Introducción al desarrollo de Software

Cecilia Jarne
cecilia.jarne@unq.edu.ar
Twitter: @ceciliajarne

Ideas básicas para empezar

- La elección del (o los) lenguaje(s) de programación.
- Herramientas, procesos de organización, desarrollo y mantenimiento
de soft.
- Estrategias y buenas prácticas.
- Desarrollo de soft colaborativo.
- Favorecer el uso de soft reutilizable.

Lenguajes compilados e interpretados

- Compiladores e Intérpretes son programas que convierten el código a lenguaje de máquina.
- Lenguaje de máquina son las instrucciones que entiende el procesador en código binario.

```
package main

import "fmt"

func main() {
  fmt.Printf("hello, world")
}
```

```
01010101111011100011010
01000101000101010010100
01010100101010100001010
00110100010101000111100
01100101001010101010010
11100011010100100100010
```

Lenguaje de alto nivel que entiende el programador

Lenguaje de máquina que entiende el procesador

Lenguajes compilados e interpretados

- El lenguaje compilado requiere un paso adicional antes de ser ejecutado, la compilación, que convierte el código a lenguaje de máquina.
  Ejemplos: C, C++, Fortran, Java, Go y Rust y muchos otros
- Un lenguaje interpretado es convertido a lenguaje de máquina a medida que es ejecutado.
  Ejemplos: Python, R, Ruby y JavaScript y otros.

Procesos de organización y desarrollo

- ¿Para quién estoy programando? Para mí, para nosotros, para otros...
- ¿Qué tareas se necesitan, cómo las vamos a implementar?
- Traducir los requerimientos en subtareas.
- Usar algún método de project management

(para ser estrictos con los tiempos de desarrollo, implementación y optimización.)

Estrategias y buenas prácticas.

• No inventar la pólvora! USAR LIBRERIAS!
(Invertir tiempo en ver si ya existe una manera de implementar la tarea)
• Observar ejemplos antes de decidir la mejor estrategia.
• Comunicarse fluidamente con quien pide el software o desarrolla junto
con nosotros.
• Documentar el proceso.
(Utilizar las herramientas existentes)

La clave es Observar ejemplos antes de decidir la mejor estrategia
ALGUNAS IDEAS DE DONDE MIRAR:

- http://stackoverflow .com/ (mi favorito!)
- http://numerical.recipes/
- https://projecteuler .net/

Desarrollo de software colaborativo.

Escribir software mas modular y reusable. Escribir frameworks y librerías.

- Software modificable.
- Que sus componentes puedan ser combinadas sin tener que recompilar
(si se puede).
- Combinar código script y compilado.
- Intentar que las componentes puedan ser (re) testeadas y (re) validadas.
- BUENA DOCUMENTACIÓN SOBRE EL SOFT QUE ESCRIBIMOS!!!!!!

Desarrollo de software colaborativo.

Algunos ejemplos:

- http : //nipy .org /nitime/index.html
- https : //root.cern.ch/
- https : //www .tensorflow .org /
- http : //scikitlearn.org /
- https : //keras.io/

Favorecer el uso de soft reusable

Escribir módulos para combinar o usar script dependiendo de las características y tamaños del problema a resolver

¿Qué hace distinto al soft científico?

- Los requerimientos no están del todo definidos a veces.
- Limitaciones en el cálculo de punto flotante pueden perjudicarnos.
- Algunas aplicaciones pueden ser usadas una vez.
- No todos los científicos sabemos programar o hacerlo del modo más eficiente.
- A veces (muchas) las implementaciones deben ser hechas por gente inexperta.

¿Por qué usar lenguaje de script?

- Portabilidad.
- No hay necesidad de recompilar.
- Disponibilidad de librerías en la propia plataforma.
- Flexibilidad.

¿Por qué usar lenguaje de script?

- Adaptabilidad.
- Posibilidad de adaptar múltiples extensiones de archivos.
- Conveniencia.
- Los script lenguajes tienen gran facilidad para pre y postproceso de datos.
- Las partes que lleven tiempos grandes se pueden hacer en lenguaje compilado.

¿Por qué usar lenguaje modular y librerías?

• Muchas tareas distintas requieren enfoques de cálculo o proceso análogos.
• Algunas tareas difieren solo en el subset de datos.
• Cálculos de operaciones comunes.
(Fast Fourier transforms, basic linear algebra, etc.)
• Los datos pueden colocarse en archivos estructurados soportados por herramientas de análisis y visualización comunes.

¿Por qué usar lenguaje modular y librerías?

- Gran potencial de re-uso de código!!!
- Los módulos independientes pueden ser validados más fácilmente.
- Provee niveles de abstracción.
  - No es necesario saber cómo funciona TODO siempre.
  - Oportunidad para hacer más clara la optimización.
- Acceso organizado a los datos.
  - datos+funciones para modificarlos.
  - control de acceso de solo lectura o lectura-escritura.

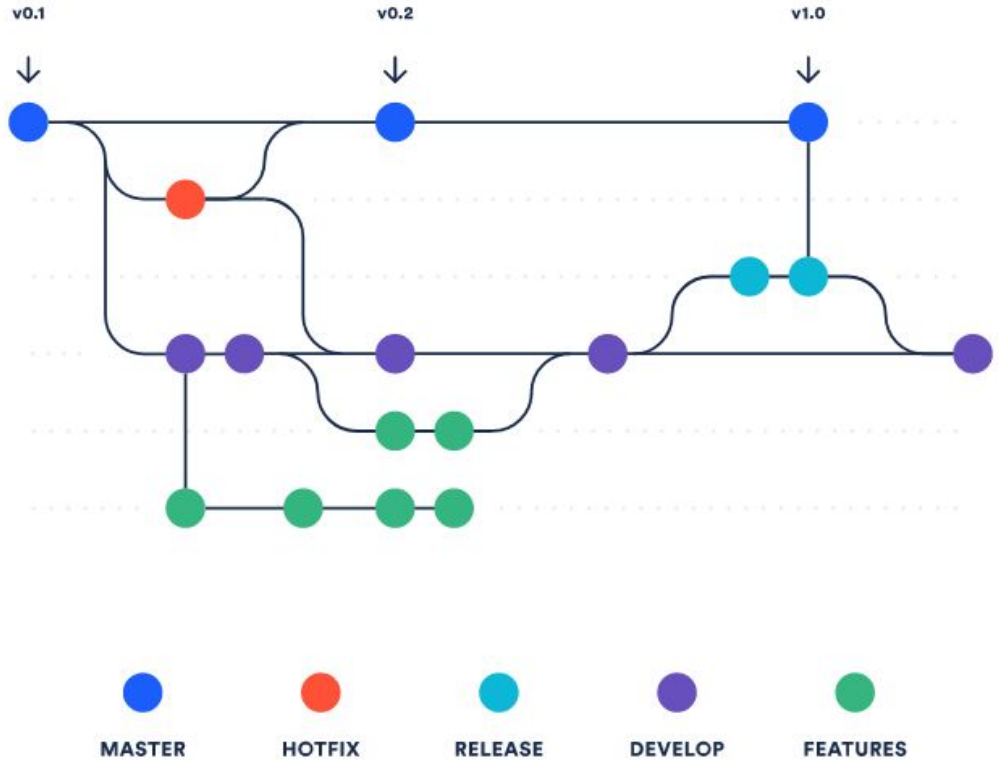
Tipos de documentación se necesitan

- Información para los desarrollares que quieren agregar código.
- Documentación API (e.g. via doxygen).
- Comentarios en el código que explican elecciones.
- Información para los usuarios:

  - Manual de referencia para los usuarios.
  - Información para los usuarios que quieren aprender a usar una
  - herramienta específica.
  - Tutoriales, HOW TO?

Arrancar a escribir la documentación desde el comienzo!

Herramientas de control de versiones



Herramientas de control de versiones

- Un lugar para colocar los archivos fuente y una herramienta comunicación entre los desarrolladores.
- Un modelo de código distribuido entre los desarrolladores.
- Trabajar con varias ramas y combinar luego.
- Hacer los cambios de a poco y no combinar cambios que no se relacionan en un mismo commit.
- Tener una documentación consistente de los cambios y las reglas para realizarlos.
- Ejemplos: Fossil; Mercurial; GIT y otros.

Conclusiones

- Intentar quitar malos hábitos de programar solo.
- Adoptar herramientas tecnológicas útiles que pueden hacer más exitosa la tarea del desarrollo colaborativo.
- O al menos poder hacer crosschek de nuestro trabajo con otros!