Ingeniería Web: Visión General IWVG

Ecosistema

Jesús Bernal Bermúdez

Introducción

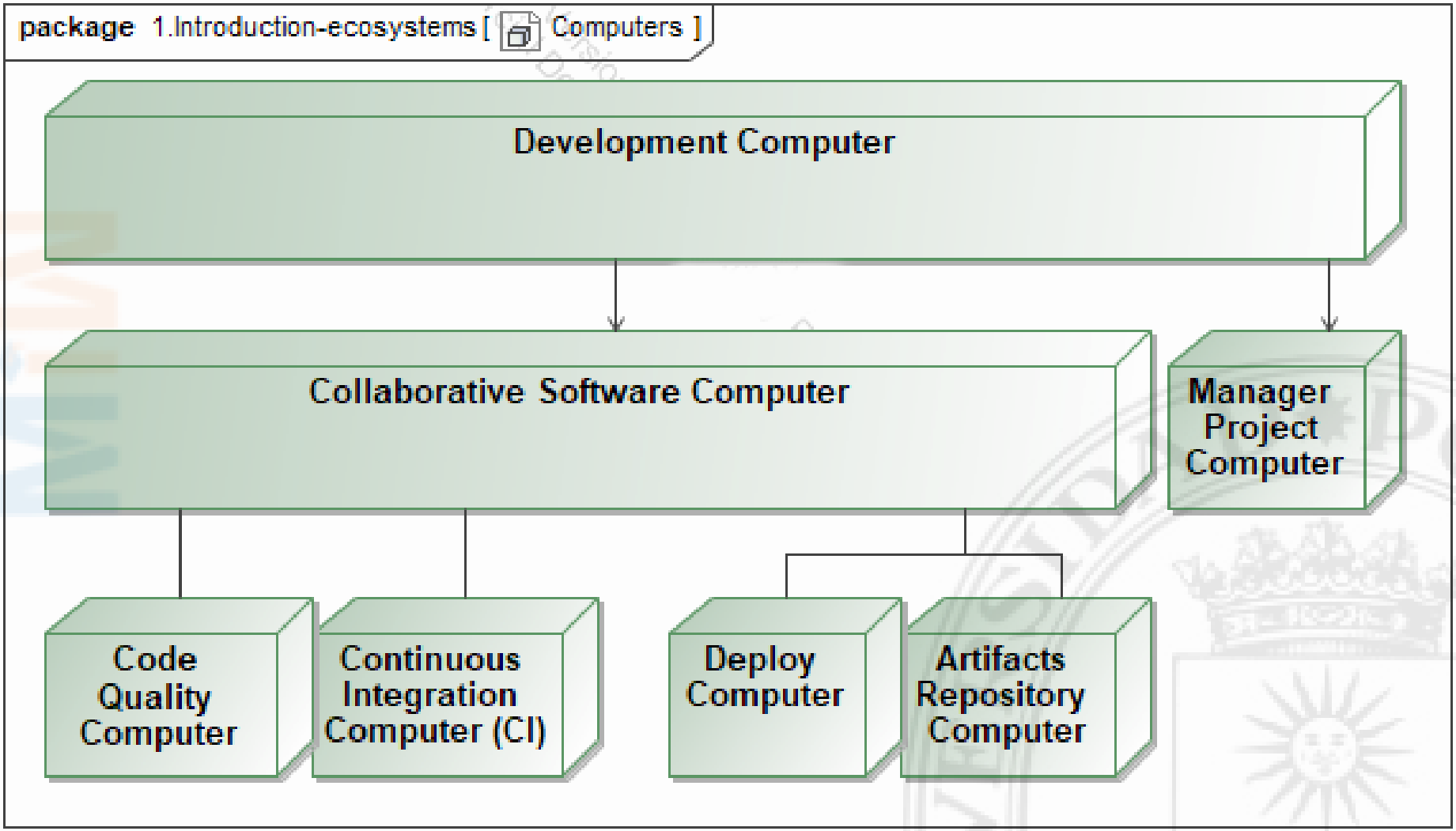
- **Ecosistema software**
  - Se define como un espacio de trabajo donde un conjunto de herramientas interactúan y funcionan como una unidad para el ***desarrollo de software colaborativo*** en todas sus fases (Gestión, Requisitos, Análisis, Diseño, Programación, Pruebas y Despliegue)

- **Forja**
  - Una forja es una plataforma para desarrollo colaborativo de software

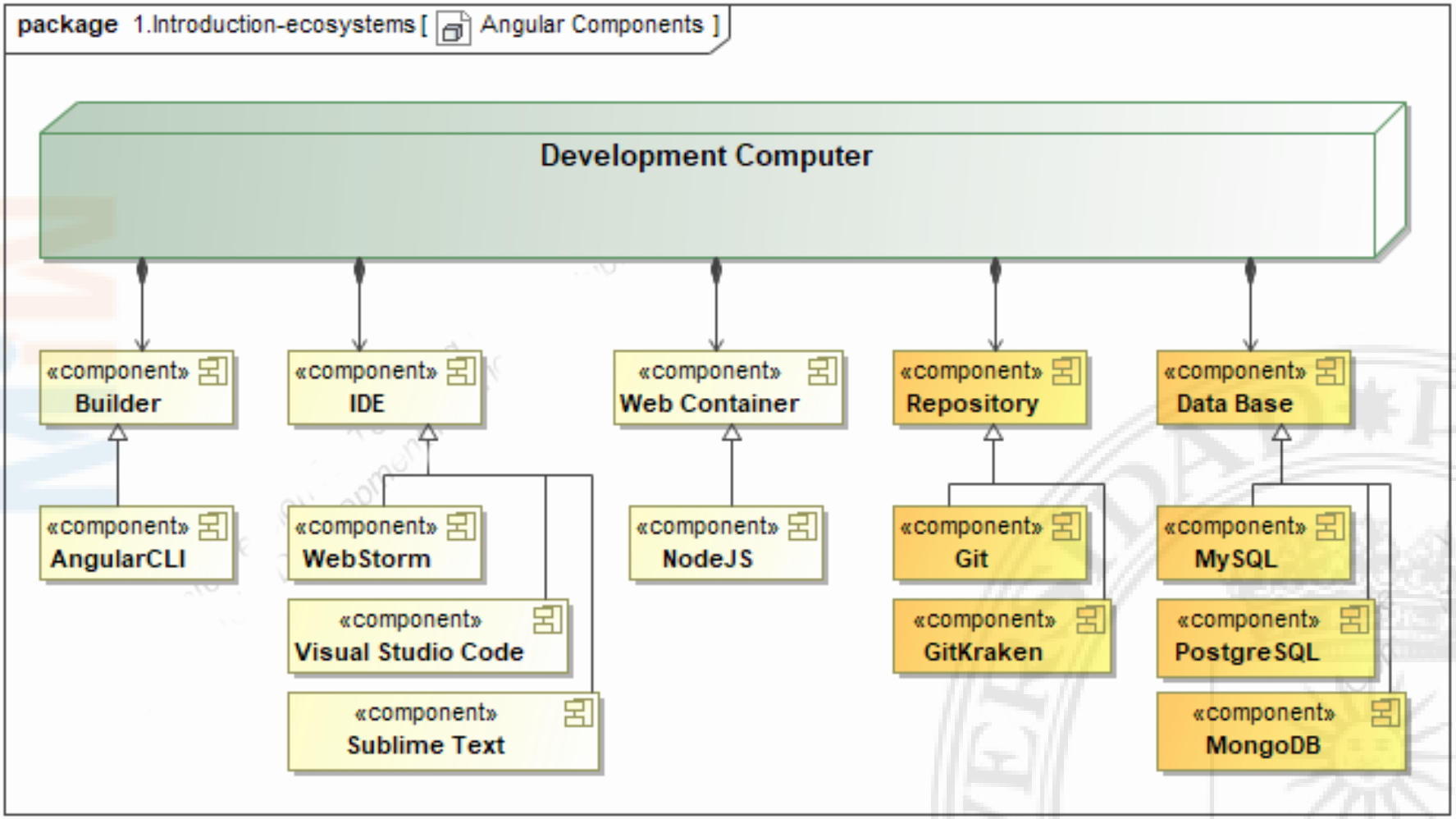
- **Entorno de Desarrollo Integrado (IDE)**
  - Se define como una aplicación informática que proporciona servicios integrales para facilitarle al desarrollador de software

- **Perfil:** ***Arquitecto QA (Quality Assurance)***

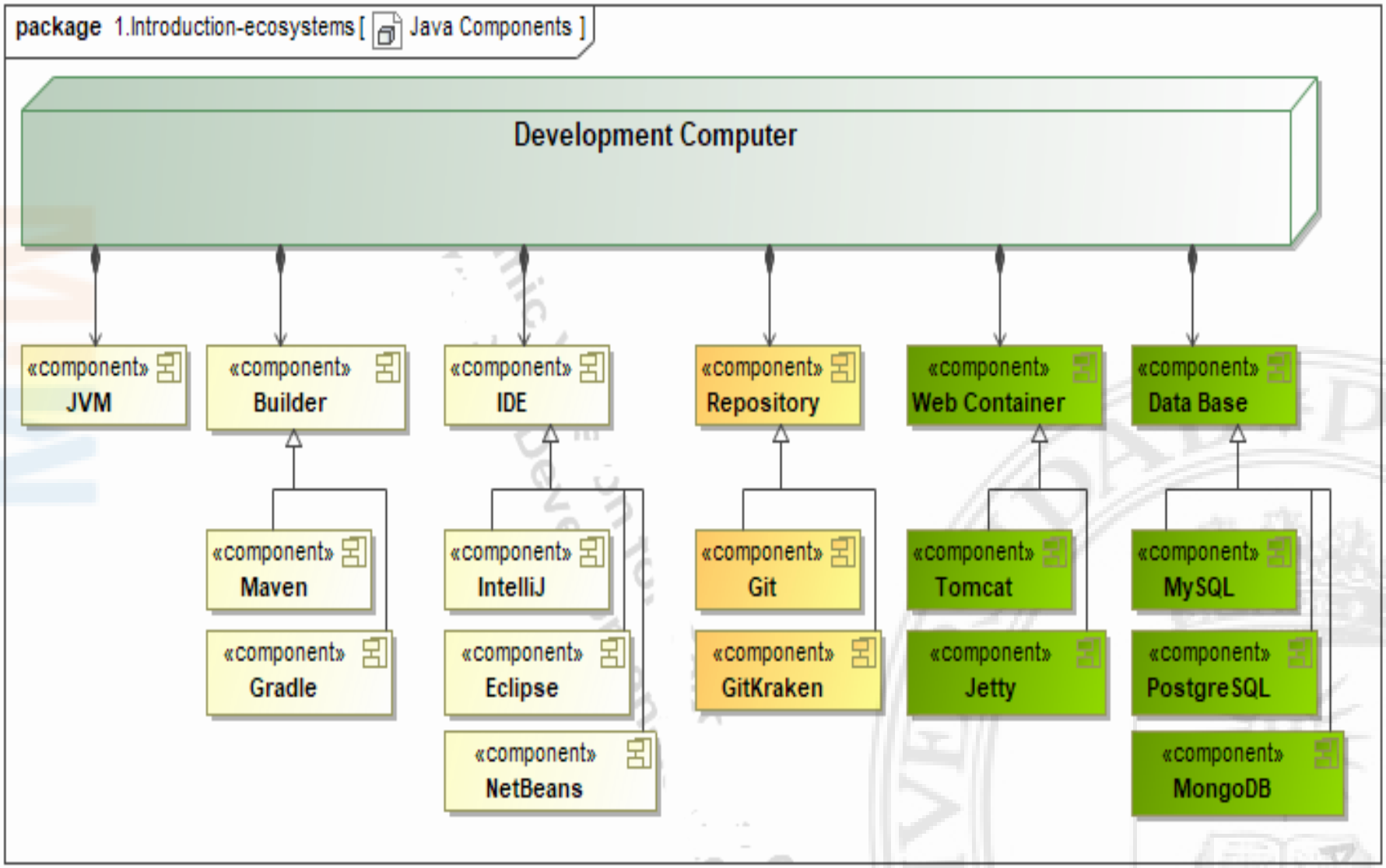
Introducción



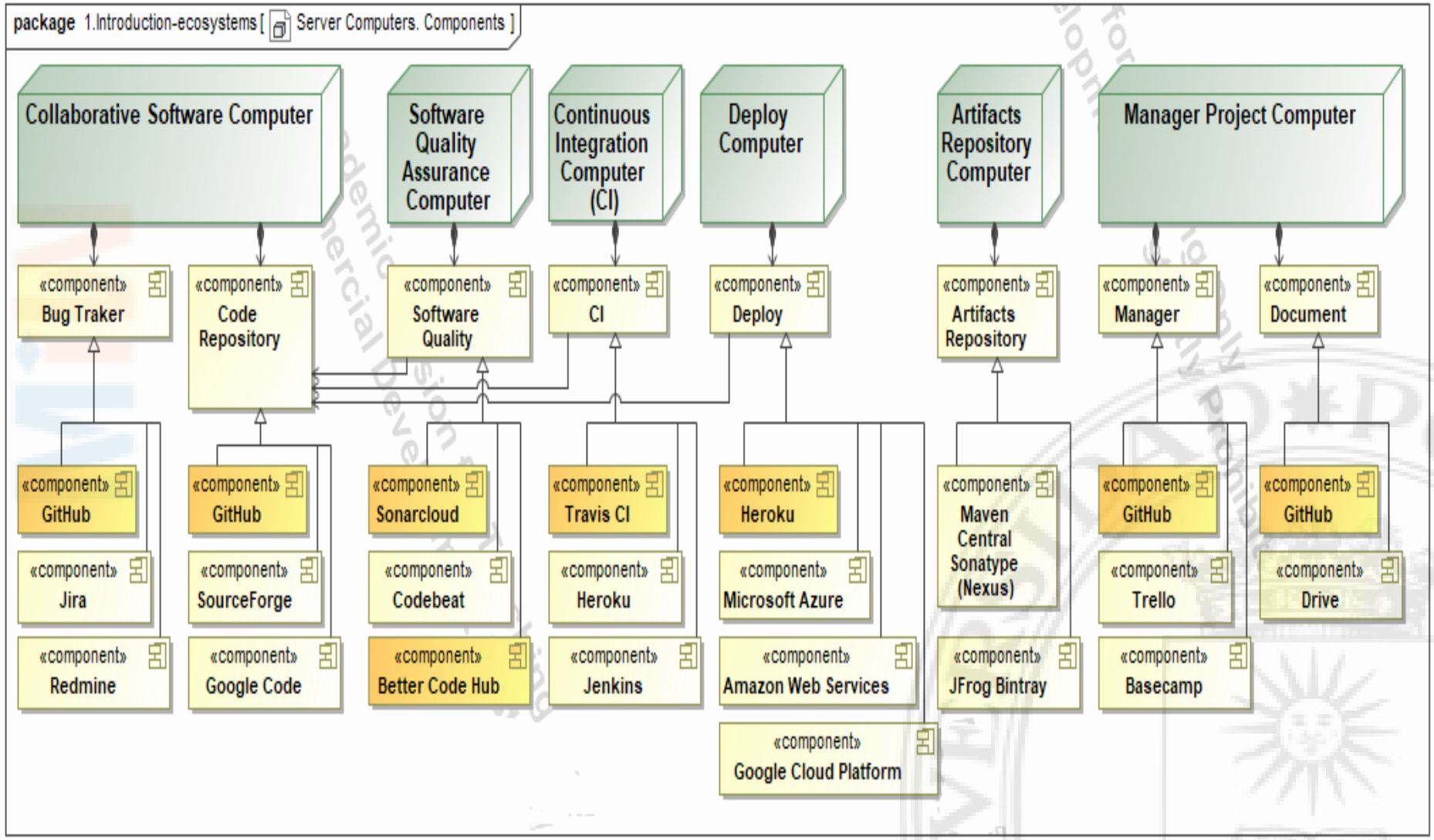
Equipo de desarrollo: Angular



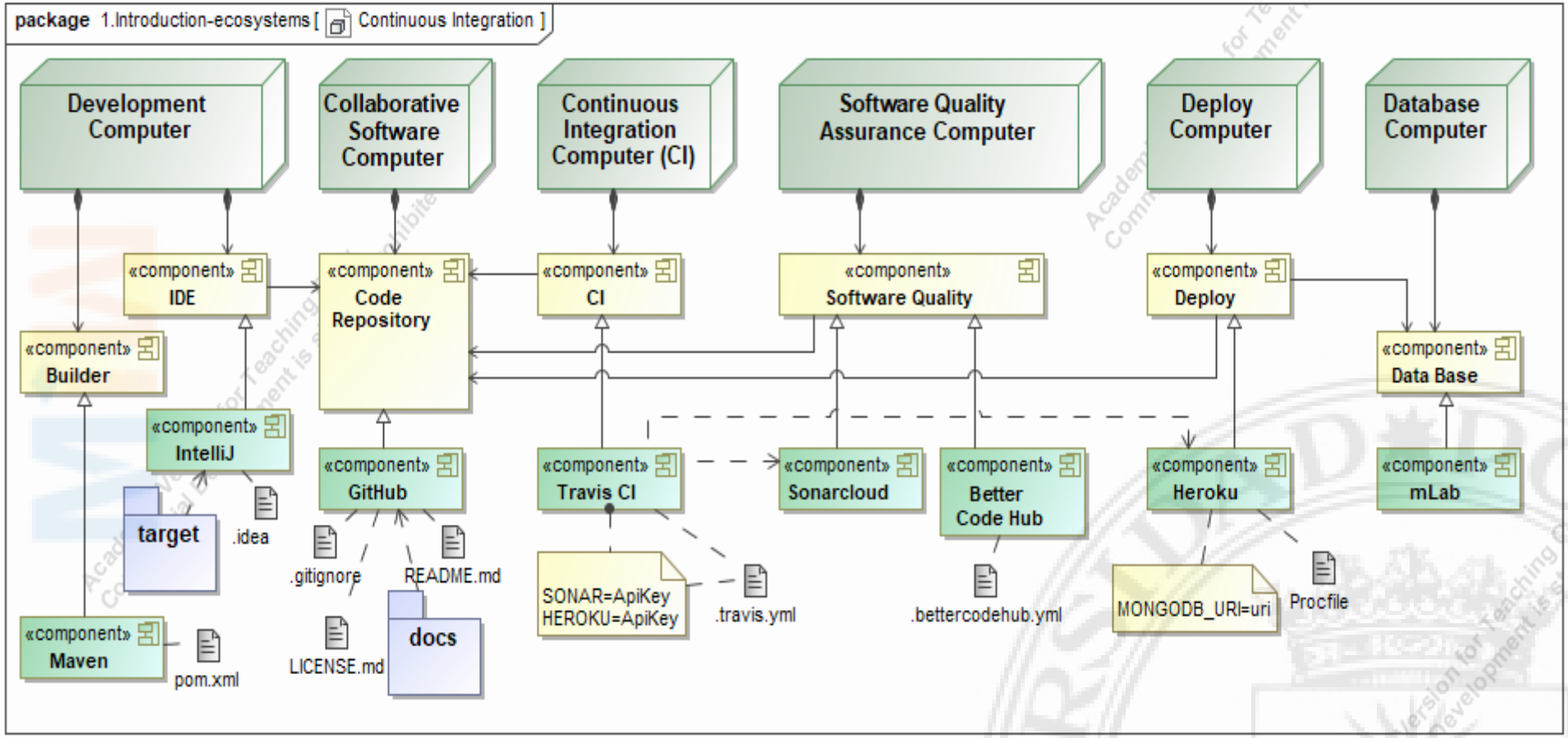
Equipo de desarrollo: Java



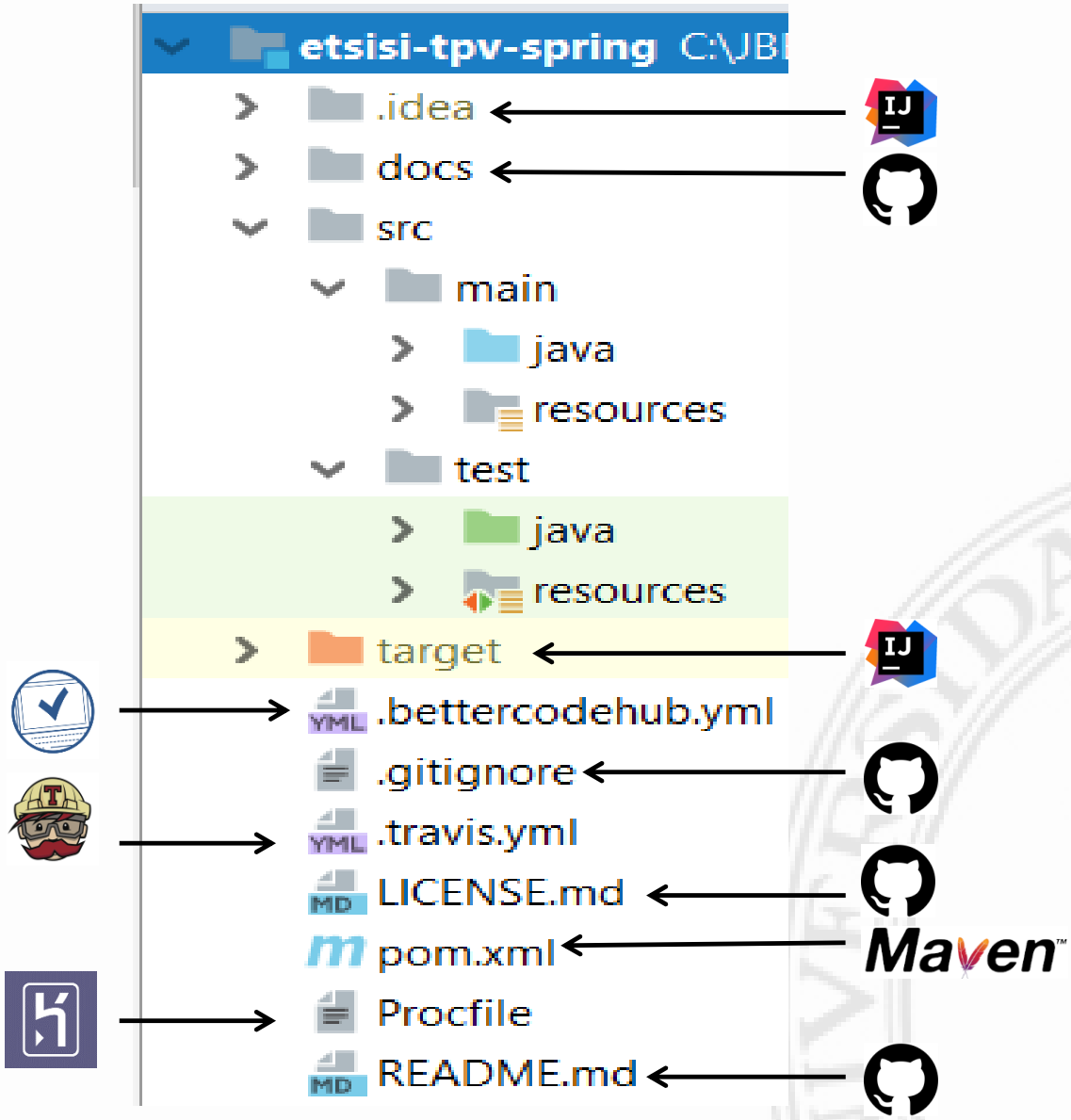
Software as a Service (SaaS)



Plataformas



Plataformas



Instalación del SDK de Java

- Para el desarrollo y compilación de aplicaciones Java, utilizaremos: Standard Edition (Java SE) o Java Development Kit (JDK) de Oracle:
  - *http://www.oracle.com/technetwork/java/javase/downloads/index.html*
- Para Instalar se realiza instalación de un *exe* sobre Windows
  - Se establece las variables de entorno ***JAVA_HOME*** (al raíz) Y ***PATH*** (al bin)
- Para la ejecución de aplicaciones Java (Delivery Platforms) es el Java Runtime Environment (**JRE**). Se instala automáticamente con Java SE
- Tutorial:
  - *http://docs.oracle.com/javase/tutorial/*
- API:
  - *http://docs.oracle.com/javase/8/docs/api/index.html*

Construcción de proyectos

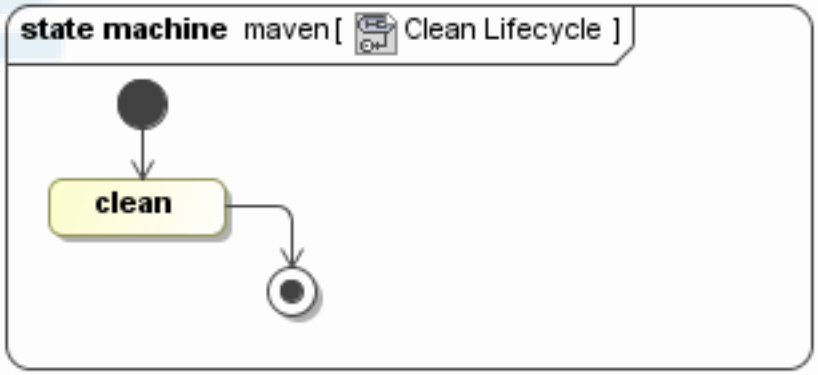
- La construcción del proyecto requiere realizar diferentes tareas como: compilación, pasar los test, pasar el control de calidad de fuentes, generar el jar, resolver las dependencias... Existen herramientas que nos ayudan y automatizan esta labor
  - ***Maven*** y ***Gradle*** (más reciente)
- ***Apache Maven*** es una herramienta de gestión de proyectos de software. Se basa en un modelo de objetos del proyecto (POM),
  - ***http://maven.apache.org/***
  - Documentación: ***http://www.sonatype.com/books/mvnref-book/reference/public-book.html***
- POM: Archivo(XML) donde se define el proceso de construcción, información del proyecto, y en general, cada una de las fases propuestas
- Instalación
  - https://maven.apache.org/
  - Para instalar bajar el zip y descomprimir
  - Se establece la variable de entorno ***M2_HOME*** (al raíz) y ***PATH*** (al bin)
- Maven crea una carpeta llamada **.m2/repository** en el perfil del usuario que contiene una copia de los artefactos

Conceptos de Maven

- **Artefacto**: Es la unidad mínima con la que trabaja Maven para gestionar sus dependencias, son los componentes software
- **Coordenadas**: Sistema con el Maven determina de forma única a cada uno de sus artefactos: ***groupId:artifactId:version***
  - ***Group Id***: identificación del grupo. Normalmente se utiliza el nombre del dominio, al reves: ***es.upm.miw.iwvg***
  - ***Artifact Id***: identificación del artefacto: ***forge***
  - ***Version***: versión del artefacto: ***1.0.0-SNAPSHOT, v1.3.4-RC (Release Candidate), v1.4.5-Release (Release)***
- **Empaquetado**: tipo de fichero generado, normalmente ***JAR*** o *WAR*
- **Repositorio**: Estructura de directorios y archivos que usa Maven para almacenar, organizar y recuperar artefactos. Existen repositorios locales, privados y remotos
  - El repositorio por defecto es el repositorio central de Maven
  - Existe un repositorio privado en la empresa para los artefactos de desarrollo
  - Existe un repositorio local, donde se copian las dependencias: ***%User%/.m2/repository***
- **Arquetipos**: Son plantillas para definir proyectos tipo con el fin de ser reutilizados

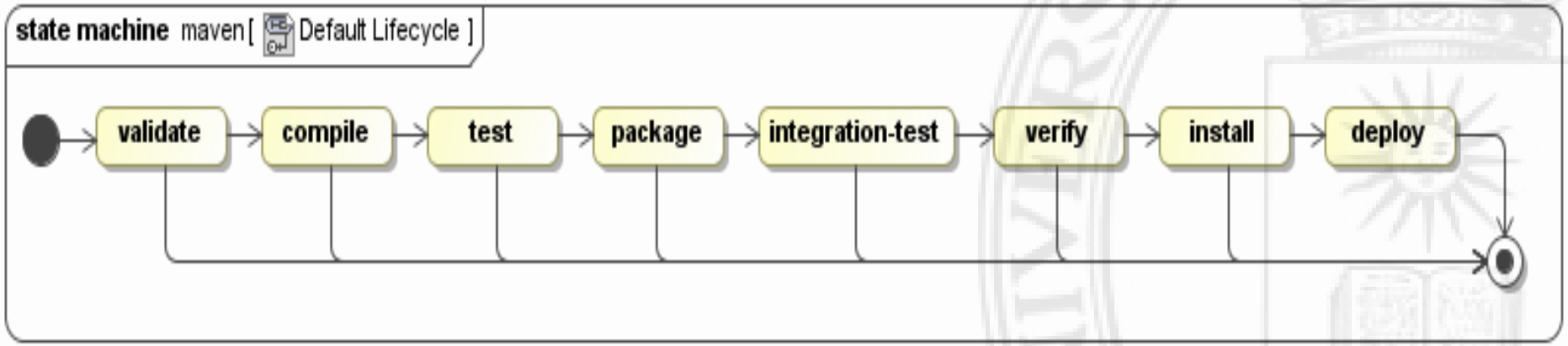
Ciclos de vida

- Clean Livecycle
  - ***clean*** **:** Elimina todos los ficheros generados por construcciones anteriores
- Comandos
  - ***> mvn clean***



Ciclos de vida

- Default Livecycle
  - ***validate*** **:** Valida el proyecto si es correcto
  - ***compile*** : Genera los ficheros .class compilando los fuentes .java
  - ***test*** : Ejecuta los test automáticos de JUnit existentes, abortando el proceso si alguno de ellos falla.
  - ***package*** : Genera el empaquetado final (jar, war...)
  - ***integration-test***: Procesar y desplegar el paquete si fuera necesario en un entorno donde se puedan ejecutar pruebas de integración
  - ***verify***: Ejecutar todas las comprobaciones para verificar la validez del paquete y que cumpla con los criterios de calidad
  - ***install*** : Copia el paquete en un directorio de nuestro ordenador, de esta manera pueden utilizarse en otros proyectos maven del mismo ordenador
  - ***deploy*** : Copia el paquete ***jar*** a un servidor remoto, poniéndolo disponible para cualquier proyecto maven con acceso a ese servidor remoto
- Comandos:
  - ***> mvn --version***
  - ***> mvn compile***
  - ***> mvn -Dmaven.test.skip=true package***
  - ***> mvn clean verify***



Plugin de objetivos

- Es una tarea específica, más pequeña que una fase de construcción, que contribuye a la construcción y gestión del proyecto
- En el ciclo de vida, va después de *package*
- Comando: *[nombre del plugin]:[función del plugin a ejecutar]*
  - ***mvn sonar:sonar***
  - ***mvn spring-boot:run***

IntelliJ IDEA

- IntelliJ IDEA es un entorno de desarrollo integrado (IDE), tiene una versión gratuita: *Community Edition*.
- Es una buena plataforma para el desarrollo de aplicaciones Java
  - ***https://www.jetbrains.com/idea/download***
  - Versión: ***Community***

Cliente Git

- Git es un software de control de versiones, distribuido, gratuito y de código abierto
- Sitio Web: ***https://git-scm.com***
- Documentación:
  - *https://git-scm.com/book/en/v2*
  - *https://git-scm.com/book/es/v2*

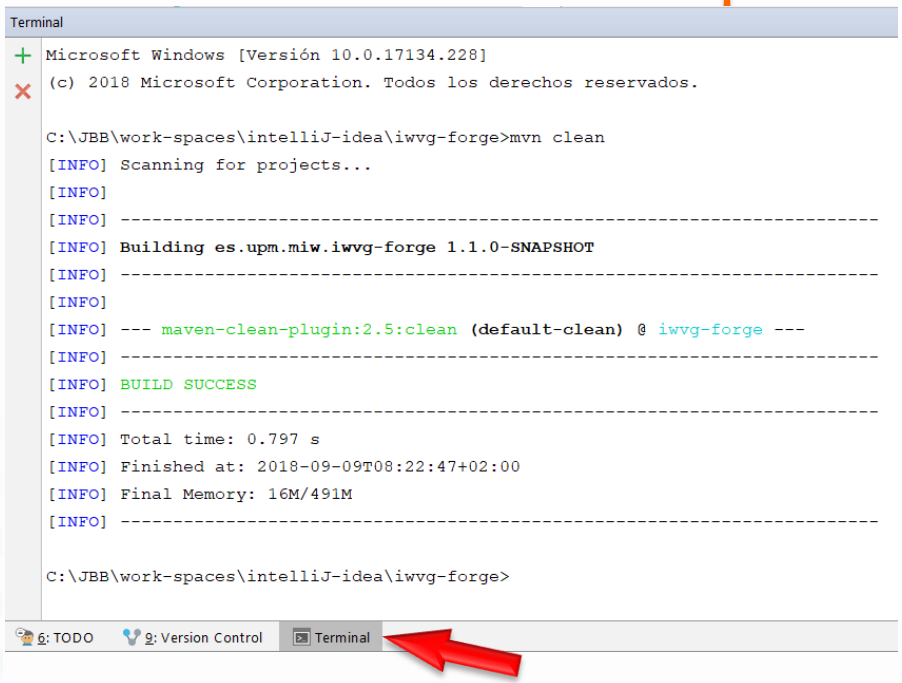
- ***Cliente:***
  - ***https://git-scm.com/download/win***

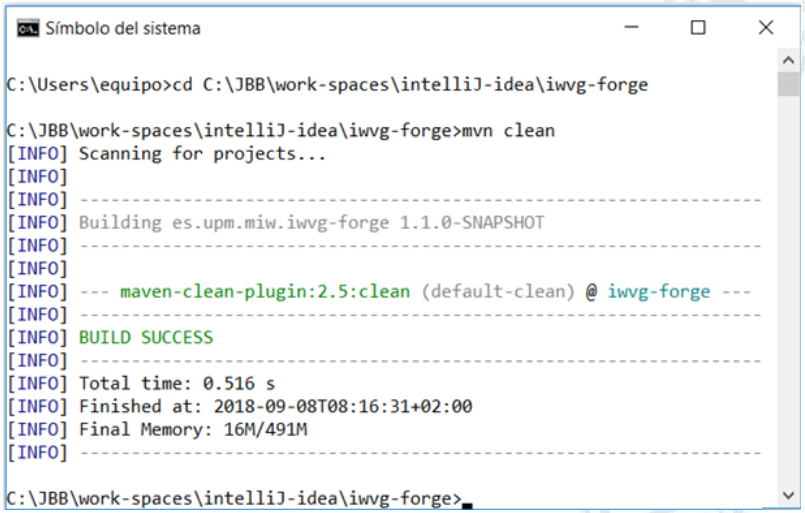
- ***GUI:***
  - ***https://git-scm.com/downloads/guis/***
  - ***https://www.gitkraken.com/***

Instalación del Entorno. Primeros pasos

1. Instalar SDK de Java. Definir variables de entorno
2. Instalar *Maven*. Definir variables de entorno
   - Se manejará maven desde la consola, nos facilitará la creación de scripts
3. Instalar cliente Git
4. Instalar IntelliJ IDEA
5. Crear la carpeta de los workspace
6. Crear un proyecto nuevo tipo maven en el workspace. Se ofrece una plantilla a modo de ejemplo:
   - https://github.com/miw-upm/iwvg-ecosystem
   - Recordar cambiar el nombre de la carpeta y del proyecto en el fichero *pom.xml*
7. Importar el proyecto desde IntelliJ IDEA
   - ***Cerrar proyecto si estuviese abierto***
   - ***>>Import Project,*** y seleccionar la carpeta del proyecto
   - > marcar ***Create Project from external model,*** elegir ***Maven***
   - ***> Next… Finish***

Primeros pasos: terminal





Clonar un proyecto Maven desde Github

- Clonar en repositorio en tu equipo:
  1. Situarse en una carpeta raíz donde se encuentran los proyectos, mediante la consola:
     - **>cd %ruta-de-la-carpeta%**
  2. Clonar el repositorio, se visualizará el contenido de la rama por defecto:
     - ***>git clone https://github.com/miw-upm/IWVG-ecosystem***
  - Actualizar el repositorio local con los cambios del remoto
    - ***>git fetch origin***
- Importar el proyecto desde IntelliJ IDEA
  - ***Cerrar proyecto si estuviese abierto***
  - ***>>Import Project,*** y seleccionar la carpeta del proyecto
  - > marcar ***Create Project from external model,*** elegir ***Maven***
  - ***> Next… Finish***

Ejercicio

- Crear un proyecto maven, llamado *ide1*
- Crear una clase Java en un paquete
- Probar comandos maven por consola

Gestión de registros: Log4j

- Un log es un registro de un evento ocurrido en un instante dado. Se suele usar para registrar datos o información sobre quién, qué, cuándo, dónde y por qué (who, what, when, where y why) un evento ocurre en una aplicación.
- Log4j es una biblioteca ***open source*** desarrollada en Java por Apache que permite elegir la salida y el nivel de granularidad de los mensajes o ***logs***.
- ***http://logging.apache.org/***
- Niveles de prioridad
  - FATAL: mensajes críticos del sistema
  - ERROR: mensajes de error de la aplicación
  - WARN: mensajes de alerta, pero que no afectan al correcto funcionamiento
  - INFO: mensajes de información
  - DEBUG: se utiliza para escribir mensajes de depuración.
  - TRACE: se utiliza para mostrar mensajes con un mayor nivel de detalle que debug.
- ***Appenders***. Puede una o varias salidas de destino
  - Consola (Console); Fichero (File); Base de datos (JDBC, JPA, MongoDB…); Correo (SMTP)
- Configuración. Permite varios sistemas de configuración (***properties***, *xml, yaml, …*)
- Ejemplo LoggerDemo
- ~~***System.out.println()***~~ ***queda encarecidamente desaconsejado!!!***

Pruebas

- Tipos
  - **Pruebas Unitarias**. Los desarrolladores prueban correcto funcionamiento de un módulo de código o clase independientemente del resto. Si existen dependencias se rompen con los mocks, son clases que simulan a otras y aportan una funcionalidad limitada
  - **Pruebas de Integración**. Los desarrolladores prueban los diferentes componentes que dependen de otros componentes
  - **Pruebas Funcionales**. Los desarrolladores prueban al sistema como un todo
  - **Pruebas de Aceptación**. Los clientes prueban las versión entregada
- Característica de calidad:
  - Automática. Clases que prueban clases
  - Cobertura: % de líneas de código ejecutadas (>70%-80%)
  - Repetibles. Cuando parte del código ha sido modificado, se vuelven a lanzar las pruebas para comprobar que no se ha alterado su funcionalidad
  - Independiente. Se prueban los módulos por separado

JUnit 5

- JUnit es un framework que nos ayuda a la realización de pruebas unitarias. Fue creado por Erich Gamma y Kent Beck.
  - ***http://www.junit.org/***
- Test Case: Clases de prueba
  - Una clase es un test, si tiene algún método de test
  - Un test es un método con la anotación ***@Test***
  - Los test deben ser independientes entre si, el orden no debe afectar al resultado
  - El nombre del método debe describir el tipo de prueba. Empiezan con la palabra "*test*"
- Test Suites: Contenedor de Test Case o Test Suites. Se crean estructura en árbol
- Ciclo de vida
  - @BeforeAll: Se ejecuta una sola vez antes de la batería de pruebas definida en la clase y el método debe ser ***static***
  - @BeforeEach: Se ejecuta antes de cada uno de los marcados con @Test (es decir, si existen varios test, se ejecuta varias veces). Suele ser una inicialización por todas las pruebas de la clase
  - @Test: Marca un método como prueba
  - @AfterEach: Se ejecuta después de cada uno de los @Test. Suele ser una liberación de recursos
  - @AfterAll: Se ejecuta al final del proceso completo y el método debe ser ***static***

  - @Disabled: Marca un método o una clase completa para que no se ejecute
  - @Tag. Para el filtrado de test
  - @DisplayName. Define un nombre del test propio
  - @RepeatedTest. Repetición de test
  - @ParameterizedTest. Define test parametrizados con diferentes valores

JUnit. Comprobaciones

- assertEquals (valor_esperado, valor_real);
  - Los valores pueden ser de cualquier tipo
  - Si son arrays, no se comprueban elemento a elemento, sólo la referencia
- assertEquals (double_esperado, double_real, double_error);
- assertNotEquals
- assertArrayEquals (array[] esperado, array[] real)
- assertTrue (condición_booleana)
- assertFalse (condición_booleana)
- assertSame (Objeto esperado, Objeto real)
  - Comprueba que son la misma referencia
- assertNotSame
- assertNull (Objeto)
  - Comprueba que el objeto es Null
- assertNotNull (Objeto objeto)
  - Comprueba que el objeto no es Null
- fail ()
  - Falla siempre
- assertThrows
  - Se debe lanzar una excepción
- assertDoesNotThrow
  - No se debe lanzar una excepción
- assertTimeout

Organización

- Paquetes
  - Fuentes: src/main/java/**
  - Pruebas: src/test/java/**
  - Se puede lanzar un solo test, los test de una clase o los test de un paquete y subpaquetes
- Léxico
  - Pruebas Unitarias
    - Si la clase se llama ***Test*, se ejecutarán en la fase Maven de *test*
  - Pruebas de Integración
    - Si la clase se llaman ***IT*, se ejecutan en la fase Maven de *integration-test*
- Metodología de Trabajo para mantenimiento del código
  1. Modificar el código de la clase y los test afectados
  2. Ejecutar los test de la clase para comprobar que todo sigue igual
  3. Ejecutar el test del paquete, para comprobar que todo sigue igual
  4. Ejecutar todos los test
- Ejemplos de test
  - Point: PointTest
  - DecimalCollection: DecimalCollectionTest
- Lanzamiento de test con IntelliJ
- Ejercicios: Crear los test para las clases del proyecto que falten
  - Fraction
  - User

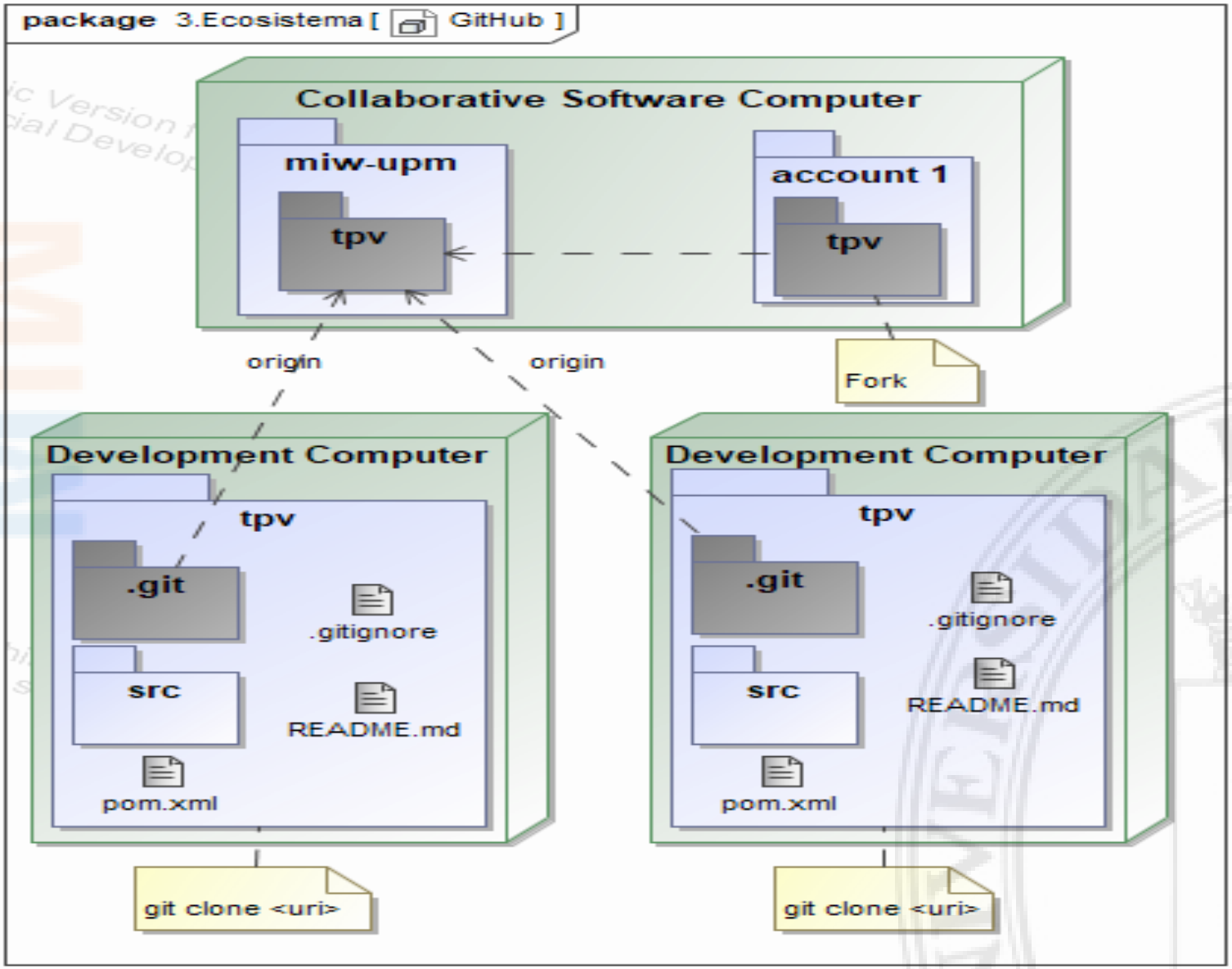
Repositorio de código. Introducción

- Tipos
  - Distribuidos. Aumenta la flexibilidad pero complica la sincronización y gestión. Ejemplos: Git, Mercurial... En la actualidad se están imponiendo estos sistemas
  - Centralizados. Dependiente de un responsable. Facilita la gestión pero reduce la potencia y flexibilidad. Ejemplos: CVS, Subversion...
- En la nube
  - GitHub. *https://github.com/*. Sitio web que lleva integrada la forja. Para código abierto es gratuito, existe una versión de pago para proyectos privados
  - Google Code. *https://code.google.com/intl/es/*
  - SourceForge. *http://sourceforge.net/*
  - Otras: Bitbucket (Atlassian), BerliOS, Gforce, Savannah...

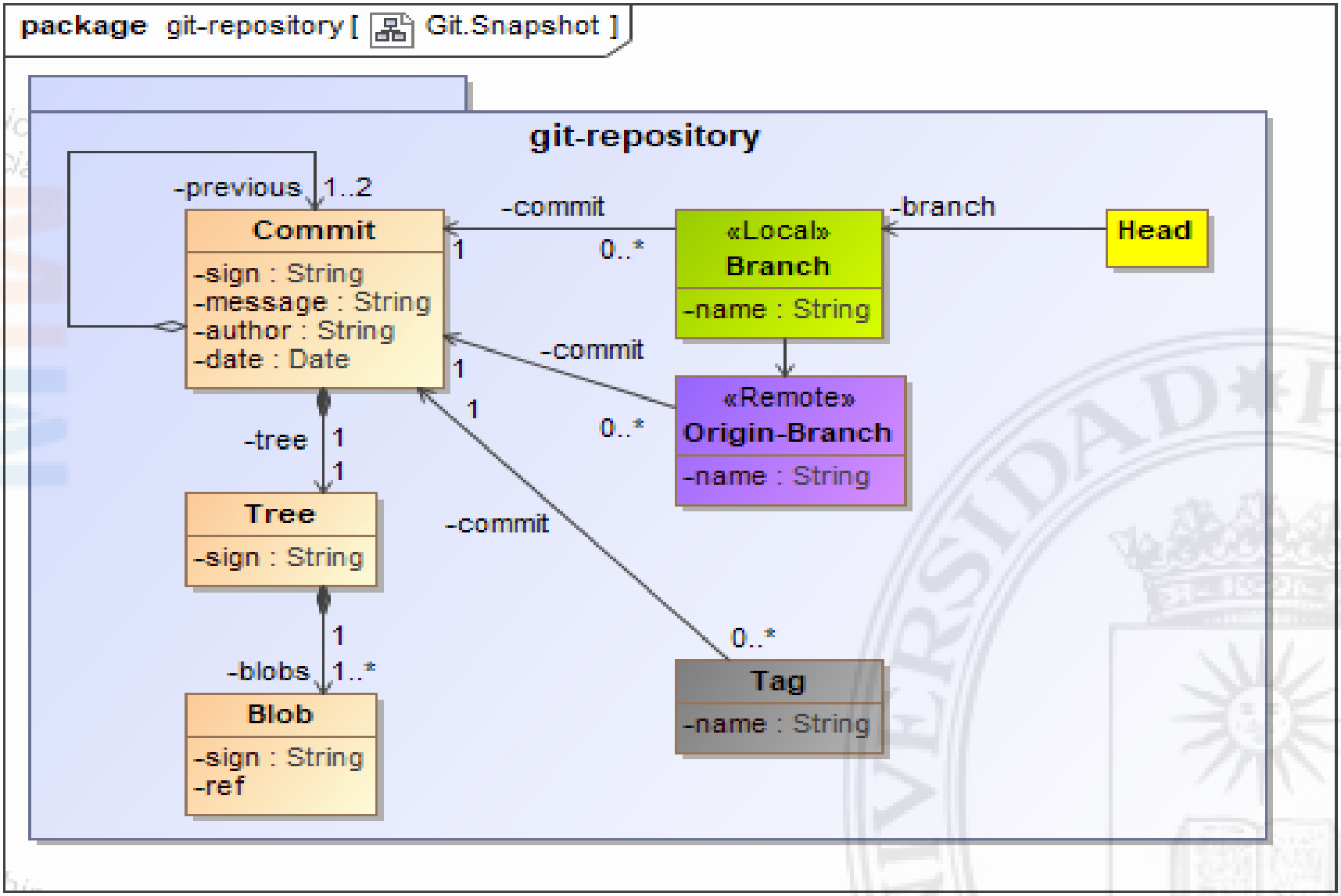
Sistema de control de versiones: Git

- https://github.com/
- Nace en 2005, tomando como experiencia el proyecto Bitkeeper (propietario).
  - En 2008 nace GitHub, Forja en Web con repositorio basado en Git.
  - En el 2018 Microsoft adquiere GitHub por 7.500 millones de dólares.
- Documentación: *https://git-scm.com/book/es/v2*.
- Características
  - Control de versiones distribuido.
  - Muy fiable, casi imposible perder el proyecto.
  - Trabaja sin necesidad de conexión al remoto, muy rápido. Se podrá sincronizar con el remoto, **pero con asistencia...**
  - **Snapshot: instantánea (commits)**
- *Git-CLI:*
  - *https://git-scm.com/download/win*
- *GUI:*
  - *https://git-scm.com/downloads/guis/*
  - *https://www.gitkraken.com/*

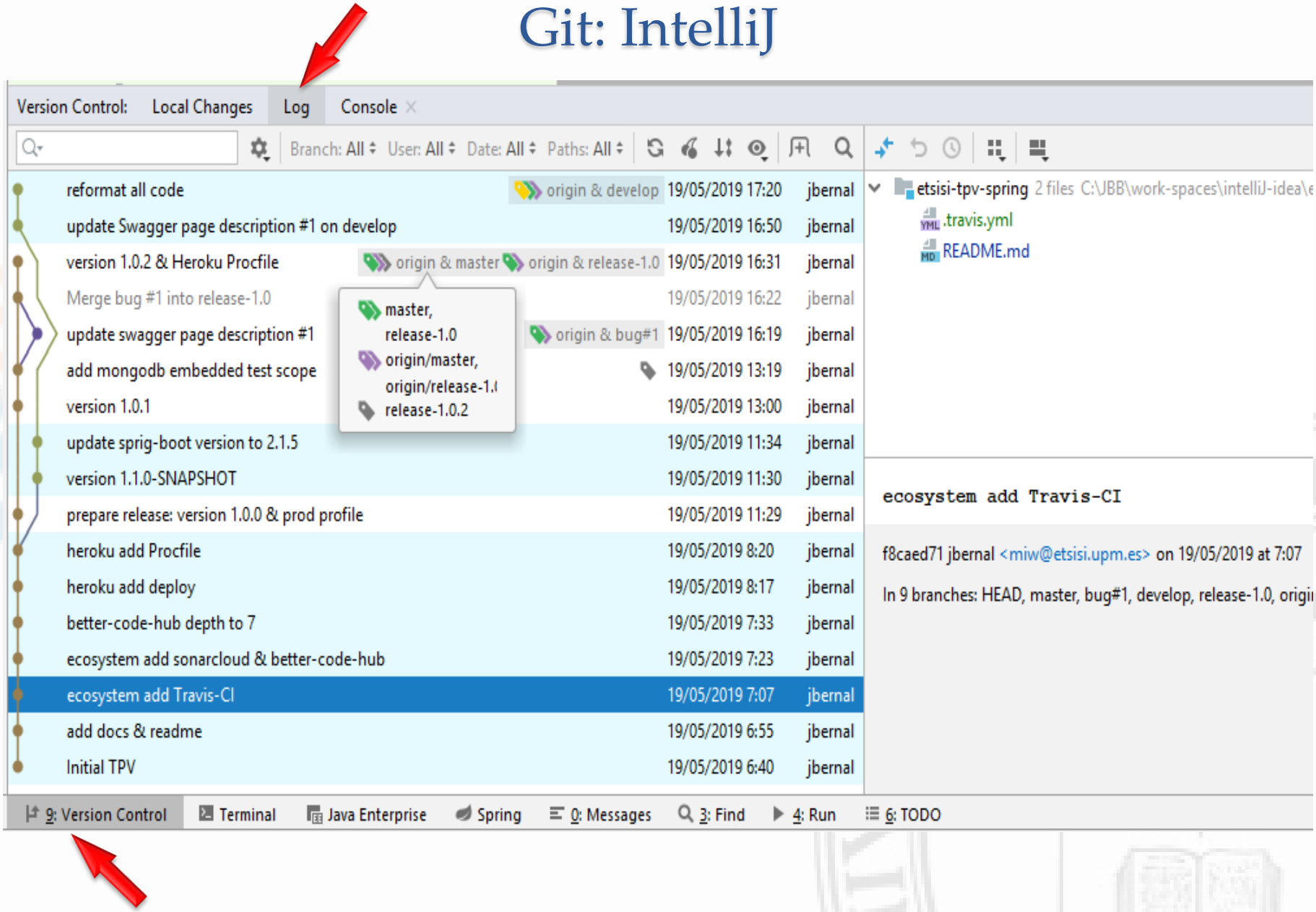
Sistema de control de versiones: Git



Instantánea (snapshot)



Git: IntelliJ



Git-CLI

- *>git config --global user.name " … "*
- *>git config --global user.email " … "*
- *>git config credential.helper store*

- *>git clone <uri>*
- *>git remote –v*
- *>git remote add origin <uri>*

- *>git checkout –b* **<branch>**
- *>git add --all*
- *>git commit -m "mi mensaje de commit"*
- *>git merge –-no-ff -m "<**message**>" <**branch**>*

- Branches exercises: Note 0
  - *master* & *develop* en los commits de las remotas, respectivamente
- Crear rama local "*tmp*" y "*dev*" asociada al commit de *develop*
  - Activar "tmp", crear varios *commits*… y fusionarla con "dev"…
  - Activar la rama *develop* y eliminar la rama *tmp & dev*

Git-CLI

- Sitio web: *https://git-scm.com/*
- Ayuda
  - *>git help --all*
  - *>git* <command> *–h*
- *Consultas*
  - *>git config --list*
  - *>git status*
  - *>git remote –v*
    - *v:--verbose*
  - >git log
    - *q para salir*
  - >git reflog
    - *q para salir. Referencia de los commits*
- Configurar Git
  - *>git config --global user.name " … "*
  - *>git config --global user.email " … "*
  - *>git config credential.helper store*
    - *Para guardar las credenciales, después de logearse*

Git-CLI

- Sitio web: *https://git-scm.com/*
- Ayuda
  - *>git help --all*
  - *>git* <command> *–h*
- *Consultas*
  - *>git config --list*
  - *>git status*
  - *>git remote –v*
    - *v:--verbose*
  - >git log
    - *q para salir*
  - >git reflog
    - *q para salir. Referencia de los commits*
- Configurar Git
  - *>git config --global user.name " … "*
  - *>git config --global user.email " … "*
  - *>git config credential.helper store*
    - *Para guardar las credenciales, después de logearse*

Git

- Crear repositorio con todos los ficheros
  - >*git init*
  - >*git add --all*
  - >*git commit -m "mi mensaje de commit inicial"*
- Commit (todos los ficheros con mensaje)
  - >*git add --all*
  - >*git commit –m "mensaje"*
  - >*git commit -am "add + commit"*
    - *No vale para ficheros nuevos*
- Ramas
  - Mostrar ramas: >*git branch*
  - Creación de rama y cambio de rama: >*git checkout –b* **<branch>**
  - Cambio de rama: >*git checkout* **<branch>**
  - Borrado de rama: >*git branch –d* **<branch>**
  - Crear rama en el commit referenciado: >*git checkout –b* **<branch> <commit>**
- Fusión de ramas:
  - Fusión sobre la rama actual: >*git merge* **<branch>**
  - Forzando commit (not fast forward*):* >*git merge –-no-ff -m* "**<message>**" **<branch>**

Git

- Recuperación de situaciones anómalas
  - Incorpora los nuevos cambios al último commit: >*git commit --amend --no-edit*
  - Cambiar el mensaje del último commit: >*git commit --amend -m "new message"*
  - Volver a un commit concreto:
    - >*git reset --hard HEAD*
    - >*git reset --hard* **<commit>**
- Repositorios remotos
  - Consultar los repositorios remotos: >*git remote -v*
  - Añadir repositorio remoto: >*git remote add origin* **<url repository>**
  - Eliminar un repositorio remote: >*git remote rm origin*
  - Subir una rama al remoto: >*git push origin* **<branch>**
  - Subir todas las ramas*:* >*git push origin –all*
  - Bajarse todas las ramas remotas: >*git fetch origin*
  - Bajarse una rama: >*git fetch origin* **<branch>**
  - Subir una rama de manera forzada ***PELIGROSO!!!****:*
    - >*git push origin* **<branch>** *--force*
- Etiquetas
  - Consulta de etiquetas*:* >*git tag*
  - Crear etiqueta: >*git tag –a* **<etiqueta>** *-m "mensaje"*
  - Borrado de etiqueta: >*git tag –d* **<etiqueta>**
  - Subir etiqueta al remoto: >*git push origin* **<etiqueta>**

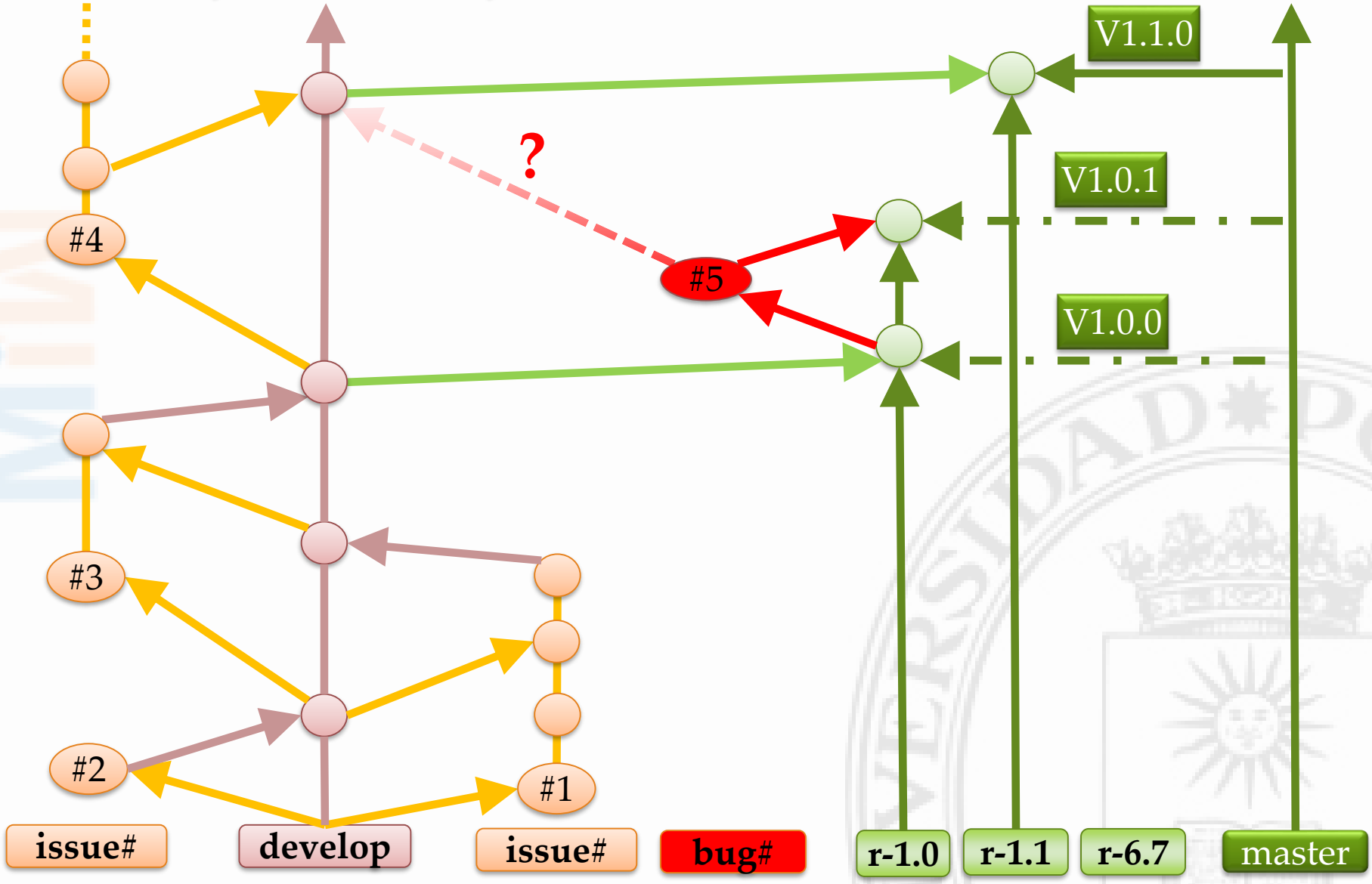
Flujo de Trabajo (workflow)

- El flujo de trabajo de un sistema de control de versiones indica cómo se relacionan los miembros del equipo para colaborar entre sí en el desarrollo del software colaborativo
  - **Flujo de trabajo centralizado (*Centralized Workflow*)**. Todos comparten el código de un repositorio central, o en Git, todos comparten la única rama… Normalmente, se realiza una fusión al final del día, el último se come la fusión!!!
  - **Flujo de trabajo ramificado (*Git Workflow*)**. Gracias a los repositorios distribuidos, ha tomado auge por la gestión efectiva de las ramas
  - **Flujo de trabajo con bifurcación (*Forking Workflow*)**. El usuario bifurca el proyecto realizando una copia completa del repositorio. Realiza las ampliaciones y realiza una petición de agregación (*pull request*). Se abre una discusión, y el dueño del repositorio decide la fusión. Normalmente, se utiliza en proyectos abiertos

Flujo de Trabajo ramificado (Git Workflow)

- Rama ***master***: apunta al último *commit* de producción, es decir, a la última versión del código liberado
- Rama ***develop***: es la rama de desarrollo, siempre estable, código que cumple los requisitos de calidad y con todos los test superados
- Ramas ***issue#XX***, *feature#XX*, *topic#XX*: parte de develop, se añaden las nuevas características y vuelve a ***develop*** cuando vuelve a ser estable
- Ramas ***release#XX***: se utiliza para estabilizar un código para salir a producción. Una vez terminada, se fusiona con ***master***
- Ramas ***bug#XX***: se utilizan para corregir errores (*bugs)* en el código en producción

Flujo de Trabajo ramificado (Git Workflow)

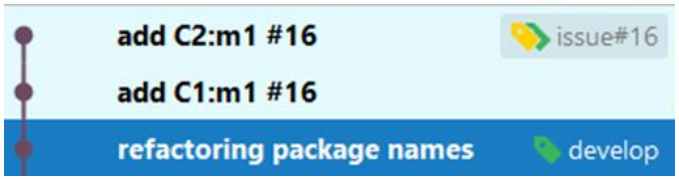


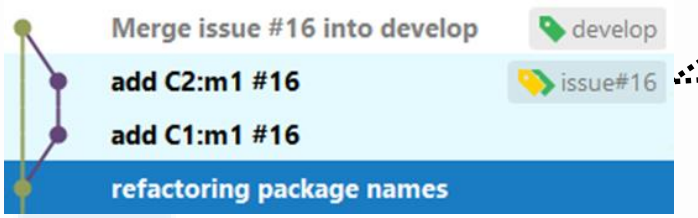
Git

útil para seguimiento del issue#


```
>git checkout -b issue#16
>git add -all
>git commit -m "add C1:m1 #16"
-> Visualizar el resultado
```

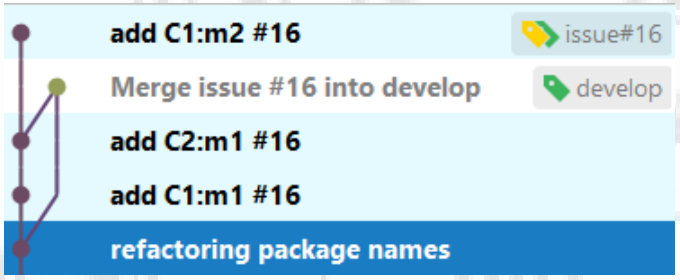
```
>git commit -am "add C1:m2 #16"
-> Visualizar el resultado
```






```
>git checkout develop
>git merge --no-ff -m "Merge issue #16 into develop" issue#16
>git checkout issue#16
-> Visualizar el resultado
```

```
>git add -all
>git commit -m "add C2:m1 #16"
-> Visualizar el resultado
```




```
>git add -all
>git commit -m "add C2:m2 #16"
>git checkout develop
>git merge --no-ff -m "Merge issue #16 into develop" issue#16
-> Visualizar el resultado
```

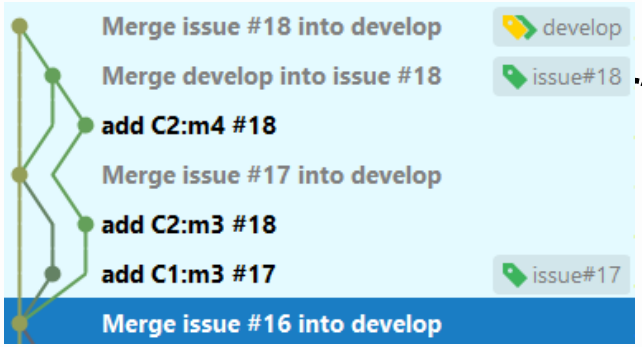
Git


```
>git branch issue#18
>git checkout -b issue#17
>git commit -am "add C1:m3 #17"
>git checkout issue#18
>git commit -am "add C2:m3 #18"
>git checkout issue#17
-> Visualizar el resultado
```

```
>git checkout develop
>git merge --no-ff -m "Merge issue #17 into develop" issue#17
-> Visualizar el resultado
```

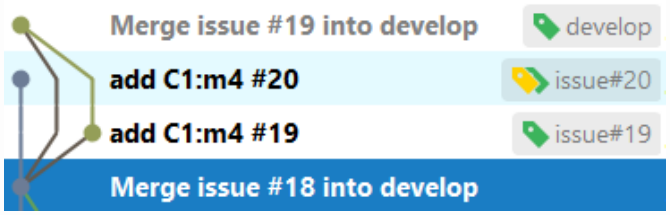

```
>git checkout issue#18
>git commit -am "add C2:m4 #18"
-> Visualizar el resultado
```

```
git merge -m "Merge develop into issue #18" develop
-> Visualizar el resultado
```




```
>git commit -am "add C2:m2 #16"
>git checkout develop
>git merge --no-ff -m "Merge issue #16 into develop" issue#16
-> Visualizar el resultado
```

Git




```
>git merge -m "Merge develop into issue #20"
```

```
C:\JBB\work-spaces\intelliJ-idea\iwvg-forge>git merge -m "Merge develop into issue #20" develop
Auto-merging src/main/java/es/upm/miw/iwvg/ecosystem/git/C1.java
CONFLICT (content): Merge conflict in src/main/java/es/upm/miw/iwvg/ecosystem/git/C1.java
Automatic merge failed; fix conflicts and then commit the result.
```

```
public class C1 {
    public String m1() { return "C1:m1"; }

    public String m2() { return "C1:m2"; }

    public String m3() { return "C1:m3"; }

    public String m4() {
<<<<<<< HEAD
        return "C1:m4 of issue#20";
=======
        return "C1:m4";
>>>>>>> develop
    }
}
```

-> Editar C1 y dejar su contenido correcto

```
>git add -all
>git commit
```

```
Merge develop into issue #20

 Conflicts:
        src/main/java/es/upm/miw/iwvg/ecosystem/git/C1.java
#
# It looks like you may be committing a merge.
# If this is not correct, please remove the file
#       .git/MERGE_HEAD
# and try again.
```

-> Establecer el mensaje final

<ESC>:wq

-> Visualizar el resultado

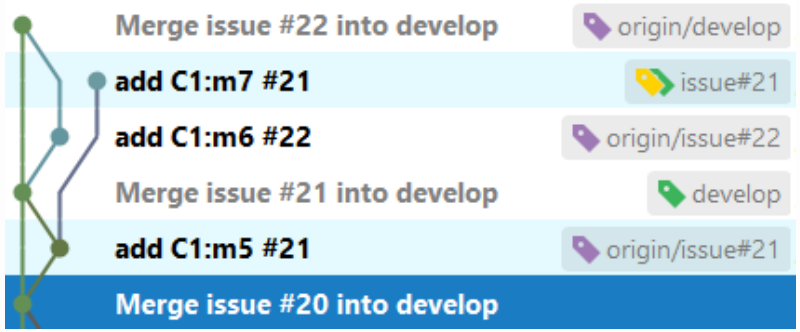
Git. Remotos

-> origin/develop y develop sincronizados

-> origin/develop NO sincronizado
-> Permitido Fast-forward

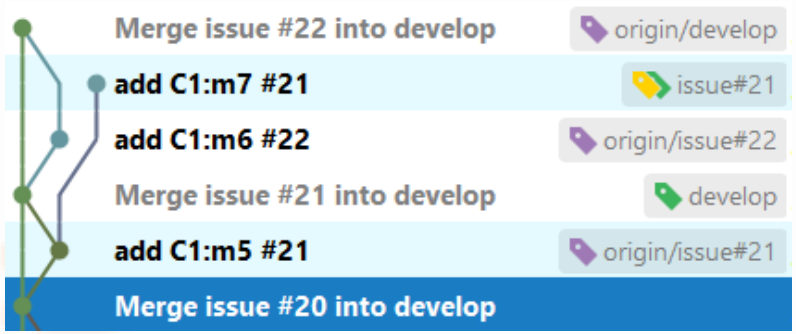

```
>git push origin --all
-> Visualizar el resultado
```

```
>git checkout issue#21
>git commit -am "add C1:m7 #21"
-> Visualizar el resultado
```

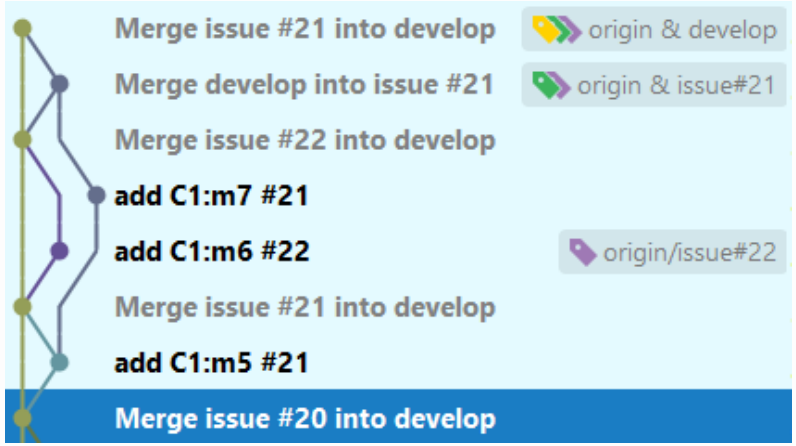



```
>git fetch origin
-> Visualizar el resultado
-> ??? siguientes pasos
```

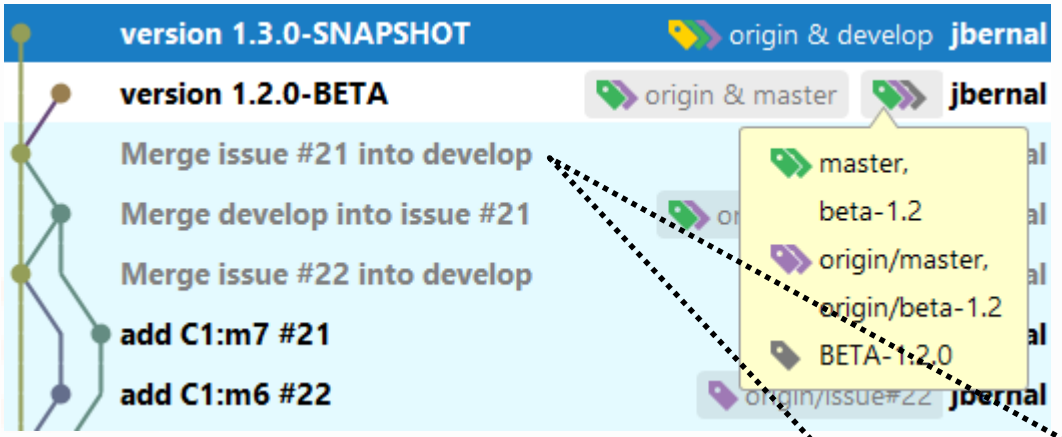
Git

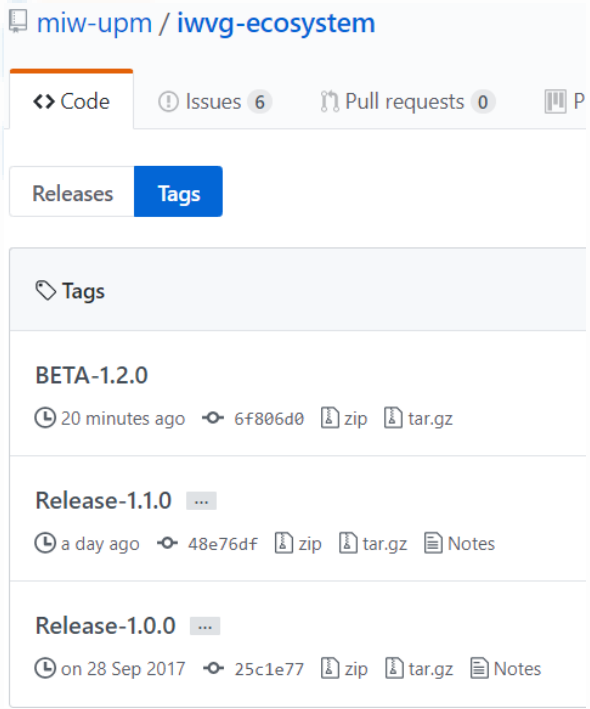



```
>git checkout develop
>git merge origin/develop

>git check issue#21
>git merge -m "Merge develop into issue #21" develop
>git checkout develop
>git merge --no-ff -m "Merge issue #21 into develop" issue#21
-> Visualizar el resultado
>git push origin --all
```



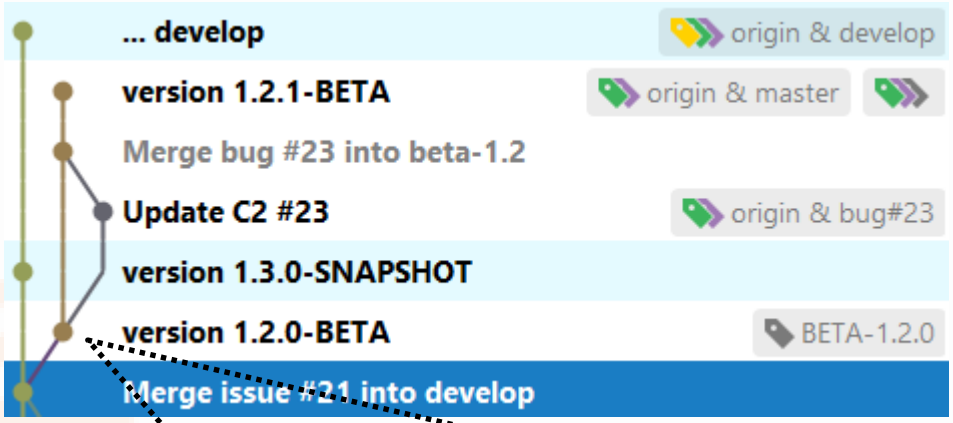
Git



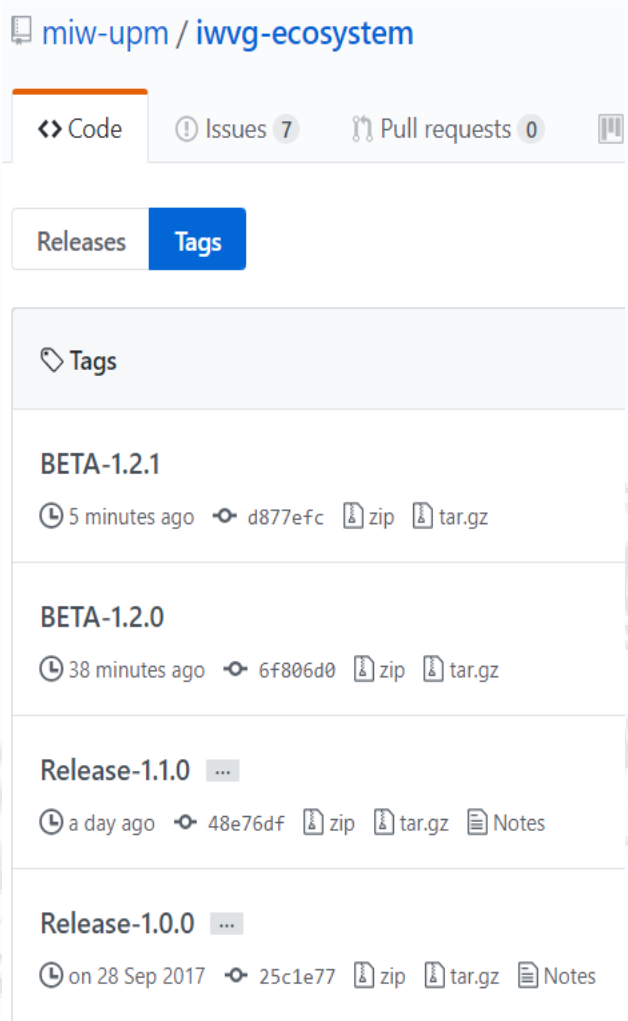



```
>git checkout -b beta-1.2
>git commit -am "version 1.2.0-BETA"
>git push origin beta-1.2
>git tag -a BETA-1.2.0 -m "BETA-1.2.0"
>git push origin BETA-1.2.0
>git checkout master
>git reset --hard 6f806d0
-> Visualizar el resultado para ver si es el esperado
>git push origin master --force

>git checkout develop
>git commit -am "version 1.3.0-SNAPSHOT"
>git push origin develop
-> Visualizar el resultado
```

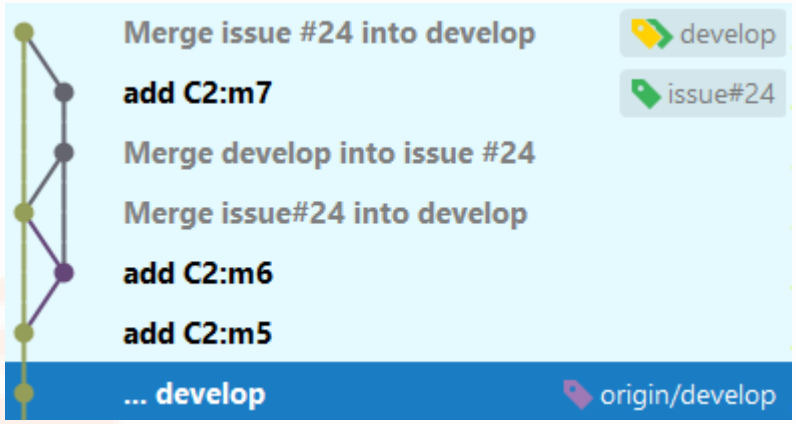
Git




```
>git checkout beta-1.2
>git checkout -b bug#23
>git commit -am "Update C2 #23"
>git checkout beta-1.2
>git merge --no-ff -m "Merge bug #23 into beta-1.2" bug#23
>git commit -am "version 1.2.1-BETA"
>git push origin beta-1.2
>git tag -a BETA-1.2.1 -m "BETA-1.2.1"
>git push origin BETA-1.2.1
>git checkout master
>git reset --hard d877efc
-> Visualizar el resultado para ver si es el esperado
>git push origin master --force

>git checkout develop
>git commit -am "… develop"
>git push origin develop
-> Visualizar el resultado
```



Git



-> Issue#24 ha realizado su mejora y la da por finalizada

- Errores de issue#24 (4)

Git

-> Issue#25, al intentar sincronizar *origin/develop*, *github* lo rechaza, realiza un *>git fetch origin*, y se encuentra con esta situación

- Errores de issue#25

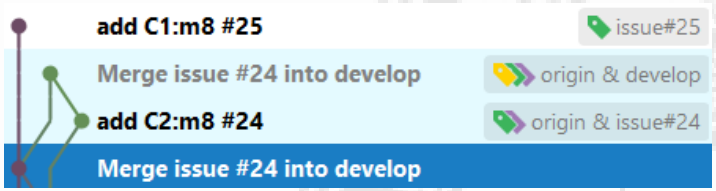
Git


```
-> Issue#26 ha realizado su mejora y la da por finalizada
```

- Errores de issue#26

Git

- Errores
  - Issue#24
    - Realizar *Merge* con *develop* con *fast-forward*
    - En el mensaje, juntar issue#24
    - El *Merge* de *devolp into issue#24* es innecesario
    - *origin/develop* no sincronizada
  - Issue#25
    - Erróneamente, se pensaba que su *develop* local estaba sincronizado
    - Una solución, sería borrar el *develop* local y situarlo en *origin/develop*, y a partir de ahí… hacerlo bien:



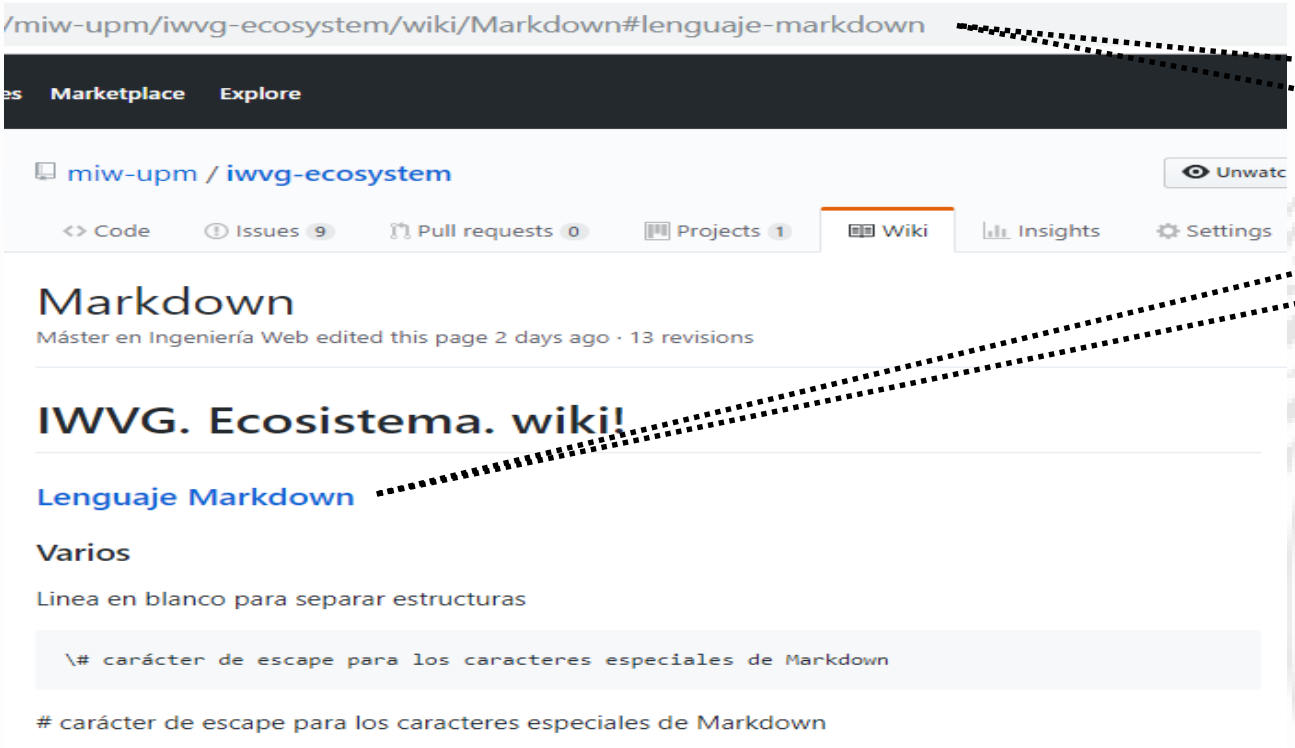
  - Issue#26
    - Ha hecho *Merge* con *develop*, sin incorporar previamente los cambios de *develop* a su *issue*

Ejercicio

- Incorporar los alumnos al repositorio *iwvg-demoX*
  - **5 por proyecto**
- Realizar cambios en paralelo sin conflicto
- Realizar cambios con conflicto

GitHub. Wiki

- Una es un sitio web cuyas páginas pueden ser editadas directamente desde el navegador, donde los usuarios crean, modifican o eliminan contenidos compartidos
- GitHub dispone de Wiki para la generación de documentación rápida y compartida a partir de lenguajes de marcado ligeros
- Dispone de varios lenguajes de marcado... Markdown

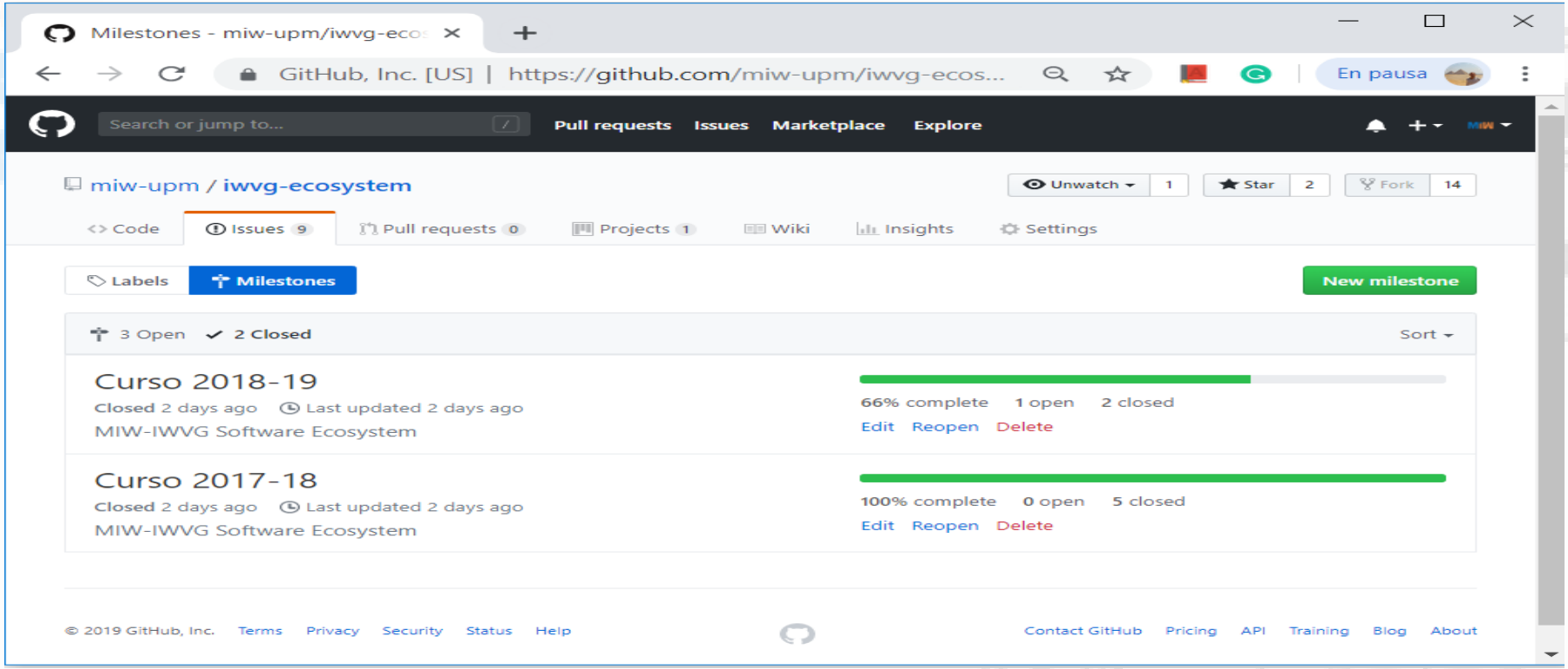


Referencia de la página y capítulo

Todos los encabezados son referenciables

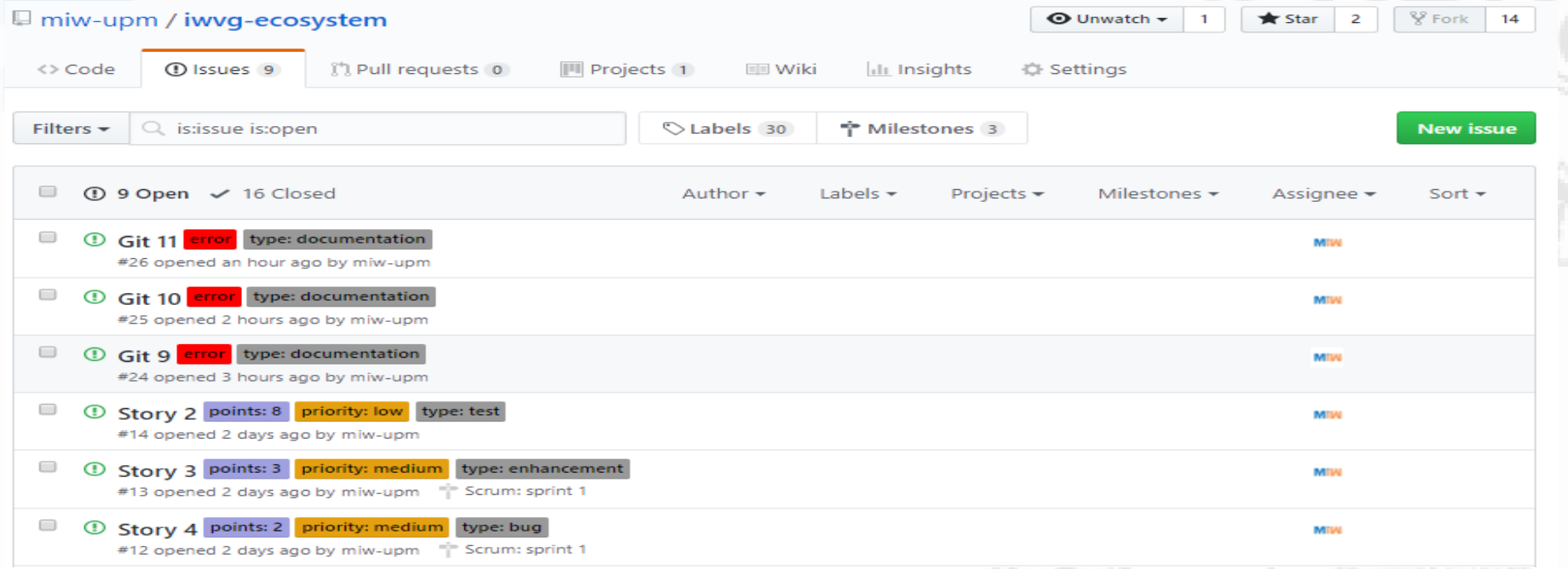
Hitos

- *Milestone*: Hito (título, descripción y fecha). Representa una de fecha de referencia, normalmente asociada a un lanzamiento o finalización de un módulo. Puede estar abierto-cerrado
- Se les puede asociar tickets
- Tiene marcado un nivel de finalización, obtenido por los tickets asociados que se encuentran cerrados



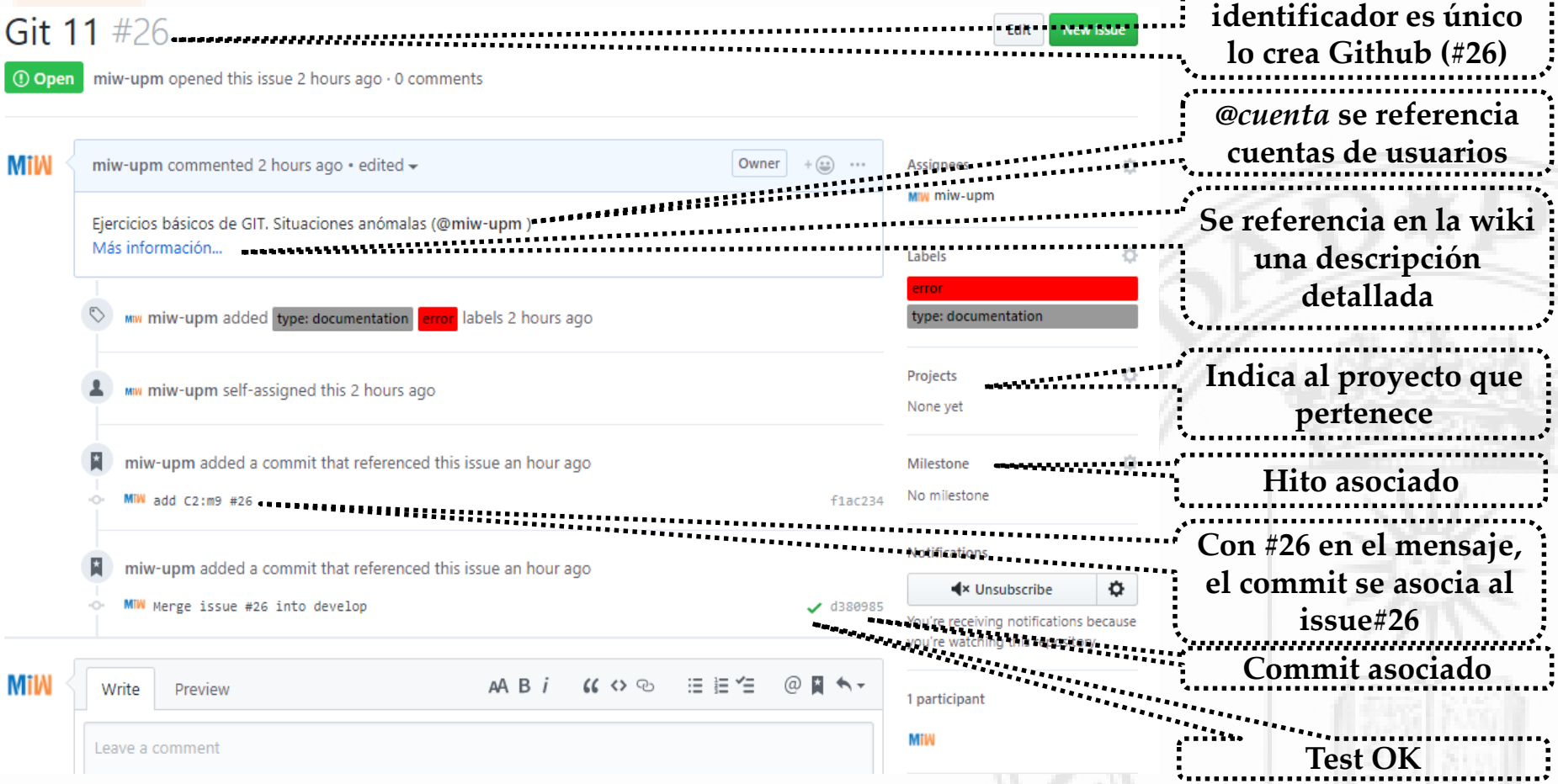
Issues

- Issues: tarea, error, ticket... (título, asignado a..., asociado a un hito..., comentario, etiquetas) (abierto-cerrado)
  - Se abre un foro de comentarios
- Etiquetas: facilitan su identificación
  - Etiquetas de prioridad: priority: high, priority: médium…
  - Etiquetas de tipo: type: test, type: documentation…
- Ticket. Organización
  - Título: representa una referencia a una página de la wiki
  - Descripción muy básica, en la wiki, se detalla en profundidad

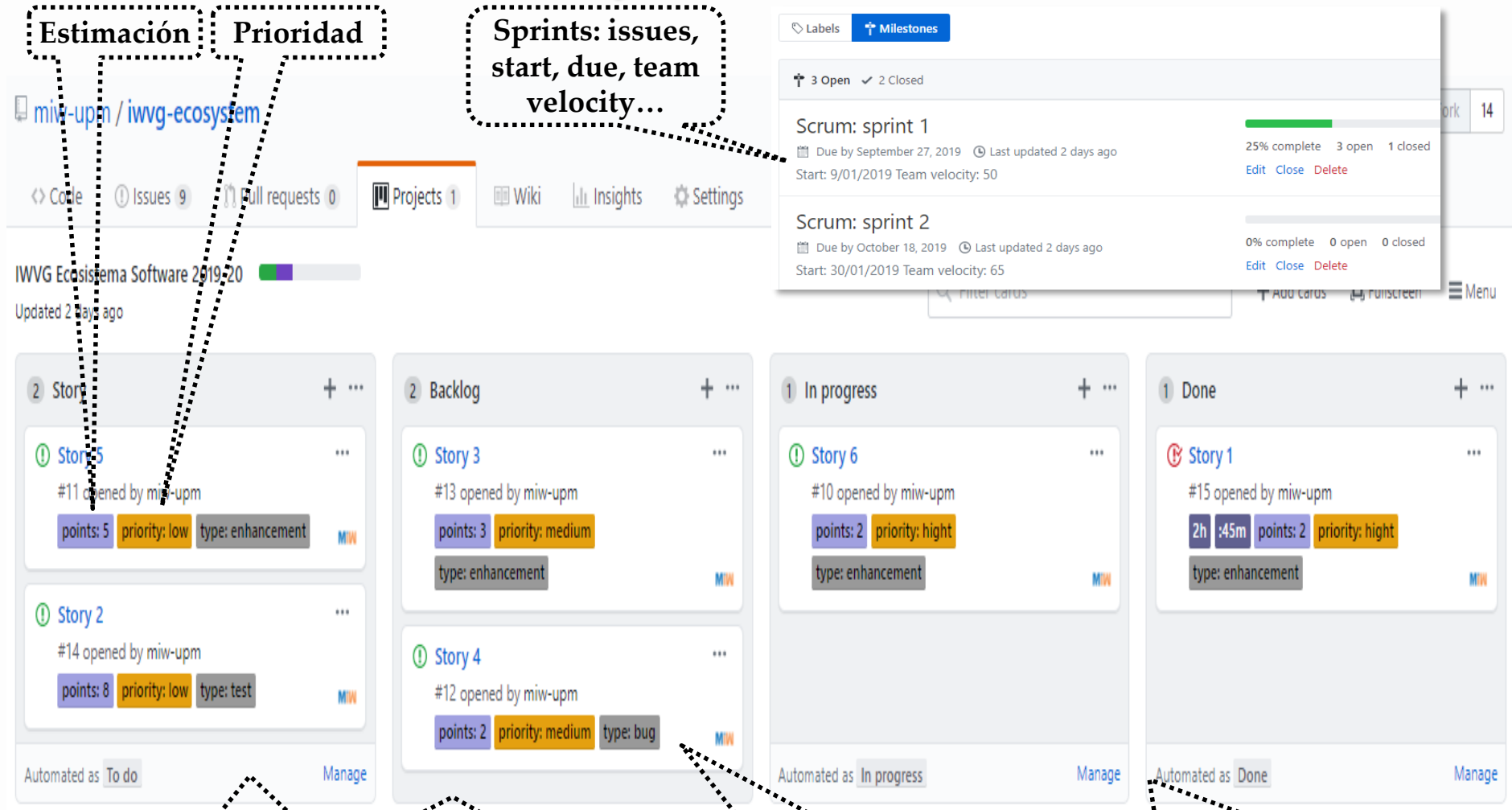


Organización de un ticket

- El ticket es una referencia a una página de la wiki
- Sólo tiene una breve descripción. En la wiki se establece toda la documentación



Proyectos: ≈ Scrum



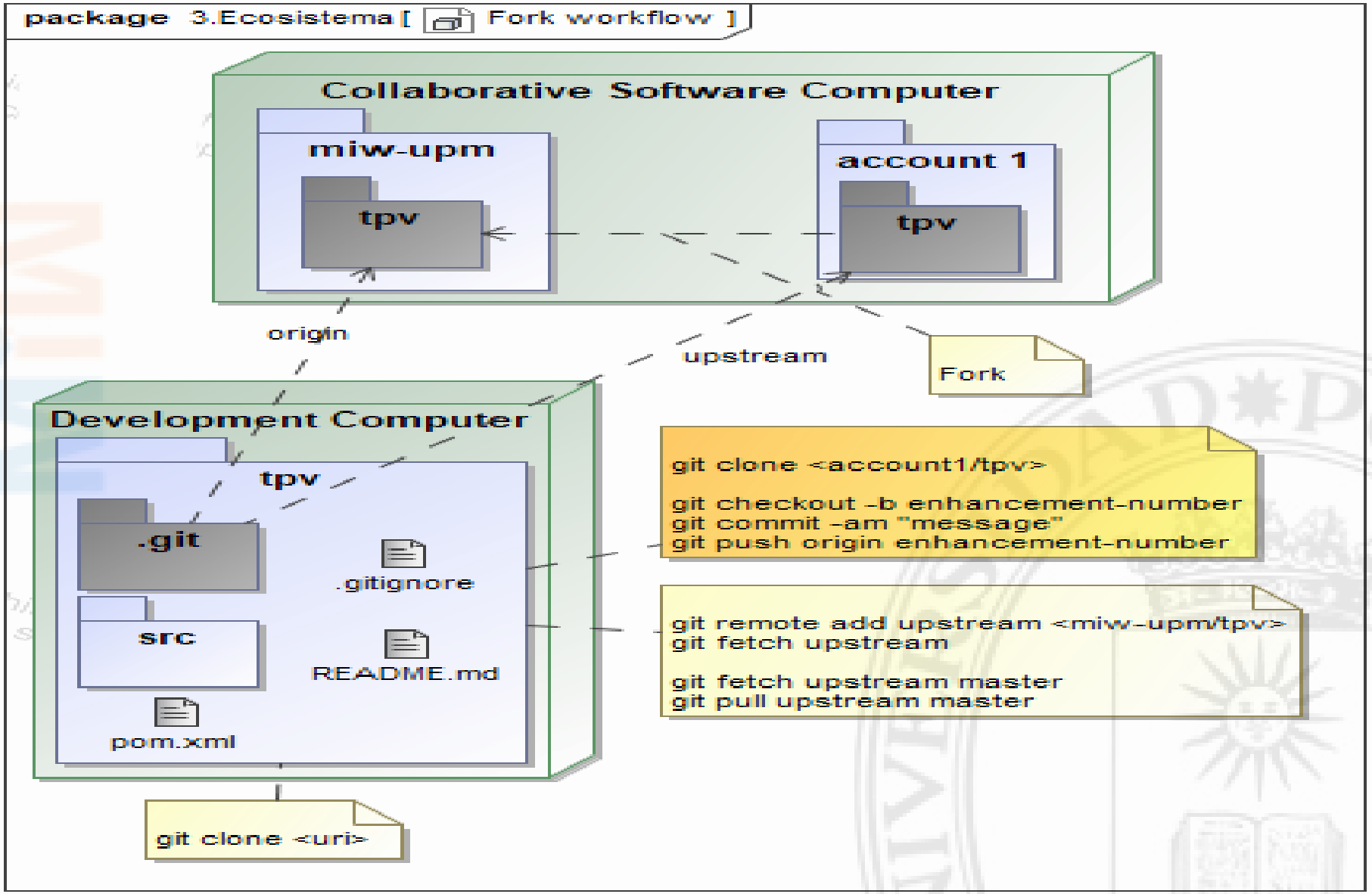
Gestión con GitHub: ≈ Scrum

1. Crear las histórias (*issues)* en el proyecto. Asociarle *prioridad*, *estimación*, *tipo*…
2. Mover las historias a la columna *backlog*, son las que desarrollan en el próximo *sprint*, asociarles el *sprint* (hito).
3. Cuando alguien inicia una historia (*issue)*, se lo asigna y lo mueve a la columna *In progress*.
4. Se crea la rama 'issue#xx' y se programa la historia… un *commit* debe ser reflejado en el historial del *issue*, se añade en el mensaje ' #xx'
5. Cuando la historia se termina, se fusiona con *develop*:
   - git merge –no-ff –m "Merge branch issue #xx into develop" issue#xx
6. Se cierra la historia (*issue#xx*) y se abandona la rama

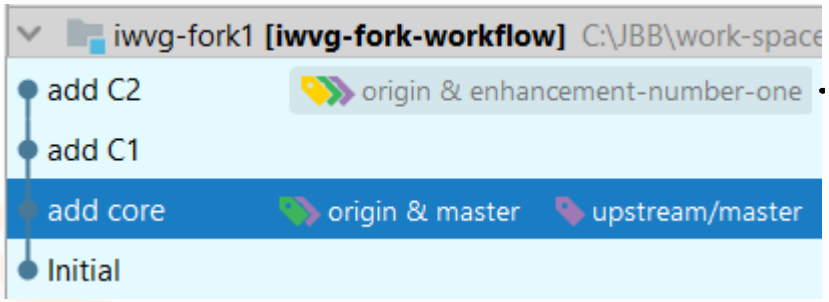
Ejercicio

- Incorporar los alumnos al repositorio *iwvg-git-workflow*
- Realizar el proyecto Demo1

Flujo de Trabajo con bifurcación (Forking Workflow)

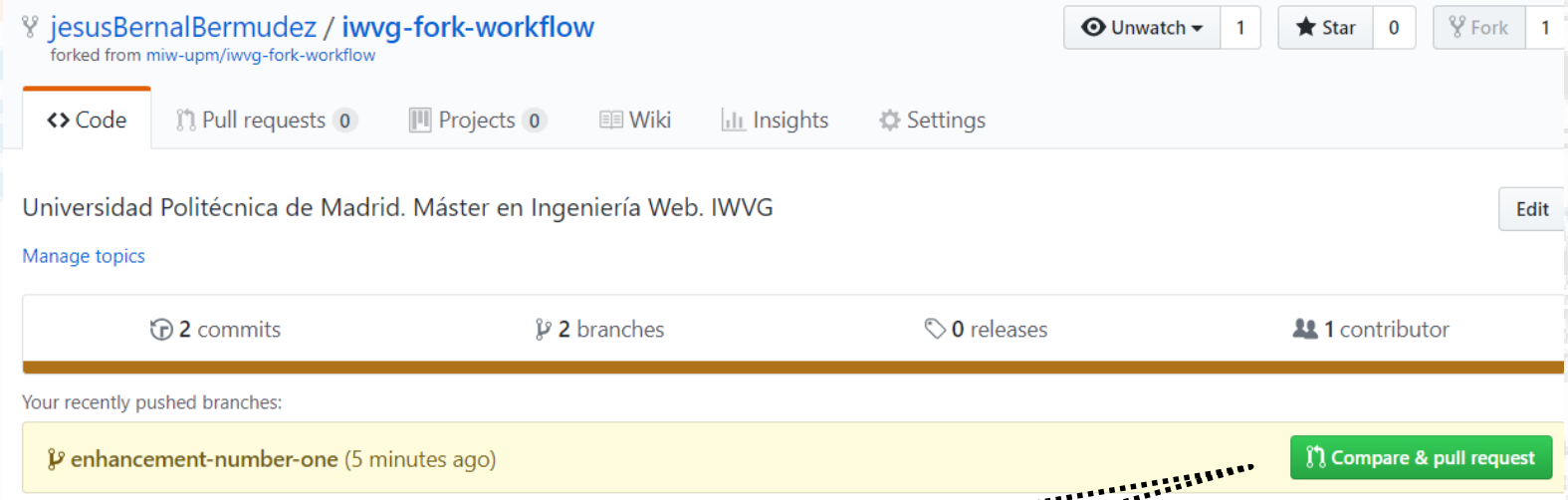


Flujo de Trabajo con bifurcación (Forking Workflow)



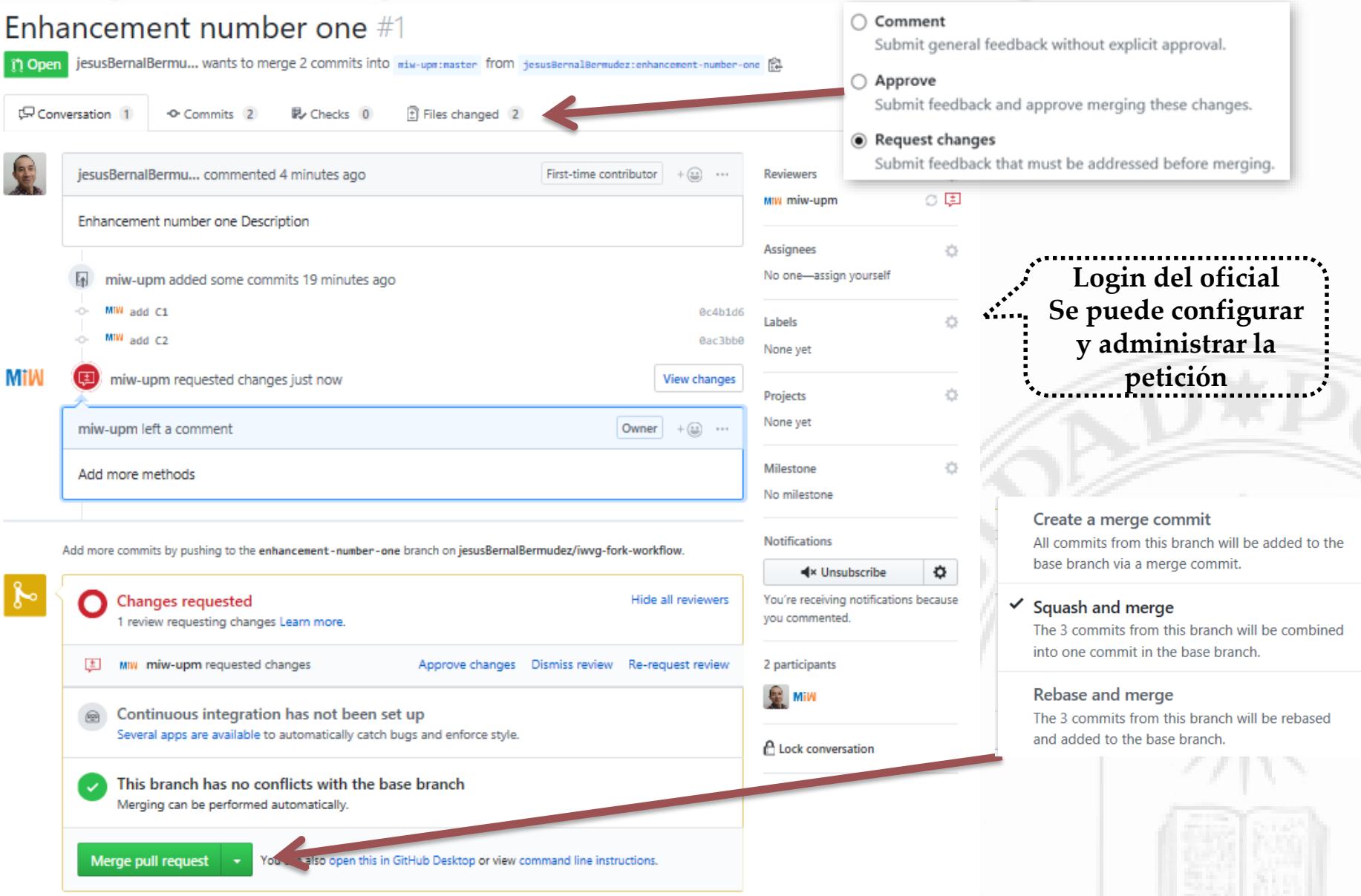
Nombres de ramas descriptivos

git fetch upstream / git pull upstream
…con frecuencia



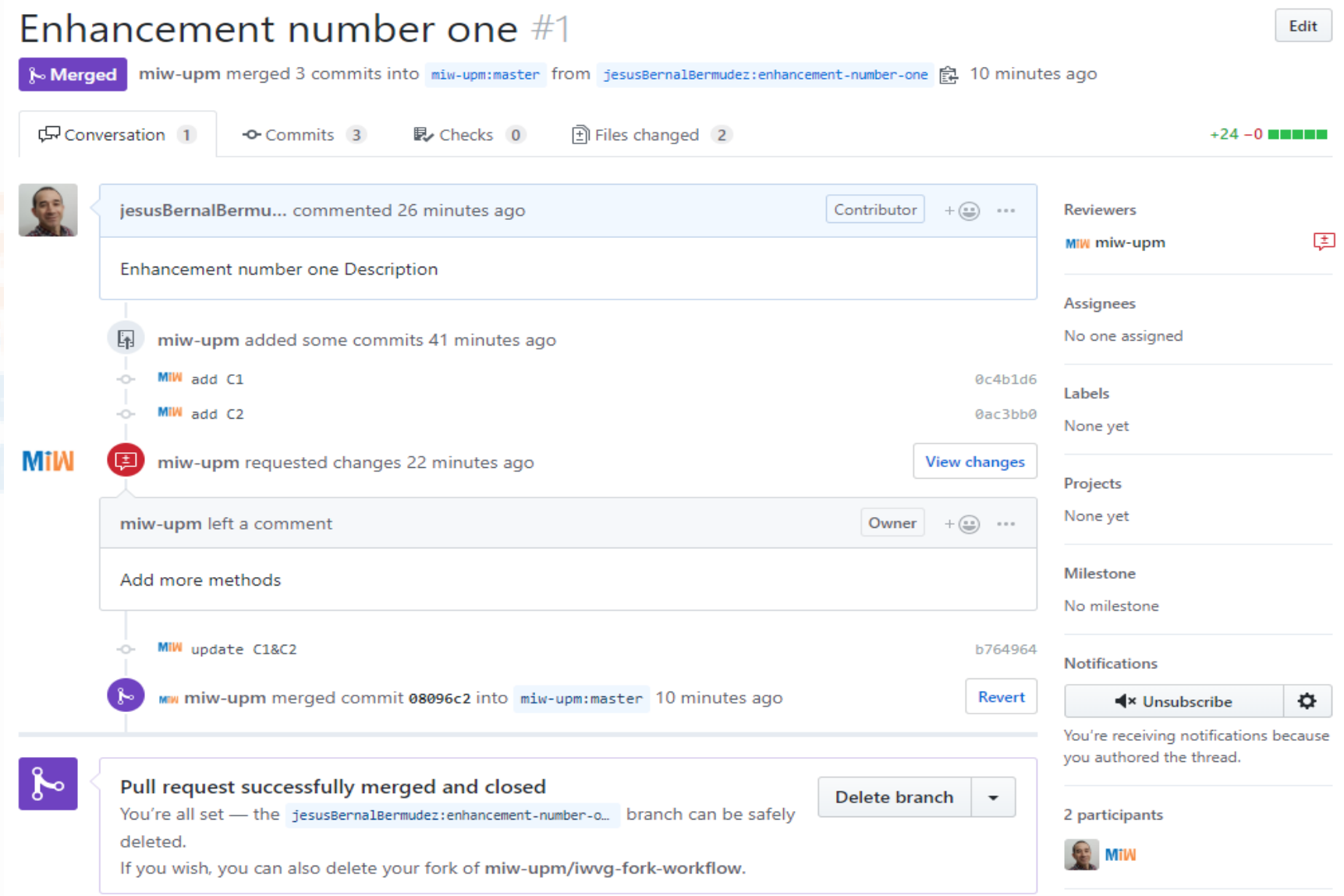
Con login del fork, al subir la rama al remoto, aparece la petición de pull request

Flujo de Trabajo con bifurcación (Forking Workflow)

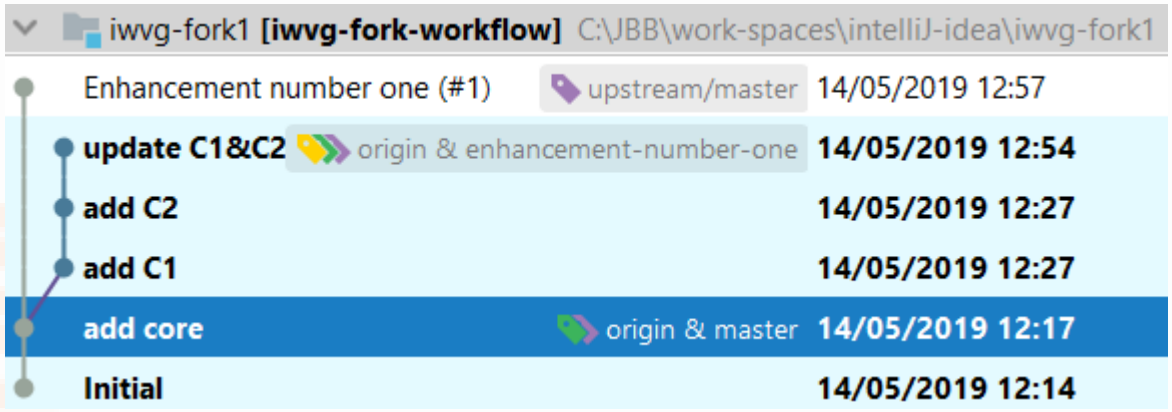


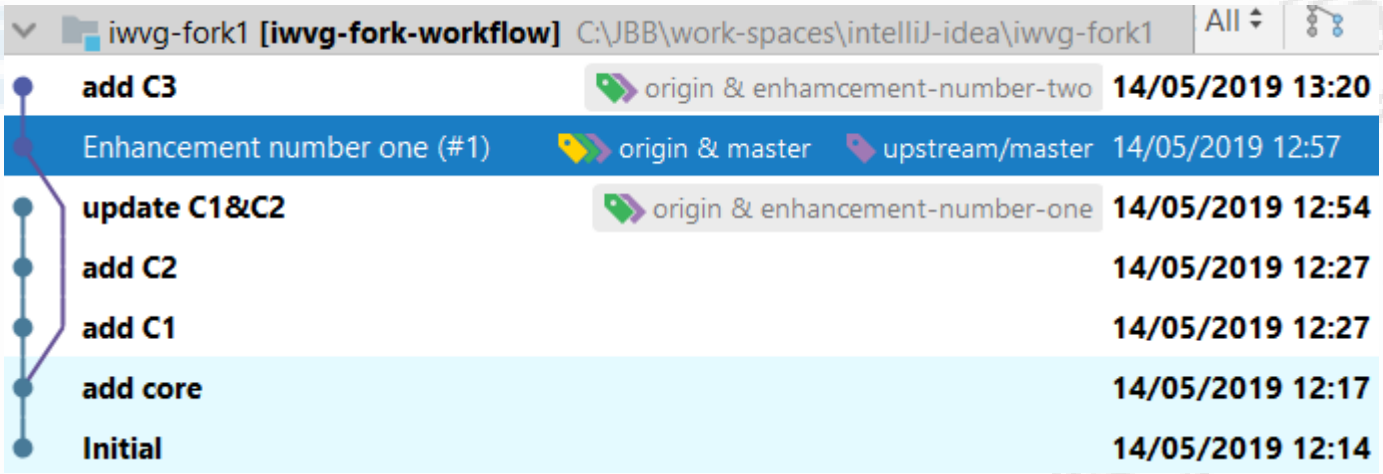
Login del oficial
Se puede configurar y administrar la petición

Flujo de Trabajo con bifurcación (Forking Workflow)



Flujo de Trabajo con bifurcación (Forking Workflow)





Ejercicio

- Los alumnos deben hacer fork al repositorio *iwvg-fork-workflow*
- Deben realizar algún cambio y solicitar un *pull request*
- Observar como evoluciona el código

Integración Continua (CI)

- La integración continua es una práctica de desarrollo de software, propuesto inicialmente por Martin Fowler, la cuál los miembros de un equipo integran su trabajo frecuentemente (diariamente) y se revisa automáticamente
- Filosofía muy relacionada con las metodologías ágiles y la programación extrema (XP)
- Cada integración se verifica mediante una herramienta de construcción para detectar los errores de integración tan pronto como sea posible
- Para ello se pueden utilizar un software especializado en automatizar tareas y que estas se ejecuten de forma automática. Las tareas a realizar pueden ser:
  - Compilación de los componentes
  - Obtener métricas de calidad de código
  - Ejecución de pruebas unitarias
  - Ejecución de pruebas de integración, de aceptación
- Buenos principios
  - Contribuir a menudo
  - No contribuir con código roto
  - Soluciona los builds rotos inmediatamente
  - Escribe tests automáticos
  - Todos los tests deben pasar

Integración Continua (CI). Procedimiento

- El equipo modifica el código, y una vez testeado, se realiza un ***commit*** + ***push*** al repositorio
- La herramienta de CI monitoriza el repositorio y se dispara cada vez que detecta un cambio
- CI ejecuta la construcción del todo el código fuente, aplicando los testeos de control de calidad, pruebas unitarias, pruebas de integración...
- CI envía correos de los resultados de la integración del nuevo código
- Incluir *Badges*

Estado del código

Integración Continua. Herramientas

- Travis CI
  - https://travis-ci.org/
  - Documentación: http://about.travis-ci.org/docs/
  - Travis CI es un sistema de integración continua gratuito en la nube para la comunidad OpenSource. Está integrado con GitHub y ofrece soporte para: Java, Groovy, Haskell, Node.js, PHP, Python, Ruby...
- Jenkins
  - http://jenkins-ci.org/
  - Jenkins es un software de integración continua de código abierto escrito en Java, evolución de Hudson
  - Jenkins tiene soporte para sistemas de control de versiones, algunas como SVN, CVS, Git y corre en un servidor de aplicaciones como por ejemplo Tomcat o Jboss permitiendo la ejecución de proyectos Ant, Maven...

Travis CI. Configuración

- Travis-CI se configura con el fichero: *.travis.yml* en la raíz del proyecto

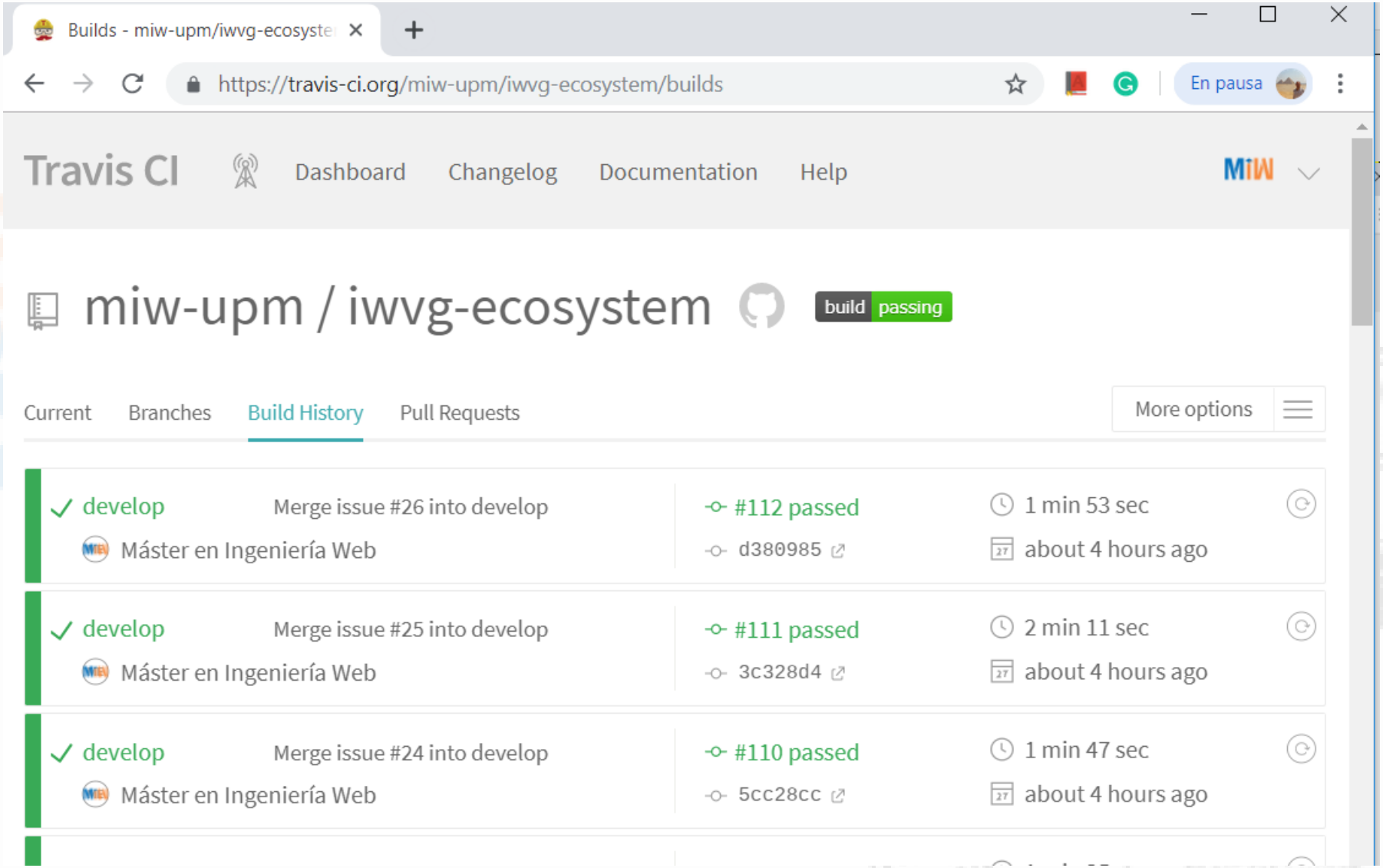
.travis.yml

```
language: java
jdk:
- oraclejdk8
branches:
  only:
  - develop
  - /^release-[0-999].[0-999]$/
  - master
notifications:
  email:
    recipients:
    - j.bernal@upm.es
# services:
# - mysql
# - mongodb
# before_install:
# - mysql -e 'CREATE DATABASE IF NOT EXISTS forge;'
script:
#Test unitario y de integracion con cobertura
- mvn org.jacoco:jacoco-maven-plugin:prepare-agent verify
```

GitHub y Travis CI

- Para empezar a Travis CI, acceder a través de GitHub OAuth. Ir a Travis CI y seguir el vínculo de iniciar sesión, situado en la parte superior: ***Sign in GitHub***
- Activar en GitHub el Servicio Hook#, se realiza desde la Web de Travis-CI. En la parte superior derecha, sobre nuestra cuenta, elegir la opción ***Accounts***, activar el interruptor para cada repositorio que desea conectar en Travis CI
- A partir de ahora, el repositorio estará monitorizado por Travis CI. Cada vez que se realice un commit, ejecuta uno de los siguientes constructores en sus servidores, esta operación puede ir despacio (varios minutos)

Travis CI


Builds - miw-upm/iwvg-ecosyste
https://travis-ci.org/miw-upm/iwvg-ecosystem/builds
En pausa
Travis CI
Dashboard
Changelog
Documentation
Help
MiW
miw-upm / iwvg-ecosystem
build passing
Current
Branches
Build History
Pull Requests
More options
develop
Merge issue #26 into develop
Máster en Ingeniería Web
#112 passed
d380985
1 min 53 sec
about 4 hours ago
develop
Merge issue #25 into develop
Máster en Ingeniería Web
#111 passed
3c328d4
2 min 11 sec
about 4 hours ago
develop
Merge issue #24 into develop
Máster en Ingeniería Web
#110 passed
5cc28cc
1 min 47 sec
about 4 hours ago

Herramientas de Análisis Estático

- Son herramientas que nos ayudan a detectar problemas del código fuente, buscando errores de seguridad, código duplicado, problemas potenciales por mala práctica…
- Se pueden conectar directamente al repositorio y realizar el análisis cada vez que este cambie
- También se pueden integrar en el proceso de la Integración Continua
- Existe una gran variedad en el mercado… pero se han elegido dos:
  - *Sonarcloud*. Se ofrece en la nube *Sonarqube* (*GNU LGPL 3*), y se integra GitHub, Maven…
  - *BetterCodeHub*. Se ofrece en la nube (*Creative Commons Attribution 3.0 License*), y se integra con GitHub…

Sonar

- Sonarqube es una plataforma abierta para gestionar la calidad del código
  - *http://www.sonarqube.org/*
- Se ofrece servicio en la nube
  - *https://sonarcloud.io/*
- Abarca las siguientes características:
  - Estándares de Codificación
  - Bugs y errores potenciales
  - Duplicaciones de código
  - Pruebas Unitarias
  - Cobertura de pruebas
  - Complejidad ciclomática
  - Control de comentarios
- Ofrece soporte para más de 20 lenguajes: Java...

Sonarcloud

- Servicio on-line de *sonarqube*
- Para crear un nuevo proyecto
  - ***>>>Help > Tutorials > Analyze a new Project***
- Para generar una clave de acceso:
  - *https://sonarcloud.io/account/security/*
- Ejecutar el análisis desde Travis-CI
  - Seguridad: la clave de acceso es mejor definirla como variable en *Travis-CI*
    - En *Travis-CI*, en el Proyecto, ir a *Settings* (*More options*), y en el apartado *Environment Variables*, definir una variable con la clave, por ejemplo, ***SONAR***
    - Para referenciarla: ***$SONAR***
  - Definir en el fichero .travis.yml:

.travis.yml

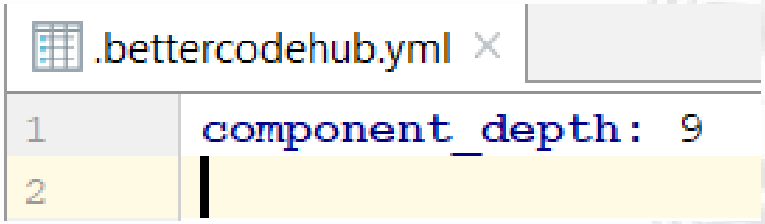
```
22  #SONARCLOUD
23  - mvn sonar:sonar -Dsonar.host.url=https://sonarcloud.io -Dsonar.organization=miw-upm-github -Dsonar.login=$SONAR
```

Nombre de la organización

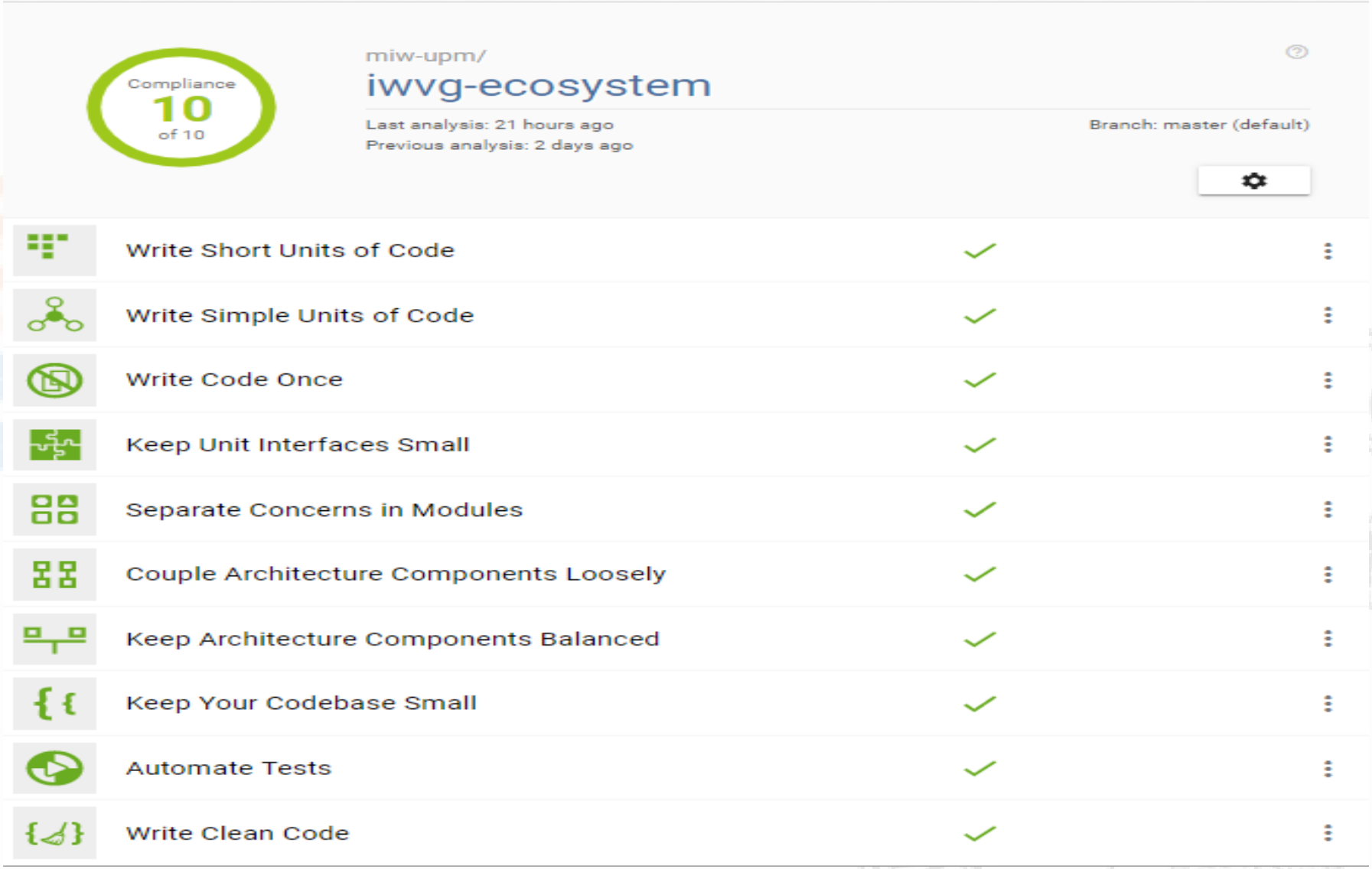
Variable de entorno de Travis-CI

Better Code Hub

- Better Code Hub verifica que su código cumpla con 10 pautas de ingeniería de software de referencia.
- Soporta Java, Kotlin, TypeScript, Python…
- Sigue los principios de:
  - Building Maintainable Software, Java Edition. Ten Guidelines for Future-Proof Code
  - Building Software Teams: Ten Best Practices for Effective Software Development
- Se conecta con GitHub, y tiene una configuración por defecto, pero se puede configurar con un fichero ***.bettercodehub.yml*** en la raíz del proyecto



Better Code Hub



HEROKU

- Heroku (*www.heroku.com*) es una plataforma *PaaS* (*Plataforma como Servicio*) comercializada por *Salesforce*
- Es posible desarrollar prácticamente con cualquier lenguaje de programación: *Java, Ruby, PHP, NodeJS, etc.*
- Las aplicaciones Heroku
  - Están preparadas para poder tener una muy alta escalabilidad. Esto se consigue utilizando los procesos *dyno*, que son instancias independientes que *Heroku* puede levantar y bajar en cualquier momento (en función de la carga de trabajo)
  - Cada instancia contiene una copia del código fuente de la aplicación
  - Es la plataforma Heroku la que se ocupa de colocar un servidor web delante de los *dyno,* de hacer las comunicaciones SSL, etc.
  - Los procesos *dyno* pueden tener una vida muy efímera (por ejemplo, para atender tan solo una petición) con lo cual no es recomendable almacenar información en memoria, ni tampoco escribir ningún fichero en disco
- Cliente ***Heroku CLI*** (*https://devcenter.heroku.com/articles/heroku-cli*)
  - ***>heroku –versión***
  - ***>heroku login***
  - ***>heroku logs --app=<proyecto> -n 100*** *//logs pasados, los 100 últimos*
  - ***>heroku logs --tail –app=<proyecto>*** *//logs en tiempo real*

HEROKU (despliegue)

- En la Web de Heroku: ***https://dashboard.heroku.com***
  1. Crear una aplicación en Heroku (Botón ***>>New***)
     - *name:* ***proyecto***
     - *zone:* ***Europe***
  2. Configurar Heroku
     - ***>>Personal>${proyecto}>Settings***
     - Es automático. Se puede elegir dependencias (***>>Add buildpack***): **`heroku/java`**
  3. Obtener el API-key: ***>>Personal>Account settings***

```
deploy:
  provider: heroku
  api_key:
    secure: $HEROKU
```

- En el proyecto, añadir el comando ***deploy*** en el fichero ***.travis.yml***
- En la web de TRAVIS-CI, crear una variable de entorno (*HEROKU*) con la *ApiKey*
- Crear el fichero *Procfile*, en la raíz del proyecto
  - ***web: java -Dserver.port=$PORT $JAVA_OPTS -jar target/xxx-0.0.0.jar***
- Subir a **Git-hub** la rama ***master***, ello dispara la integración continua con ***Travis-CI***, y si no existen errores, se despliega automáticamente en ***Heroku***, la ruta del despliegue será:
  - *https://***<proyecto>***.herokuapp.com/***

Spring Boot: introducción

- Spring es un framework ligero de código abierto para facilitar el desarrollo de Aplicaciones Empresariales modernas (APIs) mediante Java: https://spring.io/
- Dependencias: POM

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.1.4.RELEASE</version>
    <relativePath/> <!-- lookup parent from repository -->
</parent>
```

```
<!-- Web -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

Spring Boot: introducción

- Configuración

```
# Application info
application.name=@project.artifactId@
build.version=@project.version@
build.timestamp=@maven.build.timestamp@
# Logs  in color
spring.output.ansi.enabled=always
# TRACE DEBUG INFO WARN ERROR FATAL
logging.level.root=WARN
logging.level.es.upm.miw=DEBUG
```

- Arranque

```
@SpringBootApplication(exclude = {ErrorMvcAutoConfiguration.class}) // Not API: /error
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);// mvn clean spring-boot:run
    }
}
```

Spring Boot: introducción

- API

```
@RestController(AdminResource.ADMINS)
public class AdminResource {

    public static final String ADMINS = "/admins";

    @Value("${application.name}")
    private String applicationName;

    @Value("${build.version}")
    private String buildVersion;

    @Value("${build.timestamp}")
    private String buildTimestamp;

    @GetMapping
    public String applicationInfo() { // http://localhost:8080/admins/info
        return "{\"version\":\""
                +this.applicationName + "::" + this.buildVersion + "::" + this.buildTimestamp
                + "\"}";
    }
}
```