UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

Comparador de métricas de evolución en repositorios software

Presentado por Miguel Ángel León Bardavío
en Universidad de Burgos — 31 de agosto de 2019
Tutor: Carlos López Nozal

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

D. profesor del departamento de Ingeniería Civil, Área de Lenguajes y Sistemas Informáticos.

Expone:

Que el alumno D. Miguel Ángel León Bardavío, con DNI 71362165L, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado *Comparador de métricas de evolución en repositorios software.*

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

En Burgos, 31 de agosto de 2019

Vº. Bº. del Tutor:

D. Carlos López Nozal

Resumen

El proceso del software es un conjunto de actividades cuya meta es el desarrollo o evolución de software. Algunos ejemplos de estas actividades son: la especificación, el diseño, la implementación, pruebas, el aseguramiento de calidad, la configuración del proyecto, etc.

Los repositorios de código, además de almacenar el código fuente de un proyecto software, pueden incluir sistemas que faciliten las actividades del proceso de software: sistemas de control de versiones, sistemas de seguimiento de incidencias, sistemas de revisión de código, sistemas de despliegue de ejecutables, etc. En la última década han surgido forjas de repositorios que permiten alojar múltiples proyectos, estas son útiles tanto para proyectos empresariales como para proyectos open source.

Las métricas de evolución ayudan a cuantificar características de los procesos software. Un ejemplo de este tipo de medidas es el *número de días de cierre*, en la que se mide el numero de días que pasan desde que se abre una incidencia hasta su cierre. Estas métricas se pueden obtener gracias a los datos estadísticos que proporcionan los repositorios.

En este TFG se diseña un software para calcular métricas de evolución sobre distintos repositorios. En el diseño se ha optado por implementar una aplicación web en Java que toma como entrada un conjunto de repositorios públicos o privados de GitLab y calcula métricas de evolución que permiten comparar los proyectos. Además, se ha procurado un diseño extensible a otras forjas de repositorios y a nuevas métricas. La aplicación ha sido probada con Trabajos Fin de Grado presentados en la Universidad de Burgos y que han sido almacenados en repositorios públicos de GitLab.

Descriptores

Métricas de evolución, repositorios de código, proceso de desarrollo de software, ciclo de vida de desarrollo de software, gestión de calidad, forjas de repositorios, comparación de proyectos software, aplicación web.

Abstract

The software process is a set of activities whose goal is the development or evolution of software. Some examples of these activities are: specification, design, implementation, testing, quality assurance, project management, etc.

The source code repositories, in addition to storing the source code of a software project, may include systems that ease the activities of the software development process: version control systems, issue tracking systems, code review systems, deployment systems, etc. Forges of source code repositories have emerged in the last decade that allow hosting multiple projects, these are useful for both business projects and open-source projects.

Evolution metrics helps to quantify features of a software development process. An example of this type of measure is the *days to close an issue*, in which the number of days that pass from when an incident is opened until its closure is measured. These metrics can be obtained from the statistics provided by the source code repositories.

In this project, a software is designed to calculate evolution metrics on different source code repositories. The design has chosen to implement a web application in Java language that takes as input a set of GitLab public or private repositories and calculates evolution metrics that allow the repositories to be compared. In addition, an extensible design to other repositories forges and new metrics has been sought. The application has been tested with Final Degree Projects presented at the University of Burgos and that have been stored in public repositories of GitLab.

Keywords

Evolution metrics, source code repositories, software development process, software development life cycle, quality management, forge of repositories, comparison of software projects, web application.

Índice general

Índice de figuras

Índice de tablas

Introducción

Un proceso software es un conjunto de actividades cuya meta es el desarrollo o evolución del software. Durante el proceso intervienen múltiples factores: el equipo de desarrollo, el tipo de producto software, la estabilidad de los requisitos funcionales, la importancia de los requisitos no funcionales como escalabilidad, seguridad, licencias, lenguaje de programación, tipo de arquitectura de computación, etc. Esto hace que el proceso sea bastante complejo.

Para superar esta complejidad se definen modelos que ayudan a definir las actividades y artefactos del proceso de desarrollo de software. Los artefactos son las salidas de las actividades y el conjunto de artefactos conforman el producto software. En el caso de Unified Process (UP) [8] se identifican las siguientes actividades o flujos de trabajo: recolección de requisitos, diseño e implementación, pruebas y despliegue. Además en UP se añaden tres flujos de trabajo de soporte: configuración de cambios, gestión de proyecto y gestión de entorno. Estos flujos de trabajo se aplican iterativamente durante varias fases del desarrollo en cada una de las cuales se incrementa el producto software con algún artefacto resultado de la actividad. La característica de iteración e incremental es recogida en otros métodos o buenas prácticas del desarrollo ágil [5]: Scrum, eXtreme Programming, Lean...

Los repositorios de código son espacios virtuales donde los equipos de desarrollo generan los artefactos colaborativos procedentes de las actividades de un proceso de desarrollo. Además de guardar los artefactos, la versión final y anteriores versiones, estos repositorios, normalmente, permiten almacenar la interacción de los miembros del equipo justificando el cambio de versión. Dependiendo del artefacto generado se utiliza distintos sistemas: foros de comunicación, sistemas de control de versiones, sistemas de gestión de incidencias, sistemas de gestión de pruebas, sistemas de revisiones de calidad,

sistemas de integración y despliegue continuo, etc. [7].

En la última década han surgido forjas de proyectos software de fácil acceso tanto para proyectos empresariales como para proyectos open-source (SourceForge [1], GitHub [2], GitLab [3], Bitbucket [4]). Estas forjas suelen integrar múltiples sistemas para dar soporte a los flujos de trabajo y registrar las interacciones entre los miembros del equipo. Además dan la posibilidad de extensión funcional con sistemas de terceros para gestionar otras actividades no soportadas directamente por la propia forja, por ejemplo Travis CI [5] para gestionar la integración continua o Codacy [6] para gestionar las revisiones automáticas de calidad.

Parece lógico considerar como hipótesis que la calidad de un artefacto software tenga alguna relación con la manera en la que el equipo de desarrollo aplica las actividades del proceso dentro del repositorio. Sommerville expone en *Ingeniería de software* [14] que la calidad del proceso es uno de los factores que afectan a la calidad del producto, junto con las tecnologías utilizadas para el desarrollo, la calidad del personal y el coste, tiempo y duración del proyecto. La validación empírica de estas hipótesis ha abierto una nueva línea de aplicación con los conjuntos de datos que se pueden extraer de estos repositorios gracias a interfaces de programación específicas que proporcionan estas forjas de repositorios y que permiten acceder a toda la información registrada.

El desafío a la comunidad científica y empresarial es constante mostrando un incremento en el interés en las aplicaciones que permitan mejorar sus sistemas de decisión. Estas aplicaciones deberán llevar un control sobre el proceso y/o sobre el producto software y ese control se podrá realizar mediante un proceso de medición. La medición puede ser de control o de medición. La primera se asocia al proceso, mientras que la segunda se asocia al producto software. Estas forjas de proyectos software están en constante evolución, tanto en sus estructuras estáticas como en sus interacciones dinámicas en los proyectos. Se registran grandes conjuntos de datos difíciles de procesar y son de estos donde se pueden obtener tantas métricas de control como de predicción.

En este TFG se diseña un software para calcular métricas de control [7]

[1] https://sourceforge.net/
[2] https://github.com/
[3] https://about.gitlab.com/
[4] https://bitbucket.org/
[5] https://travis-ci.org/
[6] https://www.codacy.com/
[7] También llamadas métricas de proceso o métricas de evolución

sobre distintos repositorios. En el diseño se ha optado por implementar una aplicación web escrita en lenguaje Java que toma como entrada un conjunto de repositorios públicos o privados de GitLab y calcula métricas de evolución que permiten comparar los proyectos. Además, se ha procurado un diseño extensible a otras forjas de repositorios y se ha facilitado la incorporación de nuevas métricas. La aplicación ha sido probada con Trabajos Fin de Grado presentados en la Universidad de Burgos y que han sido almacenados en repositorios públicos de GitLab.

1.1. Estructura de la memoria

La memoria de este trabajo se estructura de la siguiente manera[8][15]:

Introducción. Introducción al trabajo realizado, estructura de la memoria y listado de materiales adjuntos.
Objetivos del proyecto. Objetivos que se persiguen alcanzar con la realización del proyecto.
Conceptos teóricos. Conceptos clave para comprender los objetivos, el proceso y el producto del proyecto.
Técnicas y herramientas. Técnicas y herramientas utilizadas durante el desarrollo del proyecto.
Aspectos relevantes del desarrollo. Aspectos destacables durante el proceso de desarrollo del proyecto.
Trabajos relacionados. Otros proyectos de la misma naturaleza y los cuales han ayudado a la realización de este.
Conclusiones y líneas de trabajo futuras. Conclusiones tras la realización del proyecto y posibilidades de mejora o expansión.

Se incluyen también los siguientes anexos:

Plan del proyecto software. Planificación temporal y estudio de la viabilidad del proyecto.
Especificación de requisitos del software. Análisis de los requisitos.
Especificación de diseño. Diseño de los datos, diseño procedimental y diseño arquitectónico.
Manual del programador. Aspectos relevantes del código fuente.
Manual de usuario. Manual de uso para usuarios que utilicen la aplicación.

[8] https://github.com/ubutfgm/plantillaLatex

Objetivos del proyecto

En este capítulo se detallarán los objetivos generales que se desean alcanzar en este proyecto, así como los objetivos más técnicos.

2.1. 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 [12] 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.

A continuación se desglosa el objetivo general en objetivos más detallados.

- 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 *sPACE: Software Project Assessment in the Course of Evolution* [12] 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.

2.2. Objetivos técnicos

Este apartado recoge los requisitos más técnicos del proyecto.

- Diseñar la aplicación de manera que se puedan extender con nuevas métricas con el menor coste de mantenimiento posible. Se aplicará un diseño basado en frameworks y en patrones de diseño [6].
- El diseño de la aplicación debe facilitar la extensión a otras plataformas de desarrollo colaborativo como GitHub o Bitbucket. Se aplicará un diseño basado en frameworks y en patrones de diseño [6].
- Aplicar el *frameworks* modelo vista controlador para separar la lógica de la aplicación para el calculo de métricas y la interfaz de usuario. Se aplicará un diseño basado en frameworks y en patrones de diseño [6].
- Crear una batería de pruebas automáticas con cobertura del 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*).

Conceptos teóricos

En este capítulo se explican conceptos relevantes para la comprensión de este proyecto y su contexto.

3.1. Evolución de software: Proceso o ciclo de vida de un proyecto software

> El ciclo de vida del software es un marco de referencia que contiene los procesos, las actividades y las tareas involucradas en el desarrollo, la explotación y el mantenimiento de un producto de software, abarcando la vida del sistema desde la definición de los requisitos hasta la finalización de su uso.
>
> ISO 12207-1

Un proceso del software es un conjunto de actividades cuya meta es el desarrollo de software desde cero o la evolución de sistemas software existentes. Para representar este proceso se utilizan modelos de procesos, que no son más que representaciones abstractas de este proceso desde una perspectiva particular. Estos modelos son estrategias para definir y organizar las diferentes actividades y artefactos del proceso. Los artefactos son las salidas de las actividades y el conjunto de artefactos conforman el producto software. Actividades comunes a cualquier modelo son:

- **Especificación:** En esta actividad se define la funcionalidad del software y los requerimientos que ha de cumplir.
- **Diseño e implementación:** En esta fase se define el diseño del software, se generan los artefactos y se realizan pruebas sobre ellos.

– **Validación:** En esta fase se debe asegurar que los artefactos generados cumplen con su especificación.

– **Evolución:** Fase asociada a la **corrección** de defectos o fallos; **adaptación** del software a cambios en el entorno en el que se utiliza; **mejora** y ampliación; y **prevención** mediante técnicas de ingeniería inversa y reingeniería como la refactorización.

Existen modelos de proceso generales como el tradicional modelo en cascada de los 80 (Ver Fig. 3.1) o el modelo incremental recogido en métodos y buenas prácticas del desarrollo ágil [5]: Scrum, eXtreme Programming, Lean... (Ver Fig. 3.2). En el caso de *Unified Process* (UP) [8] se identifican las siguientes actividades o flujos de trabajo: recolección de requisitos, diseño e implementación, pruebas y despliegue. Además en UP se añaden tres flujos de trabajo de soporte: configuración de cambios, gestión de proyecto y gestión de entorno. Estos flujos de trabajo se aplican iterativamente durante varias fases del desarrollo en cada una de las cuales se incrementa el producto software con algún artefacto resultado de la actividad.

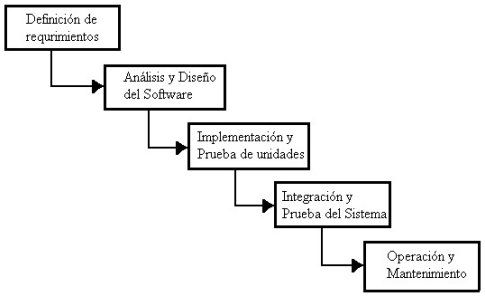

Figura 3.1: Modelo de proceso en cascada [1]

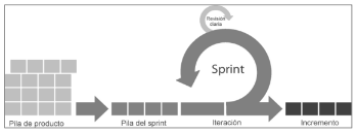

Figura 3.2: Modelo de proceso incremental: Scrum [5]

Sin embargo, estos modelos generales deben ser extendidos y adaptados para crear modelos más específicos. No existe un único proceso ideal para construir todos los productos software, ya que este proceso depende de la naturaleza del proyecto y de otros factores como el equipo de desarrollo, la estabilidad de los requisitos funcionales, la importancia de los requisitos no

funcionales como escalabilidad, seguridad, licencias, lenguaje de programación, tipo de arquitectura de computación, etc. Todos estos factores hacen que el proceso sea bastante complejo y que se requiera un modelo diferente para cada proyecto.

3.2. Repositorios y forjas de proyectos software

En el apartado anterior se habla sobre la complejidad de un proceso software y que este puede ser representado por modelos que ayudan a organizar las diferentes actividades. En este apartado se hablará sobre metodologías y herramientas que pueden ayudar en más de una actividad del ciclo de vida.

Los repositorios de código son espacios virtuales donde los equipos de desarrollo generan los artefactos colaborativos procedentes de las actividades de un proceso de desarrollo. Estas herramientas permiten a un equipo de desarrollo trabajar en paralelo, lo que en ingeniería del software es complicado ya que , por lo general, miembros del mismo equipo necesitan trabajar sobre el mismo fichero y esto genera conflictos. Normalmente estos espacios se encuentran en servidores por motivos de seguridad y facilitar a los miembros del equipo puedan acceder al repositorio.

Un buen repositorio no solo permite almacenar los artefactos generados por cada una de las actividades del ciclo de vida del software. Sino que, también, permite llevar un historial de cambios e incluso ayudará a entender el contexto de la aplicación: quién ha realizado los cambios y porqué, es decir que permite almacenar las interacciones entre los miembros del equipo. Para ello se utilizan distintos sistemas, dependiendo del artefacto generado: foros de comunicación, sistemas de control de versiones como *Git*, sistemas de gestión de incidencias, sistemas de gestión de pruebas, sistemas de revisiones de calidad, sistemas de integración y despliegue continuo, etc. [7].

Además de estos repositorios han surgido en la última década forjas de proyectos software de fácil acceso tanto para proyectos empresariales como para proyectos open-source (SourceForge [9], GitHub [10], GitLab [11], Bitbucket

[9] https://sourceforge.net/
[10] https://github.com/
[11] https://about.gitlab.com/

[12]). Este mismo proyecto está almacenado en un repositorio en GitLab[13], ver Fig. 3.3.

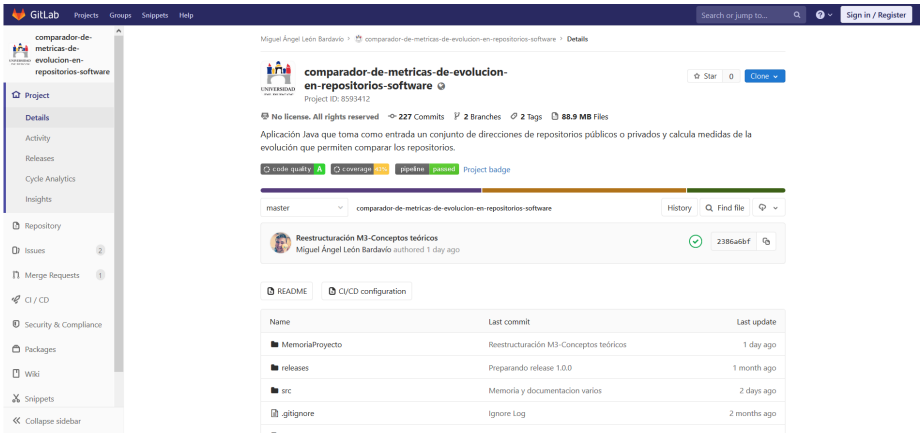

Figura 3.3: Captura de este proyecto almacenado en GitLab

Estas forjas suelen ofrecer servidores para almacenar repositorios e integran múltiples sistemas para dar soporte a los flujos de trabajo y registrar las interacciones entre los miembros del equipo, también ofrecen posibilidades para usar estos sistemas en un servidor particular. Además dan la posibilidad de extensión funcional con sistemas de terceros para gestionar otras actividades no soportadas directamente por la propia forja, como Travis CI [14] para gestionar la integración continua o Codacy [15] para gestionar las revisiones automáticas de calidad como se puede observar en la Fig. 3.4.

[12] https://bitbucket.org/

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

[14] https://travis-ci.org/

[15] https://www.codacy.com/

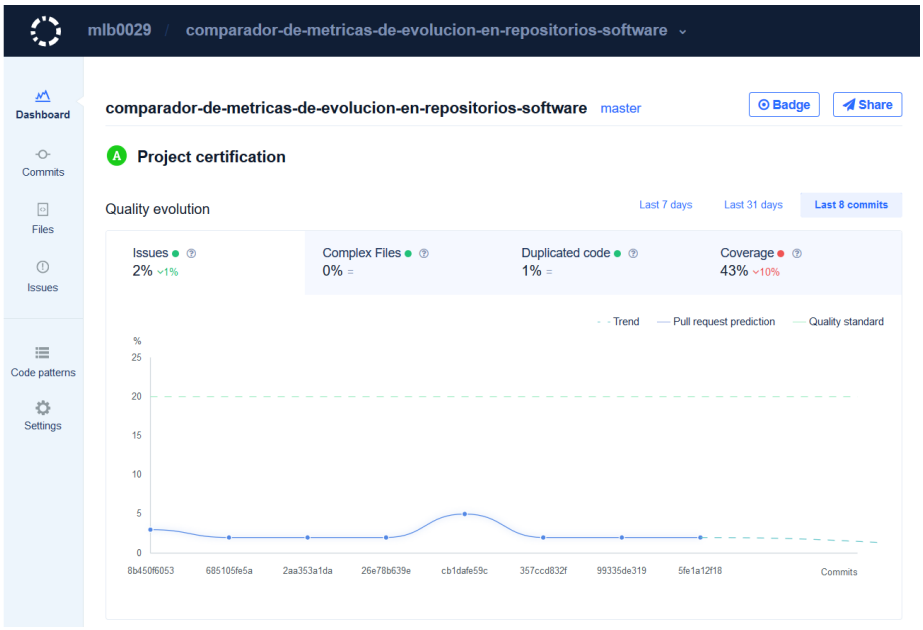

Figura 3.4: Revisión automática de calidad realizada con Codacy sobre este proyecto

Actualmente estas forjas han tenido una gran aceptación entre la comunidad de desarrolladores y existen muchos desarrollos de software de tendencia que las utilizan. En la Fig. 3.5 se aprecia como cambia la tendencia de utilización de dichas forjas en el tiempo. Actualmente la forja predominante es claramente GitHub pero se ve un incremento en el uso de GitLab.

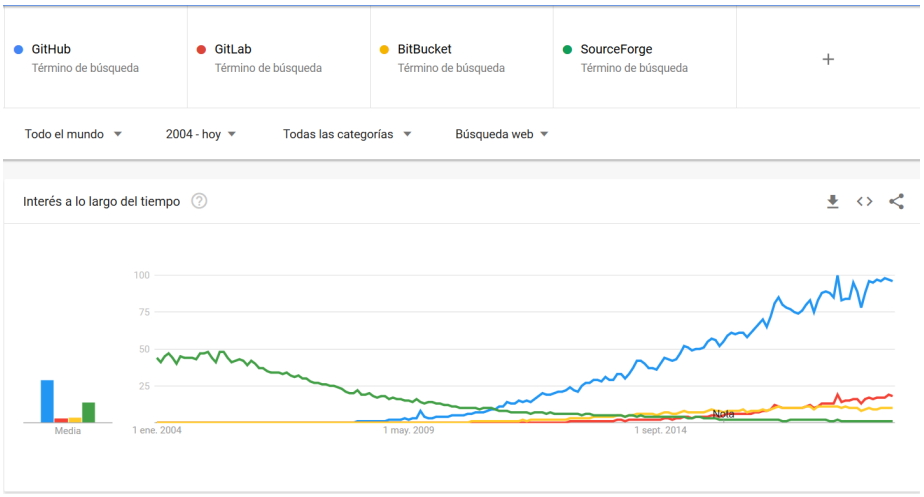

Figura 3.5: Comparativa de tendencia de búsqueda de Google desde 2004 con los términos de distintas forjas de proyectos software

Estas forjas de proyectos software están en constante evolución, tanto en sus estructuras estáticas como en sus interacciones dinámicas en los proyectos y se registran grandes conjuntos de datos difíciles de procesar. Sin embargo, las forjas de proyectos software proporcionan interfaces de programación específicas que permiten acceder a toda la información registrada.

El desafío a la comunidad científica y empresarial es constante mostrando un incremento en el interés en las aplicaciones de minería que mejoren sus sistemas de decisión [7]. Estos datos que registran las forjas de repositorios pueden ser utilizados para mejorar estos sistemas de decisión en función de la evolución del proyecto.

GitHub vs. GitLab

Se ha hablado anteriormente de las forjas de repositorios como GitHub o GitLab y se puede observar en la Fig. 3.5 la tendencia en el uso de diferentes forjas. Se observa como GitHub predomina sobre las demás y como crece el uso de GitLab. En esta sección se comparan los aspectos más relevantes de estas dos tendencias según los servicios que ofrecen.

CI/CD - Continuous Integration/Continuous Delivery

La integración y despliegue continuo son prácticas sirve para para construir, probar y, en caso de tratarse de una página web o una aplicación web, desplegar la aplicación una vez se combinen los cambios en el repositorio

central, como se puede observar en Fig. 3.6. Ambos ofrecen la posibilidad de realizar este proceso mediante software de terceros como *Travis CI*. Sin embargo, GitLab ofrece ejecutores o *runners* propios para llevar este proceso desde GitLab. De hecho, en este trabajo con repositorio en GitLab se ha realizado este proceso definiendo un flujo de trabajos o *pipeline* de la forma mostrada en la Fig. 3.7.

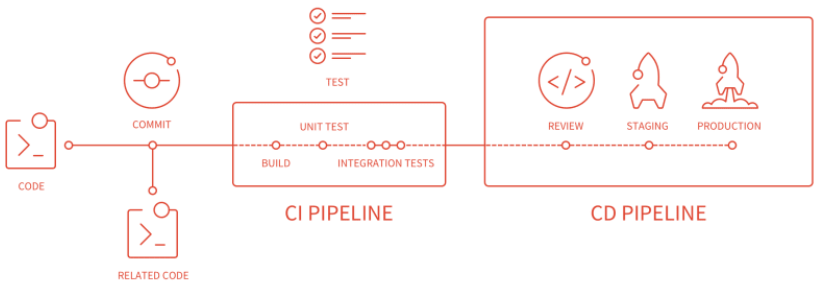

Figura 3.6: CI/CD con GitLab [3]

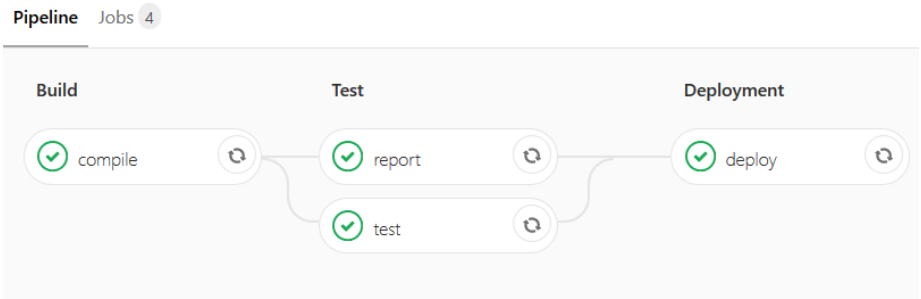

Figura 3.7: Flujo de trabajo para el proceso de CI/CD definido para este proyecto

Estadísticas e informes

Ambos ofrecen estadísticas e informes sobre los datos que registran de los repositorios y pueden ser accedidos visualmente desde la web del repositorio o desde APIs de programación. Por ejemplo, las métricas que trabaja este proyecto se calculan a partir de datos proporcionados por estas APIs.

Algo que ofrece GitLab y no GitHub es la monitorización del rendimiento de las aplicaciones que se hayan desplegado.

Importación y exportación de proyectos

A diferencia de GitHub, GitLab ofrece la posibilidad de importar proyectos desde otras fuentes como GitHub, Bitbucket, Google Code, etc. También es posible exportar proyectos de GitLab a otros sistemas.

Sistema de seguimiento de incidencias (issues)

Ambos cuentan con un sistema de seguimiento de incidencias (*issue tracking system* o *issue tracker*), permiten crear plantillas para las incidencias, adornarlas con Markdown[16], usar etiquetas o *labels* para categorizarlas, asignarlas a uno o varios miembros del equipo y bloquearlas para que solo puedan comentarlas los miembros del equipo.

Sin embargo GitLab da un paso más y permite asignar peso a las tareas, crear *milestones*, asignar fechas de vencimiento, marcar la incidencia como confidencial, relacionar incidencias, mover o copiar incidencias a otros proyectos, marcar incidencias duplicadas, exportarlas a CSV, entre otras cosas. Otros aspectos destacables de GitLab en cuanto a este tema son los gráficos Burndown de los milestones (ver Fig. 3.8), acciones rápidas y la gestión de una lista de quehaceres (*todos*) de un usuario cuándo a este se le asignan incidencias.

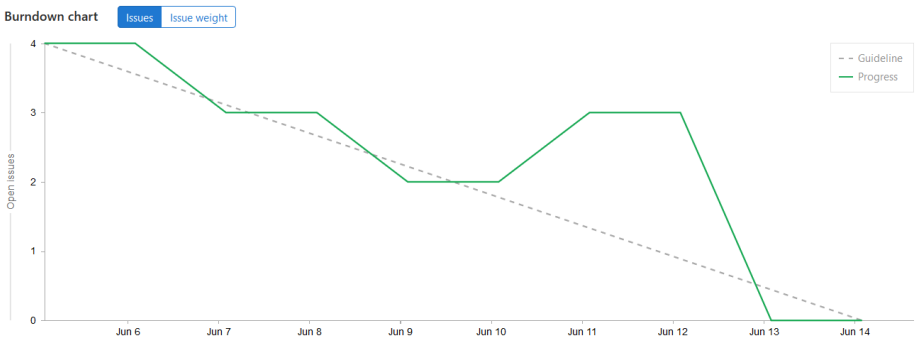

Figura 3.8: Ejemplo de gráfico burndown

A diferencia de GitLab, GitHub mantiene un historial de cambios en los comentarios de una incidencia; permite asignar las incidencias a listas mediante “drag and drop"; proporciona información útil al pasar el ratón por encima de elementos de la web como usuario, issues, etc.

[16]Markdown es un lenguaje de marcado que facilita la aplicación de formato a un texto empleando una serie de caracteres de una forma especial[9]

Wiki

En ambas forjas es posible disponer de una wiki para el proyecto.

Otros aspectos destacables

- GitHub permite repositorios 100 % binarios
- GitLab permite tener una instancia propia de GitLab en un servidor particular, lo que permite que se pueda gestionar software adicional dentro del servidor como sistemas de detección de intrusos o un monitor de rendimiento.
- GitLab permite elegir miembros del equipo como revisores de "*merge requests*".
- El código de Gitlab EE puede ser modificado para ajustarlo a las necesidades de seguridad y desarrollo.
- Ambos incluyen APIs que permiten realizar aplicaciones que se integren con GitLab o GitHub. Esto ha sido clave para la realización de este proyecto, como se ha mencionado anteriormente.
- GitLab nos proporciona un IDE[17] web para realizar modificaciones sobre el código desde el mismo GitLab, también incluye un terminal web para el IDE que permite, por ejemplo, compilar el código.
- Ambos permiten la integración con repositorios Maven[18]

3.3. Calidad de un producto software

El software debe tener la funcionalidad y el rendimiento requeridos por el usuario, además de ser mantenible, confiable, eficiente y fácil de utilizar.

La calidad de un producto de software no tiene que ver solo con que se cumplan todos los requisitos funcionales, sino también otros requerimientos no funcionales que no se incluyen en la especificación como los de mantenimiento, eficiencia y usabilidad.

Sommerville enumera en *Software Engineering* [14] los principales factores que afectan a la calidad del producto, como se puede observar en la Fig. 3.9:

[17] Integrated Development Environment - Entorno de Desarrollo Integrado
[18] Software de gestión de proyectos software: https://maven.apache.org/

- Calidad del proceso
- Tecnología de desarrollo
- Calidad del personal
- Costo, tiempo y duración

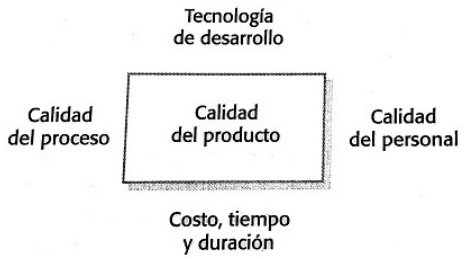

Figura 3.9: Principales factores de calidad del producto de software[14]

Para llegar a tener un software de calidad hay que tener en cuenta todos los factores mencionados anteriormente en cada una de las tres fases de la **administración de la calidad**: aseguramiento, planificación y control.

Aseguramiento de la calidad. Se encarga de establecer un marco de trabajo de procedimientos y estándares que guíen a construir software de calidad.

Planificación de la calidad. Selección de procedimientos y estándares para un proyecto software especifico.

Control de la calidad. La fase de control es la que se encarga de que el equipo de desarrollo cumpla los estándares y procedimientos definidos en el plan de calidad del proyecto. Esta fase puede realizarse mediante revisiones de calidad llevados a cabo por un grupo de personas y/o mediante un proceso automático llevado a cabo por algún programa.

Control de la calidad: medición

La fase de control de calidad es en la que se vigila que se sigan los procedimientos y estándares definidos en el plan de calidad. Pero estos podrían no ser adecuados o siempre pueden mejorar, por lo que en esta fase se puede valorar el mejorarlos como se puede observar en la Fig. 3.10.

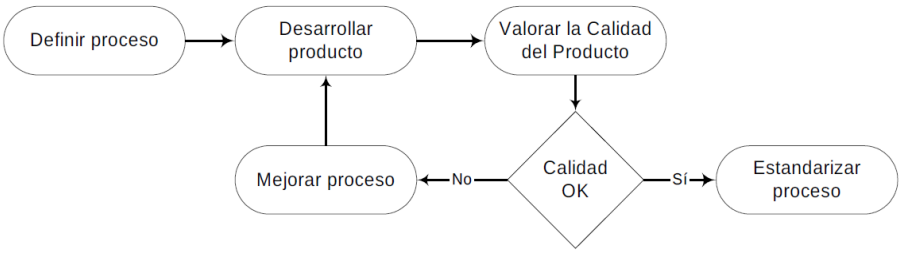

Figura 3.10: Calidad basada en procesos [14]

Este proceso puede ser llevado a cabo mediante revisiones llevadas a cabo por un grupo de personas o por medio de programas que automaticen este proceso. El desafío a la comunidad científica y empresarial es constante mostrando un incremento en el interés de aplicaciones que permiten mejorar sus sistemas de decisión. Estas aplicaciones deberán llevar un control sobre el proceso y/o sobre el producto software y ese control se podrá realizar mediante un proceso de medición, esto ofrece una medida cuantitativa de los atributos del producto y del proceso software.

La medición del software es un proceso en el cual se asignan valores numéricos o simbólicos a atributos de un producto o proceso software. Una métrica de software es una medida cuantitativa del grado en que un sistema, componente o proceso software posee un atributo dado. Las métricas son de control o de predicción. Las **métricas de control** se asocian al proceso de desarrollo del software, por ejemplo, la media de días que se tarda en cerrar una incidencia; y las **métricas de predicción** se asocian a productos software, por ejemplo, la complejidad ciclomática de una función. Y ambos tipos de métricas influyen en la toma de decisiones administrativas como se observa en la Fig. 3.11. Los repositorios y las forjas facilitan la obtención de datos para este proceso de medición.

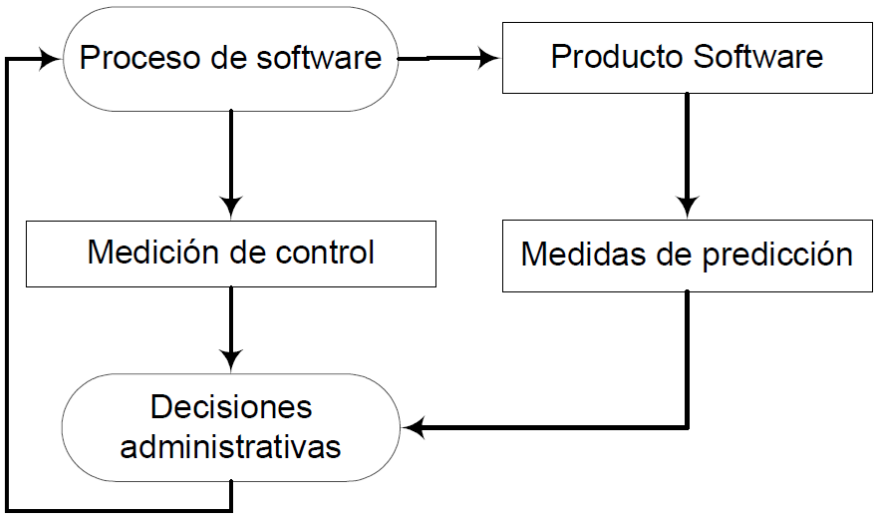

Figura 3.11: Métricas de control y métricas de predicción[14]

Este proyecto se centra solo en la obtención de métricas de evolución que permitirán controlar y evaluar el proceso del desarrollo de un producto software. Por tanto se dejarán las métricas de predicción para otros trabajos y se detallará más sobre las de control en el siguiente apartado.

Métricas de control: medición de la evolución o proceso de software

En la Fig. 3.9 se muestra la calidad de proceso como factor que afecta directamente a la calidad de producto. Parece lógico considerar como hipótesis que la calidad de un artefacto software tenga alguna relación con la manera en la que el equipo de desarrollo aplica las actividades del ciclo de vida del software dentro del repositorio. La validación empírica de estas hipótesis ha abierto una nueva línea de aplicación con los conjuntos de datos que se pueden extraer de los repositorios gracias a interfaces de programación específicas que proporcionan estas forjas de repositorios y que permiten acceder a toda la información registrada.

Una plataforma de desarrollo colaborativo como GitLab puede presentar herramientas para controlar la evolución de un proyecto software. Por ejemplo un sistema de control de versiones (VCS - *Version Control System*) como Git, un sistema de seguimiento de incidencias (*Issue tracking system*), un sistema de integración continua (CI - *Continuous Integration*), un sistema de despliegue continuo (CD - *Continuous deployment*), etc. Todas estas

herramientas facilitan la comunicación entre los miembros de un equipo de desarrollo, ayudan a gestionar los cambios que producen cada uno de los miembros y proporcionan mediciones de proceso. Estas mediciones se pueden utilizar para obtener métricas de control que ayuden evaluar y mejorar la evolución del proyecto.

Las métricas de control que se utilizan en este proyecto provienen una Master Tesis titulada *sPACE: Software Project Assessment in the Course of Evolution* [12]. A continuación se describen las métricas que se implementan en este proyecto usando la plantilla de definición de la norma ISO 9126.

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[5].
- **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*

Framework de medición

Para la implementación de las métricas se ha seguido la solución basada en frameworks propuesta en *Soporte de Métricas con Independencia del Lenguaje para la Inferencia de Refactorizaciones* [10]. El objetivo del *framework* es la reutilización en la implementación del cálculo de métricas. Este diseño, mostrado en la Fig. 3.12, permite:

- Facilitar el desarrollo de nuevas métricas
- Personalizar de los valores límite inferior y superior ya que estos pueden variar dependiendo del contexto en el que se calculen las métricas.
- Crear un grupo de configuraciones de métricas, de forma que se podrían calcular todas las métricas de ese grupo y almacenar los resultados en un colector de tipo *MetricResult*

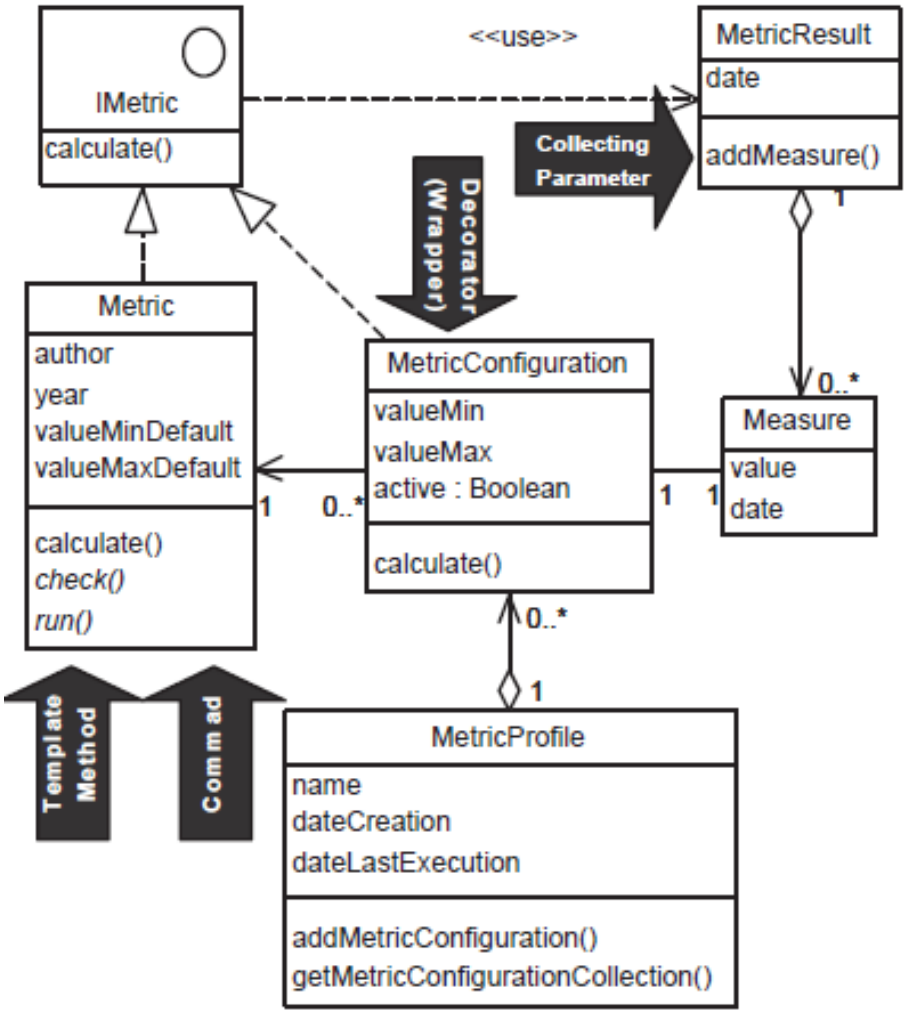

Figura 3.12: Diagrama del framework para el cálculo de métricas con perfiles que almacena valores umbrales.

En el diagrama UML de la Fig. 3.12 se muestran las entidades principales del framework de medición y la relación entre ellas, especificando la navegabilidad y la multiplicidad. Las anotaciones en forma de flecha oscura especifican los patrones de diseño[6] aplicados en el framework.

Para crear una nueva métrica, esta deberá extender de la clase abstracta *Metric* e implementar los métodos *check()* y *run()*. El método *calculate()* de la clase *Metric* utiliza el patrón de diseño **método plantilla**[19] que utiliza

[19] https://refactoring.guru/design-patterns/template-method

estos dos métodos, que deberán ser implementados por las clases concretas que hereden de *Metric*. Un ejemplo de este método sería:

```
...
Value calculate(Entity entity,
                MetricConfiguration metricConfig,
                MetricsResults metricsResults)
{
        Value value;
        if(check(entity))
        {
                value = run(entity);
                metricsResults.addMeasure(new Measure(metricConfig, value
        }
        return value;
}
...
```

Siendo *entity* la entidad que se está midiendo, *metricConfig* la configuración de valores limite que se está utilizando, *metricsResults* el lugar donde se almacena el resultado y *value* el valor medido en el método *run()*. La plantilla establece que primero se comprueba que se pueda calcular la métrica y en ese caso se calcula y se añade a la coleción *metricsResults*. Este método delega en las subclases el comportamiento de los métodos *check()* y *run()*. Además, almacena en el objeto colector *metricsResults*, pasado como argumento, el valor medido para la configuración de la métrica para posibilitar el análisis y presentación de los resultados posteriores.

MetricConfiguration toma el rol de decorador en el patrón de diseño **decorador**[20] que permite configurar los valores límite de las métricas. Implementa la misma interfaz que *Metric*, *IMetric*, y esta asociado a una métrica. Su método *calculate()* simplemente llamará al método *calculate()* de la métrica (*Metric*) a la que esta asociada la configuración.

Un perfil de métricas agrupa un conjunto de configuraciones de métricas para un contexto dado, por ejemplo, para un conjunto de proyectos realizados por alumnos de la universidad en su realización del TFG. Se podría instanciar un *MetricResult* para almacenar los resultados de toda esta colección de configuraciones de métricas y bastaría solo con recorrer el perfil usando el método *calculate()* de cada configuración.

[20] https://refactoring.guru/design-patterns/decorator

Este TFG ha adaptado este framework en el paquete "motor de métricas", y se han realizado unas pocas modificaciones. Las modificaciones más destacadas son:

- Se ha aplicado a las métricas concretas el patrón ***Singleton***[21], que obliga a que solo haya una única instancia de cada métrica; y se ha aplicado el patrón ***Método fábrica***[22] tal y como se muestra en la Fig. 3.13, de forma que *MetricConfiguration* no este asociada con la métrica en si, 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. 3.14.
  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 en valor límite superior y el valor límite inferior; y para oras un valor aceptable es aquel que supere el límite inferior.
  EvaluationFunction es una interfaz funcional[23] 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 [2].
  Esto permite definir los tipos de los parámetros y de retorno de una función que se puede almacenar en una variable. De esta forma se puede almacenar en una variable la forma en la que se puede evaluar la métrica.

[21] https://refactoring.guru/design-patterns/singleton
[22] https://refactoring.guru/design-patterns/factory-method
[23]

- 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

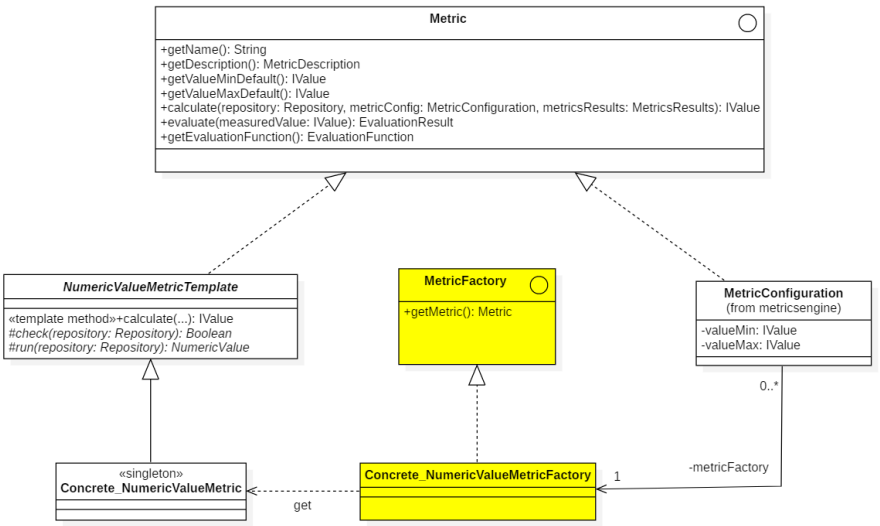

Figura 3.13: Patrones “singleton” y “método fábrica” sobre el framework de medición

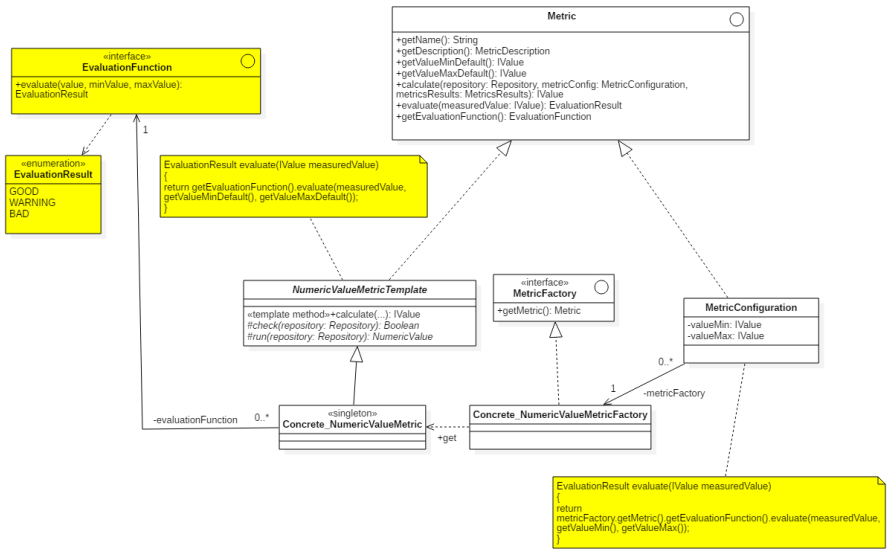

Figura 3.14: Añadido al framework de medición la evaluación de métricas

Técnicas y herramientas

Este capitulo muestra las herramientas que se han utilizado para la alcanzar los objetivos del proyecto.

4.1. Herramientas utilizadas

Entorno de desarrollo

Eclipse IDE for Java EE Developers . Entorno de programación Java para aplicaciones web. Se ha utilizado la version: 2018-09 (4.9.0).
Enlace a página de descarga:
https://www.eclipse.org/downloads/

Java SE 11 (JDK) . *Java Development Kit*. Se ha utilizado la versión v11.0.1.
Enlace a página de descarga:
https://www.oracle.com/technetwork/java/javase/downloads/index.html

Apache Maven . Gestor de proyectos software que ayuda en la construcción del proyecto, la generación de documentación, generación de informes, gestión de dependencias, integración con un sistema de control de versiones, etc. Se ha utilizado la versión v3.6.0.
Enlace a página de descarga:
https://maven.apache.org/download.cgi

Apache Tomcat . Contenedor de aplicaciones web con soporte de servlets Java. Sirve para desplegar la aplicación. Se ha utilizado la versión v9.0.13.

Enlace a página de descarga:
https://tomcat.apache.org/download-90.cgi

Logging

SLF4J . Fachada de logging.
Enlace a página de descarga:
https://www.slf4j.org/download.html

Log4j 2 . Logger. Se ha utilizado la versión v2.11.2.
Enlace a página de descarga:
https://logging.apache.org/log4j/2.x/download.html

Pruebas

JUnit5 . Conjunto de bibliotecas para el desarrollo de pruebas unitarias. Se ha utilizado la versión v5.3.1.
Enlace a página de descarga:
https://junit.org/junit5/

Frameworks y librerías específicas para el proyecto

Comentario de revisión...Además de los enlaces de la aplicación añade los enlaces de tu repositorio

gitlab4j-api . Framework de conexión a GitLab API. Se ha utilizado la versión v4.9.14.
Enlace:
https://github.com/gitlab4j/gitlab4j-api
Se ha preferido frente a timols/java-gitlab-api[24] tras realizar un estudio sobre sus métricas de evolución y una comparativa sobre la documentación. Concluyendo en que **gitlab4j-api** contiene mejor documentación, mejor evolución y a día de hoy se sigue desarrollando.

Apache Commons Math . Librería que se utiliza para matemáticas descriptivas utilizada para el cálculo de cuartiles. Se ha utilizado la versión v3.6.1.
Enlace a página de descarga:
https://commons.apache.org/proper/commons-math/download_math.cgi

[24] https://github.com/timols/java-gitlab-api

Interfaz gráfica

Vaadin . Framework para desarrollo de interfaces Web con Java. Se ha utilizado la versión v13.0.0
Enlace:
https://vaadin.com/

CI/CD y Calidad de código

GitLab . Plataforma de desarrollo colaborativo en la que se ha almacenado el proyecto en un repositorio Git.
Enlace:
https://gitlab.com/users/sign_in

Codacy . Herramienta de generación automática de informes de calidad de código.
Enlace:
https://www.codacy.com/

JaCoCo . Librería para cobertura de código en Java. Se ha utilizado la versión v0.8.3.
Enlace:
https://www.eclemma.org/jacoco/

Heroku . Herramienta para despliegue continuo.
Enlace:
https://id.heroku.com/login

Documentación

LaTeX . Sistema de composición de textos.
Enlace:
https://www.latex-project.org/

TeXstudio . Entorno de desarrollo de documentos LaTeX.
Enlace:
https://www.texstudio.org/

Zotero . Herramienta de gestión de fuentes bibliográficas.
Enlace:
https://www.zotero.org/

Técnicas

A lo largo del proyecto se han utilizado numerosos patrones de diseño [6] como Singleton, Factory Method, Wrapper, Builder, Listener, etc.

Para el motor de métricas se ha utilizado como base el framework propuesto en *Soporte de Métricas con Independencia del Lenguaje para la Inferencia de Refactorizaciones* [10]. Ver Fig. 3.12.

Aspectos relevantes del desarrollo del proyecto

Este capítulo recoge los aspectos destacables durante el desarrollo del proyecto.

5.1. Motivación de la elección

La elección de este trabajo fue motivada por su relación con las asignaturas de la mención de ingeniería del software, detacando las asignaturas:*Gestión de Proyectos*, *Diseño y Mantenimiento del Software*, *Desarrollo Avanzado de Sistemas Software*, en las que se enseña como desarrollar software de calidad usando repositorio Software.

5.2. Evolución del proyecto

Tras la elección del proyecto se acordó definir una evolución que siga las bases del modelo Scrum. Tomando un proceso de desarrollo incrementa con revisión de las iteraciones cada dos semanas. Se han definido sprints de dos semanas. Estas reuniones constaban de dos partes:

- Revisión del sprint: En las que se revisaba el incremento generado, los problemas que hubo durante su desarrollo, las soluciones que se han implementado o que se plantean para el siguiente sprint.
- Planificación del siguiente sprint: Se definían las nuevas tareas.

Las primeras fases del desarrollo fueron de investigación y configuración. Luego se planteó un diseño inicial, que fue la base para la implementación de nuevas funcionalidades aunque, con el tiempo, se fue modificando el diseño inicial para adaptarlo a las nuevas funcionalidades o para resolver ciertos problemas que aparecían. En las siguiente secciones se detalla más a fondo cada una de estas etapas de desarrollo.

En la Fig. 5.15 se muestran los primeros *sprints* del proyecto. Por cada sprint se observa su descripción la fecha de apertura y cierre y el número de issues asociadas.

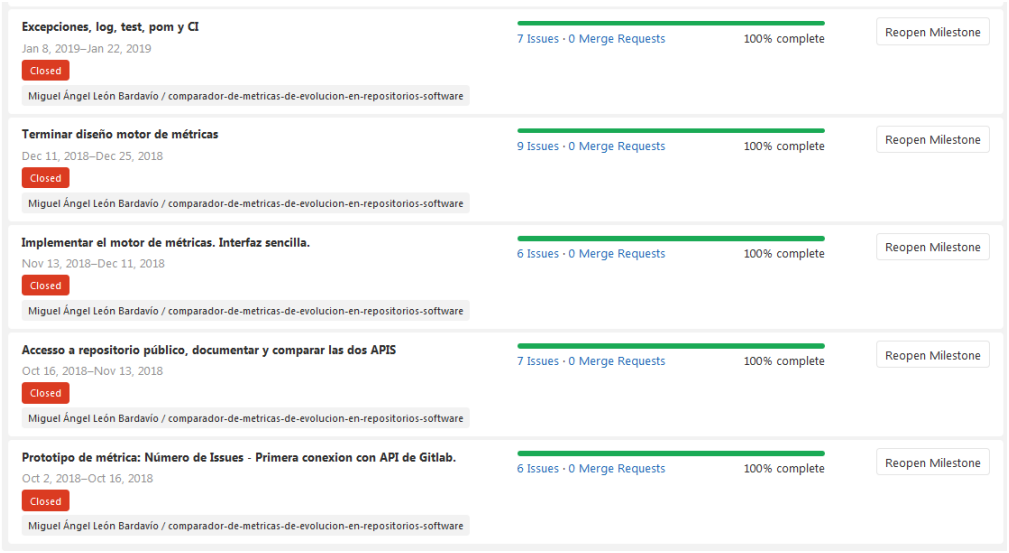

Figura 5.15: Milestone iniciales del proyecto en GitLab.

5.3. Documentación

La fase de documentación duró un mes y fue a la par con la etapa de configuración.

Se recopiló información sobre trabajos relacionados como *Activiti-Api*[13], *Soporte de Métricas con Independencia del Lenguaje para la Inferencia de Refactorizaciones* [10] y *sPACE: Software Project Assessment in the Course of Evolution* [12] y se estudió en qué entornos y qué herramientas se utilizarían para el desarrollo del proyecto.

Uno de los estudios más relevantes fue la elección de un API Java que permitiese la conexión a GitLab. Había tres opciones:

- Crear un framework propio de conexión a GitLab a partir de GitLab API. Añadía cierta complejidad al proyecto al tener que desarrollar otro módulo más, pero permitía poder definir las funciones que se necesitaban.

- Usar timols/java-gitlab-api[25][11]. Al principio fue la solución que se escogió, pero posteriormente se descubrió otro API bastante mejor. La documentación es bastante pobre y la evolución del proyecto software estaba parada o no evolucionaba bien, tenían demasiadas incidencias abiertas y no ofrecía gran parte de la funcionalidad que aportaba GitLab API.

- Usar [26][4]. Es un proyecto bastante decente y, a día de hoy, sigue creciendo. Tiene un alto porcentaje de incidencias cerradas, un gran número de releases, y evolución constante. Este es el API con el que se ha desarrollado este proyecto.

Otro de las decisiones más difíciles fue la versión de Java. Recientemente se lanzó la versión de Java 11 y es la que inicialmente se utilizó en el proyecto. Ha habido bastantes problemas de compatibilidad, pero se han ido solucionando a lo largo del tiempo.

5.4. Configuración del flujo de trabajo del desarrollo software

Compilación y empaquetado

El proyecto se iba a desarrollar en Java desde el principio, era un requisito no funcional. La versión más moderna que había cuando se comenzó el proyecto era Java 11. La decisión de utilizar esta versión de Java generó muchos problemas de compatibilidad de versiones con otras bibliotecas en el desarrollo. Comentario de revisión...Enumerar cuales tenián problemas ¿ Vaadin?

En el proyecto se ha utilizado alguna de las nuevas versiones de Java no vista durante los curso. En el siguiente fragmento de código se describe un ejemplo de código que obtiene una colección de repositorios del usuario actual.

```
...
if (connectionType != EnumConnectionType.NOT_CONNECTED) {
```

[25] https://github.com/timols/java-gitlab-api
[26] https://github.com/gitlab4j/gitlab4j-api

```
    return gitLabApi.getProjectApi().getUserProjectsStream(
                            currentUser.getId(),
                            new ProjectFilter())
                    .map(p ->
                            new Repository(p.getWebUrl(), p.getName(), p.getId()))
                    .collect(Collectors.toList());
                            }
...
```

5.5. Automatización de tareas de desarrollo

Comentario de revisión...Describir el uso de Maven Ventajas e incovenientes

Pruebas

Para las pruebas se disponían de datos de otros repositorios de GitLab que se han presentado como TFG en el Grado de Ingeniería Informática en la Universidad de Burgos. Esto ha facilitado en gran medida el trabajo de pruebas. Es destacable la funcionalidad de acceso a los repositorios por el concepto de grupo definido en Gitlab. Esto fue posible gracias a que la empresa Hewlett Packard SCDS en su colabaración con TFG con la UBU organiza su propuestas de TFG en GitLab en grupos para organizarlos por cursos académicos.

Otro aspecto destacable es la uso con funciones avanzadas en la implementación de test unitarios. Como ejemplo en el siguiente fragmento de código se recoge como parametrizar un test para especificar un ficheros que recojan los múltiples casos de prueba. De esta forma se simplifica mucho las prueba.

Comentario de revisión...se puede formatear mejor el código

```
@ParameterizedTest(name = "Run with User = \"{0}\" and
        Password = \"{1}\" must throw an exception.")
@CsvFileSource(resources = "/testConnectUserPasswordWrong.csv",
        numLinesToSkip = 1, delimiter = ';', encoding = "UTF-8")

public void testConnectUserPasswordWrong(String user, String password){
        assertThrows(RepositoryDataSourceException.class, () -> {
                repositoryDataSource.connect(user, password);},
                getErrorMsg("testConnectUserPasswordWrong",
                "Wrong user-password should throw an exception"));
        assertEquals(EnumConnectionType.NOT_CONNECTED,
                repositoryDataSource.getConnectionType(),
                getErrorMsg("testConnectUserPasswordWrong",
                "Connection type must be 'NOT_CONNECTED'"));
}
```

Calidad continua

En los últimos sprints, correspondientes a la entrega de prototipos funcionales al tutor, se definía una tarea para analizar la deuda técnica que indicaba la Codacy. Se analizaban las tareas de mejoras propuesta por Codacy según su orden de prioridad y se intentaban eliminar. De esta forma además de aprender de las revisiones automáticas de Codacy se controlaba la deuda técnica, en el prototipo evolutivo. La valoración final del proyecto ha sido la máxima (A) sin ninguna revivión de calidad de alta prioridad. Comentario de revisión...Añadir url de codacy de tu proyecto y una captura de pantalla donde aparezca la A.

Integración y despliegue continuo CI/CD

Se han utilizado los sistemas de integración continua y despliegue continuo de GitLab para controlar el correcto funcionamiento de la aplicación después de un cambio y para mejorar la calidad de las revisiones. La integración continua se ha configurado para que cada vez que se realiza un coomit en la rama principal del repositorio se compruebe que compila correctamente, que se ejecutan los test configurados en maven y se empaquete la aplicación Web en fichero war y se despligue en Heroku. Si alguna tarea falla se indica en el repositorio con un icono redondo rojo con aspa blanca. Comentario de revisión...añadir una captura de pantalla con el badge Heroku del Readme de Gitlab y explicar como se actualiza cada vez que se realiza un commit. Comentario de revisión...añadir una captura de pantalla con los check verdes en la lista de commit del repositorio de GitLab

Trabajos relacionados

6.1. Activiti-Api

> Aplicación que permite realizar un estudio del estado de un proyecto en GitHub mediante distintas métricas de evolución. Permitiendo tener una visión del ritmo de trabajo del proyecto, duración, numero de participantes y actividad.
>
> dba0010/Activiti-Api

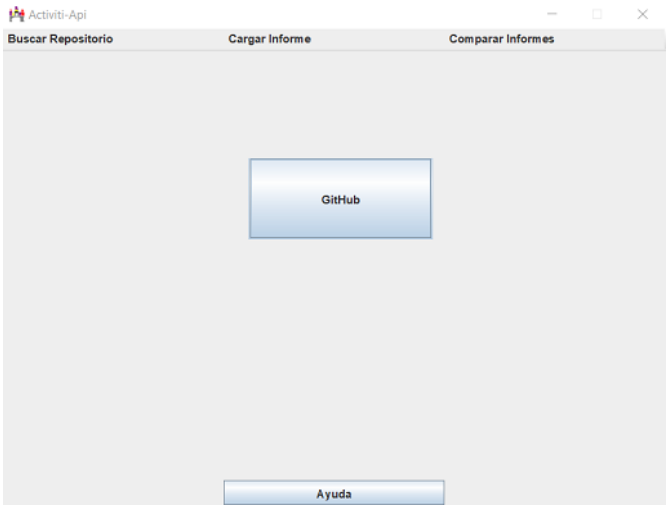

Figura 6.16: Ventana principal de Activiti-Api

En un proyecto bastante parecido implementado como aplicación de escritorio escrita en lenguaje Java. En algunos aspectos ha sido modelo para el desarrollo de este proyecto software. Esta alojado en

GitHub[27] y se puede obtener y ejecutar. Se muestra la ventana principal en la Fig. 6.16.

El aspecto que más llamó la atención para este proyecto es la forma de buscar repositorios. Permite establecer una conexión a GitHub iniciando sesión mediante usuario y contraseña o entrar en "Modo desconectado", como se muestra en la Fig. 6.17. Una vez establecida la conexión permite mostrar las métricas de evolución de un proyecto, para ello hay que indicar un usuario del cual se quiere obtener un proyecto y seleccionar el proyecto de un desplegable con los proyectos de ese usuario, ver Fig. 6.18.

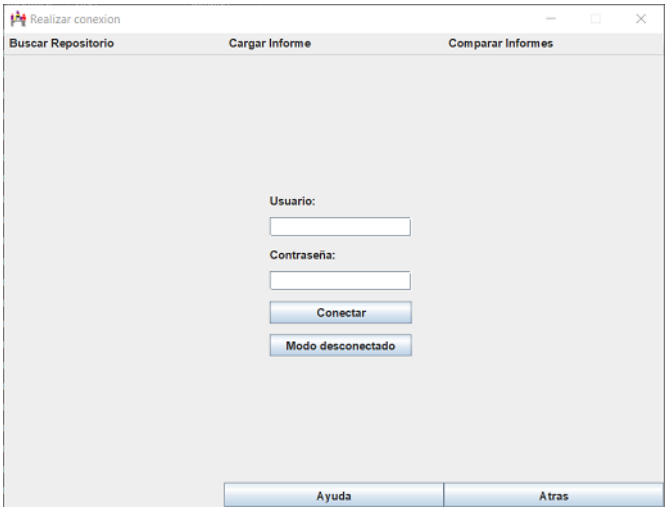

Figura 6.17: Conexión a GitHub mediante Activiti-Api

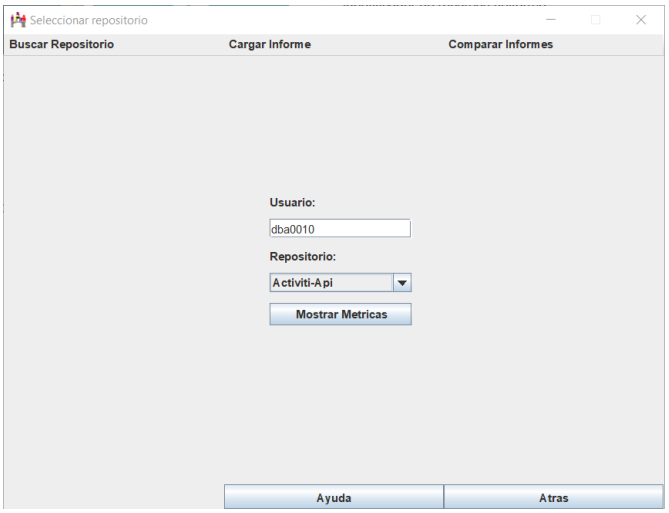

Figura 6.18: Selección de un proyecto para obtener las métericas en Activiti-Api

[27] https://github.com/dba0010/Activiti-Api

Comparando Activiti-Api con este proyecto

A continuación se muestran las diferencias del producto software de "Activiti-Api" con el producto generado en "Comparador de métricas de evolución en repositorios software".

Interfaz

Es una aplicación de escritorio, mientras que en este proyecto se ha optado por implementar una aplicación web.

Conexión

Para obtener información de los proyectos, Activiti-Api permite establecer una conexión a GitHub mientras que este proyecto permite establecer conexión a GitLab. Aunque este proyecto permite extender la funcionalidad a otras forjas de repositorios como GitHub.

Activiti-Api permite dos tipos de conexión: iniciar sesión a GitHub mediante usuario y contraseña o "Modo desconectado" (establece una conexión pública).

Este proyecto permite iniciar sesión a GitLab mediante usuario y contraseña o por uso de un token de acceso personal. Además también permite establecer una conexión pública o directamente trabajar sin conexión sobre la aplicación. Dependiendo de la conexión escogida se limitará la funcionalidad de la aplicación. Por ejemplo, sin conexión no es posible añadir repositorios. Además se muestra al usuario de la aplicación el tipo de conexión escogida en todo momento y la información de sesión iniciada en caso de que se haya iniciado sesión en GitLab.

Gestión de proyectos y evaluación de métricas

Activiti-Api solo permite trabajar con un proyecto al mismo tiempo o comparar dos proyectos. La comparación se ha definido de forma estática durante el desarrollo del proyecto, permaneciendo invariable en tiempo de ejecución.

Este proyecto permite añadir múltiples proyectos, evaluarlos y compararlos mediante el cálculo estadístico de cuartiles para hallar los valores umbrales de cada métrica. Por tanto la comparación puede ser dinámica a partir de los proyectos que se escojan para la comparativa, aunque también se han definido unos valores umbrales predefinidos a partir de unas estadísticas obtenidas de un conjunto de datos obtenidos a partir de TFGs y publicado en GitHub [28]. Este proyecto permite gestionar perfiles de métricas con diferentes umbrales para distintos contextos de aplicación.

Activiti-Api genera un informe con los resultados de las métricas de un proyecto, varios gráficos y permite generar un informe de comparativa entre dos proyectos. Mientras que este proyecto muestra los resultados de varios proyectos en forma de tabla.

Mantenibilidad y extensibilidad

Este proyecto ha preparado un framework para poder extender la fuente de datos, es decir las forjas. Ha sido implementado para obtener datos desde GitLab, pero es fácilmente extensible a otras forjas como GitHub. Activiti-Api no permite esta extensibilidad e incluye demasiadas dependencias con GitHub API.

Ambos proyectos siguen la solución basada en frameworks propuesta en *Soporte de Métricas con Independencia del Lenguaje para la Inferencia de Refactorizaciones* [10]. El objetivo del *framework* es la reutilización en la implementación del cálculo de métricas. De hecho, Activiti-Api ha servido de ejemplo para la implementación del motor de métricas de este trabajo.

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

6.2. Soporte de Métricas con Independencia del Lenguaje para la Inferencia de Refactorizaciones

De este trabajo se ha basado la construcción del subsistema "motor de métricas", como se puede ver en la sección 3.3.

6.3. Software Project Assessment in the Course of Evolution - Jacek Ratzinger

Es de este trabajo de donde se han obtenido las métricas de control con las que se trabaja en este proyecto. Hay una explicación detallada en el apartado 3.3.

Conclusiones y Líneas de trabajo futuras

7.1. Conclusiones

En este proyecto se ha aprendido a configurar una aplicación web en Java, realizar interfaces gráficas usando solo Java, configurar un proyecto de forma que se facilite la integración y despliegue continuos y la importancia que tiene para un equipo de desarrollo software, la importancia del uso de herramientas que midan la calidad de código, la importancia que tienen las métricas de proceso a la hora de gestionar un proyecto software, la comodidad y eficiencia que tienen las metodologías ágiles.

7.2. Lineas de trabajo futuras

Se definen en lista ideas que podrían realizarse en el futuro:

- Extender la funcionalidad a nuevas métricas de evolución
- Extender las plataformas de desarrollo colaborativo a otras como GitHub, Bitbucket
- GitLab ofrece la posibilidad a los usuarios de tener su propia instancia de GitLab en un servidor propio. Por ahora solo se puede conectar al host "https://gitlab.com/", se podría ampliar esta funcionalidad permitiendo realizar una conexión a servidores propios
- Realizar un histórico de mediciones y almacenarlo en una base de datos
- Hacer que la aplicación web sea adaptable (*responsive*)
- Internacionalizar la aplicación
- Los proyectos y perfiles de métricas importados y exportados se almacenan en un buffer en memoria. Mientras el proyecto sea pequeño no hay problema, pero conforme vaya creciendo habría que implementar otros sistemas de persistencia como bases de datos o ficheros.
- Se podría permitir seleccionar varios proyectos de la tabla de la página principal para poder gestionar varios proyectos a la vez. Por ejemplo: crear un perfil de métricas sólo con los proyectos seleccionados, evaluar solo los proyectos seleccionados, eliminar varios proyectos a la vez, volver a obtener métricas de varios proyectos al mismo tiempo.

- La aplicación web solo soporta una sesión, se podría preparar para poder explotarlo en un entorno multisesión.

Bibliografía

[1] Archivo:Modelo Cascada Secuencial.jpg.

[2] FunctionalInterface (Java Platform SE 8).

[3] GitLab Continuous Integration & Delivery.

[4] GitLab4j API (gitlab4j-api) provides a full featured Java client library for working with GitLab repositories via the GitLab REST API: gitlab4j/gitlab4j-api, July 2019. original-date: 2014-02-18T05:09:14Z.

[5] Iubaris Info 4 Media SL. . Scrum Manager: Temario Troncal I. Versión 2.6.1:94, January 2019.

[6] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Patrones De Diseño: Elementos De Software Orientado a Objetos Reutilizable*. Addison-Wesley, Madrid, 1 ed. en es edition, 2002.

[7] Diego Güemes-Peña, Carlos López-Nozal, Raúl Marticorena-Sánchez, and Jesús Maudes-Raedo. Emerging topics in mining software repositories. *Progress in Artificial Intelligence*, pages 1–11, January 2018.

[8] Ivar Jacobson, Grady Booch, and James Rumbaugh. *El proceso unificado de desarrollo de software*. Addison Wesley, Madrid, 2000. OCLC: 46834249.

[9] Iván Lasso. Qué es Markdown, para qué sirve y cómo usarlo, September 2013.

[10] Raúl Marticorena, Yania Crespo, and Carlos López. Soporte de Métricas con Independencia del Lenguaje para la Inferencia de Refactorizaciones. In *X Jornadas Ingenieria del Software y Bases de Datos (JISBD 2005), Granada, Spain ISBN: 84-9732-434-X*, pages 59–66, September 2005.

[11] Tim Olshansky. A wrapper for the Gitlab API written in Java. Contribute to timols/java-gitlab-api development by creating an account on GitHub, July 2019. original-date: 2013-05-19T02:48:09Z.

[12] Jacek Ratzinger. *sPACE: Software Project Assessment in the Course of Evolution*. PhD Thesis, 2007.

[13] rlp0019. Software para la autoevaluación del proceso de Trabajo de Fin de Grado (TFG) en GitHub.: rlp0019/Activiti-Api, June 2019. original-date: 2019-02-17T19:30:04Z.

[14] Ian Sommerville. *Ingeniería de software*. Pearson Education Limited, México, 6ª edition, 2002.

[15] TFGM_GII UBU. Plantilla Latex. Contribute to ubutfgm/plantillaLatex development by creating an account on GitHub, June 2019. original-date: 2016-10-14T10:17:24Z.