Universidad del País Vasco Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

GRADO EN INGENIERÍA EN TECNOLOGÍA DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

NUEVA METODOLOGÍA DE ENSEÑANZA DE PROGRAMACIÓN BASADA EN TECNOLOGÍAS DE CONTROL DE VERSIONES (NME-CV)

Alumno: Martín García-Cuevas, Eder

Director: Ferro Vázquez, Armando

Curso: 2017-2018

Fecha: 24-07-2018

RESUMEN TRILINGÜE

La implantación de planes de estudio bajo la reforma propugnada por el modelo de Bolonia, exige nuevos métodos de enseñanza orientados hacia el aprendizaje práctico. Asignaturas como la enseñanza de lenguajes de programación disponen actualmente de herramientas de desarrollo software que facilitan el control de versiones y que son fundamentales en la formación del alumnado. Este estudio plantea el desarrollo de una metodología nueva de enseñanza de asignaturas de programación que puedan hacer un uso adecuado de las herramientas de control de versiones y de entornos de desarrollo profesionales disponibles en el mercado.

Bolonia ereduak aldeztutako erreformaren menpe dauden ikasketa-planen ezarpenak, ikaskuntza praktikoari orientatutako irakaskuntza-metodo berriak exijitzen ditu. Programazio-lengoaien irakaskuntza bezalako jakintzagaiak badituzte, gaur egun, bertsio-kontrola errazten dituzten software-garapenerako erramintak, eta hauek ikasleen heziketarako nahitaezkoak dira. Estudio honek bertsio-kontrolerako erramintak eta merkatuan erabiltzeko moduan dauden garapen ingurune profesionalak erabiltzen dituen programazio jakintzagaien irakaskuntzarako metodologia berri baten garapena proposatzen du.

The implementation of curricula under the reform advocated by the Bologna model requires new teaching methods oriented towards practical learning. Subjects such as the teaching of programming languages currently have software development tools that facilitate the version control and they are fundamental in the training of students. This study proposes the development of a new methodology for the teaching of programming subjects that can make appropriate use of the version control tools and professional development environments available in the software development market.

ÍNDICE

ÍNDICE DE FIGURAS Y TABLAS

TABLA DE ACRÓNIMOS

ACRÓNIMO	SIGNIFICADO
API	*Application Programming Interface*
CI	*Continuous Integration*
DB	*DataBase*
DVCS	*Distributed Version Control System*
ECTS	*European Credit Transfer System*
EIB	*Escuela de Ingeniería de Bilbao*
GRAPHQL	*Graph Query Language*
GPL	*General Public Licence*
HTTP	*HyperText Transfer Protocol*
HTTPS	*HyperText Transfer Protocol Secure*
JSON	*JavaScript Object Notation*
LMS	*Learning Management System*
NME-CV	*Nueva Metodología de Enseñanza de programación basada en tecnologías de Control de Versiones*
OAUTH	*Open Authorization*
REST	*REpresntational State Transfer*
UPV/EHU	*Universidad del País Vasco/Euskal Herriko Unibertsitatea*
URL	*Uniform Resource Locator*
VCS	*Version Control System*

INTRODUCCIÓN

Es evidente que las soluciones software están cada vez más presentes en nuestro día a día en cantidad de temas. Los avances tecnológicos en áreas como la electrónica y el desarrollo de software han sido espectaculares en las últimas décadas y acercan poco a poco a la sociedad a nuevas invenciones que parecen más propias de la ciencia ficción. Robots cada vez más humanos e independientes, electrodomésticos inteligentes, domótica… Poco a poco, deja de sorprender encontrar titulares en la prensa tanto clásica como digital sobre cualquiera de estos temas y muchos más relacionados que afectan a nuestra vida cotidiana.

En concreto, la integración del software en dispositivos de uso cotidiano aporta, además de una mejora en las funcionalidades de los mismos, un mayor grado de adaptabilidad y facilidad de uso. El interés general de la sociedad por el uso del software crece a pasos agigantados, de forma que cada vez más personas eligen dedicar su vida laboral al desarrollo y mantenimiento de diversas aplicaciones y servicios para múltiples plataformas disponibles. Pero, para ello, se hace necesaria una formación adecuada de las personas en el uso de la tecnología disponible. Aumentan las matrículas en cursos de enseñanza de distintos lenguajes de programación, y en diferentes disciplinas y especializaciones. Esta formación especializada es también necesaria en titulaciones técnicas tanto en grados medios como universitarios, en cursos específicos de formación online, etc.

Al llegar a las asignaturas de programación, el alumno, normalmente, se encuentra con una forma de evaluación a la que ya estará más o menos acostumbrado pero que no era habitual hasta hace relativamente poco. Actualmente, en el ámbito académico en general, y en el universitario en particular a través del conocido como Plan Bolonia, se establece que los alumnos han de someterse a una evaluación continua a lo largo del curso académico. Para ello, los profesores no deben limitarse a impartir el contenido teórico de la asignatura, si no que deben complementarlo con una serie de ejercicios prácticos que les ayuden a asimilar los diferentes conceptos. Los resultados de las tareas prácticas, además, deben tener su peso correspondiente en la evaluación final, logrando que aprobar una asignatura no consista únicamente en lo bien o mal que se ha resuelto un examen final, sino que también se potencia el trabajo continuado y la asimilación de competencias prácticas demostrables.

Por tanto, para poder seguir este esquema, es necesario realizar un seguimiento de los avances del alumno a lo largo del curso académico, lo que en ocasiones puede resultar tedioso, complicado o, directamente, imposible. Atendiendo a esta necesidad, en los últimos años se han desarrollado nuevas herramientas que permiten solventar algunos de los problemas derivados.

Si bien los instrumentos facilitados son suficientes en algunas ocasiones, hay casos en los que no llegan a ser una solución completa, como ocurre en las asignaturas de programación. Aunque existe la tecnología necesaria, el soporte facilitado para apoyar la educación no permite, por

ejemplo, ejecutar automáticamente pruebas al código elaborado por los alumnos y tampoco es posible gestionar fácilmente el control de versiones.

Surge por tanto la necesidad de diseñar una metodología capaz de aplicar de manera sencilla y eficiente la evaluación continua a las asignaturas de programación utilizando para ello tecnologías más modernas ya disponibles por los equipos de desarrollo de software.

CONTEXTO

Actualmente, los estudios en las universidades españolas se rigen por el conocido como Plan Bolonia. Nacido en el último año del pasado siglo, se desarrolló con el fin de reformar los estudios universitarios a nivel europeo.

Una de las bases consistía en establecer un sistema internacional de créditos, lo que supuso la creación de los ECTS (*European Credit Transfer System*) En este nuevo sistema, no se contabilizan los créditos usando únicamente las horas de clase impartidas por el profesorado, sino que también se tiene en cuenta el trabajo que debe realizar el alumno, ya sea en casa o a través de distintas tareas propuestas por el profesor (trabajos, prácticas de laboratorio, etc).

De esta forma, el trabajo del alumno a lo largo del curso también se ve recompensado en la nota final, dando lugar a una evaluación continua. Sin embargo, esto introduce una nueva dificultad, y es que el cuerpo docente ha de realizar un seguimiento de los avances de sus alumnos, recogiendo de forma fácil y ordenada todos los trabajos, informes sobre las prácticas…

En la Universidad del País Vasco (en adelante, UPV/EHU) el problema se ha resuelto a través de herramientas como eGela, que es una implementación del conocido servicio Moodle. Mediante la plataforma, los profesores de las distintas asignaturas de cualquier grado encuentran facilidades para realizar este seguimiento. Por ejemplo, pueden crear un espacio virtual para que los alumnos suban los resultados de sus trabajos para su posterior evaluación numérica, permitiendo, además, dar un *feedback* de la corrección. Incluso se pueden realizar distintos tests para poner a prueba los conocimientos del alumnado que también sirven para la nota final. Todos estos procesos quedan registrados en la plataforma, lo que permite que la información sea consultada o modificada en cualquier momento.

La Escuela de Ingeniería de Bilbao (en adelante EIB) no es una excepción y también hace uso de eGela en los distintos grados que oferta. Debido a las competencias que deben asimilar los matriculados en estos cursos, es necesario que en varios de ellos se enseñe a programar en diferentes asignaturas. Es en estos casos cuando la implementación de Moodle empieza a mostrar carencias, ya que no es posible aprovechar tecnologías ya desarrolladas que faciliten las labores de seguimiento, como ya se ha comentado en el apartado anterior.

En uno de los intentos por resolver este problema, se desarrolló el programa Monitor. Usado de forma independiente a eGela en la asignatura Arquitectura de Sistemas de Información del tercer curso del grado en Ingeniería en Tecnología de Telecomunicación, este programa permite corregir automáticamente los códigos desarrollados por los alumnos a lo largo de las prácticas de la asignatura, así como hacerles saber sus progresos vía una página web.

Esta herramienta, desde luego, facilita el seguimiento de esa asignatura, pero, además de ciertas carencias técnicas, acaba convirtiéndose en una solución específica para un contexto muy concreto, por lo que sigue sin existir una solución unificada que sea aplicable, como poco, a los grados de la EIB.

Por otro lado, en el mismo grado sólo se enseña a usar la tecnología de control de versiones en una pequeña práctica de la asignatura Tecnología de Ingeniería Telemática del cuarto curso, aunque el tiempo asignado no es suficiente para comprenderla del todo, aun cuando es imprescindible en el desarrollo software.

El control de versiones, también denominado por sus siglas en inglés VCS (*Version Control System*), es la tecnología que gestiona los cambios que se han realizado sobre un fichero o un conjunto de ellos guardados en repositorios (almacenes de versiones), permitiendo recuperar la forma que tenían anteriormente. Aunque podría usarse en cualquier campo que haga uso de cualquier tipo de ficheros digitales, su principal uso ha sido en la programación, ya que permite identificar los cambios que va sufriendo un programa en constante evolución durante el proceso de desarrollo, así como restaurar versiones antiguas desde las que partir en nuevas adaptaciones o dar soporte a los usuarios que todavía la usen, entre otros.

Aunque para un ingeniero técnico de telecomunicaciones conocer dicha tecnología no sea imprescindible, le beneficiaría bastante debido al peso que adquiere la programación en su titulación (más todavía para los alumnos que cursan la especialidad de Telemática). Sin embargo, es difícil encajarlo en las pocas asignaturas de programación ofertadas, ya que se necesita enseñar otros conceptos más necesarios.

OBJETIVOS Y ALCANCE

Se plantea como objetivo principal del trabajo **el desarrollo de un estudio técnico que permita crear una nueva metodología de enseñanza práctica de asignaturas sobre programación basada en la utilización de herramientas disponibles en el mercado para el desarrollo de software profesional y que facilite la evaluación continua del alumno de forma sencilla a lo largo de un curso académico**. Se entiende por sencilla que sea accesible para profesores que estén familiarizados con el uso de plataformas orientadas a la educación y/o a la programación de cara a que pueda ser aplicada en el mayor número de asignaturas posibles. Además, los alumnos deben ser capaces de aprender a usar estas plataformas sin demasiado esfuerzo adicional.

Además de este, también se plantean una serie de objetivos adicionales

- **Sentar las bases para la realización de un proyecto piloto funcional**. Los resultados del estudio deben orientar de forma clara el enfoque de la solución para facilitar la implementación de un proyecto piloto que permita validar tanto la nueva metodología de enseñanza como las soluciones técnicas adoptadas.
- **Plantear una arquitectura de referencia**. Para poder llevar a cabo el punto anterior, es muy importante que se plantee qué módulos tienen que existir y cómo deben estar integrados.
- **Decidir el uso de las plataformas o tecnologías disponibles a integrar**. No es necesario utilizar exclusivamente una plataforma, sino que se abre la posibilidad a integrar varias de ellas con las respectivas tecnologías que subyacen. Por tanto, es necesario decidir qué papel cumplirán dentro de la arquitectura.
- **Solución aplicable en la EIB**. Aunque se busca una metodología lo más general posible, por el momento es suficiente con que sea válido para su implementación en el contexto de los grados que oferta la EIB.

Se quiere hacer especial hincapié en que se trata de un estudio, por lo que no se proporciona ninguna clase de código ni desarrollo. Además, como se ha mencionado en el objetivo principal, el estudio se limita a conocer mejor las herramientas ya existentes en el mercado, integrando las distintas soluciones en vez de desarrollar nuevas. Aun así, para la toma de decisiones sobre el enfoque del diseño, se considera la opción de desarrollar algunas herramientas que permitan automatizar algunos procesos de las plataformas estudiadas, haciendo uso de los recursos adicionales de las que éstas proveen, como puede ser el caso de funciones implementables a través de distintas APIs (*Application Programming Interface*).

Para este trabajo, se ha asignado un nombre en forma de acrónimo a dicha metodología para una mejor referencia en los puntos posteriores. Dicho acrónimo es NME-CV (*Nueva Metodología de Enseñanza de programación basada en tecnologías de Control de Versiones*).

BENEFICIOS

La realización de este estudio conlleva una serie de beneficios que han sido divididos en tres categorías: técnicos, económicos y socioculturales.

Técnicos

B.TE.1. Mejoras en la evaluación continua

En el apartado técnico, este es el mayor beneficio, y es que se va a mejorar el sistema de evaluación continua de una asignatura de programación, creando una metodología más eficiente que la actual y agilizando los procedimientos de las mismas.

B.TE.2. Arquitectura

Como ya se ha mencionado, uno de los objetivos es plantear una arquitectura de referencia para la implementación de la NME-CV. Esto facilitará su posterior introducción en los diferentes contextos de cada centro educativo, aunque en principio es suficiente con que sea válido para la EIB.

Económicos

B.EC.1. Aumento de matriculaciones

El mayor beneficio económico aportado es que las mejoras en la enseñanza que aporta la NME-CV se verán reflejadas en una mejor imagen y fama, lo que se traducirá en más matriculaciones en los centros educativos que la implementen.

B.EC.2. Disponibilidad de una plataforma técnica de soporte a la nueva metodología

Aunque la realización del estudio no conlleva la implementación de una solución técnica definitiva, a medio plazo si será necesario desarrollar una solución piloto basada en las propuestas de diseño del estudio que permitirá disponer de una plataforma técnica de despliegue de este nuevo método de enseñanza que tendrá aspectos innovadores y diferenciadores sobre lo disponible en el mercado, lo cual permitirá una explotación con beneficios económicos importantes.

Socioculturales

B.SC.1. Mejores profesionales

Sin lugar a dudas, el mayor beneficio sociocultural es precisamente el fin último de cualquier metodología de enseñanza: formar mejores profesionales. El alumno, además de familiarizarse con el VCS, mejorará al ser consciente de sus errores de forma rápida y podrá sentirse motivado y satisfecho al ver cómo los ha solucionado.

En la programación, esto se traduce en la creación de programas y aplicaciones más complejas, eficientes y seguras, características de las que se beneficiará la sociedad en general.

B.SC.2. Ahorro de tiempo

Otra ventaja, esta vez aplicable dentro de los centros educativos, es que ahorrará tiempo a los profesores a la hora de evaluar, tiempo que podrán aprovechar en atender al resto de sus responsabilidades.

Además, al obtener resultados inmediatos, los alumnos podrán conocer si su trabajo es o no correcto más rápido, pudiendo organizarse mejor para completar el resto de sus tareas. Aparte, se quedarán atascados en la resolución de sus ejercicios prácticos con menos frecuencia, por lo que podrán avanzar más rápido.

ANÁLISIS DEL ESTADO DEL ARTE

Como ya se ha mencionado en el contexto, el VCS es la tecnología que permite gestionar los cambios introducidos en los ficheros dentro de un repositorio. Puesto que se trata precisamente de la base de la NME-CV, no se profundiza más en esta tecnología, sino que sólo se menciona que se va a asumir el uso de Git sobre otras soluciones debido a lo extendido de su uso.[1]

Para poder desarrollar correctamente la metodología, es necesario usar algún servicio web que sirva como espacio virtual para almacenar la información referente a la asignatura, permitiendo recoger ordenadamente tanto documentos con su contenido teórico-práctico como con los resultados de las prácticas de los alumnos. Debido a las características de la NME-CV, se estudiarán dos tipos de tecnologías: LMS y forjas.

Se denomina LMS (*Learning Management System*) a las aplicaciones software para la administración, documentación y seguimiento de cursos educativos de distintos niveles. Están pensadas para dar la posibilidad de gestionar la realización de un curso online, aunque se pueden utilizar también en otras modalidades, como puede ser en la educación semipresencial (en inglés, *blended learning*) que combina la impartición de clases presenciales con trabajo online realizable fuera del aula.

Algunos ejemplos de LMS son:

- **Moodle**.[2] Constituye la solución más popular del mercado. Licenciado bajo GPL, permite poner en marcha la plataforma con rapidez. Sin embargo, puede resultar poco intuitivo para los novatos y difícil de mantener en entornos con muchos usuarios.
- **Blackboard Learn**. Esta solución es casi totalmente opuesta a Moodle. De pago y con algunas dificultades para su instalación, aunque más intuitiva y útil en entornos con muchos usuarios.
- **Chamilo**. Esta es otra solución con licencia GPL. Resulta más sencilla de usar en comparación a otros LMS. Trata de poner su enfoque sobre aspectos más sociales.

En cualquier caso, suelen disponer de plugins que permiten añadir funcionalidades adicionales.

Se denomina forja (*forge* en inglés) a los servicios web para facilitar, estimular y concentrar los proyectos de desarrollo software de la comunidad, creando una plataforma colaborativa. Para ello, suelen integrar soluciones para la comunicación entre desarrolladores (foros, wikis, etc),

[1] Para más información sobre Git y GitHub, consultar el Anexo I.
[2] Para más información sobre Moodle, consultar el Anexo II.

VCS o descarga de repositorios, lo que permite compartir el código tanto con colaboradores como con usuarios interesados en el programa.

Algunos ejemplos de este servicio son:

- **GitHub**.[3] A día de hoy, es la mayor forja del mundo. Hace especial hincapié en introducir un componente social al proceso y dispone de un montón de funciones a través de su API para mejorar las prestaciones del servicio. Desde hace algunos años, ha entrado en el sector educativo a través de GitHub Education y GitHub Classroom[4], que pretenden servir de apoyo a la enseñanza de programación.
- **SourceForge**. Fue la primera forja de Internet y referencia indiscutible hasta la llegada de GitHub, que terminó por superarla. A día de hoy, ambas plataformas ofrecen características muy similares.
- **GitLab**. También es una solución muy similar a GitHub, aunque destaca sobre sus competidores en materia de seguridad.

Otro punto a estudiar es la forma en la que se realizan las pruebas al software entregado por los alumnos. Para ello, el profesor debe desarrollar unos pequeños códigos que fuercen a cada programa a seguir ejecutarse bajo unas ciertas condiciones, de forma que se demuestre el correcto funcionamiento del mismo bajo distintas situaciones.

Para poder ejecutar estos test de forma automatizada, se puede usar la tecnología derivada del concepto CI (*Continuous Integration*). El concepto en sí se cimienta en la idea de que los desarrolladores software deben enviar de forma periódica los cambios introducidos en el software a un repositorio común y ejecutar pruebas para así facilitar el proceso de integración de los distintos módulos a desarrollar. Usando este concepto como base, en los últimos años se han desarrollado herramientas que permiten facilitar su aplicación, automatizando las tareas. Las más conocidas de éstas son Travis y Jenkins.

Para poder gestionar de forma automática todo lo que ocurre en las diferentes piezas que conforman la NME-CV, primero es necesario conocer el momento en que ocurre un cierto evento en alguno de los módulos para después recoger y procesar la información relevante. Para ello, y teniendo en cuenta que la metodología podría llegar a tener varios módulos separados y comunicados a través de Internet, se procede a estudiar los webhooks.

Los webhooks son, básicamente, callbacks HTTP (*HyperText Transfer Protocol*), o sea, códigos que se ejecutan en el momento en el que se produzca un evento en una web y que se resuelve mediante un mensaje HTTP-Request hacia una URL (*Uniform Resource Locator*) configurada de

[3] Ver nota 1.
[4] Para más información sobre GitHub Classroom, consultar el Anexo III.

antemano. El interés que suscita esta tecnología es que algunas plataformas (como GitHub, por ejemplo) los implementan enviando en dicho mensaje HTTP-Request tipo POST la información sobre el evento que acaba de ocurrir.

De esta forma, el usuario puede desarrollar un software que procese la información contenida en el mensaje y, conocida la situación, actuar de la forma deseada. El uso de esta tecnología obliga a disponer de un servidor accesible a través de Internet.

DESCRIPCIÓN DE REQUERIMIENTOS

A continuación, se muestran las especificaciones que debe cumplir la NME-CV agrupadas por temática. Además, se añade un código para su posterior referencia desde otros apartados del trabajo.

Administración de usuarios

E.US.1. Cuentas personales

Tanto alumnos como profesores deben disponer de al menos una cuenta en alguna de las plataformas a integrar que permita reconocer al alumno que está realizando el trabajo. Además, la identificación de un alumno a través del nombre de su cuenta debe ser lo más sencillo posible.

E.US.2. Grupo cerrado

De una forma u otra, el alumnado debe estar agrupado y organizado, teniendo ellos y sólo ellos acceso a los contenidos. Sin embargo, debido al flujo de alumnos entre cursos académicos, esta organización debe ser flexible en cuanto a entradas y salidas.

Administración de la asignatura

E.AS.1. Almacenamiento de resultados

No basta con corregir códigos automáticamente. Para poder realizar correctamente la evaluación continua, es necesario que los resultados puedan almacenarse en algún lado para ser consultados posteriormente.

E.AS.2. Gestión documental

Como parte del seguimiento, es interesante que se pueda gestionar fácilmente la adición y eliminación de documentos relevantes para la asignatura, sea contenido teórico, enunciado de prácticas, enlaces de interés…

Fiabilidad de los resultados

E.FI.1. No introducir vulnerabilidades ante plagios

Aunque el sistema no sea capaz de detectar explícitamente que el trabajo de un alumno es en verdad un plagio, por lo menos no debe introducir nuevos puntos débiles que puedan aprovecharse para la realización de los mismos.

Procesos de la metodología

E.ME.1. Realización de pruebas automáticamente

El principal problema de la evaluación continua es que requiere que los profesores estén constantemente corrigiendo y evaluando trabajos. Se busca automatizar este proceso para hacerlo más rápido y eficiente, aunque el profesorado sigue teniendo que desarrollar las pruebas que se deben de realizar, ya que dependen de cada ejercicio a resolver.

E.ME.2. Metodología lo más automatizada posible

Mientras que en la especificación **E.ME.1** se obliga a que la corrección sea automática, el resto de tareas no tienen por qué serlo, aunque la mayoría deben serlo, de cara a que todo el proceso sea lo más rápido y eficiente posible.

Red social

E.RS.1. Componente social

Aunque no es estrictamente necesario para seguir el desarrollo de una asignatura, resulta muy interesante poder aprovechar herramientas con las que los alumnos puedan mantenerse en contacto tanto con sus compañeros como con sus profesores. De esta forma, son capaces de consultar y resolver dudas en cualquier momento, incluso compartir ideas sobre la resolución de un problema.

ANÁLISIS DE ALTERNATIVAS

A continuación, se muestra el proceso seguido para resolver ciertos problemas de ingeniería de bastante relevancia en el trabajo. Para ello, se ha realizado un análisis de alternativas que ayude en la búsqueda de dicha solución. Se empezará identificando las problemáticas a estudiar para después enumerar las múltiples alternativas posibles. Una vez conocido esto, se procederá a la definición de los criterios de selección para, finalmente, decidir la mejor solución en base a la puntuación que obtengan las distintas alternativas.

Dos de las problemáticas estudiadas están estrechamente relacionadas, de forma que la identificación de las alternativas de la segunda depende directamente de la solución propuesta en la primera. Es por ello que se ha decidido realizar el análisis completo a cada problemática de forma secuencial.

Todas las problemáticas, alternativas y criterios disponen de sus correspondientes códigos de referencia.

Problemática P.1: Sobre la orientación de la metodología

Como se ha visto en el análisis del estado del arte, existen un par de tecnologías que podrían funcionar como base de la NME-CV, por lo que la metodología podría orientarse hacia cualquiera de las dos, permitiendo explotar completamente una de las ramas de funcionalidades en detrimento de la otra.

La solución a esta problemática es clave tanto para el desarrollo de la arquitectura de referencia como para el del piloto funcional y también cobra especial relevancia a la hora de exportarla a otros contextos. Todo esto se debe a que el decantarse por una opción u otra limita el acceso a una parte del mercado. Además, está estrechamente relacionado con el trabajo de integración que supone dotar a la metodología de todas las funciones necesarias, lo que puede poner obstáculos a otros centros educativos que quieran implementarla.

Alternativas sobre la orientación de la metodología

A.1.1. Basada en LMS

Una primera solución posible a la problemática P.1 es hacer uso exclusivo de las aplicaciones LMS estudiadas en el análisis del estado del arte.

De cara a la arquitectura de referencia, usar únicamente LMS trae consigo una gran ventaja, y es que, aunque podrían seguir siendo necesarios algunos equipos adicionales (como un servidor para recibir y procesar los eventos, una base de datos o ambas a la vez), se pueden centralizar la mayoría de las funcionalidades de la metodología en una única plataforma. Gracias a ello, las personas involucradas en el proceso de aprendizaje no necesitan ni varias cuentas ni aprender a usar más de una plataforma.

Sin embargo, en este tipo de aplicaciones no existe una forma sencilla de aplicar ni la tecnología VCS y ni las pruebas al código. Para poder integrar ambas, es necesario buscar o desarrollar plugins, módulos o similares que permitan aumentar las funcionalidades de las aplicaciones LMS instaladas (además de automatizar los procesos), lo que supone una sobrecarga de trabajo importante.

Por otro lado, esta alternativa puede resultar realmente problemática en muchos centros educativos porque ya podrían estar haciendo uso de alguna aplicación de este tipo para el resto de los cursos ofertados. Éstas suelen encontrarse bajo el control del centro y no del profesor por lo que, si los administradores de la aplicación no están dispuestos a aplicar los cambios necesarios, la única solución sería que el profesor instalase otra aplicación LMS para sus asignaturas, lo que, a la postre, podría generar un sentimiento de rechazo por parte del alumnado.

A.1.2. Basada en forjas

Otra alternativa válida es usar únicamente forjas.

En cuanto a la arquitectura, se necesita una única plataforma para albergar la mayoría de las necesidades de la metodología, aportando los mismos beneficios descritos en la anterior alternativa.

Para poder integrar las funcionalidades propias de LMS, también es necesario buscar/desarrollar alguna herramienta que permita dotar a los repositorios de las funcionalidades deseadas.

No obstante, esta integración resulta algo más sencilla que en el anterior caso, debido a que sería suficiente con usar la tecnología de repositorios ya incluida en las forjas. Habría que gestionar ficheros que contengan la información correspondiente al seguimiento de una asignatura, añadiendo el desarrollo de un programa que permita realizar este proceso de forma más cómoda para cumplir con la especificación **E.ME.2**.

Otro punto negativo a tener en cuenta es que no se aprovechan de ninguna manera las aplicaciones LMS que ya pueda tener instaladas el centro educativo (como en el caso de la EIB). Además de lo que ya supone de por sí, se le suma el rechazo que puede generar en el alumnado el tener que usar LMS para todas las asignaturas salvo las de programación, en las que usarían una plataforma totalmente distinta.

A.1.3. Solución mixta

Una última solución interesante consiste en olvidarse de la idea de usar sólo una de las tecnologías y apostar por utilizar ambas a la vez en su correspondiente contexto, o sea, disponiendo de una aplicación LMS para la gestión del seguimiento de la asignatura y valiéndose de los servicios de una forja con el fin de que los alumnos suban su código para su corrección.

En este caso, se potencian ambos aspectos, permitiendo acceder a todas y cada una de las funcionalidades que ofrecen las dos tecnologías. De esta forma, no es necesario realizar ni desarrollos ni búsquedas para ampliar las funcionalidades, sino que basta con realizar un trabajo de integración, en el que también se podrían usar equipos adicionales. Además, esta solución sí permite aprovechar el trabajo realizado previamente en la puesta en marcha de una aplicación LMS.

La gran desventaja de esta alternativa es que se pierden los beneficios por usar una única plataforma, de tal manera que se deben usar dos cuentas como mínimo: una para la LMS y otra para la forja.

Criterios de selección sobre la orientación de la metodología

C.1.1. Trabajo de integración necesario (50%)

El primer punto a evaluar para la elección de la solución es el esfuerzo que requiere integrar todas las funcionalidades. Se considera la más importante ya que tiene un efecto directo sobre la sencillez de la que se habló en el objetivo principal.[5]

Se valora asignando una nota de **0 a 10**. Una **mayor** nota corresponde a un menor trabajo necesario.

C.1.2. Impacto sobre la arquitectura (30%)

El siguiente punto a valorar son las consecuencias que tiene la aplicación de una alternativa en el diseño de la arquitectura de referencia, o sea, si es necesario usar una o dos plataformas para poder dotar a la metodología de las funciones básicas. Aunque se considera de cierta relevancia, se cree más importante simplificar la implantación de la metodología que usar una o dos plataformas, ya que resulta *a priori* menos laborioso aprender a usar una plataforma que realizar el trabajo de integración.

Se valora asignando una nota de **0 o 10**. Una nota de 0 corresponde a la necesidad de usar más de una plataforma. Una nota de 10 corresponde a poder usar una única plataforma.

C.1.3. Probabilidad de rechazo (20%)

Por último, se valora la probabilidad de que el alumnado no se sienta a gusto con dicha orientación debido a las confusiones que puede acarrear. No se considera la más importante, pero sí a tener en cuenta debido a que influye negativamente en la visión

[5] Ver Objetivos y alcance.

que tiene el alumnado sobre la metodología, lo que podría causar una pérdida de la motivación en el estudio.

Se valora asignando una nota de **0 a 10**. Una mayor nota corresponde a una menor probabilidad de rechazo.

Selección de la solución sobre la orientación de la metodología

En la siguiente tabla, se encuentra la valoración de cada alternativa, así como la elección de una solución:

	C.1.1 (50%)	***C.1.2 (30%)***	***C.1.3 (20%)***	***VALORACIÓN FINAL (sobre 10)***
A.1.1	2	10	4	3,8
A.1.2	5	10	0	5,5
A.1.3	9	0	8	6,1

Tabla 1. Selección de solución a la problemática P.1

La solución adoptada es por tanto la alternativa **A.1.3**, o sea, **orientar la metodología de forma mixta**. Los motivos de su victoria son la **facilidad de integración** con respecto a las otras alternativas (**C.1.1**) y su **alta probabilidad** de aceptación (**C.1.3**), pese a obligar a incluir **otra plataforma** en la arquitectura (**C.1.2**).

Problemática P.2: Sobre la arquitectura de referencia

Como ya se ha mencionado, uno de los objetivos adicionales del trabajo es el planteamiento de una arquitectura de referencia que sirva de base para el desarrollo de un piloto funcional. Esta problemática trata sobre la forma que ésta adopta.

Para poder realizar el estudio, se parte de la solución adoptada en la problemática **P.1**, o sea, sabiendo que se hace uso de una aplicación LMS y de una forja. Conocido esto, se valora la forma en la que se combinan ambas tecnologías, lo que resulta importante a la hora de concretar la forma de la arquitectura.

Alternativas sobre la arquitectura de referencia

A.2.1. Centrada en LMS

La primera alternativa consiste en convertir a la aplicación LMS en la plataforma principal, o sea, centralizar aquí la mayoría de las gestiones y usar la forja para lo justo y necesario, o sea, para los repositorios.

En este caso, se aprovechan al máximo las funciones de LMS, usándola como el espacio en el que el profesorado pone a disposición del alumnado todos aquellos recursos que

puedan ser de utilidad: contenido teórico-práctico de la asignatura, guías docentes, enlaces de interés, notas, calendarios y, en general, cualquier elemento que consideren importante para el seguimiento de la asignatura, exceptuando todo lo que tenga relación directa con el trabajo de codificación de los alumnos. Además, la aplicación se usaría también como centro de la red social, en detrimento de las soluciones que proponen las forjas.

La principal ventaja de esta alternativa es que se aprovechan mejor los recursos que ofrecen las LMS, que son el centro de cualquier seguimiento educativo, añadiendo, simplemente, las funcionalidades de las que carece para el contexto de la programación. Sumado al hecho de lo extendido del uso de LMS en cualquier ámbito académico, también se obtiene una solución sencilla a la que tanto alumnos como profesores están acostumbrados.

El mayor problema es que, al usar únicamente los repositorios, se desperdicia el componente de desarrollo colaborativo que ofrecen junto con las facilidades para comentar o abrir un debate sobre un código concreto. Adicionalmente, el usar una forja de forma tan básica puede ser negativo para los alumnos, que, aparte de no familiarizarse del todo bien con su uso, podrían adquirir una falsa percepción de lo que es en realidad, reduciéndolo a un repositorio en línea.

A.2.2. Centrada en forja

Se trata de la solución opuesta a la alternativa **A.2.1**. En este caso, la pieza central de la arquitectura de NME-CV es la forja en lugar de la aplicación LMS.

Aquí, las funciones que más se aprovechan son las de las forjas, de forma que los alumnos suben a sus repositorios sus códigos y, a través de otros específicos, acceden a la información referente a la asignatura, como enunciados de prácticas o notas. Además, éste se convertiría en el centro social en el que se puedan comunicar alumnos con profesores. La aplicación LMS, por el contrario, en este contexto, se usa como almacén para los documentos no relacionados con la programación en sí, sino con otros aspectos de la asignatura.

Al usar las forjas como base, los alumnos sí se familiarizan perfectamente con su uso, aprendiendo a utilizar una tecnología que casi seguro que acaban usando en un futuro. Además, se ven favorecidos por usar un entorno social centrado en hablar alrededor del código de formas de las que también harán uso tarde o temprano.

El punto negativo se encuentra en el propio aprovechamiento. Las forjas no están pensadas para seguir una asignatura, mientras que las aplicaciones LMS nacieron para ello. Al introducir en las primeras algunas funciones propias de la segunda, se está trabajando en balde, pues se podrían aprovechar los recursos de forma más eficiente.

Además, las funcionalidades de desarrollo colaborativo que introducen las forjas, si bien son interesantes, no adquieren tanta importancia en la NME-CV.

A.2.3. Centrada en ambas

Una última alternativa consiste en usar ambas plataformas al completo, aprovechando todos y cada uno de los recursos que ofrecen.

En este caso, hay que separar bien las funcionalidades de ambas, o sea, usar la aplicación LMS para realizar el seguimiento de la asignatura (de la forma descrita en la alternativa **A.2.1**) y la forja como centro del apartado de programación, aprovechando también la compartición de código y el desarrollo colaborativo. En cuanto a las soluciones para la red social, ya que ambas ofrecen sus soluciones, se propone usar las dos a la vez: las de LMS para discutir sobre la asignatura en sí y las de las forjas para debatir sobre el código a desarrollar en cada práctica.

Como ya se ha dicho, la gran ventaja de adoptar esta solución es poder aprovechar todas las funcionalidades que vienen incluidas en ambos tipos de plataformas, exprimiéndolas al máximo. Y como se usa completamente la forja, también se ayuda a los alumnos a familiarizarse con ellas.

Por otro lado, poder usar todo supone un problema. Está bien poder utilizar más recursos, pero al final se añaden funcionalidades con poca probabilidad de ser usadas en entornos educativos, lo que añade una cierta complejidad al sistema de forma innecesaria.

Criterios de selección sobre la arquitectura de referencia

C.2.1. Aprovechamiento de recursos (40%)

El primer criterio de evaluación es la medida de lo que se aprovechan los recursos que ofrecen las plataformas. Se considera uno de los puntos importantes de la valoración, ya que un mal aprovechamiento complicaría la metodología y podría hacerla fracasar.

Se valora asignando una nota de **0 a 10**. Una mayor nota corresponde a un mejor aprovechamiento.

C.2.2. Sencillez (40%)

Al igual que ocurría en el criterio **C.1.2**, se valora la sencillez de implementación y utilización para poder cumplir con el objetivo principal del trabajo.

Se valora asignando una nota de **0 a 10**. Una mayor nota corresponde a una mayor sencillez.

C.2.3. Familiarización (20%)

El hacer uso de tecnologías muy usadas en el ámbito laboral supone que el alumnado se familiarice con ellas mientras estudia, lo que supone un valor añadido frente a sus competidores en el mercado laboral. En este criterio, se valora si mediante la alternativa se puede ayudar a que el alumno obtenga dicha familiaridad. Aun así, no se considera tan importante como los otros dos criterios porque no es uno de los objetivos de la NME-CV.

Se valora asignando una nota de **0, 5 o 10**. Una nota de 0 corresponde a una familiarización nula. Una nota de 5 corresponde a una familiarización parcial. Una nota de 10 corresponde a una familiarización completa.

Selección de la solución sobre la arquitectura de referencia

En la siguiente tabla, se encuentra la valoración de cada alternativa, así como la elección de una solución:

	C.2.1 (40%)	***C.2.2 (40%)***	***C.2.3 (20%)***	***VALORACIÓN FINAL (sobre 10)***
A.2.1	8	9	5	7,8
A.2.2	2	5	10	4,8
A.2.3	10	4	10	7,6

Tabla 2. Selección de solución a la problemática P.2

Por tanto, la solución adoptada es la **A.2.1**, o sea, **centrarse en el uso de una aplicación LMS**, usando la forja para lo justo y necesario. Los motivos de su victoria son la buena nota que adquiere en los criterios de mayor valoración, pues se **aprovechan adecuadamente los recursos** de la aplicación y los importantes de la forja (**C.2.1**), resulta **sencillo** de implementar y usar (**C.2.2**), aunque se **pierde parcialmente** la ventaja que supone familiarizar al alumno con el uso de las forjas (**C.2.3**).

Problemática P.3: Sobre la evaluación continua

Ya se conoce la forma que tomará la arquitectura de referencia. Ahora, queda conocer cómo se realizan las pruebas al código, de cara a corregirlos de forma automática como se recoge en la especificación **E.ME.1**.

En esta problemática, se estudia la forma en la que se lleva a cabo este proceso, que será iniciado siempre por el alumno, aunque carezca de influencia en la corrección en sí.

Alternativas sobre la evaluación continua

A.3.1. Realizado por el alumno

Como primera alternativa, se plantea la opción de que sea el alumno quien ejecute los test en su propio equipo, después de que el profesor ponga a su disposición los ficheros con las distintas pruebas a realizar junto con un script que permita enviar los resultados, ya sea mandándolos por correo electrónico, dejándolos accesibles a través de algún servicio web o mediante cualquier otro método de transferencia de ficheros.

La principal ventaja de adoptar esta solución es que resulta relativamente sencilla tanto para el profesor como para el alumno, al desarrollar las pruebas y al ejecutarlas respectivamente. Aparte, el alumno se familiariza con el proceso de pruebas, aprendiendo a ejecutarlas y comprendiendo cómo se realizan.

El mayor problema en este caso es que el alumno debe estar pendiente de las actualizaciones o cambios que puedan sufrir los ficheros de pruebas a lo largo del curso, por lo que un despiste en este proceso de revisión puede acabar falseando los resultados. Además, un alumno con buenos conocimientos en la materia y una dudosa ética podría tratar de manipular las pruebas para obtener siempre el resultado deseado.

A.3.2. Usando un servidor

Una nueva alternativa a la problemática **P.3** es usar una máquina bajo el control del profesorado para la realización de pruebas. En este caso, el alumno subiría los resultados de su trabajo a un servidor accesible vía web.

En este caso, el proceso de pruebas es totalmente transparente para el alumno, que no debe realizar ningún paso más aparte de subir los ficheros al servidor. Para el profesor también supone una ventaja, y es que posee el control total sobre los ficheros de pruebas, pudiendo subirlos y actualizarlos con la certeza de que, desde ese momento, todos los alumnos han de superar las mismas pruebas, sin que haya ningún rezagado ni tramposo.

Sin embargo, tiene una serie de problemas algo graves. Por ejemplo, sería necesario desarrollar un programa que detecte el momento en el que se ha realizado una subida y que fuera capaz de reconocer la práctica que se está ejecutando. Además, al ser accesible vía web, sería necesario que tanto profesores como alumnos pudieran autenticarse para poder autorizar la ejecución de pruebas, por lo que requerirían de una cuenta adicional además de las dos que necesitan para acceder a la aplicación LMS y a la forja. Y, por si fuera poco, el alumno no se familiarizaría en absoluto con el proceso de pruebas.

A.3.3. CI

La última alternativa a considerar es usar CI a través de las aplicaciones disponibles por la red que se pueden instalar en los repositorios de ciertas forjas.

De base, no es tan distinta a la alternativa **A.3.1**, pues es el alumno el que dispone de los ficheros de pruebas, obteniendo casi las mismas ventajas y desventajas. La diferencia entre las dos radica en dos puntos.

Por un lado, el alumno aprende a usar una metodología cada vez más usada, familiarizándose con los conceptos que subyacen, lo que supondría un valor añadido a la hora de buscar empleo. En comparación con **A.3.1**, supone que no sólo aprende a ejecutar pruebas, sino que también coge la costumbre de intentar los distintos módulos más a menudo.

Por otro lado, tratar con esta tecnología puede resultar más complejo para las partes involucradas en el proceso, pues se trata de una metodología más potente y complicada de usar de base.

Criterios de selección sobre la evaluación continua

C.3.1. Fiabilidad de los resultados (40%)

El primer punto a valorar lo fiables que son los resultados obtenidos en las correcciones, teniendo en cuenta que un alumno sería capaz de manipular los ficheros de pruebas. Se considera la más importante ya que unos resultados falsos acabarían por hacer fracasar la metodología.

Se valora asignando una nota de **0 a 10**. Una mayor nota corresponde a una mayor fiabilidad.

C.3.2. Sencillez (35%)

El siguiente punto a valorar es el mismo que se ha ido usando en el resto de problemáticas, o sea, valorar lo sencillo que resulta aplicar la solución.

Se valora asignando una nota de **0 a 10**. Una mayor nota corresponde a una mayor sencillez.

C.3.3. Familiarización (25%)

Por último, se valora en menor grado la familiaridad que obtiene el alumno no sólo con el proceso de pruebas, sino también con otras buenas prácticas de programación. No se considera tan importante al no ser un objetivo esencial de la NME-CV.

Se valora asignando una nota de **0, 5 o 10**. Una nota de 0 corresponde a una familiarización nula. Una nota de 5 corresponde a una familiarización con el proceso de

pruebas. Una nota de 10 corresponde a una familiarización tanto con las pruebas como con otros procesos relevantes relacionados.

Selección de la solución sobre la evaluación continua

En la siguiente tabla, se encuentra la valoración de cada alternativa, así como la elección de una solución:

	C.3.1 (40%)	***C.3.2 (35%)***	***C.3.3 (25%)***	***VALORACIÓN FINAL (sobre 10)***
A.3.1	**3**	**8**	**5**	**5,25**
A.3.2	10	3	0	5,05
A.3.3	5	3	10	5,55

Tabla 3. Selección de solución a la problemática P.3

La solución adoptada es la **A.3.3**, o sea, usar **CI**. Los motivos de su victoria son que, pese a no ser tan fiable como la **A.3.2** (usar un servidor), es **algo más fiable** (**C.3.1**) que su competidor directo (**A.3.1**) al considerar que una metodología algo más compleja (**C.3.2**) como los es la CI puede repeler a más gente con ganas de falsear los resultados. Además, se aprovecha el uso de CI para **enseñar buenas prácticas** de programación a los alumnos (**C.3.3**).

DESCRIPCIÓN DE LA SOLUCIÓN Y ARQUITECTURA DE REFERENCIA

En base a lo descrito en los puntos anteriores, se procede a continuación a concretar una propuesta para la NME-CV, proporcionando una arquitectura de referencia para el posterior piloto. Cabe destacar que, esta vez, se plantea un caso válido para su aplicación en la EIB, dejando de lado las referencias generales a la tecnología y especificando una implementación concreta. El motivo de esto es realizar una propuesta en firme para un mejor aprovechamiento futuro.

En la siguiente página se encuentra la arquitectura de referencia, que se explica en las sucesivas páginas.

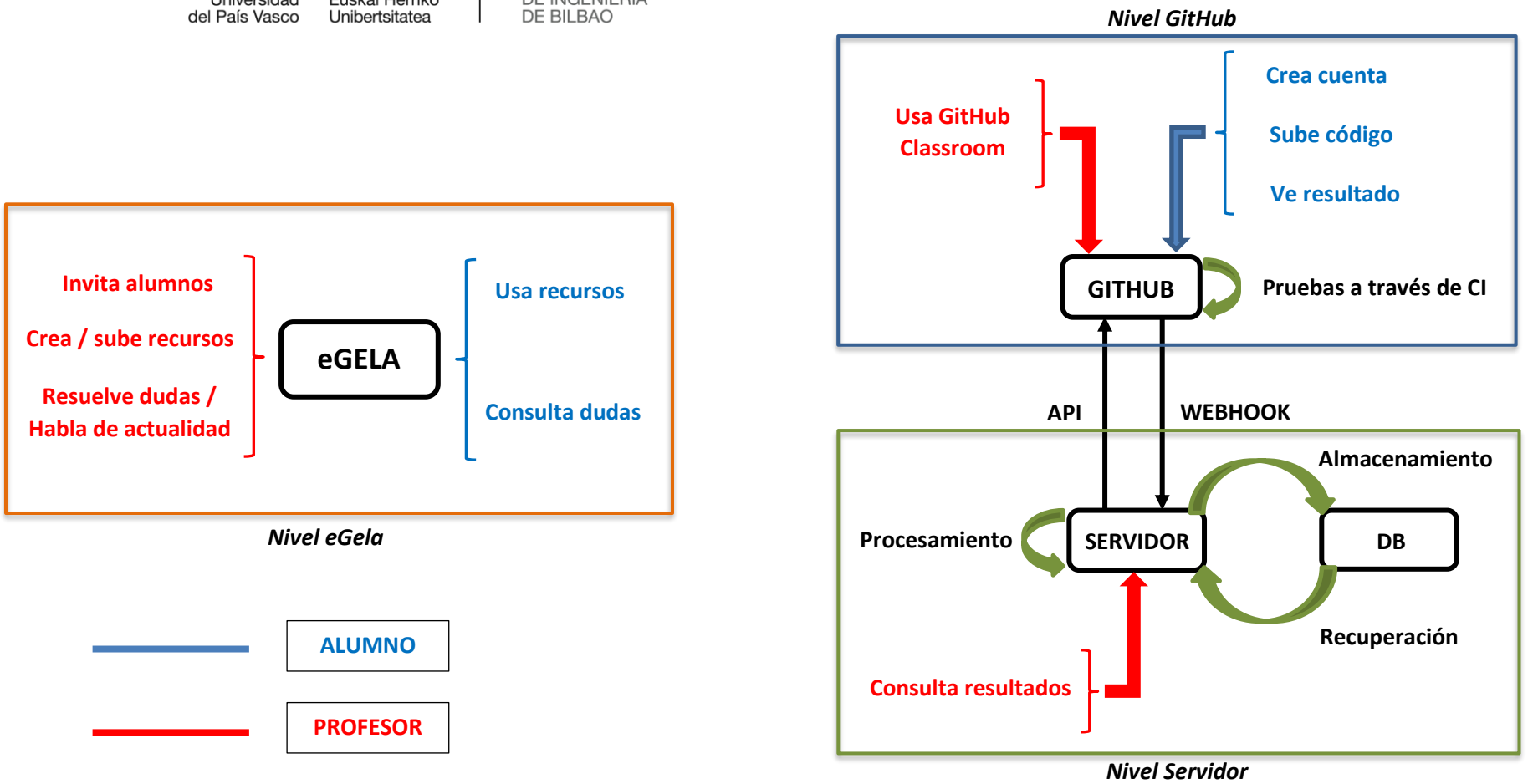

Figura 1. Arquitectura de referencia

Como puede verse en la **Figura 1**, se ha optado por usar una aplicación LMS (eGela), una forja (GitHub) y un servidor conectado a una base de datos que se comunica directamente con la plataforma anterior.

El primer factor que se ha tenido en cuenta a la hora de desarrollar la arquitectura de referencia es el resultado del análisis de la problemática **P.1**, en la que se concluía que debían usarse dos plataformas diferenciadas para así poder simplificar la metodología. En la arquitectura, corresponden a eGela y GitHub, que se usan de forma totalmente independiente al considerar innecesaria su comunicación, que sólo aportaría complejidad.

El otro factor, por el que se introduce el servidor, se debe a la necesidad de automatizar algunos procesos dentro de GitHub, pues la forma que ofrece para esto es usar webhooks (que deben alcanzar a un equipo accesible a través de Internet) para informar de los distintos eventos ocurridos y una API accesible a través de Internet para llevar a cabo distintas funciones sin necesidad de usar la versión web, tras recuperar y procesar la información enviada en los webhooks.

Por tanto, resulta obligatorio disponer de un servidor que se comunique con GitHub para aprovechar estos recursos. Dentro del servidor, además, se añade una base de datos para poder almacenar y recuperar la información relevante a la evaluación, en concreto, los resultados de las pruebas al código, que también llegan al servidor a través de los webhooks.

Terminada la explicación general, se procede a describir cada uno de los distintos niveles por separado, incluyendo las funciones que deben realizar.

eGela

Para empezar, se estudia el uso de eGela, la implementación de Moodle de la UPV/EHU, por lo que corresponde al uso de la tecnología de aplicaciones LMS.

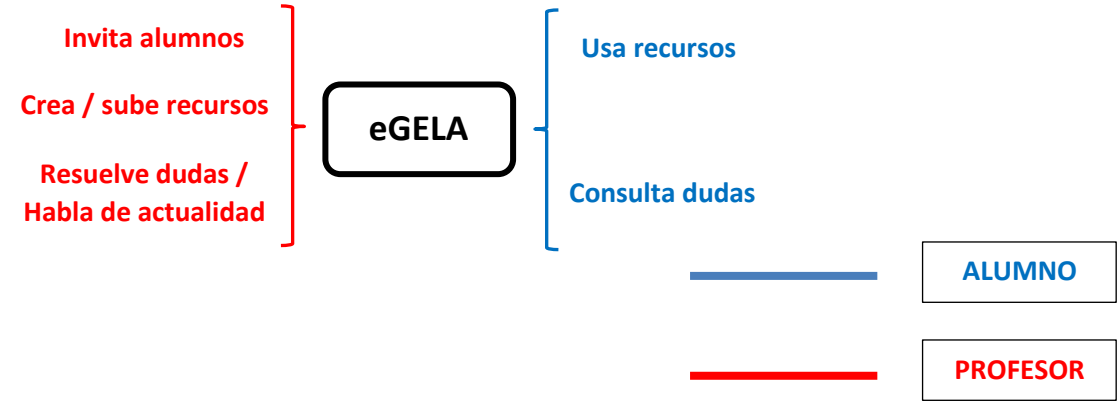

Figura 2. Nivel eGela en la arquitectura

Como se ha explicado durante el análisis de alternativas de la problemática **P.2**, ésta es la pieza central de la NME-CV. Es aquí donde el profesorado pone a disposición de los alumnos los distintos recursos y donde todos pueden comunicarse a través de las soluciones que dispone para ello. Para poder hacer todo esto, tanto profesores como alumnos requieren de una cuenta en eGela, proporcionada por la propia universidad según los criterios y políticas de uso de ésta.

Por tanto, el flujo de tareas a realizar en eGela son:

1. El profesor **da acceso** a los alumnos matriculados en la asignatura al curso de eGela, con el rol de alumno para controlar los permisos.
2. El profesor sube o deja visibles los **ficheros que tengan relación con la asignatura**, siguiendo la política de subida acordada durante la planificación de la asignatura. Entre los ficheros, están los enunciados de las prácticas. Es así como se cumple la especificación **E.AS.2**.
3. El profesor pone a disposición de los alumnos recursos como **foros o wikis** para para poder resolver dudas o comentar la actualidad de la asignatura.
4. El alumno **puede empezar** con la realización de las prácticas con la información proporcionada.
5. Cuando le surja alguna duda, **hace uso de los foros o wikis**. Las dudas se resuelven gracias a alumnos y profesores, cumpliendo con la especificación **E.RS.1**.
6. Adicionalmente, el profesor puede usar otras funciones de eGela como **encuestas, tests o calendarios** según crea conveniente.

GitHub

Para poder usar la tecnología VCS, se utiliza una forja. Para este caso, se ha elegido hacer uso de GitHub, debido a su gran popularidad (lo que supone una mayor probabilidad de que los alumnos la acaben usando en un futuro) y por su acercamiento al mundo académico a través de GitHub Classroom. El lado negativo de este uso es que tanto alumnos como profesores deben disponer de cuentas en esta asignatura también.

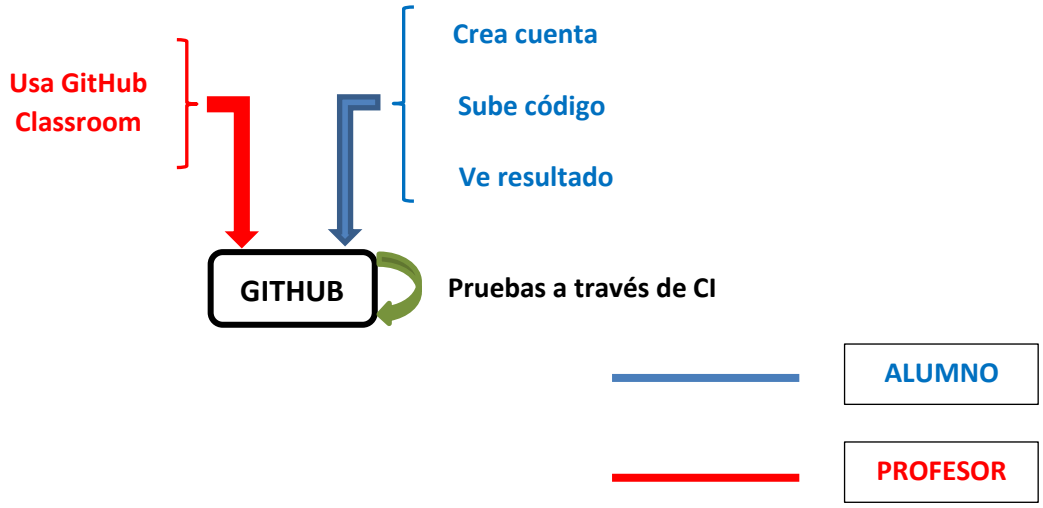

Figura 3. Nivel GitHub en la arquitectura

Aquí, los alumnos deben disponer de una serie de repositorios privados donde puedan almacenar sus códigos. Deben ser privados para así evitar copias y cumplir con la especificación **E.FI.1**. El número de repositorios por alumno varía en función de la naturaleza de los ejercicios prácticos:

- Si no están relacionados entre sí, deberían disponer de un repositorio para cada uno de los ejercicios a fin de separarlos.
- Si están relacionados y, más aún, si cada uno empieza donde acaba el anterior, sería interesante usar un único repositorio y aprovechar la potencia de la tecnología VCS.

Para poder generarlos de forma cómoda, se hace uso de GitHub Classroom. Para poder disponer de este servicio, todos los alumnos deberán estar dentro de una organización de GitHub que represente al grupo completo, estando cada uno de los usuarios dentro del rol correspondiente, lo que cumple con la especificación **E.US.2** teniendo en cuenta las cuentas de eGela.

Aparte, GitHub Classroom permite relacionar cuentas de GitHub con información sobre el alumno (email, nombre, DNI, etc), por lo que gracias al uso del servicio se puede identificar a los alumnos. Otra opción válida es obligar a crear las cuentas eligiendo los nombres de usuario de tal manera que se facilite la identificación (por ejemplo, obligando a que coincida con el usuario de la dirección de correo electrónico facilitada por la UPV/EHU a cada uno de sus alumnos). En este esquema, se recomienda usar la primera opción para que los alumnos que ya dispongan de una cuenta en GitHub puedan usarla sin problemas, aunque la segunda se considera aceptable. En cualquier caso, se cumple con la especificación **E.US.1** si se le suman las cuentas creadas para eGela.

Los repositorios de los alumnos, además, han de tener instalados alguna aplicación de CI (como, por ejemplo, Travis) para la ejecución de pruebas automatizadas, configuradas por el profesor. Así, se cumple con la especificación **E.ME.1**.

Servidor

Para poder cumplir con las dos especificaciones restantes, es necesario automatizar más la metodología y almacenar los resultados de las pruebas. Es aquí donde entra en juego el último nivel de la arquitectura.

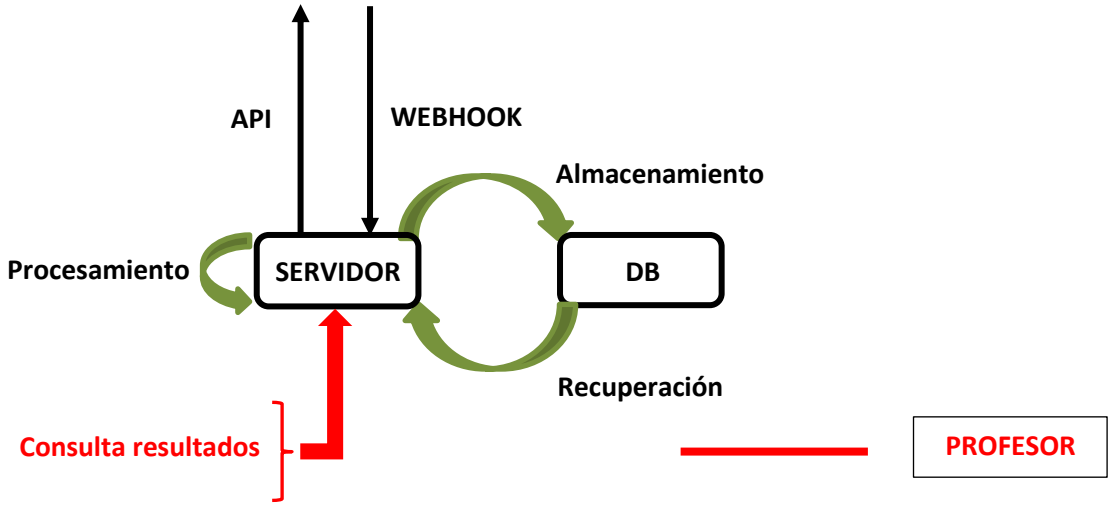

Figura 4. Nivel Servidor en la arquitectura

Como se ha mencionado al explicar la arquitectura de referencia, la función principal de esta máquina es la de comunicarse con GitHub, procesando la información que ésta le envíe a través de los webhooks y realizando distintas acciones de forma automatizada a través de la API de GitHub[6], cumpliendo con la especificación **E.ME.2**.

Finalmente, para cumplir la especificación **E.AS.1**, cierta información con interés para almacenar (como los resultados de las pruebas) debe ser enviada al servidor a través de los webhooks y guardada en una base de datos (que se aloja en el mismo equipo físico del servidor por sencillez, aunque podría llegar a separarse en un futuro para añadir más robustez al sistema) para su posterior consulta, que se realizaría por el profesor a través de algún script o conectándose directamente al servidor.

[6] Para más información acerca del uso de la API y los webhooks de GitHub, ver Anexo IV (sobre la API) y Anexo V (sobre los webhooks).

Vistas todas estas características, el flujo de trabajo entre los niveles de GitHub y Servidor:

1. El profesor **crea una organización** en GitHub e **invita a los alumnos** a medida que se vayan registrando.
2. El alumno **crea su cuenta** en GitHub siguiendo los criterios que se les puedan haber indicado.
3. El profesor **crea un nuevo *assignment*** a través de GitHub Classroom cada vez que los alumnos tengan que resolver un ejercicio práctico independiente de los anteriores, configurando repositorios privados como mínimo. El resto de parámetros quedan a juicio del profesorado. [7]
4. El alumno **acepta el *assignment*** y se le asigna automáticamente un nuevo repositorio.
5. A través del servidor, se **instalan** los webhooks y la aplicación de CI en cada repositorio.
6. El profesor **entrega** a los alumnos los **ficheros de configuración** de la aplicación de CI.
7. El alumno **resuelve** los ejercicios prácticos.
8. Cuando crea el ejercicio terminado, el alumno **sube sus códigos** al repositorio y **ejecuta las pruebas** a través de la aplicación de CI.
9. Los resultados de las pruebas son **enviados al servidor** y se le **notifica** al alumno el resultado de las mismas, preferiblemente a través de elementos propios de GitHub como las *issues*.
10. El profesor **consulta los avances** de sus alumnos.

[7] Para más información sobre este proceso, ver Anexo III.

DESCRIPCIÓN DE FASES Y TAREAS

En este apartado, se describen las distintas fases a completar para el desarrollo del piloto que valide la metodología, siguiendo la arquitectura de referencia presentada en el apartado anterior (por lo que se realiza en el contexto de la EIB). Cabe destacar que se ha decidido no mostrar la planificación seguida a lo largo del desarrollo de este trabajo ya que se considera de poco interés, debido a que ésta consistiría en una descripción lineal de la obtención de información para los apartados vistos hasta el momento.

Para poder realizar el piloto funcional, se ha dividido el trabajo en seis fases y éstas, a su vez, en varias tareas.

A continuación, se muestra una descripción de cada fase y sus tareas relacionadas:

1. **Formación desarrollador:**

   La primera fase del trabajo consiste en formar al equipo de desarrollo en las tecnologías a usar en la parte del servidor que se comunica con GitHub, que es el único desarrollo necesario para la NME-CV. La fase dura 7 días.

   Las tareas a realizar son:

   1.1. API GitHub: Estudiar el uso de la API que proporciona GitHub (3 días).

   1.2. Webhooks: Estudiar el uso de los webhooks que ofrece GitHub (3 días).

   1.3. DB: Repasar conceptos relacionados con las DB (*DataBase*), o sea, con las bases de datos (1 día).

   Estas tareas se realizan en serie. Se llevan a cabo tanto por ingenieros sénior como por júnior.

2. **Diseño del servidor**

   Estudiadas las tecnologías, es el momento de empezar a diseñar las herramientas para poder dotar al servidor de todas las funcionalidades requeridas por el mismo, o sea, comunicarse con GitHub y almacenar los resultados. Este paso se divide en una revisión de la arquitectura propuesta en este trabajo y el diseño detallado del servidor. La fase dura 4 días.

   Las tareas a realizar son:

   2.1. Revisión arquitectura: Revisar la arquitectura propuesta en este trabajo para tener claro todos los conceptos relacionados (1 día).

   2.2. Diseño API GitHub: Diseño detallado de las funciones a realizar a través de la API de GitHub (3 días).

2.3. Diseño webhooks: Diseño detallado de las funciones a realizar a través de los webhooks ofrecidos por GitHub (3 días).

2.4. Diseño DB: Diseño detallado de la forma de la base de datos (2 días).

En este caso, los diseños detallados se realizan en paralelo una vez se haya revisado la arquitectura. Se llevan a cabo por ingenieros sénior.

3. Desarrollo y pruebas unitarias

Una vez diseñado, toca ponerse a trabajar. Además del desarrollo de cada uno de las herramientas del anterior punto, se realizan las pruebas unitarias a cada una de ellas a fin de comprobar su correcto funcionamiento por separado. Esta es la fase más larga de todas, con una duración de 11 días.

Las tareas a realizar son:

3.1. Uso API GitHub: Desarrollar y probar el módulo que haga uso de la API de GitHub (10 días).

3.2. Uso webhooks: Desarrollar y probar el módulo que trabaje con webhooks (10 días).

3.3. Uso DB: Desarrollar y probar la base de datos (5 días).

Todos los desarrollos se realizan en paralelo. Se llevan a cabo por ingenieros júnior.

4. Integración

Una vez finalizado el desarrollo de las herramientas, se procede a integrarlas, o sea, juntarlas y comprobar que el sistema completo funciona. Este es el paso inmediatamente anterior a la demostración de que la NME-CV es válida. La fase dura 4 días.

Las tareas a realizar son:

4.1. API GitHub + Webhooks: Integrar el uso de la API con el de los webhooks (2 días).

4.2. API GitHub + WebHooks + DB: Integrar lo anterior con la base de datos (2 días).

Ambas tareas se realizan en serie. Se llevan a cabo por ingenieros júnior.

5. Validación

Con todo preparado, llega el momento de validar los resultados obtenidos con el fin de conocer si ha sido un éxito o si, por el contrario, es necesario revisar el trabajo realizado hasta el momento. Este es el último paso del desarrollo del piloto como tal. La fase dura 5 días.

Las tareas a realizar son:

5.1. Prototipos: Realizar los prototipos demostrativos que se crean necesarios (3 días).

5.2. Propuesta final: Conocidos todos los resultados hasta la fecha, proponer la forma final que adquiere (1 día).

5.3. Análisis conjunto: Comentar y analizar todo el proceso y organizar los siguientes pasos.

Todas las tareas se realizan en serie. Se llevan a cabo por ingenieros júnior.

6. Formación profesor

Finalmente, una vez está todo listo para el uso de la NME-CV, sólo queda formar a los profesores en su uso. La fase dura 7 días.

Las tareas a realizar son:

6.1. NME-CV: Formar en el uso de la metodología en sí (2 días).

6.2. eGela avanzado: Formar en el uso avanzado de la plataforma eGela para usarla correctamente dentro de la metodología (2 días).

6.3. GitHub + GitHub Classroom: Formar en el uso de GitHub y GitHub Classroom para usarlos correctamente dentro de la metodología (2 días).

Todas las tareas se realizan en serie. Se llevan a cabo por los ingenieros sénior.

Por tanto, conocidos los días dedicados a cada fase, se aprecia que el proyecto requiere de 38 días laborables, lo que se traduce en 7 semanas y 3 días de trabajo si se estiman semanas de 5 días hábiles.

DIAGRAMA DE GANTT

A continuación, se muestra el diagrama de Gantt con la organización para el desarrollo del piloto funcional junto con una tabla resumen de las tareas y su duración. Debido a fines prácticos, se ha estimado el comienzo del proceso a fecha de 3 de junio de 2019, pensando en aprovechar las fechas de verano en las que disminuye las horas de clase a impartir por los profesores. Aun así, no se considera una obligación empezar en dicho momento, pudiendo adelantarlo o atrasarlo el tiempo que se considere oportuno.

Nombre	Trab...
FORMACIÓN DESARROLLADOR	**7d**
API GITHUB	3d
WEBHOOKS	3d
DB	1d
DISEÑO DEL SERVIDOR	**9d**
REVISIÓN ARQUITECTURA	1d
DISEÑO API GITHUB	3d
DISEÑO WEBHOOKS	3d
DISEÑO DB	2d
DESARROLLO Y PRUEBAS UNITARI/	**25d**
USO API GITHUB	10d
USO WEBHOOKS	10d
USO DB	5d
INTEGRACIÓN	**4d**
API GITHUB + WEBHOOKS	2d
API GITHUB + WEBHOOKS + DB	2d
VALIDACIÓN	**5d**
PROTOTIPOS	3d
PROPUESTA FINAL	1d
ANÁLISIS CONJUNTO	1d
FORMACIÓN PROFESOR	**7d**
NME-CV	2d
eGELA AVANZADO	2d
GITHUB + GITHUB CLASSROOM	3d

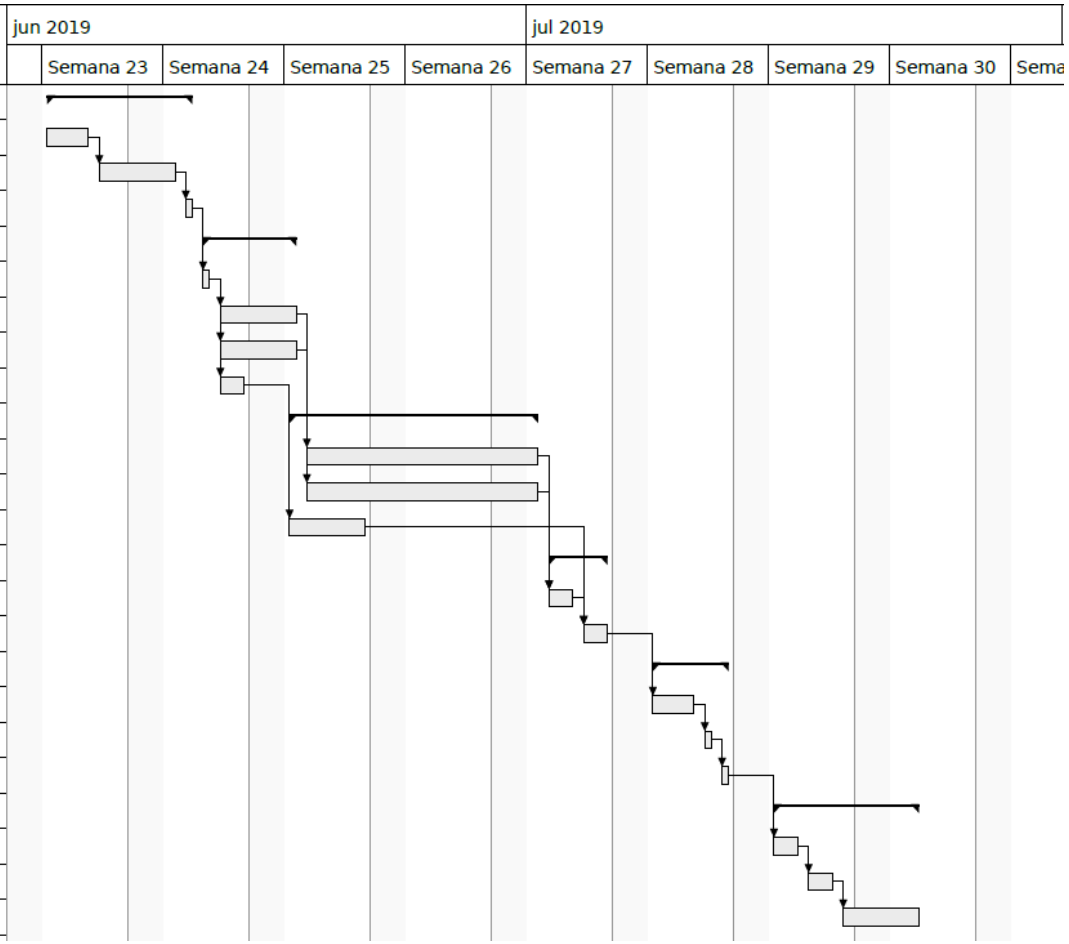

Figura 5. Diagrama de Gantt

WBS	NOMBRE	INICIO	FIN	DURACIÓN	TRABAJO
1	**FORMACIÓN DESARROLLADOR**	**Jun 3**	**Jun 11**	**7d**	**7d**
1.1	API GITHUB	Jun 3	Jun 5	3d	3d
1.2	WEBHOOKS	Jun 6	Jun 10	3d	3d
1.3	DB	Jun 11	Jun 11	1d	1d
2	**DISEÑO DEL SERVIDOR**	**Jun 12**	**Jun 17**	**4d**	**9d**
2.1	REVISIÓN ARQUITECTURA	Jun 12	Jun 12	1d	1d
2.2	DISEÑO API GITHUB	Jun 13	Jun 17	3d	3d
2.3	DISEÑO WEBHOOKS	Jun 13	Jun 17	3d	3d
2.4	DISEÑO DB	Jun 13	Jun 14	2d	2d
3	**DESARROLLO Y PRUEBAS UNITARIAS**	**Jun 17**	**Jul 1**	**11d**	**25d**
3.1	USO API GITHUB	Jun 18	Jul 1	10d	10d
3.2	USO WEBHOOKS	Jun 18	Jul 1	10d	10d
3.3	USO DB	Jun 17	Jun 21	5d	5d
4	**INTEGRACIÓN**	**Jul 2**	**Jul 5**	**4d**	**4d**
4.1	API GITHUB + WEBHOOKS	Jul 2	Jul 3	2d	2d
4.2	API GITHUB + WEBHOOKS + DB	Jul 4	Jul 5	2d	2d
5	**VALIDACIÓN**	**Jul 8**	**Jul 12**	**5d**	**5d**
5.1	PROTOTIPOS	Jul 8	Jul 10	3d	3d
5.2	PROPUESTA FINAL	Jul 11	Jul 11	1d	1d
5.3	ANÁLISIS CONJUNTO	Jul 12	Jul 12	1d	1d
6	**FORMACIÓN PROFESOR**	**Jul 15**	**Jul 23**	**7d**	**7d**
6.1	NME-CV	Jul 15	Jul 16	2d	2d
6.2	eGELA AVANZADO	Jul 17	Jul 18	2d	2d
6.3	GITHUB + GITHUB CLASSROOM	Jul 19	Jul 23	3d	3d

Tabla 4. Resumen de tareas

DESCRIPCIÓN DEL PRESUPUESTO

En este apartado, se proporcionan dos presupuestos diferentes: el primero de ellos, se corresponde con una aproximación al coste que supone la realización del estudio, mientras que el segundo tiene en cuenta los gastos asociados al desarrollo del piloto, también de forma aproximada.

En cualquier caso, se han tenido en cuenta los siguientes datos:

- **Horas diarias**: 8 h/día
- **Horas anuales**: 1750 h/año
- **Precio ingeniero sénior**: 80 €/h
- **Precio ingeniero júnior**: 30 €/h
- **Precio servidor web dedicado**: 120€
- **Precio de equipos**: 1500€
- **Número de equipos**: 3
- **Vida útil de equipos**: 5 años
- **Material fungible**: 200€
- **Costes indirectos (para el desarrollo)**: 7%
- **Partida de imprevistos (para el desarrollo)**: 10%
- **Licencia software**: 0€[8]

Presupuesto del estudio

En este caso, la única partida existente es la de horas internas, ya que no se ha hecho uso de ningún material adicional atribuible al proyecto. Por tanto, en este caso, el presupuesto total equivale a dicha partida.

Horas Internas

Concepto	*Coste Unitario*	*Nº Unidades*	*CosteTotal*
Ingeniero Senior	80 €/h	48 h	3.840 €
Ingeniero Junior	30 €/h	120 h	3.600 €
		Subtotal	***7.440 €***

Tabla 5. Partida de Horas Internas y Presupuesto Total del estudio

[8] Se presupone la adquisición de repositorios privados de GitHub gratuitos según se explica en el Anexo III.

Presupuesto de la implantación

Para la implantación, se tienen en cuenta los puntos anteriores, o sea, los gastos que supone en el contexto de la EIB y siguiendo la planificación dispuesta en los dos apartados anteriores.

Horas Internas

Concepto	***Coste Unitario***	***Nº Unidades***	***CosteTotal***
Ingeniero Senior	80 €/h	184 h	14.720,00 €
Ingeniero Junior	30 €/h	328 h	9.840,00 €
		Subtotal	***24.560,00 €***

Tabla 6. Partida de Horas Internas de la implantación

Amortizaciones

Concepto	***Coste Unitario***	***Nº Unidades***	***CosteTotal***
Equipo 1	0,171 €/h	160 h	27,43 €
Equipo 2	0,171 €/h	104 h	17,83 €
Equipo 3	0,171 €/h	56 h	9,60 €
		Subtotal	***54,86 €***

Tabla 7. Partida de Amortizaciones de la implantación

Costes Fijos

Concepto	**Coste Unitario**	**Nº Unidades**	**CosteTotal**
Material Fungible	200 €	1	200,00 €
Servidor Web dedicado	120 €	1	120,00 €
		Subtotal	***320,00 €***

Tabla 8. Partida de Costes Fijos de la implantación

Presupuesto Total

Concepto	**CosteTotal**
Horas Internas	24.560,00 €
Amortizaciones	54,86 €
Costes Fijos	320,00 €
Subtotal 1	***24.934,86 €***
Gastos indirectos (7%)	1.745,44 €
Subtotal 2	***26.680,30 €***
Imprevistos (10%)	2.668,03 €
TOTAL	***29.348,33 €***

Tabla 9. Presupuesto total de la implantación

Teniendo en cuenta que cada matrícula de primer año en el grado en Ingeniería en Tecnología de Telecomunicación de la EIB son 1.151,4€, se concluye que se requieren algo más de 25

matrículas completas para recuperar el dinero invertido. Al considerar estos números de matriculación algo elevados, la principal fuente de recuperación del dinero sería la descrita en el beneficio **B.EC.2**.

CONCLUSIONES

No es que la enseñanza de programación en la EIB ni en el resto de cursos no sea adecuada ni correcta, pero, como todo al fin y al cabo, es mejorable. Actualmente, en los entornos de desarrollo profesionales se hacen uso de distintas tecnologías como el control de versiones y la integración continua que podrían ser realmente útiles en la adquisición de competencias de programación durante la formación de los alumnos.

Aprovechándolas, se podrían matar dos pájaros de un tiro: por un lado, se podría simplificar una metodología exigente en cuanto a trabajo continuo como es la propugnada por el Plan Bolonia y, por otro lado, se podría mejorar la formación de alumnos de programación a través de tecnologías de las que probablemente acaben haciendo uso.

Mediante la NME-CV, se puede conseguir todo esto de forma sencilla, usando software gratuito y bien documentado, lo que se traduce en grandes facilidades tanto técnicas como económicas a la hora de implementarlo. Además, no exige un gran cambio en las metodologías típicas que usan LMS, sino que se limita a construir sobre esa base, añadiendo simplemente las funcionalidades de las que carece.

Este, sin embargo, sólo es el estudio, el primer paso. Queda mucho trabajo por delante antes de poder implantarlo siquiera en la EIB. Pero antes de todo eso, hay que pasar a la siguiente fase: el desarrollo de un piloto funcional que valide este trabajo.

BIBLIOGRAFÍA

https://developer.github.com/

https://education.github.com/

https://classroom.github.com/videos

https://education.github.community/

https://education.github.community/t/my-guides-to-using-github-classroom-for-teachers-and-students/15717

https://education.github.community/t/assigning-and-submitting-with-github-classroom/23583

https://docs.moodle.org/35/en/Main_page

https://ubc-mds.github.io/2017-08-24-teaching-with-github/

https://en.wikipedia.org/wiki/Learning_management_system

https://www.computerweekly.com/blog/Open-Source-Insider/What-is-a-software-forge

https://aws.amazon.com/es/devops/continuous-integration/

https://git-scm.com/book/es/v1/Empezando

https://chamilo.org/es/

http://uupinfo.org/research/working/bradford.pdf

https://docs.travis-ci.com/user/getting-started/

ANEXO I: GIT Y GITHUB

Git es un DVCS (*Distributed Version Control System*), lo que quiere decir que todos los clientes disponen de una copia de la base de datos de versiones usada en los VCS. Esto mejora notablemente la robustez del sistema, ya que ésta puede ser consultada en cualquier momento, sin necesidad de conectarse a ningún equipo externo, lo que permite trabajar de forma local.

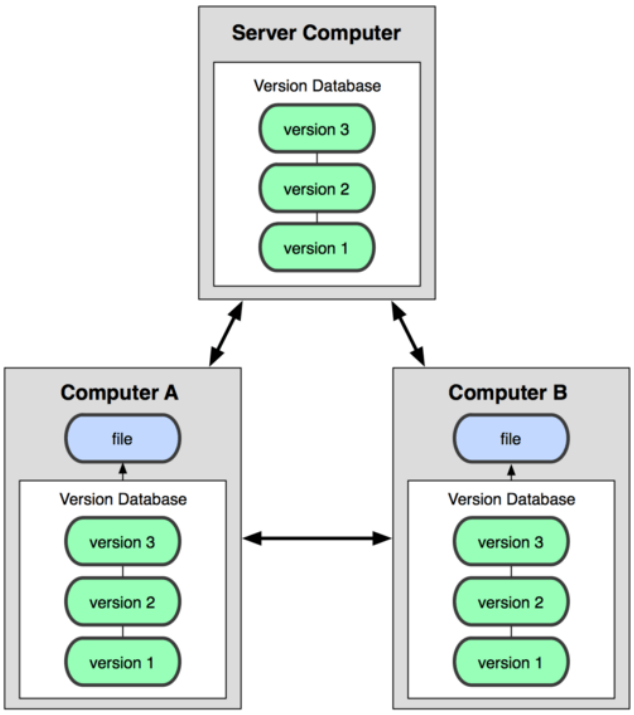

Figura 6. Esquema de un DVCS.

Otra de las características de Git es que, para registrar los cambios a lo largo del tiempo, almacena instantáneas (*snapshots* en inglés) en lugar de diferencias, que es lo que hacen la mayoría de herramientas. Cuando se confirma el cambio en el contenido de un fichero, se realiza una *"foto"* de todos los ficheros que conforman el proyecto y se guarda una referencia a la instantánea, aunque, por razones de eficiencia, no se vuelven a almacenar los ficheros que no hayan sido modificados, sino que, en su lugar, se guarda un enlace al mismo.

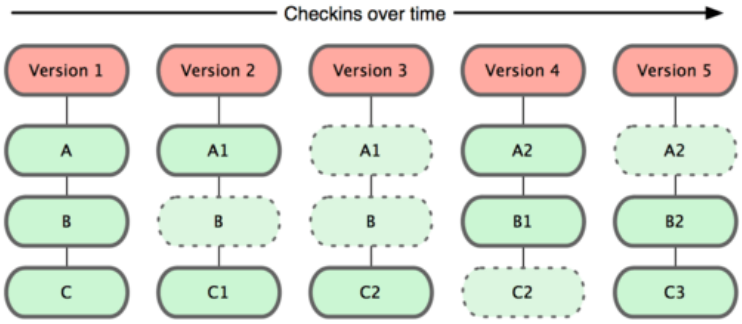

Figura 7. Esquema de registro de cambios por instantáneas

Además, Git garantiza la integridad de la información mediante un checksum asociado a cada versión del proyecto, por lo que es capaz de detectar cambios en cualquier fichero, protegiendo el sistema, por ejemplo, frente a corrupciones de datos durante el envío de información.

Años más tarde, apareció una nueva herramienta llamada GitHub, una forja que usa Git como base de su sistema de control de versiones. A través de su interfaz web, GitHub permite realizar las mismas funciones que Git (aunque se puede acceder a sus servicios mediante el cliente de consola de Git), alojando los proyectos en repositorios tanto públicos como privados, pudiendo crear los del primer tipo de forma ilimitada y gratuita, lo que permite distribuir código de forma rápida.

GitHub siempre ha apostado por añadir un componente social al desarrollo de los programas, facilitando la comunicación con los desarrolladores por parte tanto de usuarios del programa como de otros desarrolladores ajenos al proyecto que tratan de aportar sus ideas. Para ello, GitHub ofrece:

- ***Issues*** que permiten abrir un canal de comunicación directa con los desarrolladores alrededor de un tema, permitiendo leer y participar a terceros.
- Las funciones ***Fork*** e ***Import*** que se usan para clonar el contenido de un repositorio propio o ajeno en uno propio.
- ***Pull request*** para solicitar la integración de los cambios realizados en un repositorio a otro. Éstas también abren un canal de comunicación.
- Además, dentro de un repositorio puede haber varios **colaboradores** que puedan editar el contenido del repositorio.

El éxito de GitHub ha sido absolutamente abrumador. Y es que se ha convertido en la mayor forja del mundo, llegando en julio de 2018 a los 28 millones de usuarios registrados y 57 millones de repositorios alojados. Tal es la popularidad de la plataforma que, recientemente, el gigante Microsoft la ha adquirido por 7.500.000.000$.

ANEXO II: MOODLE

Moodle es una aplicación LMS con licencia GPL y escrita en PHP. Constituye una de las soluciones más populares en el mercado, alcanzando cifras de aproximadamente 131 millones de usuarios y presencia en 234 países.

Sitios registrados	**102.951**
Países	234
Cursos	15.033.858
Usuarios	130.895.050
Inscripciones	570.606.380
Posts en foros	281.616.458
Recursos	132.949.207
Preguntas de tests	834.866.714

Tabla 10. Datos de uso de Moodle (07/18). Fuente: https://moodle.net/stats/

Teniendo un servidor web, una base de datos y PHP instalado, cualquier entidad puede hacer uso del programa Moodle. Desde ese momento, los administradores del sistema son capaces de crear aulas virtuales denominadas cursos a los que invitar a distintos usuarios, asignándoles los roles de profesor y alumno.

Los del primer tipo son los que tienen el control completo sobre el curso, pudiendo editar el contenido del mismo. Normalmente, esto se realiza a través de la adición de distintos tipos de archivos, que pueden contener explicaciones teóricas, enunciados de prácticas, código inicial… Aun así, se puede complementar con una serie de actividades como las que se muestran a continuación:

- **Tareas** a las que los alumnos deban subir archivos para su posterior evaluación. Además, se puede configurar una fecha límite tras la cual se cerrará la opción de subida.
- **Talleres** en los que los alumnos puedan acceder al trabajo de sus compañeros para evaluarlos.
- Pruebas tipo **test**.
- **Encuestas**.
- **Foros** para permitir la comunicación entre todos los usuarios.
- **Wikis**.

Todos estos recursos, además, pueden ser ocultados por cualquier profesor, de modo que pueda tener todos los recursos subidos y mostrarlos poco a poco a medida que avance el curso, para después ocultarlos otra vez antes de que empiece el siguiente año académico, lo que permite no repetir el mismo trabajo de subida año tras año.

ANEXO III: GITHUB CLASSROOM

GitHub tiene una sección llamada GitHub Education en el que propone usar la plataforma como medio para la enseñanza. Para ello, proporciona una serie de ofertas tanto para alumnos como para profesores en una serie de herramientas y tecnologías.

Además, también propone el uso de un servicio desarrollado por ellos: GitHub Classroom. Éste consiste en usar una organización ya existente para crear lo que se denomina como *classroom*, en la que participan profesores (como administradores) y alumnos, estando todos ellos dentro de la organización.

Llegados a este punto, el profesor puede empezar a mandar trabajo a sus alumnos a través de los llamados *assignments*, que no son más que tareas a resolver. El funcionamiento es sencillo:

- Una vez se crean y configuran, se genera un enlace de invitación que el profesor debe poner a disposición de los alumnos.
- Cuando los alumnos lo reciben, deben entrar y aceptar la tarea.
- En ese momento, se crea un repositorio dentro de la organización, propiedad del profesor, en el que el alumno puede entrar en calidad de colaborador.

A partir de aquí, GitHub Classroom deja de actuar, pasando a funcionar todo a través de GitHub normal.

Como se ha mencionado antes, es necesario configurar los *assignments*. Los parámetros son:

- **Nombre** del *assignment*.
- **Prefijo** para identificar la tarea. El nombre de cada repositorio se forma con este prefijo y el nombre de usuario del alumno.
- Selección de repositorios **públicos o privados**.
- Posibilidad de dar **permisos de administración** a los alumnos dentro de sus repositorios.
- Creación de un **enlace de invitación** (por el momento, debe permanecer marcado).
- Repositorio desde el que **importar** el código (opcional). Si se especifica alguno, cada uno de los repositorios del *assigment* se crea importando el contenido de éste.
- **Fecha límite** para la entrega.

Para poder mantener un mejor control sobre los alumnos, el profesor puede gestionar la relación entre las cuentas de los alumnos y la identidad real. Para ello, en la pestaña de configuración de *Rooster Management* debe subir un fichero CSV que contenga una lista con las identidades de los alumnos (también puede introducirlas una por una) para que, en

el momento de aceptar su primer *assignment*, el alumno escoja el elemento de la lista que le haga referencia a él, quedando relacionado con la cuenta de usuario de GitHub.

Cabe destacar además que existe un problema con la seguridad en GitHub Classroom, y es que cualquier persona con el enlace puede aceptar el *assigment*, lo que puede ser una fuente de ataques a los sistemas que lo implementen. Aun así, GitHub Classroom se sigue usando en varios centros educativos, por lo que no parece que existan muchos problemas a la hora de aplicarlo.

Económicamente hablando, GitHub Classroom, en principio, hace necesario disponer de algún plan de pago que otorgue la posibilidad de crear repositorios privados de manera ilimitada, ya que son más interesantes para aplicar en el ámbito académico. Por suerte, el plan de equipo (que dispone de esta ventaja), es gratuito tanto para profesores verificados como para facultades académicas que pretendan usar GitHub para la investigación académica sin ánimo de lucro.

ANEXO IV: GITHUB API

GitHub dispone de una API accesible a través del enlace *https://api.github.com* que va por su versión 4. Existe un punto de conflicto, y es que, actualmente, conviven las versiones 3 y 4, ya que API v3 usa REST (*REpresentational State Transfer*) y API v4, por el contrario, GraphQL (*Graph Query Language*). A fin de tener una mejor visión de la situación actual, se procede a explicar un poco de ambas.

API REST v3

GitHub dispone de una API REST para el uso de diversas funciones internas.

Toda la información se transporta en JSON (*JavaScript Object Notation*). En éste, los campos en blanco se envían como *null*. Se distinguen además dos tipos de búsquedas: las resumidas y las detalladas:

- Las primeras se dan cuando se solicita una lista de recursos. En estos casos, se devuelve información sobre cada elemento de la lista, sólo que, con el fin de mejorar el rendimiento, se omiten algunos datos computacionalmente costosos. Para obtenerlos, es necesario realizar una búsqueda detallada del mismo.

- Las segundas se dan cuando se solicita un único recurso. En este caso, se dispone de esos datos omitidos en las resumidas, aunque el usuario necesita autenticarse para acceder a ciertos datos protegidos (no se puede acceder a los datos de un repositorio privado a no ser que se tenga permiso dentro de dicho repositorio, por ejemplo).

Para implementar esta API en los distintos lenguajes de programación existen una serie de librerías.[9] De todos modos, para los ejemplos, simplemente, se hace uso del programa cURL (a través de su cliente de consola) para llamar al servicio REST, de forma que se comprenda mejor la acción subyacente.

Para poder conocer las funciones disponibles y cómo llamarlas, hay que acudir a la documentación de la API.[10]

Como primer ejemplo, se estudia la forma de llamar a la función *zen*, que lo único que hace es devolver una frase correspondiente a las filosofías de diseño del equipo de desarrollo (lo que puede ser útil, por ejemplo, para saber si la API está operativa):

```
curl https://api.github.com/zen
```

[9] Disponibles en https://developer.github.com/v3/libraries/.
[10] Disponible en el siguiente enlace: https://developer.github.com/v3/.

Como se puede apreciar, el uso básico es el típico de los servicios REST: lanzar un mensaje HTTP del tipo que sea contra una cierta URL con un dominio fijo y un path variable que indique la operación a realizar.

A la hora de ejecutar una función el path hacia la misma puede ser constante (como el caso anterior, en el que la URL siempre toma la misma forma) o puede ser variable, dependiendo de si alguno de los fragmentos del path es en verdad un parámetro. Esto último es habitual, por ejemplo, en las funciones relacionadas con repositorios. La función siempre es la misma, pero el nombre del repositorio es distinto en cada caso, por lo que hay que especificarlo. En la documentación, estos campos aparecen con dos puntos (*:*) por delante. Por ejemplo, para listar los repositorios pertenecientes a un usuario, el path a añadir según la documentación es:

```
/users/:username/repos
```

Por tanto, para recuperar los repositorios del usuario *alice*:

```
/user/alice/repos
```

Conocido el uso básico, se explican a continuación algunas de las características de la API.

Una de las funciones clave de la API es la autenticación, que limita el acceso a la información. Por ejemplo, si uno busca información sobre su propio usuario:

- En caso de no autenticarse, sólo se obtiene la **información pública**.
- En caso de autenticarse, también obtiene **información confidencial**.

Cabe destacar que aquí también entra en juego la autorización, por lo que si un usuario, por autenticado que esté, solicita información acerca de otro, sólo se le concede la pública.

La autenticación también es importante de cara a realizar peticiones. Un usuario sin autenticar sólo puede realizar 60 peticiones a la API, mientras que el límite de un usuario autenticado se encuentra en las 5000 peticiones. Para conocer el estado de este límite, se puede usar los campos de la cabecera HTTP *X-RateLimit-Limit*, *X-RateLimit-Remaining* y *X-RateLimit-Reset* para conocer el límite, las peticiones restantes y el tiempo hasta restaurarlas (todos los campos de la cabecera que empiecen por *X-* son propios de la API).

Para poder realizar el proceso de autenticación, existen varias alternativas. La primera de ellas es la básica, que consiste en añadir la opción -u del comando curl:

```
curl -u [usuario] https…
```

Siendo *[usuario]* el nombre de usuario con el que se pretende identificar. Después de ejecutar el comando solicita la entrada de la contraseña a través de la terminal.

Esta opción no es válida si el usuario que se autentica tiene activada la autenticación en dos factores, en cuyo caso, la ejecución de cualquier función siempre devuelve un mensaje de error *401 Unauthorized*. Para poder evitar esta limitación, es necesario pasar al segundo método.

La segunda forma de autenticarse es mediante el uso de tokens OAuth (*Open Authorization*). En líneas generales, este método consiste en generar en el usuario de GitHub un token de autenticación, que no deja de ser una cadena de texto generada automáticamente que representa a un usuario, que podrá usar para autenticarse. Sin embargo, este token no dispone de ninguna autorización, por lo que, se le deben añadir una serie de permisos para poder ejecutar funciones de la API.[11]

Para poder usar este método, es necesario añadir el siguiente campo a la cabecera HTTP-Request:

```
Authorization token [token]
```

Siendo *[token]* el token OAuth generado previamente.

Para algunas funciones, también puede ser necesario solicitar un cambio en el formato de devolución. Para ello, hay que añadir la siguiente cabecera al mensaje HTTP-Request:

```
Accept: [formato]
```

Siendo *[formato]* el nuevo formato en el que se quiere recuperar la información.

Finalmente, se adjunta un listado con algunas de las funciones que se pueden realizar en la APIv3, ordenadas según ámbito de uso:

- Repositorios:
  - Listar repositorios de un usuario
  - Crear un repositorio
  - Borrar un repositorio
- Commits:
  - Listar commits
  - Recuperar un único commit
  - Comparar dos commits
- Webhooks:
  - Listar webhooks
  - Crear un webhook
  - Borrar un webhook
- Issues:
  - Listar issues propios
  - Listar issues de un repositorio
  - Recuperar un único issue
  - Crear un issue

[11] Para más información, ver Anexo VI.

GraphQL API v4

La versión más actualizada de la API de GitHub usa tecnología GraphQL desplegada sobre HTTP. Esto supone que sigue siendo accesible a través de la red (usando el comando curl, por ejemplo). Sin embargo, existen ciertas diferencias con respecto al caso anterior.

- Mientras que en REST había que usar una URL distinta para cada función, en esta nueva versión sólo es necesario conocer una. Concretamente, *https://api.github.com/graphql.*
- Pese a usar HTTP, GraphQL exige que se le pase un código con las operaciones a realizar, por lo que todos los mensajes son del tipo POST. Este fichero a subir es un JSON un tanto *sui generis*, pues sólo tiene un parámetro denominado *query* al que se le pasa el código de búsqueda JSON en una única línea. Por ejemplo:

```
{ "query": "query {viewer { login }}" }
```

- El formato de devolución de datos sigue siendo JSON, aunque puede llegar a devolverlo todo en una línea, dificultando la comprensión de los datos.
- Debido a la propia naturaleza de la tecnología, en un único JSON, se pueden enviar acciones para realizar en una única petición varias funciones. Esto supone que el límite de 5000 peticiones deja de ser válido, ya que se podrían realizar el equivalente a más de 5000 peticiones en una sola. Esto supone una ventaja computacional para el usuario, pero para el servidor sigue siendo igual de costoso. Por ello, se define un máximo de 5000 puntos/hora. Para saber cuántos puntos se gastan en cada petición, se puede explícitamente el paso de dicho gasto o, mejor aún, calcular cuánto costaría.[12]

Existen además una serie de limitaciones:

- Todas las conexiones deben usar el parámetro *first* y *last* para devolver sólo un número limitado de recursos.
- Los parámetros *first* y *last* deben tener un entero entre 1 y 100 como valor.
- Cada consulta no puede solicitar más de 500.000 nodos. [13]

[12] Para más información: https://developer.github.com/v4/guides/resource-limitations/#node-limit.
[13] Ver nota 10.

Al igual que en la APIv3, es necesario recurrir a la documentación para conocer las funciones que se pueden realizar.[14] Se procede a listar algunas funciones interesantes realizables a través de la APIv4:

- Recuperar información repositorios:
- Recuperar información sobre *issues*
- Recuperar información sobre usuarios
- Crear proyectos
- Crear columna en proyecto
- Crear carta en columna

Vistas las distintas versiones de la API, se puede concluir que, aunque sí pueden encontrarse muchos tipos distintos de consultas, lo cierto es que no se puede decir lo mismo del limitado número de mutaciones disponibles en la v4.

[14] La documentación de la API se encuentra en el enlace https://developer.github.com/v4/.

ANEXO V: WEBHOOKS EN GITHUB

Como ya se ha mencionado a lo largo del trabajo, en GitHub existe la posibilidad de crear webhooks para informar de ciertos eventos que ocurran dentro de un repositorio u organización. Se pueden instalar 20 webhooks por cada evento en cada repositorio.[15]

Caben destacar varias cosas relacionadas con los eventos:

- El evento *push* tiene un payload más detallado cuando se usa en webhooks.
- Todos los eventos tienen, además, los campos *sender* (usuario que disparó el evento), *repository* u *organization* (repositorio u organización en la que ocurrió el evento) y, en el caso de las GitHub Apps, *installation* (instalación con la que se relaciona el evento).
- Los payloads no pueden ocupar más de 5MB. Si un payload ocupase más, el webhook no se activaría.
- Además, los payloads contienen algunas cabeceras especiales:
    - *X-GitHub-Event*: nombre del evento disparado.
    - *X-GitHub-Delivery*: GUID del emisor.
    - *X-Hub-Signature*: seguridad adicional en caso de que se configure el webhook como *secret*.
- Existe también un evento especial llamado *ping* que se envía nada más crear un webhook para comprobar que se ha creado correctamente.

Para crear un webhook mediante la interfaz gráfica (sin usar la APIv3) hay que seguir los siguientes pasos:

1. Entrar en la **configuración** del repositorio o de la organización.
2. Entrar en la **pestaña *Webhooks***.
3. Hacer click en el **botón *Add Webhook***.
4. **Configurar** el webhook según los parámetros
    4.1. ***Payload URL***: **URL a la que enviar** los mensajes generados por el webhook.
    4.2. ***Content type***: **Formato** en el que recibir la información.

[15] La lista de eventos a los que se puede suscribir está en el siguiente enlace: https://developer.github.com/v3/activity/events/types/.

4.3. ***Secret***: Configuración de un token para **añadir seguridad** el proceso de envío de webhooks.

4.4. **Eventos** a los que subscribirse.

La información introducida para configurar el webhook puede ser editada en la pestaña *Webhooks* mencionada en el paso 2.

ANEXO VI: CREAR TOKEN OAUTH EN GITHUB

En el Anexo III se ha visto que para usar la API es recomendable disponer de un token OAuth para poder identificarse. Éstos se asocian a un usuario (no organización) y es necesario otorgarles unos ciertos permisos agrupados bajo los llamados *scopes*. Esto quiere decir que un token con el *scope notification* tiene autorización para ejecutar ciertas funciones predefinidas alrededor de las notificaciones, no pudiendo cambiar quitar ninguno de los permisos del conjunto. Aun así, la mayoría de *scopes* disponen de subagrupaciones para dotar de cierta flexibilidad.

Para crear un token OAuth, hay que seguir los siguientes pasos:

1. Entrar en la pestaña de **configuración** del usuario.
2. Entrar en la **pestaña *Developer Settings*** y una vez allí en ***Personal access token***.
3. Hacer click en ***Generate new token***.
4. **Configurar** el token
   4.1. ***Token description***: identificador del token.
   4.2. ***Select scopes***: configurar los *scopes* del token.
5. Una vez creado, el token se mostrará junto con un mensaje que indica que es necesario **apuntar el token** ya que **no volverá a mostrarse**, por lo que es necesario guardarlo de la manera deseada.

Al igual que ocurría con los webhooks, la información relativa al token puede editarse en cualquier momento volviendo a la pestaña correspondiente.