UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

Evolution Metrics Gauge

Comparador de métricas de evolución en repositorios software
Documentación Técnica

Presentado por Miguel Ángel León Bardavío
en Universidad de Burgos — 16 de septiembre de 2019
Tutor: Carlos López Nozal

Índice general

Índice de figuras

Índice de tablas

Apéndice A

Plan de Proyecto Software

A.1. Introducción

Una de las primeras actividades de un proyecto software es la planificación. En esta fase se estima el tiempo, esfuerzo y dinero que se espera invertir en el proyecto. El objetivo principal de la fase de planificación es estimar si se puede realizar el proyecto con éxito y, en ese caso, tener una guía de referencia para el proceso de desarrollo. Este anexo recoge los documentos generados en este proceso y se divide en dos partes:

Planificación temporal. En esta parte se estima el tiempo y el esfuerzo que se requieren para la realización del proyecto.

Estudio de la viabilidad. Esta parte se estima si el proyecto es viable tanto económica como legalmente, por tanto se dividirá en dos secciones:

- **Viabilidad económica.** Análisis del coste y del beneficio que supondría la realización del proyecto en un entorno laboral real.
- **Viabilidad legal.** Análisis de las leyes que se aplicarían desde el comienzo del proyecto. En un proyecto software tienen especial importancia las licencias y la Ley de Protección de Datos.

A.2. Planificación temporal

La planificación temporal del proyecto ha tomado como referencia el modelo de ciclo de vida **SCRUM** [7]:

- Se ha aplicado un desarrollo incremental y evolutivo.
- Se han realizado iteraciones (*sprints*) de dos semanas. Se finalizaba el sprint con una reunión entre el tutor y el alumno que daba comienzo al siguiente sprint y que constaba de dos partes:
  - Una parte de revisión del sprint en la que se exponía una parte operativa del producto realizada durante el sprint.
  - Otra de planificación del siguiente sprint en la que se determinaba el trabajo y los objetivos a alcanzar durante el siguiente sprint. Esto quedaba reflejado como una pila de tareas que se debían completar durante el sprint y que han sido registradas en el sistema de gestión de incidencias de GitLab.

Sprints

Aún habiendo seguido un proceso incremental, según la naturaleza de los sprints y el orden en el que se han llevado a cabo, en el proceso del desarrollo se pueden diferenciar varias etapas:

- Una primera etapa de **investigación** de las herramientas que se utilizarán durante el proceso y de **configuración** del entorno de desarrollo.
- En la segunda etapa se aprecian tareas de **diseño e implementación** de la parte lógica de la aplicación. Se diseña el framework de conexión a forjas de repositorios, se implementa el framework descrito en *Soporte de Métricas con Independencia del Lenguaje para la Inferencia de Refactorizaciones* [6] para el cálculo de métricas y se diseñan los modelos de datos que serán utilizados por la aplicación.
- Durante la segunda etapa se apreció que se debía mejorar el flujo de trabajo con herramientas de automatización de actividades como pruebas, despliegue, revisión de calidad, etc.
- Etapa de diseño e implementación de la **interfaz gráfica** y **mejora de funcionalidades**.
- Etapa de **documentación** en la que se escribe la memoria y los anexos. También se preparan videotutoriales y manuales de usuario.

A continuación se describe cada una de las etapas del proyecto y sus respectivas actividades. Se acompañará la descripción con capturas de pantalla

tomadas del repositorio del proyecto en GitLab [1] y de *gráficos burndown* de los *milestones* asociados a cada uno de los sprints del ciclo de vida del proyecto.

Investigación y configuración

La etapa de investigación y configuración comenzó a principios de octubre y duró **3 sprints** (6 semanas) y sus decisiones afectaron a todo el proyecto. Se recopiló información sobre trabajos relacionados como *Activiti-Api* [1], *Soporte de Métricas con Independencia del Lenguaje para la Inferencia de Refactorizaciones* [6] y *sPACE: Software Project Assessment in the Course of Evolution* [8], y se estudió los entornos y herramientas que se utilizarían para el desarrollo del proyecto.

En la Fig. A.1 se pueden apreciar los milestones definidos durante esta etapa, su descripción, la fecha de apertura y cierre, y el número de issues asociadas. La mayoría de issues asociados a estos milestones están etiquetadas como '*Research*' (Estudio o investigación) y '*Project Configuration*' (Configuración del proyecto).

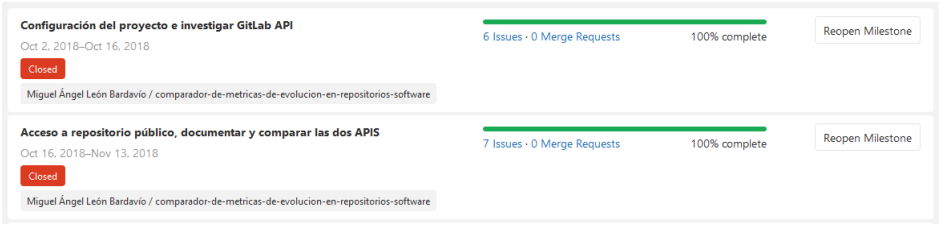


Figura A.1: Milestones asociados a la etapa de investigación y configuración

Del primer milestone no se ha podido sacar un gráfico burndown, debido a problemas con GitLab. Sin embargo, se muestra gráfico burndown del segundo milestone en la Fig. A.2.

[1] Enlace al repositorio del proyecto: https://gitlab.com/mlb0029/comparador-de-metricas-de-evolucion-en-repositorios-software

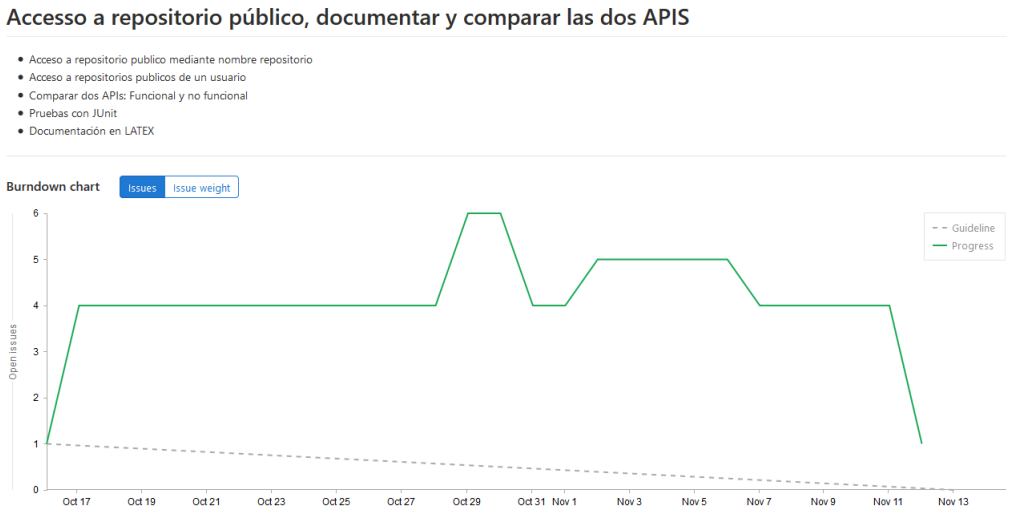


Figura A.2: Gráfico burndown del segundo sprint Acceso a repositorio público, documentar y comparar las dos APIS

Diseño e implementación

Después de esta etapa de investigación, comenzó una etapa de diseño. Las tareas que se definen se relacionan con el diseño de la parte lógica del sistema software a construir. Esta etapa de diseño tuvo duración de **5 sprints** y se dedicó un sprint adicional para la configuración del flujo de trabajo que se explica en el siguiente apartado.

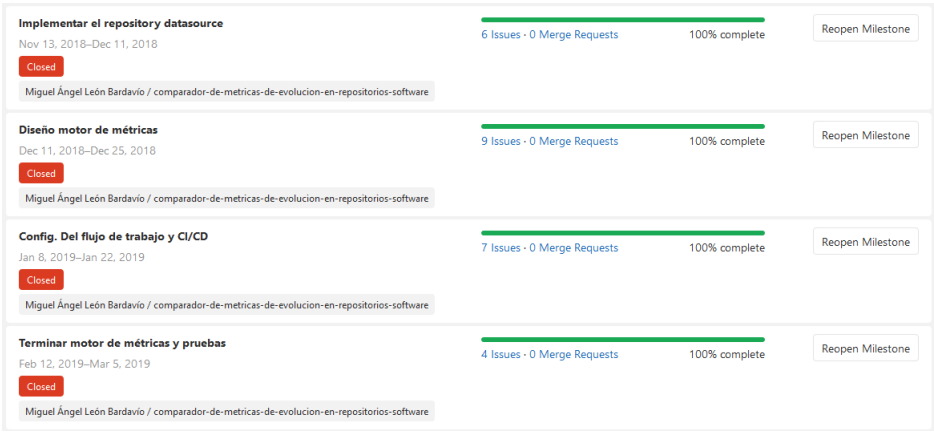


Figura A.3: Sprints de la etapa de diseño e implementación y de la etapa de configuración del flujo de trabajo

En la Fig. A.3 se muestran 3 milestones asociados a esta etapa y otro

asociado a la etapa siguiente, que pausó la etapa de diseño durante un sprint. Los milestones asociados a la etapa de diseño son:

- ***Implementar el repository datasource*** (2 sprints). Consta de una tarea (#22) etiquetada como '*Design*' para implementar el framework de conexión a GitLab y obtener la información de los repositorios. El resto de tareas están mal definidas, se han etiquetado como '*Future*' (Para hacer en el futuro) o '*Badly Defined*' (Mal definidas) y se definieron en los milestones posteriores.
- ***Diseño motor de métricas*** (1 sprint). Todas las tareas en este milestone se han definido con label '*Design*' (Diseño). Es destacable el gráfico burndown de este milestone, como se puede observar en el segundo gráfico de la Fig. A.4.
- ***Terminar motor de métricas y pruebas*** (2 sprints). Consta de cuatro issues: 3 de aumento de características (etiqueta '*Feature*') y una de pruebas (etiqueta '*Test*').

En la Fig. A.4 se muestran los gráficos burndown de los dos primeros milestones de esta etapa. No se puedo obtener el gráfico del tercer milestone.

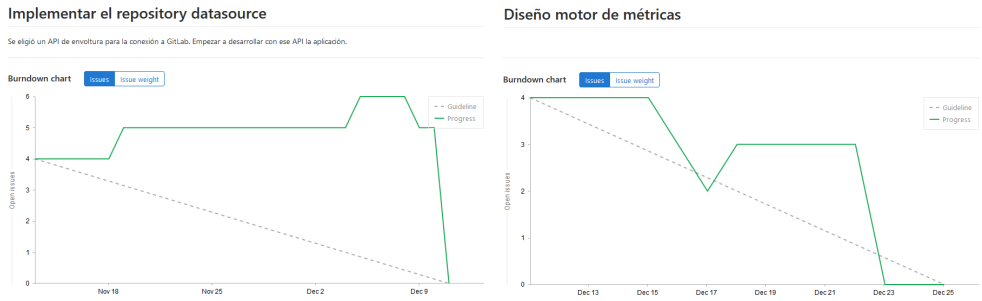


Figura A.4: Gráficos burndown de

El primero muestra la evolución del primer milestone (*Implementar el repository datasource*). Se aprecia la definición incorrecta de issues a mitad del sprint, que se cerraron al final del milestone marcándolas como '*Future*' o '*Badly Defined*'.

En el segundo gráfico se muestra la evolución del segundo milestone (*Diseño motor de métricas*). La evolución fue bastante buena, completando las tareas antes de tiempo. Se observa que se definió una issue (#30) en mitad del sprint. Dicha tarea se marcó con la etiqueta 'Question' y fue para

preguntar dudas al tutor. Es destacable la evolución de esa tarea, siendo posible apreciarla a partir de los comentarios realizados en la misma.

Configuración del flujo de trabajo

Dentro de la fase de diseño se decidió dedicar **un sprint** a configurar el proyecto para facilitar el flujo de trabajo y las reuniones de revisión/planificación del sprint. Las principales tareas para mejorar el flujo de trabajo fueron:

- Configurar la gestión del proyecto con Maven
- Configurar los procesos de integración y despliegue continuo con GitLab (CI/CD)
- Realizar pruebas unitarias con JUnit y automatizar su ejecución gracias a Maven y los *pipelines* (CI/CD) de GitLab .
- Configurar revisiones automáticas de calidad y de cobertura de las pruebas gracias a Codacy, Jacoco y GitLab.
- Configurar un entorno en Heroku donde poder desplegar la aplicación y así poder ser revisada por el tutor fácilmente.
- Configurar badges [2] para representar los el estado de la calidad de código, cobertura, despliegue y los trabajos de CI/CD.

En la Fig. A.3 se muestra este sprint en el milestone llamado '*Config. Del flujo de trabajo y CI/CD*'. Consta de 7 tareas de configuración, pregunta e investigación junto con un par de tareas de test. No se ha obtenido el gráfico burndown de este milestone.

Implementación de la interfaz gráfica y mejora de las características

Es la etapa más extensa, duró 14 semanas (**7 sprints**) y se destacan las etiquetas 'GUI' y 'Feature'.

La etapa comienza con un estudio de la herramienta Vaadin para crear la interfaz gráfica y la viabilidad de su uso en el proyecto. Para ello, se realizaron varias pruebas de integración con el entorno de desarrollo (Versión de Java,

[2]Son placas que aportan información rápida sobre el estado del proyecto en ciertos aspectos como la cobertura, la calidad de código o el estado del proceso de CI/CD

Eclipse, Maven, etc), se realizó un primer prototipo sencillo y se comprobó que se podría hacer una interfaz gráfica con la licencia gratuita. Una vez terminado con éxito estas pruebas de viabilidad, comenzó el desarrollo de la interfaz gráfica del proyecto.

La interfaz gráfica da un aspecto más visible de lo que el proyecto podría llegar a ser. Los requisitos funcionales evolucionaban conforme evolucionaba la interfaz. Por ejemplo, se decidió agregar varias formas de añadir un repositorio y añadir más formas de conectarse a GitLab.

Se han definido 9 milestones, cometiendo el error de definir varios milestones a la vez para separar la naturaleza de las issues. En la Fig. A.5 se aprecia la descripción, tiempo y número de issues asociadas a cada milestone.

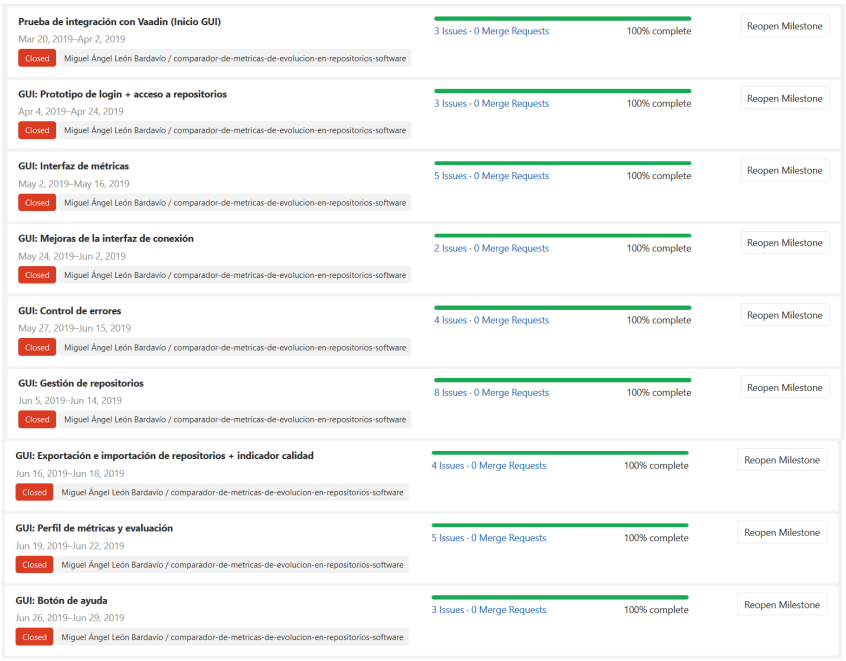


Figura A.5: Milestones de la etapa de implementación de la interfaz gráfica y mejora de las características

Algunos de los gráficos burndown más destacables de esta etapa se muestran en la Fig. A.6

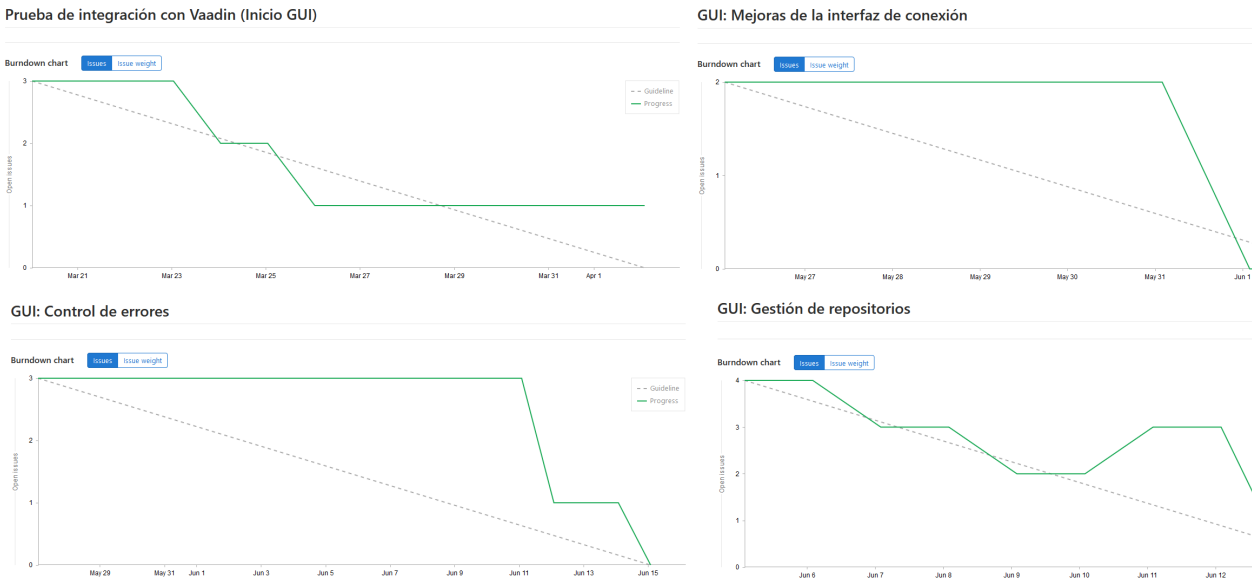


Figura A.6: Milestones de la etapa de implementación de la interfaz gráfica y mejora de las características

Documentación

Esta etapa comenzó al terminar el desarrollo de la aplicación. En esta se realiza la memoria y los anexos del proyecto, así como videotutoriales u otros manuales de usuario. Esta fase tiene una duración aproximada de **5 sprints**. Se tiene la aproximación debido a que se fue trabajando sin crear su milestone correspondiente. Actualmente se ha creado uno que define las actividades a realizar, pero no describe el tiempo invertido.

A.3. Estudio de viabilidad

Viabilidad económica

En esta sección se realiza un análisis coste-beneficio del proyecto de haberse realizado en un entorno laboral real.

Costes

Costes de personal

El proyecto ha sido realizado por un empleado a media jornada (20 horas semanales) durante 11 meses (octubre 2018 — septiembre 2019). Se

considera:

- Salario neto mensual: 600 €
- Una retención del IRPF por actividad profesional con carácter general del 15 % [9]
- Una retribución a la Seguridad Social de 29,9 % [2]: Contingencias comunes (23,6 %) + Desempleo de tipo general (5,5 %) + FOGASA (0,20 %) + Formación profesional (0,6 %)

Realizando cálculos, se obtiene:

Concepto	**Coste**
Salario mensual neto	600 €
Retención IRPF (15 %)	163,34 €
Seguridad Social (29,9 %)	325,59 €
Salario mensual bruto	1088,93 €
Total 11 meses	11.978,23 €

Tabla A.1: Costes de personal

Costes de hardware

Teniendo en cuenta que la amortización de equipos para procesos de información es de 8 años [10], que el equipo portátil en el que se ha trabajado (único gasto hardware) ha tenido un coste de 648,76 € y que ha sido utilizado durante 11 meses:

Concepto	**Coste**	**Coste amortizado**
Ordenador portátil	648.76 €	74.30 €
Total	648.76 €	74.30 €

Tabla A.2: Costes de hardware

Costes de software

El único software de pago que se ha utilizado es *Windows 10 Home* con coste de 145 €. Teniendo en cuenta el coste del sistema operativo, que la amortización para Sistemas y programas informáticos es de 6 años [10] y se ha utilizado durante 11 meses:

Concepto	Coste	Coste amortizado
Windows 10 Home	145 €	22 €
Total	145 €	22 €

Tabla A.3: Costes de software

Otros costes

Concepto	Coste
Memoria impresa y cartel	50 €
Alquiler de oficina	200 €
Internet	165 €
Material de oficina	25 €
Total	440 €

Tabla A.4: Costes varios.

Coste total

Concepto	Coste
Personal	11.978,23 €
Hardware	74.30 €
Software	22 €
Otros	440 €
Total	12.514,53 €

Tabla A.5: Costes totales.

Beneficios

Por el momento, se distribuye la aplicación de forma gratuita y sin publicidad, por lo que no se obtienen beneficios. Par adquirir beneficios se podría:

- Introducir publicidad
- Desplegar la aplicación web en un servidor de producción y cobrar por el servicio.

Viabilidad legal

En esta sección se realiza un estudio de las licencias del software utilizado para el desarrollo de la aplicación. En la Fig. A.7 con las herramientas utilizadas y las licencias que tienen. La información se ha obtenido del repositorio de Maven [3] al recoger las dependencias del proyecto en el fichero pom.xml:

Grupo	Artefacto	URL	Licencias
com.gavinmogan	codacy-maven-plugin	https://mvnrepository.com/artifact/com.gavinmogan/codacy-maven-plugin/1.2.0	MIT
com.heroku.sdk	heroku-maven-plugin	https://mvnrepository.com/artifact/com.heroku.sdk/heroku-maven-plugin/2.0.9	MIT
com.vaadin	flow-server-production-mode	https://mvnrepository.com/artifact/com.vaadin/flow-server-production-mode	Apache 2.0
com.vaadin	vaadin-bom	https://mvnrepository.com/artifact/com.vaadin/vaadin-bom/13.0.0	Apache 2.0
com.vaadin	vaadin-core	https://mvnrepository.com/artifact/com.vaadin/vaadin-core/13.0.0	Apache 2.0
com.vaadin	vaadin-maven-plugin	https://mvnrepository.com/artifact/com.vaadin/vaadin-maven-plugin/13.0.0	Apache 2.0
log4j-core	log4j-core	https://mvnrepository.com/artifact/org.apache.logging.log4j/log4j-core/2.11.2	Apache 2.0
org.apache.commons	commons-math3	https://mvnrepository.com/artifact/org.apache.commons/commons-math3/3.6.1	Apache 2.0
org.apache.logging.log4j	log4j-api	https://mvnrepository.com/artifact/org.apache.logging.log4j/log4j-api/2.11.2	Apache 2.0
org.apache.logging.log4j	log4j-slf4j-impl	https://mvnrepository.com/artifact/org.apache.logging.log4j/log4j-slf4j-impl/2.11.2	Apache 2.0
org.apache.maven.plugins	maven-compiler-plugin	https://mvnrepository.com/artifact/org.apache.maven.plugins/maven-compiler-plugin/3.8.0	Apache 2.0
org.apache.maven.plugins	maven-failsafe-plugin	https://mvnrepository.com/artifact/org.apache.maven.plugins/maven-failsafe-plugin/2.22.1	Apache 2.0
org.apache.maven.plugins	maven-surefire-plugin	https://mvnrepository.com/artifact/org.apache.maven.plugins/maven-surefire-plugin/2.22.1	Apache 2.0
org.apache.maven.plugins	maven-war-plugin	https://mvnrepository.com/artifact/org.apache.maven.plugins/maven-war-plugin/3.2.2	Apache 2.0
org.claspina	confirm-dialog	https://mvnrepository.com/artifact/org.claspina/confirm-dialog/1.0.0	Apache 2.0
org.gitlab4j	gitlab4j-api	https://mvnrepository.com/artifact/org.gitlab4j/gitlab4j-api/4.9.14	MIT
org.jacoco	jacoco-maven-plugin	https://mvnrepository.com/artifact/org.jacoco/jacoco-maven-plugin/0.8.3	EPL 1.0
org.junit.jupiter	junit-jupiter-api	https://mvnrepository.com/artifact/org.junit.jupiter/junit-jupiter-api/5.3.2	EPL 2.0
org.junit.jupiter	junit-jupiter-engine	https://mvnrepository.com/artifact/org.junit.jupiter/junit-jupiter-engine/5.3.2	EPL 2.0
org.junit.jupiter	junit-jupiter-params	https://mvnrepository.com/artifact/org.junit.jupiter/junit-jupiter-params/5.3.2	EPL 2.0
org.slf4j	slf4j-api	https://mvnrepository.com/artifact/org.slf4j/slf4j-api/1.7.28	MIT

Figura A.7: 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 Fundación Eclipse informa que la versión 1.0 está en desuso y que los proyectos deben migrar a la versión 2.0, por lo tanto no nos restringe la versión 1.0 del plugin *jacoco-maven-plugin*, sino la versión 2.0.

La licencia más restrictiva es la *Eclipse Public License* que poseen las librerías que se utilizan para pruebas.

Nuestro proyecto puede tener la licencia *GNU General Public License v3.0*, que permite el uso comercial, modificación, distribución, uso privado y el uso de patentes. Aunque no sea compatible, la nueva versión de *EPL* permite utilizar la versión 2 de la *GPL* de *GNU* y posteriores como licencia secundaria en una parte especifica del código, eso garantiza la compatibilidad de ese

[3] https://mvnrepository.com/

código con esas versiones de la GPL [3]. Por lo tanto, se puede licenciar el código bajo *GPL v3.0* y el código de pruebas bajo licencia *EPL* o compatibles como *Apache v2.0*.

Apéndice B

Especificación de Requisitos

B.1. Introducción

Este anexo tiene como objetivo analizar y documentar las necesidades funcionales y no funcionales que deberán ser soportadas por el software desarrollado.

B.2. Objetivos generales

El objetivo general de este TFG es diseñar una aplicación Web en Java que permita obtener un conjunto de métricas de evolución del proceso software a partir de repositorios de GitLab, para permitir comparar los distintos procesos de desarrollo software de cada repositorio. La aplicación se probará con datos reales para comparar los repositorios de Trabajos Fin de Grado del Grado de Ingeniería Informática presentados en GitLab. Con más detalle:

- Se obtendrán medidas de métricas de evolución de uno o varios proyectos alojados en repositorios de GitLab.
- 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:
  - 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

- El objetivo de obtener las métricas es poder evaluar el estado de un proyecto comparándolo con otros proyectos de la misma naturaleza. 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 calcularán dinámicamente y se almacenarán en perfiles de configuración de métricas.
- Se dará la posibilidad de almacenar de manera persistente estos perfiles de métricas para permitir comparaciones futuras. Un ejemplo de utilidad es guardar los valores umbrales de repositorios por lenguaje de programación, o en el caso de repositorios de TFG de la UBU por curso académico.
- 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.

B.3. Objetivos técnicos

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

- Diseñar la aplicación de manera que se puedan extender con nuevas métricas con el menor coste de mantenimiento posible. Para ello, se aplicará un diseño basado en frameworks y en patrones de diseño [4].
- El diseño de la aplicación debe facilitar la extensión a otras plataformas de desarrollo colaborativo como GitHub o Bitbucket.
- Aplicar el *frameworks 'modelo-vista-controlador'* para separar la lógica de la aplicación y la interfaz de usuario.
- Crear una batería de pruebas automáticas con cobertura por encima del 90 % 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.
- Utilizar un sistema de integración y despliegue continuo.

- Diseñar una correcta gestión de errores definiendo excepciones de biblioteca y registrando eventos de error e información en ficheros de *log*.
- Aplicar nuevas estructuras del lenguaje Java para el desarrollo, como son expresiones lambda.
- Utilizar sistemas que aseguren la calidad continua del código que permitan evaluar la deuda técnica del proyecto.
- 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 y el desarrollador. Además de poder ser utilizado por un usuario, la aplicación deberá estar preparada para que en un futuro se pueda extender en cuanto a número de métricas y forjas de repositorios.

B.5. Catalogo de requisitos

Este apartado enumera los diferentes requisitos que el sistema software deberá satisfacer. Se divide en dos apartados. El primero detalla los servicios que prestará el sistema al usuario final o a otros sistemas; el segundo detalla funciones más técnicas de cómo se desarrollará el proceso software y otras propiedades del sistema como eficiencia o mantenibilidad.

Requisitos funcionales

- **RF-1** Establecer conexión con GitLab: El usuario debe poder establecer distintos tipos de conexión a GitLab.
    - **RF-1.1** 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-1.2** 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-1.3** El usuario podrá establecer una conexión pública a GitLab para poder obtener los repositorios públicos.
- **RF-1.4** El usuario podrá utilizar la aplicación sin establecer una conexión. Aunque no tenga acceso a los repositorios de GitLab.
- **RF-1.5** El usuario deberá elegir un tipo de conexión al entrar por primera vez a la aplicación.
- **RF-1.6** La aplicación mostrará al usuario en todo momento la conexión que está utilizando
- **RF-1.7** El usuario podrá cambiar de conexión teniendo en cuenta que solo puede haber un tipo de conexión activo en un instante dado

- **RF-2** Gestión de proyectos. El usuario podrá calcular las métricas de proyectos 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-2.1** El usuario podrá añadir un proyecto siempre que tenga una conexión a GitLab (con sesión o pública)
    - **RF-2.1.1** 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-2.1.2** 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-2.1.3** El usuario podrá añadir un proyecto a partir de su URL, siempre que tenga acceso desde la conexión establecida
  - **RF-2.2** El usuario no podrá añadir un proyecto que ya esté en la tabla
  - **RF-2.3** 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-2.4** El usuario podrá eliminar un proyecto de la tabla
  - **RF-2.5** El usuario podrá volver a obtener las métricas de un repositorio que ya haya añadido, siempre que tenga acceso desde la conexión establecida
  - **RF-2.6** El usuario podrá exportar los proyectos y sus métricas a un fichero con formato '.emr'

- **RF-2.7** El usuario podrá exportar los resultados de las métricas de todos los proyectos en un fichero con formato CSV
- **RF-2.8** El usuario podrá importar y añadir repositorios a la tabla desde un fichero con formato '.emr', respetando el requisito RF2.2 de no añadir un repositorio ya existente
- **RF-2.9** El usuario podrá importar repositorios a la tabla, sobrescribiendo los de la tabla.
- **RF-2.10** Se podrán filtrar los proyectos por su nombre
- **RF-2.11** Se puede ordenar los repositorios por nombre, fecha de medición y por cualquiera de las métricas
- **RF-2.12** Al ordenar por métricas habrá que tener en cuenta que habrá medidas que no se habrán calculado por falta de datos de GitLab

▪ RF-3 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*'

- **RF-3.1** Se podrá cargar un perfil de métricas que contenga valores umbrales mínimo y máximo y se utilizarán para evaluarlas
- **RF-3.1** El usuario podrá cargar un perfil por defecto creado a partir de un conjunto de datos [1] de un estudio empírico de las métricas de evolución del software en trabajos finales de grado[5]
- **RF-3.2** El usuario podrá cargar un perfil creado a partir de los repositorios que existan en la tabla
- **RF-3.3** El usuario podrá exportar el perfil de métricas que tenga cargado a un fichero con formato emmp
- **RF-3.4** El usuario podrá importar un perfil de métricas que haya guardado anteriormente para evaluar los repositorios que tenga en la tabla
- **RF-3.5** En un instante, solo puede haber un perfil de métricas cargado
- **RF-3.6** Por defecto se cargará el perfil por defecto

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

Definición de las métricas

En el requisito RF-2 se entiende que se debe 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*
- **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*

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

Requisitos no funcionales

- **RNF-1** Se debe proporcionar un diseño extensible a otras forjas de repositorios como GitHub o Bitbucket
- **RNF-2** Se debe proporcionar un 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 aplicar el *frameworks 'modelo-vista-controlador'* para separar la lógica de la aplicación y la interfaz de usuario
- **RNF-6** Se debe crear una batería de pruebas automáticas con cobertura por encima del 90 % en los subsistemas de lógica de la aplicación.
- **RNF-7** 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.
- **RNF-8** Se debe diseñar una correcta gestión de errores definiendo excepciones de biblioteca y registrando eventos de error e información en ficheros de *log*.
- **RNF-9** Se deben utilizar sistemas que aseguren la calidad continua del código que permitan evaluar la deuda técnica del proyecto.
- **RNF-9** Se ha de probar la aplicación con ejemplos reales

Apéndice C

Especificación de diseño

C.1. Introducción

Este apartado recoge el porqué de las decisiones finales que se han tomado acerca del diseño del software desarrollado, dividido en el diseño de datos, diseño arquitectónico y diseño de la interfaz de usuario.

C.2. Diseño de datos

El diseño de los datos es el diseño que tienen las entidades. Al no disponer de una base de datos, las entidades que se gestionan son clases Java.

Las clases más importantes a la hora de definir los datos que trabaja la aplicación están definidas en el paquete datamodel, estás son:

- Repository. Sirve como definición de los datos que se obtienen de un repositorio de GitLab. 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 se espera que se calculen las métricas. Estas serán almacenadas aquí.
  - Evaluación del proyecto (*projectEvaluation*). Sirve como indicador del estado del proceso del proyecto, se espera almacenar el

porcentaje de métricas evaluadas como 'buenas' para el proyecto cada vez que se añadan unos nuevos resultados de métricas.

- Se ha definido la igualdad de los repositorios por su URL, ID y nombre.

- RepositoryInternalMetrics. Sirve para separar lo que define a un proyecto de los datos que se necesitan para calcular las métricas. Contiene: 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, y el tiempo de vida del proyecto. Se define el método *equals* a partir de la fecha de medición.

- User. Sirve para almacenar información que se obtiene de GitLab acerca del usuario que ha iniciado sesión: el ID de usuario, la URL del avatar, el e-mail, el nombre y el nombre de usuario.

C.3. Diseño arquitectónico

En este apartado se describen las clases y la estructura de los paquetes de los que consta el proyecto software.

Paquete repositorydatasource

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

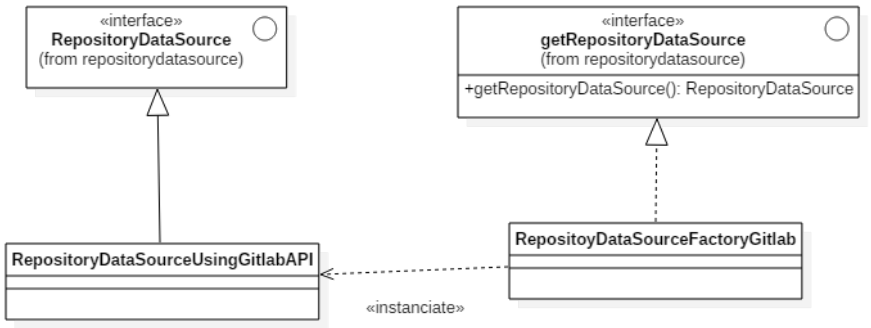


Figura C.1: Paquete reposutorydatasource

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.

Paquete metricsengine

El paquete 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; y 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.
  La intencionalidad de esto 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 han añadido los métodos *evaluate* y *getEvaluationFunction* en la interfaz *IMetric*, ver Fig. C.3.
  Esto permitirá interpretar y evaluar los valores medidos sobre los valores límite de la métrica o configuración de métrica. 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.

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

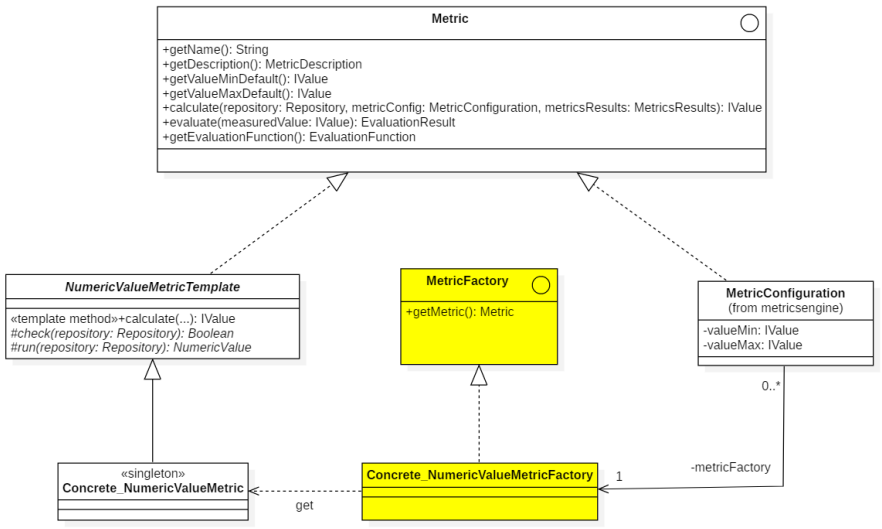


Figura C.2: Patrones “singleton” y “método fábrica” sobre el framework de medición

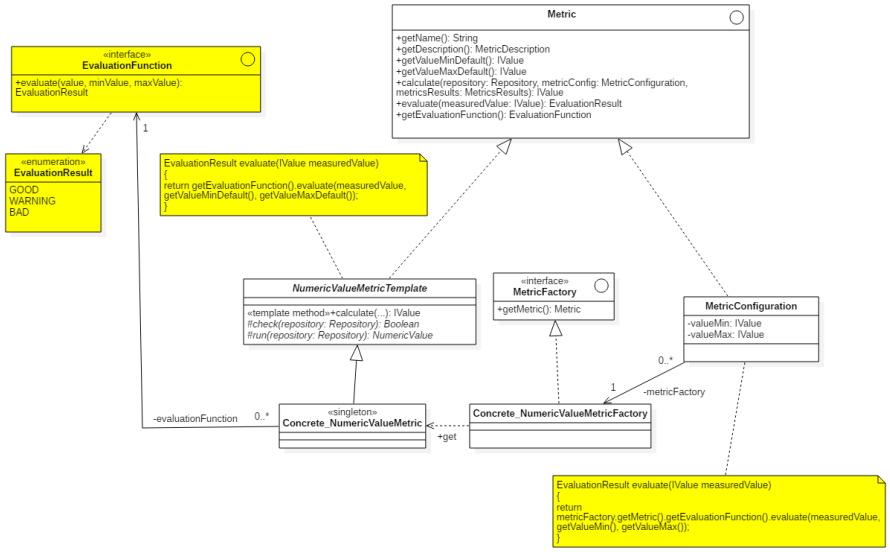


Figura C.3: Añadido al framework de medición la evaluación de métricas

Paquete gui

En este paquete se ha definido toda la interfaz gráfica.

Paquete exceptions

En este paquete se han definido las excepciones de la aplicación, además se han trabajan 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:

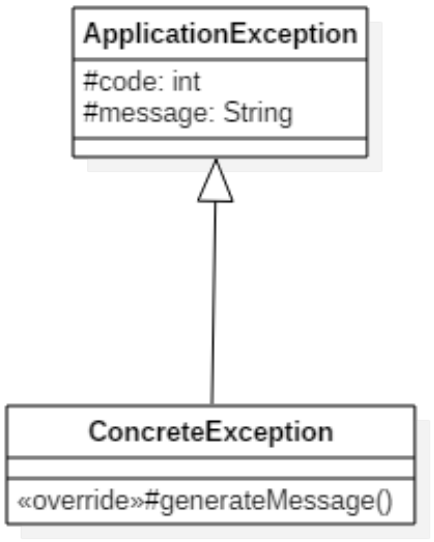


Figura C.4: Paquete exceptions

Paquete datamodel

El paquete datamodel contiene las tres clases que se han descrito anteriormente en la sección C.2.

Paquete app

Contiene las clases que integran todos los módulos y sirve de conexión entre la interfaz de usuario y la parte lógica de la aplicación.

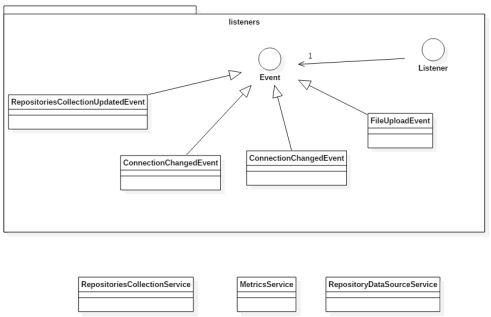


Figura C.5: Paquete app

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.

También cuenta con un paquete para el patrón observador. En este se definen la interfaz *Listener* y la interfaz *Event*, y se implementan algunos 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

Este documento detalla asuntos técnicos de programación.

D.2. Estructura de directorios

El código fuente presenta la siguiente estructura:

- /.gitignore. Contiene los ficheros y directorios que el repositorio git no tendrá en cuenta
- /.gitlab-ci.yml. Contiene las etapas y trabajos que se han definido para que se ejecuten en una máquina virtual proporcionada por GitLab (*runner*) tras hacer un commit. Permite la integración y el despliegue continuo.
- /README.md. Fichero con información relevante sobre el proyecto.
- /pom.xml. Fichero de configuración del proyecto Maven.
- /system.properties. Fichero con propiedades del proyecto. Ha sido necesario su uso para el despliegue en Heroku.
- /MemoriaProyecto. Memoria del proyecto según la plantilla definida en https://github.com/ubutfgm/plantillaLatex.
- /releases. Carpeta para publicar los ficheros WAR. Las releases enlazarán a estos ficheros.
- /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. Generado por Vaadin.
- /src/main/webapp/frontend. Ficheros *.css* utilizados 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 log.
- /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 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. Define el motor de métricas.
- /src/main/java/metricsengine/numeric_value_metrics. Métricas definidas por el programador 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 una forja de repositorios como GitLab.

D.3. Manual del programador

Se explica en este apartado algunas bases para entender como continuar la programación de la aplicación y los puntos de extensión que se han definido. Cada apartado de esta sección se centra en cada uno de los módulos que contiene la aplicación.

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

Framework de conexión

El framework de conexión a una plataforma de desarrollo colaborativo 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 para conectar el API de la forja de repositorios con el resto de la aplicación. Una solo es una fábrica que pretende instanciar un *RepositoryDataSource*. Esta última es la que contiene las funciones para establecer una conexión y obtener los datos de los repositorios. El único cambio que habría que hacer despues de implementar estas dos interfaces es cambiar la llamada a la fábrica.

Motor de métricas

El motor de métricas se ha desarrollado 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.

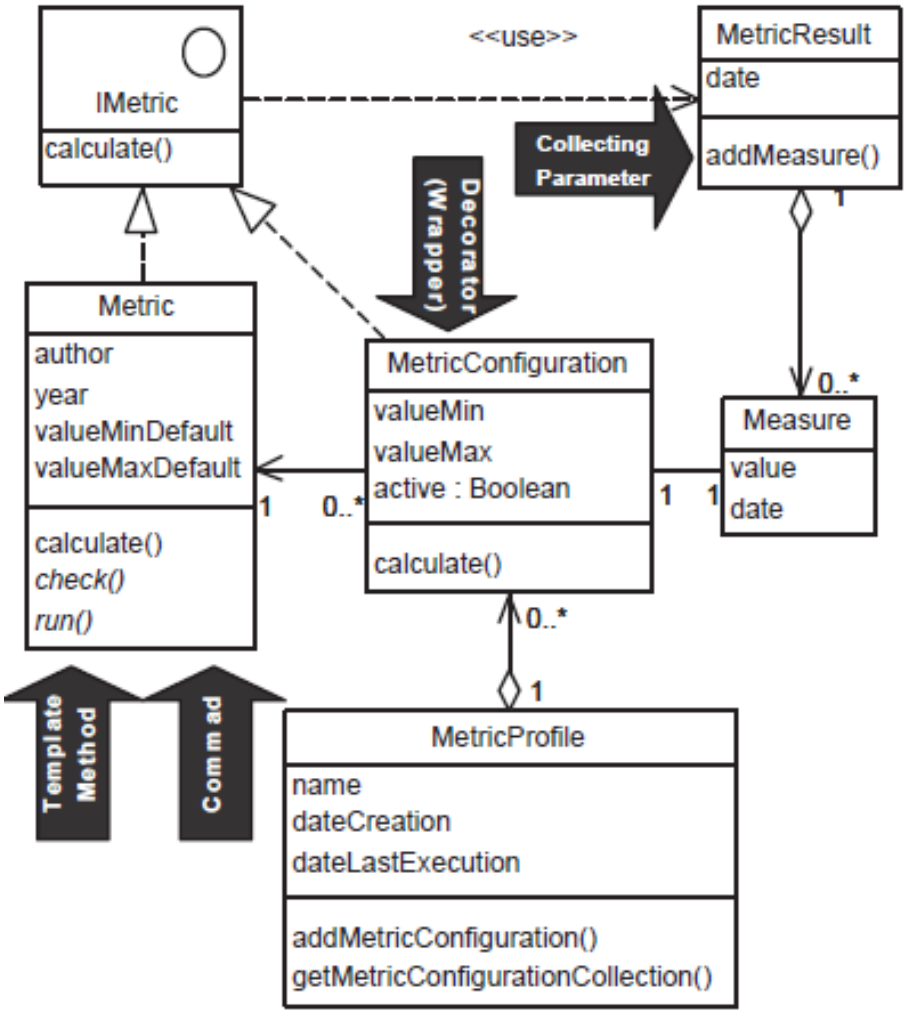


Figura D.1: Diagrama del Framework para el cálculo de métricas con perfiles.

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 descargado desde GitLab.

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.

Apéndice E

Documentación de usuario

E.1. Introducción

Este documento detalla cómo un usuario, puede utilizar la aplicación una vez desplegada en un servidor.

E.2. Requisitos de usuarios

Los requisitos para poder utilizar la aplicación son:

- Tener la aplicación desplegada en algún servidor.
- Disponer de conexión al servidor y tener instalado un navegador web con el que poder acceder a la aplicación. Se recomienda:
  - Google Chrome Versión 75.0.3770.100 o superior
  - Firefox Quantum Versión 67.0.4 o superior
  - IE11 Versión 11.829.17134.0 o superior
  - Opera Versión:62.0.3331.18 o superior

E.3. Instalación

Al tratarse de una aplicación web, esta no requiere instalación. Solo es necesario desplegar en un servidor la aplicación (el fichero WAR `target/evolution-metrics-gauge-[version].war` que se genera al al ejecutar `mvn install` o que se pública en las *releases* [1] del proyecto en GitLab).

[1]Releases del proyecto: https://gitlab.com/mlb0029/comparador-de-metricas-de-evolucion-en-repositorios-software/-/releases

E.4. Manual del usuario

La aplicación toma como entrada un conjunto de repositorios públicos o privados de GitLab y calcula métricas de evolución que permiten comparar los proyectos gracias al cálculo estadístico de cuartiles.

Conceptos

Para utilizar la aplicación es importante entender 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.

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

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

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.

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.

Las métricas que se gestionan en la aplicación

Además de los conceptos se debe comprender las métricas que se calculan en la aplicación y saber interpretarlas. La métricas proceden de la Master Tesis titulada *sPACE: Software Project Assessment in the Course of Evolution* [8].

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*

Arranque de la aplicación: Establecer una conexión a GitLab

La primera vez que se utiliza la aplicación se mostrará un diálogo que le pedirá elegir un tipo de conexión, tal y como se muestra en la figura E.2.

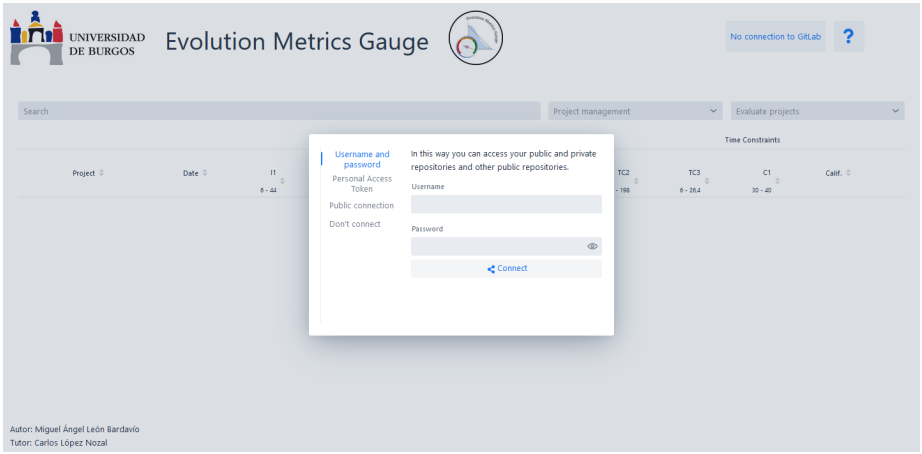
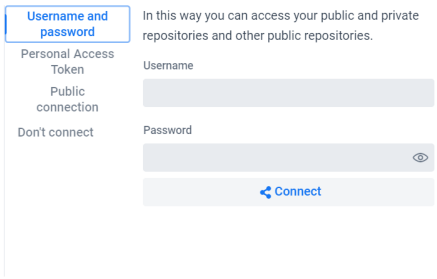
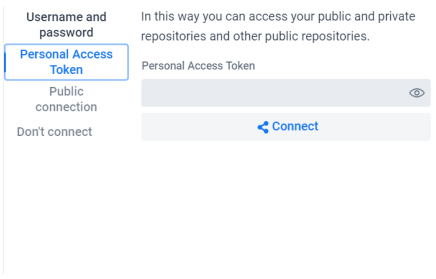


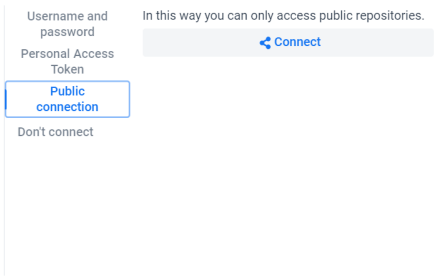
Figura E.1: Diálogo de conexión



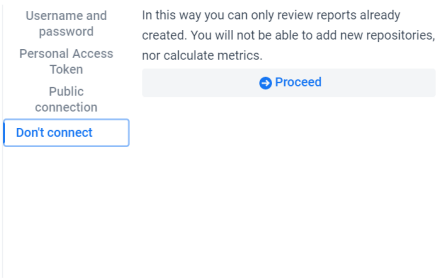
(a) Establecer conexión iniciando sesión mediante usuario y contraseña



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



(c) Establecer una conexión publica



(d) No establecer conexión a GitLab

Figura E.2: Distintas formas de establecer una conexión

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:

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.2a.

Iniciar sesión en GitLab mediante *Personal Access Token*. Se establece una conexión a GitLab 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.2b.
Para generar un *Personal Access Token* desde GitLab hay que iniciar sesión desde la web y entrar en la configuración del usuario. En el apar-

tado de *Access Token* se debe dar un nombre, opcionalmente una fecha de expiración y los permisos. Para utilizar la aplicación se necesitan estos permisos: *api*, *read_user*, *read_repository*, *read_registry*. Una vez finalizado, pulsar sobre el botón "*Create personal access token*", copiar el token y utilizarlo. Una vez se salga de la ventana en la que se muestra el token, no volverá a aparecer, por lo que se recomienda copiarlo en algún fichero bien protegido. Ver figura E.3.

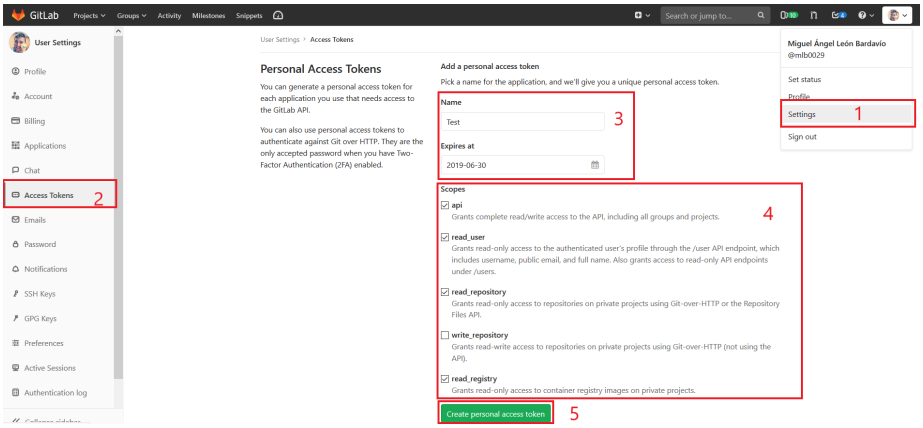


Figura E.3: Crear un *Personal Access Token* desde GitLab

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.2c.

No utilizar ninguna conexión. No se realizará ninguna conexión a GitLab, de esta forma se impide al usuario añadir nuevos repositorios desde GitLab ni volver a calcular métricas de los proyectos existentes. Sin embargo se permite gestionar los proyectos existentes e importar los proyectos 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.2d.

Página principal

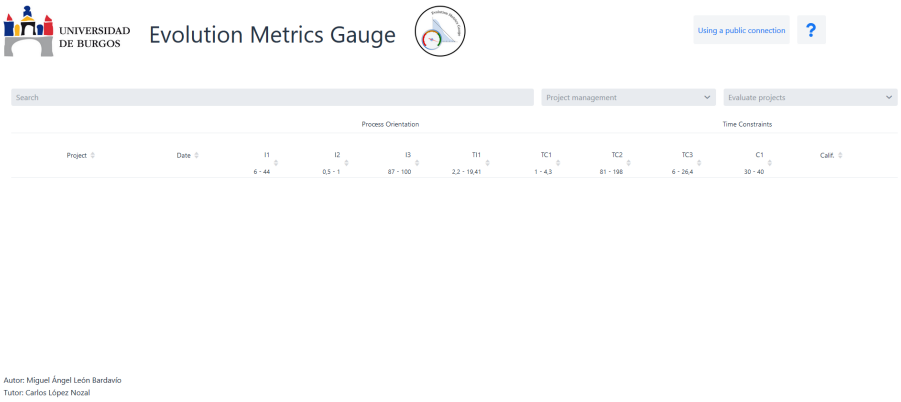


Figura E.4: Página principal

Una vez elegida por primera vez el tipo de conexión deseado se accede a la página principal, como se observa en la figura E.4.

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 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.5a si existe una conexión y el diálogo de la figura E.5b si no existe conexión. Al pulsar sobre "*Connect*" o sobre "*Close connection*", respectivamente, se abrirá el diálogo de conexión de las figuras E.2 y E.2.

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://gitlab.com/mlb0029/`

(a) Cerrar conexión

(b) Establecer conexión

Figura E.5: Modificar tipo de conexión

comparador-de-metricas-de-evolucion-en-repositorios-software/wikis/home.

Listado de proyectos

En el cuerpo de la página principal se pueden gestionar los proyectos. Consta de una barra de búsqueda, dos menús, y una tabla que visualiza las métricas de los proyectos que se añadan.

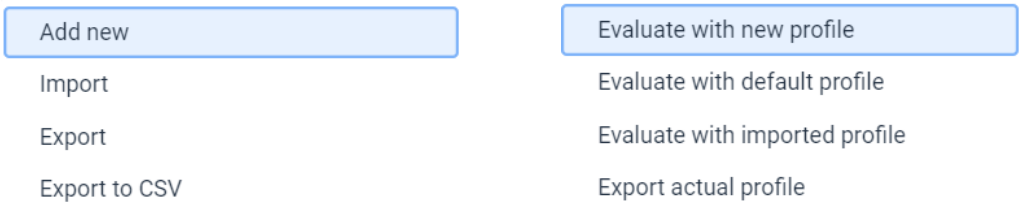
En el **cuadro de búsqueda** se filtrarán los repositorios por su nombre según se vaya escribiendo.

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

- ***Add new.*** Permite añadir uno o varios proyectos (disponible unicamente cuando hay una conexión a GitLab).
- ***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.

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

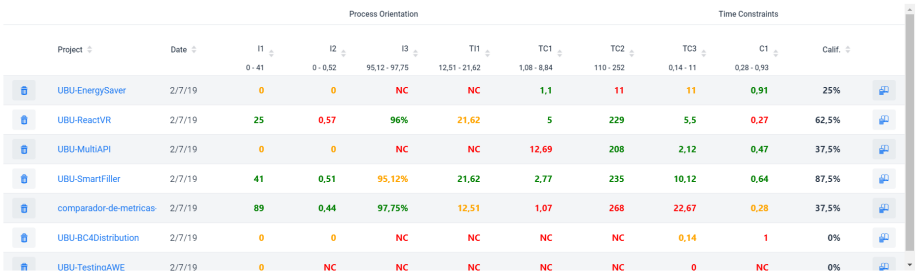
- ***Evaluate with new profile.*** Permite evaluar los proyectos calculando los valores umbrales de cada métrica a partir de los repositorios actuales.
- ***Evaluate with default profile.*** Permite evaluar los proyectos con un perfil por defecto creado a a partir de un conjunto de datos[2] de un estudio 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.



	Project	Date	Process Orientation						Time Constraints			
			I1	I2	I3	TI1	TC1	TC2	TC3	C1	Calif.	
			0 - 41	0 - 0,52	95,12 - 97,75	12,51 - 21,62	1,08 - 8,84	110 - 252	0,14 - 11	0,28 - 0,93		
	UBU-EnergySaver	2/7/19	0	0	NC	NC	1,1	11	11	0,91	25%	
	UBU-ReactVR	2/7/19	25	0,57	96%	21,62	5	229	5,5	0,27	62,5%	
	UBU-MultiAPI	2/7/19	0	0	NC	NC	12,69	208	2,12	0,47	37,5%	
	UBU-SmartFiller	2/7/19	41	0,51	95,12%	21,62	2,77	235	10,12	0,64	87,5%	
	comparador-de-metricas	2/7/19	89	0,44	97,75%	12,51	1,07	268	22,67	0,28	37,5%	
	UBU-BC4Distribution	2/7/19	0	0	NC	NC	NC	NC	0,14	1	0%	
	UBU-TestingAWE	2/7/19	0	NC	NC	NC	NC	NC	0	NC	0%	

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

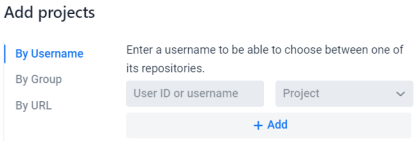
[2] https://github.com/clopezno/clopezno.github.io/blob/master/agile_practices_experiment/DataSet_EvolutionSoftwareMetrics_FYP.csv

La tabla presenta las siguientes columnas:

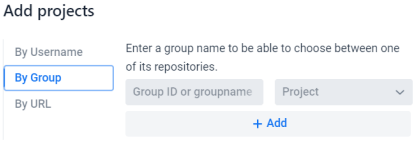
- **Botón de eliminar.** Permite eliminar de la tabla el proyecto seleccionado.
- ***Project.*** Nombre del proyecto con enlace al repositorio de GitLab. Si el nombre es demasiado largo y se corta, se puede utilizar el tooltip que aparece al pasar el ratón por encima del nombre del proyecto. Puede que 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*) y Constantes de tiempo (*Time Constraints*).
  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

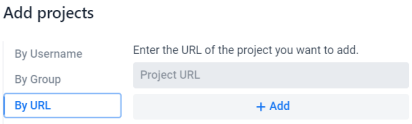
Para añadir un nuevo proyecto, el tipo de conexión deberá ser distinto de "Sin conexión" (*No connection to GitLab*), es decir que debe haber una conexión. Seleccionar la opción "*Add new*" del menú "*Project management*", ver figura E.6a. Se abrirá un diálogo como el de la figura E.8. Para cancelar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



(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

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. **OJO**: NO se pide confirmación de eliminación.

Recalcular métricas de un proyecto

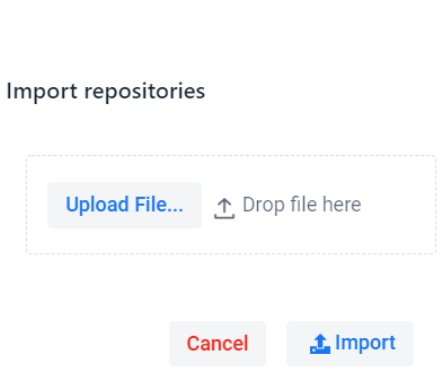
Para recalcular las métricas de un proyecto debe haber una conexión a GitLab y esta debe tener permisos sobre el proyecto deseado (si es un proyecto privado, debe haber iniciado sesión una persona con permisos de lectura sobre el proyecto). Cumpliendo esos requisitos, solo hay que identificar el proyecto y pulsar sobre el botón correspondiente de recalcular, situado en la parte derecha de la tabla.

Importar proyectos

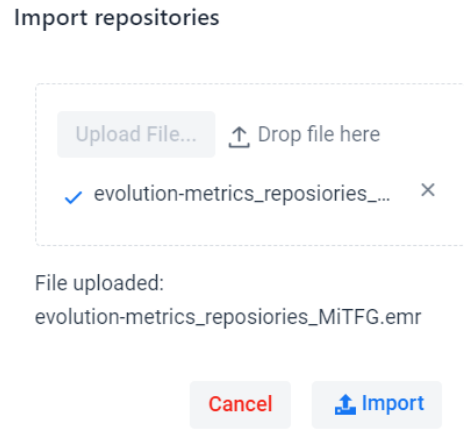
Los proyectos se pueden importar independientemente del tipo de conexión que exista. Se mostrará un diálogo que permite seleccionar o arrastrar al cuadro un fichero con formato *.emr*, ver figura E.9. Se puede seleccionar otro fichero en caso de haber escogido un fichero no deseado. Una vez se cargue 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.

Al pulsar sobre "Import" puede ocurrir:

- Que no haya ningún proyecto en la tabla, entonces se importarán todos los proyecto del fichero.



(a) Diálogo para importar repositorios



(b) Diálogo de importar con fichero cargado

Figura E.9: Diálogo para importar repositorios

- 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.

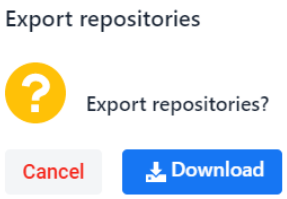


Figura E.10: Diálogo de exportación

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 [3][**?**]. 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)

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

- 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

Junto con las métricas se proporciona un indicador (*Calif.*) del porcentaje de issues 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] David Blanco Alonso. Activiti-api - monitor multiplataforma de la actividad de un proyecto. URL https://github.com/dba0010/Activiti-Api. original-date: 2015-05-09T16:20:30Z.

[2] Ministerio de Empleo y Seguridad Social. Bases y tipos de cotización 2019. URL http://www.seg-social.es/wps/portal/wss/internet/Trabajadores/CotizacionRecaudacionTrabajadores/36537. [Online; Accedido 15-Septiembre-2019].

[3] Free Software Foundation. Lista de licencias con comentarios. URL https://www.gnu.org/licenses/license-list.html.

[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. ISBN 84-7829-059-1.

[5] Carlos López Nozal. Measuring of agile practices in final year project. URL https://github.com/clopezno/clopezno.github.io/tree/master/agile_practices_experiment. 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. URL https://www.researchgate.net/profile/Yania_Crespo/publication/221595114_Soporte_de_Metricas_con_Independencia_del_Lenguaje_para_la_Inferencia_de_Refactorizaciones/links/09e4150b5f06425e32000000/Soporte-de-Metricas-con-Independencia-del-Lenguaje-para-la-

Inferencia-de-Refactorizaciones.pdf. [Online; Accedido 03-Octubre-2018].

[7] Scrum Master. *Scrum Manager: Temario Troncal I.* v. 2.61 edition. URL https://www.scrummanager.net/files/scrum_manager.pdf. [Online; Accedido 19-Noviembre-2018].

[8] Jacek Ratzinger. sPACE: Software project assessment in the course of evolution. URL http://www.inf.usi.ch/jazayeri/docs/Thesis_Jacek_Ratzinger.pdf. [Online; Accedido 03-Octubre-2018].

[9] Agencia Tributaria. Cuadro informativo tipos de retención aplicables (2019), . URL https://www.agenciatributaria.es/AEAT.internet/Inicio/La_Agencia_Tributaria/Campanas/Retenciones/Cuadro_informativo_tipos_de_retencion_aplicables__2019_.shtml. [Online; Accedido 15-Septiembre-2019].

[10] Agencia Tributaria. Tabla de coeficientes de amortización lineal, . URL https://www.agenciatributaria.es/AEAT.internet/Inicio/_Segmentos_/Empresas_y_profesionales/Empresas/Impuesto_sobre_Sociedades/Periodos_impositivos_a_partir_de_1_1_2018/Base_imponible/Amortizacion/Tabla_de_coeficientes_de_amortizacion_lineal_.shtml. [Online; Accedido 15-Septiembre-2019].