UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

Asistente de prácticas ágiles para repositorios en GitHub

Presentado por Lucas Olmedo Díez
en Universidad de Burgos — Junio de 2025
Tutor: Carlos López Nozal

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

D. Carlos López Nozal, profesor del departamento de nombre departamento, área de nombre área.

Expone:

Que el alumno D. Lucas Olmedo Díez, con DNI 71306075A, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado título de TFG.

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, Junio 2025

Vº. Bº. del Tutor:	Vº. Bº. del co-tutor:
D. Carlos López Nozal	D. nombre co-tutor

Resumen

Las prácticas ágiles han demostrado ser fundamentales en el desarrollo de software, ya que promueven la entrega iterativa, la mejora continua y una mejor colaboración entre los desarrolladores. Su aplicación es observable en los repositorios open source de plataformas como GitHub, que facilitan la gestión del código, la documentación y la colaboración entre desarrolladores. En el contexto académico, GitHub es ampliamente utilizado por los estudiantes para la gestión de sus Trabajos de Fin de Grado (TFG). Sin embargo, muchos enfrentan dificultades para aplicar correctamente las prácticas ágiles en sus repositorios, lo que puede afectar la calidad y eficiencia de su desarrollo. El objetivo de este trabajo es ayudar a los estudiantes a aplicar prácticas ágiles en sus repositorios de GitHub mediante una aplicación web que evalúa su grado de adopción. Para ello, la aplicación analiza el uso del repositorio según consultas y reglas basadas en prácticas ágiles, comparando los resultados con valores de referencia fijos o con repositorios seleccionados como modelo. De esta manera, la herramienta facilita la identificación de áreas de mejora y fomenta la adopción de buenas prácticas en la gestión y desarrollo de proyectos académicos en GitHub. La aplicación web está escrita en Node js y Angular y se puede acceder a un despliegue (añadir url) y a su código (añadir url).

Descriptores

Métricas de calidad, proceso de desarrollo de software, metodologías ágiles, integración continua, documentación continua, GitHub, análisis de repositorios, aplicaciones web.

Abstract

Agile practices have proven to be essential for software development. This is because they promote iterative delivery, continuous improvement, and better collaboration between developers. Its usage is visible in open source repositories from platforms like GitHub, which make code administration, documentation, and collaboration between developers much easier. In the academic context, GitHub is heavily used by students for their TFG's administration. However, many of them face difficulties to apply correctly agile methodology in their projects, which can affect their developments' quality and efficiency. The objective of this project is to help students apply agile practices in their GitHub repositories by a web app that evaluates the usage of agile methodology. To achieve that, the app analyzes the use of the repository using queries and rules based on agile practices, comparing results with fixed reference values or repositories selected as a comparison source. This way, the tool helps the user identify areas of improvement, and encourages the user adopting agile practices in their academic GitHub's projects' administration and development. The web app is written in Node js and Angular, and can be acceded to its deployment (URL) and its code (URL).

Keywords

Evolution metrics, software developing process, agile methodology, continuous integration and documentation, GitHub, repository analysis, web applications.

Índice general

Índice de figuras

Índice de tablas

1. Introducción

El desarrollo de software es un proceso complejo que involucra múltiples factores técnicos y organizativos. En el ámbito técnico, es fundamental garantizar el cumplimiento de los requisitos funcionales y no funcionales, tales como mantenibilidad, escalabilidad, eficiencia y calidad del código[6]. A nivel organizativo, la gestión efectiva de equipos de desarrollo, la planificación de tareas y la optimización del tiempo y recursos son aspectos clave para el éxito de un proyecto. Para abordar esta complejidad, se han desarrollado metodologías y herramientas que facilitan la gestión del proceso de desarrollo, promoviendo la colaboración y la mejora continua.

Las metodologías ágiles, como Scrum[11], Kanban[10] y eXtreme Programming[2], han demostrado ser eficaces para gestionar proyectos de software de manera flexible e iterativa. Estas metodologías enfatizan la entrega incremental de valor, la integración continua, la colaboración entre equipos y la adaptación constante a los cambios en los requisitos del proyecto. En este contexto, la gestión de repositorios de código, como GitHub y GitLab, desempeña un papel crucial al proporcionar funcionalidades avanzadas para el control de versiones, la gestión de Issues, la revisión de código y la automatización de flujos de integración y despliegue continuo.

Sin embargo, la gran cantidad de datos generados en los repositorios de software dificulta la evaluación de la calidad del desarrollo y la documentación de este, así como la identificación de áreas de mejora.

Diversos estudios han explorado la evaluación y aplicación de buenas prácticas ágiles en entornos de desarrollo de software. Por ejemplo, el trabajo de Chen, Chen y Hsueh (2024)[3] analiza cómo la integración de metodologías ágiles en la enseñanza del desarrollo de software mejora la organización y gestión de proyectos a través del uso de plataformas como GitHub. Su

estudio destaca la importancia de la gestión de Issues y Commits como indicadores clave del rendimiento de los desarrolladores, así como el papel de las herramientas automatizadas en la evaluación del trabajo en equipo y la documentación del código. Además, enfatizan que el aprendizaje basado en proyectos (Project-Based Learning, PBL) y la incorporación de prácticas ágiles permiten mejorar la capacidad de los estudiantes para abordar problemas complejos y adaptarse a entornos cambiantes. Estos hallazgos refuerzan la necesidad de implementar sistemas que analicen y promuevan el uso de buenas prácticas en repositorios de software, alineándose con el enfoque propuesto en este proyecto para mejorar la calidad del desarrollo mediante métricas objetivas y recomendaciones basadas en metodologías ágiles.

Además, investigaciones recientes han demostrado el impacto positivo de GitHub en la enseñanza de metodologías ágiles y en el desarrollo colaborativo de software. Raibulet y Arcelli Fontana (2018) [9] describen una experiencia en la que estudiantes de ingeniería de software participaron en proyectos colaborativos utilizando GitHub como plataforma de desarrollo. En este estudio, los estudiantes trabajaron en equipos aplicando principios ágiles y emplearon herramientas como SonarQube para evaluar la calidad del software. Los resultados evidenciaron que el uso de GitHub no solo mejoró la gestión del código y la colaboración entre los participantes, sino que también facilitó la integración de buenas prácticas de desarrollo en un entorno educativo. Estos hallazgos refuerzan la relevancia y necesidad de proporcionar herramientas que analicen y promuevan el uso de metodologías ágiles en proyectos de software, alineándose con los objetivos de este TFG.

En respuesta a esta necesidad, este TFG propone el desarrollo de una aplicación web que analiza la información de un repositorio de GitHub y, mediante la aplicación de buenas prácticas ágiles y métricas de calidad, ofrece asistencia a los desarrolladores para mejorar su proceso de trabajo.

El recurso bibliográfico Subway Map to Agile Practices [1], desarrollado por Agile Alliance, recopila un conjunto de prácticas ágiles ampliamente utilizadas en la industria del software para mejorar la eficiencia y la calidad en el desarrollo de proyectos. Agile Alliance es una organización global sin ánimo de lucro dedicada a promover y difundir los valores y principios ágiles, proporcionando recursos, estudios y guías que facilitan la adopción de estas metodologías en diversos entornos de trabajo. Entre las prácticas ágiles destacadas en esta colección se encuentran la integración continua, la revisión de código, las retrospectivas y la gestión visual del trabajo, todas ellas orientadas a optimizar la colaboración y el flujo de desarrollo. Basándose en

estas prácticas, el sistema desarrollado en este proyecto evaluará aspectos clave del proceso de desarrollo, como la velocidad de trabajo (frecuencia de Commits e Issues cerradas), la documentación continua, la calidad de la documentación en Commits e Issues (uso de descripciones, etiquetas y herramientas de documentación) y la implementación de pruebas automatizadas. Para ello, se emplearán métricas cuantitativas que permitirán visualizar el rendimiento del equipo y la evolución del proyecto a lo largo del tiempo, ofreciendo recomendaciones que fomenten la mejora continua y la adopción de buenas prácticas ágiles.

Este proyecto busca facilitar la adopción de metodologías ágiles y mejorar la calidad del software y su documentación mediante la automatización del análisis de repositorios. A través de este enfoque, los equipos de desarrollo podrán optimizar sus procesos, mejorar la colaboración y garantizar la entrega de software más robusto y bien documentado.

En el contexto docente de los Trabajos de Fin de Grado (TFG), las recomendaciones generadas por la aplicación pueden clasificarse en diferentes reglas de metodologías ágiles según su aplicabilidad y alcance. Se incluyen:

DevOps - Automated Build: El repositorio incluye ficheros que automaticen el desarrollo (La aplicación analiza si se incluyen workflows de acciones de GitHub).

DevOps - Version Control: El repositorio aprovecha en todo lo posible las herramientas de GitHub para el control de versiones (La app analiza la frecuencia de los Commits y la calidad de estos (incluyen título, descripción y referencias a Issues).

DevOps, Extreme Programming - Continuous integration: El repositorio tiene señales de integración continua activa (La aplicación analiza el porcentaje de éxito de los workflows y la frecuencia de ejecución de estos y de Pull Requests).

Scrum - Definition of Done: El repositorio comprueba que se dan por terminadas las tareas o historias de usuario correctamente (La app observa la diferencia entre Issues cerradas y abiertas, y que las cerradas no se vuelvan a reabrir).

Scrum - Backlog Quality: El repositorio incluye numerosas issues y de calidad para formar el backlog del prpyecto (La aplicación cuenta el nº de issues y analiza que estas estén bien documentadas, incluyendo descripción con imágenes, personas asignadas y etiquetas)

Scrum, Extreme Programming - Iterations: El repositorio se está desarrollando mediante iteraciones o sprints (La aplicación analizará si se usan milestones en Issues y Pull Requests, y la frecuencia de Commits, merges y Releases).

Extreme Programming - Velocity: El repositorio tiene indicios de medición de velocidad de trabajo (Se anañizará si se usan etiquetas que indiquen story points y el número de Issues cerradas en cada Milestone y cada 15 días).

Extreme programming - Frequent Releases: El repositorio incluye Releases y se van creando nuevas cada cierto tiempo.

Extreme programming - Collective Ownership: Todos los miembros del equipo pueden modificar cualquier parte del repositorio en cualquier momento (La aplicación comprueba que todos los autores hagan Commits y Pull Requests, sean asignados a Issues, hagan de revisores a Pull Requests de otros, así como la actividad en el tiempo de estos autores).

Extreme programming - Pair Programming: Los miembros del repositorio practican la programación por parejas (Los mensajes de los Commits y Pull Requests contienen menciones a otros miembros con @, y los Issues o también contienen menciones, o tienen al menos 2 personas asignadas).

2. Objetivos del proyecto

Este apartado explica de forma precisa y concisa cuales son los objetivos que se persiguen con la realización del proyecto. Se puede distinguir entre los objetivos marcados por los requisitos del software a construir y los objetivos de carácter técnico que plantea a la hora de llevar a la práctica el proyecto.

2.1. Objetivos funcionales

Estos objetivos se centran en las funcionalidades que debe ofrecer al usuario la aplicación *Asistente de prácticas ágiles para repositorios en GitHub* para funcionar como se planteó correctamente, y ser una aplicación útil y fácil de usar

- **Análisis en profundidad de los repositorios**: La aplicación web analiza detalladamente y en profundidad todas las métricas de calidad de proceso y características que pueden resultar útiles de un repositorio durante el proceso de desarrollo de software. Esto ofrece al usuario una gran variedad de campos para mejorar su repositorio y facilitar su trabajo, por ejemplo, mejorando las decripciones y detalles de las Issues, o la frecuencia de subida de Commits.

- **Ajuste temporal del análisis**: El usuario puede ajustar distintos parámetros del análisis que ofrece la aplicación para determinar en qué medida se analizarán los repositorios, como por ejemplo el intervalo de tiempo de vida de los repositorios que se analizará, o los días utilizados como parámetro para calcular métricas de calidad de proceso relacionadas con las frecuencias de colaboraciones del repositorio.

- **Uso de las métricas de calidad de proceso obtenidas de los repositorios**: La aplicación es capaz de, dadas todas las métricas de calidad de proceso sacadas del repositorio, ofrecer al usuario sugerencias para mejorar su repositorio y rendimiento. Esto se hace en forma de reglas, que indican de una manera muy visual en qué aspectos falla el repositorio y en qué aspectos está mejor, indicando detalladamente el por qué y qué apartados mejorar para solucionarlo

- **Puesta en práctica de las metodologías ágiles**: La aplicación utiliza las bases teóricas de las metodologías ágiles recogidas en [1] para basar los resultados ofrecidos al usuario tras analizar el repositorio. El usuario puede acceder a URLs con contenido explicativo breve sobre el por qué se analizan ciertas métricas de calidad de proceso de cierta forma para determinar si un repositorio aprueba o suspende una regla.

- **Asistencia a estudiantes**: Se debe ofrecer una herramienta útil para los alumnos que estén usando repositorios de GitHub, realizando sus TFG, o se encuentren en cualquier momento del proceso de desarrollo de software mediante la aplicación de la herramienta de GitHub. La aplicación ofrece al usuario diferentes maneras de mejorar el rendimiento y calidad de su trabajo, explicando al mismo el porque esas maneras son buenas y útiles para desarrollar su repositorio

2.2. Objetivos no funcionales

Estos objetivos abarcan los obstáculos requeridos de superar para desarrollar la aplicación, como la tecnología y herramientas utilizadas para la programación y producción, y las prácticas y metodologías de programación usadas.

- **Mantenibilidad, modularidad y modificabilidad** : Se ha optado por Angular por su arquitectura basada en componentes, su flexibilidad y facilidad para la maquetación y el diseño de interfaces, su integración con formularios reactivos y su estructura dividida en módulos reutilizables. Esto, junto al uso del servicio HTTP para enviar y recibir datos al BackEnd asincrónicamente, gestionando respuestas y errores de forma controlada han permitido diseñar una aplicación web fácil de mantener y modificar.

- **Usabilidad de la aplicación web** La aplicación debe ser agradable a la vista y fácil de entender. Se busca una simplicidad que ofrezca

al usuario una experiencia fácil y rápida de entender. Para ellos se busca indicar de forma clara y concisa al usuario lo que está pasando, qué es lo que está haciendo, y qué significa lo que está viendo en todo momento. Esto se logra mediante la implementación de formularios reactivos con validación dinámica y lógica condicional para distintos modos de análisis (por intervalos de tiempo, fechas relativas, etc.) entre otras características, como los tabs de navegación, que hagan la aplicación usable y cómoda para al usuario.

- **Reusabilidad**: Aplicación de principios como DRY (Don't Repeat Yourself), SOLID (Especialmente el principio Open-Closed, por ejemplo para facilitar el añadido de métricas de calidad de proceso y reglas nuevas) y separación de responsabilidades, además del uso de TypeScript para mantener tipado estricto y detección temprana de errores.

3. Conceptos teóricos

En este apartado se exponen los principales conceptos teóricos necesarios para comprender el funcionamiento y objetivos del proyecto *Asistente de prácticas ágiles para repositorios en GitHub*, desde los fundamentos del control de versiones y GitHub, hasta las metodologías ágiles y métricas de calidad en el desarrollo software.

3.1. Control de versiones de repositorios en GitHub

El control de versiones es una práctica fundamental en el desarrollo de software moderno, ya que permite llevar un registro completo y detallado de los cambios realizados en el código fuente a lo largo del tiempo. Entre los distintos sistemas existentes, Git se ha consolidado como el más popular gracias a su eficiencia, flexibilidad y enfoque distribuido. Este enfoque significa que cada desarrollador posee una copia completa del historial del proyecto, lo que permite trabajar de forma autónoma y sin depender de una conexión constante a un servidor central.

GitHub, por su parte, es una plataforma de desarrollo colaborativo basada en Git que añade funcionalidades adicionales orientadas a la gestión de proyectos, la colaboración remota y la revisión de código. GitHub proporciona una interfaz web amigable que facilita la visualización del historial, la discusión de cambios y la organización del trabajo en equipo. Entre sus funcionalidades clave destacan:

- Repositorios y ramas: Permiten organizar el código en estructuras jerárquicas y paralelas. Las ramas facilitan el desarrollo simultáneo

de nuevas funcionalidades o corrección de errores sin interferir con el código principal.

- Issues: Herramienta integrada para el seguimiento de tareas, errores y solicitudes de mejora. Facilitan la comunicación entre los miembros del equipo y permiten priorizar el trabajo pendiente.
- Pull requests (PRs): Mecanismo para proponer y discutir cambios antes de su integración en la rama principal. Incluyen herramientas de revisión de código y validación automática mediante tests o workflows.
- Releases: Etiquetas versionadas que marcan hitos importantes del software, como entregas estables o versiones beta. Permiten identificar fácilmente el estado de madurez del proyecto.
- Estadísticas y métricas de calidad de proceso: GitHub ofrece paneles con datos sobre actividad reciente, contribuciones individuales, frecuencia de commits, entre otros indicadores clave del desarrollo del proyecto.

El análisis automático de estas funcionalidades y datos permite extraer información objetiva sobre la calidad, salud y evolución de un proyecto software. Esto es especialmente útil para evaluar repositorios de forma sistemática, identificar buenas prácticas o detectar áreas que requieren atención.

El análisis automático de estas métricas permite extraer conclusiones sobre la calidad, salud y evolución de un proyecto software.

3.2. Métricas de calidad de repositorios

Evaluar la calidad de un repositorio de software no solo implica revisar el código, sino también analizar cómo se gestiona y se desarrolla el proyecto en su conjunto. Esta última evaluación es la que desarrolla este proyecto empleando las metodologías ágiles. Para ello, se emplean métricas cuantitativas derivadas de la actividad registrada en el repositorio, que permiten realizar una evaluación objetiva y reproducible. Algunas de las métricas más relevantes incluyen:

- Número medio de commits: Indica la frecuencia con la que se realizan cambios en el código. Una actividad constante sugiere un proyecto en

desarrollo activo, mientras que largos periodos sin commits pueden señalar abandono o pausas.

- Tiempo medio de cierre de issues: Mide la eficiencia en la resolución de tareas o problemas. Tiempos de respuesta cortos suelen estar asociados a una buena organización y seguimiento, mientras que los retrasos pueden indicar falta de recursos o planificación.
- Frecuencia de publicación de releases: Refleja el ritmo de entregas del proyecto. Una cadencia regular sugiere una metodología estructurada, como la entrega incremental propia de metodologías ágiles.

Estas métricas permiten identificar patrones de trabajo, detectar cuellos de botella y sugerir mejoras conforme a las buenas prácticas del desarrollo ágil y colaborativo. Además, se pueden utilizar para monitorizar detalladamente la evolución de un mismo proyecto a lo largo del tiempo en cada apartado del repositorio.

3.3. Metodologías ágiles

Las metodologías ágiles representan un conjunto de principios y prácticas orientadas al desarrollo de software de manera iterativa, incremental y centrada en la colaboración continua entre los miembros del equipo y con los clientes. Frente a los modelos tradicionales, como el ciclo en cascada, las metodologías ágiles fomentan la adaptabilidad, la retroalimentación constante y la entrega temprana de valor.

Dentro del universo ágil destacan marcos de trabajo como Scrum o Extreme Programming (XP), que han sido adoptados ampliamente en la industria. Estos marcos establecen roles, eventos y artefactos que ayudan a estructurar el trabajo y mejorar la comunicación del equipo. En este proyecto, se siguen principios inspirados en la guía [1], la cual agrupa diversas prácticas ágiles aplicables al desarrollo de software. Entre las más relevantes se encuentran:

- Iteraciones y retrospectivas: Ciclos de trabajo cortos y repetitivos (sprints) al final de los cuales se evalúa lo realizado y se planifican mejoras para el siguiente ciclo.
- Historias de usuario y definición de “Hecho”: Las historias de usuario representan funcionalidades desde el punto de vista del usuario. La

definición de "Hecho" asegura que todo el equipo tiene un criterio común sobre cuándo una tarea está completada.

- Integración continua y revisión de código: Prácticas técnicas que permiten detectar errores tempranamente, asegurar la calidad del código y facilitar la colaboración mediante herramientas automatizadas y revisiones cruzadas entre desarrolladores.
- Visibilidad del trabajo en curso: Uso de herramientas como tableros Kanban, issues o sistemas de gestión de tareas para proporcionar una visión clara y compartida del estado del proyecto.

La aplicación desarrollada analiza la actividad del repositorio para comprobar si estas prácticas están siendo implementadas correctamente. Para ello, aplica un conjunto de reglas automáticas y sugerencias inteligentes, evaluando tanto la estructura del repositorio como su dinámica de trabajo. Esto permite no solo diagnosticar el grado de alineación con metodologías ágiles, sino también fomentar la mejora continua en la gestión de proyectos software.

3.4. Análisis temporal y evaluación del proceso

Una de las principales innovaciones que ofrece la aplicación es la incorporación de un análisis temporal personalizado, que permite estudiar cómo evoluciona el proceso de desarrollo a lo largo del tiempo. Esta funcionalidad permite seleccionar intervalos temporales específicos (como un rango de meses determinado) o relativos (por ejemplo, la primera mitad del proyecto, el tercer cuarto, etc), y aplicar las métricas correspondientes exclusivamente a ese periodo. Este tipo de análisis temporal posibilita una evaluación dinámica y contextualizada del desarrollo del proyecto.

Entre los principales beneficios de este enfoque se encuentran:

- Evaluación de la evolución de la calidad del repositorio: Al observar cómo cambian las métricas a lo largo del tiempo, es posible identificar tendencias positivas o negativas en aspectos como frecuencia de commits, resolución de issues o participación del equipo.
- Análisis del impacto de cambios metodológicos: Se pueden estudiar los efectos de la introducción de nuevas prácticas de gestión, como la

adopción de Scrum, integración continua o tableros Kanban, evaluando cómo influyen en la actividad y organización del equipo.

- Medición de la madurez y estabilidad del equipo de desarrollo: Los equipos que evolucionan positivamente tienden a mostrar una estabilización en sus métricas, una mejor distribución del trabajo y un incremento en la colaboración. Estas señales permiten inferir el grado de madurez del equipo.

Este análisis es especialmente útil en contextos formativos, como el seguimiento de proyectos académicos en la Universidad de Burgos. En estos casos, permite a los tutores y alumnos evaluar el proceso de desarrollo de forma empírica, justificar la aplicación de metodologías ágiles en prácticas reales, y fomentar la reflexión crítica sobre el trabajo realizado.

3.5. Evaluación de la agilidad en proyectos académicos

El uso de la aplicación en entornos educativos proporciona una herramienta de evaluación automatizada y formativa que resulta especialmente valiosa para la enseñanza de metodologías ágiles. En particular, permite evaluar de forma objetiva y continua el grado en que los equipos de estudiantes aplican los principios ágiles en proyectos de software, como los que se desarrollan en las asignaturas de la Universidad de Burgos.

Gracias al análisis de datos extraídos directamente desde los repositorios de GitHub, la aplicación proporciona retroalimentación inmediata y personalizada a los estudiantes, permitiéndoles:

- Autoevaluar su desempeño: Identificando prácticas que se están aplicando correctamente y aquellas que requieren mejora.
- Detectar deficiencias metodológicas: Como falta de actividad, ausencia de releases o escasa participación en la resolución de issues.
- Recibir sugerencias automáticas: Basadas en reglas predefinidas y métricas objetivas, que orientan al alumno hacia una mejor implementación del enfoque ágil.

Para los docentes, esta herramienta facilita una evaluación más justa, uniforme y basada en evidencia, permitiendo valorar aspectos que habitualmente son difíciles de medir, como:

- Grado de implementación de marcos ágiles (como iteraciones o control de versiones), más allá de una simple declaración teórica.

- Compromiso del equipo con las prácticas ágiles, medido a través de la participación activa y la frecuencia de interacción en el repositorio.

- Documentación y trazabilidad del trabajo en GitHub, lo que incluye el uso de issues, descripciones de commits, pull requests revisadas, y releases etiquetadas.

Así, el análisis no solo tiene valor evaluativo, sino también educativo, ya que fomenta el aprendizaje autónomo y refuerza la comprensión de las buenas prácticas en el desarrollo ágil.

3.6. Reglas de evaluación basadas en prácticas ágiles

La aplicación analiza la actividad de un repositorio de GitHub para determinar el grado de adopción de prácticas ágiles en el desarrollo software. Para ello, se basa en 10 reglas clave inspiradas en el *Subway Map to Agile Practices* [1], una recopilación de buenas prácticas organizada por familias (Scrum, Extreme Programming, DevOps, etc.).

A continuación se detallan estas reglas, su justificación y el modo en que son evaluadas por la aplicación.

DevOps – Automated Build

- **Descripción:** El proyecto debe incluir automatización del proceso de construcción, integración o despliegue.

- **Criterio de evaluación:** El asistente detecta si existen ficheros de workflows (por ejemplo, bajo `.github/workflows`) que definan procesos automáticos usando GitHub Actions.

- **Importancia:** Favorece la repetibilidad, estandarización y reducción de errores humanos.

DevOps – Version Control

- **Descripción:** Uso activo y adecuado del control de versiones.
- **Criterio de evaluación:** Se analiza la frecuencia de los commits, que estos tengan título, descripción y que estén vinculados a issues mediante `#id` o `fixes #id`.
- **Importancia:** Mejora la trazabilidad, control del código y colaboración entre miembros.

DevOps, Extreme Programming – Continuous Integration

- **Descripción:** El repositorio debe integrar cambios frecuentemente, ejecutando pruebas automáticas.
- **Criterio de evaluación:** Se mide el porcentaje de éxito de los workflows automáticos, la frecuencia de su ejecución y el uso de Pull Requests para introducir cambios.
- **Importancia:** Detecta errores pronto y reduce conflictos al integrar código.

Scrum – Definition of Done

- **Descripción:** Las tareas deben darse por completadas de forma inequívoca.
- **Criterio de evaluación:** El asistente compara el número de issues abiertas y cerradas, asegurando que las cerradas no se vuelven a abrir.
- **Importancia:** Establece una línea clara sobre lo que se considera terminado y entrega de valor.

Scrum – Backlog Quality

- **Descripción:** El backlog del proyecto debe estar compuesto por issues claras, bien documentadas y priorizadas.

- **Criterio de evaluación:** Se comprueba que las issues tengan descripción (preferiblemente con imágenes o checklists), personas asignadas y etiquetas.
- **Importancia:** Permite gestionar el trabajo pendiente de forma organizada.

Scrum, Extreme Programming – Iterations

- **Descripción:** El proyecto debe avanzar por ciclos iterativos como sprints.
- **Criterio de evaluación:** Se analiza si se utilizan milestones, así como la frecuencia de commits, merges y releases por ciclo.
- **Importancia:** Favorece la entrega frecuente de valor y mejora continua.

Extreme Programming – Velocity

- **Descripción:** El equipo debe ser capaz de medir su velocidad de entrega de funcionalidades.
- **Criterio de evaluación:** Se revisa el uso de etiquetas con story points o complejidad, así como el número de issues cerradas por milestone o cada 15 días.
- **Importancia:** Permite planificar mejor la carga de trabajo y tomar decisiones basadas en datos.

Extreme Programming – Frequent Releases

- **Descripción:** El proyecto debe generar versiones o releases con frecuencia.
- **Criterio de evaluación:** Se comprueba la existencia de releases en GitHub y la periodicidad con que se publican.
- **Importancia:** Permite validar funcionalidad de forma incremental y tener feedback temprano de usuarios.

Nombre	Criterio evaluado	Práctica Agil asociada
Automated Build	Presencia de workflows de GitHub Actions	DevOps
Version Control	Frecuencia y calidad de commits	DevOps
Continuous Integration	Éxito y frecuencia de workflows y PRs	DevOps, XP
Definition of Done	Issues cerradas correctamente	Scrum
Backlog Quality	Calidad y estructura de issues	Scrum
Iterations	Uso de milestones y ciclos de commits	Scrum, XP
Velocity	Story points y métricas cada 15 días	XP
Frequent Releases	Releases frecuentes en el tiempo	XP
Collective Ownership	Participación múltiple en tareas y PRs	XP
Pair Programming	Menciones y asignaciones múltiples	XP

Tabla 3.1: Resumen de reglas ágiles evaluadas por la aplicación

Extreme Programming – Collective Ownership

- **Descripción:** Todo miembro del equipo puede modificar cualquier parte del código.
- **Criterio de evaluación:** Se analiza que haya varios autores haciendo commits, revisando PRs, resolviendo issues y participando activamente en diferentes partes del repositorio.
- **Importancia:** Fomenta la colaboración y reduce los cuellos de botella por eExtreme Programmingertos únicos.

Extreme Programming – Pair Programming

- **Descripción:** Los miembros colaboran en el desarrollo a través de la programación en pareja.
- **Criterio de evaluación:** Se detectan menciones a otros miembros en commits, PRs o issues (@usuario), así como asignaciones múltiples en tareas.
- **Importancia:** Mejora la calidad del código y favorece el aprendizaje mutuo.

Resumen de reglas evaluadas por la aplicación

Estas reglas no solo ayudan a evaluar objetivamente el proceso de desarrollo de un repositorio, sino que también proporcionan a los usuarios sugerencias claras sobre cómo mejorar su adopción de prácticas ágiles.

3.7. Referencias y métricas

Las métricas y reglas empleadas por el asistente están fundamentadas en investigaciones previas y guías ampliamente reconocidas dentro del ámbito del desarrollo ágil y la ingeniería del software. En particular, se han tomado como base:

El Agile Subway Map [1], que ofrece una representación estructurada de prácticas ágiles agrupadas por niveles de madurez y propósito (por ejemplo, entrega continua, mejora del equipo, colaboración).

Estudios sobre calidad del software en entornos colaborativos, que analizan el impacto de determinadas métricas de actividad y colaboración en la sostenibilidad y efectividad de los proyectos de código abierto o académico.

Estas métricas se implementan mediante un sistema de reglas automatizadas que evalúan si ciertos umbrales se cumplen o no, y que pueden ser ajustadas según el contexto (proyecto académico, industrial, etc.). Este enfoque modular permite adaptar el análisis a diferentes necesidades, manteniendo siempre un criterio objetivo y reproducible.

3.8. Resumen de herramientas utilizadas

Herramientas	Frontend	Backend	Base de Datos	Documentación
HTML5	X			
TypeScript	X			
Angular 19.2.3	X			
Node.js		X		
XP		X		
GitHub API		X		
Mongoose		X	X	
MongoDB			X	
MongoDB Compass			X	
JWT		X		
Swagger (OpenAPI)		X		
LaTeX				X
Overleaf				X
Git	X	X	X	X

Tabla 3.2: Herramientas y tecnologías utilizadas en cada parte del proyecto

4. Técnicas y herramientas

4.1. Metodología de desarrollo

Durante el desarrollo del proyecto se ha seguido un enfoque iterativo e incremental, inspirado en las metodologías ágiles, especialmente en la metodología **Scrum**. Aunque no se ha implementado un Scrum completo con sprints y reuniones formales, sí se ha adoptado la filosofía de desarrollo en pequeñas iteraciones, con validaciones frecuentes de funcionalidad y mejora continua.

Cada iteración se centraba en funcionalidades concretas: la recolección de datos, la visualización de estadísticas, la implementación de reglas de evaluación y la mejora de la experiencia de usuario. Esta división permitió avanzar progresivamente, validando en cada fase los resultados parciales antes de pasar a la siguiente.

Además, se ha hecho uso de herramientas como **Zube** para la organización de tareas y la planificación de Issues. Esto ha facilitado el seguimiento del progreso del desarrollo.

4.2. Tecnologías y herramientas utilizadas

Frontend: Angular

Para la implementación del cliente se ha optado por el framework **Angular**, en su versión 19.2.3. Angular ofrece una arquitectura basada en componentes, módulos reutilizables, servicios inyectables y soporte integrado para formularios reactivos. Algunas de sus características más destacadas y útiles en este proyecto han sido:

- Formularios reactivos con validación dinámica.
- Módulos reutilizables para organizar mejor el código.
- Inyección de dependencias para desacoplar lógica y facilitar pruebas.
- Integración con bibliotecas como Angular Material para las interfaces de usuario.

Se consideraron otras opciones como **React**, pero se optó por Angular debido a su enfoque más completo y estructurado, más adecuado para una aplicación de análisis con múltiples pantallas y lógica interna, además de la existencia de experiencias previas con el framework.

Backend: Node.js

Para el backend se ha utilizado **Node.js**, un entorno de ejecución para JavaScript que permite desarrollar aplicaciones del lado del servidor de manera eficiente y escalable. Se ha estructurado el servidor como una API REST que recibe peticiones desde el cliente Angular, procesa los datos y devuelve los resultados del análisis.

- **Modularidad y ecosistema**: Se han utilizado módulos como `express` para la definición de rutas y controladores, y bibliotecas específicas para consumir APIs de GitHub y procesar datos.
- **Facilidad de integración**: Al estar escrito en JavaScript, el backend se integra de forma natural con el frontend desarrollado en Angular, facilitando la comunicación y compartición de lógica o estructuras comunes, lo que facilita el desarrollo.
- **Manejo eficiente de errores y validaciones**: Se ha implementado una estructura de control de errores robusta, validación de parámetros de entrada, y respuestas claras al cliente ante peticiones incorrectas o fallos del análisis.

Se consideraron otras alternativas como Python (con frameworks como FastAPI o Flask), pero se optó por Node.js por su rapidez de desarrollo, la familiaridad con el lenguaje y su amplia adopción en aplicaciones web modernas.

Comunicación cliente-servidor

La comunicación entre el cliente Angular y el backend FastAPI se realiza a través de peticiones HTTP RESTful, usando el módulo `HttpClient` de Angular. La respuesta del backend incluye los datos analizados y evaluados a partir de los repositorios proporcionados por el usuario.

Base de datos: MongoDB y MongoDB Compass

Para el almacenamiento de información relacionada con los análisis realizados y la persistencia de datos relevantes, se ha utilizado **MongoDB**, una base de datos NoSQL orientada a documentos. Esta elección se debe a su flexibilidad, escalabilidad y facilidad de uso con Node.js.

- **Modelo de documentos**: MongoDB permite trabajar con documentos en formato JSON, lo cual se adapta perfectamente a los datos semiestructurados obtenidos del análisis de los repositorios de GitHub.
- **Integración con Node.js**: Se ha utilizado la biblioteca `mongoose` para gestionar la conexión, definición de esquemas y operaciones con la base de datos desde el backend Node.js, facilitando el trabajo con modelos y validaciones.
- **Visualización y gestión con MongoDB Compass**: Para la gestión visual y análisis manual de los datos almacenados se ha utilizado **MongoDB Compass**, una interfaz gráfica que permite explorar documentos, ejecutar consultas, y visualizar estadísticas de uso de la base de datos de forma sencilla e intuitiva.

La elección de MongoDB frente a bases de datos relacionales como MySQL o PostgreSQL se justifica por la naturaleza no estructurada de los datos analizados y la necesidad de una estructura flexible que pueda evolucionar fácilmente sin migraciones complejas, además se su buena integración con el backend Node. js.

Diseño de la interfaz de usuario

Se ha utilizado la biblioteca **Angular Material** para implementar una interfaz moderna y coherente visualmente. Esta biblioteca proporciona componentes reutilizables (botones, tablas, inputs, spinners, etc.) con accesibilidad integrada y estilos basados en Material Design.

El diseño se ha centrado en la claridad y la simplicidad, mostrando los resultados del análisis de manera visual e intuitiva, junto con retroalimentación clara (colores, iconos, puntuaciones) para ayudar al usuario a interpretar los resultados.

Gestión del código fuente

El código fuente del proyecto se ha gestionado con **Git**, utilizando GitHub como repositorio remoto. Se ha trabajado con ramas para separar funcionalidades y mantener el código estable en la rama principal.

Despliegue continuo de la aplicación web durante el desarrollo

Para el testing y el desarrollo del proyecto se han utilizado dos herramientas que han permitido el despliegue continuo de la aplicación web a través de la integración continua lograda gracias a la compatibilidad entre estas dos herramientas y el repositorio de GitHub donde se ha ido subiendo el código del backend y el frontend

- **Despliegue continuo del backend**: Se ha utilizado la herramienta de React para lograr el despliegue continuo del backend desde el repositorio de GitHub
- **Despliegue continuo del frontend**: Para el frontend se ha utilizado la herram,ienta de Vercel, que permite un despliegue continuo rápido y eficiente, que se puede probar desde cualquier dispositivo por cualquier persona gracias a la opción de usar despliegues de carácter público.

Estas herramientas de despliegue han permitido el testeo de tanto el tutor como diferentes usuarios que han ayudado al desarrollo de la aplicación web, y a la búsqueda de errores y bugs.

4.3. Otras herramientas de apoyo

- **Visual Studio Code**: Editor principal usado para escribir tanto el frontend como el backend.
- **LaTeX**: Utilizado para la redacción de esta memoria, por su capacidad de estructurar documentos técnicos de manera clara y profesional.

- **Zube y GitHub**: Para documentar el código, organizar las tareas de desarrollo y las decisiones técnicas dentro del propio repositorio se ha utilizado la herramienta de Zube, recomendada por el profesor y conectada con el repositorio de GitHub.

- **GitHub Desktop**: Herramienta gráfica de GitHub que permite visualizar claramente los cambios realizados en todo el repositorio de forma individual y comparativa antes de subir todos los cambios con un solo Commit.

4.4. Comparativa de tecnologías evaluadas

Durante la fase inicial del proyecto se analizaron distintas tecnologías. La siguiente tabla resume los principales aspectos que se compararon:

Tecnología	Facilidad de uso	Rendimiento	Documentación
Angular	Alta	Media-Alta	Muy completa
React	Media	Alta	Muy completa
Vue	Alta	Media	Buena
FastAPI	Alta	Alta	Muy completa
Flask	Alta	Media	Muy completa
Django	Media	Media	Muy completa

Tecnología	Facilidad de uso	Rendimiento	Documentación
Node.js	Media-Alta	Alta	Muy completa
Express.js	Alta	Alta	Muy completa
FastAPI	Alta	Alta	Muy completa
Flask	Alta	Media	Muy completa
Django	Media	Media	Muy completa
Spring Boot	Baja	Muy Alta	Muy completa

Base de Datos	Facilidad de uso	Rendimiento	Documentación
MongoDB	Alta	Alto (NoSQL)	Muy completa
MySQL	Medio-Alta	Alto	Muy completa
PostgreSQL	Media	Muy Alto	Muy completa
SQLite	Alta	Medio-Bajo	Buena
Firebase Realtime DB	Muy alto	Media	Buena
Redis	Media	Muy Alto (caché)	Buena

La elección final se basa en la compatibilidad entre tecnologías, la curva de aprendizaje, y el tipo de aplicación (analítica, con formularios complejos y visualización estructurada).

5. Aspectos relevantes del desarrollo del proyecto

Durante la implementación de este Trabajo de Fin de Grado, se han tomado decisiones técnicas y de diseño que han marcado de forma clara el rumbo y la calidad final del proyecto *Asistente de prácticas ágiles para repositorios en GitHub*. Este capítulo describe los aspectos más destacados del desarrollo, sus motivaciones y las soluciones adoptadas, con el objetivo de que puedan servir de referencia para otros estudiantes o desarrolladores que se enfrenten a retos similares.

5.1. Motivación para el proyecto

Desde el principio la idea del proyecto era asistir al usuario al desarrollo de la creación de trabajos en repositorios de GitHub, pero no al código de los trabajos, si no al repositorio en sí. En GitHub hay un montón de herramientas y opciones que pasan desapercibidas, especialmente para alumnos de la carrera de Ingeniería Informática que no tienen tanta experiencia con esta plataforma. Muchos hacen proyectos enteros sin saber siquiera lo que es una Issue, una Release, que pueden agregar implementación continua usando GitHub Actions, etc. Por ello se decidió que en este Trabajo de Fin de Grado se desarrollaría una App que ayudara al desarrollo de repositorios en el ámbito docente para facilitiar y agilizar la planificación trabajo de los alumnos en GitHub.

5.2. Las metodologías Ágiles como base teórica

Para ayudar a los usuarios a trabajar con sus repositorios necesitaríamos unas bases teóricas que ayudaran a justificar el por qué de los resultados que la app proporcionaría, y finalmente se eligieron las metodologías Ágiles, un concepto con el que los alumnos del Grado de Ingeniería Informática están familiarizados, especialmente por la asignatura del tercer curso "Gestíon de Proyectos". Finalmente se así decidió mostrar los resultados de los análisis de los repositorios en forma de reglas de agilidad sacadas del Agile Subway Map [1] como Integración Continua, Control de Versiones, etc. Estas reglas evalúan aspectos como la frecuencia de commits, la coherencia en los mensajes, la asignación de issues, y la distribución del trabajo.

Posteriormente se desarrolló un módulo de evaluación de estas reglas que, a partir de los datos obtenidos vía la API de GitHub, genera una puntuación por cada regla y un resumen visual en el FrontEnd que permite observar tanto rápidamente los puntos fuertes y débiles del repositorio, como detalladamente las bases teóricas de la metodología Ágil que se cumplen o no, explicando el por qué. Esta funcionalidad fue clave para transformar el análisis en una herramienta útil para no solo alumnos, sino también profesores o equipos de desarrollo.

5.3. La tecnología de Node js y Angular

: Se optó por usar la tecnología de Angular para el FrontEnd por su gran capacidad para diseñar aplicaciones web eficaces, fáciles de maquetar y estilizar, y gran variedad de herramientas como servicios, componentes, módulos, y librerías que ayudan a desarrollar fácilmente una aplicación totalmente funcional y agradable al usuario. Previamente a este proyecto ya había trabajado en varias aplicaciones que usaban esta tecnología, lo cual ayudó desde un comienzo a tener clara la estructura de esta parte del código.

También se optó por usar Node js en el BackEnd debido a la similaridad del lenguaje (JavaScript) con el que usa Angular para sus componentes que implementan la funcionalidad del código (TypeScript).

Estas decisiones permitieron generar un ejecutable funcional, autocontenible y sin dependencias externas, mejorando la portabilidad y experiencia del usuario final.

5.4. Selección de MongoDB como sistema de almacenamiento

MongoDB fue elegido como base de datos por su naturaleza flexible y su excelente integración con Node.js y su facilidad de uso. Además, al tratarse de una aplicación local, se optó por una base de datos embebida que no requiriese configuración externa por parte del usuario.

5.5. Arquitectura modular y comunicación entre capas

La aplicación se estructura en tres capas claramente diferenciadas: BackEnd (Node.js), base de datos (MongoDB) y FrontEnd (Angular)). Esta separación permitió trabajar en cada módulo de manera independiente y facilitó la escalabilidad del sistema.

El BackEnd ofrece una API REST con rutas específicas para analizar repositorios, extraer estadísticas y aplicar reglas basadas en metodologías ágiles, todo esto guardando los resultados en una base de datos en MONGODB, de la cual extraerlos posteriormente para mandarlos al FrontEnd una vez se requieran a través de diferentes llamadas y joins. Por su parte, el FrontEnd permite al usuario interactuar de forma visual, seleccionando repositorios, rangos de fechas y recibiendo evaluaciones contextualizadas. La comunicación se realiza mediante peticiones HTTP sobre `localhost`, garantizando así una experiencia fluida sin requerir conexión a internet.

5.6. Integración de lenguaje natural para resúmenes y explicaciones

Con el objetivo de ofrecer una experiencia más didáctica y comprensible, se exploró el uso de modelos de lenguaje (LLM) para generar explicaciones en lenguaje natural a partir de los resultados del análisis.

Inicialmente se evaluaron opciones como OpenAI GPT y modelos open-source, priorizando la claridad y la coherencia en las respuestas. Finalmente, debido a que la mayoría de mensajes de resultado que aparecerían tras el análisis serían similares estructuralmente y bastante cortos dentro de lo posible, se decidió que como posible mejora del sistema, se podría integrar

una capa de generación de lenguaje que permita resumir los informes y ofrecer feedback personalizado.

5.7. Desarrollo iterativo y depuración mediante testing manual

Dado el alcance de la aplicación y su complejidad funcional, se adoptó un enfoque de desarrollo iterativo. Cada módulo (visualización, análisis, integración con GitHub, etc.) fue construido y probado individualmente a través de la basede datos de Mongo y la consola del BackEnd antes de su integración global.

Uno de los principales retos cerca del final del proyecto fue la depuración en entorno empaquetado. Al final se optó por lanzar las releases en forma de BackEnds empaquetados que accederían al FrontEnd y a las variables de entorno en la misma carpeta y, por motivos de calidad, sería necesario insertar registros de log visibles en pantalla o mediante archivos para detectar errores silenciosos, así como el progreso de análisis de repositorios de la aplicación.

5.8. Autoformación y búsqueda activa de soluciones

El proyecto ha supuesto un esfuerzo significativo de aprendizaje autónomo en varias tecnologías: Angular, Node js, MongoDB o GitHub REST API. Muchos de los problemas encontrados fueron resueltos consultando documentación oficial, foros técnicos (como Stack Overflow o GitHub Issues), recursos especializados, o comparándolos con problemas similares tenidos en proyectos previos (Especialmente en la parte del FrontEnd en Angular).

6. Trabajos relacionados

La funcionalidad principal y más común de este proyecto es el análisis de reposiotrios de GitHub, y la comparación entre estos. Hay una gran variedad de herramientas similares que también analizan repositorios, y/o los comparan entre sí.

Se han seleccionado algunas de estas herramientas debido a su similaridad con la aplicación que presenta este proyecto a la hora de analizar aspectos clave de repositorios. Estas herramientas son: *Criticality-Score* [8], *Agile-Metrics* [4], *Activiti-API* [5] *Comparador-de-metricas-de-evolucion-en-repositorios-ftware* [7]

Estas herramientas realizan análisis puntuales de repositorios, pero no permiten al usuario especificar periodos temporales personalizados. Los proyectos mencionados abordan el análisis de proyectos de software desde distintas perspectivas, sirviendo como referentes valiosos para el desarrollo del presente trabajo.

Criticality-Score se centra en medir la importancia de los proyectos open source para priorizar su seguridad, pero no evalúa métricas cualitativas ni trabaja con umbrales.

Agile-Metrics recopila indicadores ágiles según la forja de repositorio utilizada, pero carece de una unificación de métricas y no trabaja con GitHub, lo que limita su alcance.

Activiti-API, más cercano al proyecto presentado en esta memoria, permite evaluar y comparar métricas de actividad, aunque sólo entre dos proyectos.

El proyecto *Comparador-de-metricas-de-evolucion-en-repositorios-Software* [7] se centra en la mejora de un proyecto previo que trata sobre los procesos de

desarrollo mediante la automatización del cálculo de métricas de evolución extraídas de plataformas como GitHub o GitLab. Su enfoque se enmarca dentro del desarrollo iterativo e incremental característico de metodologías ágiles como Scrum o XP, y busca proporcionar a los jefes de proyecto una herramienta que les permita detectar posibles ineficiencias en los ciclos de desarrollo.

Herramientas	**Análisis temporal**	**Selección personalizada de fechas**	**Uso de intervalos de tiempo absolutos y relativos**	**Enfoque docente**	**Base teórica sobre metodologías Ágiles**	**Comparación de múltiples repositorios**
Asistente de prácticas ágiles para repositorios en GitHub	Sí	Sí	Sí	Sí	Sí	Sí
Comparador de metricas de evolucion en repositorios Software	Sí	No	No	No	No	Sí
Activiti-API	No	No	No	No	No	Sí (Dos repositorios)
Agile-Metrics	Sí	No	No	No	No	No
Criticality-Score	No	No	No	No	No	No

Tabla 6.1: Comparación de herramientas y trabajos relacionados

El proyecto presentado en este TFG amplía las funcionalidades de las herramientas mencionadas con una mayor capacidad de comparación y evaluación de métricas mediante el análisis de la evolución temporal usando fechas e intervalos personalizados, resultando así en una herramienta avanzada tanto para análisis retrospectivos en detalle como para evaluación continua durante el desarrollo.

Esta funcionalidad permite al usuario seleccionar rangos de fechas personalizados sobre los cuales se calcularán las métricas, facilitando así el seguimiento detallado del progreso, mantenimiento y salud de un proyecto a lo largo del tiempo. Esta característica no está presente las herramientas analizadas, donde los análisis se realizan de forma estática y centrados en el estado actual del repositorio, aunque se utilicen fechas para la obtención de algunas métricas.

Gracias a esta funcionalidad temporal, el usuario puede, por ejemplo, evaluar cómo ha cambiado la calidad del repositorio desde el inicio del mismo, o comprobar si el uso de las prácticas ágiles de desarrollo han tenido impacto en la velocidad y el desarrollo del mismo. Esta capacidad resulta especialmente útil en contextos docentes (como proyectos fin de carrera o asignaturas de carreras relacionadas a la informática), donde se requiere

un seguimiento detallado y didáctico de la evolución del trabajo de los estudiantes.

Además, incluye la posibilidad de realizar tanto análisis relativos como análisis absolutos. Esto es una opción útil a la hora de ver cómo evolucionan las estadísticas cuantitativas del repositorio, lo que permite no sólo observar si un repositorio mejora o empeora con el paso de unidades de tiempo, sino también con el paso de las distintas fases de desarrollo del proyecto. Por ejemplo, se puede observar si la definición de "Done" se respeta de forma más consistente con el paso de los sprints..

Los análisis cuantitativos realizados por la aplicación (como el promedio de subidas de releases, la frecuencia de commits, o la media de issues cerradas en distintos periodos temporales—) permiten relacionar directamente el comportamiento observable del proyecto con el cumplimiento de ciertas prácticas ágiles.

Por ejemplo, una alta frecuencia de commits con regularidad en el tiempo puede asociarse con una correcta integración continua (CI), una práctica clave en metodologías como Scrum y XP. Del mismo modo, un bajo tiempo medio de cierre de issues sugiere una gestión eficiente del product backlog, lo cual refleja un buen refinamiento del mismo y una comunicación efectiva entre los miembros del equipo.

Además, los intervalos personalizados permiten correlacionar periodos de desarrollo con eventos concretos (entregas, sprints, cambios de equipo, etc.) para evaluar su impacto. Esto se alinea con la filosofía ágil de desarrollo continuo, ya que el análisis permite detectar patrones de mejora o estancamiento y aplicar ajustes informados.

Otro aspecto relevante que potencia esta herramienta es su capacidad para analizar repositorios reales públicos entregados por estudiantes en la Universidad de Burgos (UBU). Esto no solo sirve como una oportunidad de evaluación objetiva del grado de aplicación de prácticas ágiles en un entorno académico, sino también como un mecanismo de autoevaluación y aprendizaje para los alumnos. Al observar sus métricas de trabajo reflejadas en estadísticas, pueden identificar áreas de mejora, ajustar su forma de colaborar, y alinearse mejor con los principios ágiles.

Así, la herramienta no solo sirve como analizador técnico, sino también como un instrumento docente útil para la enseñanza y práctica de metodologías ágiles en entornos universitarios.

7. Conclusiones y Líneas de trabajo futuras

El desarrollo de este proyecto ha permitido abordar de forma práctica y teórica el análisis automatizado de la calidad y la agilidad en proyectos software, con especial atención a su aplicación en entornos académicos. A lo largo del trabajo se ha construido una herramienta funcional que integra métricas objetivas, principios metodológicos y capacidades analíticas para evaluar la evolución y el cumplimiento de buenas prácticas en repositorios gestionados mediante GitHub.

Conclusiones generales

Una de las principales aportaciones del proyecto ha sido la demostración de que es posible realizar un seguimiento detallado y automatizado de la calidad del desarrollo de un proyecto utilizando exclusivamente datos disponibles públicamente en plataformas como GitHub. Este enfoque presenta importantes ventajas en términos de escalabilidad, reproducibilidad y objetividad, y puede aplicarse tanto en contextos profesionales como educativos.

Desde un punto de vista metodológico, el proyecto ha validado el uso de métricasde calidad de proceso centradas en la actividad, colaboración y mantenimiento como indicadores fiables de la salud y madurez de un proyecto. Asimismo, se ha confirmado que el análisis temporal aporta una dimensión adicional imprescindible para comprender la evolución de trabajo de un equipo y la implementación progresiva de metodologías ágiles.

Conclusiones técnicas

Desde el plano técnico, se destacan los siguientes logros:

- Se ha desarrollado una herramienta capaz de recopilar y analizar datos estructurados desde la API de GitHub, procesando la información relevante de Commits, Issues, Pull Requests, Releases, ficheros workflow y de la actividad de los miembros del equipo.
- Se ha implementado un sistema de métricas y reglas evaluativas, basado en trabajos de referencia como el *Agile Subway Map*, permitiendo una evaluación automática de prácticas ágiles y de calidad técnica del repositorio. Gracias a la naturaleza del código que cumple el principio abierto / cerrado es sencillo agregar nuevas reglas en el futuro.
- Se ha incorporado un módulo de análisis temporal dinámico, que permite filtrar y segmentar los datos en función de intervalos absolutos o relativos, lo que amplía notablemente las posibilidades de evaluación y seguimiento.
- Se ha demostrado la viabilidad del uso académico de la herramienta, facilitando tanto la autoevaluación por parte de los estudiantes como la evaluación objetiva por parte del profesorado.

Valoración crítica del proyecto

Aunque el proyecto ha cumplido con los objetivos establecidos inicialmente, se han identificado ciertos aspectos que podrían mejorarse o extenderse en futuros desarrollos:

- La interfaz de visualización y presentación de resultados podría enriquecerse mediante el uso de dashboards interactivos, gráficas dinámicas y alertas visuales que faciliten una interpretación más intuitiva por parte del usuario.
- Otra posible mejora sería permitir una configuración más flexible y adaptativa, dependiendo del tipo de proyecto (tamaño del equipo, duración, objetivos pedagógicos), que permita elegir qué reglas se evaluarán o incluir nuevas personalizadas.

- El sistema podría beneficiarse de la integración con otras plataformas complementarias a GitHub, como Jira o Trello, para obtener una visión más completa del proceso de gestión ágil cuando se utilicen otras herramientas en paralelo.

Líneas de trabajo futuras

A partir de los resultados obtenidos y de la experiencia adquirida en este proyecto, se proponen varias líneas de trabajo futuro que permitirían ampliar el alcance, mejorar la utilidad y profundizar en la capacidad analítica del sistema desarrollado:

1. Ampliación del conjunto de métricas y reglas: Incorporar nuevas métricas relacionadas con la calidad del código, como el número de revisiones por Pull Request, cobertura de código, o uso de despliegues.
2. Desarrollo de un modelo de puntuación integral: Diseñar un sistema de puntuación global que integre múltiples dimensiones (actividad, colaboración, mantenimiento, calidad) para ofrecer una evaluación unificada del estado del proyecto.
3. Validación empírica con más proyectos académicos y profesionales: Realizar estudios de caso con distintas asignaturas, universidades o incluso equipos reales del sector para validar la robustez y adaptabilidad del sistema en diversos contextos.
4. Incorporación de aprendizaje automático: Aplicar técnicas de machine learning para detectar patrones de comportamiento, predecir cuellos de botella o sugerir automáticamente acciones de mejora basadas en proyectos similares.

Reflexión final

En definitiva, este proyecto representa un primer paso hacia el uso de herramientas automatizadas para la evaluación del proceso de desarrollo software en repositorios de GitHub, integrando principios de ingeniería de software, analítica de datos y metodologías ágiles. El enfoque propuesto no solo permite mejorar la gestión técnica de proyectos, sino que también fomenta el aprendizaje reflexivo y la mejora continua en contextos formativos. Las posibilidades de extensión son numerosas y abren la puerta a una línea de investigación y desarrollo con gran potencial académico y profesional.

Bibliografía

[1] Agile Alliance. Subway map to agile practices, 2023.

[2] Kent Beck and Cynthia Andres. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 2nd edition, 2004.

[3] Wei-Yang Chen, Hsi-Min Chen, and Yen-Tse Hsueh. Work in progress: Github-driven learning: A case study in agile-infused software design courses. In *Proceedings of the 2023 8th International Conference on Information Systems Engineering*, ICISE '23, page 13–18, New York, NY, USA, 2024. Association for Computing Machinery.

[4] Giuliano Da Grisa. agile-metrics: Consumer and process-oriented agile software metrics. https://github.com/DaGrisa/agile-metrics, 2018.

[5] dba0010. Activiti-Api: RESTful API for Activiti workflows. https://github.com/dba0010/Activiti-Api, 2015.

[6] International Organization for Standardization (ISO). Iso/iec 25000, software engineering - software product quality requirements and evaluation (square), 2005.

[7] Joaquín García Molina. GII_O_MA_19.07: Comparador de métricas de evolución en repositorios de software. https://github.com/Joaquin-GM/GII_O_MA_19.07-Comparador-de-metricas-de-evolucion-en-repositorios-Software, 2022.

[8] Open Source Security Foundation. criticality$_s core$: $Helpsidentifycriticalopensourceprojects.$, 2024.

Claudia Raibulet and Francesca Arcelli Fontana. Collaborative and teamwork software development in an undergraduate software engineering course. *Journal of Systems and Software*, 144:409–422, 2018.

Paul Roe and Tony Richards. Applying kanban in software engineering: A case study. *Journal of Software Engineering Research and Development*, 5(1):12–25, 2017.

Jeff Sutherland. *Scrum: The Art of Doing Twice the Work in Half the Time.* Currency, 2014.