores, templates, etc.).
- Se debe entregar el código fuente del programa y un documento pdf con todo el análisis, diseño, implementación y pruebas del proyecto, empaquetados en un archivo con el nombre clave `pry-objetos-2018_1.zip`.