Plataformas de Desarrollo de Software

Tema 1. Introducción a las plataformas de desarrollo de software

Índice

Esquema

INTRODUCCIÓN A LAS PLATAFORMAS DE DESARROLLO DE SOFTWARE

CONCEPTOS

Contexto desarrollo software

- Personas
- Procesos/métodos
- Herramientas

Concepto de plataforma en la asignatura

- Biblioteca o librería
- Caja de herramientas (Toolkit)
- Marco de referencia (Framework)
- Software Development Kit (SDK)
- Intérprete (Engine)

VALOR Y UTILIDAD

Valor en mejorar la productividad y eficiencia del equipo de desarrollo:

- 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

Criterios para seleccionar plataformas:

- Utilidad
- Contexto tecnológico
- Normas de la empresa
- Experiencia previa
- Capacidad de integración
- Coste de uso
- Coste de aprendizaje

HERRAMIENTAS

Plataformas control de versiones

- Concurrent Versioning System
- Subversion
- Git
- GitHub
- GitLab
- Bitbucket

1.1. Introducción y objetivos

En este tema se introduce el concepto de plataforma de apoyo al desarrollo de software junto con el equipo y el proceso o metodología. En primer lugar, veremos la **terminología** y aclararemos el concepto de plataforma de desarrollo de software que usaremos en la asignatura frente a términos como librería, caja de herramienta, marco de referencia, kit de desarrollo o intérprete, entre otros.

Seguidamente, veremos que existen múltiples formas de **organizar y clasificar** las herramientas, ya sea por utilidad, base tecnológica, parte del proceso que apoya o capacidades de integración. Definiremos la clasificación que vamos a usar durante la asignatura y que motiva la descomposición en temas.

Posteriormente, veremos las principales **ventajas** del uso de plataformas de apoyo al desarrollo de software junto con algunos criterios para seleccionarlas. Finalmente, conoceremos las **principales herramientas** de apoyo al control de versiones.

Con el estudio de este tema pretendemos alcanzar los siguientes 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.**

1.2. Contexto de las plataformas de desarrollo de software

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. 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** que guíen en la construcción del software. Si partimos del trabajo realizado por 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. Esta asignatura tiene como objetivo dotar al estudiante de una visión global y eminentemente práctica de herramientas, *toolkits* y plataformas que ayuden en todo el ciclo de vida del desarrollo de software. Por ello nos centraremos en este último punto 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.

Ideas clave

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.

En la actualidad existen múltiples conceptos y términos asociados a herramientas que pueden ayudar al desarrollo de software, como:

- **Biblioteca o librería**: es una colección de código relacionado con una tarea específica o un conjunto de tareas estrechamente relacionadas que operan aproximadamente al mismo nivel de abstracción. Por lo general, carece de cualquier propósito o intención propia y está destinada a ser utilizada e integrada con el código del cliente para ayudar a ejecutar sus tareas.

- **Caja de herramientas (*toolkit*)**: históricamente, un kit de herramientas es una biblioteca más enfocada, con un propósito definido y específico. Actualmente, este término ha caído en desuso y se usa casi exclusivamente para *widgets* gráficos y componentes de interfaz gráfica de usuario (GUI). La mayoría de las veces, un conjunto de herramientas operará en una capa más alta de abstracción que una biblioteca y, a menudo, consumirá y usará las bibliotecas por sí mismo. A diferencia de las bibliotecas, el código del kit de herramientas se suele usar para ejecutar la tarea del código del cliente, como construir una ventana, cambiar su tamaño, etc.

- **Marco de referencia (*framework*):** históricamente, un marco era un conjunto de bibliotecas y módulos interrelacionados que se separaban en categorías generales o específicas. Los marcos generales estaban destinados a ofrecer una plataforma completa e integrada para crear aplicaciones al ofrecer funcionalidad general, como gestión de memoria multiplataforma, abstracciones de subprocesos múltiples,

estructuras dinámicas y estructuras genéricas en general. Los marcos específicos se desarrollaron para tareas únicas como los sistemas de comando y control para sistemas industriales y las primeras pilas de redes y operaron a un alto nivel de abstracción. Se usaron juegos de herramientas similares para llevar a cabo la ejecución de las tareas de códigos de cliente. Actualmente, la definición de un marco se ha enfocado más y adopta el principio de **inversión de control,** por lo que el marco lleva a cabo el flujo del programa, así como la ejecución.

- **Kit de desarrollo de software (*software development kit,* SDK):** es una colección de herramientas para ayudar al programador a crear e implementar código y/o contenido que está específicamente diseñado para ejecutarse en una plataforma muy particular o de una manera muy particular. Un SDK puede consistir simplemente en un conjunto de bibliotecas que deben ser utilizadas de una manera específica solo por el código del cliente y que pueden compilarse normalmente, hasta un conjunto de herramientas binarias que crean o adaptan activos binarios para producir su aplicación en producción.

- **Intérprete (*engine*):** un motor (en términos de interpretación de código) es un binario que ejecutará contenido personalizado o procesará datos de entrada de alguna manera. Los motores de juegos y gráficos son quizás los usuarios más frecuentes de este término y se utilizan casi universalmente con un SDK para apuntar al propio intérprete, como el *unreal development kit* (UDK), pero también existen otros motores, como los motores de búsqueda y los motores de gestión de bases de datos.

- **Plataforma:** el término está más orientado a las infraestructuras físicas (máquinas o *cloud*) o lógicas (sistemas operativos o entornos virtuales) sobre las que se ejecutan las aplicaciones.

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.

Las **clasificaciones** de las herramientas de software nos ayudan a comprender los tipos de herramientas de software y su papel en el soporte de las actividades del proceso de software. Hay varias formas de clasificar las herramientas de software, cada una de las cuales nos brinda una perspectiva diferente sobre estas.

- Una perspectiva **funcional** donde las herramientas se clasifican según su función específica.
- Una perspectiva de **proceso** donde las herramientas se clasifican de acuerdo con las actividades de proceso que soportan.
- Una perspectiva de **integración** en la que las herramientas se clasifican según cómo se organizan en unidades integradas que brindan soporte para una o más actividades de proceso.

Por ejemplo, Sommerville (2015) clasifica las soluciones por su perspectiva funcional de esta forma:

Clasificación de herramientas de desarrollo de software	
Herramientas de planificación	Herramientas de técnicas de revisión y evaluación de programas (PERT), herramientas de estimación, hojas de cálculo
Herramientas de edición	Editores de texto, editores de diagramas, procesadores de texto
Herramientas de gestión de cambios	Herramientas de trazabilidad de requisitos, sistemas de control de cambios
Herramientas de gestión de configuración	Sistemas de gestión de versiones, herramientas de creación de sistemas
Herramientas de creación de prototipos	Lenguajes de muy alto nivel, generadores de interfaz de usuario
Herramientas de soporte de métodos	Editores de diseño, diccionarios de datos, generadores de código
Herramientas de procesamiento del lenguaje	Compiladores, intérpretes
Herramientas de análisis de programas	Generadores de referencias cruzadas, analizadores estáticos, analizadores dinámicos
Herramientas de prueba	Generadores de datos de prueba, comparadores de archivos
Herramientas de depuración	Sistemas de depuración interactivos
Herramientas de documentación	Programas de maquetación de páginas, editores de imágenes
Herramientas de reingeniería	Sistemas de referencia cruzada, sistemas de reestructuración de programas

Tabla 1. Clasificación funcional de herramientas. Fuente: basado 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 herramientas de desarrollo de software según SWEBOK	
Herramientas de requisitos software	▸ Modelado de requisitos ▸ Trazabilidad de requisitos
Herramientas de diseño	▸ Diseño de arquitectura o componentes
Herramientas de construcción de software	▸ Editores de programas ▸ Compiladores y generadores de código ▸ Intérpretes ▸ Depuradores
Herramientas de prueba de software	▸ Pruebas de software ▸ Generadores de pruebas ▸ Marcos de ejecución de pruebas ▸ Evaluación de las pruebas ▸ Gestión de pruebas ▸ Análisis de rendimiento
Herramientas de mantenimiento de software	▸ Comprensión ▸ Reingeniería
Herramientas de gestión de configuración de software	▸ Defectos, mejoras, problemas, seguimiento de problemas ▸ Gestión de versiones ▸ Liberar y compilar
Herramientas de gestión de ingeniería de software	▸ Planificación y seguimiento de proyectos ▸ Gestión de riesgos ▸ Medición
Herramientas de proceso de ingeniería de software	▸ Modelado de procesos ▸ Gestión de procesos ▸ Entornos de ingeniería de software asistida por computadora (CASE) integrados ▸ Software centrado en procesos
Herramientas de calidad de software	▸ Revisión y auditoría ▸ Análisis estático
Otras herramientas	

Tabla 2. Clasificación funcional de herramientas. Fuente: basado en Roongkaew, 2013.

Para estructurar las diversas plataformas de apoyo al desarrollo de software, hemos realizado una clasificación propia que combina, en ocasiones, una vista funcional, en otras una perspectiva tecnológica o de arquitectura del sistema y, por último, una vista metodológica o de proceso.

De esta forma, la clasificación de plataformas de desarrollo de software que se ha seguido para la asignatura se estructura en base a las siguientes categorías, estructuradas en diversos temas:

Clasificación de plataformas de desarrollo de software
Plataformas para requisitos
Plataformas de ingeniería de software
Plataformas low-code y no-code
Plataformas para el desarrollo en Java
Plataformas para el desarrollo en .NET
Plataformas para el desarrollo de servicios
Plataformas para el desarrollo móvil
Plataformas para pruebas
Plataformas para el despliegue de sistemas distribuidos

Tabla 3. Clasificación de plataformas de software en la asignatura. Fuente: elaboración propia.

En el contexto de la asignatura, las plataformas de desarrollo de software que analicemos serán lo más independientes posibles al proceso o metodología seguido para la construcción. Perseguiremos el mismo objetivo a nivel tecnológico y de arquitectura de la solución, aunque aquellas específicas de Java, .NET, móvil o arquitecturas de sistemas distribuidos basadas en servicio tendrán un contexto tecnológico concreto y acotado.

1.3. 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. En general, alguno de los principales beneficios del uso de plataformas de apoyo al desarrollo de software son:

- **Reducir los errores manuales:** la programación estructurada y apoyada en herramientas produce programas que son más fáciles de probar y, una vez probados, más sencillos de mantener.
- **Facilitar las pruebas:** con herramientas apropiadas, se simplifica el trabajo de validar y verificar un desarrollo antes de su salida a producción.
- **Reducir sobrecostes y retrasos:** las técnicas de diseño y desarrollo actuales pueden hacer que el proceso de desarrollo sea más fácil de controlar, reduciendo riesgos tecnológicos y facilitando la toma de decisiones durante el seguimiento del proyecto.
- **Facilitar la reutilización:** mediante las técnicas y soluciones existentes, se maximiza y facilita que las organizaciones reutilicen el software existente.
- **Reducir costes de migración o conversión:** una necesidad frecuente es migrar tecnológicamente sistemas legados. Con las herramientas apropiadas, se puede reducir significativamente el esfuerzo y coste asociado para evolucionar un software existente a otro entorno o lenguaje.

- **Reducir el coste operativo:** con plataformas de software, podemos optimizar nuestro sistema para reducir la utilización de los recursos de la máquina necesarios.
- **Mejorar la calidad del software:** las herramientas nos pueden ayudan a escribir mejor código, detectar antes los errores y facilitar su futuro mantenimiento.

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.

A la hora de seleccionar las posibles plataformas de apoyo al desarrollo de software en una organización, existe una **serie de factores** a considerar. No todas las herramientas son necesariamente apropiadas para un proyecto determinado, mientras que otras pueden ser críticas. Cada equipo debe considerar una serie de factores al momento de decidir sobre las herramientas que se pueden utilizar en el desarrollo de software de cada proyecto. Alguno de los factores a considerar son:

- **Utilidad:** es el factor principal al decidir si usar un tipo de herramienta y qué implementación de esa herramienta es la utilidad que proporcionará para la finalización general del proyecto.
- Aplicabilidad a nuestro **contexto tecnológico:** no todas las herramientas se aplican a todos los entornos o contextos tecnológicos. Por ejemplo, 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 más grandes y a menudo también en las más pequeñas, el uso de ciertas herramientas será obligatorio para lograr objetivos o cumplir con 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: hasta cierto punto, casi 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, ya que los desarrolladores suelen tener opiniones sólidas sobre estos asuntos y no suelen tener vergüenza de expresarlas.

- **Capacidad de integración:** la integración de una herramienta con otras herramientas puede tener un gran impacto en el valor que agrega al equipo y al proyecto. Algunas integraciones toman la forma de integración de «conveniencia» —es decir, la utilidad de control de código fuente se integra con el IDE de modo que un desarrollador hace que un archivo se desproteja automáticamente una vez que comienza a editarlo—. Otra integración más profunda fusiona información y responde a eventos entre herramientas para brindar un alto valor al equipo y a otros grupos dentro de la organización. 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.

1.4. Plataformas para el control de versiones

Vamos a presentar algunas plataformas de apoyo al control de versiones, por ser una **práctica común y general** de aplicación a todo tipo de desarrollo de software.

En ingeniería de software, el **control de versiones** (también conocido como control de revisión, control de código fuente o gestión de código fuente) 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.

Los cambios generalmente se identifican mediante un **código de número o letra** denominado número de revisión, nivel de revisión o simplemente revisión. Por ejemplo, un conjunto inicial de archivos es «revisión 1». Cuando se realiza el primer cambio, el conjunto resultante es «revisión 2» y así sucesivamente. 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.

La necesidad de una **forma lógica de organizar y controlar** las revisiones ha existido casi desde que existe la escritura, pero el control de revisiones se volvió mucho más importante y complicado cuando comenzó la era de la informática. La numeración de ediciones de libros y de revisiones de especificaciones son ejemplos que se remontan únicamente a la era de la impresión. Hoy en día, los sistemas de control de revisión más capaces (así como complejos) son los que se utilizan en el desarrollo de software, en los que 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.

Ideas clave

Los principales propósitos y valores del control de versiones son los siguientes:

- 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. La rama de origen a veces se la llama rama principal, rama ascendente (o simplemente ascendente, especialmente si las sucursales son mantenidas por diferentes organizaciones o individuos) o corriente de respaldo. Las ramas secundarias son ramas que tienen un padre: una rama sin un padre se conoce como troncal o línea principal.

Las principales plataformas para el control de versiones son:

Plataformas para el control de versiones
Concurrent Versioning System
Subversion
Git
GitHub
GitLab
Bitbucket

Tabla 4. Plataformas para el control de versiones. Fuente: elaboración propia.

Concurrent Versions System (CVS)

Es un sistema de control de versiones, un componente importante de la gestión de configuración de software (*source configuration management*, SCM). Utilizándolo, puede **registrar el historial** de fuentes, archivos y documentos. Cumple una función similar a los paquetes de software libre RCS, PRCS y Aegis. Fue uno de los primeros sistemas de control de versiones más ampliamente utilizado, especialmente en muchos proyectos de software libre (CVS, 2022).

Si bien CVS almacena el historial de archivos individuales en el mismo formato que RCS, ofrece las siguientes **ventajas significativas** por sobre este último:

- Puede ejecutar **secuencias de comandos** para registrar operaciones de CVS o aplicar políticas específicas del sitio.

- El cliente/servidor CVS permite que los desarrolladores dispersos por geografía o módems lentos funcionen como un solo equipo. El historial de versiones se almacena en **un único servidor central** y las máquinas cliente tienen una copia de todos los archivos en los que están trabajando los desarrolladores. Por lo tanto, la red entre el cliente y el servidor debe estar activa para realizar operaciones CVS (como registros o actualizaciones), pero no para editar o manipular las versiones actuales de los archivos. Los clientes pueden realizar todas las mismas operaciones que están disponibles localmente.

- En los casos en que varios desarrolladores o equipos quieran mantener cada uno su propia versión de los archivos, debido a la geografía y/o la política, las sucursales de proveedores de CVS pueden **importar** una versión de otro equipo (incluso si no usan CVS) y luego CVS pueden fusionar los cambios de la rama del proveedor con los archivos más recientes si eso es lo que se desea.

- Permite **cambios concurrentes,** por lo que más de un desarrollador puede trabajar en los mismos archivos al mismo tiempo.

- CVS provee una base de datos de **módulos flexibles** que proporciona un mapeo simbólico de nombres a componentes de una distribución de software más grande. Aplica nombres a colecciones de directorios y archivos. Un solo comando puede manipular toda la colección.

- Los servidores CVS se ejecutan en la **mayoría** de las variantes de Unix y también hay disponibles clientes para Windows NT/95, OS/2 y VMS. CVS también funcionará en lo que a veces se denomina modo de servidor en repositorios locales en Windows 95/NT.

Apache Subversion

También conocido como **Subversion** o **SVN**, es un sistema de control de versiones de código abierto bajo la licencia de Apache, que nació como evolución de CVS para mejorar algunas de sus funcionalidades (Blokdyk, 2018).

CVS solo rastrea la modificación archivo por archivo, mientras que SVN rastrea una **confirmación completa** como una nueva revisión, lo que significa que es más fácil seguir el historial de su proyecto. A su vez, todo el software de control de fuente moderno utiliza el concepto de revisión, por lo que es mucho más fácil migrar desde SVN que desde CVS.

También está el problema del **compromiso atómico.** Es posible que dos personas que trabajen juntas en CVS puedan entrar en conflicto entre sí y se pierdan algunos datos, lo que pondría a su cliente en un estado inconsistente. Cuando se detectan a tiempo, estos problemas no son importantes porque sus datos todavía están en algún lugar, pero pueden ser una molestia en un entorno estresante.

Entre las **características clave** de SVN, se incluyen: gestión de inventario, gestión de seguridad, seguimiento de historial, controles de acceso de usuarios, recuperación de datos y gestión de flujo de trabajo. SVN es fácil de implementar con cualquier lenguaje de programación. Además, ofrece almacenamiento consistente para manejar archivos binarios y de texto.

Git

Es la base de los controladores de versiones actuales. La principal diferencia entre Git y cualquier otro anterior es la forma en la que Git estructura sus datos. Conceptualmente, la mayoría de los otros sistemas almacenan información como una lista de cambios basados en archivos (Chacon y Straub, 2014). Estos otros sistemas (CVS, Subversion, Perforce, Bazaar, etc.) estructuran la información que almacenan como un conjunto de archivos y los cambios realizados en cada archivo a lo largo del tiempo (esto se describe comúnmente como control de versión basado en delta).

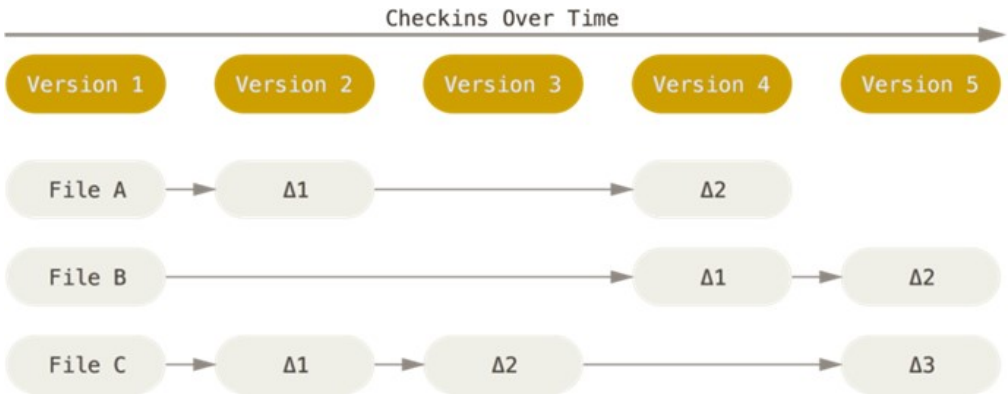

Figura 1. Modelo de gestión de cambios por deltas. Fuente: Chacon y Straub, 2014.

Git no estructura ni almacena sus datos de esta manera, sino que lo hace como una **serie de instantáneas** de un sistema de archivos en miniatura. Con Git, cada vez que confirma o guarda el estado de su proyecto, básicamente se toma una imagen de cómo se ven todos sus archivos en ese momento y almacena una referencia a esa instantánea. Para ser eficiente, si los archivos no han cambiado, Git no vuelve a almacenar el archivo, solo un enlace al archivo idéntico anterior que ya ha almacenado.

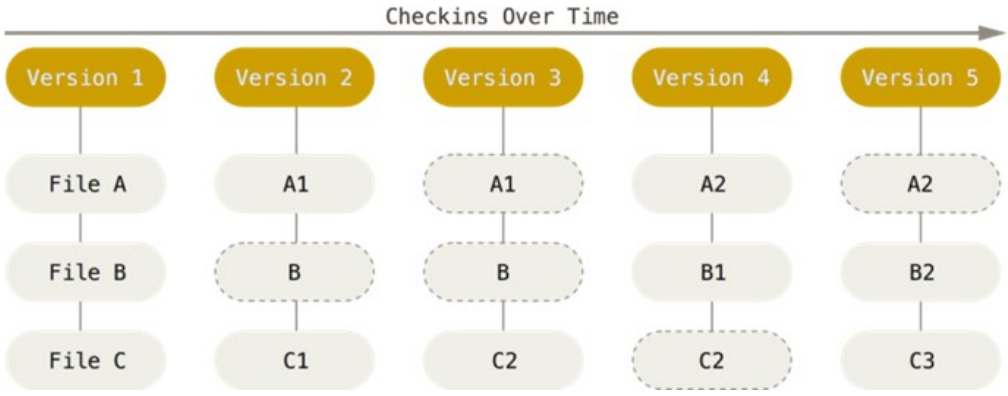

Figura 2. Modelo de gestión de cambios por instantáneas. Fuente: Chacon y Straub, 2014.

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. Adquirida por Microsoft en 2018, en la actualidad tiene versiones gratuitas para equipos limitados y soluciones *open source* y versiones de pago para empresas y equipos grandes (GitHub, s. f.).

Algunas de sus características son:

- **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.

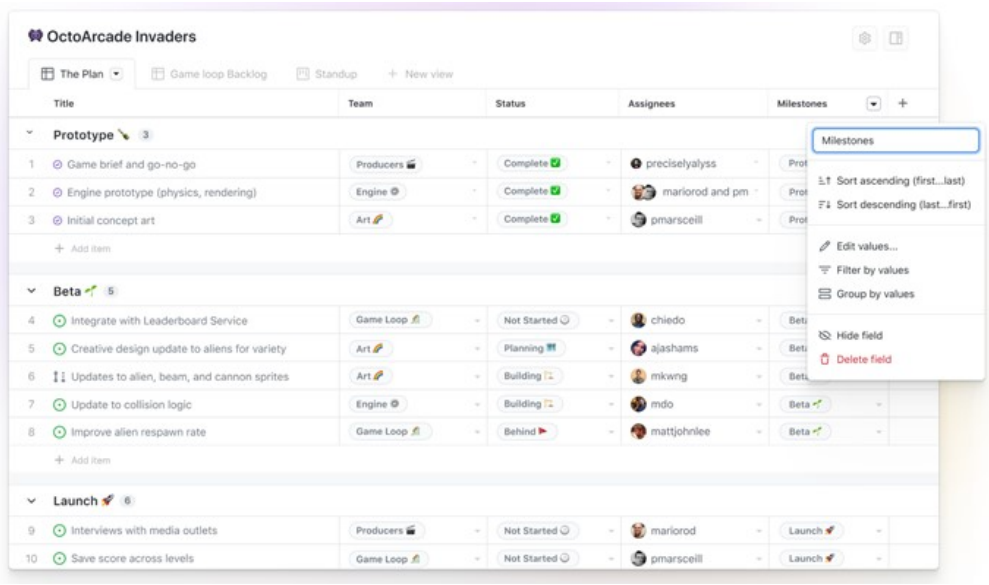

Figura 3. Vista de GitHub *issues*. Fuente: GitHub, s. f.

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 (GitLab, 2022).

El software GitLab se escribió originalmente en el lenguaje de programación Ruby y algunas partes se reescribieron posteriormente en el lenguaje de programación Go. 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). La tecnología de software actual utilizada incluye Go, Ruby on Rails y Vue.js.

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.

GitHub vs GitLab: GitHub tiene una mayor disponibilidad y se focaliza más en el rendimiento de la infraestructura, mientras que GitLab se centra más en ofrecer un sistema basado en funciones con una plataforma integrada y centralizada. Adicionalmente, GitLab puede descargarse e instalarse en los servidores corporativos.

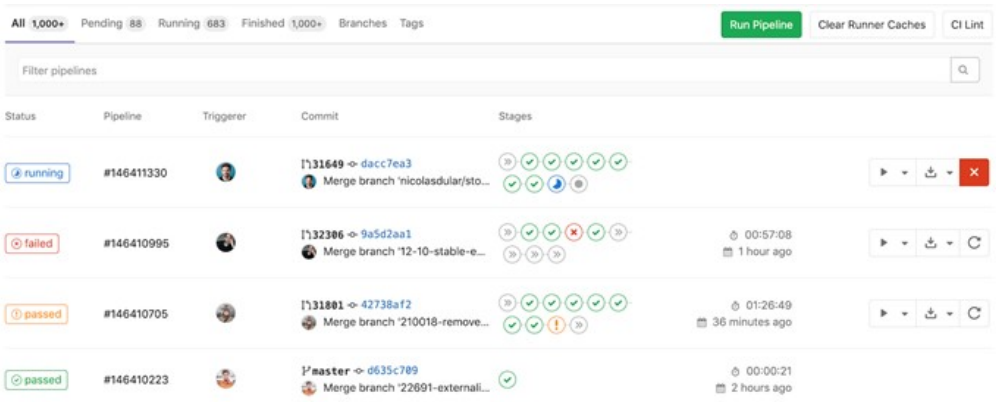

Figura 8. Vista de GitLab *pipelines*. Fuente: GitLab, s. f.

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*.

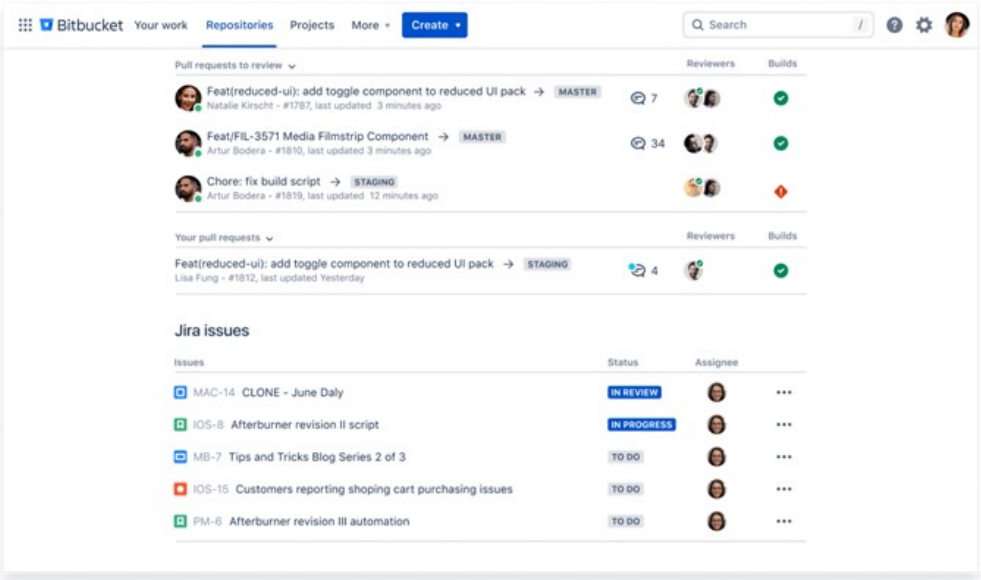

Figura 9. Vista de BitBucket. Fuente: Bitbucket, s. f.

1.5. Referencias bibliográficas

Bitbucket. (s. f.) *Code & CI/CD, optimized for teams using Jira*. https://bitbucket.org/

Blokdyk, G. (2018). *Apache subversion: The ultimate step-by-step guide*. Createspace Independent Publishing Platform.

Chacon, S. y Straub, B. (2014). *Pro Git.* Apress. https://git-scm.com/book/en/v2/Getting-Started-What-is-Git%3F

Nongnu. (s. f.). *CVS - Concurrent Versions System*. https://www.nongnu.org/cvs/

GitHub. (s. f.). *Project planning for developers.* https://github.com/features/issues

GitLab. (2022). *The one DevOps platform*. https://about.gitlab.com/

GitLab. (s. f.) *CI/CD pipelines*. https://docs.gitlab.com/ee/ci/pipelines/

Humphrey, W. (1989). *Managing the Software Process*. Pearson Education.

Humphrey, W. S. (2000a). *The Personal Software Process (PSP)*. Carnegie Mellon Software Engineering Institute. https://doi.org/10.1184/R1/6585197.V1

Humphrey, W. S. (2000b). *The Team Software Process (TSP)*. Caregie Mellon Software Engineering Institute. https://resources.sei.cmu.edu/asset_files/TechnicalReport/2000_005_001_13754.pdf

Roongkaew, W. y Prompoon, N. (2013). Software engineering tools classification based on SWEBOK taxonomy and software profile. *2013 Second International Conference on Informatics & Applications (ICIA)*, pp. 122-128. https://ieeexplore.ieee.org/document/6650241

Sommerville, I. (2015). *Software Engineering* (10ª ed.). Pearson.

Comenzando con Git y GitHub: una guía completa de control de versiones, gestión de proyectos y trabajo en equipo para el nuevo desarrollador

Tsitoara, M. (2019). *Beginning Git and GitHub: A comprehensive guide to version control, project management, and teamwork for the new developer* (1ª ed.). Apress.

Libro que proporciona una explicación del control de versiones de software utilizando Git y el uso de la plataforma GitHub.

Git para equipos

Westby, H. y Jane, E. (2015). *Git for Teams*. O'Reilly Media.

Esta guía práctica ofrece un enfoque único del control de versiones que prioriza a las personas y también explica cómo el uso de Git como punto focal puede ayudar a su equipo a trabajar mejor en conjunto. Aprenderá a planificar y seguir un flujo de trabajo de Git que no solo garantice que logre los objetivos del proyecto, sino que también se adapte a las necesidades inmediatas y al crecimiento futuro de su equipo.

Test

1. Una colección de herramientas para ayudar al programador a crear e implementar código/contenido que está específicamente diseñado para ejecutarse en una plataforma muy particular o de una manera muy particular se denomina:

A. Marco de referencia (*framework*).

B. Kit de desarrollo de software (*software development kit*, SDK).

C. Caja de herramientas (*toolkit*).

D. Biblioteca.

2. Relaciona las características con las diversas plataformas *low-code*:

GitLab	1
Bitbucket	2
Subversion	3
GitHub	4

A	Git de Atlassian
B	Git de Microsoft
C	Git instalable *on premises*
D	No se basa en Git

3. Selecciona cuál es el principal valor de las plataformas de desarrollo de software:

A. Facilitar las pruebas.

B. Mejorar la productividad y eficiencia del equipo de desarrollo.

C. Reducir costes de migración o conversión.

D. Reducir el coste operativo.

4. Selecciona cuál de las siguientes funcionalidades debe realizarse primero de forma manual y sin apoyo de herramientas:

A. Plantear una solución técnica.

B. Entender los requisitos.

C. Definir cómo probar.

D. Todas las anteriores.

5. Ordena, del uno (más antigua) al cuatro (más reciente), el orden en el que se crearon las plataformas de apoyo a la gestión de la configuración:

GitHub	1
Git	2
CVS	3
SVN	4

A	Uno (más antigua)
B	Dos
C	Tres
D	Cuatro (más reciente)

6. El control de versiones es una práctica, integrada en la gestión de la configuración del software, responsable de gestionar cambios en:

A. Solo en el código fuente.

B. En el código y en los requisitos

C. En el código, su documentación o cualquier colección de información relevante para su desarrollo o evolución.

D. Solo en la documentación.

7. ¿Qué estado no forma parte de un fichero bajo un control de versiones con Git?

A. Modificado (*modified*).

B. Bifurcado (*branched*).

C. Preparado (*staged*).

D. Comprometido (*committed*).

8. ¿Qué funcionalidad no forma parte de un control de versiones?

A. Deshacer ediciones específicas sin perder el trabajo realizado mientras tanto.

B. Proporcionar acceso a las versiones históricas de un proyecto.

C. Garantizar la trazabilidad del código fuente respecto a otra documentación del proyecto, como requisitos, incidencias, diseños, etc.

D. Integrar el trabajo que se realiza simultáneamente por diferentes miembros del equipo.

9. En el ámbito del control de versiones, si utilizamos bifurcaciones, los objetos se gestionan en:

A. Deltas.

B. *Releases*.

C. Ramas.

D. Instantes.

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

A. Un equipo de personas formadas y especializadas en roles que trabajan en equipo.

B. Un método y procesos que guíen en la construcción del software.

C. Un conjunto de herramientas que facilitan y aceleran el desarrollo.

D. Todas las anteriores.