UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Gºen Ingeniería en Informática

Trabajo final del GºIng.Informática:

Evalúa y compara la actividad del repositorio en GitHub

Presentado por Roberto Luquero Peñacoba
en junio de 2019
Tutor Carlos López Nozal

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Gºen Ingeniería en Informática

D. Carlos López Nozal, profesor del departamento de Ingeniería Civil, área de Lenguajes y Sistemas Informáticos.

Expone:

Que el alumno D. Roberto Luquero Peñacoba, con DNI 45573812W, ha realizado el Trabajo final del GºIng.Informática titulado: Evalúa y compara la actividad del repositorio en GitHub.

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 a 6 de junio de 2019

Carlos López Nozal

Índice de contenido

Índice de ilustraciones

Índice de tablas

Resumen

Vivimos en una época en la que la cantidad de proyectos e información en plataformas como GitHub es inmensa y puede ser utilizada para poder obtener una idea aproximada de la situación del trabajo que uno mismo está llevando a cabo.

Gracias a diferentes herramientas, podemos obtener esa información y tratarla, para obtener una idea general de las estadísticas que rodean al trabajo alojado en un repositorio. La motivación de este proyecto es dar respuesta a no sólo el cuándo se ha desarrollado un proyecto sino también al cómo.

Para ello, se utilizan métricas de evolución del software, que nos muestran valores sobre la colaboración de todos los autores o cómo se ha modificado el proyecto a lo largo del tiempo.

En resumen, este trabajo tiene como propósito el poder realizar comparaciones entre distintos proyectos, para obtener una medida aproximada de la actividad que estamos teniendo en nuestro proyecto con respecto a otros. Para ello, se utilizará Activiti-Api como base, un proyecto desarrollado por David Blanco Alonso que nos permite obtener determinadas métricas que se usarán en las comparaciones y GitHub como repositorio donde alojar nuestro trabajo.

Descriptores

Repositorio GitHub, analizar desarrollo, comparar repositorios, aplicación de escritorio.

Abstract

We live in a time in which the huge amount of information and projects in repositories such as GitHub is enormous and can be used to get an approximate idea about the situation of the work we are doing.

Thanks to different tools, we can get that information and use it to get a general idea about the statistics surrounding the project hosted in a repository. The motivation of this project is to answer not only when but also how has a project been developed.

For that task, software evolution metrics are used to get information about the colaboration of all the authors or how the project has been modified during the time it has been in development.

In summary, this project has the purpose of allowing the comparisons of different projects to obtain an approximate average of the activity we are having in our project compared to others. To achieve all of this, Activiti-Api, a project developed by David Blanco Alonso that allows us to get some metrics used in comparisons, will be used as a base and GitHub will be used as the repository to host our project.

Keywords

GitHub repository, analize development, compare repositories, desktop aplication.

I - *INTRODUCCIÓN*

Cada día existen más personas o empresas que deciden comenzar un nuevo proyecto, ya sea para uso personal o comercial, y que pasan las horas desarrollando sin prestar atención a una de las partes más importantes de un proyecto: la calidad del desarrollo.

Puede parecer a primera vista para mucha gente que el desarrollo de un proyecto no tiene mayor importancia que la de conseguir que el código del producto vaya avanzando y deciden no dedicar la atención o los recursos necesarios a mejorar el desarrollo del mismo. La realidad es que esta falta de cuidado en el proceso de desarrollo se ve reflejada en el producto final, que puede presentar un descenso de la calidad general debido a una mala gestión del desarrollo.

Existe una gran cantidad de aplicaciones dedicadas a analizar la calidad del código desarrollado, pero pocos que se dediquen a analizar el proceso de desarrollo del mismo, por lo que este TFG tiene como objetivo el proporcionar una forma de obtener medidas que sirvan para evaluar y, como consecuencia, poder mejorar las prácticas llevadas a cabo en el desarrollo de un proyecto alojado en GitHub.

1. Esquema

En este documento voy a exponer la información más relevante obtenida durante la realización del proyecto que he mencionado anteriormente.

Para ello, desglosaré el trabajo realizado o la información necesaria en los siguientes apartados:

- **Introducción:** pretende presentar las ideas y motivaciones del proyecto.
- **Objetivos:** donde se detallan las tareas y objetivos que se han llevado a cabo y el fin de las mismas.
- **Conceptos teóricos:** explica los distintos aspectos teóricos necesarios para la realización o comprensión del proyecto.
- **Técnicas y herramientas:** presenta las herramientas utilizadas y su funcionamiento, así como los aspectos relevantes de las técnicas utilizadas.
- **Aspectos relevantes del proyecto:** incluye los aspectos que han sido los más destacados dentro de la realización del proyecto, tanto aquellos que han presentado una gran dificultad como los que considero más importantes.
- **Trabajos relacionados:** muestra una serie de trabajos previos a este, que guardan algún

tipo de relación y han servido para la realización del mismo.

- **Conclusiones:** contiene un resumen de los objetivos y conocimientos alcanzados mediante la realización de este proyecto.

Junto a la memoria se presentan los siguientes anexos:

- **Plan de proyecto software:** que contiene la planificación temporal.
- **Especificación de requisitos:** contiene los objetivos generales, el catálogo de requisitos del sistema y los requisitos funcionales y no funcionales presentes en la aplicación.
- **Especificación de diseño:** contiene las decisiones tomadas sobre el diseño del proyecto.
- **Documentación técnica del programador:** que contiene los datos más relevantes con respecto al código.
- **Documentación de usuario:** que contiene los pasos necesarios para realizar cada requerimiento funcional correctamente.

II - Objetivos del proyecto

El proyecto tiene como objetivo fundamental la mejora del software Activiti-Api, desarrollado por David Blanco Alonso en un TFG anterior.

1. Objetivos generales

El desglose de la mejora puede dividirse en dos partes, el propio código y el aspecto gráfico. En el código las mejoras son las siguientes:

- Refactorización del código y mejora del mantenimiento del proyecto.
- Añadir la funcionalidad de comparar un proyecto con los guardados en la base de datos mediante percentiles.

Y la parte gráfica consta de:

- Eliminar completamente la interfaz anterior, hecha con Java Swing, y crear una totalmente nueva utilizando Java FX.
- Mejorar la facilidad de uso de la aplicación, para que sea más intuitiva.

2. Objetivos técnicos

- Aplicar un patrón de diseño asistente (wizard) que permita mejorar la facilidad de uso de la aplicación.
- Utilizar Git como sistema de control de versiones mediante la plataforma GitHub.
- Utilizar herramientas que permitan la integración, testeo y despliegue continuos: Adobe Ant, Travis CI, Codacy, Junit y JaCoCo.
- Aplicar metodología ágil SCRUM, utilizando ZenHub y GitHub para su gestión.
- Aumentar la cantidad de test realizados sobre el código para probar las funcionalidades.
- Programar un lector capaz de extraer y guardar datos en archivos .csv.

3. Objetivos personales

- Aprender a desarrollar interfaces mediante JavaFX.
- Afianzar y aplicar los conocimientos sobre las metodologías ágiles (en concreto SCRUM).
- Afianzar y aplicar los conocimientos sobre el control de versiones Git mediante la plataforma GitHub.
- Afianzar y aplicar los conocimientos sobre la integración, testeo y despliegue continuos.
- Aplicar los conocimientos aprendidos en varias asignaturas.

III - Conceptos teóricos

1. Integración continua

La integración continua es una práctica del desarrollo del software en la cual los desarrolladores de un proyecto juntan los cambios en un repositorio común, en este caso GitHub, tras lo cual se compilan las versiones, en caso de que exista más de una, y se realizan los distintos test para comprobar si existen errores en cada una de ellas, para poder generar reportes luego. [1]

Los objetivos clave de la integración continua consisten en la posibilidad de detectar y solventar errores con gran rapidez, mejorar la calidad del software y código desarrollado y, en general, reducir el tiempo que se tarda en validar, publicar nuevas versiones o probar el código, debido a la automatización de todo el proceso. [2]

Nació como solución a la forma de trabajar que existía anteriormente, la cual consistía en que cada desarrollador trabajase en privado y únicamente se combinasen todas las partes al finalizar el proyecto, lo que tenía como resultado una gran cantidad de errores y problemas de incompatibilidad que producían un retardo considerable en las actualizaciones de las versiones para los clientes.

Es por ello que a menudo la integración continua está asociada con las metodologías de programación extrema o desarrollo ágil.

En este proyecto se ha seguido una integración continua compuesta por:

- Compilación continua del código desarrollado y subido mediante un commit a GitHub.
- Pruebas continuas que se han compilado y ejecutado sobre cada cambio para comprobar el correcto funcionamiento del proyecto en cada commit.
- Calidad y mejora continua, refactorizando el código inicial para una mejor calidad (gracias a la ayuda de Codacy) y facilitar el desarrollo de futuras funciones, siempre manteniendo la funcionalidad del proyecto.

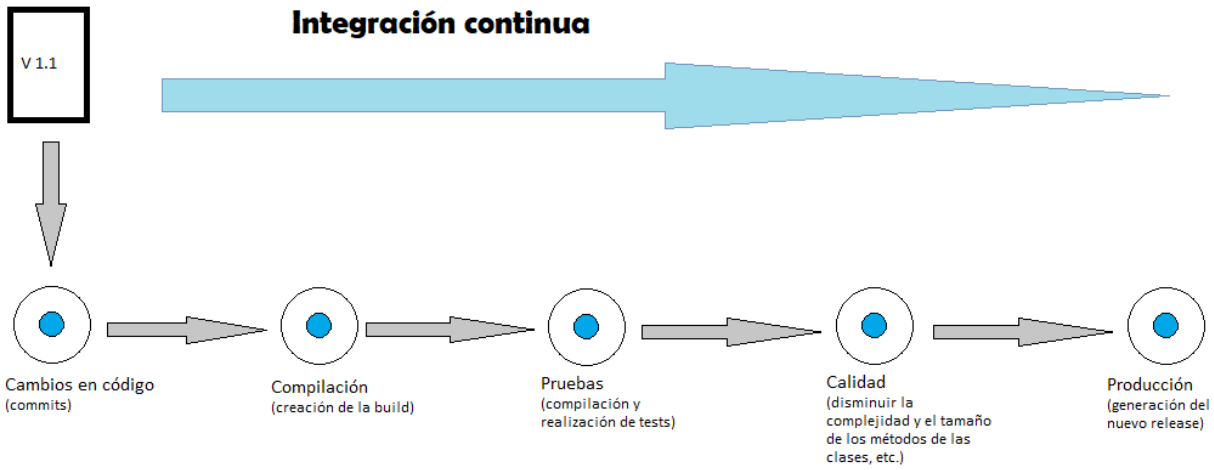


Ilustración 1: Proceso de integración continua llevado a cabo en el proyecto.

2. Calidad del software y medidas de calidad

Todo proyecto tiene como objetivo desarrollar software de la mayor calidad, que cumpla con los requisitos y expectativas de los clientes a los que va dirigido, o que incluso llegue a superarlos. [3]

La calidad del software se considera entonces como las características propias del software que se quieren controlar y asegurar, así como la aptitud del software para poder satisfacer las necesidades de los clientes.

Para medir la calidad se utilizan las medidas o indicadores, constituidos como una métrica o un conjunto de éstas, que proporcionan una indicación cuantitativa acerca de la extensión, la cantidad, la capacidad u otras cualidades del software. [4]

Para definir esas métricas, se ha de tener en cuenta que estén caracterizadas de forma efectiva y que se validen para probar su valor. Para definirlas, se ha de determinar un rango de valores significativo y que la métrica se adapte a ese valor. Además, hay que tener en cuenta un conjunto de factores:

- Debe ser relativamente fácil calcular la métrica y no debe exigir demasiado tiempo o esfuerzo.
- Las métricas deben ser intuitivas y guardar relación con el atributo que se está midiendo.
- También han de presentar resultados que no permitan la ambigüedad.
- El cálculo matemático de las métricas, en caso de que lo haya, no debe emplear medidas que den como consecuencia unidades extrañas.
- Las métricas deben ayudar a conseguir un producto final de alta calidad.

En el proyecto se ha tratado de mantener la funcionalidad inicial en todo momento, tratando de mejorar la facilidad de uso y el aspecto visual, así como incluyendo mejoras en la herramienta.

En cuanto a las métricas, en el proyecto se han incluido únicamente métricas de proceso, es decir, aquellas que nos permiten conocer el proceso realizado para la creación del proyecto, poder medirlo y obtener conclusiones para minimizar el tiempo de desarrollo o bien mejorar determinado aspecto del proceso técnico que de lugar a pérdidas de calidad del proyecto final.

Los valores umbrales para la obtención de conclusiones son el primer cuartil como valor mínimo y el tercer cuartil como el máximo. Dependiendo de las métricas, se utilizan distintas comparaciones:

- En aquellas que se recomiendo superar el primer cuartil, los valores inferiores al mismo serán tomados como malos, los valores iguales que el primer cuartil serán tomados como valor intermedio y finalmente los mayores como buenos.
- Finalmente, si necesitamos que estén entre ambos valores, únicamente el rango entre el primer y tercer cuartil serán considerados buenos, los valores exactamente iguales al primer o tercer cuartil como intermedios y el resto de valores serán negativos.

3. Pruebas de software

Las pruebas de software son una actividad del proceso de control de calidad. El objetivo de la realización de estas pruebas es proporcionar información objetiva sobre la calidad del producto o el encontrar posibles errores que existan en el código. [5]

Su clasificación depende del punto de vista, por lo que voy a exponer los puntos de vista de las pruebas utilizadas. Desde el punto de vista de la posibilidad de ejecución de código, he utilizado:

- Dinámicas, si ejecutan el código desarrollado.

Atendiendo al tipo de ejecución nos encontramos con otros dos tipos que he utilizado indistintamente:

- Manuales, si dependen del factor humano para poder ejecutarlas.
- Automáticas, si se pueden realizar sin que una persona se encargue de ejecutarlas.

Finalmente, dependiendo de lo que comprueben, tenemos las pruebas funcionales con test como los unitarios, de componentes, de integración, etc. y las no funcionales, con test como los de compatibilidad, de seguridad, de stress, etc.

4. Refactorizaciones

La refactorización es un proceso utilizado en la ingeniería del software, que consiste en modificar los elementos internos de la programación para asegurar una mejor mantenibilidad, reducir los defectos del código, aumentar la consistencia o presentar mayor claridad. Todo ello se realiza sin variar la funcionalidad externa del software. [6]

A lo largo del desarrollo de este proyecto he intentado centrar las refactorizaciones en mejorar la mantenibilidad del código existente, en reducir los code smell, por ejemplo los métodos muy largos, en eliminar la duplicidad del código y tratar de reducir todo lo posible la complicidad del mismo.

5. Métricas de evolución

En el punto Calidad del software y medidas de calidad, explico que son las métricas y para que se utilizan. En concreto, las métricas utilizadas en este proyecto han sido las métricas de evolución o métricas de medición del desarrollo de un proyecto.

Las métricas utilizadas se han obtenido de las métricas existentes en la tesis de Jacek Ratzinger, sPACE Software Project Assessment in the Course of Evolution, centrándonos en aquellas que podemos obtener del repositorio y que nos presentan información relevante sobre el desarrollo del mismo. [7]

Las seleccionadas se dividen en varios grupos y en cada una de ellas voy a adjuntar la una tabla con la descripción de la métrica en formato ISO 9126:

5.1. Equipo de desarrollo

Estas métricas permiten conocer qué personas trabajan en el desarrollo del proyecto, así como su grado de actividad en el mismo.

- **CambioPorAutor:** indica la cantidad de cambios que ha realizado cada uno de los auto-

res.

CambioPorAutor

Propósito	¿Cuántos commits ha realizado cada usuario?
Formula	CA cambio por autor
Interpretación	CA > 0 mejor valores altos
Tipo de escala	Absoluta
Tipo de medida:	CA contador
Fuente de medición	Repositorio GitHub de un proyecto

Tabla 1: Métrica CambioPorAutor según el estándar ISO 9126.

- **ContadorAutor:** indica el número total de autores en el proyecto normalizado sobre el total de cambios.

ContadorAutor

Propósito	¿Cuál es la relación entre el número de desarrolladores y el número de cambios del proyecto?
Formula	CA = NA (Numero autores) / NC (Numero cambios)
Interpretación	: CA > 0 mejor valores bajos
Tipo de escala	Absoluta
Tipo de medida	NA contador, NC contador
Fuente de medición	Repositorio GitHub de un proyecto

Tabla 2: Métrica ContadorAutor según el estándar ISO 9126.

- **IssuesPorAutor:** indica el número de issues que ha creado cada autor.

IssuesPorAutor

Propósito	¿Cuántos issues ha realizado cada usuario?
Formula	IPA issues por usuario
Interpretación	IPA >= 0 mejor valores altos
Tipo de escala	Absoluta
Tipo de medida	IPA contador
Fuente de medición	Repositorio GitHub de un proyecto

Tabla 3: Métrica IssuesPorAutor según el estándar ISO 9126.

5.2. Proceso de orientación

Agrupa las métricas que muestran el camino seguido por el equipo de trabajo con respecto a los issues del proyecto. Presentan alguna variación con las definidas en la tesis, puesto que las tareas declaradas en GitHub no pueden clasificarse mediante prioridades.

- **ContadorTareas:** indica el número total de tareas normalizado sobre el total de cambios.

ContadorTareas

Propósito	¿Cuál es el volumen medio de trabajo de las tareas?
Formula	CT = NT (Numero de tareas) / NTC (Numero total de cambios)
Interpretación	CT >= 0 mejor valores intermedios
Tipo de escala	Ratio
Tipo de medida	NT contador, NTC contador
Fuente de medición	Repositorio GitHub de un proyecto

Tabla 4: Métrica ContadorTareas según el estándar ISO 9126.

- **MediaDiasCierre:** indica la media de días tardados en cerrar un issue.

MediaDiasCierre

Propósito	¿Cuánto se tarda de media en cerrar una issue?
Formula	MDC = D (suma de los días) / NIC (numero de issues cerradas)
Interpretación	MDC >= 0 mejor valores bajos
Tipo de escala	Ratio
Tipo de medida	D contador, NIC contador
Fuente de medición	Repositorio GitHub de un proyecto

Tabla 5: Métrica MediaDiasCierre según el estándar ISO 9126.

- **NumeroCambiosSinMensaje:** indica el número de cambios del proyecto en los que no se ha escrito ningún mensaje que indique los cambios realizados.

NumeroCambiosSinMensaje

Propósito	¿Cuántos cambios se han realizado sin mensaje?
Formula	NCSM numero de cambios sin mensaje
Interpretación	NCSM >= 0 mejor valores bajos
Tipo de escala	Absoluta
Tipo de medida	NCSM contador
Fuente de medición	Repositorio GitHub de un proyecto

Tabla 6: Métrica NumeroCambiosSinMensaje según el estándar ISO 9126.

- **NumeroIssues:** indica el número total de issues creadas en el proyecto.

NumeroIssues

Propósito	¿Cuántas issues se han creado en el repositorio?
Formula	NI numero de issues
Interpretación	NI >= 0 mejor valores altos
Tipo de escala	Absoluta
Tipo de medida	NI contador
Fuente de medición	Repositorio GitHub de un proyecto

Tabla 7: Métrica NumeroIssues según el estándar ISO 9126

- **NumeroIssuesCerradas:** indica el número de issues creados que han sido cerrados.

NumeroIssuesCerradas

Propósito	¿Cuántas issues se han cerrado en el repositorio?
Formula	NIC numero de issues cerradas
Interpretación	NIC >= 0 mejor valores altos
Tipo de escala	Absoluta
Tipo de medida	NIC contador
Fuente de medición	Repositorio GitHub de un proyecto

Tabla 8: Métrica NumeroIssuesCerradas según el estándar ISO 9126.

- **PorcentajeIssuesCerradas:** indica el porcentaje del total de issues que han sido cerradas.

PorcentajeIssuesCerradas

Propósito	¿Proporción de issues cerradas en el repositorio en función de las creadas?
Formula	PIC = NIC (Número de issues cerradas) * 100 / NI (Número de issues)
Interpretación	0 <= PIC <= 100 mejor valores altos
Tipo de escala	Ratio
Tipo de medida	NIC contador, NI contador
Fuente de medición	Repositorio GitHub de un proyecto

Tabla 9: Métrica PorcentajeIssuesCerradas según el estándar ISO 9126.

5.3. Restricciones temporales

Diversas métricas que ayudarán a comprender el tiempo que se dedica a las acciones o cambios llevados a cabo, para poder calcular un coste si es necesario.

- **CommitPorDia:** indica el número de commits que se han realizado cada día de la semana.

CommitPorDia

Propósito	¿Cuántos commits se han realizado cada día de la semana?
Formula	CPD commits por día
Interpretación	CPD >= 0 mejor valores altos
Tipo de escala	Absoluta
Tipo de medida	CPD contador
Fuente de medición	Repositorio GitHub de un proyecto

Tabla 10: Métrica CommitPorDia según el estándar ISO 9126.

- **CommitPorMes:** indica el número de commits que se han realizado cada mes.

CommitPorMes

Propósito	¿Cuántos commits se han realizado cada mes?
Formula	CPM commits por mes
Interpretación	CPM >= 0 mejor valores altos
Tipo de escala	Absoluta
Tipo de medida	CPM contador
Fuente de medición	Repositorio GitHub de un proyecto

Tabla 11: Métrica CommitPorMes según el estándar ISO 9126.

- **ContadorCambiosPico:** indica el número de cambios realizados en el mes con mayor número de ellos, normalizado sobre el total de cambios.

ContadorCambiosPico

Propósito	¿Cuál es la proporción de trabajo realizado en el mes con mayor número de cambios?
Formula	CCP = NCMP (Número de cambios en el Mes Pico) / NTC (Número total de cambios)
Interpretación	0 <= CCP <= 1 Mejor valores intermedios
Tipo de escala	Ratio
Tipo de medida	NCMP contador, NTC contador
Fuente de medición	Repositorio GitHub de un proyecto

Tabla 12: Métrica ContadorCambiosPico según el estándar ISO 9126.

- **DiasPrimerUltimoCommit:** indica los días trascurridos entre el primer commit y el último.

DiasPrimerUltimoCommit

Propósito	¿Cuantos dias han pasado entre el primer y el ultimo commit?
Formula	DPUC dias pasados
Interpretación	DPUC >= 0 mejor valores altos
Tipo de escala	Absoluta
Tipo de medida	DPUC contador
Fuente de medición	Repositorio GitHub de un proyecto

Tabla 13: Métrica DiasPrimerUltimoCommit según el estándar ISO 9126.

- **MediaDiasCambio:** indica el número de días de media que pasan entre cada cambio.

MediaDiasCambio

Propósito	¿Cuantos días de media pasan entre cambios?
Formula	MDC = D (Días desde el primer al último cambio) / NTC (Número total de cambios)
Interpretación	MDC >= 0 mejor valores bajos
Tipo de escala	Absoluta
Tipo de medida	D contador, NTC contador
Fuente de medición	Repositorio GitHub de un proyecto

Tabla 14: Métrica MediaDiasCambio según el estándar ISO 9126.

- **RatioActividadCambio:** indica el número total de cambios producidos entre el número de meses.

RatioActividadCambio

Propósito	¿Cuál es el número medio de cambios por mes?
Formula	RAC = (NTC = Número total de cambios) / NM (Número de meses)
Interpretación	RAC > 0 Mejor valores intermedios
Tipo de escala	Ratio
Tipo de medida	NTC contador, NM contador
Fuente de medición	Repositorio GitHub de un proyecto

Tabla 15: Métrica RatioActividadCambio según el estándar ISO 9126.

- **RelacionMesPico:** indica el mes en el que más cambios se han realizado.

RelacionMesPico

Propósito	¿Cuál es el mes en que más cambios se han realizado?
Formula	RMP mes en el que más cambios se han realizado
Interpretación	
Tipo de escala	Nominal
Tipo de medida	RMP mes
Fuente de medición	Repositorio GitHub de un proyecto

Tabla 16: Métrica RelacionMesPico según el estándar ISO 9126.

- **UltimaModificacion:** indica la última fecha en la que se realizó un commit.

UltimaModificacion

Propósito	¿Cuando se realizo el último cambio en el repositorio?
Formula	UM fecha
Interpretación	
Tipo de escala	Absoluta
Tipo de medida	UM fecha
Fuente de medición	Repositorio GitHub de un proyecto

Tabla 17: Métrica UltimaModificacion según el estándar ISO 9126.

5.4. Relevancia del proyecto

Es la última de las métricas utilizadas, plantea de forma superficial la importancia del proyecto, contando la cantidad de usuarios que han marcado el proyecto como favorito.

- **NumeroFavoritos:** indica la cantidad de usuarios que han marcado como favorito el proyecto, y por lo tanto, que están interesados en el mismo.

NumeroFavoritos

Propósito	¿Cuántos usuarios tienen marcado el proyecto como favorito?
Formula	NF favoritos
Interpretación	NF >= 0 mejor valores altos
Tipo de escala	Absoluta
Tipo de medida	NF contador
Fuente de medición	Repositorio GitHub de un proyecto

Tabla 18: Métrica NumeroFavoritos según el estándar ISO 9126.

IV - Técnicas y herramientas

En este apartado voy a mencionar las herramientas utilizadas y una breve descripción sobre ellas y el uso que se les ha dado. Además, mencionaré aquellas que se han valorado como alternativas a las utilizadas en algún momento del desarrollo del proyecto.

1. Herramientas utilizadas

1.1. Desarrollo Java

1.1.A. Eclipse Java IDE

Eclipse Java es un entorno de desarrollo integrado perteneciente a la plataforma de software Eclipse, soportada por la comunidad también llamada Eclipse. [8]

Además del IDE también proporciona herramientas y complementos necesarios para el desarrollo del proyecto, tales como un cliente Git, editor de XML o las herramientas de desarrollo Java. [9]

Eclipse Java es el único IDE utilizado en la programación de cada uno de los issues existentes en el repositorio, que ha permitido a través de la integración con algunas herramientas programar el código de tanto el proyecto como los tests o la automatización realizada con Ant.

1.2. Integración continua

1.2.A. Apache Ant

Apache Ant es una librería de Java y una herramienta de línea de comandos que suministra una serie de tareas integradas que permiten compilar, ensamblar, probar y ejecutar aplicaciones Java. [10]

Apache Ant se ha utilizado para automatizar varias partes del proceso y la integración continua llevada en él: compilación de código, compilación y ejecución de tests, envío del coverage a la herramienta Codacy, limpieza de archivos no necesarios, creación del javadoc o la generación de la distribución.

1.2.B. Codacy

Codacy es una herramienta que permite la revisión automática de código que permite a los desarrolladores ubicar errores o código problemático para poder solventarlo. [11]

Codacy se utiliza como herramienta de análisis de código, detectando errores o defectos en el proyecto que empeorasen el código o pudiesen dar lugar a problemas y poder corregirlos en Eclipse.

1.2.C. GitHub

GitHub es la plataforma de desarrollo colaborativo más grande del mundo, que permite alojar proyectos utilizando el sistema de control de versiones Git. [12]

GitHub es la plataforma utilizada para llevar un seguimiento del desarrollo del TFG mediante metodología ágil. El repositorio existente contiene cada una de las actualizaciones realizadas sobre el proyecto, para dejar documentado todo su desarrollo.

1.2.D. Travis CI

Travis es un servicio de integración continua utilizado para realizar las build y los test de proyectos ubicados en GitHub. [13]

Travis CI se ha utilizado para probar que las build sean correctas en el TFG, que puedan ser compiladas y por lo tanto que se puedan llevar a cabo el resto de acciones.

1.2.E. ZenHub

ZenHub es una aplicación web de mantenimiento que permite tratar y distribuir los distintos issues que existen en un proyecto en GitHub en diferentes paneles, para indicar el grado de progresión que lleva. Permite también todas las opciones que se dan en GitHub, pero de una forma más ordenada apoyando la metodología ágil. [14]

ZenHub se utiliza como panel de control de los issues existentes en el repositorio GitHub del TFG, añadiendo estimaciones del tiempo necesario para cada uno.

1.3. Documentación

1.3.A. Apache OpenOffice

Apache OpenOffice es una suite de oficina de código abierto, disponible en múltiples idiomas y para varias plataformas. Ofrece aplicaciones para el procesamiento de palabras, hojas de cálculo, presentaciones, etc. [15]

OpenOffice se utiliza como procesador de texto y software para la creación y presentación de la documentación del TFG, que se ha desarrollado a lo largo de la creación del proyecto.

1.3.B. Zotero

Zotero es una herramienta que permite recolectar, organizar, citar, etc. las fuentes utilizadas en investigaciones o proyectos. [16]

Zotero se ha utilizado para mantener organizadas todas las fuentes bibliográficas utilizadas en el desarrollo del TFG, tanto aquellas pertenecientes a libros con información sobre las métricas o refactorizaciones existentes en el proyecto, como las páginas web que he consultado para obtener información sobre distintos apartados.

1.4. Patrón de diseño

1.4.A. Asistente (Wizard)

El patrón wizard suele utilizarse para que cualquier tipo de usuario pueda realizar una tarea medianamente compleja sin necesitar experiencia previa.

Consisten en dividir la tarea en pasos pequeños que simplifiquen la experiencia frente a realizar toda la tarea de una sola vez. Estos pequeños pasos pueden ser distintos dependiendo de la información introducida en los pasos anteriores. [17]

El patrón wizard se ha utilizado en las distintas operaciones que ofrece la aplicación desarrollada.

1.5. Metodología

1.5.A. SCRUM

SCRUM es el nombre que se otorga a los marcos de desarrollo ágiles.

Consiste en aplicar buenas prácticas para mejorar el trabajo colaborativo y obtener el mejor resultado posible en el desarrollo de un proyecto. Aplica una metodología de desarrollo incremental, dividido en sprints en los cuales se realizan reuniones para poner en común el trabajo realizado y realizar revisiones. [18]

SCRUM se ha utilizado como metodología ágil en el desarrollo del proyecto, aunque el grupo de personas implicadas no ha sido el recomendado, puesto que únicamente ha constado de dos personas (el tutor Carlos López y yo).

1.6. Liberías

1.6.A. Apache Commons Math

Apache Commons Math es una librería que provee un framework y la implementación de estadísticas descriptivas, distribuciones de frecuencia, etc. [19]

Apache Commons Math se ha utilizado para el cálculo de los cuartiles utilizados en la compa-

ración con los proyectos de la base de datos.

1.6.B. Gson

Gson es una librería de código abierto escrita en Java que permite la serialización y deserialización de objetos Java y su guardado en la notación utilizada por Json. [20]

Gson se ha utilizado en conjunto con el core de GitHub para poder extraer los datos de las métricas.

1.6.C. Hamcrest

Hamcrest es una librería que asiste a la creación de los test presentes en Junit, lo que permite más posibilidades y una mejor lectura. [21]

Hamcrest se ha utilizado para mejorar el mantenimiento y lectura de los test del proyecto, debido a que permite realizar una mejor explicación de las tareas llevadas a cabo en cada uno y un mejor estructuramiento de los mismos.

1.6.D. JaCoCo

JaCoCo o Java Code Coverage Tools es una librería que permite conocer que partes del código de un proyecto han sido ejecutadas durante las pruebas realizadas. [22]

JaCoCo es una herramienta utilizada en el TFG para medir la cantidad total del código creado que ha sido probado mediante los tests programados con JUnit y Hamcrest.

1.6.E. JavaHelp

JavaHelp es una librería que permite crear ventanas de ayuda para las aplicaciones desarrolladas en Java de forma sencilla. Esta librería está pensada para Java Swing. [23]

JavaHelp se ha utilizado mediante la inclusión de un botón swing no visible sobre el que se realiza un click cuando se solicita la ayuda mediante un evento.

1.6.F. JUnit

Junit es un conjunto de bibliotecas utilizadas para realizar pruebas unitarias de aplicaciones Java. [24]

JUnit es la biblioteca utilizada para la implementación de todas las pruebas realizadas sobre el código del TFG que han permitido encontrar errores y corregirlos.

2. Herramientas valoradas

2.1. Documentación

2.1.A. LaTeX

LaTeX es un sistema de composición de textos orientado a la creación de documentos escritos. Gracias a sus características y a las posibilidades que ofrece, es habitualmente usado en la creación de artículos y libros científicos. [25]

LaTeX ha sido valorado como opción frente a OpenOffice, pero la falta de conocimientos sobre el mismo habría supuesto el ralentizar la generación de la documentación.

2.2. Motor

2.2.A. GitStats

GitStats es un generador de estadísticas para repositorios Git. Revisa el repositorio y genera estadísticas de su historial en formato HTML. [26]

GitStats ha sido valorado como opción para realizar el TFG en vez de utilizar Activiti-Api, que contaba con un motor para obtener las métricas, por lo que ha supuesto una ventaja en ese aspecto.

2.3. Web frente a escritorio

2.3.A. Vaadin

Vaadin es una plataforma que permite el desarrollo de aplicaciones web mediante el uso de componentes web y un framework para aplicaciones Java. Permite la implementación de HTML5 utilizando el lenguaje Java. [27]

Vaadin ha sido valorado como opción para realizar el TFG en formato web frente al camino tomado que ha sido el realizar luna aplicación de escritorio. La motivación ha consistido principalmente en el desconocimiento inicial de los sistemas distribuidos y la propia plataforma.

V - Aspectos relevantes del desarrollo del proyecto

Este apartado recoge las distintas motivaciones a la hora de tomar cada decisión en el proyecto, tanto las tomadas con respecto a las herramientas a utilizar como

1. Selección del proyecto

En la fase de selección de proyecto decidí consultar a Carlos si tenía algún TFG disponible debido principalmente al interés que me ha generado la asignatura Desarrollo Avanzado de Sistemas Software, en la que se tiene como objetivo el enseñar buenas prácticas para mejorar la mantenibilidad y calidad del código desarrollado, un tema sobre el que ha crecido mi interés en gran medida con el paso de los cursos.

Se ofrecieron varias opciones, de entre las que se escogió la aplicada en este proyecto: el análisis del desarrollo de proyectos mediante el uso de métricas.

2. Metodologías utilizadas

Una vez conocida la idea sobre la que iba a tratar el proyecto, se decidió mantener una metodología ágil durante el desarrollo, de entre las cuales, SCRUM fue la elegida. El motivo fue poner en práctica esta metodología utilizada tan ampliamente en el ámbito laboral, para familiarizarse con ella. Es cierto que SCRUM se aplica en pequeños grupos de trabajo, pero debido a que el proyecto ha de ser de carácter personal, únicamente hemos intervenido dos personas, el tutor Carlos López y yo.

Las características generales de la metodología seguidas han sido:

- Desarrollo incremental e iterativo, refactorizando el código que fuese demasiado complejo o presentase defectos que impidiesen la correcta ejecución del código y posteriormente añadiendo las nuevas funcionalidades.
- Se establecieron una serie de sprints de una duración de 15 días, en los que se realizaba una reunión para presentan los cambios realizados, buscar posibles defectos y finalmente planificar las tareas que se iban a realizar en el siguiente sprint.
- Se asignó una estimación del tiempo que llevaría cada tarea y se dio prioridad a unas sobre otras mediante ZenHub.

Se estableció la fecha del primer sprint, donde se planteó la toma de decisiones con respecto a las herramientas que se iban a utilizar para cada aspecto del proyecto, las cuales quedan reflejadas en las issues del repositorio GitHub. [28]

Finalmente, para el desarrollo de la interfaz se ha utilizado una metodología mediante ensayo y error, creando nuevos proyectos para probar las distintas posibilidades y aplicándolas al proyecto final una vez funcionaban correctamente.

3. Documentación

Una vez se tomaron las decisiones sobre las herramientas a emplear en el proyecto, tuve que informarme acerca de las mismas como queda reflejado en varias de las issues del repositorio. [29]

Documentación consultada sobre JavaFX:

- Using JavaFX UI Controls (Alla Redko de Oracle). [30]
- JavaFX Dialogs (Marco Jakob). [31]

La documentación para Zotero consistió básicamente en una sesión práctica en una de las reuniones de split con el tutor.

Documentación consultada sobre Vaadin para evaluar su uso:

- Vaadin Tutorial (Vaadin). [32]

Finalmente, la documentación sobre el cálculo de cuartiles, realizado mediante Apache Commons Math:

- Class Percentile (Apache Commons Math). [33]

4. Valoración y mantenimiento de Activiti-Api

Como se decidió realizar una aplicación de escritorio y utilizar el motor de Activiti-Api, era necesaria una valoración de los diferentes aspectos ofrecidos por Activiti-Api, el TFG presentado por David Blanco Alonso unos años atrás.

Es esa valoración se decidió mantener:

- Todas las definiciones de las métricas, así como sus valores: cadena, conjunto, entero, fecha y largo.
- Todo el motor que permite identificar cada una de las métricas.
- Y finalmente, el lector, que es el encargado de establecer una conexión y obtener los datos del repositorio GitHub a través de varias interfaces.

A pesar de mantener todas esas partes, no estaba contento con la calidad de las mismas, por lo que refactoricé parte de ellas para mejorar su mantenimiento y aumentando la cantidad de código probado mediante los test, para poder detectar errores más fácilmente (que realmente no queda reflejado en el porcentaje mostrado en el repositorio, puesto que Codacy tiene en cuenta absolutamente todos los archivos y la interfaz gráfica a la hora de calcular ese porcentaje).

motormetricas	88,9 %	463	58	521
motormetricas.csv	96,1 %	1.218	50	1.268
lector	97,5 %	1.819	47	1.866
metricas	96,9 %	1.041	33	1.074
lector.csv	91,3 %	190	18	208
percentiles	97,4 %	453	12	465
motormetricas.valores	100,0 %	192	0	192

Ilustración 2: Coverage del código probable mediante JUnit (esceptúando interfaz gráfica).

El porcentaje no probado corresponde en la mayoría al lanzamiento de excepciones (errores

de entrada de archivos, etc.).

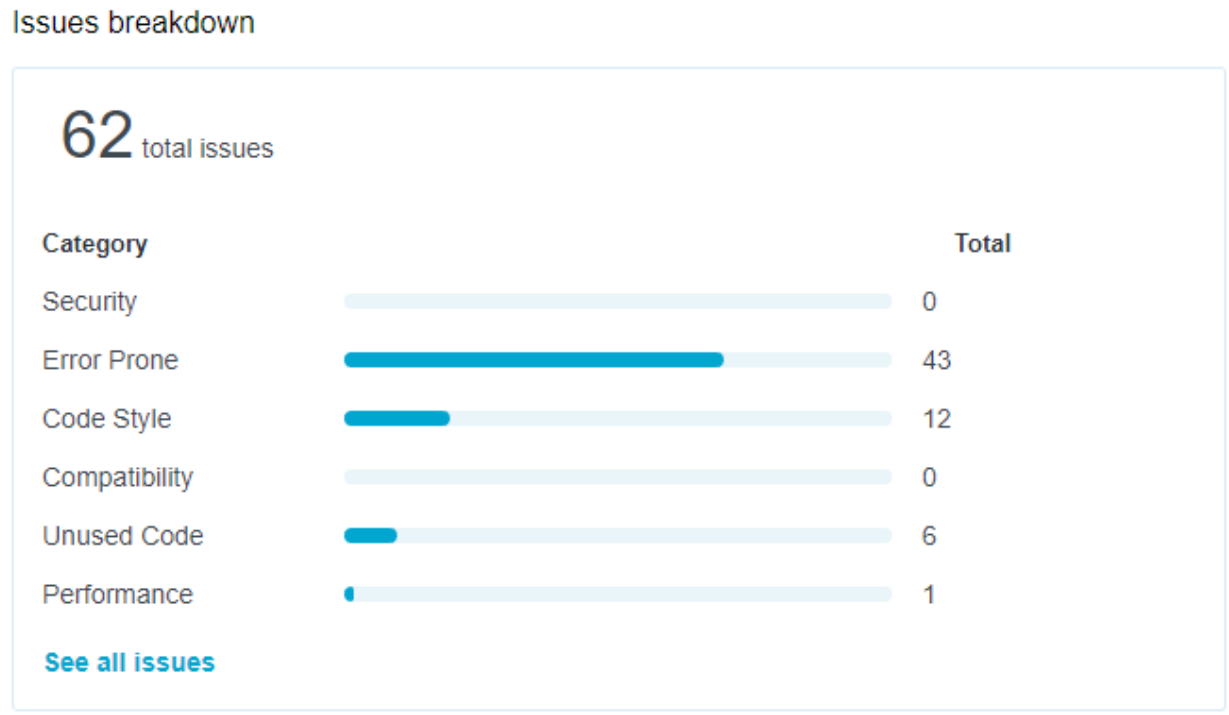


Ilustración 3: Defectos mostrados por Codacy de Activiti-Api

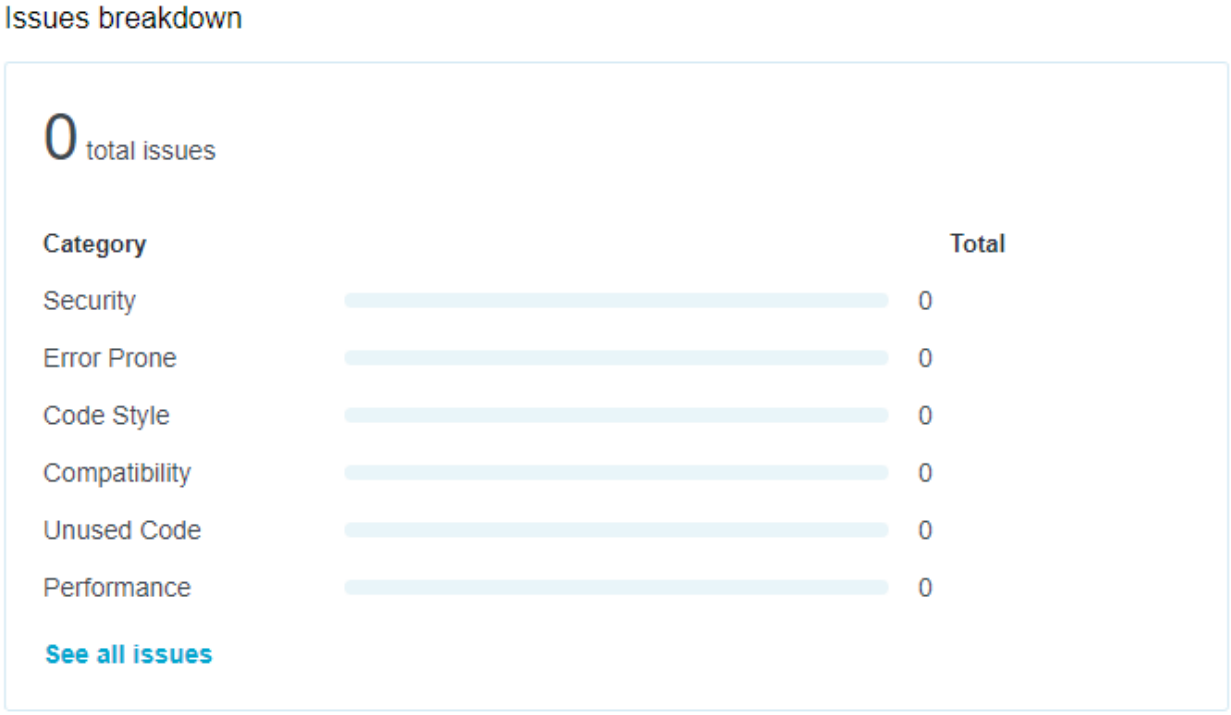


Ilustración 4: Defectos mostrados por Codacy del proyecto actualmente.

Además de lo anterior, se ha tratado de reducir al mínimo el código duplicado, la complicidad de las clases y los defectos detectados por Codacy.

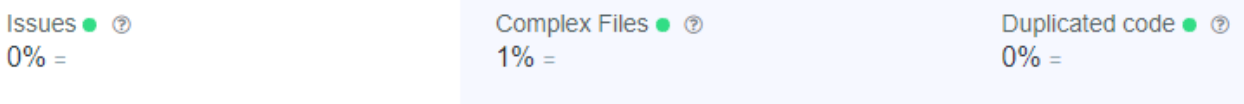


Ilustración 5: Complicidad y duplicidad del proyecto medido por Codacy.

5. Integración continua

La integración continua ha sido otro de los puntos a los más tiempo se ha dedicado durante las primeras etapas del proyecto.

Para poder integrar todas las herramientas necesarias, primero hubo que trabajar con GitHub y modificar la estructura del proyecto [34], puesto que los archivos de configuración de las herramientas necesitaban estar presentes en primer plano.

Gracias a la inclusión y configuración de Apache Ant en el proyecto, las tareas de compilar, generar la documentación javadoc, compilar los test, ejecutar los test, generar el coverage y crear la distribución de la aplicación junto con la base de datos en formato .csv de forma automática simplemente ejecutando una instrucción del fichero build.xml.

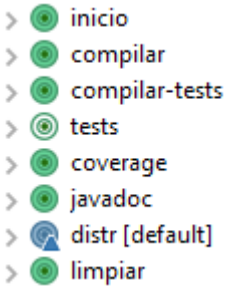


Ilustración 6: Funciones automáticas creadas con Ant

También he realizado la configuración de Travis CI y Codacy en el proyecto, lo que me ha permitido compilar cada uno de los commits subidos al repositorio GitHub mediante Travis CI y enviar los resultados a Codacy para poder detectar defectos en el código que más tarde he arreglado.

Finalmente he incluido las librerías Jubit y Hamcrest para la posible ejecución de test que prueben los diferentes aspectos del código implementado y he añadido y configurado la librería JaCoCo para poder enviar los resultados de los test a Codacy y comprobar la cobertura de código que prueban, aparte de la ya presentada por el propio Junit.

6. Desarrollo de la interfaz

Otro de los objetivos que se marcaron desde un principio fue el cambio total de la interfaz, utilizando javaFX en vez de Java Swing que es lo que había utilizado Activiti-Api.

Después de documentarnos correctamente y realizar diversas pruebas en pequeños proyectos externos poniendo en práctica lo aprendido, se decidió utilizar un patrón asistente (wizard) para las diversas operaciones que la aplicación ofrece, que aunque no presentan una dificultad elevada, hace más sencillo su uso y reduce la probabilidad de cometer errores.

Ilustración 7: Ejemplo de una de las escenas de la aplicación con patrón asistente.

En cuanto al diseño general de la interfaz, se decidió crear una interfaz simple, con colores que no llamen mucho la atención (colores con poca saturación) para que no molesten a la vista o puedan distraer de lo verdaderamente importante, que son las funcionalidades que ofrece.

En general, el cambio ha sido bastante notorio:

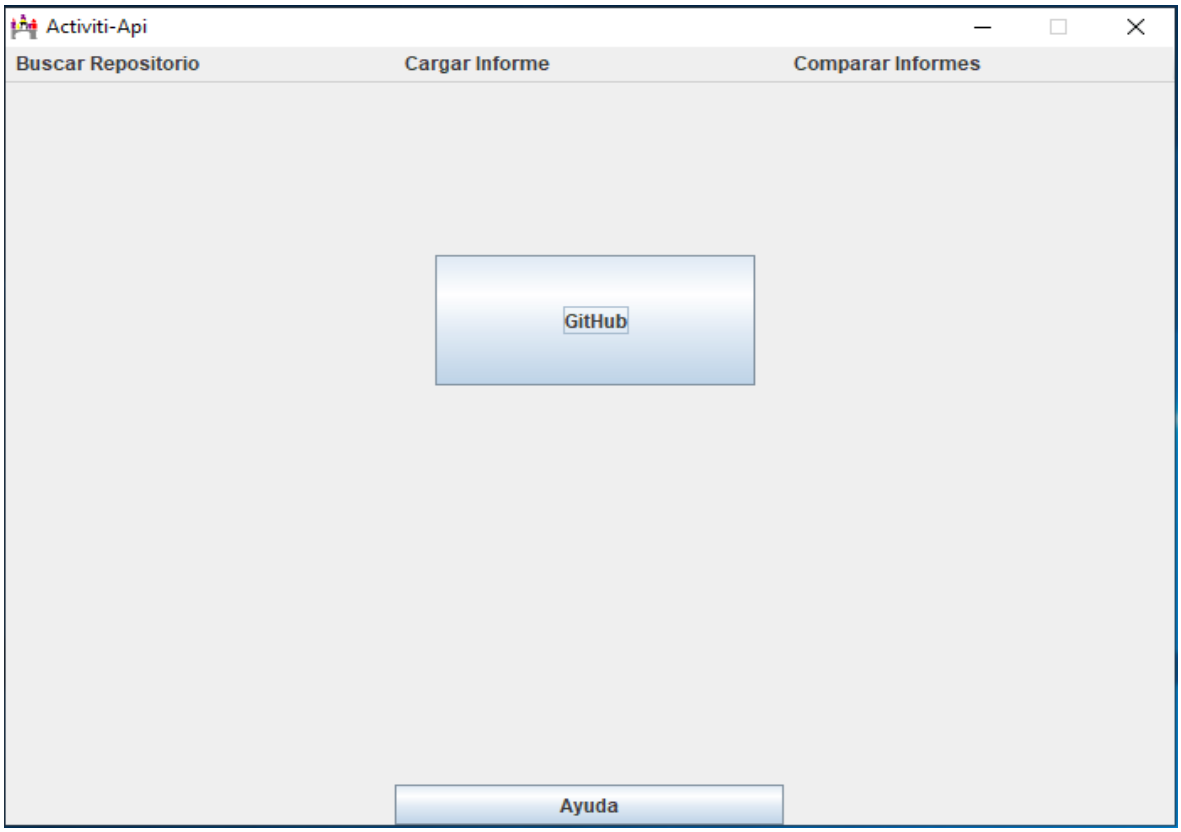


Ilustración 8: Pantalla de inicio de Activiti-Api

Ilustración 9: Pantalla de inicio de este proyecto.

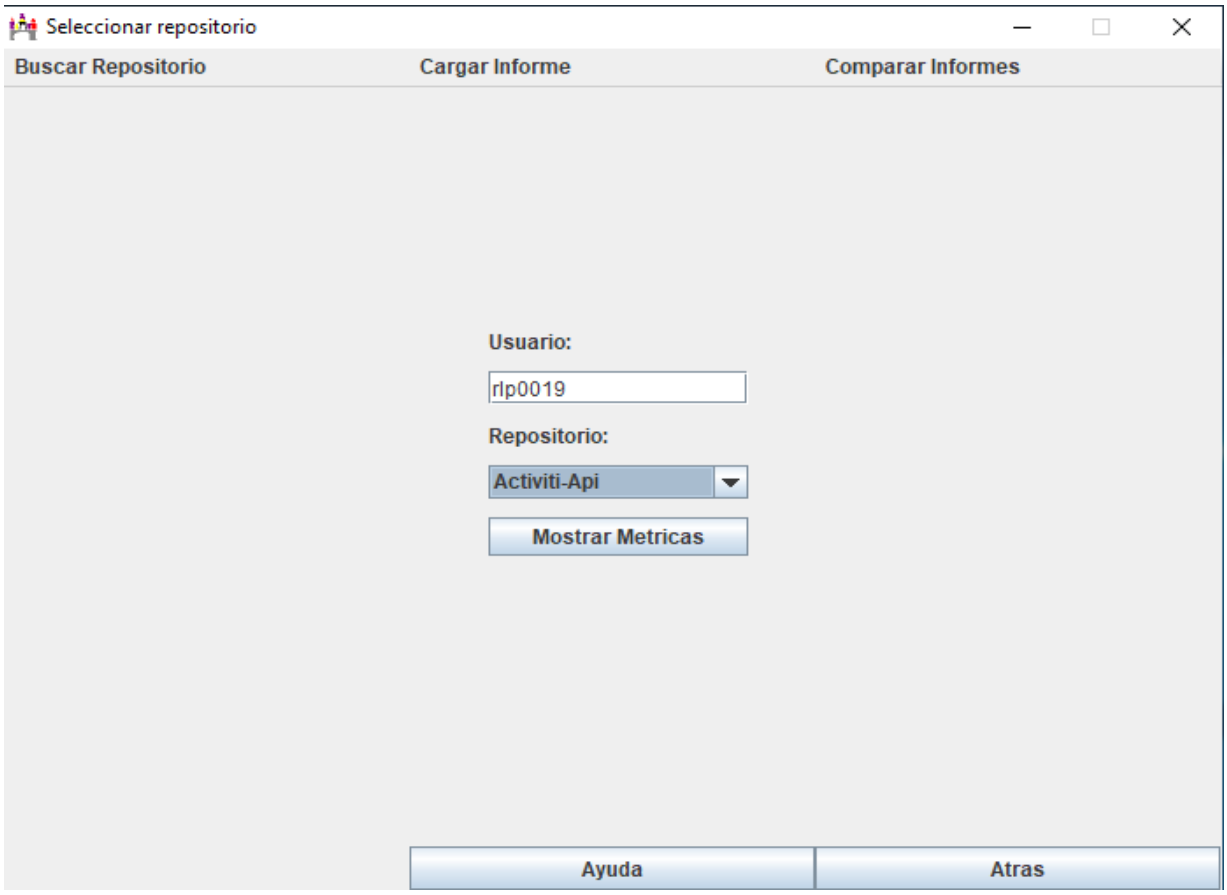


Ilustración 10: Pantalla de búsqueda de repositorio de Activiti-Api

Ilustración 11: Pantalla de búsqueda de repositorio de este proyecto.

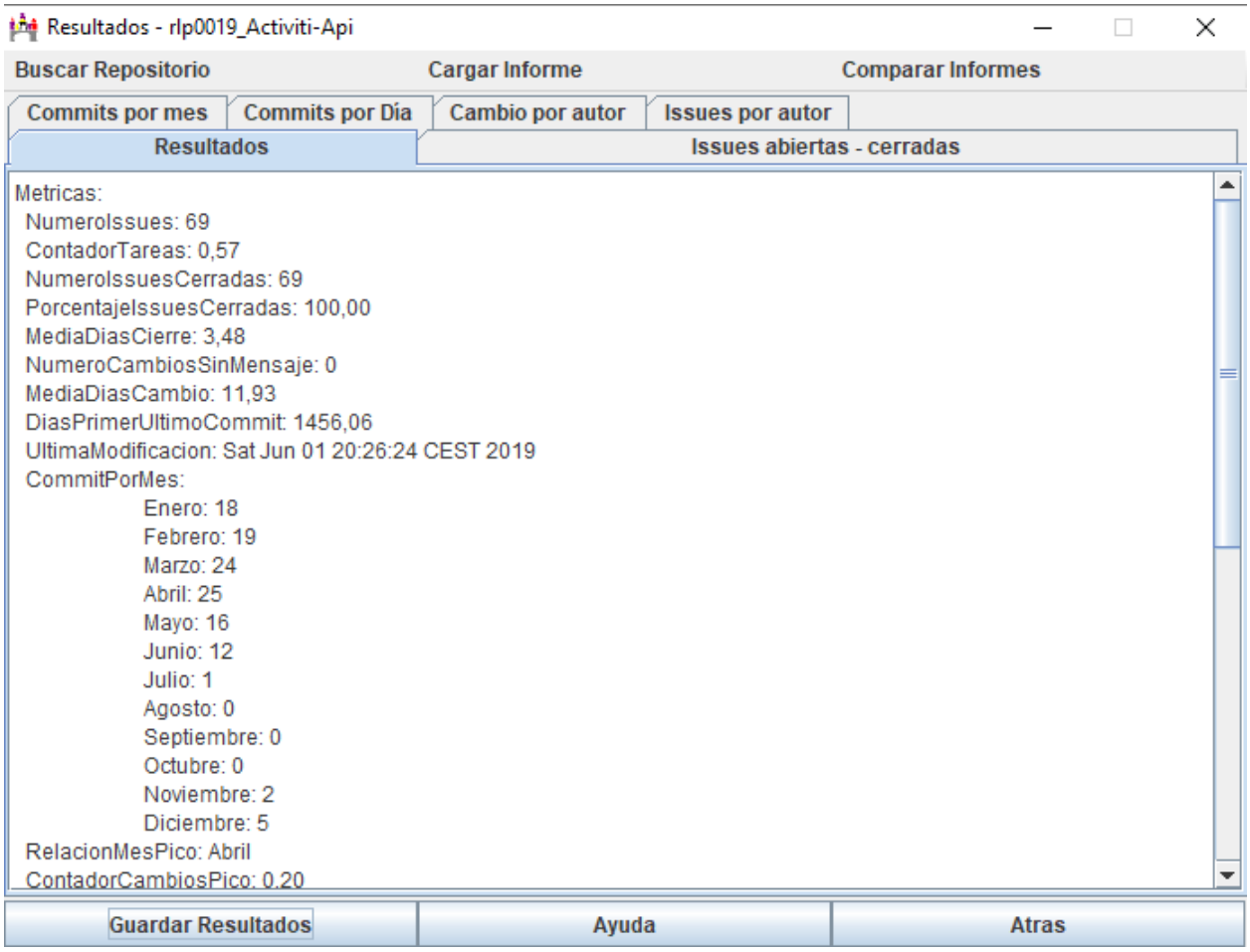


Ilustración 12: Pantalla de resultado de Activiti-Api.

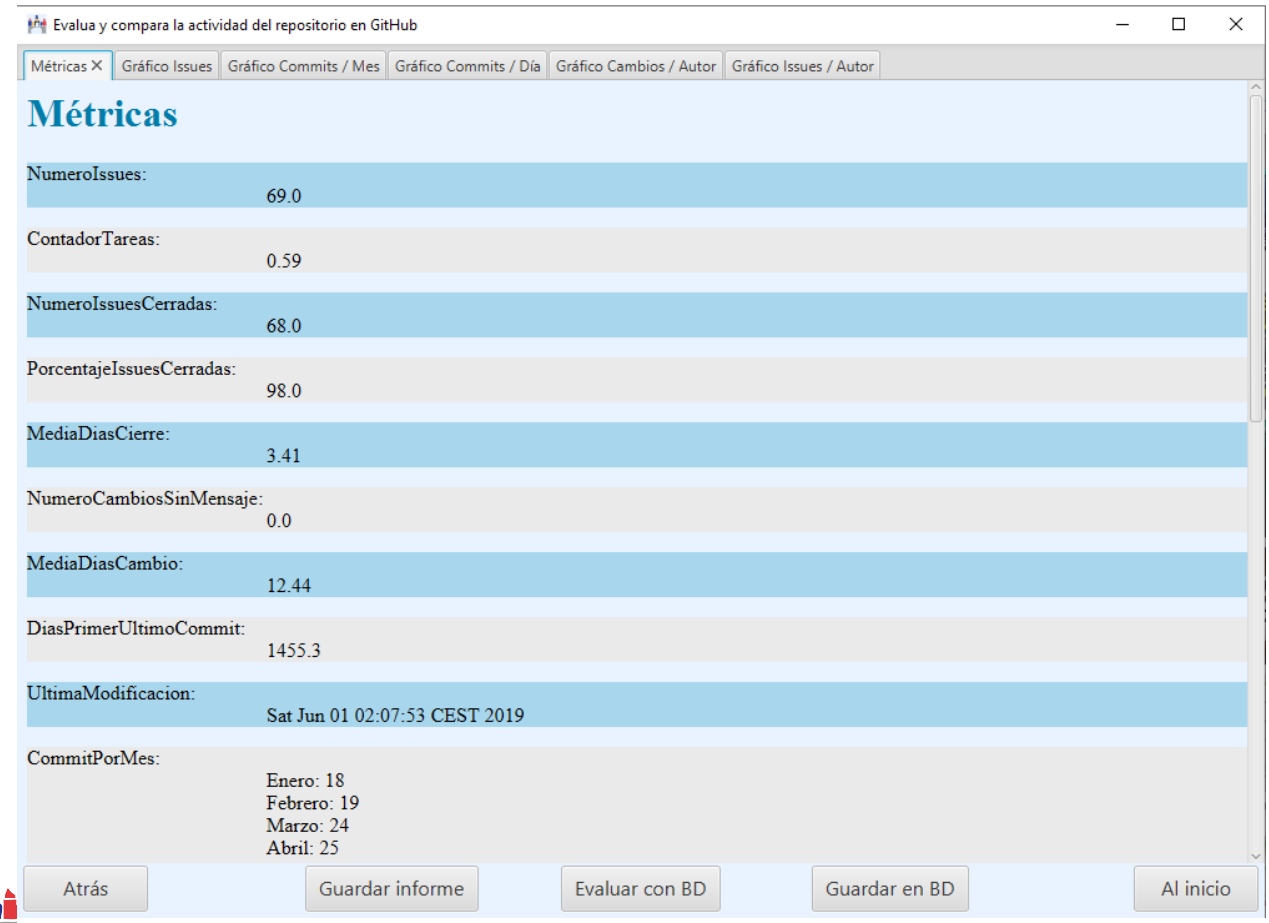
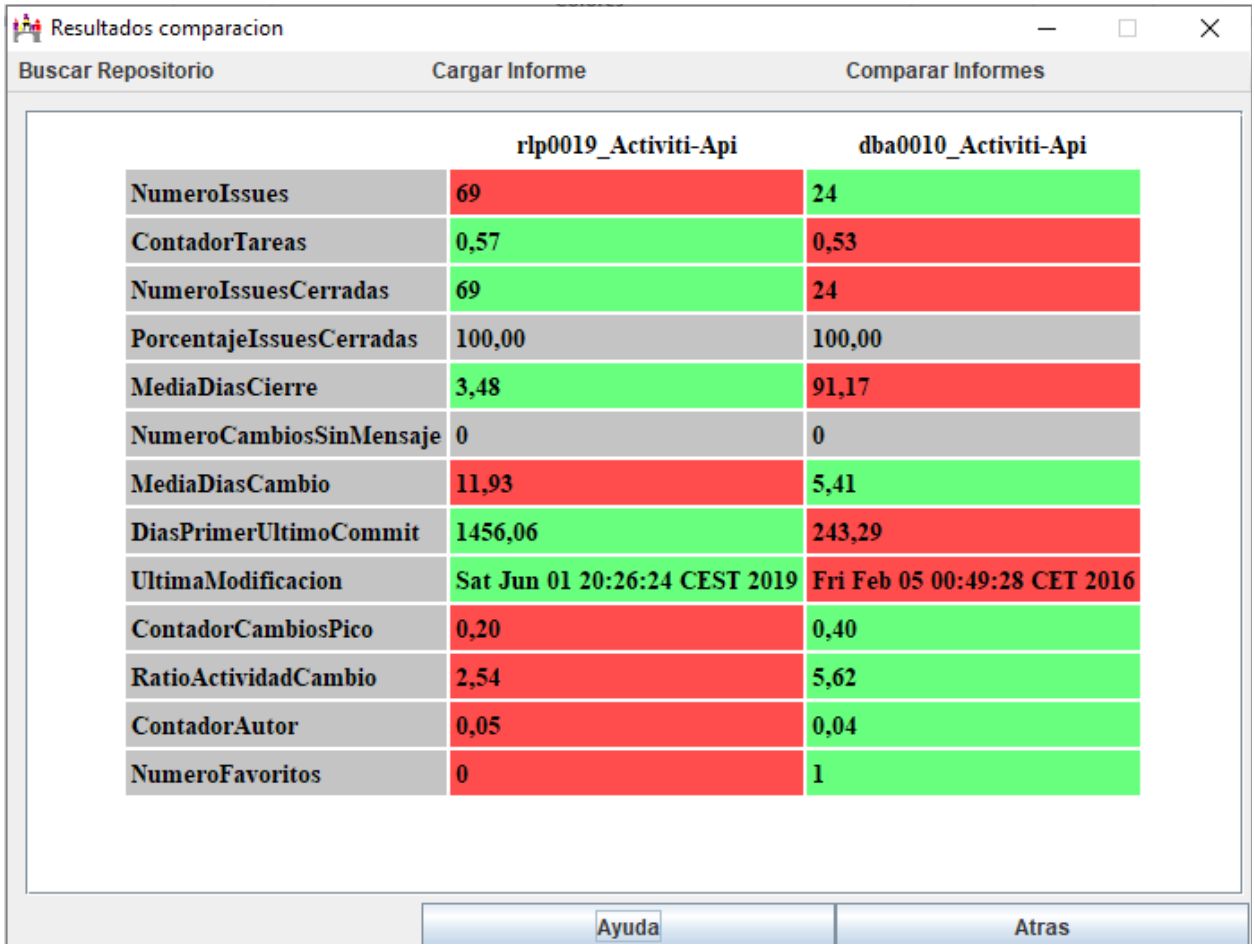
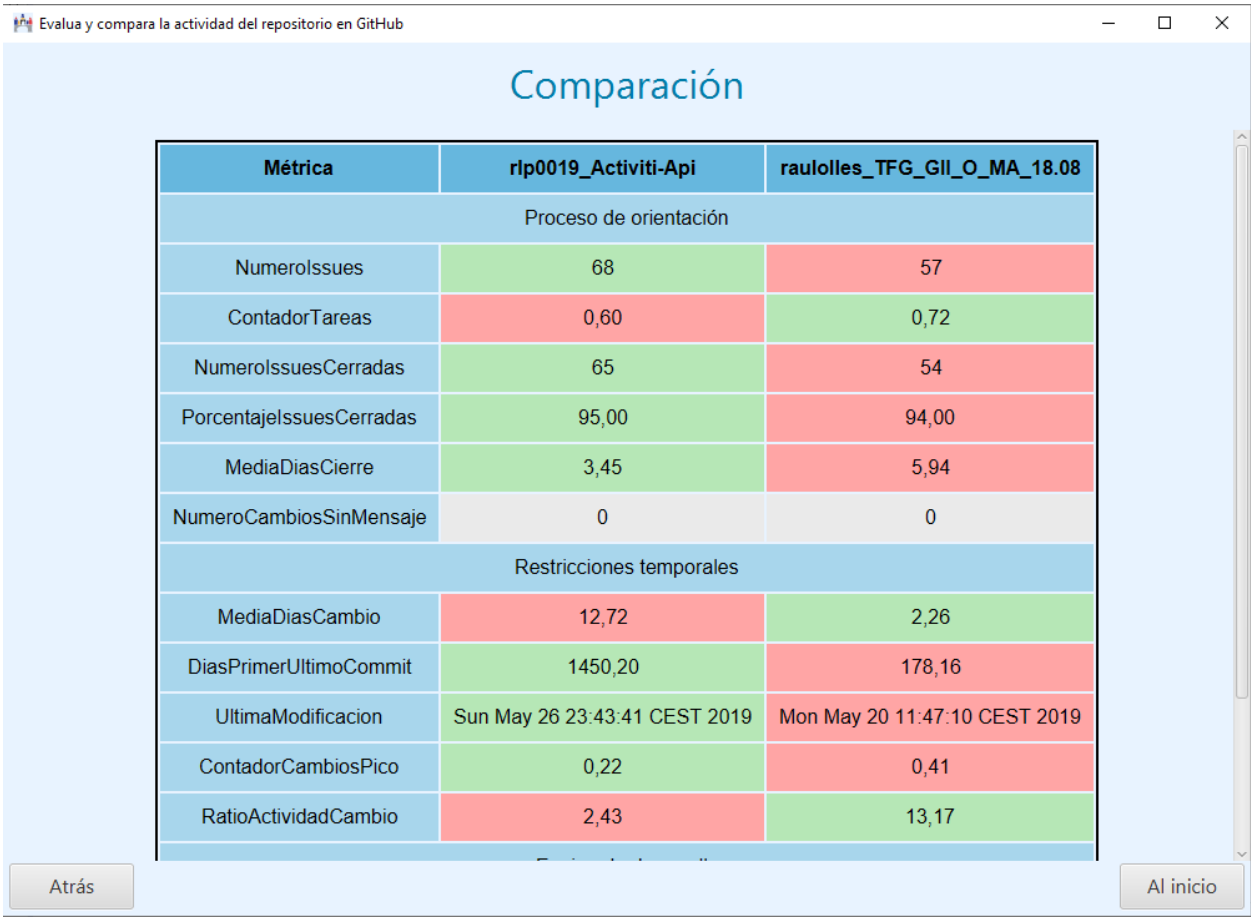


Ilustración 13: Pantalla de resultados de este proyecto.



	rlp0019_Activiti-Api	dba0010_Activiti-Api
NumeroIssues	69	24
ContadorTareas	0,57	0,53
NumeroIssuesCerradas	69	24
PorcentajeIssuesCerradas	100,00	100,00
MediaDiasCierre	3,48	91,17
NumeroCambiosSinMensaje	0	0
MediaDiasCambio	11,93	5,41
DiasPrimerUltimoCommit	1456,06	243,29
UltimaModificacion	Sat Jun 01 20:26:24 CEST 2019	Fri Feb 05 00:49:28 CET 2016
ContadorCambiosPico	0,20	0,40
RatioActividadCambio	2,54	5,62
ContadorAutor	0,05	0,04
NumeroFavoritos	0	1

Ilustración 14: Pantalla de comparación de informes de Activiti-Api.



Métrica	rlp0019_Activiti-Api	raulolles_TFG_GII_O_MA_18.08
Proceso de orientación		
NumeroIssues	68	57
ContadorTareas	0,60	0,72
NumeroIssuesCerradas	65	54
PorcentajeIssuesCerradas	95,00	94,00
MediaDiasCierre	3,45	5,94
NumeroCambiosSinMensaje	0	0
Restricciones temporales		
MediaDiasCambio	12,72	2,26
DiasPrimerUltimoCommit	1450,20	178,16
UltimaModificacion	Sun May 26 23:43:41 CEST 2019	Mon May 20 11:47:10 CEST 2019
ContadorCambiosPico	0,22	0,41
RatioActividadCambio	2,43	13,17

Ilustración 15: Pantalla de comparación de informes de estre proyecto.

7. Internacionalización

Uno de los últimos pasos tomados ha sido la internacionalización de la interfaz gráfica desarrollada.

Para ello se ha realizado un método estático en cada panel, que carga los distintos archivos de configuración con los valores de los textos a mostrar por cada elemento dependiendo del idioma que se seleccione mediante la opción del menú de la pantalla de inicio.

La ventaja que presenta esta forma de realizarlo es que el lenguaje puede cargar dinámicamente, y no es necesario reiniciar la propia aplicación para que surta efecto.

Esta forma de realizarlo presentaba un problema, los distintos tipos de alertas (Alert) se crean utilizando el idioma establecido en la máquina virtual de Java en el momento de su creación y no permiten cambiar el texto de sus botones como se desee, por lo que la forma de solucionarlo ha sido cambiando la región de la máquina virtual mediante la clase Locale, que permite cambiar también el lenguaje y volviendo a crear las propias alertas.

VI - Trabajos relacionados

Existen varios trabajos que tienen como objetivo el mismo que el de este proyecto, el proporcionar métricas o datos objetivos que puedan ser utilizados para mejorar la evolución de los proyectos.

1. Trabajos teóricos

1.1. sPACE

sPACE o Software Project Assessment in the Course of Evolution es una tesis desarrollada por Jacek Ratzinger y que ha servido para obtener las definiciones de las métricas que utiliza Activiti-Api, que permiten obtener una visión general sobre la evolución de un proyecto, su calidad o el trabajo en equipo llevado a cabo. [35]

sPACE es una tesis mucho más amplia, que también abarca el minado de datos, la refactorización, los defectos o la predicción de las acciones que se van a llevar a cabo en un proyecto software.

2. Herramientas

2.1. Activiti-Api

Activiti-Api es un proyecto que se apoya en los datos que proporciona GitHub sobre un repositorio y muestra una serie de métricas basadas en sPACE: Software Project Assessment in the Course of Evolution, una tesis llevada a cabo por Jacek Ratzinger. [36]

Activiti-Api tiene como objetivo el obtener una cuantificación del trabajo realizado, a la vez que se busca tener una visión sobre el desarrollo, su evolución y estado actual de un proyecto.

Las métricas utilizadas se dividen en 4 grupos principales:

- Medición de equipo, que indican el número de personas trabajando en el proyecto, así como una aproximación al trabajo que han realizado, y entre las que se encuentran los cambios por cada autor, los issues por autor o el contador de autores.
- El proceso de orientación, que muestran el comportamiento de los miembros del proyecto con respecto a los issues. En este grupo se muestran el número total de issues, el número de issues cerrados o la media de días que se ha tardado en cerrar un issue, entre otros.
- Restricciones temporales, donde encontraremos las medidas que indican el tiempo invertido en cada una de las acciones para poder prever una evolución del proyecto. Entre estas tenemos, por ejemplo, la media de días por cambio, la fecha de la última modificación o los días entre el primer y el último commit, etc.
- Importancia del proyecto, que únicamente tiene el número de favoritos del proyecto.

2.2. GitStats

GitStats es un generador de estadísticas sobre repositorios git. Se dedica a estudiar el repositorio y generar estadísticas interesantes de su historial en formato HTML. [37]

Los datos sobre las que genera las estadísticas son los siguientes:

- Estadísticas generales como el número total de archivos, de líneas de código, de commits o autores.
- Actividad como los commits por cada hora del día, por cada día, por cada semana, por cada hora de la semana, por cada mes del año, por cada mes y año y por cada año.
- Autores, con una lista con su nombre, su porcentaje de commit, las fechas de su primer y último commit y su edad, así como el autor del mes y el autor del año.
- Archivos y su cantidad por por fecha o las extensiones.
- Líneas de código por su fecha.

2.3. StatCVS

StatCVS es un software de código abierto que utiliza JFreeChart para generar documentos HTML que contienen tablas y gráficos que describen el desarrollo de un repositorio CVS. [38]

Algunos ejemplos de esos gráficos son:

- Las líneas de código por cada autor.
- La actividad de cada autor.
- La cantidad de ficheros.
- La media de tamaño de cada fichero, etc.

2.4. StatSVN

StatSVN es una aplicación de código abierto basada en StatCVS y que también utiliza JFreeChart para generar nuevamente documentos HTML con tablas y gráficos sobre el desarrollo de un repositorio Subversión. [39]

Los gráficos y tablas que genera son las mismas que StatCVS.

3. Ventajas y debilidades del proyecto

En cuanto a las ventajas podemos encontrar:

- Debido a que el lenguaje utilizado es Java, la aplicación funciona en los dispositivos que puedan tener el sistema instalado (probado en Windows y Linux).
- Es fácil de utilizar gracias al patrón asistente, por lo que no es necesaria un gran experiencia para poder utilizarlo.
- Presenta una gran versatilidad en la comparación con los proyectos de la base de datos, puesto que permite guardar aquellos que se deseen.
- La aplicación permite guardar los resultados en fichero de texto, para disponer de los resultados de forma local y sin tener que recurrir de nuevo a ella.

En cuanto a las desventajas:

- Aunque el proyecto presenta unos valores, éstos han de ser interpretados, y de no tener la experiencia suficiente pueden surgir confusiones.
- Aunque la interfaz del proyecto está traducida a dos idiomas: español e inglés, la ayuda únicamente está disponigle en español y las métricas están definidas en español, por lo que sus nombres no serán descriptivos para aquellos que no lo entiendan.
- La aplicación no consigue distinguir cuando terminan los commits del proyecto original y cuando empiezan los del fork (debido a que los commits realizados en el original son historia del fork también), por lo que devuelve ambos juntos, lo que presenta interferencias en las métricas temporales en los forks.
- La aplicación necesita tener acceso a internet y no es posible utilizarse sin él.

VII - Conclusiones y líneas de trabajo futuras

En esta parte de la memoria voy a tratar sobre las conclusiones obtenidas tras la realización del proyecto, así como las posibles líneas de trabajo que podría seguir el mismo.

1. Conclusiones

- El objetivo principal planteado al inicio del proyecto se ha completado. He completado una aplicación que, gracias al motor del TFG utilizado como base, extrae valores concretos sobre la evolución de un proyecto y permite compararlos con los guardados en la base de datos .csv.

- He podido aprender JavaFX y practicar el desarrollo frontend para el diseño de interfaces.

- Las herramientas y metodologías utilizadas a lo largo de la realización del proyecto son frecuentemente usadas a día de hoy en muchas empresas de software.

- Gracias a la integración continua y al uso de Ant se ha conseguido reducir el tiempo empleado en la compilación, generación de documentación y distribución del proyecto.

- La utilización de la librería Junit junto con Hamcrest ha permitido realizar tests que han ayudado tanto a la hora de detectar errores como a la hora de poder probar las distintas funciones implementadas antes de haber programado la interfaz.

2. Líneas de trabajo futuras

Existen una serie de líneas de trabajo que han surgido durante la realización del proyecto que podrían ser sobre las que se trabajase en un futuro:

- Se podrían aumentar la cantidad de repositorios soportados, ya que a día de hoy únicamente puede funcionar en GitHub.

- Traducir los nombres de las métricas al inglés podría suponer una ventaja respecto al español debido a que es más utilizado a nivel global.

- Traducir la ayuda al inglés y otros idiomas que se quisieran añadir.

- Mejorar la detección de los commits para que la aplicación logre diferenciar entre las realizadas en el proyecto original y las realizadas en el fork.

- Permitir al usuario el guardar preferencias para usar por defecto durante uso de la aplicación, como por ejemplo la conexión que se quiera utilizar o la carpeta en la que iniciar a buscar los ficheros al importarlos o guardarlos.

- Para las funciones en las que no es necesario extraer datos de un repositorio (importar un fichero y comparar dos ficheros), evaluar si es posible realizarlas sin disponer de acceso a internet.

- Finalmente, si se considerase necesario, se podría ampliar el número de lenguajes disponibles.

VIII - Bibliografía

[1] Wikipedia, “Integración continua,” *Wikipedia, la enciclopedia libre*. [Online]. Available: https://es.wikipedia.org/w/index.php?title=Integraci%C3%B3n_continua&oldid=113681060. [Accessed: 30-Mar-2019].

[2] Amazon Web Services, “Integración continua del software | Pruebas automatizadas | AWS,” *Amazon Web Services, Inc.* [Online]. Available: https://aws.amazon.com/es/devops/continuous-integration/. [Accessed: 30-Mar-2019].

[3] Wikipedia, “Calidad de software,” *Wikipedia, la enciclopedia libre*. [Online]. Available: https://es.wikipedia.org/w/index.php?title=Calidad_de_software&oldid=110582332. [Accessed: 30-Mar-2019].

[4] Carlos Macallums, “Medidas, métricas e indicadores de calidad de software - Algorítmica y Programación.” [Online]. Available: https://sites.google.com/site/portafoliocarlosmacallums/unidad-ii/medidasmetricaseindicadoresdecalidaddesoftware. [Accessed: 30-Mar-2019].

[5] Wikipedia, “Pruebas de software,” *Wikipedia, la enciclopedia libre*. [Online]. Available: https://es.wikipedia.org/w/index.php?title=Pruebas_de_software&oldid=114338983. [Accesed: 30-Mar-2019].

[6] Wikipedia, “Refactorización,” *Wikipedia, la enciclopedia libre*. [Online]. Available: https://es.wikipedia.org/w/index.php?title=Refactorizaci%C3%B3n&oldid=108259973. [Accesed: 30-Mar-2018].

[7] [35] Jacek Ratzinger, *sPACE Software Project Assessment in the Course of Evolution*. [Online]. Available: https://www.inf.usi.ch/jazayeri/docs/Thesis_Jacek_Ratzinger.pdf. [Accesed: 15-Feb-2019].

[8] Wikipedia, “Eclipse (software),” *Wikipedia, la enciclopedia libre*. [Online] Available: https://es.wikipedia.org/w/index.php?title=Eclipse_(software)&oldid=114741216. [Accesed: 30-Mar-2019].

[9] Eclipse Foundation, “Eclipse IDE for Java Developers | Eclipse Packages.” [Online]. Available: https://www.eclipse.org/downloads/packages/release/mars/r/eclipse-ide-java-developers. [Accessed: 30-Mar-2019].

[10] The Apache Software Foundation, “Apache Ant.” [Online]. Available: https://ant.apache.org/. [Accessed: 15-May-2019].

[11] XebiaLabs, “Codacy,” *XebiaLabs*. [Online]. Available: https://xebialabs.com/technology/codacy/. [Accessed: 30-Mar-2019].

[12] Wikipedia, “GitHub,” *Wikipedia, la enciclopedia libre*. [Online]. Available: https://es.wikipedia.org/w/index.php?title=GitHub&oldid=114296186. [Accesed: 30-Mar-2019].

[13] Wikipedia, “Travis CI,” *Wikipedia*. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Travis_CI&oldid=884966461. [Accesed: 30-Mar-2019].

[14] ZenHub, “ZenHub - Agile Project Management for GitHub,” *ZenHub*. [Online]. Available: https://www.zenhub.com. [Accessed: 30-Mar-2019].

[15] Apache Software Foundation, “Por qué Apache OpenOffice.” [Online]. Available: https://www.openoffice.org/es/por-que/. [Accessed: 30-Mar-2019].

[16] R. R. C. for H. and N. M. Corporation for Digital Scholarship, “Zotero | Your personal research assistant.” [Online]. Available: https://www.zotero.org/. [Accessed: 31-Mar-2019].

[17] E. Cerrillos, “Patrones de diseño de interfaz,” 19-Oct-2017. [Online]. Available: https://usersmatter.co/2017/10/19/patrones-de-diseno-de-interfaz/. [Accessed: 02-Jun-2019].

[18] K. Schwaber, *Agile Project Management with Scrum*, 1st ed. Microsoft Press, 2004. [Online]. Available: https://ptgmedia.pearsoncmg.com/images/9780735619937/samplepages/9780735619937.pdf. [Accesed: 01-Abr-2019]

[19] “Math – The Commons Math User Guide - Statistics.” [Online]. Available: http://commons.apache.org/proper/commons-math/userguide/stat.html. [Accessed: 03-Jun-2019].

[20] “Gson,” *Wikipedia, la enciclopedia libre*. [Online]. Available: https://es.wikipedia.org/w/index.php?title=Gson&oldid=115173669. [Accesed: 02-Jun-2019].

[21] Wikipedia, “Hamcrest,” *Wikipedia*. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Hamcrest&oldid=853185952. [Accesed: 30-Mar-2019].

[22] Wikipedia, “Java code coverage tools,” *Wikipedia*. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Java_code_coverage_tools&oldid=888557925. [Accesed: 30-Mar-2019].

[23] “Ejemplo sencillo con JavaHelp.” [Online]. Available: http://www.chuidiang.org/java/herramientas/javahelp/ejemplo-javahelp.php. [Accessed: 03-Jun-2019].

[24] Wikipedia, “Junit,” *Wikipedia, la enciclopedia libre*. [Online]. Available: https://es.wikipedia.org/w/index.php?title=JUnit&oldid=102263091. [Accesed: 30-Mar-2019].

[25] Wikipedia, “LaTeX,” *Wikipedia, la enciclopedia libre*. [Online]. Available: https://es.wikipedia.org/w/index.php?title=LaTeX&oldid=113802867. [Accessed: 30-Mar-2019].

[26] Heikki Hokkanen “hoxu,” “GitStats - git history statistics generator.” [Online]. Available: http://gitstats.sourceforge.net/. [Accessed: 30-Mar-2019].

[27] Wikipedia, “Vaadin,” *Wikipedia, la enciclopedia libre*. [Online]. Available: https://es.wikipedia.org/w/index.php?title=Vaadin&oldid=114714106. [Accesed: 30-Mar-2019].

[28] “Issues · rlp0019/Activiti-Api.” [Online]. Available: https://github.com/rlp0019/Activiti-Api/issues?utf8=%E2%9C%93&q=is:issue+label:%22Evaluaci%C3%B3n+alternativas%22. [Accessed: 03-Jun-2019].

[29] “Issues · rlp0019/Activiti-Api.” [Online]. Available: https://github.com/rlp0019/Activiti-Api/issues?utf8=%E2%9C%93&q=is:issue+label:Documentaci%C3%B3n. [Accessed: 03-Jun-2019].

[30] “Using JavaFX UI Controls: About This Tutorial | JavaFX 2 Tutorials and Documentation.” [Online]. Available: https://docs.oracle.com/javafx/2/ui_controls/jfxpub-ui_controls.htm. [Accessed: 03-Jun-2019].

[31] “JavaFX Dialogs (official) | code.makery.ch.” [Online]. Available: https://code.makery.ch/blog/javafx-dialogs-official/. [Accessed: 03-Jun-2019].

[32] “Vaadin Tutorial | Vaadin Framework 8 | Vaadin 8 Docs.” [Online]. Available: https://vaadin.com/docs/v8/framework/tutorial.html. [Accessed: 03-Jun-2019].

[33] “Percentile (Apache Commons Math 3.5 API).” [Online]. Available: http://commons.apache.org/proper/commons-math/javadocs/api-3.5/org/apache/commons/math3/stat/descriptive/rank/Percentile.html. [Accessed: 03-Jun-2019].

[34] “Refactorización arquitectónica de repositorio GitHub. · Issue #14 · rlp0019/Activiti-Api.” [Online]. Available: https://github.com/rlp0019/Activiti-Api/issues/14. [Accessed: 03-Jun-2019].
[36] “Activiti-Api/Monitor multiplataforma de la actividad de un proyecto.pdf at master · dba0010/Activiti-Api.” [Online]. Available: https://github.com/dba0010/Activiti-Api/blob/master/Activiti-api/Documentaci%C3%B3n/Monitor%20multiplataforma%20de%20la%20actividad%20de%20un%20proyecto.pdf. [Accessed: 03-Jun-2019].

[37] Heikki Hokkanen “hoxu,” “GitStats - git history statistics generator.” [Online]. Available: http://gitstats.sourceforge.net/. [Accessed: 30-Mar-2019].

[38] Appendium, “StatCVS - Repository Statistics - Introduction.” [Online]. Available: http://statcvs.sourceforge.net/. [Accessed: 30-Mar-2019].

[39] Appendium, “StatSVN - Repository Statistics - Introduction.” [Online]. Available: http://statsvn.org/. [Accessed: 30-Mar-2019].

[40] N. E. Fenton and S. L. Pfleeger, *Software metrics*, vol. 1. Chapman & Hall London, 1991.

[41] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts, *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 1999.

[42] M. Lippert and S. Roock, *Refactoring in Large Software Projects: Performing Complex Restructurings Successfully*. John Wiley & Sons, 2006.

[43] W. C. Wake, *Refactoring workbook*. Boston: Addison-Wesley, 2004.

Impreso en Burgos el martes, 4 de junio de 2019