# Tema 1. Introducción a las plataformas de desarrollo de software

## Objetivos:

Entender el contexto de las plataformas de apoyo al desarrollo de software frente a metodologías y organización de equipos y personas.

* Comprender el valor y la utilidad que tienen las plataformas de desarrollo de software.
* Identificar los criterios principales que nos ayuden a seleccionar las plataformas en cada caso.
* Conocer algunas plataformas de apoyo al control de versiones.

El desarrollo de software es una tarea compleja que requiere la interacción entre múltiples equipos y personas para la construcción y evolución de soluciones que resuelven problemas con base en una correcta extracción de necesidades y traducción a requisitos técnicos.

## Contexto de las plataformas de desarrollo de software

Para llevar a cabo la construcción del software, se requiere de:

* Un equipo de personas formadas y especializadas en roles y que trabajen en equipo.
* Un método y procesos (metodología) que guíen en la construcción del software. Se parte del trabajo de Watts Humphrey en la década de los 90 (Humphrey, 1989), esta disciplina ha estado liderada por el Software Engineering Institute bajo el término «mejora de procesos de software» (software process improvement, SPI). Creó marcos metodológicos para organizaciones (capability maturity model integration, CMMI), equipos (team software process, TSP) (Humphrey, 2000b) o programadores (personal software process, PSP), entre otros (Humphrey, 2000a).
* Un conjunto de herramientas que facilitan y aceleran el desarrollo. Necesitamos la visión global y eminentemente práctica de herramientas, toolkits y plataformas que ayuden en todo el ciclo de vida del desarrollo de software. Lo central es esto en la asignatura de la construcción del software, pero siempre con la perspectiva de que son herramientas que facilitan y aceleran para dar soporte a una forma de abordar el desarrollo de software (metodología), que viene dirigida y ejecutada por un equipo (personas).

Antes de seleccionar las herramientas que pueden ayudar a desarrollar software, elige el proceso. Primero el proceso, después las herramientas.

Una herramienta de desarrollo de software puede ser cualquier programa de software o utilidad que ayude a los desarrolladores o programadores de software a crear, editar, depurar, mantener y/o realizar alguna tarea específica de programación o desarrollo. Las herramientas se diseñaron inicialmente para extender o complementar los lenguajes de programación al proporcionar la funcionalidad y las características que estos lenguajes no tenían. Por lo general, son utilidades independientes *que brindan o respaldan una tarea particular dentro de cualquier fase del ciclo de desarrollo/programación.* Entre herramientas se conocen las bibliotecas o librerías; caja de herramientas o *toolkit;* Marco de referencia (framework); Kit de desarrollo de software (software development kit, SDK); Intérprete (engine); Plataforma:

En el contexto de la asignatura, hablaremos de plataformas de desarrollo de software para hacer referencia a cualquier solución informática que acelera las labores de un equipo de desarrollo de software en cualquiera de las fases del ciclo de vida.

Clasificación puede ser según perspectiva funcional, de proceso y de integración.

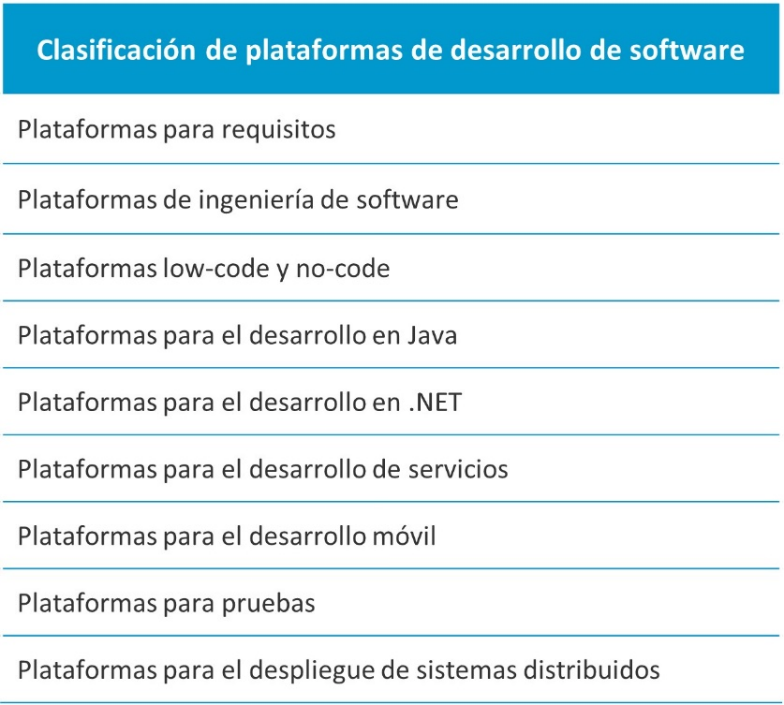
La clasificación funcional de herramientas. Basada en Sommerville, 2015:



El cuerpo de conocimiento de ingeniería de software (software engineering body of knowledge, SWEBOK) del Instituto de Ingenieros Eléctricos y Electrónicos (IEEE) define esta clasificación basada en tipos y subtipos de herramientas (Roongkaew, 2013):



Clasificación de plataformas de software en la asignatura:



## Valor y utilidad de las plataformas de desarrollo de software

Las herramientas de apoyo al desarrollo de software tienen por objetivo mejorar la productividad y eficiencia del equipo de desarrollo, automatizando la ejecución de procesos dentro del ciclo de vida de la construcción del software, simplificando la comunicación y reduciendo errores manuales. Ayudan a:

* Reducir los errores manuales
* Facilitar las pruebas
* Reducir sobrecostes y retrasos
* Facilitar la reutilización
* Reducir costes de migración o conversión
* Reducir el coste operativo
* Mejorar la calidad del software

Aunque las plataformas de apoyo al desarrollo de software pueden suponer una mejora en la eficiencia del desarrollo, entender los requisitos, plantear una solución técnica, diseñar la arquitectura y definir cómo probar son tareas clave que hay que realizar con esfuerzo cognitivo y sin apoyo de herramientas.

No todas las herramientas van a servir siempre, para el análisis, considerar:

**Utilidad** que proporcionará para la finalización general del proyecto

**Contexto tecnológico:** existen soluciones más adecuadas en función de si vamos a crear una solución de escritorio, web, móvil o *serverless* en la nube.

**Normas de la empresa:** en organizaciones (grandes o las más pequeñas), el uso de ciertas herramientas será obligatorio según la política establecida. La estandarización de las herramientas puede ayudar a una organización a mover a los desarrolladores fácilmente entre proyectos según sea necesario y brinda a la gerencia la seguridad de que se siguen procesos similares entre diferentes proyectos y equipos de proyectos, lo que da como resultado una calidad de producto homogénea.

**Experiencia previa del equipo** **con la herramienta:** todo el software tiene una curva de aprendizaje. La selección de herramientas específicas puede verse influenciada por el nivel de experiencia que los desarrolladores ya tengan con ellas. Esa experiencia específica también se puede utilizar como recurso para decidir si una herramienta puede ser útil en el proyecto o no, dadas las opiniones de los desarrolladores.

**Capacidad de integración** Un ejemplo de esta integración más profunda es cuando el control de código fuente se integra con el seguimiento de errores, que a su vez se integra con el seguimiento de incidentes (o problemas del cliente). Un cambio de código registrado en el control de fuente puede indicar al rastreador de errores que hay una solución disponible para un error informado. Ese error puede estar asociado con un informe de incidente, que luego se puede actualizar con el estado del trabajo que se ha realizado que podría, eventualmente, resolver ese incidente.

**Coste de uso:** existen herramientas comerciales y una gran base de herramientas *open source* y de uso gratuito, que debemos considerar en función del contexto, de la utilidad y del presupuesto disponible.

**Coste de aprendizaje:** todo software tiene una curva de aprendizaje, en un grado u otro. Las herramientas complejas también pueden requerir tiempo y esfuerzo para implementarlas con el equipo e integrarlas en el software de desarrollo existente. Además de la implementación inicial y la curva de aprendizaje, muchas herramientas requieren cierto tiempo y esfuerzo para usarlas. Esta sobrecarga debe tenerse en cuenta al evaluar el valor general de la herramienta.

## Plataformas para el control de versiones

El control de versiones es una práctica, integrada en la gestión de la configuración del software, responsable de gestionar cambios en el código de programas informáticos, su documentación o cualquier colección de información relevante para su desarrollo o evolución. Puede identificarse con **letra** y **número** (ver 1, ver2, …, por ejemplo); cada revisión está asociada con una marca de tiempo y la persona que realiza el cambio. Las revisiones se pueden comparar, restaurar y, con algunos tipos de archivos, fusionar.

Desde que existe la escritura, se hace revisión de versiones, pero la era informática lo llevó a otro nivel; en *el desarrollo de software es donde están los más complejos*. Un equipo de personas puede realizar cambios en los mismos archivos al mismo tiempo. Los sistemas de control de versiones suelen ejecutarse como aplicaciones independientes, pero el control de revisiones también está integrado en varios tipos de software, como procesadores de texto y hojas de cálculo, documentos web colaborativos y sistemas de gestión de contenido.

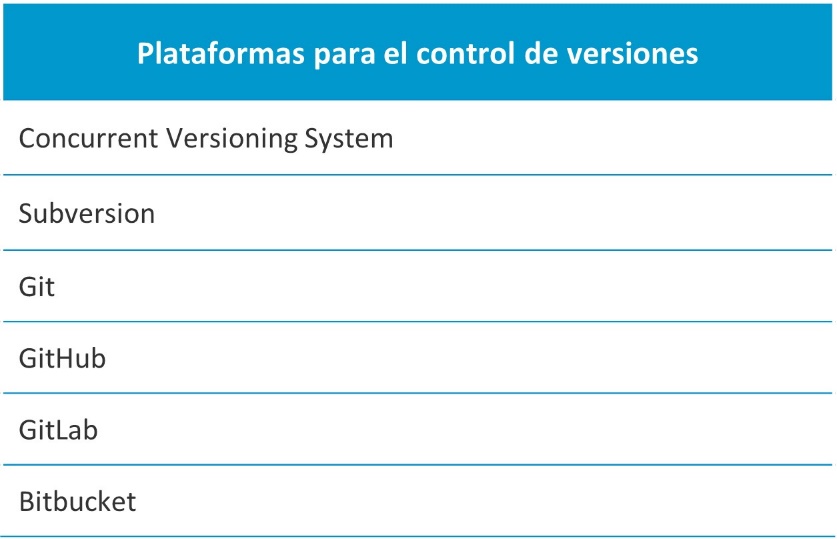
Propósitos y valores del control de versiones:

* **Varias personas pueden trabajar simultáneamente** en un solo proyecto. Cada uno trabaja y edita su propia copia de los archivos y depende de ellos compartir los cambios realizados por ellos con el resto del equipo.
* También permite que una persona use **varias** **computadoras** para trabajar en un proyecto, por lo que es valioso incluso si está trabajando sola.
* **Integra el trabajo** que se realiza simultáneamente por diferentes miembros del equipo. En algunos casos raros, cuando dos personas realizan ediciones conflictivas en la misma línea de un archivo, el sistema de control de versiones solicita la asistencia humana para decidir qué se debe hacer.
* Proporciona acceso a las **versiones históricas** de un proyecto. Este es un seguro contra fallas en la computadora o pérdida de datos. Si se comete algún error, puede retroceder fácilmente a una versión anterior.
* También es posible **deshacer ediciones** específicas sin perder el trabajo realizado mientras tanto. Se puede saber fácilmente cuándo, por qué y quién editó cualquier parte de un archivo.
* Adicionalmente, simplifica la **trazabilidad** del código fuente respecto a otra documentación del proyecto, como requisitos, incidencias, diseños, etc.

Estas son las consideraciones básicas para tener en cuenta antes de usar o comprar un sistema de control de versiones:

* **Tipología:** existen repositorios centralizados o distribuidos.
* **Comercial contra abierto:** evalúe su proyecto, ya sea que utilice un control de versiones comercial o de código abierto.
* **Tamaño del equipo:** compruebe el número máximo de usuarios por cuenta o repositorio.
* **Tamaño del almacenamiento:** determine el espacio de almacenamiento disponible.
* **Múltiples proyectos:** evalúe la cantidad de proyectos permitidos por cuenta.
* **Funciones:** verifique la disponibilidad de herramientas adicionales, como rastreadores de errores, seguimiento de tiempo, documentación, etc.

En el ámbito del control de versiones, la *bifurcación* (*branching*) es la duplicación de un objeto bajo el control de versiones —como un archivo de código fuente o un árbol de directorios—. A partir de entonces, cada objeto se puede modificar por separado y en paralelo para que los objetos se vuelvan diferentes. En este contexto, los objetos se denominan **ramas**. Los usuarios del sistema de control de versiones pueden ramificar cualquier rama. Las ramas también se conocen como árboles, arroyos o líneas de código y reciben nombre según si vienen de la principal, si tienen padre o no.



Los enunciados previamente a Git

A screenshot of a computer screen

Description automatically generated

**Git**, que toma una *instantánea* del cambio

A chart with text and words

Description automatically generated with medium confidence

Git tiene tres estados principales en los que pueden residir sus archivos: modificado, preparado y confirmado:

* **Modificado (modified):** significa que ha cambiado el archivo pero aún no lo ha confirmado en su base de datos.
* **Preparado (staged):** significa que ha marcado un archivo modificado en su versión actual para pasar a su próxima instantánea de confirmación.
* **Comprometido (committed):** significa que los datos se almacenan de forma segura en su base de datos local.

El **directorio** de Git es donde almacena los metadatos y la base de datos de objetos para su proyecto. Esta es la parte más importante de Git y es lo que se copia cuando se clona un repositorio desde otra computadora. El flujo de trabajo básico de Git es algo así:

* Modificas archivos en tu árbol de trabajo.
* Organizas selectivamente solo aquellos cambios que desea que sean parte de su próxima confirmación, que agrega solo esos cambios al área de preparación.
* Realizas una confirmación, que toma los archivos tal como están en el área de preparación y almacena esa instantánea de forma permanente en su directorio de Git.

**GitHub**

Es una plataforma como servicio en la nube para el control de versiones de nuestros proyectos basada en Git. Ha sido una de las referencias para proyectos open source, con millones de proyectos accesibles a través de sus repositorios. Basado en Git, adquirida por Microsoft en 2018, con características:

* **Codificación colaborativa:** contribuya a los proyectos rápidamente con la configuración automática del entorno. Asegúrese de ver los cambios que le interesan y cree una comunidad en torno a su código.
* **Automatización y CI/CD:** automatice la integración y distribución continuas (CI/CD), pruebas, planificación, gestión de proyectos, etiquetado de problemas, aprobaciones, incorporación y más.
* **Seguridad:** asegure el código mientras lo escribe. Revise automáticamente cada cambio en su base de código e identifique vulnerabilidades antes de que lleguen a producción.
* **Múltiples aplicaciones cliente:** integre GitHub en cualquier dispositivo conectado. Omita la interfaz de usuario visual con una interfaz de texto rápida y potente. Acceda a GitHub desde su escritorio macOS o Windows.
* **Administración del proyecto:** gestione las solicitudes de funciones y los errores. Coordine iniciativas grandes y pequeñas con tablas de proyectos, tableros y listas de tareas. Realice un seguimiento de lo que entrega hasta el despliegue.
* **Administración del equipo:** simplifique la gestión de accesos y permisos en sus proyectos y equipos. Actualice los permisos, agregue nuevos usuarios a medida que crece y otorgue a todos los permisos exactos que necesitan. Sincroniza con Okta y Azure Active Directory.

**GitLab**

Es un servicio de alojamiento de repositorio Git basado en la web. Ofrece toda la funcionalidad de control de revisión distribuida y gestión de código fuente (SCM) de Git, además de agregar sus propias características. GitLab Enterprise Edition se basa en Git e incluye funciones adicionales. Inicialmente, era una solución de administración de código fuente para colaborar dentro de un equipo en el desarrollo de software que evolucionó a una solución integrada que cubría el ciclo de vida del desarrollo de software y luego a todo el ciclo de vida de desarrollo y operaciones (DevOps)

Sigue un modelo de desarrollo de núcleo abierto en el que la funcionalidad principal se publica bajo una licencia de código abierto (MIT), mientras que las funcionalidades adicionales están bajo una licencia propietaria.

Algunas de sus características son:

* **Planificación**: permite la planificación y gestión de carteras a través de épicas, grupos (programas) e hitos para organizar y realizar un seguimiento del progreso. Independientemente de su metodología, desde Waterfall hasta DevOps, el enfoque simple y flexible de planificación de GitLab satisface las necesidades de pequeños equipos y grandes empresas.
* **Creación**: cree, vea y administre código y datos de proyectos a través de poderosas herramientas de bifurcación.
* **Verificación**: mantenga estrictos estándares de calidad para el código de producción con pruebas e informes automáticos.
* **Empaquetar**: cree una cadena de suministro de software coherente y fiable con la gestión de paquetes integrada.
* **Seguridad**: capacidades de seguridad, integradas en su ciclo de vida de desarrollo. GitLab proporciona pruebas de seguridad de aplicaciones estáticas (SAST), pruebas de seguridad de aplicaciones dinámicas (DAST), escaneo de contenedores y escaneo de dependencias para ayudarlo a entregar aplicaciones seguras junto con el cumplimiento de licencias.
* **Generación** **de** **versiones**: la solución de CD integrada de GitLab le permite enviar código sin intervención, ya sea en uno o mil servidores.
* **Monitorización**: ayude a reducir la gravedad y la frecuencia de los incidentes. Obtenga comentarios y las herramientas que lo ayudarán a reducir la gravedad y la frecuencia de los incidentes para que pueda lanzar software con frecuencia y confianza.
* **Gestión del desarrollo**: ayuda a los equipos a administrar y optimizar su ciclo de vida de entrega de software con métricas y conocimientos de flujo de valor para optimizar y aumentar su velocidad de entrega.

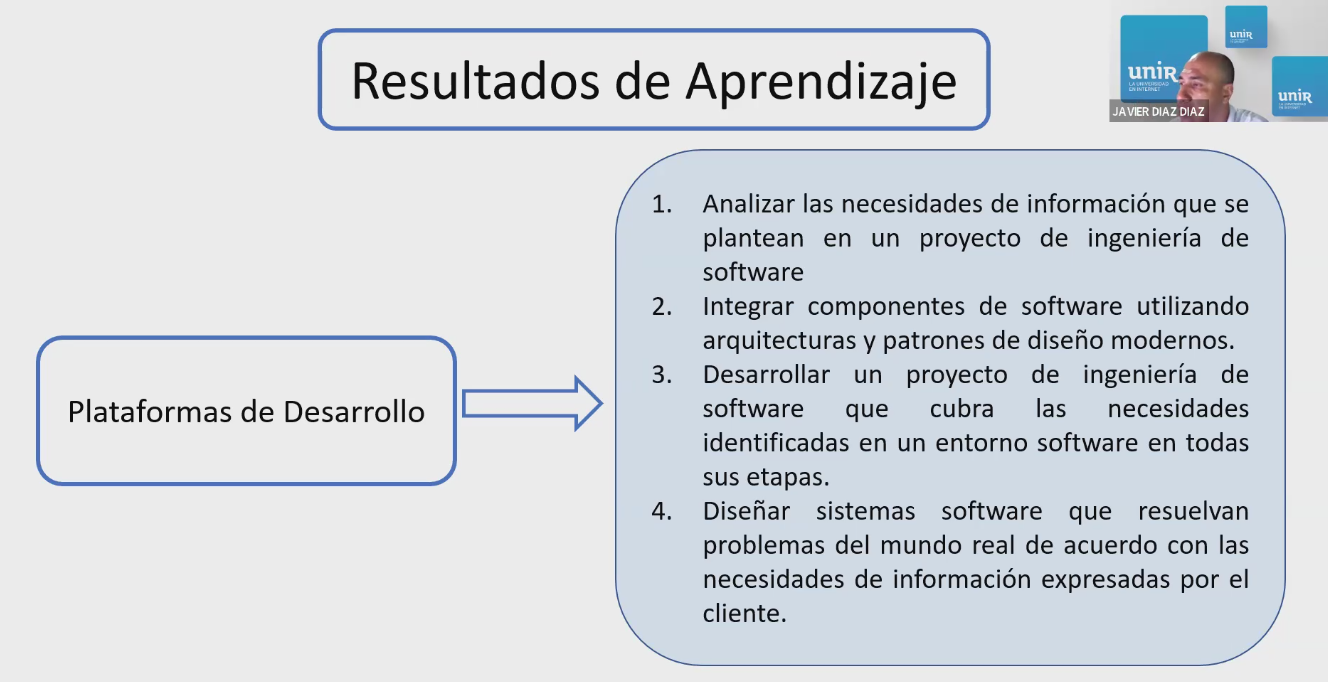
**Bitbucket**

Es la solución de Atlassian de gestión de repositorios Git diseñada para equipos profesionales. Brinda un lugar central para administrar los repositorios de Git, colaborar en su código fuente y guiarlo a través del flujo de desarrollo (Bitbucket,s. f.). Proporciona características como:

* Control de acceso para restringir el acceso a su código fuente.
* Control de flujo de trabajo para hacer cumplir un flujo de trabajo de proyecto o equipo.
* Solicitudes de extracción con comentarios en línea para colaborar en la revisión del código.
* Integración de Jira para una trazabilidad completa del desarrollo.
* API de descanso completo para crear funciones personalizadas para su flujo de trabajo si aún no están disponibles en nuestro marketplace.

## Clase presencial

Presentación de Javier Díaz Díaz y lo siguiente:





A group of text boxes

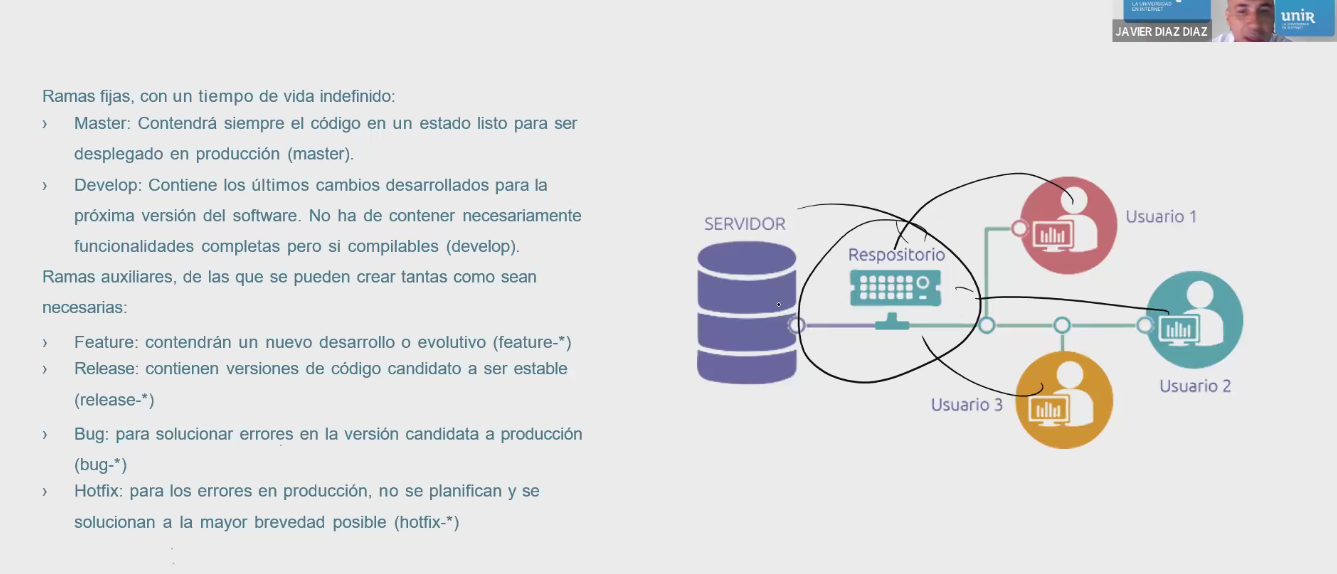
Description automatically generated with medium confidence

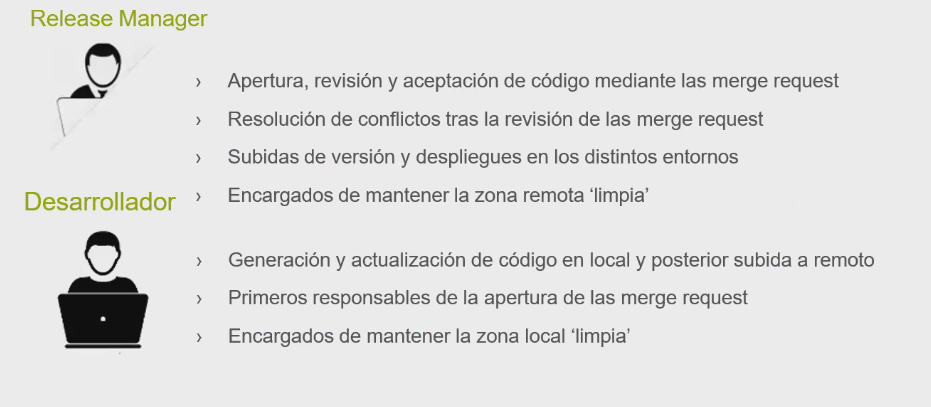


Para desarrollo se puede utilizar, por ejemplo, **Visual Studio Code**. Pero existen otras que pueden consumir más recurso como **NetBeans** (<https://netbeans.apache.org/front/main/index.html>) y **Eclipse** (<https://www.eclipse.org/downloads/>). Para contenedores, como **Docker,** existe **Kubernetes.** Para gestión de proyectos existe **Jira,** que maneja Scrum (by Atlassian)

**DevOps**

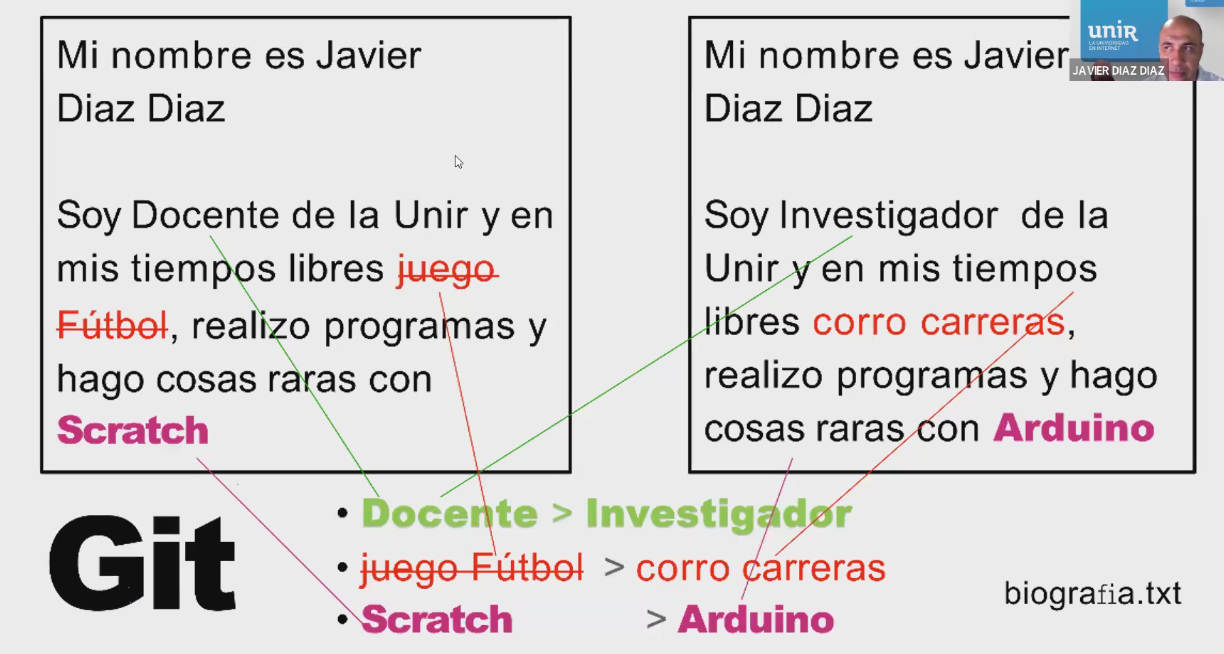
****

****

****

**A close-up of a logo

Description automatically generated**

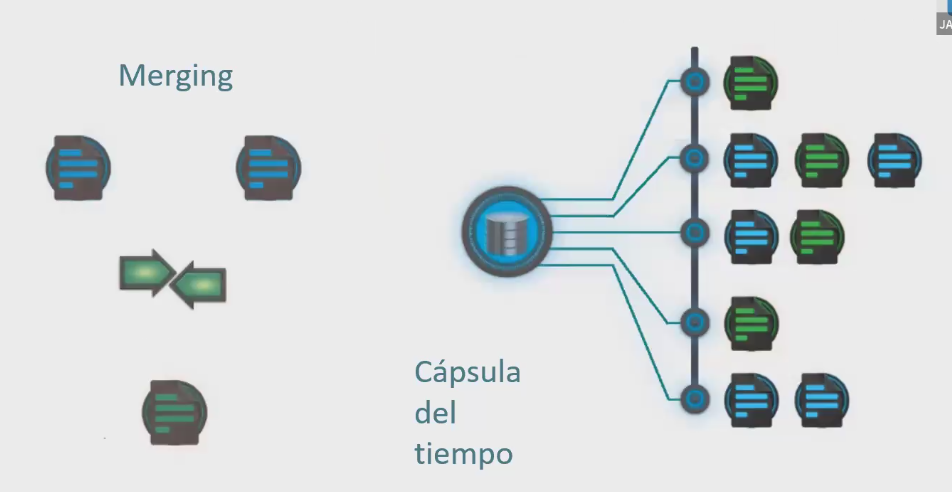
****

**A screenshot of a computer program

Description automatically generated**

****

****

****

**A close-up of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**