

SCRUM Developer

Manual del alumno

Mayra Liliana Castorena Luna, Alejandro Garcia Fdz, Jose Arturo Mora Soto y Arturo Lagunas Inocencio

© 2015 - 2016 Mayra Liliana Castorena Luna, Alejandro Garcia Fdz, Jose Arturo Mora Soto y Arturo Lagunas Inocencio

Índice general

1 ScrumBut	1
1.1 ¿Vale la pena utilizar SCRUM?	2
1.2 Discrepancia entre el modelo y el problema	3
2 Integración Continua	6
2.1 Ciclo de vida de la integración continua	7
2.2 Ventas y desventajas de la integración continua	8
3 Vagrant y maquinas virtuales	9
3.1 Instalación de Vagrant en Windows	9
3.2 Instalación de Vagrant en Linux	9
3.3 Instalación de Vagrant en Mac OS	9
3.4 Instalación de VirtualBox en Windows	10
3.5 Instalación de VirtualBox en Linux	10
3.6 Instalación de VirtualBox en Mac OS	10
3.7 Configuración de Putty.	13
4 Introducción al lenaguaje Python	17
Instalación	17
5 Comentarios útiles	18
6 Doctest: Ejercicio	20
7 Retrospectiva del día 1	22
8 Repositorio de código Git	24
8.1 instalacion de Git	27
9 Introducción a Gogs	28
10 Estándar de código	33
10.1 Python Enhancement Proposals (PEP8).	34
11 Pretty Printers: Autopep8	36

ÍNDICE GENERAL

12 Introducción a Jenkins	38
13 Complejidad Ciclomática	42
14 Refactoring	44
15 Unit Test	47
16 Code Coverage	49
17 Gitignore	50
18 Pre-commit hooks	52
19 Ejercicio Generación de tickets: Crear aplicación con Flaskr	54
20 Tarea: investigación de las herramientas vistas con los lenguajes: Java, PHP, C++, .Net, R, Ruby.	56
21 Retrospectiva del día 3	57
22 Pruebas de interfaz de usuario: Selenium IDE	60
23 Seguridad: SQL Injection, OWASP	63

1 ScrumBut

La siguiente imagen muestra el ciclo de vida del marco de trabajo conocido como SCRUM. Se dará un repaso para recordar el ciclo de vida.

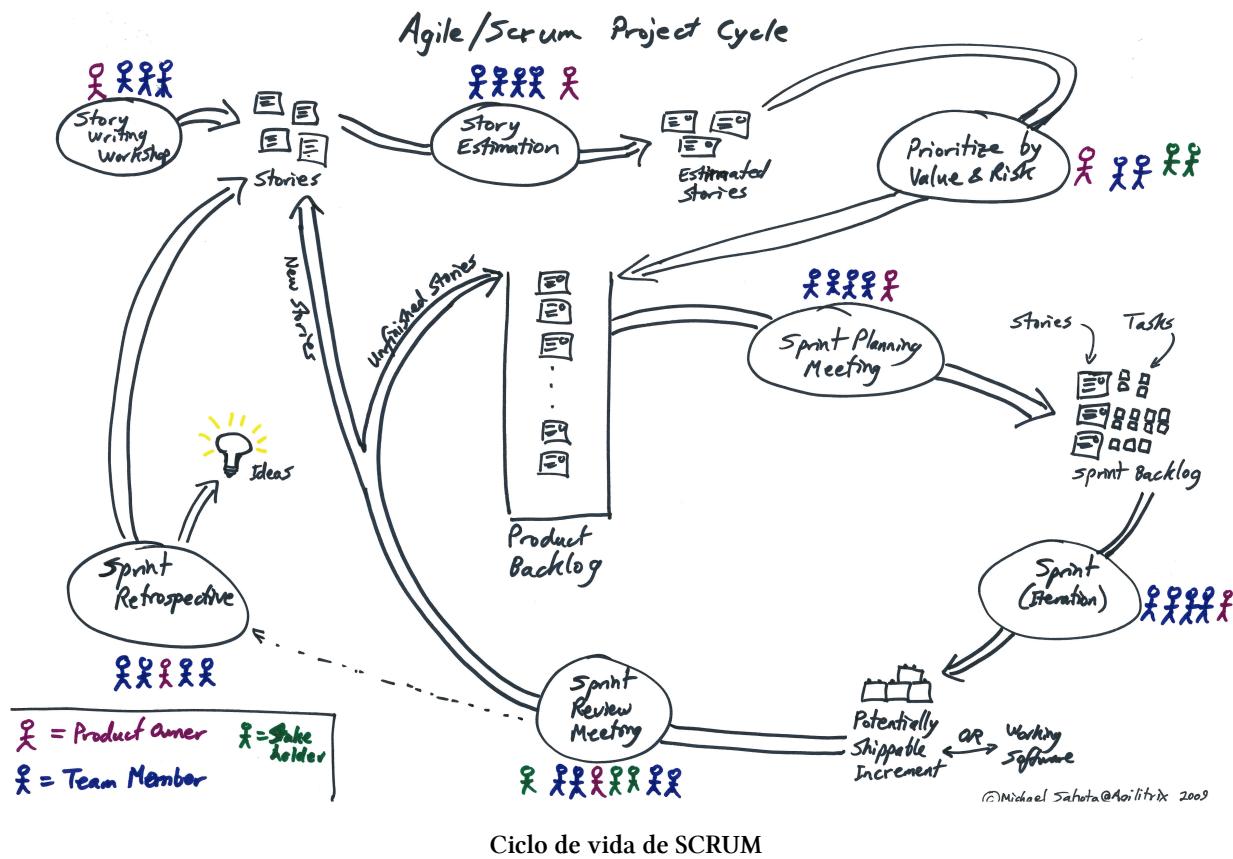


Figura 1.1: Ciclo de vida de SCRUM

Los principales elementos de la metodología SCRUM son lo siguientes:

1. El Equipo Scrum
 - El propietario del producto (Product Owner)
 - El Equipo de Desarrollo (Developers)
 - El Scrum Master
2. Eventos de SCRUM
 - La iteración (Sprint)
 - Planificación de la iteración (Sprint Planning)

- Reunión diaria de SCRUM (Daily Scrum)
 - Revisión de la iteración (Sprint Review)
 - Retrospectiva del Sprint (Sprint)
3. Artefactos de Scrum
 - Product Backlog
 - Sprint Backlog
 - Incremento
 4. Transparencia de Artefacto
 - Definición de “Hecho” (Done)

El hecho de que un equipo de trabajo desarrolle software haciendo uso de algunos de los elementos de SCRUM no significa que lo esté llevando a cabo correctamente, es ese el momento en el que lo que realmente se hace es un *SCRUM-But*. Scrum-But se refiere a las razones por las cuales el equipo no puede sacar el máximo provecho de SCRUM para resolver sus problemas y hacer realidad los beneficios del desarrollo ágil de software utilizando Scrum, tal y como lo refieren la [Scrum Alliance¹](#) y [Scrum.org²](#).

Algunas de las justificaciones que suelen escucharse en los equipos de desarrollo que hacen *SCRUM-But* son:

- “Nosotros utilizamos Scrum, pero... ¿Quién tiene un Daily Scrum todos los días? es demasiado!. Tan sólo tenemos uno por semana.”
- “Nosotros utilizamos Scrum, pero... Las retrospectivas son una pérdida de tiempo. Por lo que no hacemos.”
- “Nosotros utilizamos Scrum, pero... No podemos construir una pieza de funcionalidad en un mes. Nuestros Sprints son 6 semanas de duración.”
- “Nosotros utilizamos Scrum, pero... A veces nuestros gerentes nos dan tareas especiales. Por lo que no siempre tenemos tiempo para cumplir con nuestra definición de “Done”.”

1.1 ¿Vale la pena utilizar SCRUM?

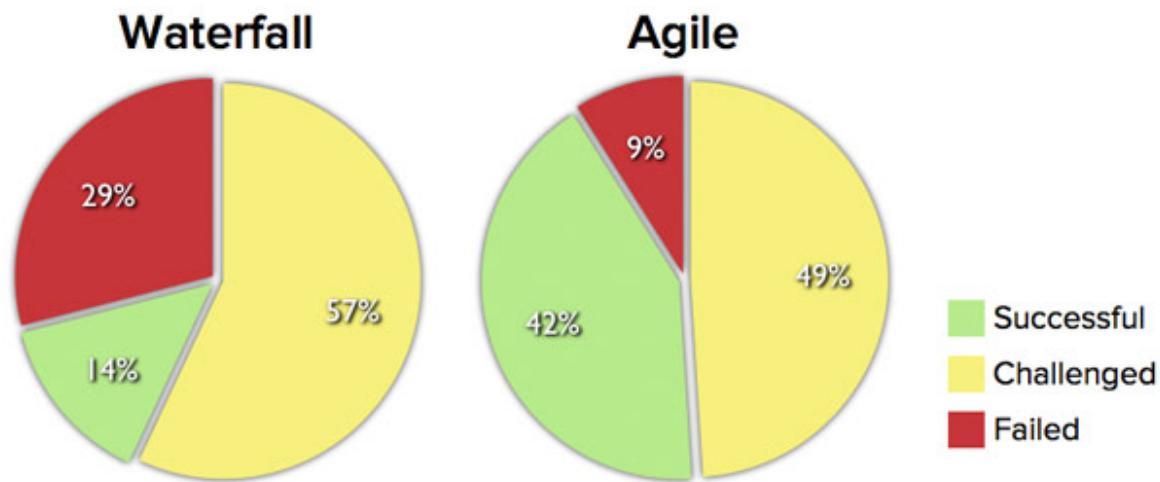
Según los indicadores del [Chaos Report 2012 de Standish Group³](#), el 86% de los proyectos desarrollados bajo los métodos tradicionales son modificados o fracasan.

En contraste el mismo estudio recalca que con el uso de métodos ágiles, tan solo el 58% de los proyectos desarrollados son modificados o fracasan tal y como se puede observar en la siguiente gráfica.

¹<https://www.scrumalliance.org/community/articles/2013/february/you-may-be-a-scrum-but>

²<https://www.scrum.org/ScrumBut>

³<https://www.standishgroup.com/>



Source: The CHAOS Manifesto, The Standish Group, 2012.

Graficas

Figura 1.2: Comparativa entre el porcentaje de éxito y fracaso del método en cascada y los métodos ágiles

El método de cascada es un malentendido desde el principio. Fue supuestamente “inventado” por el diario “Gestion de desarrollo de sistemas de Software grandes” por el Dr. Winston W. Royce en 1970, pero la implementación que se ha descrito anteriormente es riesgoso e invita al fracaso.

En este punto cabe hacerle cierta justia al método de cascada, el cuál fue formalmente presentado por primera vez en 1970 por el Dr. Winston W. Royce, quien en su artículo “[Managing the Development of Large Software Systems](#)”⁴ aclara que lo que presenta es un modelo erróneo y no necesariamente funcional, sin embargo, fue tomado por la industria del software como un estándar de facto, lo cuál, ha provocado los altos porcentajes de fallo y fracaso en proyectos de software presentados anteriormente.

1.2 Discrepancia entre el modelo y el problema

¿Es un bebé una versión más pequeña de un adulto?, la respuesta es no, ya que un bebé no es un adulto pequeño, un bebé es un ser independiente. Por lo tanto creer que un bebé es un adulto en forma pequeña o de menor tamaño es algo erróneo. Las empresas como la NASA, IBM del departamento de la defensa crearon sus propios modelos de calidad como CMM para sus empresas, las cuales, están constituidas por un gran equipo de trabajo, manejan al rededor de 800 mil trabajadores, por lo cuál se cataloga como una empresa grande, al igual que IBM.

⁴http://leadinganswers.typepad.com/leading_answers/files/original_waterfall_paper_winston_royce.pdf



Adulto pequeño

Figura 1.3: Pintura renacentista de un bebé con forma de adulto.

Entonces ¿porqué se pretende adaptar procesos que fueron desarrollados para manejar grandes organizaciones a empresas mucho más pequeñas? segun datos oficiales cerca del 90% del desarrollo de software en países como México y España se lleva a cabo por pequeñas y medianas empresas (PyMEs) o incluso micro-pymes de dos o tres personas, las cuales, no cuentan ni con la mitad de los trabajadores pertenecientes a las ya mencionadas, es lo mismo que creer que un bebé es un adulto

pequeño. Es por ello que estas empresas necesitan adaptar “macro procesos” a sus necesidades y no las “empresas a los procesos”. Es aquí donde el uso de métodos ágiles puede marcar una diferencia.

2 Integración Continua

En palabras de [Martin Fowler⁵](#), unos de los [principales exponentes del desarrollo ágil⁶](#), la **integración continua** es una práctica de ingeniería de software donde los miembros de un equipo integran su trabajo de manera frecuente, usualmente una persona realiza una tarea de integración diariamente. Cada integración es verificada mediante una compilación automatizada, que incluye las pruebas, con la finalidad de detectar errores de integración tan pronto como sea posible. El uso de esta práctica ha demostrado que ayuda a reducir significativamente los problemas de integración y permite a los equipos de desarrollo construir software cohesivo con mayor rapidez.

Uno de los conflictos que se presentan al no realizar integraciones continuas en los proyectos, es la ruptura del sistema al realizar e implementar cambios que aún no son probados; la falta de integración continua genera inestabilidad en el código además de un ciclo de retroalimentación muy largo, por mencionar solo algunos de los problemas derivados de la falta de uso de esta importante práctica de ingeniería de software.

⁵<http://www.martinfowler.com/aboutMe.html>

⁶<http://www.agilemanifesto.org/>

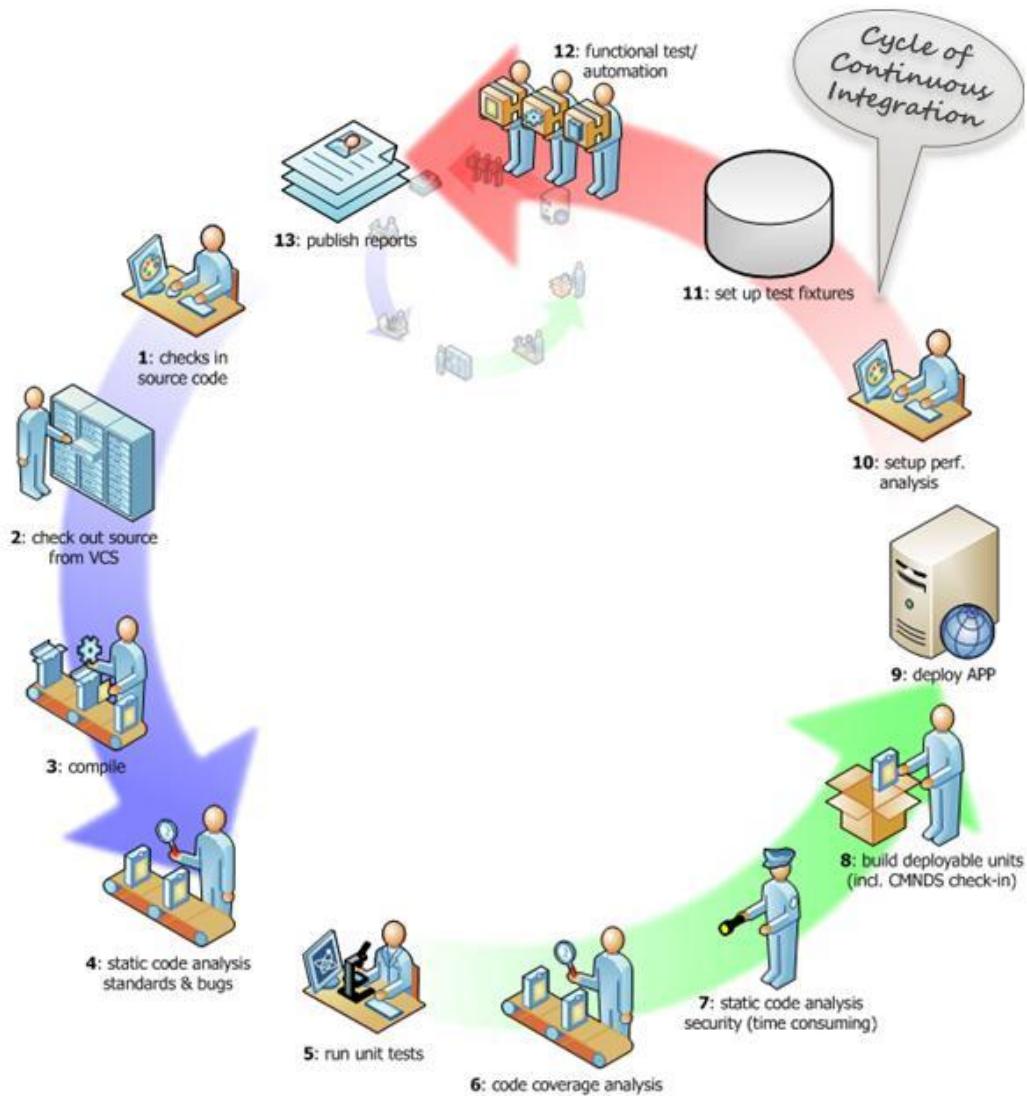


Figura 2.1: Ciclo de vida de la integración continua

2.1 Ciclo de vida de la integración continua

Tal y como se muestra en la Fig. 2.1, el ciclo de vida de esta práctica de ingeniería de software es un proceso continuo, y que termina, hasta que el software alcanza una madurez en su integración lo suficientemente buena para que el producto de software desarrollado sea liberado al mercado o entregado a su usuario final. A continuación se describen brevemente cada una de las actividades del ciclo de vida de la integración continua.

2.1.1 Publicar código fuente de la máquina local

Checks in source code. ### 2.1.2 Descargar código fuente del servidor de control de versiones 2. Checks out source from VCS. ### 2.1.3 Compilar el código 3. Compile. ### 2.1.4 Realizar análisis estático de los estándares de programación y de errores 4. Static code analysis standards & bugs. ### 2.1.5 Ejecutar pruebas unitarias 5. Run unit test. ### 2.1.6 Realizar análisis de cobertura de código 6. Code coverage analysis. ### 2.1.7 Realizar análisis de seguridad estático del código fuente 7. Static code analysis security (time consuming). ### 2.1.8 Compilar unidades de código lista para distribución 8. build deployable units. ### 2.1.9 Desplegar la aplicación 9. Deploy APP. ### 2.1.10 Definir el análisis de rendimiento 10. Setup perf analysis. ### 2.1.11 Definir banco de pruebas 11. Set up test fixtures. ### 2.1.12 Realizar pruebas funcionales y de automatización 12. Functional test / automation. ### 2.1.13 Publicar reportes 13. Publish reports.

2.2 Ventas y desventajas de la integración continua

Como toda práctica en la ingeniería de software, la integración continua tiene sus ventajas y sus desventajas, a continuación en la Tabla 1 se mencionan las más relevantes.

Ventajas	Desventajas
Detectar y solucionar problemas de integración de forma continua.	Requiere compromiso de la empresa (infraestructura).
Disponibilidad constante de una versión para pruebas.	Mantenimiento de alto nivel.
Ejecución inmediata de las pruebas unitarias.	Un experto para el manejo (Build master)
Monitorización continua de las métricas de calidad del proyecto.	

Tabla 1: Ventajas y desventajas de continuous integration

A continuación se nombran algunas de las herramientas que se pueden utilizar:

- GitHub. Repositorio para control de versiones
- Travis-CI. Sistema distribuido de integración continua libre integrado con GitHub
- CPD (Continuing Professional Development)
- PMD (Project Mess Detector)
- Coverity.
- FxCop. Herramienta de análisis de código
- Jenkins. Servidor de integración continua open source

3 Vagrant y maquinas virtuales

Cuando se está desarrollando en una máquina propia que no es de la oficina y los integrantes del proyecto tienen preferencias por alguna tecnología (sistema operativo, base de datos etc), puede generar problemas de configuración y al realizar cambios en el repositorio de código, por lo cual utilizar una máquina virtual es una buena solución para evitar este tipo de conflictos de configuración.

Vagrant es una herramienta para la creación y configuración de entornos de desarrollo virtuales. Está disponible para Linux, Mac OS X, Windows, y otras plataformas.

3.1 Instalación de Vagrant en Windows

Descargar el paquete de instalación [aquí⁷](#) y ejecutarlo, esta instalación no difiere a las instalaciones normales de Windows, se siguen los pasos que aparecerán en la pantalla. Es necesario descargar la Version 1.7.

NOTA: Descargar Putty [aquí⁸](#) si el sistema operativo es Windows. Putty no se configura en la instalación, se ejecuta hasta que utilizamos el comando “vagrant up” (que será un poco más adelante) y se realiza la configuración.

3.2 Instalación de Vagrant en Linux

Para la instalación de Vagrant desde consola teclear el siguiente comando:

```
1 `sudo apt-get install vagrant`
```

Para desechar la máquina virtual cruda, salir de la sesión SSH y ejecuta el siguiente comando:

```
1 `vagrant destroy`
```

3.3 Instalación de Vagrant en Mac OS

Para mayor información sobre la máquina virtual Vagrant para MAC OS pueden consultar los siguientes links: [Enlace 1⁹](#), [Enlace 2¹⁰](#)

⁷<https://www.vagrantup.com/downloads.html>

⁸<http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html>

⁹<https://www.virtualbox.org/wiki/Downloads>

¹⁰https://www.virtualbox.org/wiki/Linux_Downloads

3.4 Instalación de VirtualBox en Windows

Para realizar la instalación descarga el paquete en la versión 5.0.4 y seguir los pasos normalmente en la siguiente liga: [Enlace 3¹¹](#)

Instalación de VirtualBox

NOTA: VirtualBox no corre en Windows con la versión 4.3, descargar la versión 5.

3.5 Instalación de VirtualBox en Linux

Se puede realizar la descarga del paquete en el siguiente link: [Enlace 1¹²](#), [Enlace 2¹³](#)

Ó realizar la instalación desde consola con el siguiente comando:

¹ `sudo apt-get install virtualbox`

```
mayra@mayra-300E4C-300E5C-300E7C:~$ sudo apt-get install virtualbox
[sudo] password for mayra:
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
virtualbox ya está en su versión más reciente.
0 actualizados, 0 se instalarán, 0 para eliminar y 198 no actualizados.
mayra@mayra-300E4C-300E5C-300E7C:~$
```

Correcta instalacion de virtualBox

3.6 Instalación de VirtualBox en Mac OS

Nota: Instalacion aun pendiente

¹¹<https://www.virtualbox.org/wiki/Downloads>

¹²<https://www.virtualbox.org/wiki/Downloads>

¹³https://www.virtualbox.org/wiki/Linux_Downloads

Después de haber realizado la instalación de Vagrant y VirtualBox desde la consola realizar el siguiente ejercicio:

- Crear una carpeta que se utilizará para desarrollar los ejercicios del curso en la ruta donde regularmente se trabaja.

```
mkdir scrum_developer
```

```
mayra@mayra-300E4C-300E5C-300E7C:~$ mkdir scrum_developer
mayra@mayra-300E4C-300E5C-300E7C:~$ ls
AdobeAIRInstaller.bin          Música
apuntes machine learning.odt   otra
curso1cimat                   out.ogv
curso2cimat                   Plantillas
Descargas                      Público
Documentos                     repitorioexpo
ejercicioflask                 scrum_developer
electrotecniapp                tweetawatt
Escritorio                     Vagrantfile
examples.desktop                Vídeos
getskype-linux-beta-ubuntu-32  VirtualBox VMs
Imágenes                        wordpress
latest.tar.gz                 xampp-linux-x64-5.6.8-0-installer.run
mozilla.pdf
mayra@mayra-300E4C-300E5C-300E7C:~$ █
```

- El instructor proporcionará un archivo con el nombre “python.box” colocar dentro de la carpeta que se acaba de crear **scrum_developer**.

```
mayra@mayra-300E4C-300E5C-300E7C:~$ cd scrum_developer/
mayra@mayra-300E4C-300E5C-300E7C:~/scrum_developer$ ls
python.box
mayra@mayra-300E4C-300E5C-300E7C:~/scrum_developer$ █
```

- Ejecuta el siguiente comando para que Vagrant reconozca que existe una máquina virtual.

```
vagrant box add developer developer.box
```

```
mayra@mayra-300E4C-300E5C-300E7C:~/scrum_developer$ vagrant box add developer developer.box
==> box: Box file was not detected as metadata. Adding it directly...
==> box: Adding box 'developer' (v0) for provider:
    box: Unpacking necessary files from: file:///home/mayra/scrum_developer/developer.box
==> box: Successfully added box 'developer' (v0) for 'virtualbox'!
mayra@mayra-300E4C-300E5C-300E7C:~/scrum_developer$
```

- Ejecutar la máquina virtual con el siguiente comando.

```
vagrant init developer
```

```
mayra@mayra-300E4C-300E5C-300E7C:~/scrum_developer$ vagrant init developer
A `Vagrantfile` has been placed in this directory. You are now
ready to `vagrant up` your first virtual environment! Please read
the comments in the Vagrantfile as well as documentation on
`vagrantup.com` for more information on using Vagrant.
mayra@mayra-300E4C-300E5C-300E7C:~/scrum_developer$
```

- Levantar la máquina virtual (Si tu sistema operativo es Windows, pasar al sub tema “3.7 Configuración de Putty” que se encuentra más adelante)

```
vagrant up
```

```
mayra@mayra-300E4C-300E5C-300E7C:~/scrum_developer$ vagrant up
Bringing machine 'default' up with 'virtualbox' provider...
==> default: VirtualBox VM is already running.
mayra@mayra-300E4C-300E5C-300E7C:~/scrum_developer$
```

- Realizar la conexión a la máquina virtual. Si tu sistema operativo es Windows es momento de configurar Putty ya que este SO no puede ejecutar el comando ssh.

1 `vagrant ssh`

```
mayra@maya-300E4C-300E5C-300E7C:~/scrum_developer$ vagrant ssh
Welcome to Ubuntu 12.04.5 LTS (GNU/Linux 3.2.0-68-virtual i686)

 * Documentation:  https://help.ubuntu.com/

 System information as of Tue Sep 22 16:29:12 UTC 2015

 System load:  0.15          Processes:      68
 Usage of /:   2.7% of 39.37GB  Users logged in:  0
 Memory usage: 6%           IP address for eth0: 10.0.2.15
 Swap usage:   0%

 Graph this data and manage this system at:
  https://landscape.canonical.com/

 Get cloud support with Ubuntu Advantage Cloud Guest:
  http://www.ubuntu.com/business/services/cloud

 New release '14.04.1 LTS' available.
 Run 'do-release-upgrade' to upgrade to it.
```

3.7 Configuración de Putty.

Configurar Putty de la siguiente manera:

1. Desde la consola teclear el comando:

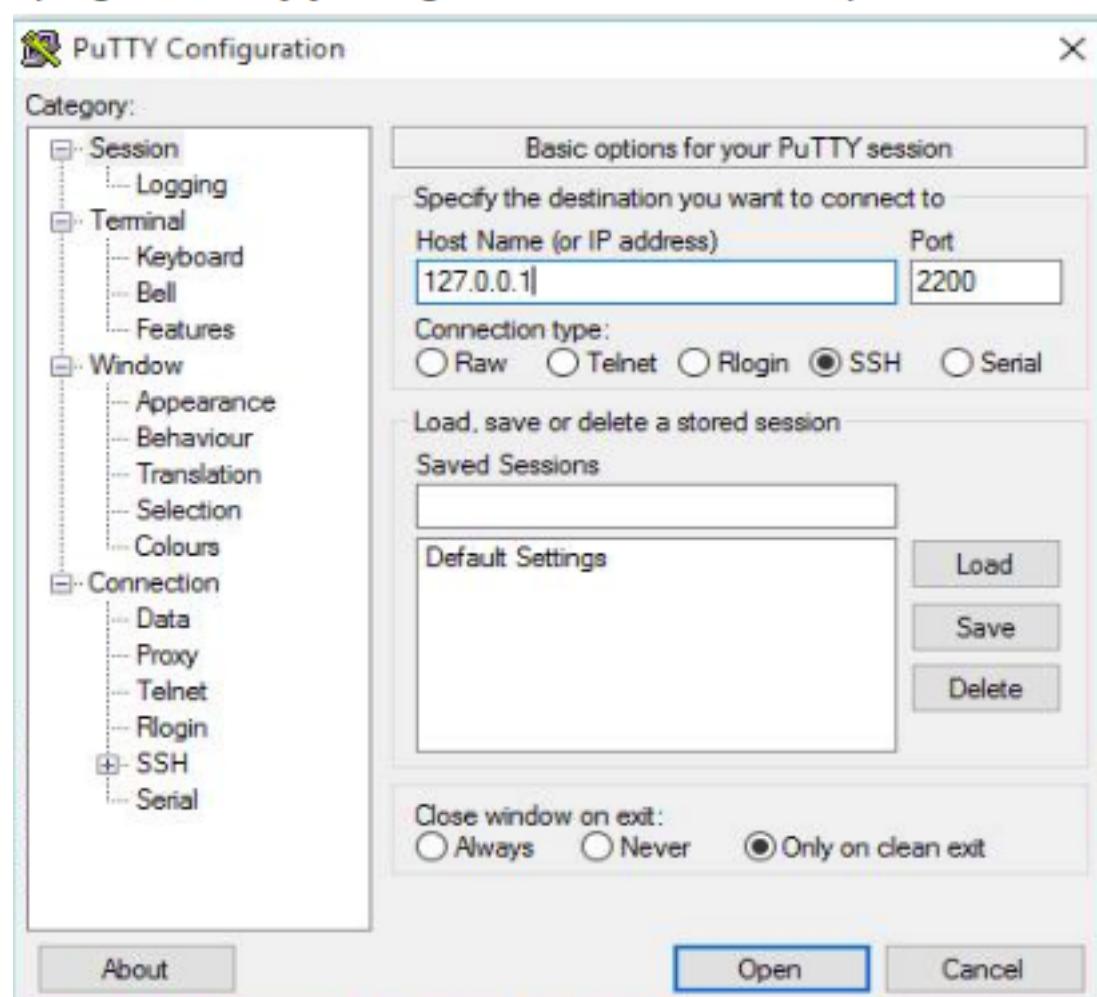
```
vagrant up
```

Te mostrará la siguiente pantalla:

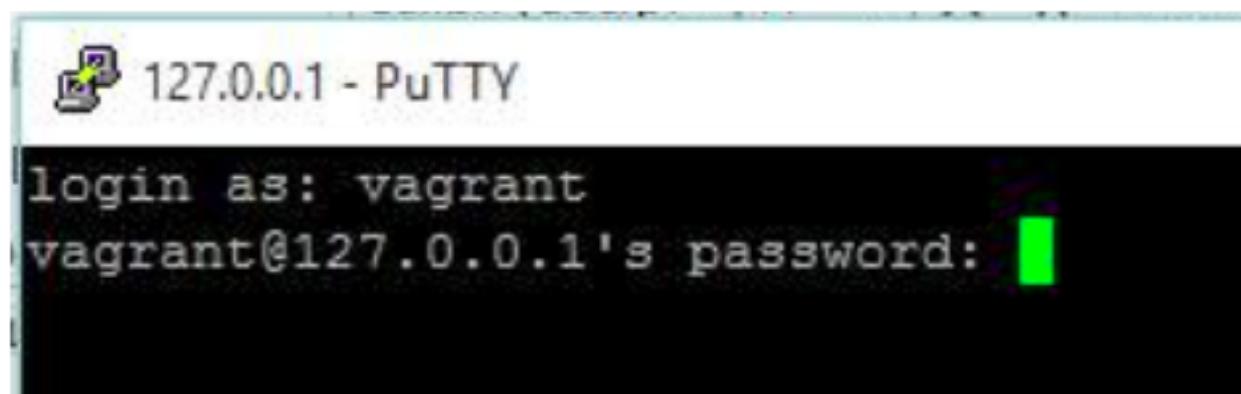
```
C:\Users\User\Videos\SCRUM\scrum>vagrant up
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Resuming suspended VM...
==> default: Booting VM...
==> default: Waiting for machine to boot. This may take a few minutes...
    default: SSH address: 127.0.0.1:2200
    default: SSH username: vagrant
    default: SSH auth method: private key
==> default: Machine booted and ready!

C:\Users\User\Videos\SCRUM\scrum>
```

1. Ejecutar el programa Putty y configurar el Host Name como aparece en SSH Address en la imagen anterior.



3.-Con los datos ingresados en Putty, te pedirá usuario: vagrant y contraseña: vagrant.



4.-Si la configuración fue exitosa te aparecerá lo siguiente en la consola.

```
vagrant@vagrant-ubuntu-precise-32: ~
Welcome to Ubuntu 12.04.5 LTS (GNU/Linux 3.2.0-68-virtual i686)

 * Documentation:  https://help.ubuntu.com/
 
 System information as of Thu Sep 24 16:25:53 UTC 2015

 System load:  0.0          Processes:      66
 Usage of /:   2.7% of 39.37GB  Users logged in:    0
 Memory usage: 6%           IP address for eth0: 10.0.2.15
 Swap usage:   0%           IP address for eth1: 192.168.0.53

 Graph this data and manage this system at:
   https://landscape.canonical.com/
 
 Get cloud support with Ubuntu Advantage Cloud Guest:
   http://www.ubuntu.com/business/services/cloud

 New release '14.04.1 LTS' available.
 Run 'do-release-upgrade' to upgrade to it.

Last login: Wed Sep 16 20:40:18 2015 from 10.0.2.2
vagrant@vagrant-ubuntu-precise-32:~$
```

- Entrar a la raíz de la máquina virtual (lo que se guarde en este directorio, se guardará tanto en local como en la máquina virtual).

```
cd /vagrant
```

```
vagrant@vagrant-ubuntu-precise-32:~$ cd /vagrant
vagrant@vagrant-ubuntu-precise-32:/vagrant$
```

En la siguiente tabla se mencionan los comandos que se utilizan en este tema (no van en el orden que se ejecutan)

Definición	Comando
Crear archivo central para la configuración del proyecto	\$mkdir nombre_de_la_carpeta
Entrar a la carpeta	\$cd nombre_de_la_carpeta
Crea el archivo de configuración de Vagrant	\$vagrant init nombre_de_la_maquina_virtual
Levantar el ambiente virtual	\$vagrant up
Para realizar la conexión a la máquina virtual (en Windows la conexión se realiza desde Putty, checar nota al final del tema)	\$vagrant ssh

Definición	Comando
Cerrar	\$exit
Apagar la máquina virtual	\$vagrant halt
Vagran reconozca que existe un archivo que define una máquina virtual	\$vagrant box add nombre_de_la_maquinavirtual scrum_developer.box
Lo que se ejecute con este comando se aplicará en la máquina virtual y en la máquina física	\$/vagrant <>
Entrar al directorio raíz	cd /vagrant
Devuelve la ruta en la que se está situado	\$pwd
Descargar archivo desde un servidor	\$vagrant box add scrum_developer http://10.13.4.7\0:8000/scrum_- developer.box
Muestra la versión instalada	python –version

Tabla: Comandos a utilizar en Vagrant.

4 Introducción al lenguaje Python

Se necesita un lenguaje que todos los integrantes del equipo puedan aprender en poco tiempo, el cual debe ser el mismo para todos, que sea compatible con las herramientas de calidad que se utilizarán, sencillo de aprender para cualquier programador Junior, sencillo de codificar, y compatible con diferentes sistemas operativos.

La selección del lenguaje para este curso es Python. Es un lenguaje de programación interpretado con las siguientes características: soporta programación orientada a objetos, programación funcional, es multiplataforma, de código abierto, sencillo debido al uso de la semántica, la sintaxis y su portabilidad.

¿Cómo se hace?

Instalación

- Linux en la versión 2.7

```
sudo apt-get install python2.7
```

- MAC OS

Python versión 2.7.9 - 2014-12-10 [Descargar¹⁴](#)

- Windows [Descargar¹⁵](#)

¿Cómo verificar la instalación?

Entrar a ms-dos y ejecutar:

```
1 `python`
```

¿Cómo saber qué versión de python está instalada?

Ejecutar:

```
1 `python --version`
```

Ejercicio: Entrar a python en la máquina virtual para realizar los ejercicios que se encuentran en el siguiente [manual¹⁶](#)

¹⁴[MacOSX64-bit/32-bitinstaller](#)

¹⁵<https://www.python.org/downloads/>

¹⁶<http://learninyminutes.com/docs/python/>

5 Comentarios útiles

Uno de los principios básicos de los desarrolladores de Software es:

“Todas las funciones deben ser documentadas”.

Habrá quien esté de acuerdo con esto y habrá quienes no, tu ¿Estas de acuerdo?, tu, ¿Documentas tus funciones?. Pues este principio es algo polémico ya que hay quienes dicen que:

“La mayoría de los comentarios son una forma de duplicación...”

¿Por qué se dice esto?

En la siguiente imagen, se puede observar cómo la documentación se duplica con el comentario de la función que se codificó:

```
/**  
 * 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;  
    }  
}
```

Ejemplo de duplicación de comentarios

¿Cómo hacer comentarios útiles y evitar la duplicación?:

Para evitar la duplicación y generar comentarios útiles debemos responder a las preguntas que se muestra en la siguiente imagen.

Who: ¿Quién hizo este código? el comando git blame nos dice qué persona hizo el código.

What: ¿Qué hace esta función? o esta clase? Esto nos lo debe decir el nombre de la función.

Where: El IDE nos dice en qué lugar es utilizada la función.

Why: El porqué o para qué sirve la función, es lo que se pone en el comentario.

How: El código dice como lo hace.



Preguntas a responder para hacer un comentario útil

Estas preguntas nos ayudan a realizar comentarios útiles, sobre todo si se comienza a documentar el código y no se sabe como comenzar.

6 Doctest: Ejercicio

La falta de pruebas durante el desarrollo de Software genera caos por la innumerable cantidad de errores o fallas que puede tener un sistema, programa o aplicación, sólo al realizar las pruebas salen a relucir estos errores, pero probado está, que entre más tarde se encuentre un error, más costoso es corregirlo.

Doctest: es un módulo que incluye Python que permite la realización de pruebas unitarias basada en la salida estándar que se obtienen en la shell, como parte de la documentación de una función y método se puede escribir directamente el caso de prueba.

TDD (Test Driven Development). Consiste en el desarrollo de dos prácticas:

1. Escribir las pruebas antes del código
2. Refactorización.

Se toma un requerimiento y se desarrolla una prueba, pensando en cómo debería ser la funcionalidad del método y verificar que ésta falle. A Continuación se desarrolla el código de la prueba y se verifica que la prueba pase satisfactoriamente, hecho esto, se procede a realizar refactorización para mejorar la funcionalidad del código. Con esto se verifica que el código está cumpliendo con los requisitos mediante la realización de las pruebas.

Existen herramientas que nos ayudan a ejecutar pruebas automatizadas como JUnit similar a Doctest.

En la siguiente tabla se mencionan los comandos y su descripción que se utilizaran para verificar la funcionalidad de las pruebas que se desarrollaran en el ejercicio sobre “Doctest”.

Definición	Comando
Correr las pruebas	<code>python -m doctest python_lab.py</code>
Abrir el archivo para editar	<code>nano "nombre del archivo"</code>

Tabla: Comandos para ejecutar el archivo de pruebas unitarias

Ejercicio: En el repositorio de código se encuentra el archivo con el nombre “python_lab.py” con pruebas unitarias, guardar el archivo en la carpeta del curso. Estas pruebas no están funcionando correctamente. Realizar las correcciones para que las pruebas puedan funcionar correctamente. Utiliza el comando “python -m doctest python_lab.py” para correr las pruebas y verificar cuáles estan correctas y cuáles no.

En la siguiente imagen se muestra el funcionamiento de “python -m doctest python_lab.py”

```
['Elijah', 'NOT PRESENT']
Got nothing
*****
File "python_lab.py", line 128, in python_lab.zero_sum_list
Failed example:
    zero_sum_list([-4, -2, 1, 2, 5, 0])
Expected:
[(-4, 2, 2), (-2, 1, 1), (-2, 2, 0), (-2, 0, 2), (1, -2, 1), (1, 1, -2), (2,
-4, 2), (2, -2, 0), (2, 2, -4), (2, 0, -2), (0, -2, 2), (0, 2, -2), (0, 0, 0)]
Got nothing
*****
27 items had failures:
 1 of   1 in python_lab.LofL_sum
 1 of   1 in python_lab.cartesian_product
 1 of   1 in python_lab.cubes
 1 of   1 in python_lab.dict2list
 1 of   1 in python_lab.dictionary_mapping
 2 of   2 in python_lab.divisible_by_3
 1 of   1 in python_lab.first_zero_sum_list
 1 of   1 in python_lab.intersection
 2 of   2 in python_lab.is_element_repeated
 1 of   1 in python_lab.list2dict
 1 of   1 in python_lab.list_average
 1 of   1 in python_lab.list_sum_zip
 2 of   2 in python_lab.minutes_in_weeks
 1 of   1 in python_lab.nextInts
 1 of   1 in python_lab.non_zero_sum_list
 1 of   1 in python_lab.odd_num_list
 1 of   1 in python_lab.pows_two
 1 of   1 in python_lab.predict_expresion
 1 of   1 in python_lab.range_and_zip
 2 of   2 in python_lab.reminder_without_mod
 1 of   1 in python_lab.set_product57
 1 of   1 in python_lab.set_product58
 1 of   1 in python_lab.square_dict
 1 of   1 in python_lab.squares_set
 1 of   1 in python_lab.value_list
 2 of   2 in python_lab.value_list_m
 1 of   1 in python_lab.zero_sum_list
***Test Failed*** 32 failures.
mayra@mayra-300E4C-300E5C-300E7C:~/curso1cimat/instructores$ █
```

Ejemplo del funcionamiento de python doctest

Nota: Para conocer al desarrollador de cada línea de código ejecutar el siguiente comando:

- 1 `git blame nombre_del_archivo`
- 2 `git blame python_lab.py`

7 Retrospectiva del día 1

En casi todos los proyectos de desarrollo de Software se presentan conflictos como la falta de claridad en los requerimientos, mala comunicación del equipo, selección de herramientas nuevas sin curso introductorio previo, por mencionar algunos.

La retrospectiva en la ingeniería de Software nos ayuda a identificar esos conflictos y a buscar una solución para mejorar en los futuros proyectos.

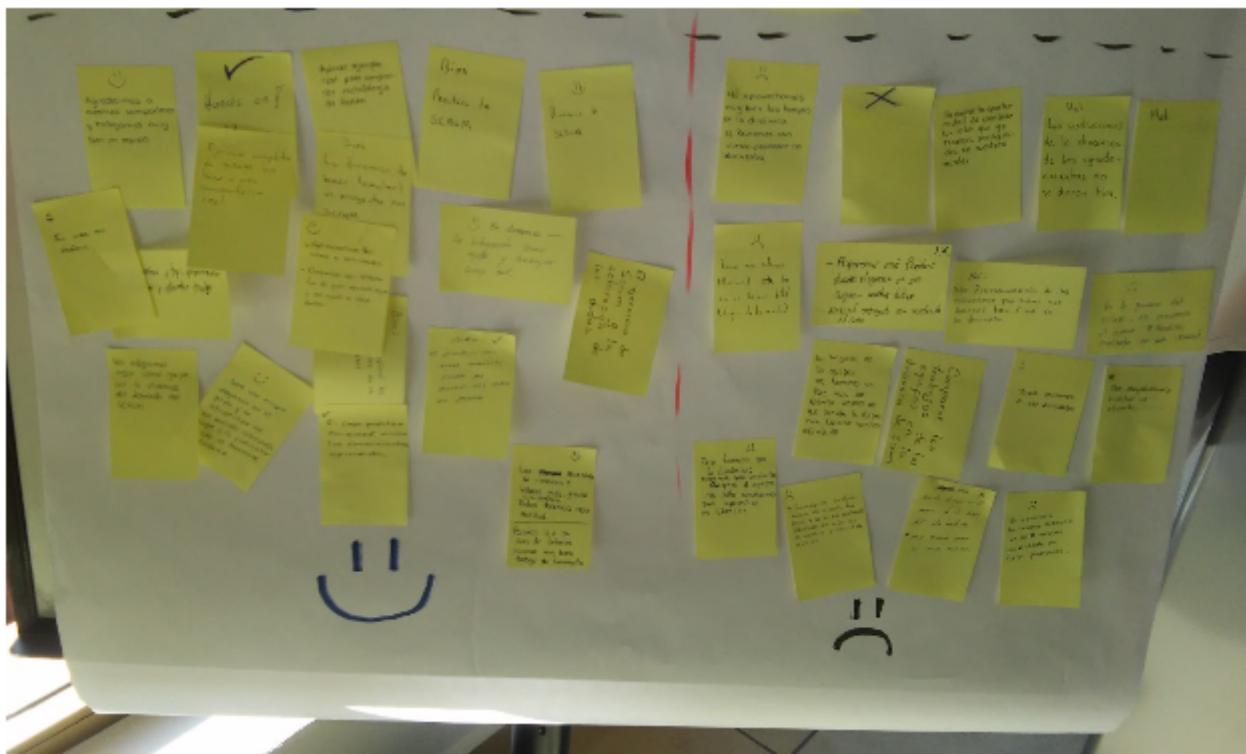
¿Cómo se hace?

La retrospectiva es una junta corta que se realiza después de la terminación de un proyecto o una iteración de proyecto, en la cuál se abordan los aciertos y errores que se han generado en el transcurso de la realización del proyecto. En la retrospectiva se propone cómo mejorar los conflictos presentados y se discute qué se está haciendo bien y se quiere seguir haciendo de la misma manera, para realizar un mejor trabajo. En un lugar visible para todos los integrantes (pizarrón, pared, ventana, cartulina) se representan dos secciones: “Qué hicimos bien y queremos repetir” y “Qué podemos/debemos mejorar”. De esta forma, se recomienda que la retrospectiva se desarrolle en torno a las siguientes preguntas:

- ¿Qué hicimos bien y queremos seguir haciendo?
- ¿Qué podemos / debemos mejorar?
- O la identificación de una propuesta

Pasos:

- Cada integrante del equipo tiene 2 post-its, uno para cada sección, pasa a ubicarlo dónde corresponde y lo lee frente al equipo.
- De la sección “Qué podemos / debemos mejorar” se toman dos propuestas entre todos los integrantes y se hace el compromiso de mejorarlos.



Ejemplo de Retrospectiva de curso en CIMAT Guanajuato Mayo 2015.

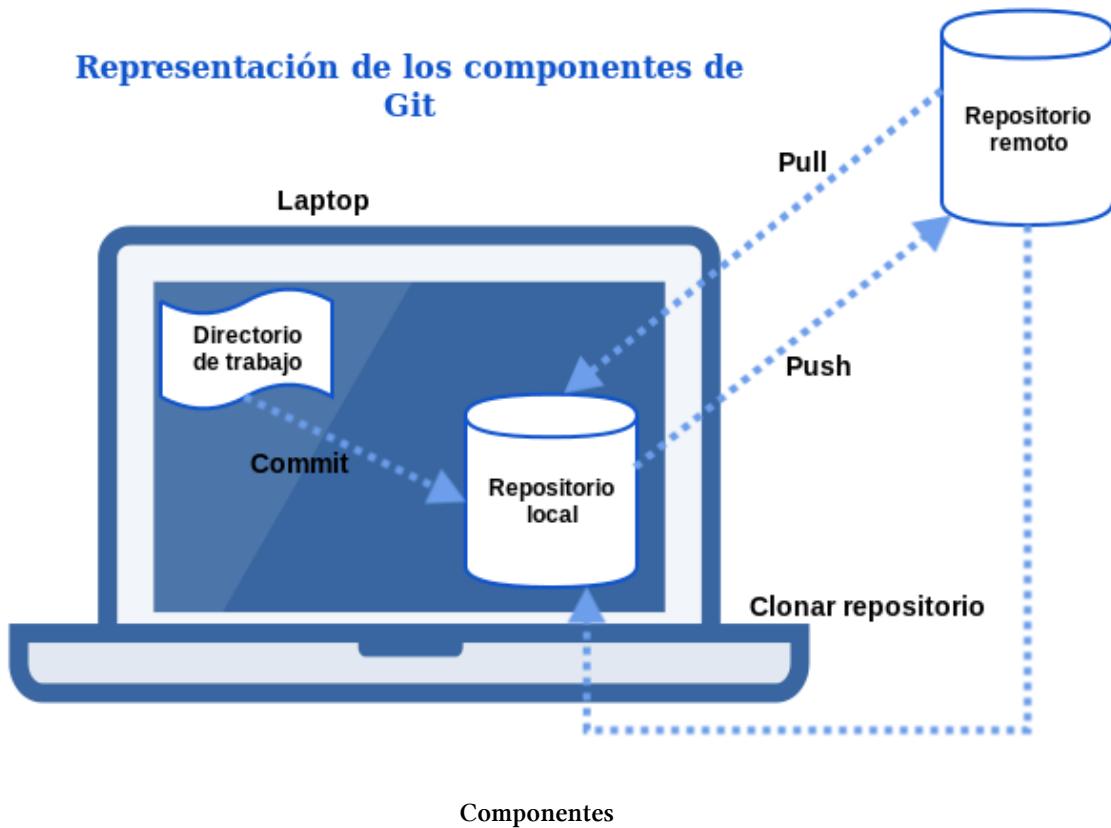
8 Repositorio de código Git

Cuando se trabaja en un proyecto de desarrollo de software, ya sea individualmente, siendo parte de un equipo de trabajo o de manera distribuida, pueden generarse problemas si no se cuenta con un repositorio de código dónde se encuentren versiones del mismo. Por ejemplo, si dos o más personas están trabajando en el mismo archivo y los dos realizan cambios a las mismas líneas de código podría generarse perdida al realizar las actualizaciones al archivo, si se está trabajando sin repositorio y toda la información se encuentra solo en una carpeta en una máquina personal, y por error se elimina ésta carpeta, se daría una pérdida completa del proyecto generando un grave problema para el desarrollador y para todo el equipo.

El repositorio de código ayuda a que el proyecto se encuentre ubicado en un lugar seguro, además se puede tener un control de las versiones, cambios realizados, fechas de modificación, fechas de última versión, el nombre del integrante que modificó el código, entre otra información relevante; si hubiera un error en alguna versión y fuera encontrado durante producción, se puede tomar una versión anterior que no tenga errores y que no haya sufrido cambios significativos y utilizarla. A continuación se aprenderá a utilizar este tipo de repositorios de código.

¿Cómo funciona Git?

La siguiente imagen muestra los componentes básicos para comprender Git.



A continuación se muestra un diagrama con el flujo del funcionamiento de Git.

Diagrama de flujo del repositorio Git

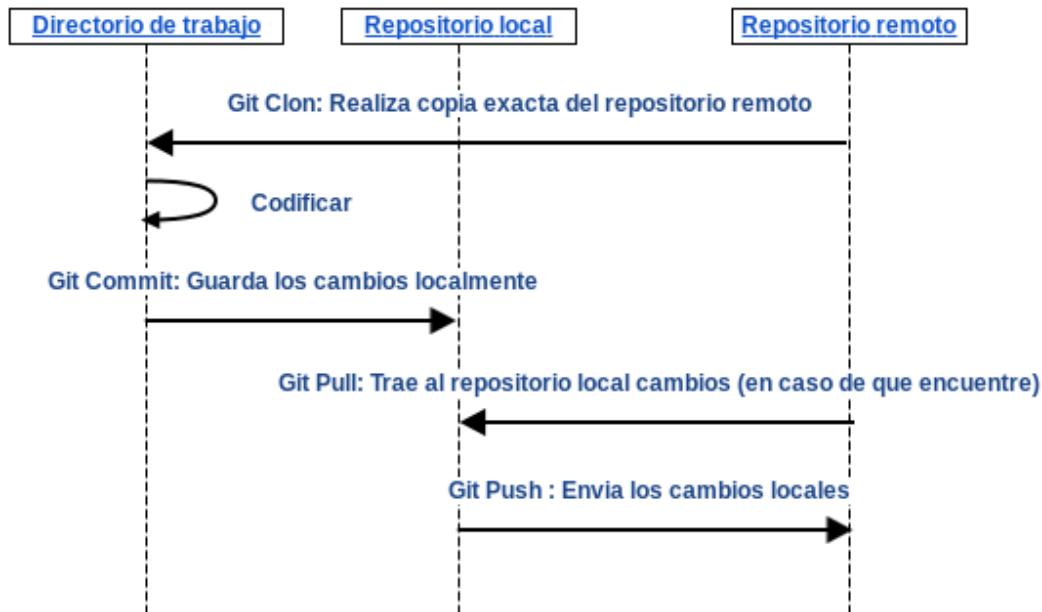


Diagrama de flujo

Comandos más utilizados

Descripción	Comandos
Para bajar por primera vez lo que se encuentra en un repositorio remoto.	git clone username@host:/path/to/repository
Bajar cambios del repositorio local	git pull
Subir cambios al repositorio Git	git push
Registrar cambios	git add nombre del archivo
Realizar un comentario al subir cambios	git commit -am "comentario"
Crea una copia local del repositorio de código	git clone http://path/to/repository
Hacer commit agregando el número de tiquet y la palabra reservada para que tome al tiquet como cerrado	git commit -am "fixes #12 header and footer"
Con "refs" se hace referencia al tiquet #12	git commit -am "refs #12 header and footer"

Existen muchos repositorios de código, uno de los más utilizados es Git. En la siguiente liga se puede

encontrar un pequeño manual que ayudará a comprender el funcionamiento de Git [aquí¹⁷](#)

8.1 instalacion de Git

8.1.1 Instalacion de Git en MAC OS

Deberá tener instalado Homebrew

```
1 `brew update`  
2 `brew install git`
```

8.1.2 Instalacion de Git en Debian/Ubuntu

```
1 `sudo apt-get update`  
2 `sudo apt-get install git`
```

8.1.3 Instalacion de Git en Windows

Descargar Git [aquí¹⁸](#) y ejecutarlo

¹⁷<https://try.github.io/levels/1/challenges/1>

¹⁸<https://git-scm.com/download/win>

9 Introducción a Gogs

La administración de un repositorio de código no es una tarea sencilla de realizar desde consola, como una solución a este problema existe una aplicación Web llamada Gogs (Go Git Service) cuyo objetivo es proporcionar un servicio para la gestión de proyectos en Git (similar a GitHub o GitLab) para servidores propios de manera más fácil, más rápida y con el menor esfuerzo posible. Esto es posible gracias a una distribución binaria e independiente de la aplicación escrita en Go que funciona en todas las plataformas soportadas por el lenguaje, incluyendo Linux, Mac OS X y Windows. Con Gogs podemos controlar Tickets, y administrar los archivos que necesitamos en nuestro proyecto así como contar con el historial de tickets.

¿Cómo se usa?

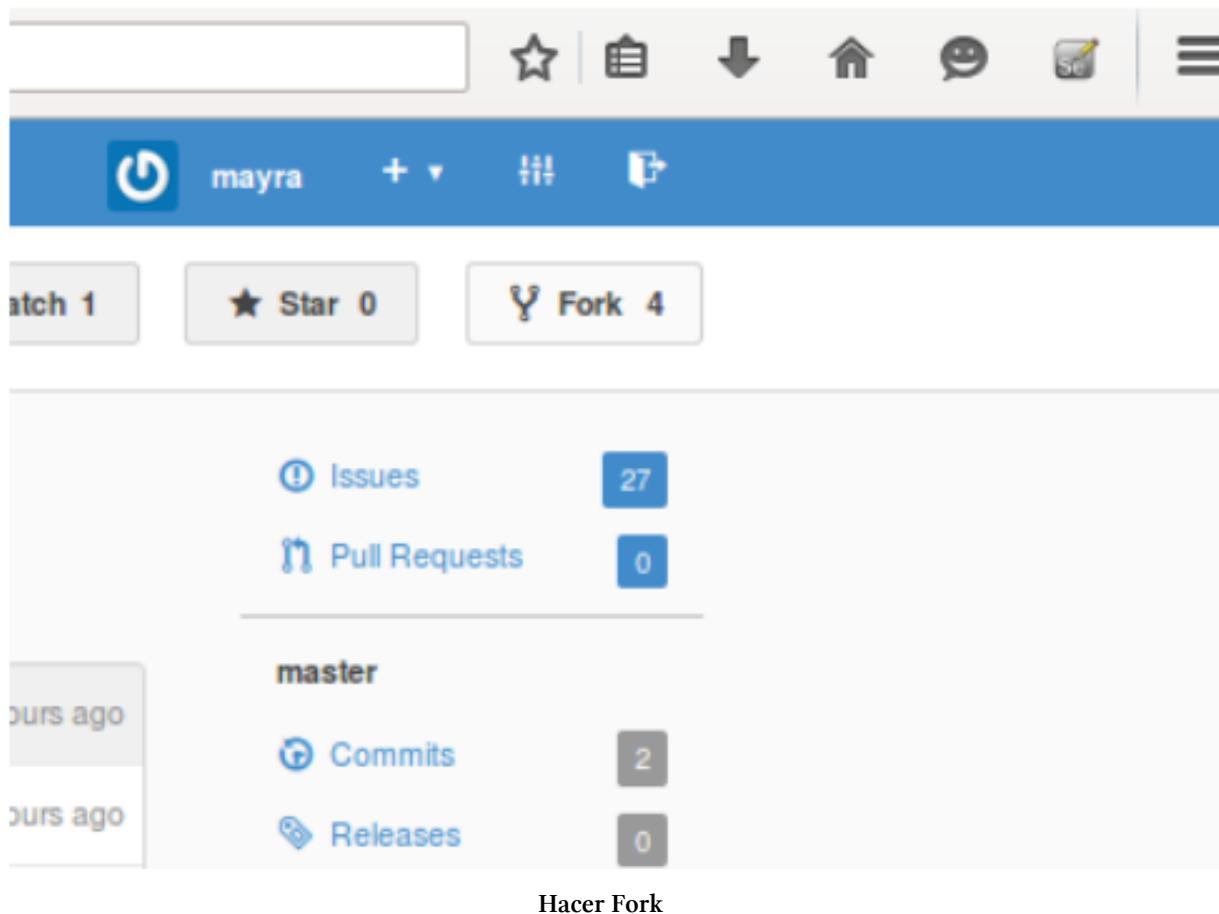
Ejercicio: El siguiente ejercicio se realizará mediante el repositorio Gogs, la interfaz es la que se puede observar a continuación:



Gogs

Pasos generales para el desarrollo del ejercicio (no son pasos específicos)

- Entrar a la url que dara el instructor, ejemplo: <http://10.13.4.78:3000/>
- Registrarse
- Reunirse en equipo, seleccionar un líder y nombre del equipo
- Solo el líder, creará un repositorio, haciendo clic en Fork, al repositorio actual (Creado por el instructor).



- En “Repository Name” asignar un nombre al nuevo repositorio (podría ser el nombre del equipo).

New Fork Repository

Owner *  mayra.castorena ▾

Fork From elviejo79/instructores

Repository Name * instructores

Visibility This repository is **Private**
You cannot alter the visibility of a forked repository.

Description Ejercicios de python_lab en equipo

Fork Repository **Cancel**

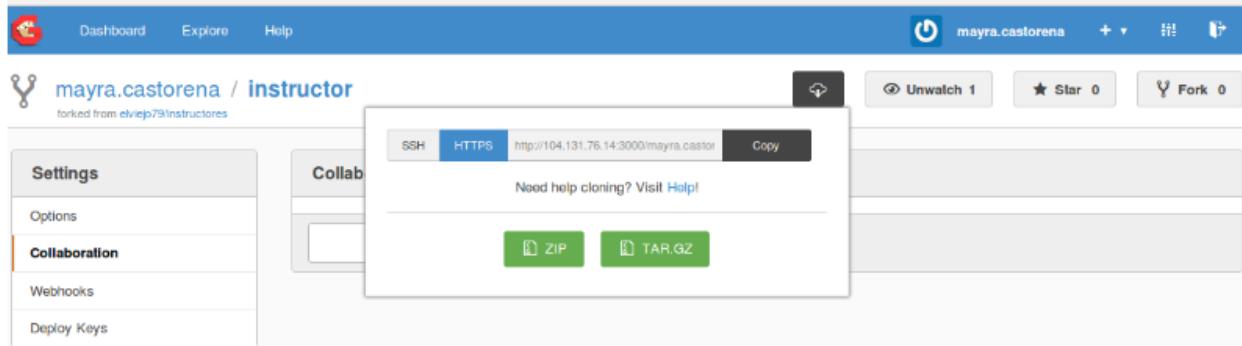
Asignar nombre

- Agregar a los compañeros de equipo como colaboradores.

The screenshot shows a GitHub repository page. At the top, there's a navigation bar with icons for Dashboard, Explore, Help, and a user icon. The repository name 'mayra.castorena / instructor' is displayed, along with a 'forked from elviejo79/instructores' link. To the right are buttons for Unwatch 1, Star 0, and Fork 0. On the left, a sidebar has 'Settings' selected under 'Collaboration'. The main area shows a 'Collaboration' section with a large input field and a 'Add New Collaborator' button.

Agregar colaboradores

- Todo el equipo hace un clon del repositorio en Vagrant desde la máquina virtual después de que el líder haya hecho el Fork.
- Copiar la url del repositorio desde Gogs.



Url

- Desde la consola entrar a /vagrant y ejecutar el comando:

git clone "pegar aqui la url del repositorio que fue copiado previamente"

```
vagrant@vagrant-ubuntu-precise-32:/vagrant/instructor$ git clone http://104.131.76.14:3000/mayra.castorena/instructor.git
Cloning into 'instructor'...
remote: Counting objects: 8, done.
remote: Compressing objects: 100% (8/8), done.
remote: Total 8 (delta 1), reused 0 (delta 0)
Unpacking objects: 100% (8/8), done.
vagrant@vagrant-ubuntu-precise-32:/vagrant/instructor$ ls
instructor  python_lab.py
vagrant@vagrant-ubuntu-precise-32:/vagrant/instructor$ cd instructor/
vagrant@vagrant-ubuntu-precise-32:/vagrant/instructor/instructor$ ls
LICENSE  python_lab.py  README.md
vagrant@vagrant-ubuntu-precise-32:/vagrant/instructor/instructor$
```

Hacer Clone

- En equipo organizarse para seguir realizando los ejercicios que se comenzaron en el tema “Doctest” de forma colaborativa con el repositorio Gogs.
- Cada uno de los integrantes deberá hacer commit, push y pull para actualizar el repositorio de código.
- Para hacer referencia a los “issue” que están en el Gogs, cada miembro del equipo deberá colocar “Refs #numero_de_issue_al_cual_se_refiere” dentro del comentario, para poder hacer referencia de la prueba al “issue” como se muestra de ejemplo en la siguiente pantalla:

```
vagrant@vagrant-ubuntu-precise-32:/vagrant/ejercicios$ git commit -am "refs #3 Solucion del ejercicio 3"  
[master 5ec935a] refs #3 Solucion del ejercicio 3  
 1 file changed, 3 insertions(+), 3 deletions(-)
```

Commit

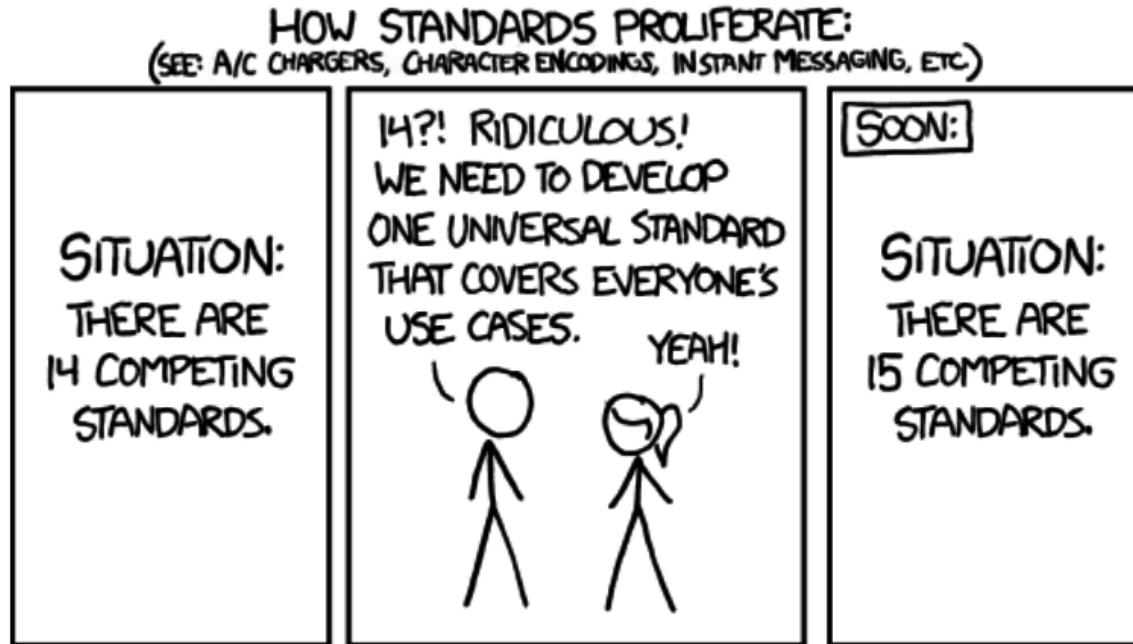
- Para cerrar el “issue” se cambiará el comando “Refs” por: “fixes #3” como se muestra en la siguiente imagen:

```
vagrant@vagrant-ubuntu-precise-32:/vagrant/ejercicios$ git commit -am "fixes #4 Solucion del ejercicio 4"  
[master 6a7439e] fixes #4 Solucion del ejercicio 4  
 1 file changed, 1 deletion(-)  
vagrant@vagrant-ubuntu-precise-32:/vagrant/ejercicios$ █
```

Commit

NOTA: Los comandos son los mismos que se encuentran en el tema “8. Repositorio de código Git”.

10 Estándar de código



By Randall Munroe

La falta de establecimiento de un estándar de código hace que sea difícil dar mantenimiento, esto complica el entendimiento de un sistema para la persona que no lo programó y disminuye la calidad del mismo. La programación con diferentes estándares de codificación en un mismo equipo de desarrollo crea conflictos a la hora de subir los cambios al repositorio.

El estándar de codificación brinda calidad a nuestro código por lo cuál es importante, sea cuál sea el lenguaje de programación en el que se esté acostumbrado a programar, ya que para todos los lenguajes de programación existe un estándar de código que se puede adoptar.

Si se está desarrollando de forma colaborativa, todos los desarrolladores deben adoptar el mismo estándar de código con lo cuál, debe parecer que el mismo desarrollador codificó todo el programa. Con un estándar de código establecido se facilitará el mantenimiento del mismo, por ejemplo, añadir nuevas características, modificar las ya existentes, depurar errores, o mejorar el rendimiento.

Por citar algunos ejemplos se mencionan los siguientes:

Identación. Usa 4 espacios por cada nivel de indentación.

¿Tabuladores o espacios? Nunca mezcles tabuladores y espacios.

Tamaño máximo de línea. Limita todas las líneas a un máximo de 79 caracteres.

No usar espacios alrededor del signo. Igual cuando se encuentre en un listado de argumentos de una función:

- Correcto: def suma(a=0, b=0):
- Incorrecto: def suma(a = 0, b = 0):

No se debe realizar comentarios obvios.

Puedes encontrar mas información sobre el estandar de código [aquí¹⁹](#)

10.1 Python Enhancement Proposals (PEP8).

Está dedicada a la recopilación de los estándares seguidos por los desarrolladores de Python a la hora de escribir código para la librería estándar, éste ejecuta las reglas automáticamente y genera un resultado de las líneas que no cumplen con el estándar de codificación.

Instalación

1 `sudo pip install flake8`

Actualización

1 `pip install --upgrade`

¿Cómo se usa?

Al ejecutar los siguientes comandos se mostraran las violaciones que estan presentes en el estandar de codigo definido por Python.

1 `flake8 Nombre del archivo.py`
2 ó
3 `pep8 nombre del archivo.py`

Donde mostrará las violaciones al estándar de código Python de la siguiente manera:

1 `pep8 ejercicio.py`

```
mayra@mayra-300E4C-300E5C-300E7C:~/otra/ejercicio$ pep8 ejercicio.py
ejercicio.py:11:80: E501 line too long (92 > 79 characters)
ejercicio.py:12:80: E501 line too long (87 > 79 characters)
ejercicio.py:13:80: E501 line too long (132 > 79 characters)
ejercicio.py:18:45: W601 .has_key() is deprecated, use 'in'
mayra@mayra-300E4C-300E5C-300E7C:~/otra/ejercicio$
```

El siguiente comando cuenta las líneas que hay con error:

¹⁹<http://mundogeek.net/traducciones/guia-estilo-python.htm>)

```
1 `flake8 python_lab.py | wc -l`
```

Ejercicio: Aplicar flake8 ó pep8 al archivo “python_lab.py” observar cuáles errores marca y corregir segun el estandar de python.

11 Pretty Printers: Autopep8

Como se mencionó en el tema anterior cumplir o adoptar un estándar de codificación beneficia a todos los desarrolladores y los problemas que podría ocasionar no seguir ningun estandar. Python cuenta con la librería Autopep8 colaborando con pep8 que indica cuáles líneas de código necesitan darse formato, Autopep8 soluciona estos errores de estándar de codificación y los corrige automáticamente. En otras palabras pep8 encuentra las líneas que necesitan ser corregidas y Autopep8 las corrige automáticamente, ahorrando al desarrollador tiempo en corregir línea por línea.

¿Cómo se usa?

Instalar desde consola

```
1 `sudo pip install autopep8`
```

Realizar la corrección del código según el estándar de python

```
1 `autopep8 --in-place nombre_del_archivo.py`
```

Ejemplo:

A continuación se muestra un fragmento de código en el cual aplicamos autopep8 para dar formato con el estándar de código de Python.

```
1 import math, sys;
2
3 def example1():
4     #####This is a long comment. This should be wrapped to fit within 72 characters.
5     some_tuple=( 1,2, 3,'a' );
6     some_variable={'long':'Long code lines should be wrapped within 79 characters.',
7                    'other':[math.pi, 100,200,300,9876543210,'This is a long string that goes on'],
8                    'more':{'inner':'This whole logical line should be wrapped.',some_tuple:[1,
9                        20,300,40000,500000000,6000000000000000]}}
10    return (some_tuple, some_variable)
11
12 def example2(): return {'has_key() is deprecated':True}.has_key({'f':2}.has_key(''));
13 class Example3( object ):
14     def __init__ ( self, bar ):
15         #Comments should have a space after the hash.
16         if bar : bar+=1; bar=bar* bar   ; return bar
17         else:
18             some_string = """
19             Indentation in multiline strings should not be touched.
20             Only actual code should be reindented.
21             """
22             return [sys.path, some_string]
```

Antes de Autopep8

Después de aplicar autopep8 este es el resultado:

```
ejercicio.py
1 import math
2 import sys
3
4
5 def example1():
6     # This is a long comment. This should be wrapped to fit within 72
7     # characters.
8     some_tuple = (1, 2, 3, 'a')
9     some_variable = {
10         'long': 'Long code lines should be wrapped within 79 characters.',
11         'other': [math.pi, 100, 200, 300, 9876543210, 'This is a long string that goes on'],
12         'more': {'inner': 'This whole logical line should be wrapped.', some_tuple: [1,
13             2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 300, 40000, 500000000, 6000000000000000]}
14     }
15
16     return (some_tuple, some_variable)
17
18 def example2():
19     return {'has_key() is deprecated': True}.has_key({'f': 2}.has_key(''))
20
21 class Example3(object):
22
23     def __init__(self, bar):
24         # Comments should have a space after the hash.
25         if bar:
26             bar += 1
27             bar = bar * bar
28             return bar
29         else:
30             some_string = """
31                 Indentation in multiline strings should not be touched.
32 Only actual code should be reindented.
33 """
34             return (sys.path, some_string)
```

Después de Autopep8

Ejercicio: Aplicar autopep8 al archivo “python_lab.py”. En caso de que aun muestre errores, corregir manualmente como en el ejercicio anterior, hasta que no marque ningun error.

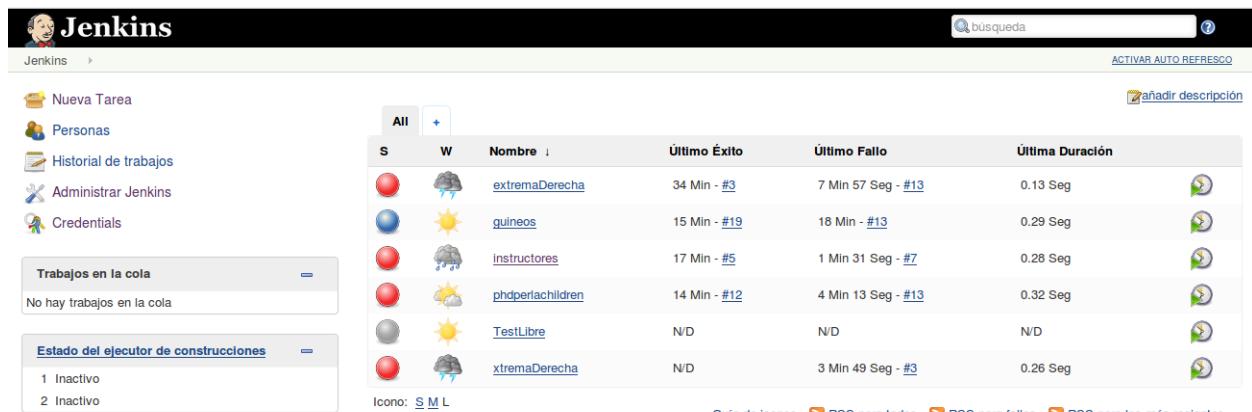
12 Introducción a Jenkins

Jenkins es una herramienta de integración continua Open Source para el desarrollo de Software. Se encuentra en ejecución en un servidor que es un contenedor de servlets, como Apache Tomcat. Soporta herramientas de control de versiones como CVS, Subversion, Git, Mercurial, Perforce, Clearcase y puede ejecutar proyectos basados en Apache Ant y Apache Maven, así como scripts de shell y programas batch de Windows.

¿Cómo se usa?

Jenkins permite configurar el tiempo de ejecución de las pruebas de integración.

Nota: Es mala idea integrar el comando de autopep8 (que se encarga de la corrección del estándar de código de Python) ya que se recomienda realizar estas correcciones antes de enviar cambios al servidor.



The screenshot shows the Jenkins dashboard with the following details:

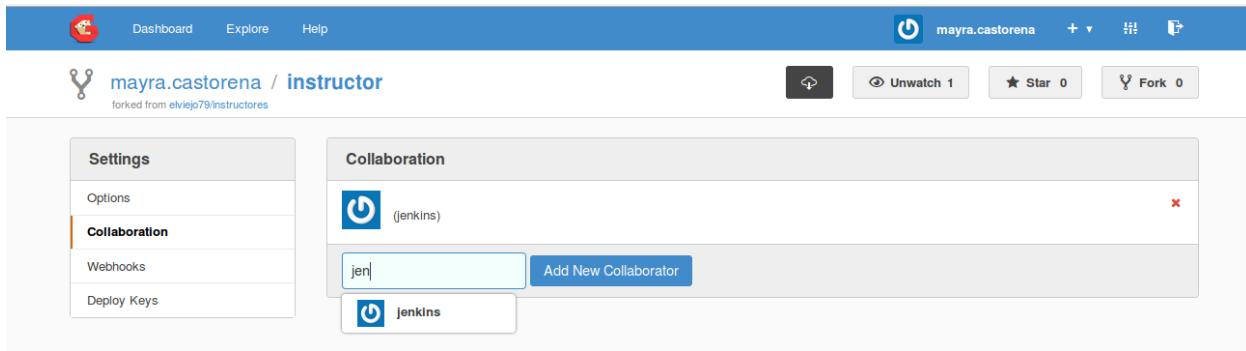
- Left sidebar:** Includes links for "Nueva Tarea", "Personas", "Historial de trabajos", "Administrar Jenkins", and "Credentials".
- Trabajos en la cola:** A section stating "No hay trabajos en la cola".
- Estado del ejecutor de construcciones:** Shows 1 Inactivo and 2 Inactivo.
- Central area:** A table listing build jobs with columns: S (Status), W (Weather icon), Nombre, Último Éxito, Último Fallo, and Última Duración.

S	W	Nombre	Último Éxito	Último Fallo	Última Duración
Red	Cloud with rain	extremaDerecha	34 Min - #3	7 Min 57 Seg - #13	0.13 Seg
Blue	Sun	guineos	15 Min - #19	18 Min - #13	0.29 Seg
Red	Cloud with rain	instructores	17 Min - #5	1 Min 31 Seg - #7	0.28 Seg
Red	Sun and cloud	phdperlachildren	14 Min - #12	4 Min 13 Seg - #13	0.32 Seg
Grey	Sun	TestLibre	N/D	N/D	N/D
Red	Cloud with rain	xtremaDerecha	N/D	3 Min 49 Seg - #3	0.26 Seg
- Bottom right:** Buttons for "Icono: S M L", "Guía de iconos", and RSS feeds for "RSS para todos", "RSS para fallas", and "RSS para los más recientes".

Panel de control de Jenkins

Ejercicio: Con ayuda del instructor configurar el servidor Jenkins.

- Desde Gogs, el líder del equipo añade un colaborador llamado “Jenkins”.



Añadir un colaborador a Gogs

- Desde Jenkins, seleccionar configuraciones y añadir en “Credentials” a Jenkins para que tenga permisos.

Configurar el origen del código fuente

Ninguno
 CVS
 CVS Projectset
 Git

Repositories Repository URL: (?)

🚫 Please enter Git repository.

Credentials: Add (?)

Avanzado...

Add Repository Delete Repository

Branches to build Branch Specifier (blank for 'any'): (?)

Add Branch Delete Branch

Navegador del repositorio: (?)

Guardar Aplicar los cambios

Creación de permisos

- En “Build triggers” indicar cada cuanto tiempo se realizará la actualización

Formato para realizar las pruebas en un periodo de tiempo.

Formato	Cada cuanto	Ejemplo	Descripción
*	Cada minuto	H/5*****	Cada 5 minutos
**	Cada hora		
***	Cada mes		
****	Cada día	* * * *0	Cada minuto, cada hora, cada mes, de todos los domingos.
		* * * * 0,1	Cada minuto, cada hora, cada mes, de todos los domingos y lunes.

En el campo “Execute shell” podemos agregar comandos que se quieren ejecutar cada determinado tiempo solo si el servidor encontró cambios, e incluso se puede indicar el directorio en la cuál se desea la ejecución del comando.

- En el campo “Execute shell” agregar el siguiente comando para que lo ejecute el servidor de manera automática:

```
python -m compileall .
```

Este comando compila todo el código a partir de esta carpeta

Ejecutar

Ejecutar linea de comandos (shell)
?

Comando

python -m compileall .

Visualizar [la lista de variables de entorno disponibles](#)
Borrar

Comando compileall

- Dar clic en “Build now” o “Construir ahora” y observa lo que sucede.



Visualización del estado del proyecto

- Añadir un nuevo comando al campo “Execute shell” Este comando nos indica si violamos el estándar de código. Para este caso asignamos un máximo de 160 caracteres por línea de código.
flake8 --max-line-length=160 .

Ejecutar linea de comandos (shell)

Comando flake8 --max-line-length=160 .

Guardar Aplicar los cambios

Comando que se asegura de evitar violaciones en el código

- Dar clic en “Build now” o “Construir ahora” y observa si tus resultados son satisfactorios.

13 Complejidad Ciclomática

Existen múltiples métricas que de manera predictiva nos ayudan a detectar áreas de nuestro código en las cuáles se presentarán defectos. Una de estas es la Complejidad Ciclomática. La cual se define formalmente como “ $CC = \text{Número de condiciones} + 1$ ”. Esta fórmula fue propuesta por McCabe como una forma simplificada y práctica para calcular la complejidad.

Con base a esta métrica podemos decidir qué funciones y métodos son los más propensos a tener defectos. En la siguiente tabla se evalúan los riesgos dependiendo de la cantidad de complejidad ciclomática.

Complejidad Ciclomática	Evaluación de riesgos
1 - 10	Programa Simple, sin mucho riesgo
11 - 20	Más complejo, riesgo moderado
21 - 50	Complejo, programa de alto riesgo
51 o más	Programa muy complejo, Muy alto riesgo

Tabla: Evaluación del riesgo según la complejidad ciclomática encontrada en un sistema.

¿Cómo se calcula?

How to find the Cyclomatic Complexity in a function / Loop	
Start with 1	
If	Add 1
while	Add 1
for	Add 1
Number of case statements in a switch	Add 1
&&	Add 1
	Add 1
?:	Add 1
Do	Add 1
Catch	Add 1

Calculo de la complejidad ciclomática

Ejercicio:

- Aplicar la siguiente instrucción al servidor Jenkins:
`--max-complexity=2 .`
- Este comando permite que el sistema que estamos integrando al servidor tenga una complejidad ciclomática de “2”, la cantidad de complejidad puede cambiar dependiendo de lo que se quiera.

Ejemplo:

```
1 `--max-complexity=1 .`
```

14 Refactoring

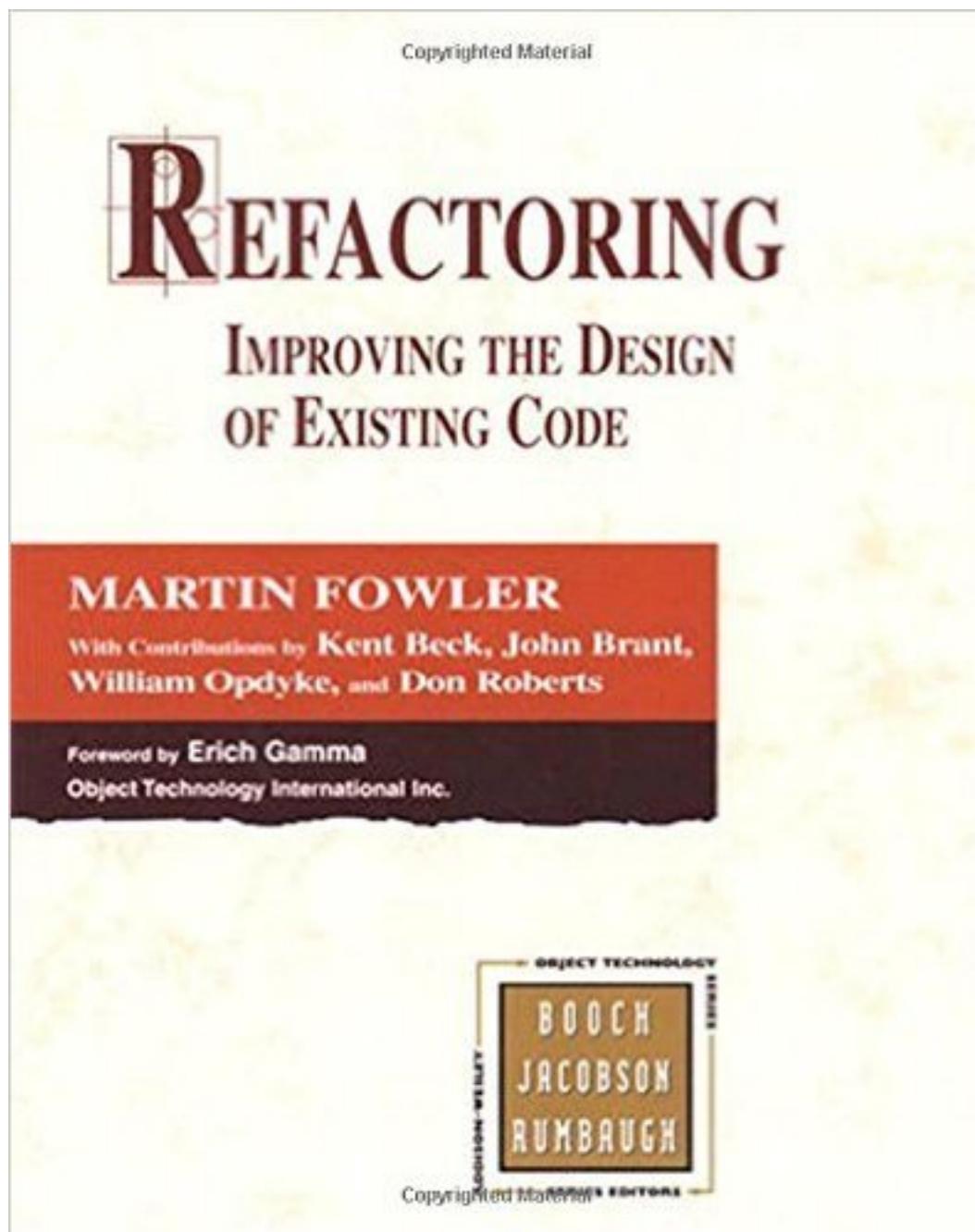
El término refactoring se refiere a una transformación del código que no afecta la funcionalidad externa del sistema para mejorar la facilidad de comprensión o cambiar su estructura y diseño, con la finalidad de facilitar el mantenimiento en el futuro.

Martin Fowler nos presenta en su libro “Refactoring: Improving the Design of Existing Code”, una excelente guía para aprender más sobre este concepto.

- REFACTORING: IMPROVING THE DESIGN OF EXISTING CODE
- Por: MARTIN FOWLER , ADDISON-WESLEY, 1999
- ISBN: 9780201485677

Datos del libro

- Nº de páginas: 320 págs.
- Encuadernación: Tapa blanda
- Editorial: ADDISON-WESLEY
- Lengua: INGLÉS
- ISBN: 9780201485677



Libro Refactoring

Uno de los conceptos que se presentan en este libro es Code Smell, este concepto se refiere a distintos tipos de errores de diseño que debemos evitar al momento de programar. En la siguiente tabla se muestra un catálogo de algunos de los Code Smells más sobresalientes.

Smell	Refactoring
<p>Comments. When you feel like writing a comment, first try “to refactor so that the comment becomes superfluous”</p>	Rename Method, Extract Method, Introduce Assertion
<p>Duplicated Code. Duplicated code is the most pervasive and pungent smell in software. It tends to be either explicit or subtle. Explicit duplication exists in identical code, while subtle duplication exists in structures or processing steps that are outwardly different, yet essentially the same</p>	Chain Constructors, Extract Composite, Extract Method, Extract Class Form Template Method, Introduce Null Object, Introduce Polymorphic Creation with Factory, Method, Pull Up Method, Pull Up Field Replace One/Many Distinctions with Composite, Substitute Algorithm, Unify Interfaces with Adapter
<p>Large Class. Fowler and Beck note that the presence of too many instance variables usually indicates that a class is trying to do too much. In general, large classes typically contain too many responsibilities</p>	Extract Class, Extract Subclass, Extract Interface, Replace Data Value with Object, Replace Implicit Language with Interpreter, Replace State-Altering Conditionals with State

Tabla: Ejemplos de Code Smells.

Se pueden encontrar todos los Code Smells del libro [aquí²⁰](#)

Ejercicio: Realizar refactoring a las pruebas de TDD del archivo “python_lab.py”.

Nota: El siguiente comando nos permite conocer cuántas líneas de código tiene un archivo:

```
1 `wc -l nombre_del_archivo`
```

²⁰<http://www.industriallogic.com/wp-content/uploads/2005/09/smellstorefactorings.pdf>

15 Unit Test

El desarrollo de una aplicación es deficiente si no se cubren pruebas unitarias, existen malas creencias sobre el desarrollo de pruebas unitarias como: “pérdida de tiempo”, “resulta complicado probar el código”, “las pruebas no son importantes”, “no ayudan a mejorar el código”, y una clásica frase de los desarrolladores “mi trabajo es desarrollar, no probar código, ese no es mi trabajo!”, todos estos son algunos ejemplos de los pretextos que pueden tomar los desarrolladores para no realizar pruebas unitarias y las cuáles están completamente equivocadas.

Una prueba unitaria es un método que prueba una unidad de código, lo cual agrega madurez y calidad al desarrollo de Software, desarrollar pruebas unitarias es una de las mejores prácticas que se deben aplicar desde el comienzo de la programación de cada individuo, para que sea adoptado como un buen hábito de desarrollo de Software. Las pruebas unitarias mejoran la calidad de las aplicaciones ya que ayudan a disminuir los errores o bugs, y es menos costoso corregir un error en la fase de desarrollo, ya que tomaría más tiempo encontrar un error cuando el sistema esté en producción. El desarrollo de la pruebas unitarias debería ser creado antes de comenzar la codificación de una manera sencilla y entendible para todos.

Como ejemplo de la redaccion de una prueba unitaria en Python, es la sigueinete funcion, la cuál nos devuelve los minutos de una semana y de dos semanas, conjuntamente se agrega el resultado que se espera obtener al correr la funcion:

Suponga que este es uno de los requerimientos de un sistema y que fue redactado en una historia de usuario.

Para explicar un poco mejor como Python ejecuta las pruebas se explicara de la siguiente manera:

- Def: define la función y el nombre de la función
- Lo que se encuentre entre comillas triples “”” es tomado como la prueba.
- Lo que va después de >>> es el resultado que esperamos.
- Y enseguida realizamos el código que ejecutara la función.

```
def minutes_in_weeks(weeks):
```

```
1 """ 1: (Task 0.5.1) Minutes in a Week
2
3 >>> minutes_in_weeks(1)
4 10080
5
6 >>> minutes_in_weeks(2)
7 20160
8 """
```

La redacción de esta prueba es fácil de entender para cualquier desarrollador o tester y también el cliente.

Después de haber redactado todas las pruebas para cada una de las historias de usuario se procede a codificar la función de la siguiente manera (podría ser de una manera diferente según la imaginación del desarrollador).

```
def minutes_in_weeks(weeks):
```

```
1 """ 1: (Task 0.5.1) Minutes in a Week
2
3 >>> minutes_in_weeks(1)
4 10080
5
6 >>> minutes_in_weeks(2)
7 20160
8 """
```

```
{.python} weeks = 2 r = [60 * 24 * 7 * weeks] print(r) ~
```

```
{.python} [20160]
```

~

16 Code Coverage

Es la cantidad de código medido que indica qué porcentaje de pruebas se están cubriendo. Verificar el Code Coverage de una aplicación aumenta la calidad, ya que entre más cobertura tenga la aplicación, se garantiza que mayor cantidad de código está siendo probado y también se detecta código innecesario, código que no se está ejecutando o identificar si es importante para el funcionamiento de la aplicación.

Para verificar el Code Coverage de una aplicación es necesario contar con dos cosas: una herramienta para la generación de las pruebas unitarias, y alguna aplicación para medir las pruebas, ambas herramientas deben ser compatibles.

Se dice que el Code Coverage es bueno si es arriba de 80%. Osea es bueno si las pruebas se están cubriendo en una totalidad arriba del 80%.

Para ello Python cuenta con un comando el cuál se agrega a la configuración de Jenkins o el servidor de integración continua que se utilice, para que estas pruebas se ejecuten cada determinado tiempo según la configuración establecida.

Ejercicio: Todos los integrantes del equipo deberán instalar en su máquina Vagrant Code Coverage para Python.

Instalar:

```
1 `sudo pip install coverage`
```

Nota: En caso de que no se encuentre instalado.

Ejercicio: Solo el líder de equipo debe agregar el siguiente comando a la configuración de Jenkins:

```
1 `--with-coverage`  
2 `--cover-min-percentage=99`
```

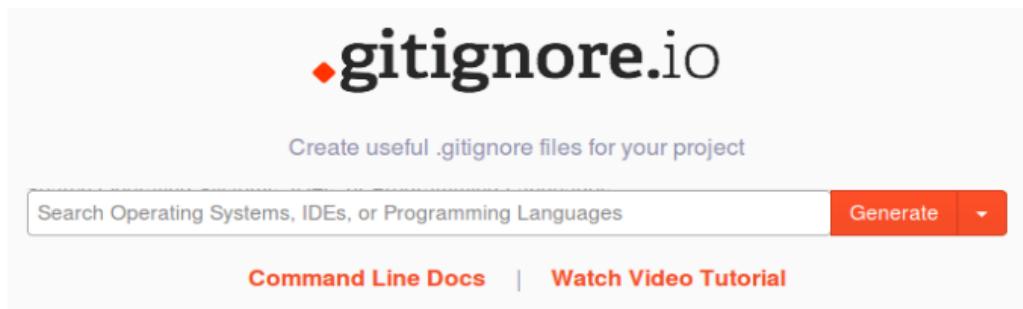
17 Gitignore

Frecuentemente se necesita que repositorios de código ignoren archivos temporales que las herramientas de desarrollo utilizadas generan. Estos archivos son innecesarios para el correcto funcionamiento del sistema que se esta desarrollando, son importantes para las herramientas de desarrollo como NetBeans, SublimeText, Eclipse etcetera, y cada una de estas, crea conflictos cuando se realiza la integracion en el repositorio.

Para solucionar este problema existe una aplicación Web llamada .gitignore. Esta aplicación genera un archivo con las extensiones de los archivos que requieren ser ignorados

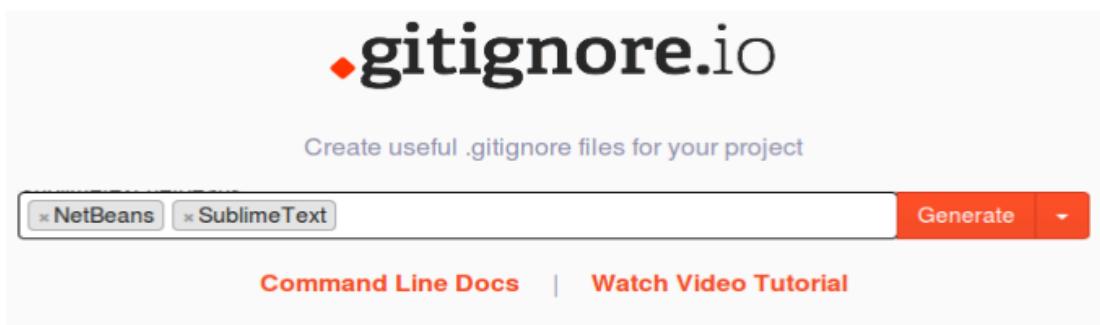
Ejercicio Con los siguientes pasos se creara el archivo .gitignore

Un buen lugar para empezar sería generar el archivo “.gitignore” aquí²¹



Página de gitignore

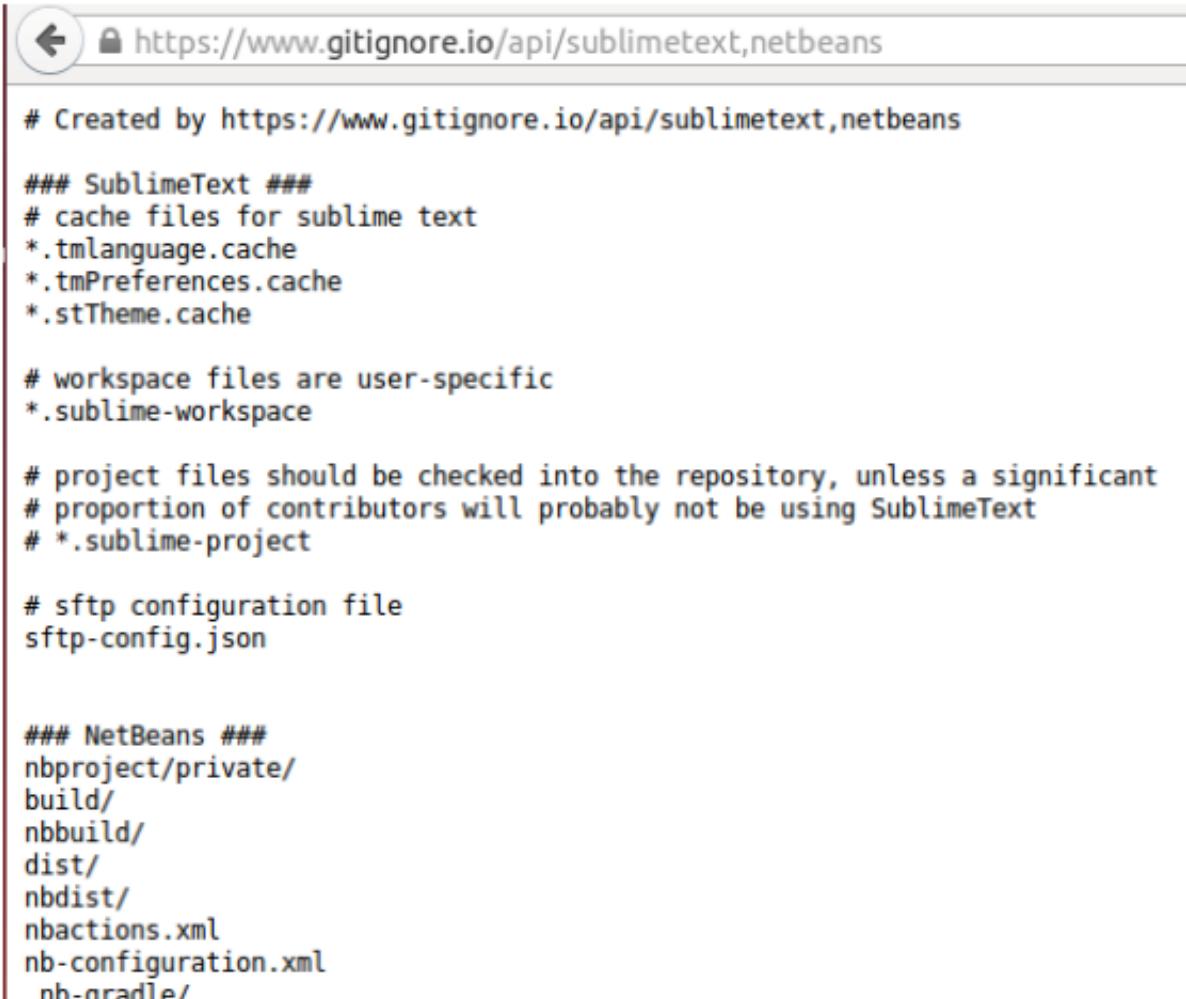
Esta aplicación es sencilla de utilizar, simplemente se escribe el nombre de las herramientas de desarrollo que usan el equipo de trabajo. Por ejemplo si alguien utiliza SublimeText y otra persona utiliza NetBeans usarán esas instrucciones. Da clic en “Generar”



Agregacion de programas que utiliza cada desarrollador

²¹<https://www.gitignore.io/>

Se generará un archivo como el siguiente:



The screenshot shows a browser window with the URL <https://www.gitignore.io/api/sublimetext,netbeans>. The page content displays a generated .gitignore file:

```
# Created by https://www.gitignore.io/api/sublimetext,netbeans

### SublimeText ###
# cache files for sublime text
*.tmLanguage.cache
*.tmPreferences.cache
*.stTheme.cache

# workspace files are user-specific
*.sublime-workspace

# project files should be checked into the repository, unless a significant
# proportion of contributors will probably not be using SublimeText
# *.sublime-project

# sftp configuration file
sftp-config.json

### NetBeans ###
nbproject/private/
build/
nbbuild/
dist/
nbdist/
nbactions.xml
nb-configuration.xml
nb-gradle/
```

Archivo que genera Gitignore

Con el archivo de gitignore.io actualizar el archivo “.gitignore” en tu repositorio de código, una sola persona del equipo realizara commit y push para guardar los cambios.

Todos los miembros del equipo deberán realizar pull para mantener su repositorio local actualizado.

Por último se necesita decirle a git que “olvide” los archivos “basura” que ya tenga guardados.

Para ello seguir las instrucciones que encontrarás [aquí²²](#)

²²<http://stackoverflow.com/a/19095988/54848>

18 Pre-commit hooks

¿Cómo evitar que los desarrolladores hagan commit al repositorio si el código no compila? o ¿Cómo asegurar que todos los mensajes de commit sigan cierto formato?

Los pre-commit hooks son pequeños programas que se ejecutan antes de que el desarrollador pueda hacer commit en su repositorio local y por ello son el lugar ideal para correr pequeñas pruebas o cosas que se deben asegurar para mantener el repositorio de código saludable.

Instalar desde la consola pre-commit-hook y algunos otros comandos de interés:

```
1 `flake8 --install-hook`  
2 ó  
3 `pip install git-pre-commit-hook`
```

NOTA: Podría pedir tener instalado flake8, ejecuta el siguiente comando:

```
1 `sudo apt-get install python-flake8`
```

Actualización:

```
1 `pip install --upgrade git-pre-commit-hook`
```

Desinstalación:

```
1 `pip uninstall git-pre-commit-hook`
```

Ejercicio:

A continuación crear un pre-commit hook que impida que se haga commit de código que no cumple con el estándar o que no compile.

- Entrar al archivo “.git” que se encuentra de manera oculta dentro de la carpeta en la cual se están realizando los ejercicios.

```
cd .git
```

Ahora entrar a la carpeta “hooks” que también se encuentra oculta

```
1 `cd hooks`
```

Dar un

```
1 `ls`
```

para que liste los archivos que contiene esta carpeta

```
vagrant@vagrant-ubuntu-precise-32:/vagrant/i2$ cd .git
vagrant@vagrant-ubuntu-precise-32:/vagrant/i2/.git$ cd hooks
vagrant@vagrant-ubuntu-precise-32:/vagrant/i2/.git/hooks$ ls
applypatch-msg.sample  pre-applypatch.sample    pre-push.sample
commit-msg.sample      pre-commit            pre-rebase.sample
post-update.sample     pre-commit.sample     update.sample
pre                  prepare-commit-msg.sample
vagrant@vagrant-ubuntu-precise-32:/vagrant/i2/.git/hooks$ █
```

Lista de archivos en la carpeta hooks

Abrir el archivo “pre-commit” con el siguiente comando:

```
1 `nano pre-commit`
```

Modificar la línea de “false” a “true”:

```
1 Flake8_STRICT, false
2      a
3 Flake8_STRICT, true
```

Guardar y salir.

Modificar el archivo python_lab.py para tratar de violar el estandar de codigo.

Tratar de hacer commit de los cambios. Mostrara como resultado un error que no dejará que suba los cambios ya que se está tratando de subir un código que no cumple con su estándar.

Para mas dudas sobre pre-commit-hook checar el siguiente enlace [aqui²³](#)

²³<http://flake8.readthedocs.org/en/latest/vcs.html>

19 Ejercicio Generación de tickets: Crear aplicación con Flaskr

El objetivo de este ejercicio es practicar el manejo de un repositorio de código para el control de versiones con el equipo de trabajo.

- El líder de equipo crea un fork del proyecto que el instructor indicará
- Agregar a los colaboradores del equipo
- Cada miembro hace un clon del fork que realizó el líder

Se desarrollará una aplicación básica que consta de:

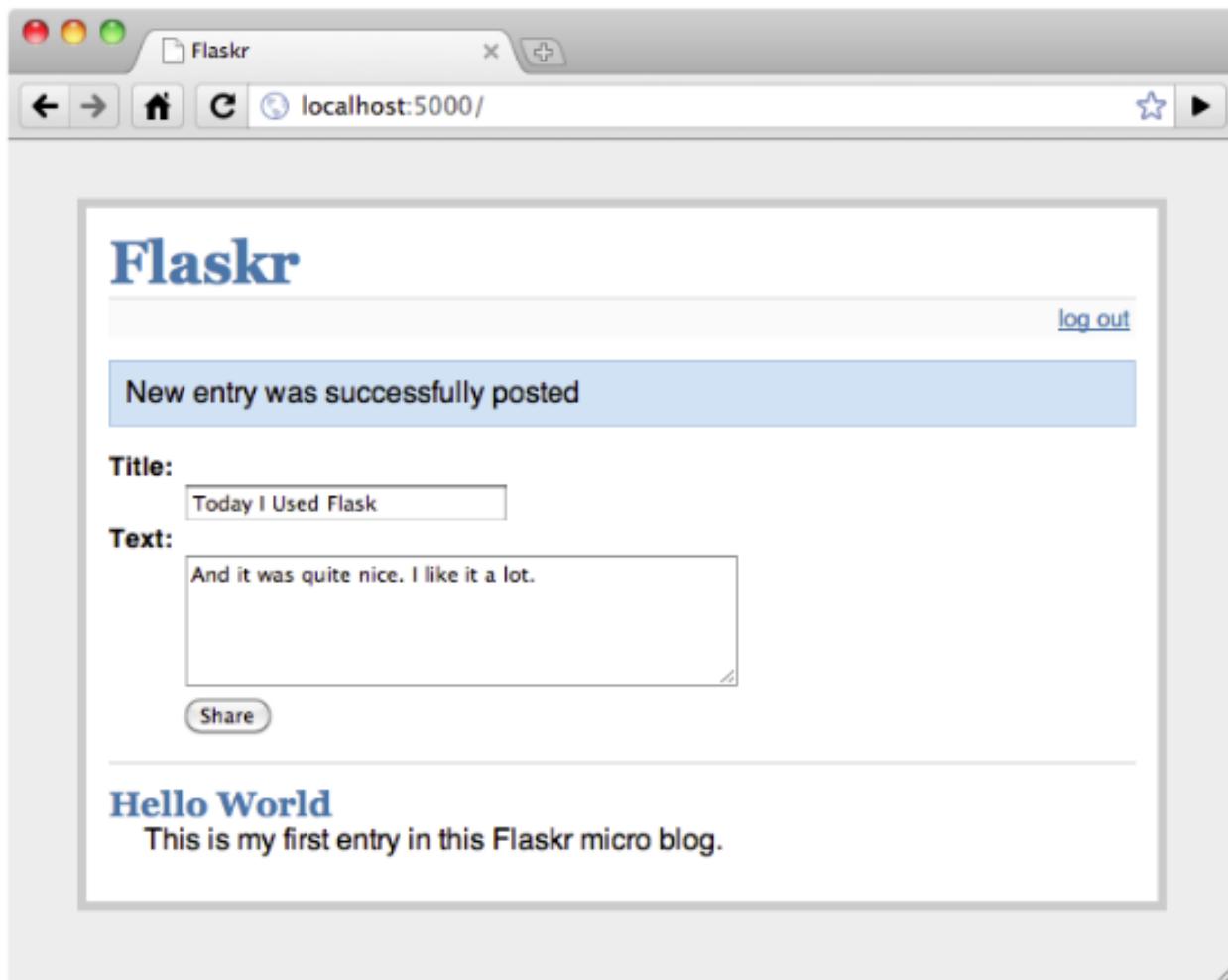
- Inicio de sesión con “usuario” y “contraseña”
- Captura de mensaje que contendrá “título” y “mensaje”.
- Visualización de todos los mensajes
- Cierre de sesión

Pasos generales:

- Generar tickets a partir del clone del repositorio.
- Cada integrante del equipo debe desarrollar un ticket y subir sus cambios al repositorio para mantenerlo actualizado
- Desarrollar la aplicación siguiendo la especificación de los tickets por cada uno de los integrantes
- Verificar el correcto funcionamiento de la aplicación

Link del tutorial para el ejercicio [aquí²⁴](#)

²⁴<http://flask.pocoo.org/docs/0.10/tutorial/>



Ejemplo de la interfaz de la aplicación que se desarrollará en el ejercicio.

NOTA: Dudas o aclaraciones verificar con el instructor del curso.

20 Tarea: investigación de las herramientas vistas con los lenguajes: Java, PHP, C++, .Net, R, Ruby.

Por equipos elegir un lenguaje de programación, del cual se expondrá los siguientes puntos:

1. Buscar el estándar de código de mi lenguaje de programación
2. Buscar la herramienta del check coding standard
3. Buscar la herramienta pretty print (autopep8)
4. ¿Mi lenguaje implementa doctest?
5. Pruebas unitarias de mi lenguaje
6. Code coverage de mi lenguaje
7. Buscar el servidor de integración continua de mi lenguaje.

NOTA: Cada equipo contará con 15 minutos para exponer.

21 Retrospectiva del día 3

Realizar retrospectivas de la misma manera durante un largo tiempo, puede afectar la mejora (que es el propósito principal de una retrospectiva), ya que se deja de aprender. Una manera de seguir aprendiendo y la retrospectiva cumpla su objetivo es aplicando maneras diferentes de realizar retrospectivas, por ejemplo: “The Joel Test”.



Avram Joel Spolsky Ingeniero en Software y escritor

The Joel Test

Joel Spolsky describe 12 preguntas para identificar si una empresa de desarrollo para identificar si se esta desarrollando código de calidad. Con estas preguntas es facil de identificar si se esta desarrollando buen código, cada pregunta vale un punto, si la respuesta es “si”, se realiza la suma de los puntos e indentificara que tan bueno o malo esta desarrolando código una empresa. Según Joel, 12 es perfecto, 11 es tolerable, pero 10 indica que tiene serios problemas.

1. Do you use source control?
2. Can you make a build in one step?
3. Do you make daily builds?
4. Do you have a bug database?
5. Do you fix bugs before writing new code?
6. Do you have an up-to-date schedule?
7. Do you have a spec for requirements?
8. Do programmers have quiet working conditions?
9. Do you use the best tools money can buy?
10. Do you have testers?
11. Do new candidates write code during their interview?
12. Do you do hallway usability testing?

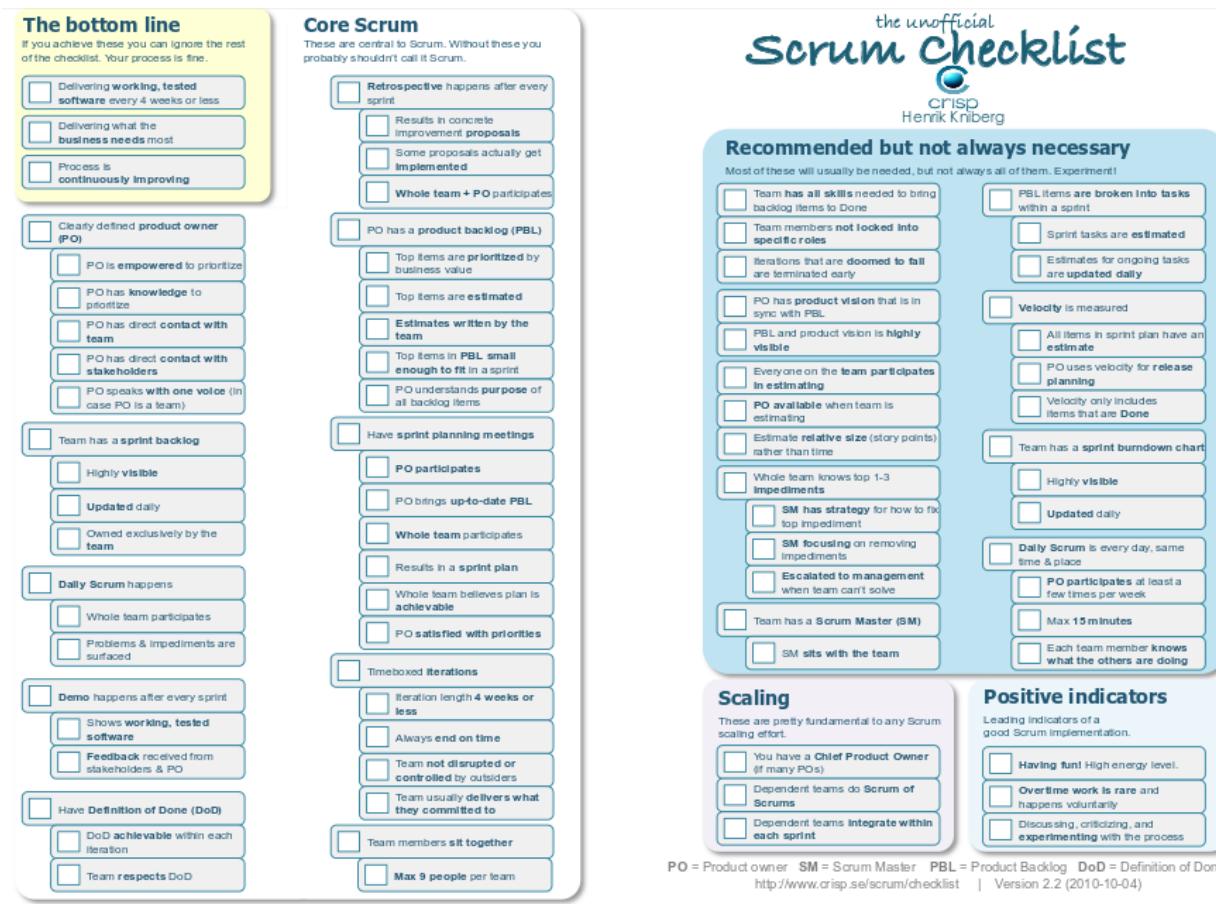
Por lo tanto, si se es dueño de una empresa o como buen emprendedor pretende iniciar con una empresa de desarrollo debera tomar en cuanta que las empresas que desarrollan con la mejor calidad, tienen una puntuacion de 12 como Microsoft, pero en general las empresas en México según los resultados de la aplicacion de este test a alumnos de la maestria en Ingenieria de Software (CIMAT) impartida en Zacatecas arrojaron que las empresas en las que han trabajado o realizado estadias o residencia profesional, calificaron a la empresa con un promedio de entre 4 y 6.

Se sabe que estos aspectos no determinan si una empresa es exitosa o no, pero cumplir con estas doce preguntas, sin duda alguna le ayudaran a formar un equipo de trabajo con disciplina y que genere Software de calidad.

Link Joel Test [aquí²⁵](#)

Otra manera de realizar las retrospectivas es aplicando el Check-list que sugiere SCRUM. Es una lista de preguntas específicas, las cuales se van marcado si el equipo de trabajo está cumpliendo con esa tarea o actividad, en caso contrario, son las preguntas que se toman en cuenta para realizar la mejora y establecer los compromisos.

²⁵<http://www.joelonsoftware.com/articles/fog0000000043.html>



Check-list de SCRUM

Link del Check-list: aquí²⁶

²⁶<https://www.crisp.se/wp-content/uploads/2012/05/Scrum-checklist.pdf>

22 Pruebas de interfaz de usuario: Selenium IDE

Las pruebas de interfaz de usuario consisten en la implementación de un conjunto de eventos en la UI, como dar clic sobre el sistema y observar que el comportamiento sea el esperado, esta tarea la mayoría de las veces es encargada a estudiantes que se encuentran en una organización realizando estadías por parte de la institución educativa, “trabajo fácil”, pero esto no es enriquecedor para los estudiantes. Para la implementación de pruebas de interfaz ya existe Software que aplica estas pruebas automáticamente, una de las herramientas es Selenium IDE (Integrated Development Environment) es un plug-in de Firefox fácil de utilizar y es generalmente la manera más eficiente de ejecutar para este tipo de aplicaciones, nos permite ahorro de tiempo en su ejecución.

¿Cómo se usa?

Instalación.

- Descargar el navegador Firefox.
- Instalar el Plug-in Selenium IDE [aquí²⁷](#)

Selenium IDE

Selenium IDE is a Firefox plugin which records and plays back user interactions with the browser. Use this to either create simple scripts or assist in exploratory testing. It can also export Remote Control or WebDriver scripts, though they tend to be somewhat brittle and should be overhauled into some sort of Page Object-y structure for any kind of resiliency.

Download latest released version [2.9.0](#) released on 09/Mar/2015 or view the [Release Notes](#) and then [install some plugins](#).

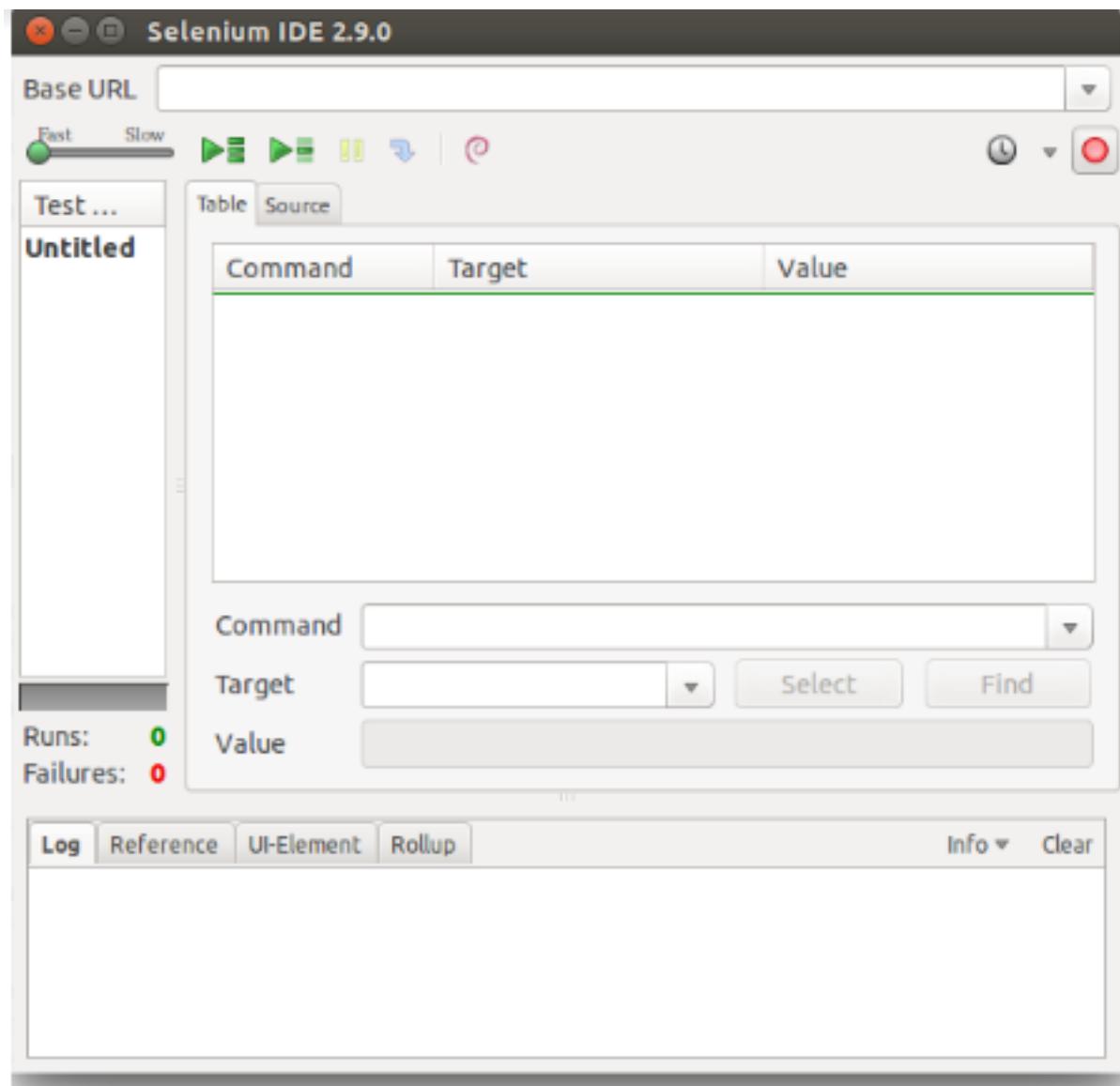
Descarga de selenium

Versión de Selenium que se debe instalar 2.0.9

- Aparecerá un ícono en la esquina derecha de tu navegador.

Al seleccionarlo mostrará la interfaz siguiente:

²⁷<http://www.seleniumhq.org/download/>



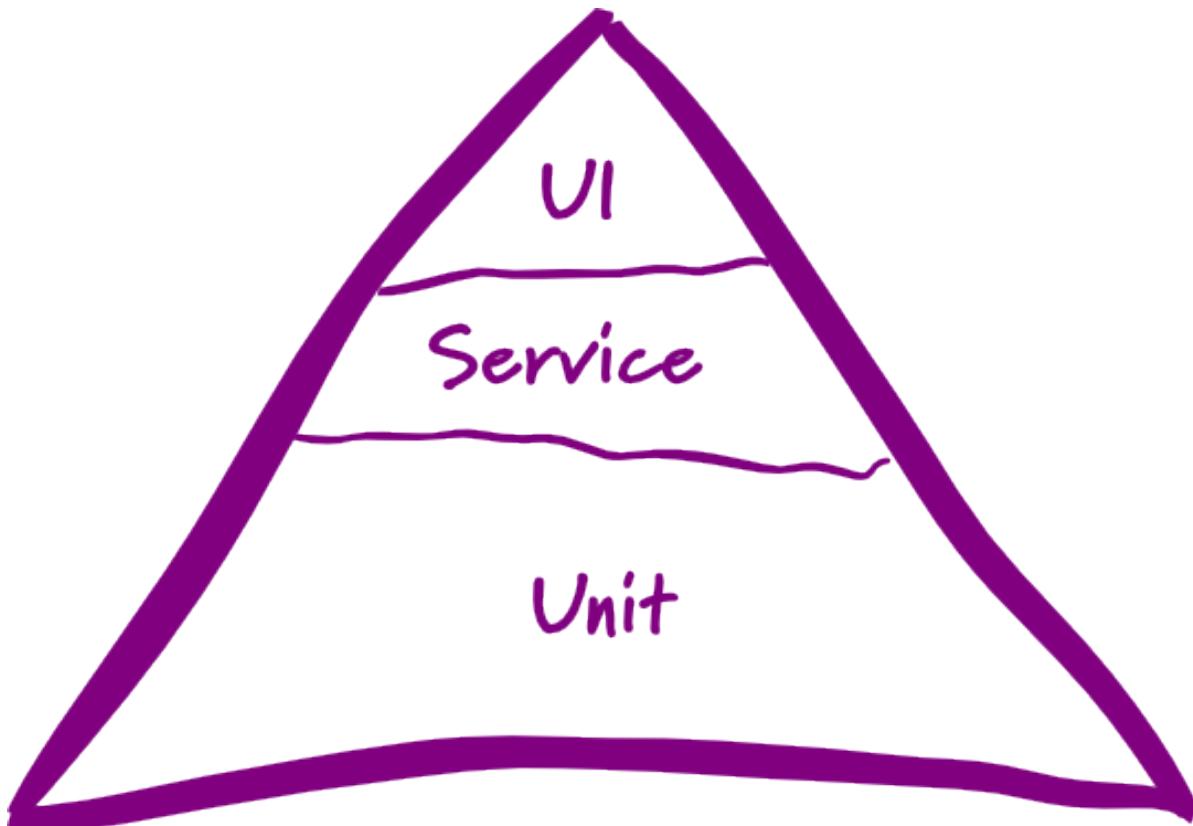
Interfaz de selenium IDE

NOTA: Para mayor información revisar el link aquí²⁸

Pirámide de pruebas

Se recomienda que la cantidad de pruebas desarrolladas y aplicadas sean de la siguiente manera, de mayor a menor (mayor cantidad de pruebas unitarias y menor la cantidad de pruebas de interfaz). Como se representa en la siguiente imagen:

²⁸http://www.seleniumhq.org/docs/02_selenium_ide.jsp



Piramide de pruebas

Ejercicio Instalar Selenium IDE y realizar pruebas de interfaz a la aplicación desarrollada en el tema “19. Ejercicio de generación de tickets”.

23 Seguridad: SQL Injection, OWASP

Los datos y la información que se generan diariamente en internet es demasiada, por lo cual las aplicaciones Web deben ser sitios seguros, y lo menos vulnerables ante ataques generados por usuarios malintencionados que hacen mal uso de sus conocimientos informáticos y que pueden dañar las aplicaciones. Diseñar aplicaciones Web sin tomar en cuenta la seguridad puede generar graves problemas en algún futuro, sin embargo desarrollar una aplicación segura no es una tarea fácil, pero sí representa un mejor trabajo.

SQL Injection. Es uno de los métodos más utilizados que aprovecha la vulnerabilidad de las aplicaciones para realizar operaciones sobre las bases de datos, el cual debe ser tomado en cuenta por los desarrolladores de aplicaciones Web.



By Randall Munroe

OWASP (Open Web Application Security Project). Es un proyecto de código abierto que proporciona recursos como artículos, libros, capacitaciones, Software de testeo, etcétera, gratuitos para determinar y combatir las causas que hacen que el software sea inseguro, y con ello promover Software seguro y de calidad.