UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

GII_O_MA_19.07 Comparador de métricas de evolución en repositorios Software Documentación Técnica

Presentado por Joaquín García Molina
Universidad de Burgos
9 de junio de 2022
Tutor: Carlos López Nozal

Índice general

Índice de figuras

Índice de tablas

Apéndice A

Plan de Proyecto Software

A.1. Introducción

Lo primero que se debería hacer al iniciar un proyecto y antes de comenzar con el desarrollo es la planificación del mismo. Es necesario realizar una estimación de coste en tiempo y dinero que nos va a suponer realizar el proyecto y una vez conozcamos esta estimación deducir si el proyecto se puede realizar exitósamente o no. Para realizar esta tarea en este documento vamos a realizar dos tareas:

Planificación temporal. Donde estimaremos el tiempo que requiere la realización del proyecto.

Estudio de la viabilidad. Donde se deduce si el proyecto es viable, tanto económica como legalmente:

- **Viabilidad económica.** Se estudia el coste que supondría haber realizado el proyecto en un entorno real y se analizan posibles maneras de monetizar el proyecto para cubrir este coste.
- **Viabilidad legal.** Se estudian las leyes aplicables al proyecto y los aspectos más relevantes a nivel legal.

A.2. Planificación temporal

Se ha optado por la metodología **SCRUM** [7], donde se trabaja con un ciclo de vida que se repite durante el proyecto. Trabajar con esta metodología ha significado que:

- Se ha trabajado de manera incremental y evolutiva.
- Se han trabajado en iteraciones (llamados *sprints* en la metodología **SCRUM**) de dos semanas. Estos *sprints* finalizan con una reunión entre el tutor y alumno donde:
  - Se llevaba a cabo la *sprint review*, donde se revisa el trabajo realizado durante el sprint que finaliza y se tratan los avances realizados así como los problemas con los que el alumno se ha encontrado.
  - Una segunda parte donde se lleva a cabo la *sprint planning*, donde se planifica el trabajo a realizar durante el siguiente *sprint* planteando tareas y seleccionándolas del *product backlog*, pasándolas a la pila de trabajo del *sprint* que comienza, el *sprint backlog*. Estas tareas se han registrado en el sistema de gestión de incidencias de GitHub, *GitHub Issues* [1] y el trabajo con *sprints* se ha llevado a cabo a través de *ZenHub* [2] integrado directamente en GitHub.

Sprints realizados

Además de la división del desarrollo del proyecto en *sprints*, podemos dividirlo en diferentes fases:

- Durante la primera fase se llevaron a cabo tareas de **investigación** y **estudio** del proyecto previo[10] así como de las bases teóricas y herramientas que se utilizarían durante el proceso.
- En la segunda fase, una vez comprendidos los conceptos planteados en el proyecto ya existente y en las bases teóricas (métricas, framework de medición, estructura del proyecto, etc.) se realizaron tareas de **configuración** del entorno de desarrollo, tanto para el código como para la memoria del proyecto.
- Tercera fase donde gracias a la investigación y estudio realizados se realizan diferentes tareas de **documentación** de la memoria del proyecto.

[1] Planificación de proyectos para desarrolladores - https://github.com/features/issues

[2] Página de inicio de ZenHub - https://www.zenhub.com/

- Cuarta fase, la más extensa de todas, donde se realizan tareas de **desarrollo** para actualizar las dependencias del proyecto, implementar nuevas métricas, mejoras en la interfaz y la integración con GitHub.
- Quinta fase final de **documentación** en la que se finaliza la memoria y se trabaja en los anexos. Finalmente se preparan los vídeos necesarios y las guías de usuario.

En la siguiente sección se van a describir estas fases, que se pueden también consultar a través de las *issues* del repositorio del proyecto:

https://github.com/Joaquin-GM/GII_O_MA_19.07-Comparador-de-metricas-de-evolucion-en-repositorios-Software

Se utilizarán de manera explicativa capturas de la gestión de los *sprints* gestionados con ZenHub.

Investigación y estudio

Esta fase inicial comenzó en octubre de 2021 y duró aproximadamente 3 sprints (6 semanas) durante los que se asentó la base teórica del proyecto y se comprendió la estructura del mismo así como los objetivos. Además del trabajo con las referencias teóricas como [8] se buscó información sobre trabajos y aplicaciones que están relacionados con la temática de este proyecto ya entreviéndose que hay muy pocas soluciones similares y ninguna con las capacidades que tendría el proyecto a su finalización.
También se realizó la formación en Vaadin que proponen los propios creadores del framework [3] así como tutoriales para que el alumno se familiarizara con LaTeX y con la herramienta utilizada TexMaker[4].

A continuación se muestran algunas de las *issues* con las que se trabajó en esta fase:

[3]Centro de aprendizaje de Vaadin - https://vaadin.com/learn/tutorials?version=v14

[4]TexMaker editor LaTeX - https://www.xm1math.net/texmaker/

Figura A.1: *Issues* relacionadas con la fase de estudio e investigación investigación

Configuración del entorno de desarrollo

En esta fase se trabajó configurando las diferentes herramientas que se iban a utilizar en el proyecto, GitHub + ZenHub para la gestión de *issues* y *sprints*, Eclipse como IDE de desarrollo, Maven y Jetty para la gestión y ejecución del proyecto en local y TexMaker como editor LaTeX para trabajar con la memoria. Esta fase duró otros tres sprints (6 semanas) y a continuación se muestran las las *issues* con las que se trabajó en esta fase:

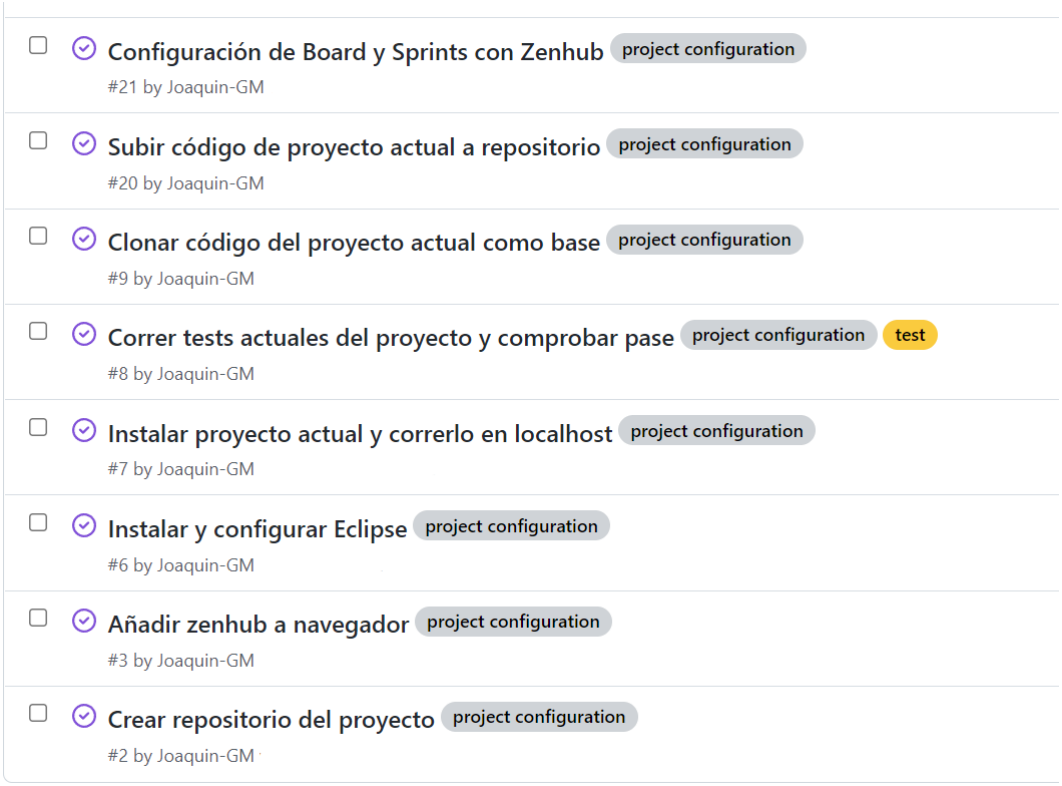


Figura A.2: *Board* del proyecto en ZenHub

También se muestra en la siguiente figura cómo quedó el tablero de ZenHub una vez estuvo integrado en el repositorio directamente en GitHub a través de la extensión de navegador de ZenHub:

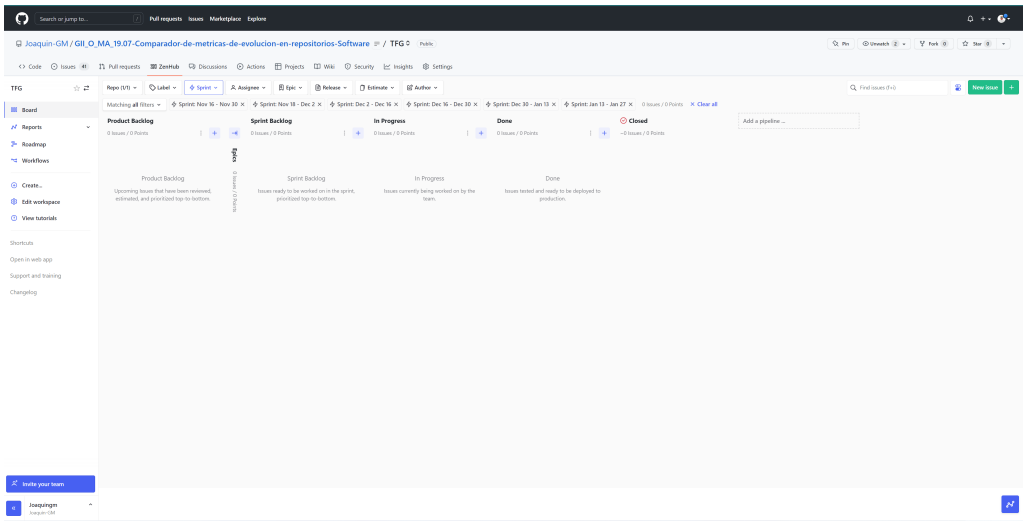


Figura A.3: Creación de nueva *issue* en ZenHub

Como se puede apreciar en la figura anterior, ZenHub provee de numeras herramientas muy útiles para la gestión de las *issues* del proyecto permitiéndonos trabajar con ellas de forma sencilla y agruparlas en *sprints*.
Para facilitar la creación de nuevas *issues* se crearon plantillas que se pueden utilizar también desde el menú de creación de *issues* de ZenHub:

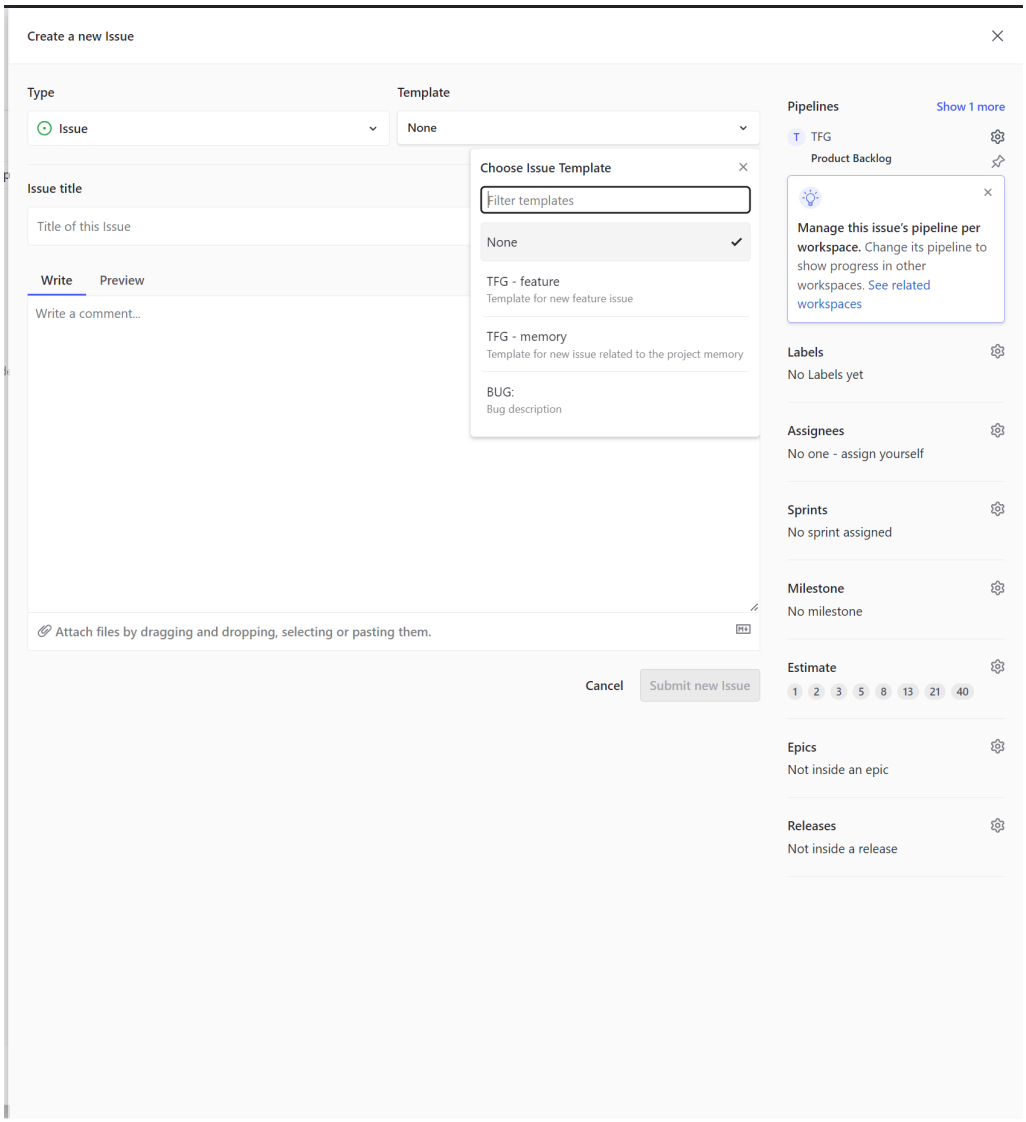


Figura A.4: Creación de nueva *issue* en ZenHub

Primera fase de documentación

En esta primera fase de documentación se trabajó durante 3 sprints (6 semanas) en los siguientes apartados de la memoria:

- Introducción y Objetivos
- Resumen y Descriptores
- Conceptos teóricos

- Técnicas y herramientas
- Aspectos relevantes del desarrollo del proyecto (que se finalizaría más adelante)

A continuación podemos ver las issues con las que se trabajó en esta fase:

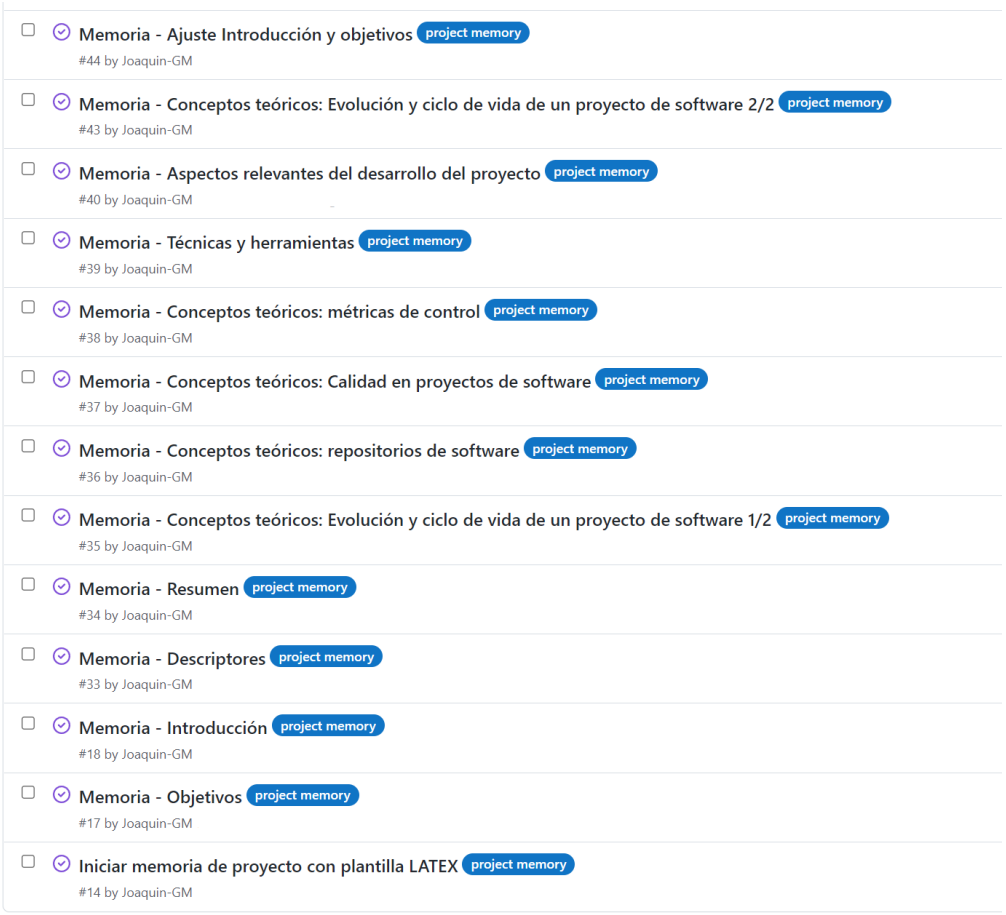


Figura A.5: Primeras *issues* relacionadas con la memoria

ZenHub también nos proporciona información sobre la gestión de las *issues* y el avance del proyecto. Podemos destacar los gráficos *burndown* que representan el avance durante el transcurso del sprint, marcando una guía orientativa de cómo debería ser. Este tipo de gráfico representa los puntos asignados a las diferentes *issues* y va bajando conforme éstas se van cerrando durante el *sprint*.

Como vemos en la siguiente figura, en algunos sprints se estimó demasiado trabajo o bien no se pudo realizar al completo y en el siguiente gráfico se puede ver cómo no se llegan a completar todos los puntos de las *issues* planteadas:

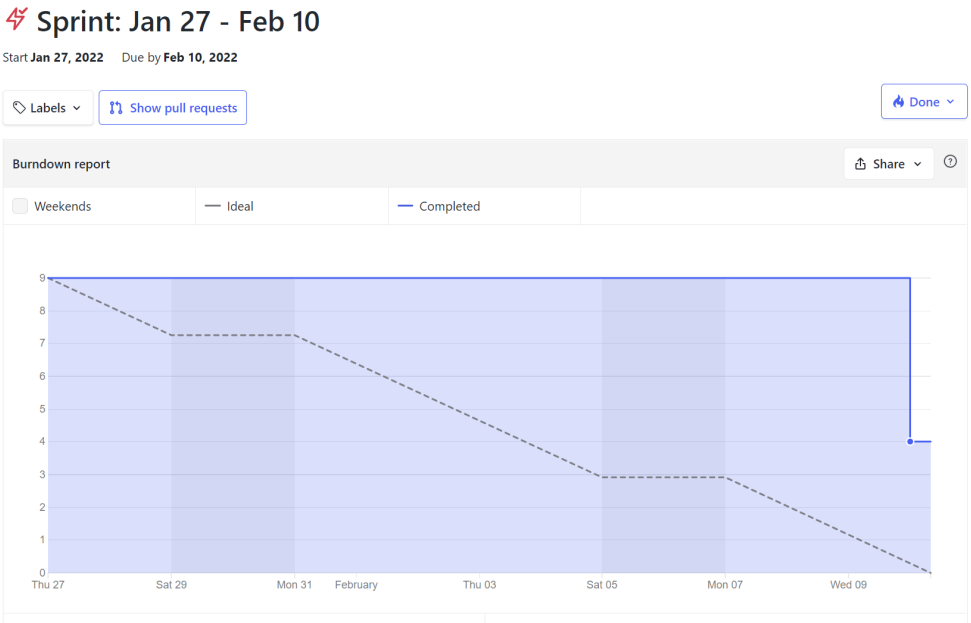


Figura A.6: Primeras *issues* relacionadas con la memoria

Fase de implementación de mejoras y nueva funcionalidad

La más larga de todas las fases con una duración de unos 7 sprints (14 semanas). En esta fase se realizaron las tareas de desarrollo entre las que destacan:

- Actualización del código previo y dependencias.
- Corrección de bugs existentes.
- Mejora del motor de cálculo de métricas para casos poco comunes.
- Mejoras de interfaz y usabilidad (leyenda, tooltips, separación por grupos de métricas, botones para cerrar diálogos, etc.)
- Nuevas métricas CI/CD: IC1, IC2, IC3, P1 y P2
- Adaptación de las funcionalidades a las nuevas métricas (export, import, actualización, etc.)

- Integración con GitHub y adaptación de la interfaz.

Las *issues* que se han realizado en esta fase son:

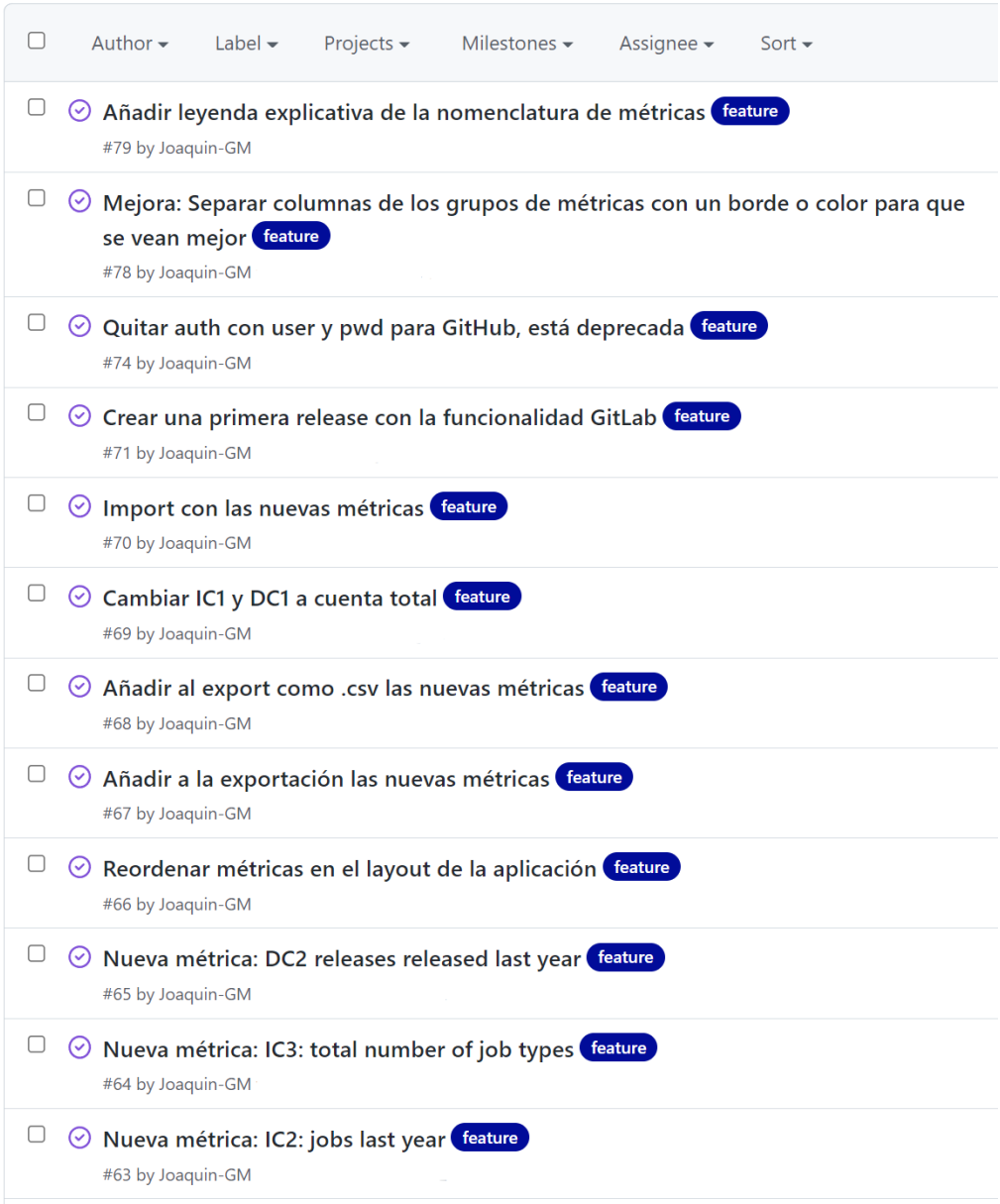


Figura A.7: *Issues* de desarrollo 1/2

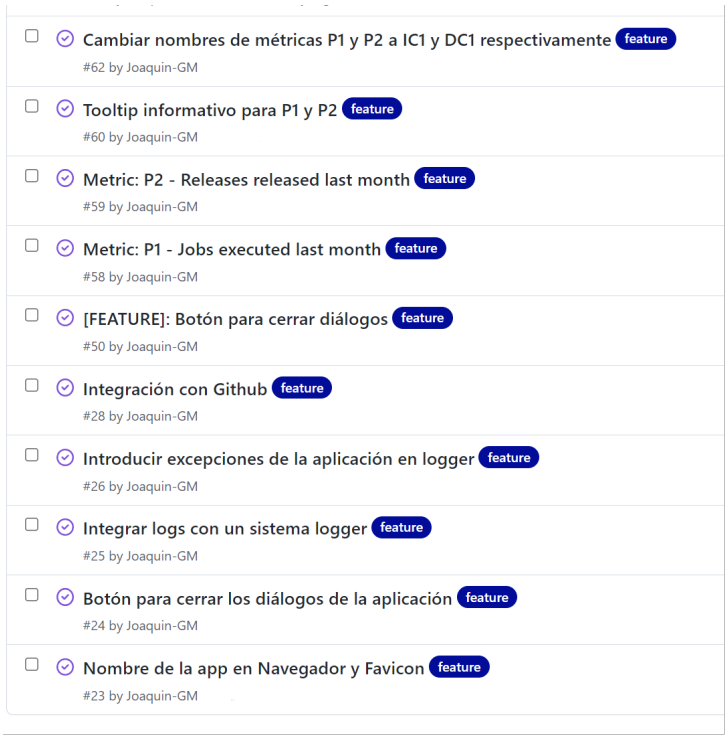


Figura A.8: *Issues* de desarrollo 2/2

Un ejemplo de gráfico *burndown* de uno de los sprint de esta fase puede verse en la siguiente figura:

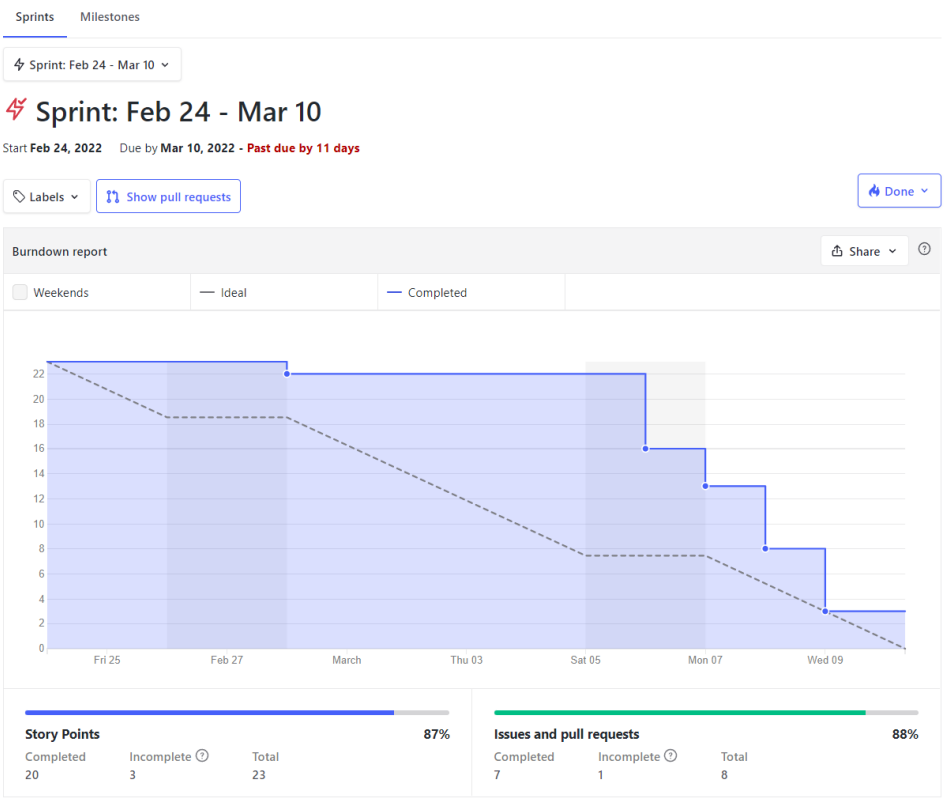


Figura A.9: *Burndown* de un sprint de desarrollo

En este caso se puede observar que sí se completaron todas las *issues* menos una. Se puede apreciar en el gráfico que durante los primeros días del sprint no se pudo terminar ninguna *issue*, si se hubiera podido conseguir nos habríamos aproximado más al avance ideal y probablemente se hubiera podido cerrar la *issue* pendiente de ese sprint.

Segunda fase de documentación

En esta última fase de documentación se trabajó durante 2 sprint (4 semanas) en los siguientes apartados:

- Memoria: Aspectos relevantes del desarrollo del proyecto
- Memoria: Trabajos relacionados
- Memoria: Conclusiones y líneas de trabajo futuras
- Anexos

- Vídeos de presentación del proyecto

A continuación podemos ver las issues con las que se trabajó en esta fase:

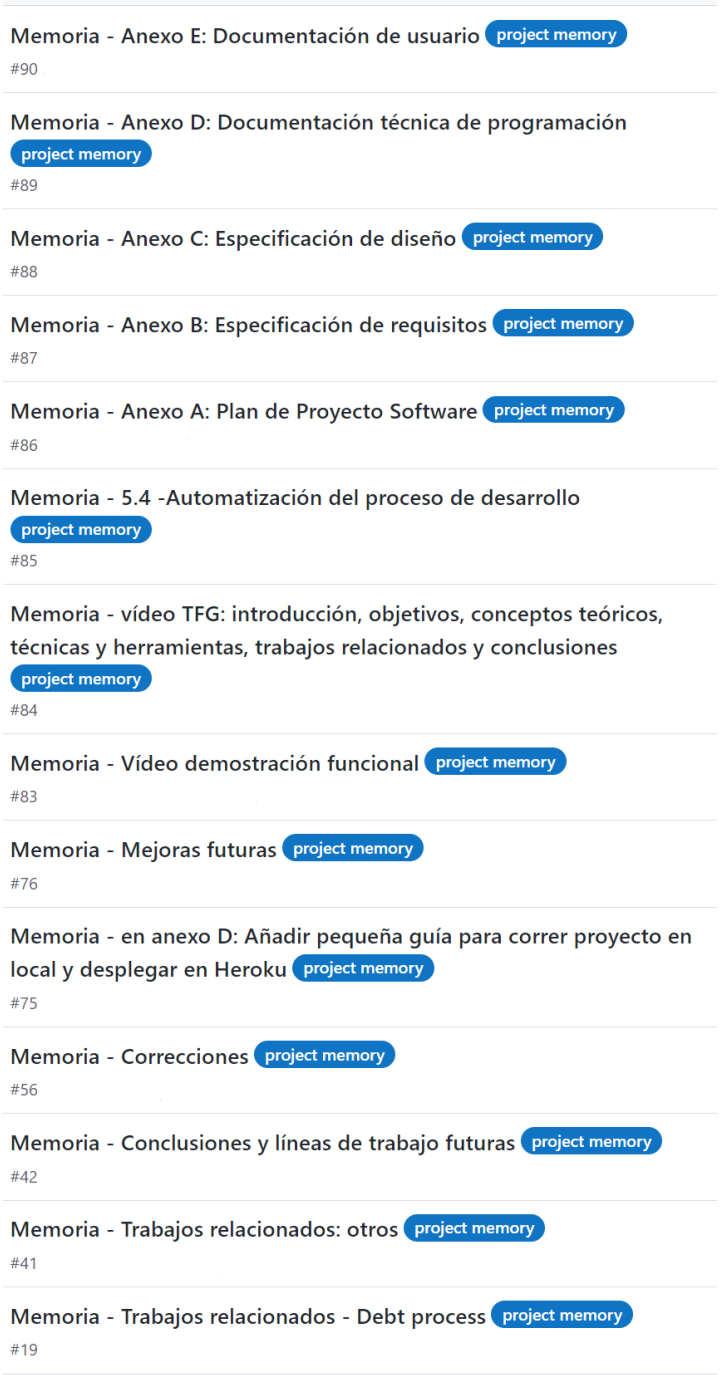


Figura A.10: Últimas *issues* relacionadas con la memoria, anexos y documentación

Un último ejemplo de gráfico *burndown* de esta fase en el que sí se fue avanzando siguiendo el avance ideal:

Figura A.11: *Burndown* del último sprint de documentación

A.3. Estudio de viabilidad

Viabilidad económica

A continuación se va a realizar un estudio del coste que supondría haber realizado el proyecto en un entorno real y se analizan posibles maneras de monetizar el proyecto para cubrir este coste.

Costes

Costes de personal

Es el mayor de los costes como en prácticamente todos los proyectos de desarrollo de software, el proyecto ha sido realizado por una persona con una jornada reducida de unas 12 horas semanales (debido a la simultaneidad del desarrollo del proyecto con la actividad laboral del alumno) durante 8 meses (mediados de octubre 2021 hasta mediados de junio de 2022). Se pueden estimar los siguientes valores:

- Salario neto mensual: 500 €
- Retención por IRPF de carácter general del 15 % [1]
- Una retribución a la Seguridad Social de 28,3 % [2]: Empresa (23,6 %) + Trabajador(Desempleo de tipo general + FOGASA+ Formación profesional)(4,70 %)

Realizando cálculos, se obtiene:

Concepto	**Coste**
Salario mensual neto	500 €
Retención IRPF (15 %)	122,15 €
Seguridad Social (29,9 %)	192,18 €
Salario mensual bruto	814,33 €
Total 8 meses	4.314,33 €

Tabla A.1: Costes de personal

Costes de hardware

Se va a considerar como coste únicamente un ordenador, hay que tener en cuenta que el período de amortización de equipos para procesos de información es de 8 años [9], que se considera un coste de 800 € para dicho ordenador y que se ha usado durante 8 meses:

Concepto	**Coste**	**Coste amortizado**
Ordenador portátil	800 €	66,67 €
Total	800 €	66,67 €

Tabla A.2: Costes de hardware

Costes de software

Todo el software empleado para el desarrollo del proyecto tiene licencias gratuitas, salvo el propio sistema operativo. Se ha utilizado Windows 11 en este caso y se ha trabajado con una licencia que tiene un coste muy reducido de unos 12 €al ser una clave OEM (que son perfectamente legales).
Teniendo en cuenta dicho coste y que la amortización para Sistemas y programas informáticos es de 6 años [9] y se ha utilizado durante 8 meses:

Concepto	**Coste**	**Coste amortizado**
Windows 10 Home	12 €	1,33 €
Total	12 €	1,33 €

Tabla A.3: Costes de software

Otros costes Se van a tener en cuenta además otros costes necesarios para poder desarrollar el proyecto en un entorno real. Se va a considerar una opción algo pesimista en la que es necesario alquilar una oficina para el desarrollador con sus costes asociados durante los 8 meses de desarrollo (alquiler unos 50 €al mes, internet unos 30 €al mes y algo de material de oficina):

Concepto	**Coste**
Alquiler de oficina	400 €
Internet	240 €
Material de oficina	40 €
Total	680 €

Tabla A.4: Costes varios.

Coste total

El coste total por tanto sería:

Concepto	**Coste**
Personal	4.314,33 €
Hardware	66,67 €
Software	1,33 €
Otros	680 €
Total	5.062,33 €

Tabla A.5: Costes totales.

Beneficios

Actualmente la aplicación tiene licencia libre y es gratuita por lo que no se obtiene ningún beneficio con ella. Para poder obtenerlos se pueden plantear diferentes alternativas:

- Pago a través de licencias de uso.
- Pago por subscripción al servicio.
- Incluir publicidad, esta solución podría ser la mejor si queremos cubrir costes y que el proyecto siga siendo de uso gratuito.

Viabilidad legal

Nuestro software al trabajar con repositorios públicos o aquellos a los que el usuario tenga acceso no puede incumplir con la privacidad de los mismos, por tanto, lo que hay que analizar son las licencias del software que se ha usado durante desarrollo de la aplicación.
En la Fig. A.12 podemos ver los artefactos utilizados en el proyecto (dependencias del proyecto en el fichero pom.xml:) y cuya información se puede obtener en el repositorio de Maven [5]

Grupo	Artefacto	URL	Licencia
com.vaadin	vaadin-bom	https://mvnrepository.com/artifact/com.vaadin/vaadin-bom	Apache 2.0
com.vaadin	vaadin-core	https://mvnrepository.com/artifact/com.vaadin/vaadin-core	Apache 2.0
com.vaadin	flow-server-production-mode	https://mvnrepository.com/artifact/com.vaadin/flow-server-production-mode	Apache 2.0
com.vaadin	vaadin-maven-plugin	https://mvnrepository.com/artifact/org.apache.maven/maven-plugin-api	Apache 2.0
org.claspina	confirm-dialog	https://mvnrepository.com/artifact/org.claspina/confirm-dialog	Apache 2.0
com.vaadin.componentfactory	tooltip	https://mvnrepository.com/artifact/com.vaadin.componentfactory/tooltip	Apache 2.0
org.gitlab4j	gitlab4j-api	https://mvnrepository.com/artifact/org.gitlab4j/gitlab4j-api	MIT
org.kohsuke	github-api	https://mvnrepository.com/artifact/org.kohsuke/github-api	MIT
com.google.code.gson	gson	https://mvnrepository.com/artifact/com.google.code.gson/gson	Apache 2.0
org.slf4j	slf4j-api	https://mvnrepository.com/artifact/org.slf4j/slf4j-api	MIT
org.apache.logging.log4j	log4j-slf4j-impl	https://mvnrepository.com/artifact/org.apache.logging.log4j/log4j-slf4j-impl	Apache 2.0
org.apache.logging.log4j	log4j-core	https://mvnrepository.com/artifact/org.apache.logging.log4j/log4j-core	Apache 2.0
org.junit.jupiter	junit-jupiter-api	https://mvnrepository.com/artifact/org.junit.jupiter/junit-jupiter-api	EPL 2.0
org.junit.jupiter	junit-jupiter-engine	https://mvnrepository.com/artifact/org.junit.jupiter/junit-jupiter-engine	EPL 2.0
org.junit.jupiter	junit-jupiter-params	https://mvnrepository.com/artifact/org.junit.jupiter/junit-jupiter-params	EPL 2.0
org.apache.commons	commons-math3	https://mvnrepository.com/artifact/org.apache.commons/commons-math3	Apache 2.0
org.apache.maven.plugins	maven-compiler-plugin	https://mvnrepository.com/artifact/org.apache.maven.plugins/maven-compiler-plugin	Apache 2.0
org.apache.maven.plugins	maven-war-plugin	https://mvnrepository.com/artifact/org.apache.maven.plugins/maven-war-plugin	Apache 2.0
org.apache.maven.plugins	maven-site-plugin	https://mvnrepository.com/artifact/org.apache.maven.plugins/maven-site-plugin	Apache 2.0
org.apache.maven.plugins	maven-deploy-plugin	https://mvnrepository.com/artifact/org.apache.maven.plugins/maven-deploy-plugin	Apache 2.0
org.apache.maven.plugins	maven-surefire-plugin	https://mvnrepository.com/artifact/org.apache.maven.plugins/maven-surefire-plugin	Apache 2.0
org.apache.maven.plugins	maven-failsafe-plugin	https://mvnrepository.com/artifact/org.apache.maven.plugins/maven-failsafe-plugin	Apache 2.0
org.jacoco	jacoco-maven-plugin	https://mvnrepository.com/artifact/org.jacoco/jacoco-maven-plugin	EPL 2.0
com.gavinmogan	codacy-maven-plugin	https://mvnrepository.com/artifact/com.gavinmogan/codacy-maven-plugin	MIT
com.heroku.sdk	heroku-maven-plugin	https://mvnrepository.com/artifact/com.heroku.sdk/heroku-maven-plugin	MIT
org.eclipse.jetty	jetty-maven-plugin	https://mvnrepository.com/artifact/org.eclipse.jetty/jetty-maven-plugin	Apache 2.0 EPL 2.0

Figura A.12: Licencias de las dependencias del proyecto

Por lo tanto, las licencias a las que está sometido nuestro proyecto son, de más a menos permisivas:

- *MIT*
- *Apache v.2.0*
- *EPL 2.0*

La licencia más restrictiva de las tres es la *Eclipse Public License* que es la que tienen las librerías que se utilizan para pruebas (JUnit).

A este proyecto se le podría dar la licencia *GNU General Public License v3.0*, que permite el uso comercial, modificación, distribución, uso privado y

[5] https://mvnrepository.com/

el uso de patentes mientas queramos que siga siendo de acceso y uso público como hasta hora. Esta licencia es compatible con las del software utilizado en el proyecto pues la versión *EPL 2.0* permite utilizar la versión 2 de la *GPL* de *GNU* y posteriores como licencia secundaria en una parte especifica del código. Por esto, se garantiza la compatibilidad de ese código con esas versiones de la GPL [3].

Apéndice B

Especificación de Requisitos

B.1. Introducción

En este anexo se analizan y documentan las diferentes necesidades funcionales y no funcionales que deberán ser soportadas por la aplicación generada en este proyecto. Se van a nombrar tanto los requisitos del proyecto original[10] como los del proyecto actual pues éste último ha de cumplir ambos.

B.2. Objetivos generales

El objetivo general de este TFG es mantener la funcionalidad implementada en el proyecto original[10] y mejorarla incluyendo nuevas métricas y realizando la integración con GitHub para poder trabajar con repositorios alojados allí. Además se implementarán mejoras de la interfaz para mejorar la usabilidad y se corregirán errores.

Por tanto se pueden nombrar los siguientes objetivos:

- Se obtendrán medidas de métricas de evolución de uno o varios proyectos alojados en repositorios tanto en GitLab como en GitHub.
- Las métricas que se desean calcular de un repositorio son algunas de las especificadas en la tesis titulada "*sPACE: Software Project Assessment in the Course of Evolution*" [8] y adaptadas a los repositorios software junto a nuevas métricas relacionadas con la integración y despliegue continuos (las 5 últimas):
  - Número total de incidencias (*issues*)

  - Cambios (*commits*) por incidencia
  - Porcentaje de incidencias cerrados
  - Media de días en cerrar una incidencia
  - Media de días entre cambios
  - Días entre primer y último cambio
  - Rango de actividad de cambios por mes
  - Porcentaje de pico de cambios
  - Número total de *jobs* ejecutados
  - Número de *jobs* ejecutados el último año
  - Número de tipos diferentes de *jobs* ejecutados
  - Número total de *releases*
  - Número de *releases* lanzadas el último año
- Se pretende ser capaces de evaluar el estado de un proyecto comparándolo con otros proyectos por medio del uso de las métricas anteriores. Para ello se deberán establecer unos valores umbrales por cada métrica basados en el cálculo de los cuartiles Q1 y Q3.
  Además, estos valores se podrán calcular dinámicamente y se almacenarán en perfiles de configuración de métricas. Estos umbrales deberán existir para todas la métricas, incluidas las nuevas.
- Se dará la posibilidad de almacenar de manera persistente estos perfiles de métricas para permitir comparaciones futuras.
- También se permitirá almacenar de forma persistente las métricas obtenidas de los repositorios para su posterior consulta o tratamiento. Esto permitiría comparar nuevos proyectos con proyectos de los que ya se han calculado sus métricas.
- La persistencia debe incluir también las nuevas métricas, tanto para la exportación como para la importación de los valores de las mismas.

B.3. Objetivos técnicos

Este apartado resume requisitos del proyecto más técnicos y centrados en el proceso y otras características no funcionales.

- Utilizar el diseño basado en frameworks y en patrones de diseño [4] existente para implementar las nuevas métricas.
- Utilizar y adaptar en lo necesario el diseño de la aplicación para extender a GitHub su funcionalidad y seguir permitiendo la extensión otras plataformas de desarrollo colaborativo como Bitbucket.
- Crear una batería de pruebas automáticas en los subsistemas de lógica de la aplicación.

- Utilizar una plataforma de desarrollo colaborativo que incluya un sistema de control de versiones, un sistema de seguimiento de incidencias y que permita una comunicación fluida entre el tutor y el alumno. Se sugiere la utilización de GitHub.
- Utilizar un sistema de integración y despliegue continuo.
- Mejorar la gestión de errores existente tratando excepciones de biblioteca y de la propia aplicación y registrar dichos eventos de error e información en ficheros de *log*.
- Probar la aplicación con ejemplos reales y utilizando técnicas avanzadas, como entrada de datos de test en ficheros con formato tabulado tipo CSV (*Comma Separated Values*).

B.4. Actores

Se consideran dos actores:

- El usuario de la aplicación, es aquella persona que utilice la aplicación.
- El desarrollador, que a su vez puede ser considerado como usuario pero que además tiene las labores de implementación y mantenimiento de la aplicación.

B.5. Catalogo de requisitos

En esta sección se van a enumerar los requisitos que el sistema tendrá que satisfacer. Están divididos en dos tipos, en el primero se basa en funcionalidad y el segundo en aspectos del proceso de desarrollo y aspectos técnicos del sistema como la mantenibilidad.

Requisitos funcionales

- **RF-1** Establecer conexión con GitHub: El usuario debe poder establecer conexión con GitHub.
    - **RF-1.1** Debido a las restriccones que tiene la API de GitHub y que se *deprecó* el acceso del usuario por usuario y contraseña[1], el usuario podrá iniciar sesión desde la aplicación sólo mediante

[1]Autenticación en GitHub API - https://docs.github.com/es/rest/overview/other-authentication-methods

un token de acceso personal a su usuario en GitHub para poder obtener los repositorios públicos y privados a los que tenga acceso.

- **RF-1.2** La aplicación mostrará al usuario en todo momento la conexión que está utilizando
- **RF-1.3** El usuario podrá cambiar de conexión teniendo en cuenta que solo puede haber un tipo de conexión con GitHub activo en un instante dado. Esta conexión será independiente de la posible conexión con GitLab.

- **RF-2** Establecer conexión con GitLab: El usuario debe poder establecer distintos tipos de conexión a GitLab.
  - **RF-2.1** El usuario podrá iniciar sesión desde la aplicación mediante usuario y contraseña a su usuario en GitLab para poder obtener los repositorios públicos y privados a los que tenga acceso.
  - **RF-2.2** El usuario podrá iniciar sesión desde la aplicación mediante un token de acceso personal a su usuario en GitLab para poder obtener los repositorios públicos y privados a los que tenga acceso.
  - **RF-2.3** El usuario podrá establecer una conexión pública a GitLab para poder obtener los repositorios públicos.
  - **RF-2.4** El usuario podrá utilizar la aplicación sin establecer una conexión. Aunque no tenga acceso a los repositorios de GitLab.
  - **RF-2.5** El usuario deberá elegir un tipo de conexión al entrar por primera vez a la aplicación.
  - **RF-2.6** La aplicación mostrará al usuario en todo momento la conexión que está utilizando
  - **RF-2.7** El usuario podrá cambiar de conexión teniendo en cuenta que solo puede haber un tipo de conexión con GitLab activo en un instante dado. Esta conexión será independiente de la posible conexión con GitHub.

- **RF-3** Gestión de proyectos. El usuario podrá calcular las métricas de proyectos tanto de GitHub como de GitLab definidas en la sección B.5: '*Definición de las métricas*' y se mostrarán los resultados en forma de tabla en la que las filas se correspondan con los proyectos y las columnas con las métricas.
  - **RF-3.1** El usuario podrá añadir un proyecto siempre que tenga una conexión a GitLab o con GitHub (dependiendo del dónde esté alojado dicho proyecto):

  - **RF-3.1.1** El usuario podrá añadir un proyecto a partir del nombre de usuario o ID del propietario y el nombre del proyecto, siempre que tenga acceso desde la conexión establecida.
  - **RF-3.1.2** El usuario podrá añadir un proyecto a partir del nombre o del ID del grupo al que pertenece el proyecto y su nombre, siempre que tenga acceso desde la conexión establecida.
  - **RF-3.1.3** El usuario podrá añadir un proyecto a partir de su URL, siempre que tenga acceso desde la conexión establecida.
- **RF-3.2** El usuario no podrá añadir un proyecto que ya esté en la tabla.
- **RF-3.3** Al añadir un proyecto a la tabla se calcularán las métricas definidas en la sección B.5: '*Definición de las métricas*' y se mostrarán en forma de tabla.
- **RF-3.4** El usuario podrá eliminar un proyecto de la tabla.
- **RF-3.5** El usuario podrá volver a obtener las métricas de un repositorio que ya haya añadido, siempre que tenga acceso desde la conexión establecida (con GitHub o GitLab dependiendo de dónde esté alojado el proyecto).
- **RF-3.6** El usuario podrá exportar los proyectos y sus métricas a un fichero con formato '.emr'.
- **RF-3.7** El usuario podrá exportar los resultados de las métricas de todos los proyectos en un fichero con formato CSV.
- **RF-3.8** El usuario podrá importar y añadir repositorios a la tabla desde un fichero con formato '.emr', respetando el requisito RF3.2 de no añadir un repositorio ya existente.
- **RF-3.9** El usuario podrá importar repositorios a la tabla, sobrescribiendo los de la tabla.
- **RF-3.10** Se podrán filtrar los proyectos por su nombre.
- **RF-3.11** Se puede ordenar los repositorios por nombre, fecha de medición y por cualquiera de las métricas.
- **RF-3.12** Al ordenar por las nuevas métricas relacionadas con CICD habrá que tener en cuenta que habrá medidas que no se habrán calculado por falta de datos de GitLab si no se ha realizado conexión *autenticada* pues GitLab API no devuelve información sobre *jobs* y *releases* con conexión pública.

- RF-4 Evaluación de métricas. Las métricas, una vez calculadas, serán evaluadas mediante un código de color (verde - bueno, naranja - peligro, rojo - malo) a partir de un perfil de métricas, que será un conjunto de valores mínimo y máximo de cada una de las métricas, a partir de la definición de la evaluación en la sección B.5: '*Evaluación de métricas*'. Esto ha de aplicar también a las nuevas métricas que se implementen no sólo a las existentes.

  - **RF-4.1** Se podrá cargar un perfil de métricas que contenga valores umbrales mínimo y máximo y se utilizarán para evaluarlas.
  - **RF-4.1** El usuario podrá cargar un perfil por defecto creado a partir de un conjunto de datos [2] de un estudio empírico de las métricas de evolución del software en trabajos finales de grado[5]
  - **RF-4.2** El usuario podrá cargar un perfil creado a partir de los repositorios que existan en la tabla
  - **RF-4.3** El usuario podrá exportar el perfil de métricas que tenga cargado a un fichero con formato emmp
  - **RF-4.4** El usuario podrá importar un perfil de métricas que haya guardado anteriormente para evaluar los repositorios que tenga en la tabla
  - **RF-4.5** En un instante, solo puede haber un perfil de métricas cargado.
  - **RF-4.6** Al inicio se cargará el perfil de evaluación por defecto.

Definición de las métricas

En el requisito RF-3 implica que se deben calcular las siguientes métricas de un proyecto:

I1 - Número total de issues (incidencias)

- **Categoría**: Proceso de Orientación
- **Descripción**: Número total de issues creadas en el repositorio
- **Propósito**: ¿Cuántas issues se han definido en el repositorio?
- **Fórmula**: NTI. NTI = *número total de issues*

[2] https://github.com/clopezno/clopezno.github.io/blob/master/agile_practices_experiment/DataSet_EvolutionSoftwareMetrics_FYP.csv

- **Fuente de medición**: Proyecto en una plataforma de desarrollo colaborativo.
- **Interpretación**: $NTI \geq 0$. Valores bajos indican que no se utiliza un sistema de seguimiento de incidencias, podría ser porque el proyecto acaba de comenzar
- **Tipo de escala**: Absoluta
- **Tipo de medida**: *NTI = Contador*

I2 - Commits (cambios) por issue

- **Categoría**: Proceso de Orientación
- **Descripción**: Número de commits por issue
- **Propósito**: ¿Cuál es el volumen medio de trabajo de las issues?
- **Fórmula**: $CI = \frac{NTC}{NTI}$. *CI = Cambios por issue, NTC = Número total de commits, NTI = Numero total de issues*
- **Fuente de medición**: Proyecto en una plataforma de desarrollo colaborativo.
- **Interpretación**: $CI \geq 1$, Lo normal son valores altos. Si el valor es menor que uno significa que hay desarrollo sin documentar.
- **Tipo de escala**: Ratio
- **Tipo de medida**: *NTC, NTI = Contador*

I3 - Porcentaje de issues cerradas

- **Categoría**: Proceso de Orientación
- **Descripción**: Porcentaje de issues cerradas
- **Propósito**: ¿Qué porcentaje de issues definidas en el repositorio se han cerrado?
- **Fórmula**: $PIC = \frac{NTIC}{NTI} * 100$. *PIC = Porcentaje de issues cerradas, NTIC = Número total de issues cerradas, NTI = Numero total de issues*

- **Fuente de medición**: Proyecto en una plataforma de desarrollo colaborativo.
- **Interpretación**: $0 \leq PIC \leq 100$. Cuanto más alto mejor
- **Tipo de escala**: Ratio
- **Tipo de medida**: *NTI, NTIC = Contador*

TI1 - Media de días en cerrar una issue

- **Categoría**: Constantes de tiempo
- **Descripción**: Media de días en cerrar una issue
- **Propósito**: ¿Cuánto se suele tardar en cerrar una issue?
- **Fórmula**: $MDCI = \frac{\sum_{i=0}^{NTIC} DCI_i}{NTIC}$. *MDCI = Media de días en cerrar una issue, NTIC = Número total de issues cerradas, DCI = Días en cerrar la issue*
- **Fuente de medición**: Proyecto en una plataforma de desarrollo colaborativo.
- **Interpretación**: $MDCI \geq 0$. Cuanto más pequeño mejor. Si se siguen metodologías ágiles de desarrollo iterativo e incremental como SCRUM, la métrica debería indicar la duración del *sprint* definido en la fase de planificación del proyecto. En SCRUM se recomiendan duraciones del *sprint* de entre una y seis semanas, siendo recomendable que no exceda de un mes [7].
- **Tipo de escala**: Ratio
- **Tipo de medida**: *NTI, NTIC = Contador*

TC1 - Media de días entre commits

- **Categoría**: Constantes de tiempo
- **Descripción**: Media de días que pasan entre dos commits consecutivos
- **Propósito**: ¿Cuántos días suelen pasar desde un commit hasta el siguiente?

- **Fórmula**: $MDC = \frac{\sum_{i=1}^{NTC} TC_i - TC_{i-1}}{NTC}$. $TC_i - TC_{i-1}$ en días; *MDC = Media de días entre cambios, NTC = Número total de commits, TC = Tiempo de commit*
- **Fuente de medición**: Proyecto en una plataforma de desarrollo colaborativo.
- **Interpretación**: $MDEC \geq 0$. Cuanto más pequeño mejor. Se recomienda no superar los 5 días.
- **Tipo de escala**: Ratio
- **Tipo de medida**: *NTC = Contador; TC = Tiempo*

TC2 - Días entre primer y último commit

- **Categoría**: Constantes de tiempo
- **Descripción**: Días transcurridos entre el primer y el ultimo commit
- **Propósito**: ¿Cuantos días han pasado entre el primer y el último commit?
- **Fórmula**: $DEPUC = TC2 - TC1$. $TC2 - TC1$ en días; *DEPUC = Días entre primer y último commit, TC2 = Tiempo de último commit, TC1 = Tiempo de primer commit*
- **Fuente de medición**: Proyecto en una plataforma de desarrollo colaborativo.
- **Interpretación**: $DEPUC \geq 0$. Cuanto más alto, más tiempo lleva en desarrollo el proyecto. En procesos software empresariales se debería comparar con la estimación temporal de la fase de planificación.
- **Tipo de escala**: Absoluta
- **Tipo de medida**: *TC = Tiempo*

TC3 - Ratio de actividad de commits por mes

- **Categoría**: Constantes de tiempo
- **Descripción**: Muestra el número de commits relativos al número de meses

- **Propósito**:¿Cuál es el número medio de cambios por mes?
- **Fórmula**: $RCM = \frac{NTC}{NM}$. *RCM = Ratio de cambios por mes, NTC = Número total de commits, NM = Número de meses que han pasado durante el desarrollo de la aplicación*
- **Fuente de medición**: Proyecto en una plataforma de desarrollo colaborativo.
- **Interpretación**: $RCM > 0$. Cuanto más alto mejor
- **Tipo de escala**: Ratio
- **Tipo de medida**: *NTC = Contador*

C1 - Cambios pico

- **Categoría**: Constantes de tiempo
- **Descripción**: Número de commits en el mes que más commits se han realizado en relación con el número total de commits
- **Propósito**: ¿Cuál es la proporción de trabajo realizado en el mes con mayor número de cambios?
- **Fórmula**: $CP = \frac{NCMP}{NTC}$. *CP = Cambios pico, NCMP = Número de commits en el mes pico, NTC = Número total de commits*
- **Fuente de medición**: Proyecto en una plataforma de desarrollo colaborativo.
- **Interpretación**: $0 \leq CCP \leq 1$. Mejor valores intermedios. Se recomienda no superar el 40 % del trabajo en un mes.
- **Tipo de escala**: Ratio
- **Tipo de medida**: *NCMP, NTC = Contador*

IC1 - Número total de *jobs* ejecutados

- **Categoría**: CI-CD
- **Descripción**: Número de *jobs* ejecutados en el proyecto.

- **Propósito**: ¿Cuál es el número total de de *jobs* ejecutados con éxito en el proyecto?
- **Fórmula**: NJE = *número total de jobs.*
- **Fuente de medición**: Proyecto en una plataforma de desarrollo colaborativo.
- **Interpretación**: $NJE \geq 0$. Un valor de cero indica que el proyecto no tiene integración y despliegues continuos y no se ha realizado ningún trabajo automatizado de despliegue.
- **Tipo de escala**: Absoluta
- **Tipo de medida**: NJE = *Contador*

IC2 - Número de *Jobs* ejecutados el último año

- **Categoría**: CI-CD
- **Descripción**: Número de *jobs* ejecutados en el último año (últimos 365 días).
- **Propósito**: ¿Cuál es el número total de *jobs* ejecutados con éxito en el proyecto durante el año previo?
- **Fórmula**: $NJELY$ = *número total de jobs ejecutados el útimo año.*
- **Fuente de medición**: Proyecto en una plataforma de desarrollo colaborativo.
- **Interpretación**: $NJELY \geq 0$. Un valor de cero indica que en el proyecto no se ha realizado ningún trabajo automatizado de despliegue durante el último año.
- **Tipo de escala**: Absoluta
- **Tipo de medida**: $NJELY$ = *Contador*

IC3 - *N*úmero de tipos diferentes dejobs ejecutados

- **Categoría**: CI-CD
- **Descripción**: Número de tipos diferentes de *jobs* ejecutados en el proyecto.

- **Propósito**: ¿Cuál es el número total de tipos de *jobs* ejecutados con éxito en el proyecto?
- **Fórmula**: $NTJE$ = *número total de tipos diferentes de jobs ejecutados en el proyecto.*
- **Fuente de medición**: Proyecto en una plataforma de desarrollo colaborativo.
- **Interpretación**: $NTJE \geq 0$. Un valor de cero indica que el proyecto no tiene integración y despliegues continuos y no se ha realizado ningún trabajo automatizado de despliegue.
- **Tipo de escala**: Absoluta
- **Tipo de medida**: $NTJE$ = *Contador*

DC1 - *N*úmero total dereleases

- **Categoría**: CI-CD
- **Descripción**: Número de *releases* lanzadas en el proyecto.
- **Propósito**: ¿Cuál es el número total de *releases* del proyecto?
- **Fórmula**: NRR = *número total de releases lanzadas en el proyecto.*
- **Fuente de medición**: Proyecto en una plataforma de desarrollo colaborativo.
- **Interpretación**: $NRR \geq 0$. Un valor de cero indica que aún no se ha lanzado ninguna *release* del proyecto.
- **Tipo de escala**: Absoluta
- **Tipo de medida**: NRR = *Contador*

DC2 - *N*úmero dereleases lanzadas el último año

- **Categoría**: CI-CD
- **Descripción**: Número de *releases* lanzadas en el proyecto en el último año (últimos 365 días).

- **Propósito**: ¿Cuál es el número total de *releases* lanzadas en el proyecto durante el año previo?
- **Fórmula**: $NRRLY$ = *número total de releases lanzadas en el proyecto en el último año.*
- **Fuente de medición**: Proyecto en una plataforma de desarrollo colaborativo.
- **Interpretación**: $NRRLY \geq 0$. Un valor de cero indica que no se ha lanzado ninguna *release* del proyecto durante el último año.
- **Tipo de escala**: Absoluta
- **Tipo de medida**: $NRRLY$ = *Contador*

Evaluación de métricas

Las métricas se evaluarán como buenas si:

- I1: El valor medido supera el umbral mínimo (Q1).
- I2: El valor medido se encuentra entre el umbral mínimo y el máximo (Q3).
- I3: El valor medido supera el umbral mínimo.
- TI1: El valor medido se encuentra entre el umbral mínimo y el máximo.
- TC1: El valor medido se encuentra entre el umbral mínimo y el máximo.
- TC2: El valor medido se encuentra entre el umbral mínimo y el máximo.
- TC3: El valor medido se encuentra entre el umbral mínimo y el máximo.
- C1: El valor medido se encuentra entre el umbral mínimo y el máximo.
- IC1: El valor medido se encuentra entre el umbral mínimo y el máximo. Si es el valor es 0 el proyecto no tiene integración y despliegue continuo.
- IC2: El valor medido se encuentra entre el umbral mínimo y el máximo Si es el valor es 0 el proyecto no tiene integración y despliegue continuo.
- IC3: El valor medido se encuentra entre el umbral mínimo y el máximo Si es el valor es 0 el proyecto no tiene integración y despliegue continuo.

- DC1: El valor medido se encuentra entre el umbral mínimo y el máximo Si es el valor es 0 el proyecto aún no tiene ninguna *release*.
- DC2: El valor medido se encuentra entre el umbral mínimo y el máximo.

Requisitos no funcionales

- **RNF-1** Se debe mantener y mejorar el diseño extensible a otras forjas de repositorios para que tal y como se haga con GitHub se pueda hacer en un futuro con otras forjas como Bitbucket.
- **RNF-2** Se debe mantener y mejorar el diseño extensible y reutilizable a nuevas métricas, siguiendo el framework descrito en *Soporte de Métricas con Independencia del Lenguaje para la Inferencia de Refactorizaciones* [6].
- **RNF-3** El diseño de la interfaz ha de ser intuitivo y fácil de utilizar.
- **RNF-4** Durante el proyecto se debe gestionar un flujo de trabajo guiado por la integración continua y el despliegue continuo.
- **RNF-5** Se debe crear una batería de pruebas automáticas en los subsistemas de lógica de la aplicación.
- **RNF-6** El proyecto debe estar ubicado en una forja de repositorios que incluya un sistema de control de versiones, un sistema de seguimiento de incidencias y que permita una comunicación fluida entre el tutor y el alumno. Se sugiere la utilización de GitHub.
- **RNF-7** Se debe diseñar una correcta gestión de errores definiendo excepciones de biblioteca y registrando eventos de error e información en ficheros de *log*.
- **RNF-8** Se ha de probar la aplicación con ejemplos reales.

Apéndice C

Especificación de diseño

C.1. Introducción

En este apartado recogen las causas por las que se han adoptado las soluciones de diseño del software desarrollado, divididas diseño de datos, arquitectónico y diseño de la interfaz de usuario.

C.2. Diseño de datos

Las entidades con las que se trabaja en el proyecto son clases de Java. Si en un futuro se trabajase con una base de datos se podrían convertir las entidades de ésta a las clases Java que se utilizan actualmente sin mayor problema.

En el paquete datamodel podemos encontrar las clases más importantes en relación a los datos:

- Repository. Sirve como definición de los datos que se obtienen de un repositorio. Estos datos son:
  - URL, nombre e ID del proyecto
  - Métricas internas (RepositoryInternalMetrics). Representan los valores de los que se obtienen las métricas externas y que interesan para satisfacer los requisitos funcionales de la aplicación.
  - Resultados de las métricas (MetricsResults). Al añadir un repositorio estos se calcularán y quedarán almacenados en una instancia de esta clase.

- Evaluación del proyecto (*projectEvaluation*). En esta clase se evalúan las diferentes métricas en función de sus umbrales y se almacena el porcentaje de métricas evaluadas como 'buenas' cada vez que se calcule alguna de ellas.
- Cabe indicar que se ha definido la igualdad de los repositorios en base a su URL, ID y nombre.
- RepositoryInternalMetrics. Esta clase contiene toda la información de un repositorio necesaria para calcular las métricas, que es la fecha de medición, el número total de issues, el número total de commits, el número de issues cerradas, una colección de días que se tardan en cerrar las issues, otra colección con las fechas de los commits, el tiempo de vida del proyecto, una colección de los *jobs* del proyecto y una colección de las *releases* del proyecto.
  Cabe indicar que se define el método *equals* a partir de la fecha de medición.
- User. Sirve para almacenar información que se obtiene de las forjas de repositorios acerca del usuario que ha iniciado sesión. Se almacenan la ID de usuario, la URL del avatar, el e-mail, el nombre y el nombre de usuario.

- Se han implementado clases personalizadas para envolver a las clases relacionadas con los *jobs* del proyecto y una colección de las *releases* devueltas por las librerías de integración con las forjas de repositorios. Esto ha sido necesario ya que las clases obtenidas por medio de las librerías de conexión no son serializables y en la aplicación necesitamos serializar los resultados de las métricas para poder exportarlos e importarlos.
  Las clases generadas son: *CustomGithubApiJob*, *CustomGithubApiRelease*, *CustomGitlbApiJob* y *CustomGitlabApiRelease*.
  Hay que indicar que en estas clases se han de almacenar como variables aquellas que se tengan que usar para los calculos de las métricas, por ejemplo el nombre de los *jobs* con *this.job = job.getName()* para el cálculo de la métrica IC3.

C.3. Diseño arquitectónico

A continuación se describe la estructura de paquetes del proyecto y las clases que contienen.

Paquete repositorydatasource

El paquete define el framework de conexión a una forja de repositorios e implementa la conexión a GitLab.

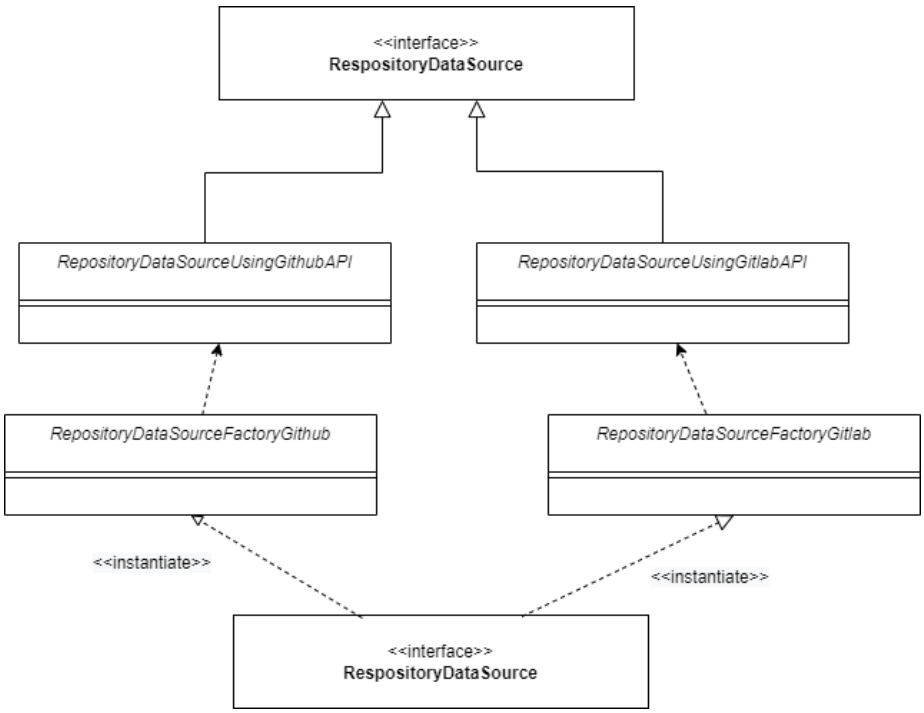


Figura C.1: Paquete repositorydatasource

Para crear un *Repositorydatasource* con acceso a otra forja solo se tiene que implementar las dos interfaces, sin necesidad de hacer más cambios en el código, como se puede ver en la imagen la implementación para GitHub y GitLab es equivalente.

Paquete metricsengine

En este paquete se define el framework de medición y sigue el diseño descrito en *Soporte de Métricas con Independencia del Lenguaje para la Inferencia de Refactorizaciones* [6], con unos pequeños cambios:

- Se ha aplicado a las métricas concretas el patrón ***Singleton*** [1], que obliga a que solo haya una única instancia de cada métrica. Además,

[1] https://refactoring.guru/design-patterns/singleton

se ha aplicado el patrón ***Método fábrica*** [2] tal y como se muestra en la Fig. C.2, de forma que *MetricConfiguration* no esté asociada con la métrica en sí, sino con una forma de obtenerla.
El objetivo de trabajar de esta manera es facilitar la persistencia de un perfil de métricas. Las métricas se podrían ver como clases estáticas, no varían en tiempo de ejecución y solo debería haber una instancia de cada una de ellas. Por ello, al importar o exportar un perfil de métricas con su conjunto de configuraciones de métricas, estas configuraciones no deberían asociarse a la métrica, sino a la forma de acceder a la única instancia de esa métrica.

- Se trabaja con los métodos *evaluate* y *getEvaluationFunction* en la interfaz *IMetric*, ver Fig. C.3, para obtener los valores de las mismas, siendo estos métodos un añadido que se realizó al framework de medición original.
  Esto permite interpretar y evaluar los valores medidos sobre los valores límite de la métrica o configuración de métrica deduciéndose si su valor es bueno o no según el perfil de evaluación.
  Por ejemplo, puede que para unas métricas un valor aceptable esté comprendido entre el valor límite superior y el valor límite inferior; y para otras un valor aceptable es aquel que supere el límite inferior.
  EvaluationFunction es una interfaz funcional [3] de tipo '*función*': recibe uno o más parámetros y devuelve un resultado. Este tipo de interfaces son posibles a partir de la versión 1.8 de Java.
  Esto permite definir los tipos de los parámetros y de retorno de una función que se puede almacenar en una variable. De este modo se puede almacenar en una variable la forma en la que se puede evaluar la métrica.

[2] https://refactoring.guru/design-patterns/factory-method

[3] Enlaces a la documentación: https://docs.oracle.com/javase/8/docs/api/java/lang/FunctionalInterface.html — https://docs.oracle.com/javase/8/docs/api/java/util/function/package-summary.html

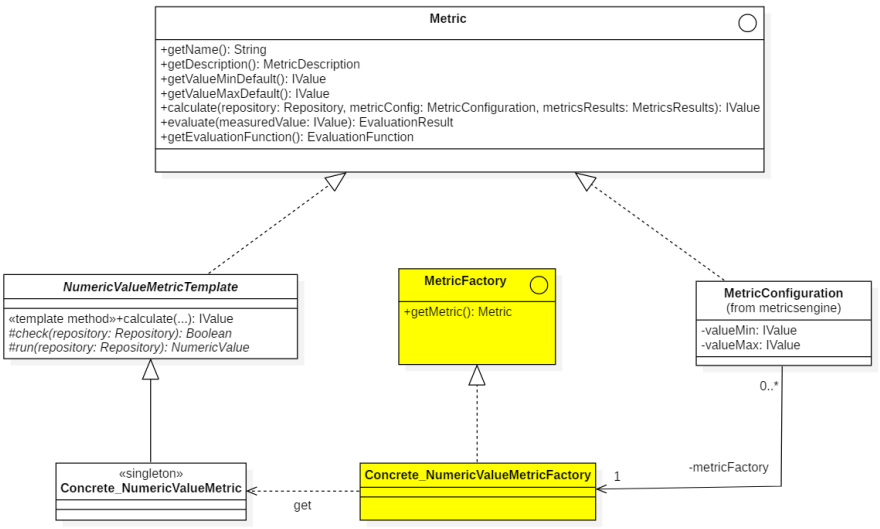


Figura C.2: Patrones “singleton” y “método fábrica” sobre el framework de medición

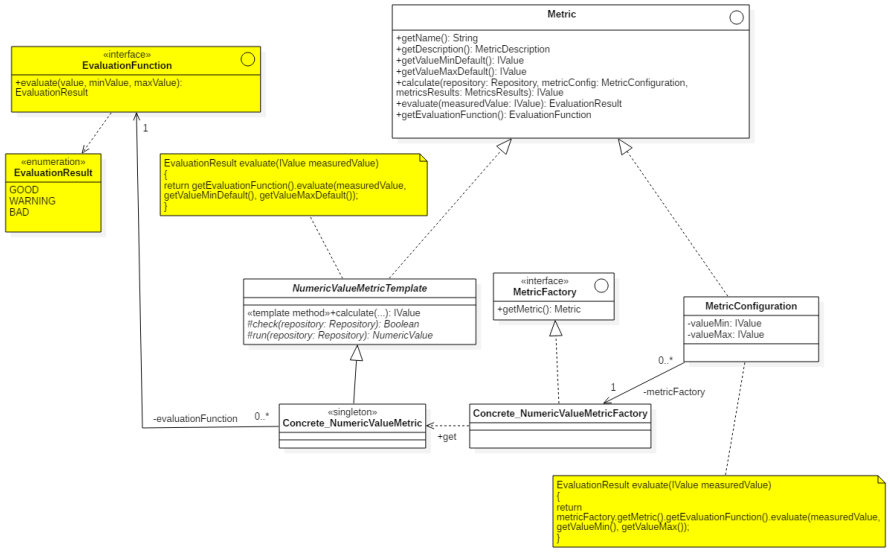


Figura C.3: Añadido al framework de medición la evaluación de métricas

Paquete gui

En este paquete se define toda la interfaz gráfica.

Paquete exceptions

En este paquete se encuentran las excepciones de la aplicación, además se trabaja con códigos de error y mensajes para poder ser identificados y comprendidos en el logger fácilmente. Se ha creado un diseño en el que, para definir nuevas excepciones, se puede extender de *ApplicationException*, copiar y adaptar los constructores y modificar con *@Override* el método *generateMessage()* con los mensajes personalizados en un *switch* para cada código de error:

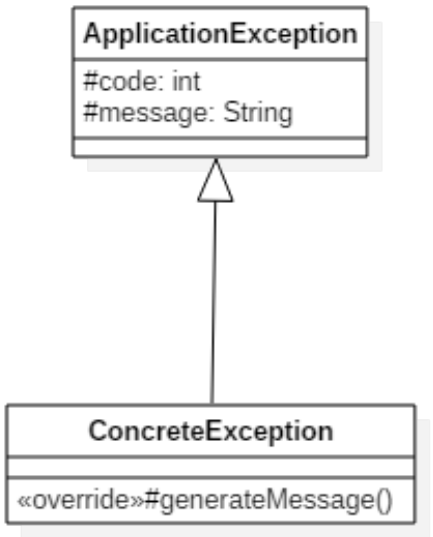


Figura C.4: Paquete exceptions

Trabajando con este diseño se han podido implementar excepciones para las nuevas métricas.

Paquete datamodel

El paquete datamodel contiene las diferentes clases que se han descrito anteriormente en la sección C.2. Si en un futuro se integraran nuevas forjas de repositorios puede que sea necesario añadir nuevas clases de envoltura para serializar los valores que obtengamos como se ha visto anteriormente.

Paquete app

Contiene las clases que sirven de conexión entre la interfaz de usuario y la lógica de la aplicación.

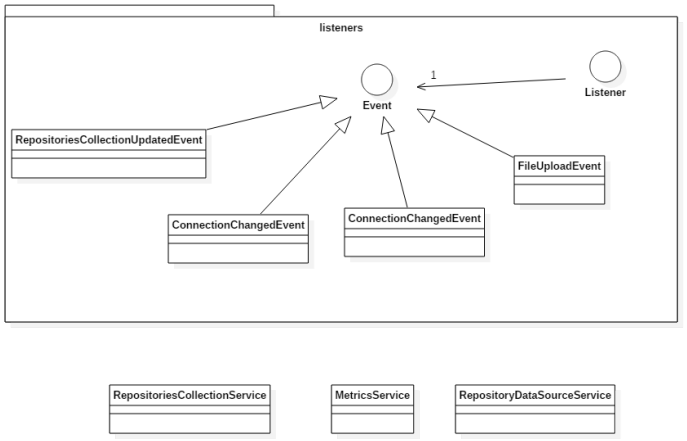


Figura C.5: Paquete app

Las clases que encontramos en este paquete son:

MetricsService Define una fachada de conexión entre el motor de métricas y el resto de componentes.

RepositoriesCollectionService Se encarga de almacenar los repositorios en una colección

RepositoriesCollectionService Define una fachada entre la fuente de datos (*repositorydatasource*) y el resto de componentes.

Paquete app.listeners

En este paquete trabajamos con el patrón observador. Se definen la interfaz *Listener* y la interfaz *Event* y se implementan algunos eventos como el de conexión a una forja o el de subida de un archivo para realizar diferentes acciones cuando se den dichos eventos.
La interfaz *Event* sirve para implementar eventos a partir de ella, estos eventos normalmente tienen información relativa al suceso que ocurre. Por ejemplo, *CurrentMetricProfileChangedEvent* contiene información sobre un cambio de perfil de métricas: el viejo perfil y el nuevo. La interfaz *Listener* define una función a la que se le pasa un evento y sirve como interfaz funcional

para ejecutar una acción (función lambda) que recibe por parámetro el evento.

Apéndice D

Documentación técnica de programación

D.1. Introducción

En este documento de tratan diferentes aspectos técnicos de programación.

D.2. Estructura de directorios

En el proyecto el código fuente presenta la siguiente estructura:

- /.gitignore. Contiene los ficheros y directorios que el repositorio git no tendrá en cuenta
- /.github/workflows/maven.yml. Contiene los *jobs* a ejecutar gracias a *GitHub Actions* tras hacer un *commit* o un *merge* a la rama **main**. Permite y define la integración y el despliegue continuo.
- /README.md. Fichero que sirve de introducción al proyecto y que contiene información importante.
- /pom.xml. Fichero de configuración del proyecto Maven.
- /system.properties. Fichero con propiedades del proyecto necesario para el despliegue en Heroku.
- /MemoriaProyecto. Memoria del proyecto según la plantilla definida en https://github.com/ubutfgm/plantillaLatex.
- /src/test/resources. Datos almacenados en ficheros CSV para proporcionar datos a test parametrizados.

- /src/test/java. Casos de prueba JUnit para la realización de pruebas. Se organiza de la misma forma que /src/main/java
- /src/main/webapp/VAADIN/themes/MyTheme. Tema principal utilizado por la aplicación el cual es generado por Vaadin.
- /src/main/webapp/frontend. Fichero *.css* y los diferentes iconos utilizado por la interfaz gráfica.
- /src/main/webapp/images. Imágenes que se muestran en la interfaz gráfica.
- /src/main/resources. Ficheros de configuración de la aplicación. En este caso el fichero log4j2.properties para configurar el sistema de logging.
- /src/main/java. Contiene todo el código fuente.
- /src/main/java/app/. Contiene fachadas que conectan la interfaz de usuario con el resto de componentes que componen la lógica de la aplicación.
- /src/main/java/app/listeners. Contiene observadores y eventos utilizados por la aplicación.
- /src/main/java/datamodel. Contiene las clases que representan el modelo de datos de la aplicación.
- /src/main/java/exceptions. Contiene excepciones definidas en la aplicación.
- /src/main/java/gui. Contiene la interfaz de usuario.
- /src/main/java/gui/views. Contiene páginas y componentes de Vaadin que componen la interfaz gráfica de la aplicación.
- /src/main/java/metricsengine. Contiene el motor de métricas.
- /src/main/java/metricsengine/numeric_value_metrics. Métricas definidas y sus respectivas fábricas (Patrón de diseño método fábrica[1]). Todas las métricas tienen resultados numéricos.
- /src/main/java/metricsengine/values. Valores que devuelven las métricas.
- /src/main/java/repositorydatasource. Framework de conexión a las diferentes forjas de repositorios como GitHub o GitLab.

D.3. Manual del programador

En este apartado se tratan algunas bases para entender como continuar trabajando en las diferentes implementaciones de la aplicación y los posibles puntos donde trabajar.

[1] https://refactoring.guru/design-patterns/factory-method

Framework de conexión

El framework de conexión a una forja de repositorios (como GitHub) está definido en el paquete *repositorydatasource*. Consta de dos interfaces, la más importante es la interfaz *RepositoryDataSource*.

En el Anexo C, se muestra un diagrama de clases en la Fig. C.1. El paquete *repositorydatasource* consta de dos interfaces que se han de implementar de forma particular para cada forja para así conectar el API de la forja de repositorios con el resto de la aplicación.
Una solo es una fábrica que sirve instanciar un *RepositoryDataSource* como por ejemplo *RepositoryDataSourceUsingGithubAPI*.
Esta última es la que contiene las funciones para establecer una conexión y obtener los datos de los repositorios. Una vez tengamos implementadas estas interfaces para una nueva forja de repositorios se tiene que cambiar la llamada a la fábrica a la nueva, por ejemplo:

```
...
public class RepositoryDataSourceFactoryGithub implements RepositoryDataSourceFactory {

@Override
public RepositoryDataSource getRepositoryDataSource() {
return RepositoryDataSourceUsingGithubAPI.getGithubRepositoryDataSource();
}

}
...
```

Motor de métricas

El motor de métricas se implementó con una base inicial a una solución propuesta en *Soporte de Métricas con Independencia del Lenguaje para la Inferencia de Refactorizaciones* [6]. El diseño se puede observar en las figuras Fig. C.2 y Fig. C.3.

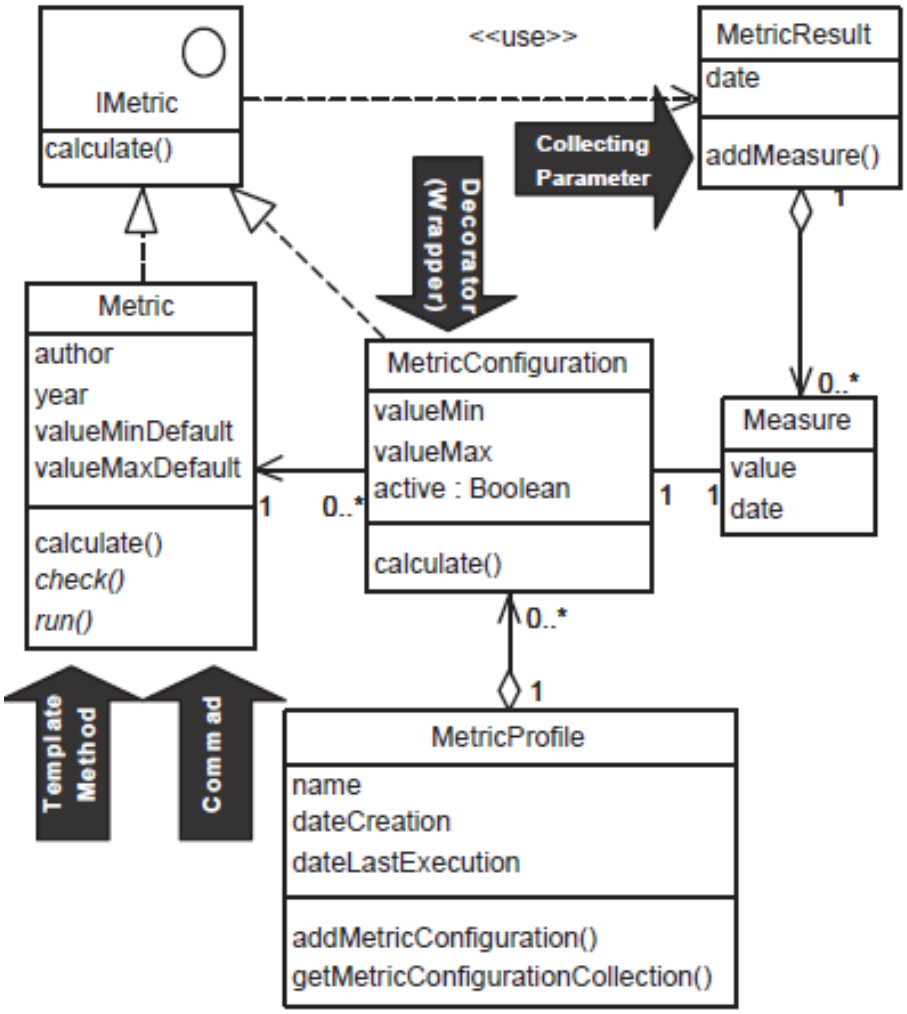


Figura D.1: Diagrama del Framework para el cálculo de métricas con perfiles.

Trabajando con este diseño se han podido implementar las nuevas métricas relacionadas con la Integración y Despliegue continuos (CICD) y de la misma manera se podrían añadir más métricas a al proyecto.

D.4. Compilación, instalación y ejecución del proyecto

Para compilar el proyecto es necesario tener Java 11 y Maven 3.6.0 o superior instalados en el equipo. Para ambas herramientas, el proceso de

instalación es el mismo: descomprimir un archivo en la carpeta que se desee, configurar las variables de entorno del sistema JAVA_JOME y CATALINA_HOME apuntando a los directorios de instalación que contienen la carpeta bin, y añadir al PATH del sistema la ruta hacia los directorios bin.

Una vez instalados Java y Maven, para compilar se utilizaría el comando mvn install desde el directorio raíz del proyecto.

Para ejecutar el proyecto en nuestra máquina utilizando Jetty (nos ayuda al detectar automáticamente cambios en el código y recompilando automáticamente) basta con ejecutar: mvn jetty:run tras haber instalado la aplicación con Maven.

Para generar un .war compilando la aplicación es necesario hacerlo en modo producción, para ello ejecutaremos: mvn clean package -Pproduction.

Por último, para desplegar la aplicación desde la consola (si no tenemos configurado el despliegue continuo), tendremos que ejecutar: heroku login o bien winpty heroku.cmd login
Hacer login en Heroku con el navegador y tras esto ejecutar:
heroku war:deploy target/[nombre-de-la-aplicacion]-[versión].war –app [nombre-de-la-aplicación-en-Heroku]

D.5. Pruebas del sistema

Se ha generado una batería de pruebas en src/test/java. Algunos de estos test son parametrizados y los valores se encuentran en ficheros *.csv* en la carpeta src/test/resources.

Para ejecutar las pruebas y obtener informes de cobertura de código podemos hacerlo desde el IDE Eclipse (*Coverage As -> JUnit Test* o bien desde consola corriendo:
mvn jacoco:report
o si lo necesitamos inicialmente:
mvn clean jacoco:prepare-agent install jacoco:report

Apéndice E

Documentación de usuario

E.1. Introducción

Este documento detalla cómo un usuario puede utilizar la aplicación.

E.2. Requisitos de usuarios

Los requisitos para poder utilizar la aplicación son:

- Que la aplicación esté desplegada en algún servidor. En este caso se encuentra desplegada en https://evolution-metrics-v2.herokuapp.com/.
- Disponer de conexión a internet (o al servidor donde esté alojada la aplicación) y tener instalado un navegador web con el que poder acceder a la aplicación. Se recomienda en sus versiones lo más actualizadas posible los siguientes navegadores:
  - Google Chrome
  - Mozilla Firefox
  - Microsoft Edge
  - Opera
  - Safari

E.3. Instalación

Al tratarse de una aplicación web, esta no requiere instalación, basta con que el usuario acceda a la url de la aplicación desplegada. Además, si

E.4. Manual del usuario

Para la utilización de la aplicación es necesario conectarse a las forjas disponibles (GitHub y GitLab), tras esto la aplicación recibe como entrada los datos de diferentes repositorios, calcula las métricas para los mismos y permite comparar los repositorios utilizando el cálculo estadístico de cuartiles.

Conceptos

Se deben conocer los siguientes conceptos:

Medición

La medición es un proceso en el que se asignan números o símbolos a atributos de entidades del mundo real, de tal forma que los caracteriza a través de reglas. En nuestro caso realizamos mediciones de diferentes aspectos de los repositorios de software.

Métrica

Medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo dado (IEEE, 1993).

Indicador

Métrica o combinación de métricas que proporcionan una visión profunda del proceso, del proyecto o del producto.

Métrica de evolución

Es una métrica que mide un atributo del proceso de desarrollo de un producto software.

Evaluación

Consiste en determinar el estado de un proyecto en relación con otros proyectos de la misma naturaleza es uno de los objetivos clave del proceso de medición.

Proyecto (software)

Proyecto en el cual se desarrolla un producto software.

Repositorio de código

Lugar dónde se almacena el código de un proyecto software. A menudo cuentan con un sistema de control de versiones.

Forja de repositorios

Lugar dónde se almacenan múltiples repositorios de código tanto públicos como privados de gran cantidad de usuarios o grupos. Cuentan con múltiples sistemas que facilitan la comunicación entre los desarrolladores y mejoran el soporte al usuario. Este proyecto se encuentra alojado en la forja GitHub.

Sistema de control de veriones (VCS - Version Control System)

Sistema que registra los cambios que se producen sobre los ficheros de un proyecto software almacenados en un repositorio de código.

Sistema de seguimiento de incidencias (Issue tracking system)

Sistema que gestiona las diferentes tareas o incidentes que se definen en un proyecto software y que pueden ser asignadas a colaboradores del proyecto. En este proyecto se ha empleado el sistema GitHub Issues.

Las métricas que se gestionan en la aplicación

Además de los conceptos anteriores, para comprender la aplicación y poder sacarle partidos, se tienen que conocer las métricas que ésta calcula y saber interpretarlas. La métricas proceden de la Master Tesis titulada *sPACE: Software Project Assessment in the Course of Evolution* [8] a las que se han añadido nuevas métricas relacionas con la Integración y Despliegue continuos:

Las métricas con las que se trabaja en la aplicación son las siguientes:

I1 - Número total de issues (incidencias)

- **Categoría**: Proceso de Orientación
- **Descripción**: Número total de issues creadas en el repositorio
- **Propósito**: ¿Cuántas issues se han definido en el repositorio?
- **Fórmula**: NTI. NTI = *número total de issues*
- **Fuente de medición**: Proyecto en una plataforma de desarrollo colaborativo.
- **Interpretación**: $NTI \geq 0$. Valores bajos indican que no se utiliza un sistema de seguimiento de incidencias, podría ser porque el proyecto acaba de comenzar

- **Tipo de escala**: Absoluta
- **Tipo de medida**: *NTI = Contador*

I2 - Commits (cambios) por issue

- **Categoría**: Proceso de Orientación
- **Descripción**: Número de commits por issue
- **Propósito**: ¿Cuál es el volumen medio de trabajo de las issues?
- **Fórmula**: $CI = \frac{NTC}{NTI}$. *CI = Cambios por issue, NTC = Número total de commits, NTI = Numero total de issues*
- **Fuente de medición**: Proyecto en una plataforma de desarrollo colaborativo.
- **Interpretación**: $CI \geq 1$, Lo normal son valores altos. Si el valor es menor que uno significa que hay desarrollo sin documentar.
- **Tipo de escala**: Ratio
- **Tipo de medida**: *NTC, NTI = Contador*

I3 - Porcentaje de issues cerradas

- **Categoría**: Proceso de Orientación
- **Descripción**: Porcentaje de issues cerradas
- **Propósito**: ¿Qué porcentaje de issues definidas en el repositorio se han cerrado?
- **Fórmula**: $PIC = \frac{NTIC}{NTI} * 100$. *PIC = Porcentaje de issues cerradas, NTIC = Número total de issues cerradas, NTI = Numero total de issues*
- **Fuente de medición**: Proyecto en una plataforma de desarrollo colaborativo.
- **Interpretación**: $0 \leq PIC \leq 100$. Cuanto más alto mejor
- **Tipo de escala**: Ratio

- **Tipo de medida**: *NTI, NTIC = Contador*

TI1 - Media de días en cerrar una issue

- **Categoría**: Constantes de tiempo
- **Descripción**: Media de días en cerrar una issue
- **Propósito**: ¿Cuánto se suele tardar en cerrar una issue?
- **Fórmula**: $MDCI = \frac{\sum_{i=0}^{NTIC} DCI_i}{NTIC}$. *MDCI = Media de días en cerrar una issue, NTIC = Número total de issues cerradas, DCI = Días en cerrar la issue*
- **Fuente de medición**: Proyecto en una plataforma de desarrollo colaborativo.
- **Interpretación**: $MDCI \geq 0$. Cuanto más pequeño mejor. Si se siguen metodologías ágiles de desarrollo iterativo e incremental como SCRUM, la métrica debería indicar la duración del *sprint* definido en la fase de planificación del proyecto. En SCRUM se recomiendan duraciones del *sprint* de entre una y seis semanas, siendo recomendable que no exceda de un mes [7].
- **Tipo de escala**: Ratio
- **Tipo de medida**: *NTI, NTIC = Contador*

TC1 - Media de días entre commits

- **Categoría**: Constantes de tiempo
- **Descripción**: Media de días que pasan entre dos commits consecutivos
- **Propósito**: ¿Cuántos días suelen pasar desde un commit hasta el siguiente?
- **Fórmula**: $MDC = \frac{\sum_{i=1}^{NTC} TC_i - TC_{i-1}}{NTC}$. $TC_i - TC_{i-1}$ en días; *MDC = Media de días entre cambios, NTC = Número total de commits, TC = Tiempo de commit*
- **Fuente de medición**: Proyecto en una plataforma de desarrollo colaborativo.

- **Interpretación**: $MDEC \geq 0$. Cuanto más pequeño mejor. Se recomienda no superar los 5 días.
- **Tipo de escala**: Ratio
- **Tipo de medida**: *NTC = Contador; TC = Tiempo*

TC2 - Días entre primer y último commit

- **Categoría**: Constantes de tiempo
- **Descripción**: Días transcurridos entre el primer y el ultimo commit
- **Propósito**: ¿Cuantos días han pasado entre el primer y el último commit?
- **Fórmula**: $DEPUC = TC2 - TC1$. $TC2 - TC1$ en días; *DEPUC = Días entre primer y último commit, TC2 = Tiempo de último commit, TC1 = Tiempo de primer commit*
- **Fuente de medición**: Proyecto en una plataforma de desarrollo colaborativo.
- **Interpretación**: $DEPUC \geq 0$. Cuanto más alto, más tiempo lleva en desarrollo el proyecto. En procesos software empresariales se debería comparar con la estimación temporal de la fase de planificación.
- **Tipo de escala**: Absoluta
- **Tipo de medida**: *TC = Tiempo*

TC3 - Ratio de actividad de commits por mes

- **Categoría**: Constantes de tiempo
- **Descripción**: Muestra el número de commits relativos al número de meses
- **Propósito**:¿Cuál es el número medio de cambios por mes?
- **Fórmula**: $RCM = \frac{NTC}{NM}$. *RCM = Ratio de cambios por mes, NTC = Número total de commits, NM = Número de meses que han pasado durante el desarrollo de la aplicación*

- **Fuente de medición**: Proyecto en una plataforma de desarrollo colaborativo.
- **Interpretación**: $RCM > 0$. Cuanto más alto mejor
- **Tipo de escala**: Ratio
- **Tipo de medida**: NTC = *Contador*

C1 - Cambios pico

- **Categoría**: Constantes de tiempo
- **Descripción**: Número de commits en el mes que más commits se han realizado en relación con el número total de commits
- **Propósito**: ¿Cuál es la proporción de trabajo realizado en el mes con mayor número de cambios?
- **Fórmula**: $CP = \frac{NCMP}{NTC}$. CP = *Cambios pico,* $NCMP$ = *Número de commits en el mes pico,* NTC = *Número total de commits*
- **Fuente de medición**: Proyecto en una plataforma de desarrollo colaborativo.
- **Interpretación**: $0 \leq CCP \leq 1$. Mejor valores intermedios. Se recomienda no superar el 40 % del trabajo en un mes.
- **Tipo de escala**: Ratio
- **Tipo de medida**: $NCMP$, NTC = *Contador*

IC1 - Número total de *jobs* ejecutados

- **Categoría**: CI-CD
- **Descripción**: Número de *jobs* ejecutados en el proyecto.
- **Propósito**: ¿Cuál es el número total de de *jobs* ejecutados con éxito en el proyecto?
- **Fórmula**: NJE = *número total de jobs.*
- **Fuente de medición**: Proyecto en una plataforma de desarrollo colaborativo.

- **Interpretación**: $NJE \geq 0$. Un valor de cero indica que el proyecto no tiene integración y despliegues continuos y no se ha realizado ningún trabajo automatizado de despliegue.
- **Tipo de escala**: Absoluta
- **Tipo de medida**: *NJE = Contador*

IC2 - Número de *Jobs* ejecutados el último año

- **Categoría**: CI-CD
- **Descripción**: Número de *jobs* ejecutados en el último año (últimos 365 días).
- **Propósito**: ¿Cuál es el número total de *jobs* ejecutados con éxito en el proyecto durante el año previo?
- **Fórmula**: $NJELY$ = *número total de jobs ejecutados el útimo año.*
- **Fuente de medición**: Proyecto en una plataforma de desarrollo colaborativo.
- **Interpretación**: $NJELY \geq 0$. Un valor de cero indica que en el proyecto no se ha realizado ningún trabajo automatizado de despliegue durante el último año.
- **Tipo de escala**: Absoluta
- **Tipo de medida**: *NJELY = Contador*

IC3 - *N*úmero de tipos diferentes dejobs ejecutados

- **Categoría**: CI-CD
- **Descripción**: Número de tipos diferentes de *jobs* ejecutados en el proyecto.
- **Propósito**: ¿Cuál es el número total de tipos de *jobs* ejecutados con éxito en el proyecto?
- **Fórmula**: $NTJE$ = *número total de tipos diferentes de jobs ejecutados en el proyecto.*

- **Fuente de medición**: Proyecto en una plataforma de desarrollo colaborativo.
- **Interpretación**: $NTJE \geq 0$. Un valor de cero indica que el proyecto no tiene integración y despliegues continuos y no se ha realizado ningún trabajo automatizado de despliegue.
- **Tipo de escala**: Absoluta
- **Tipo de medida**: $NTJE = Contador$

DC1 - *N*úmero total dereleases

- **Categoría**: CI-CD
- **Descripción**: Número de *releases* lanzadas en el proyecto.
- **Propósito**: ¿Cuál es el número total de *releases* del proyecto?
- **Fórmula**: $NRR = número\ total\ de\ releases\ lanzadas\ en\ el\ proyecto.$
- **Fuente de medición**: Proyecto en una plataforma de desarrollo colaborativo.
- **Interpretación**: $NRR \geq 0$. Un valor de cero indica que aún no se ha lanzado ninguna *release* del proyecto.
- **Tipo de escala**: Absoluta
- **Tipo de medida**: $NRR = Contador$

Arranque de la aplicación: Establecer una conexión a GitLab

Para poder añadir repositorios desde las forjas es necesario realizar una conexión a cada una de ellas a través de los botones que podemos encontrar en la parte superior derecha de la interfaz como se ve en la siguiente figura E.3.

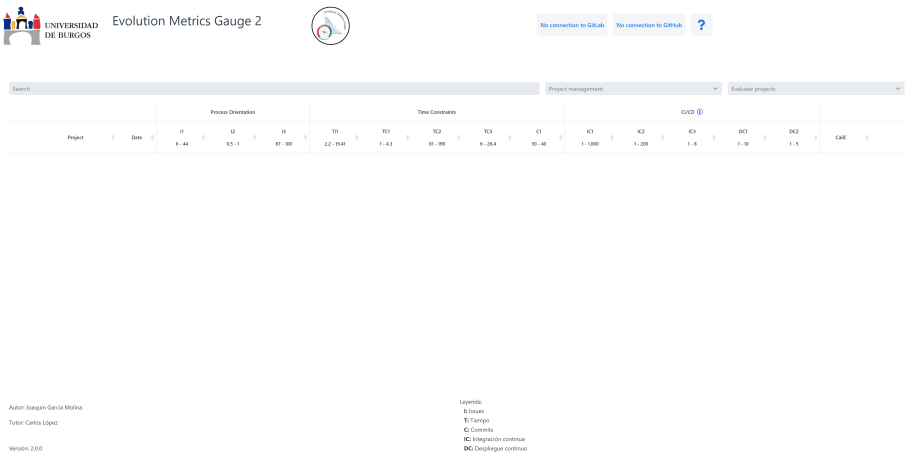
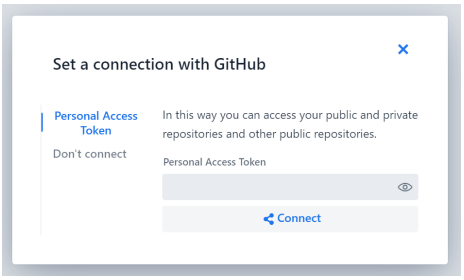
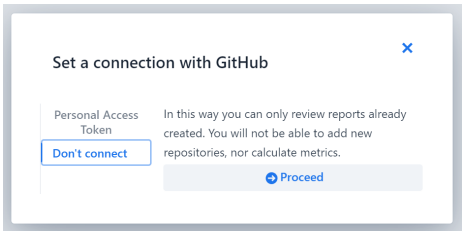


Figura E.1: Botones de conexión

Para realizar la conexión con GitHub debido a cómo trabaja su API sólo podemos conectarnos utilizando un token personal [1]:



(a) Establecer conexión iniciando sesión mediante *Personal Access Token*



(b) No establecer conexión a GitHub

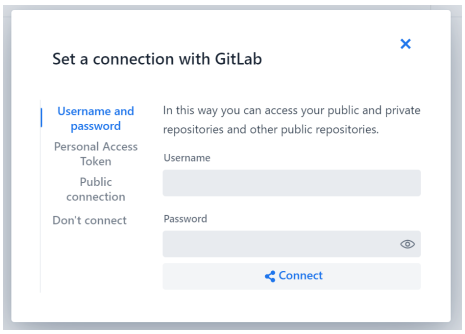
Figura E.2: Conexión con GitHub

Para realizar la conexión con GitLab tenemos varias opciones [2] :

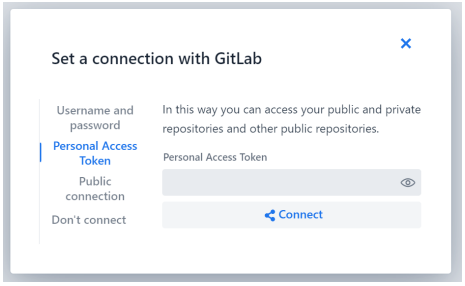
Hay 4 posibilidades, que permiten establecer una conexión con la sesión iniciada (mediante usuario y contraseña o mediante *PA Token*), una conexión pública o utilizar la aplicación sin conexión:

[1] Crear un token de acceso personal para GitHub: https://docs.github.com/es/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token

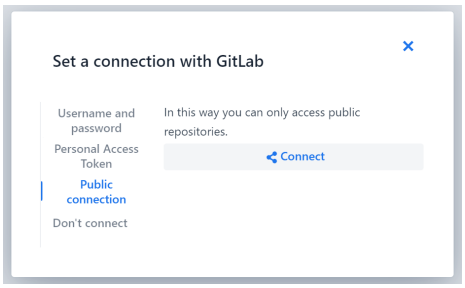
[2] Crear un token de acceso personal para GitLab: https://docs.gitlab.com/ee/user/profile/personal_access_tokens.html



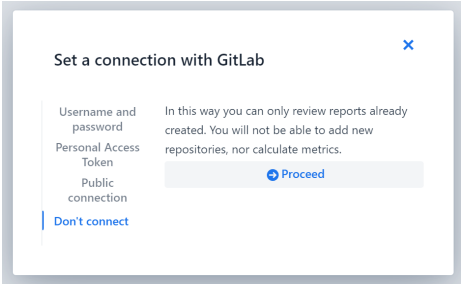
(a) Establecer conexión iniciando sesión mediante usuario y contraseña



(b) Establecer conexión iniciando sesión mediante *Personal Access Token*



(c) Establecer una conexión publica



(d) No establecer conexión a GitLab

Figura E.3: Distintas formas de establecer una conexión con GitLab

Iniciar sesión en GitLab mediante usuario y contraseña. Se establece una conexión a GitLab iniciando sesión mediante un nombre de usuario y una contraseña. De esta forma se puede acceder a todos los repositorios públicos y privados accesibles por el usuario. Ver figura E.3a.

Iniciar sesión mediante *Personal Access Token*. Se establece una conexión iniciando sesión mediante un *Personal Access Token*. De esta forma se puede acceder a todos los repositorios públicos y privados accesibles por usuario, como ocurría en el caso anterior. Si se accede a GitLab desde una cuenta externa a GitLab como Google o GitHub, esta opción es la única manera de iniciar sesión con su cuenta de GitLab. Ver figura E.4.

Usar una conexión pública hacia GitLab. Se establece una conexión pública a GitLab sin iniciar sesión, por lo que solo se podrá acceder a repositorios públicos, no a los privados. Ver figura E.3c.
Además es importante indicar que no se calcularán las métricas relacionadas con CICD ya que la API de GitLab no devuelve los valores

necesarios para calcularlas con este tipo de conexión (se necesita alguna de las dos anteriores).

No utilizar ninguna conexión. No se realizará ninguna conexión, de esta forma se impide al usuario añadir nuevos repositorios desde las dos forjas ni volver a calcular métricas de los proyectos existentes. Sin embargo, se permite gestionar los proyectos ya existentes, permitiendo su exportación e importación desde un fichero con extensión .emr. También se permite usar el sistema de perfil de métricas sin ninguna restricción: se puede crear un nuevo perfil, importar, exportar y utilizar el perfil por defecto de la aplicación. Ver figura E.3d.

Página principal

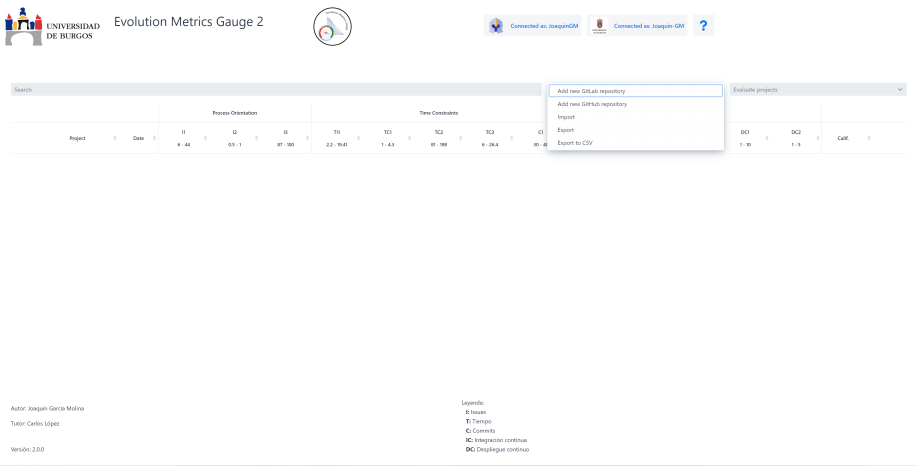


Figura E.4: Página principal

Una vez establecidas las conexiones, se puede comenzar a trabajar con los repositorios como se observa en la figura **??**.

Cambiar el tipo de conexión

En la parte superior se puede observar el botón de conexión, que indica el tipo de conexión actual.

- Si se ha iniciado sesión mediante usuario y contraseña o mediante un *personal access token*, se mostrará la imágen del usuario y el texto: "Connected as: <nombre de usuario>"
- Si se ha establecido una conexión pública, se mostrara el texto: "Using a public connection"
- Y si no se ha establecido ninguna conexión, el texto mostrado será: "No connection to GitHub/GitLab".

Para cambiar el tipo de conexión es obligatorio cerrar (si existe), la conexión actual. Por ello, al pulsar sobre el botón de conexión, se muestra el diálogo de la figura E.5b si existe una conexión y el diálogo de la figura E.5d si no existe conexión. Al pulsar sobre "*Connect*" o sobre "*Close connection*", respectivamente, se abrirá alguno de los7 diálogos de conexión de la figura E.3

(a) Cerrar conexión

(b) Conectado como...

(c) Sin conexión a GitLab

(d) Sin conexión a GitHub

Figura E.5: Modificar tipo de conexión

Botón de ayuda

A la derecha del botón de conexión se encuentra un botón que da acceso a este manual en la Wiki del proyecto: https://github.com/Joaquin-GM/GII_O_MA_19.07-Comparador-de-metricas-de-evolucion-en-repositorios-Software/wiki.

Listado de proyectos

En la parte central de la interfaz principal se gestionan los proyectos. Consta de una barra de búsqueda, dos menús, y una tabla que visualiza las métricas de los proyectos que se añadan.

En el **cuadro de búsqueda** se filtrarán los repositorios por su nombre según se vaya escribiendo.

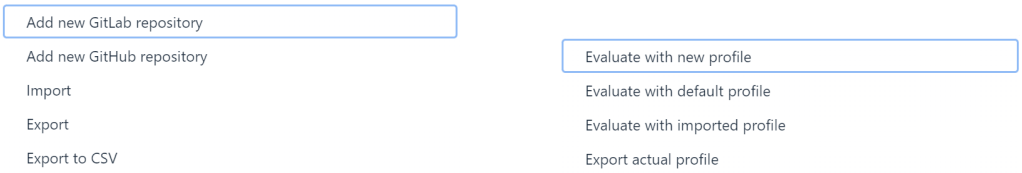
En el **menú de "*Project management*"** existen las siguientes opciones, como se muestra en la figura E.6a:

- ***Add new GitLab repository.*** Permite añadir un nuevo repositorio de GitLab (disponible sólo con conexión).
- ***Add new GitHub repository.*** Permite añadir un nuevo repositorio de GitHub (disponible sólo con conexión).

- ***Import.*** Permite importar proyectos a partir de un fichero previamente exportado.
- ***Export.*** Permite exportar todos los proyectos existentes a un fichero, lo que permitirá su posterior importación. Se almacena en un fichero con foromato ".emr".
- ***Export to CSV.*** Permite generar un fichero CSV que contenga toda la información de la tabla de proyectos. Este fichero no servirá para importar los proyectos posteriormente si no que sirve para realizar análisis de los resultados obtenidos por ejemplo en una hoja de cálculo.

En el **menú de "*Evaluate projects*"** existen estas opciones, como se muestra en la figura E.6b:

- ***Evaluate with new profile.*** Permite evaluar los proyectos calculando los valores umbrales de cada métrica a partir de los repositorios actuales.
- ***Evaluate with default profile.*** Permite evaluar los proyectos con un perfil por defecto creado a a partir de un conjunto de datos[3] de un estudio empírico de las métricas de evolución del software en trabajos finales de grado[5].
- ***Evaluate with imported profile.*** Permite evaluar los proyectos a partir de un perfil de métricas previamente exportado.
- ***Export actual profile.*** Permite exportar el perfil de métricas actual para su posterior importación. Se almacena en un fichero con foromato ".emmp".



(a) Menú: "*Project management*" (b) Menú: "*Evaluate projects*"

Figura E.6: Menús del listado de repositorios

La **tabla** muestra los valores medidos de las métricas para cada proyecto, ver figura E.7.

[3] https://github.com/clopezno/clopezno.github.io/blob/master/agile_practices_experiment/DataSet_EvolutionSoftwareMetrics_FYP.csv

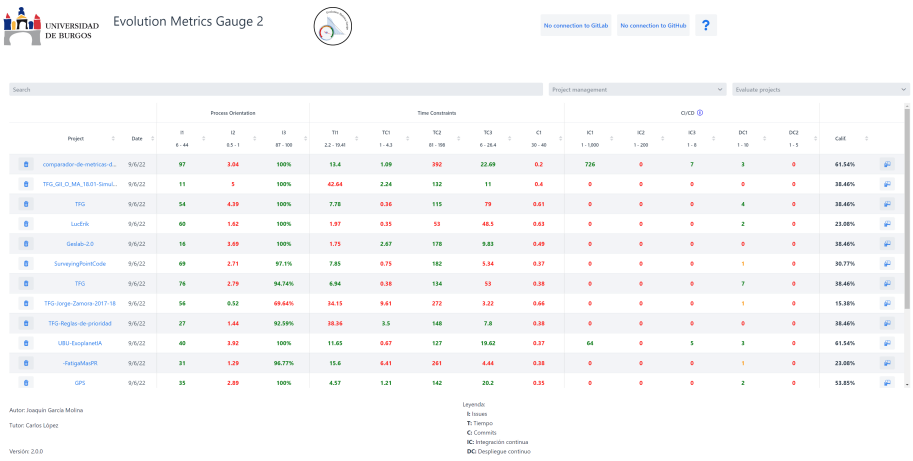


Figura E.7: Tabla que muestra los valores medidos de las métricas para cada proyecto

Nótese que se tienen los resultados sin conexiones ya que se han importado por archivo.
La tabla presenta las siguientes columnas:

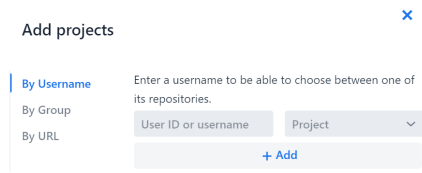
- **Botón de eliminar.** Permite eliminar de la tabla el proyecto seleccionado.
- ***Project.*** Nombre del proyecto con enlace al repositorio de GitLab. Si el nombre es demasiado largo y se corta, se puede utilizar el tooltip que aparece al pasar el ratón por encima del nombre del proyecto. Puede que el enlace no funcione si el proyecto se ha eliminado o si no se tienen los permisos de acceso según la sesión de GitLab DEL NAVEGADOR (no de la aplicación).
- ***Date.*** Fecha de la última vez que se obtuvieron las métricas del proyecto.
- **Métricas.** Valor medido de las métricas y un color que evalúa la medida en relación a un perfil de métricas.
  Las métricas están clasificadas por categoría: Proceso de orientación (*Process Orientation*), Constantes de tiempo (*Time Constraints*) e Integración y Despliegue continuos (*CICD*)
  En la cabecera se muestra el nombre de la métrica pero aparecerá la descripción en forma de tooltip al pasar el puntero del ratón por encima del nombre. Un **perfil de métricas** es un conjunto de valores mínimo y máximo definidos para cada métrica. Los valores que se

encuentran debajo de cada nombre de las métricas en la cabecera son sus valores mínimo y máximo separados por un guión. En el tooltip se muestra el valor mínimo como Q1 y el valor máximo cómo Q3.

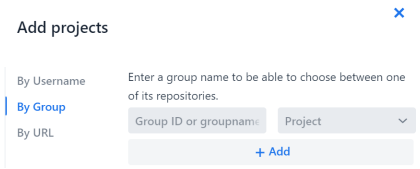
- **Botón de recalcular métricas.** Permite volver a obtener las métricas del proyecto (si es posible según la conexión actual a GitLab de la aplicación) y evaluar las nuevas métricas de acuerdo al perfil actual. Se mostrará un mensaje de aviso si se han recalculado correctamente y un mensaje de error en caso contrario.

Añadir un proyecto

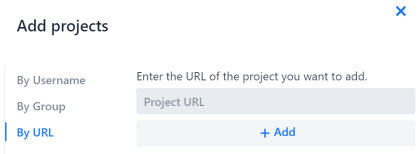
Para añadir un nuevo proyecto, el tipo de conexión deberá ser distinto de "Sin conexión" (*No connection to GitLab*), es decir que debe haber una conexión. Seleccionar la opción "*Add new*" del menú "*Project management*", ver figura E.6a. Se abrirá un diálogo como el de la figura E.8. Para cancelar se puede pulsar *Esc* o hacer clic fuera del diálogo. Existen tres posibilidades



(a) Menú: "*Añadir por pertenencia a usuario*"



(b) Menú: "*Añadir por pertenencia a grupo*"



(c) Menú: "*Añadir proyecto mediante su URL Web*"

Figura E.8: Menús del listado de repositorios

para añadir un proyecto:

Añadir por pertenencia a un usuario. Se solicita en el campo izquierdo del formulario el nombre de usuario o su ID de GitLab del cual se desean cargar los proyectos en campo desplegable de la derecha. Se mostrará un mensaje en rojo si el usuario no se existe "*User not*

found" y un mensaje si el usuario existe "*User found*", en ese caso se cargarán todos los proyectos del usuario en el desplegable de la derecha. Seleccionar uno de sus proyectos y pulsar sobre el botón "*Add*". Se mostrarán los repositorios públicos (incluyendo *forks*) del usuario. No se mostrarán proyectos privados a menos que se haya establecido una conexión con sesión y el usuario especificado en el campo de la izquierda coincida con el usuario que haya iniciado sesión.

Añadir por pertenencia a un grupo. El funcionamiento es el mismo que en el caso anterior, con la diferencia de que se solicita el nombre de grupo o ID del grupo en el campo izquierdo. Si el grupo es privado y el tipo de conexión es pública o el usuario que haya iniciado sesión no tiene acceso al grupo, se mostrará un mensaje en rojo de la misma forma que si el grupo no existiera "*Group not found*". Si se encuentra el grupo, se mostrará el mensaje "*Group found*" y se cargarán todos los proyectos del usuario en el desplegable de la derecha. Seleccionar uno de sus proyectos y pulsar sobre el botón "*Add*".

Añadir por URL Web. Se solicita la URL Web del proyecto de GitLab. Si no se encuentra, ya sea porque no existe o por la conexión actual, se mostrará un mensaje en rojo al pulsar sobre "***Add***": "*Project not found. It doesn't exists or may be inaccessible due to your connection level.*"

Al añadir un nuevo proyecto, se calcularán por primera vez sus métricas y se evaluarán de acuerdo al perfil de métricas actual. Si no se ha creado o importado ningún perfil, se evaluará según el perfil por defecto.

No se permite volver a añadir un proyecto existente a la tabla.

Eliminar un proyecto

Para eliminar un proyecto basta con identificarlo en la tabla y pulsar sobre el icono de basura correspondiente, situado a la izquierda de la tabla. **Atención**: NO se pide confirmación de eliminación.

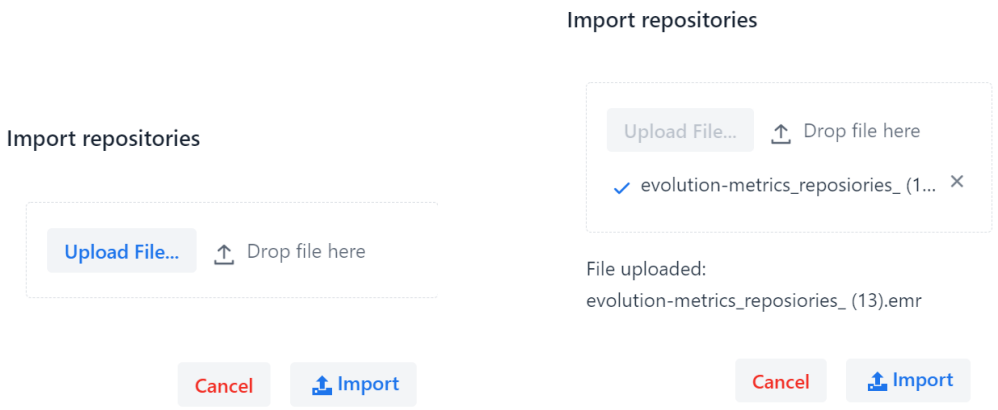
Recalcular métricas de un proyecto

Para recalcular las métricas de un proyecto debe haber una conexión a la forja correspondiente y esta debe tener permisos sobre el proyecto deseado (si es un proyecto privado, debe haber iniciado sesión una persona con permisos de lectura sobre el proyecto). Cumpliendo esos requisitos, solo

hay que identificar el proyecto y pulsar sobre el botón correspondiente de recalcular, situado en la parte derecha de la tabla.

Importar proyectos

Los proyectos se pueden importar independientemente del tipo de conexión que exista. Se mostrará un diálogo que permite seleccionar o arrastrar al cuadro un fichero con formato *.emr*, ver figura E.9. Se puede seleccionar otro fichero en caso de haber escogido un fichero no deseado. Una vez se cargue el fichero, pulsar sobre "Import". Se mostrará un mensaje de error en caso de que el fichero este corrompido (ha sido modificado por una herramienta externa a la aplicación), lo que indica que el fichero no podrá volver a utilizarse.



(a) Diálogo para importar repositorios

(b) Diálogo de importar con fichero cargado

Figura E.9: Diálogo para importar repositorios

Al pulsar sobre "Import" puede ocurrir:

- Que no haya ningún proyecto en la tabla, entonces se importarán todos los proyecto del fichero.
- Que haya algún proyecto en la tabla, entonces se preguntará si se desea añadir los proyecto al listado actual ("Append"); o sobrescribir el listado actual con los proyecto del fichero ("Overwrite"), en ese casó se borrará el listado actual y se añadirán los proyecto del fichero).
  En ningún caso se permite añadir proyectos que ya están en la tabla.

En caso de que el fichero contenga proyectos existentes, prevalecerán los de la tabla en caso de seleccionar "Append".

Exportar proyectos

Para exportar proyectos debe haber al menos un proyecto en la tabla. Se puede exportar a un fichero *.emr* para su posterior importación o en un fichero *.csv*. Para exportar hay que seleccionar la opción correspondiente en el menú "*Project management*", ver figura E.6a. El dialogo para la exportación se muestra en la figura E.10. Basta con pulsar sobre "*Download*" para poder descargar el fichero.

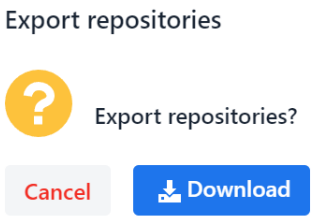


Figura E.10: Diálogo de exportación

También podemos descargarlos en formato CSV como se ha indicado anteriormente como se puede ver en la figura: E.11.

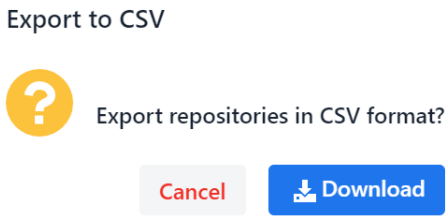


Figura E.11: Diálogo de exportación como CSV

Evaluar los proyectos

Evaluar un proyecto es comparar las métricas que se han medido de los proyectos en relación a un perfil de métricas en el que se definen los valores mínimos y máximos de cada métrica. El resultado de la evaluación puede

ser bueno (la medida se pinta en verde en la tabla), malo (la medida se pinta en rojo) o "advertencia" (la medida equivale al valor mínimo o al valor máximo).

Para evaluar los proyectos (se evalúan todos) hay que elegir el perfil de métricas con el que se va a evaluar. Por defecto se coge un perfil de métricas en el que los valores mínimos se corresponden con los cuartiles Q1 y los valores máximos con cuartiles Q3 de un conjunto de medidas tomadas sobre otros TFGs [4][**?**]. Se puede elegir otro perfil de métricas según la opción elegida en el menú "*Evaluate projects*" de la figura E.6b:

- ***Evaluate with new profile.*** Coge como entrada todas las medidas de la tabla y calcula, por cada métrica, los cuartiles Q1 y Q3 y los define como valor mínimo y valor máximo de la métrica.
- ***Evaluate with default profile.*** Permite evaluar los proyectos con el perfil por defecto mencionado anteriormente.
- ***Evaluate with imported profile.*** Permite importar el perfil de métricas de un fichero *.emmp*. El perfil se debe haber creado y exportado anteriormente.

Las métricas se evalúan como buenas si:

- I1: El valor medido supera el umbral mínimo (Q1).
- I2: El valor medido se encuentra entre el umbral mínimo y el máximo (Q3).
- I3: El valor medido supera el umbral mínimo.
- TI1: El valor medido se encuentra entre el umbral mínimo y el máximo.
- TC1: El valor medido se encuentra entre el umbral mínimo y el máximo.
- TC2: El valor medido se encuentra entre el umbral mínimo y el máximo.
- TC3: El valor medido se encuentra entre el umbral mínimo y el máximo.
- C1: El valor medido se encuentra entre el umbral mínimo y el máximo.

[4] https://github.com/clopezno/clopezno.github.io/blob/master/agile_practices_experiment/DataSet_EvolutionSoftwareMetrics_FYP.csv

- IC1: El valor medido se encuentra entre el umbral mínimo y el máximo. Si es el valor es 0 el proyecto no tiene integración y despliegue continuo.
- IC2: El valor medido se encuentra entre el umbral mínimo y el máximo Si es el valor es 0 el proyecto no tiene integración y despliegue continuo.
- IC3: El valor medido se encuentra entre el umbral mínimo y el máximo Si es el valor es 0 el proyecto no tiene integración y despliegue continuo.
- DC1: El valor medido se encuentra entre el umbral mínimo y el máximo Si es el valor es 0 el proyecto aún no tiene ninguna *release*.
- DC2: El valor medido se encuentra entre el umbral mínimo y el máximo.

Junto con las métricas se proporciona un indicador (*Calif.*) del porcentaje de las mismas que se han evaluado como buenas (en verde).

Exportar perfil de métricas

Se puede exportar el perfil de métricas a un fichero *.emmp* para su posterior importación. Para ello, seleccionar la opción correspondiente del menú "*Evaluate projects*" de la figura E.6b: "*Export actual profile*". El diálogo para la exportación es similar al de la figura E.10. Basta con pulsar sobre "*Download*" para poder descargar el fichero que contendrá el perfil de métricas actual.

Bibliografía

[1] Cuadro informativo tipos de retención aplicables (2022).

[2] Ministerio de Empleo y Seguridad Social. Bases y tipos de cotización 2022. https://www.seg-social.es/wps/portal/wss/internet/Trabajadores/CotizacionRecaudacionTrabajadores/36537.

[3] Free Software Foundation. Lista de licencias con comentarios.

[4] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Patrones De Diseño: Elementos De Software Orientado a Objetos Reutilizable*. Addison-Wesley, 1 ed. en es edition.

[5] Carlos López Nozal. Measuring of agile practices in final year project. original-date: 2016-09-27T09:36:06Z.

[6] Raúl Marticorena Sanchez, Yania Crespo, and Carlos López Nozal. Soporte de métricas con independencia del lenguaje para la inferencia de refactorizaciones. [Online; Accedido 03-Octubre-2018].

[7] Scrum Master. *Scrum Manager: Temario Troncal I.* v. 2.61 edition. [Online; Accedido 19-Noviembre-2018].

[8] Jacek Ratzinger. sPACE: Software project assessment in the course of evolution. [Online; Accedido 03-Octubre-2018].

[9] Agencia Tributaria. Tabla de coeficientes de amortización lineal. https://sede.agenciatributaria.gob.es/Sede/impuesto-sobre-sociedades/que-base-imponible-se-determina-sociedades/amortizaciones.html?faqId=42c3904421205710VgnVCM100000dc381e0aRCRD.

[10] Miguel Ángel León Bardavío. Evolution metrics gauge - comparador de métricas de evolución en repositorios software. https://gitlab.com/mlb0029/comparador-de-metricas-de-evolucion-en-repositorios-software.