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 — 3 de julio
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 Çomparador de métricas de evolución en repositorios software"de TFG.

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

En Burgos, 3 de julio de 2019

Vº. Bº. del Tutor:

D. Carlos López Nozal

Resumen

Aplicación Web en Java que toma como entrada un conjunto de direcciones de repositorios públicos o privados y calcula medidas de la evolución que permiten comparar los repositorios.

Descriptores

gitlab, proyectos, repositorios, métricas, análisis, comparador, software

Abstract

Java Web Application that takes a set of addresses of public or private repositories and calculates measures of evolution that allow comparing the repositories.

Keywords

gitlab, projects, repositories, metrics, analysis, comparator, software

Índice general

Índice de figuras

Índice de tablas

Introducción

> Una suposición subyacente de la administración de la calidad es que la calidad del proceso de desarrollo afecta directamente a la calidad de los productos a entregar [6, pág 543].

La calidad del proceso es uno de los factores que determinan la calidad del producto software, tal y como expone Sommerville en *Ingeniería de software*[6].

Por esa razón, este trabajo se centra en crear un registro de medidas de proceso de proyectos software alojados en GitLab para, posteriormente, evaluarlos en relación con otros proyectos.

1.1. Estructura de la memoria

La memoria se estructura de la siguiente manera[1][7]:

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.

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

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.

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

En primer lugar se detallan requisitos generales que surgen a partir del planteamiento del problema y los objetivos que se desean conseguir con este proyecto.

- Se desea obtener 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 las siguientes:
  - 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 mínimo y máximo de cada métrica basados en el cálculo de los cuartiles Q1 y Q3, respectivamente.
- Se desea poder almacenar esta información para poder utilizarlo posteriormente.

2.2. Objetivos técnicos

Este apartado recoge los requisitos más técnicos del proyecto.

- Desarrollar la aplicación de manera que se puedan extender las métricas que se calculan con el menor coste posible.
- La aplicación se debe desarrollar de tal manera que facilite la extensión a otras plataformas de desarrollo colaborativo como GitHub o Bitbucket.
- Separar, en la medida de lo posible, la lógica de la aplicación y la interfaz de usuario.
- Crear una batería de pruebas automáticas que cubran la mayor parte del sistema desarrollado.
- 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.
- Generar documentación.
- Utilizar sistemas que aseguren la calidad de código.

Conceptos teóricos

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

3.1. Calidad de proceso y producto software

Todo proyecto de software tienen como objetivo desarrollar software de alta calidad, que cumpla con los requisitos acordados. Sin embargo, 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.

Sommerville enumera en *Software Engineering* [6] los principales factores que afectan a la calidad del producto, como se puede observar en la figura 3.1:

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

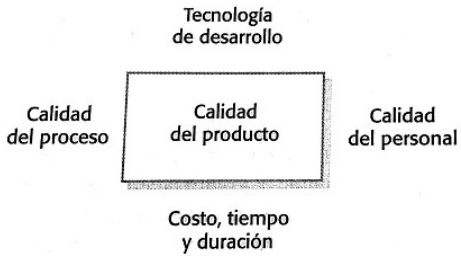

Figura 3.1: Principales factores de calidad del producto de software[6]

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, planeación y control.

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 un proceso de medición.

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 es una medida relacionada con un atributo de un proceso o producto software (por ejemplo). Las métricas son de control o de predicción. Las **métricas de control** se asocian al proceso de desarrollo del software (p. ej. Media de días ue se tarda en cerrar una incidencia)y las **métricas de predicción** se asocian a productos software (p. ej. Complejidad ciclomática de una función). Y ambos tipos de métricas influyen en la toma de decisiones administrativas como se observa en la figura 3.2.

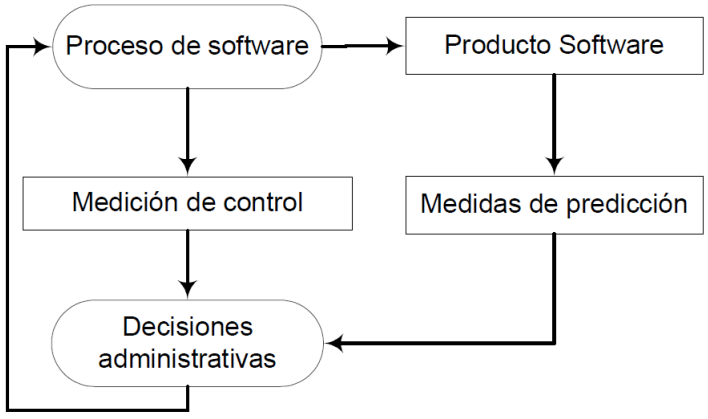

Figura 3.2: Métricas de control y métricas de predicción[6]

3.2. Evolución de software

Este proyecto se centra en las métricas de control, es decir las que están asociadas al proceso de desarrollo o de evolución del software.

Conceptos de evolución

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 sirven para llevar un control sobre la evolución de un proyecto software ya que 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 proceso que ayuden a mejorar la evolución del proyecto comparandolas con otras mediciones de otros proyectos.

Métricas de evolución o de proceso

Como se ha explicado antes, este proyecto trata de calcular métricas de proceso. Las métricas que se utilizan en este proyecto provienen una Master Tesis titulada *sPACE: Software Project Assessment in the Course of Evolution* [4]:

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 >= 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 = NTC/NTI. NTC = Número total de commits, NTI = Numero total de issues
- **Fuente de medición**: Proyecto en una plataforma de desarrollo colaborativo.
- **Interpretación**: CI >= 0, Si se acerca a 1 se definen bien las issues, si alto: no se definen bien las issues, si bajo: desarrollo del proyecto lento
- **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 = NTIC*100/NTI. NTIC = Número total de issues cerradas, NTI = Numero total de issues
- **Fuente de medición**: Proyecto en una plataforma de desarrollo colaborativo.
- **Interpretación**: 0 <= PIC <= 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 = SUM(DCI) / NTIC . NTIC = Número total de issues cerradas, DCI = Días en cerrar la issue
- **Fuente de medición**: Proyecto en una plataforma de desarrollo colaborativo.
- **Interpretación**: MDCI >= 0. Cuanto más pequeño mejor.
- **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**: MDEC = [Sumatorio de (TCi-TCj) desde i=1, j=0 hasta i=NTC] / NTC. NTC = Número total de commits, TC = Tiempo de Commit
- **Fuente de medición**: Proyecto en una plataforma de desarrollo colaborativo.
- **Interpretación**: MDEC >= 0. Cuanto más pequeño mejor.
- **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. 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 >= 0. Mejor valores altos
- **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 = NTC / 12

- **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 - Número de commits en el mes 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**: CCP = NCMP / NTC. 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 <= CCP <= 1. Mejor valores intermedios
- **Tipo de escala**: Ratio
- **Tipo de medida**: NCMP, NTC = Contador

3.3. 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* [2]. El framework cuyo diseño se muetra en la figura 3.3 es independiente del lenguaje y su objetivo es la reutilización en la implementación del cálculo de métricas.

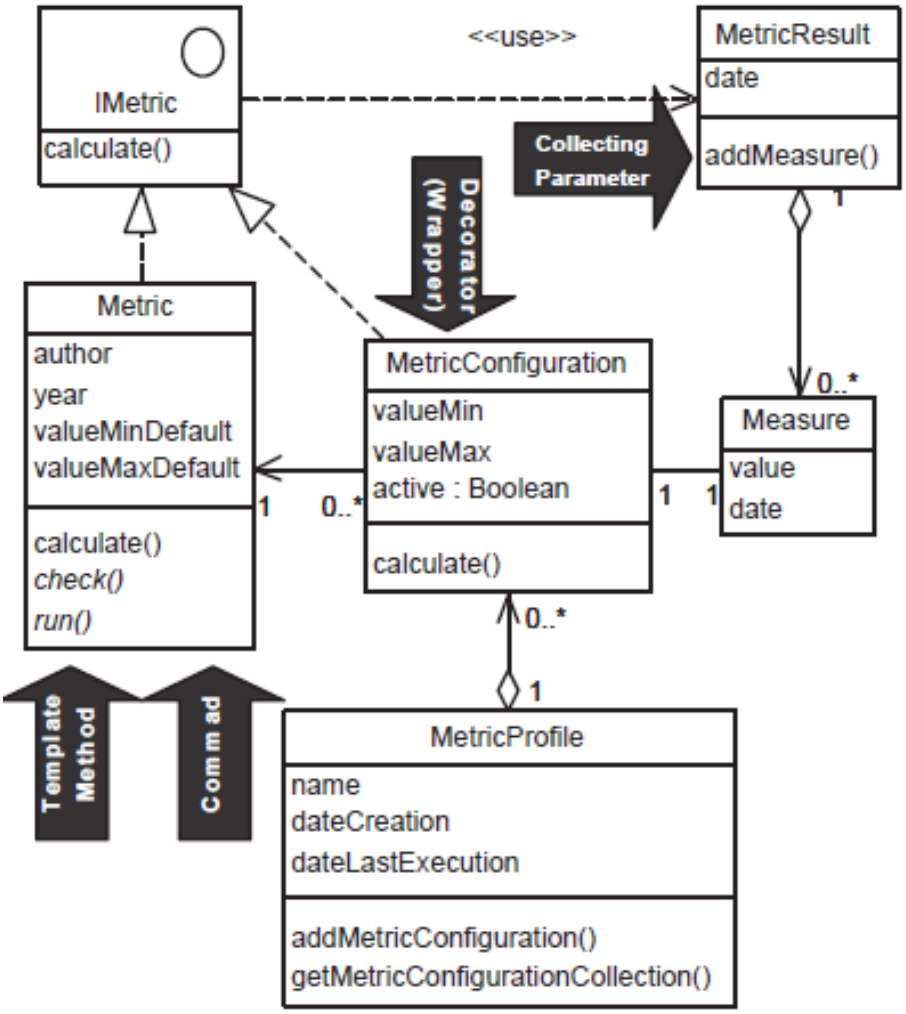

Figura 3.3: Diagrama del Framework para el cálculo de métricas con perfiles.

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

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[2] 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

[2] 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 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* [2]. Ver figura 3.3.

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 la asignatura de *Desarrollo Avanzado de Sistemas Software*, en la que se enseña como desarrollar software de calidad.

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.

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*[5], *Soporte de Métricas con Independencia del Lenguaje para la Inferencia de Refactorizaciones* [2] y *sPACE: Software Project Assessment in the Course of Evolution* [4] 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[3][3]. 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 [4][1]. 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.

[3] https://github.com/timols/java-gitlab-api
[4] https://github.com/gitlab4j/gitlab4j-api

5.4. Configuración del proyecto

Ha sido una de las etapas más complicadas del proyecto, con frecuencia aparecían problemas con herramientas que se elegían por problemas de compatibilidad con otras herramientas, escasa documentación, etc.

El proyecto se iba a desarrollar en Java desde el principio. La versión más moderna que había entonces era Java 11. Al principio generó muchos problemas porque otras herramientas no la soportaban, pero al final se consiguió

5.5. CI/CD

Se han utilizado los sistemas de integracion 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.

Trabajos relacionados

6.1. Activiti-Api

[5] 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.

Es un proyecto bastante parecido y del cual se han asentado muchas bases para este proyecto.

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

[5] Estre trabajo ha sido la base para implementar el motor de métricas de la aplicación.

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

[5] Todas as métricas utilizadas en este proyecto han surgido de las definiciones de esta tesis.

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] 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.

[2] 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.

[3] 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.

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

[5] 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.

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

[7] 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.