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 — 2 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, 2 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 [3, 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*[3].

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][5]:

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* [3] 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

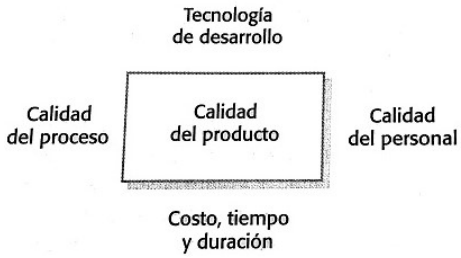

Figura 3.1: Principales factores de calidad del producto de software[3]

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.

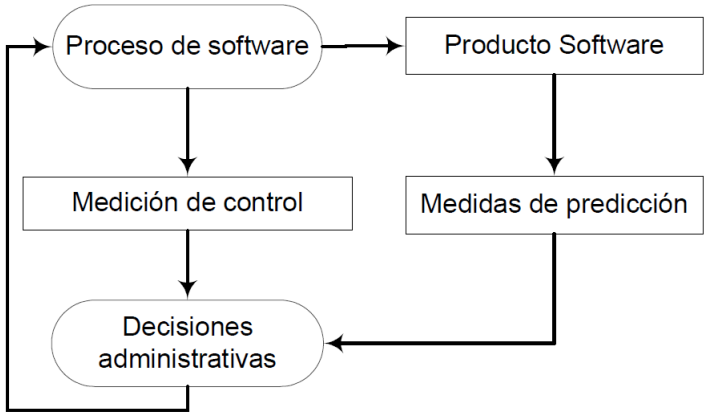

Figura 3.2: Métricas de control y métricas de predicción[3]

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

Un sistema de Gestión de Configuración del Software (SCM) es capaz de gestionar la evolución y cambios del código fuente en el tiempo [1, 4]. En este apartado describiremos los conceptos básicos que utilizaremos para recoger, documentar, almacenar y recuperar los diferentes cambios que se produzcan en las entidades que formen parte nuestro sistema. Un proyecto software está compuesto por múltiples ficheros y directorios. Llamaremos ítem a cualquier fichero o directorio cuya evolución en el tiempo está controlada por un sistema de Gestión de Configuración del Software (SCM). Una vez que estén bajo control, será posible ver su historial de cambios o revisiones y recuperar un estado anterior. A medida que los ítems evolucionan en el tiempo, se van creando nuevas revisiones de los mismos. Como es lógico, algunos ítems sufrirán más cambios que otros a lo largo del desarrollo. El SCM almacena todas las revisiones de cada ítem, de manera que es posible volver al estado de uno de estos elementos en un momento dado. La creación de revisiones ocurre a través de las operaciones de desprotección (check-out) y protección (check-in). Cuando se va a hacer una modificación en un ítem, éste se desprotege para editarlo. Para crear una nueva revisión del ítem de forma que se pueda recuperar su estado en este punto, se protege, lo que indica al SCM que debe almacenar los nuevos contenidos del ítem. Al conjunto de revisiones de un ítem se le denomina historia y resume la evolución de ese ítem en el tiempo. Una rama es un contenedor de revisiones, capaz de almacenar la evolución de los ítems como se muestra en la Ilustración 1. Las ramas pueden contener revisiones de más de un ítem. De hecho, esta es la situación más habitual. Ilustración 1: Concepto de rama

Una etiqueta o label es el modo de marcar revisiones para poder agruparlas según un cierto criterio, que normalmente fija el usuario. Cuando se aplica una etiqueta, se crea una instantánea de la situación de los ítems en el tiempo. Más tarde esa instantánea puede ser referenciada con facilidad para identificar ese momento específico. Una etiqueta es, en definitiva, un nombre más fácil de recordar que se asigna a un conjunto particular de revisiones. Las etiquetas se aplican siempre a las revisiones de los ítems que se encuentren actualmente en el espacio de trabajo o workspace, que es la zona donde el SCM puede mantener ítems bajo el control de versiones. A

efectos prácticos, el workspace no dejará de ser un directorio en el disco. Los ítems, sus revisiones, las ramas donde se almacenan las revisiones y las etiquetas que agrupan las revisiones se almacenan en un repositorio, que será el espacio principal donde el SCM guarda todos los objetos.

Métricas de evolución

Las métricas que se calculan de los proyectos son un conjunto de métricas que proceden de la Master Tesis titulada *sPACE: Software Project Assessment in the Course of Evolution* [2].

I1 - Número total de issues

- **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**: Repositorio de un gestor de repositorios
- **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 por issue

- **Categoría**: Proceso de Orientación
- **Descripción**: Número de commits por issue
- **Propósito**: ¿Cuántos commits realizados por cada issue?
- **Fórmula**: CI = NTC/NTI. NTI = Numero total de issues, NTC = Número total de commits
- **Fuente de medición**: Repositorio de un gestor de repositorios
- **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**: NTI, NTC = 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**: Repositorio de un gestor de repositorios
- **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**: Repositorio de un gestor de repositorios
- **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**: ¿Cúanto tiempo suele 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**: Repositorio de un gestor de repositorios
- **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. TC2 = Tiempo de último commit, TC1 = Tiempo de primer commit.
- **Fuente de medición**: Repositorio de un gestor de repositorios
- **Interpretación**: DEPUC >= 0
- **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**: Repositorio de un gestor de repositorios
- **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**: Repositorio de un gestor de repositorios
- **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 [8]. Este framework

(3.3) es independiente del lenguaje y su objetivo es la reutilización en la implementación del cálculo de métricas.

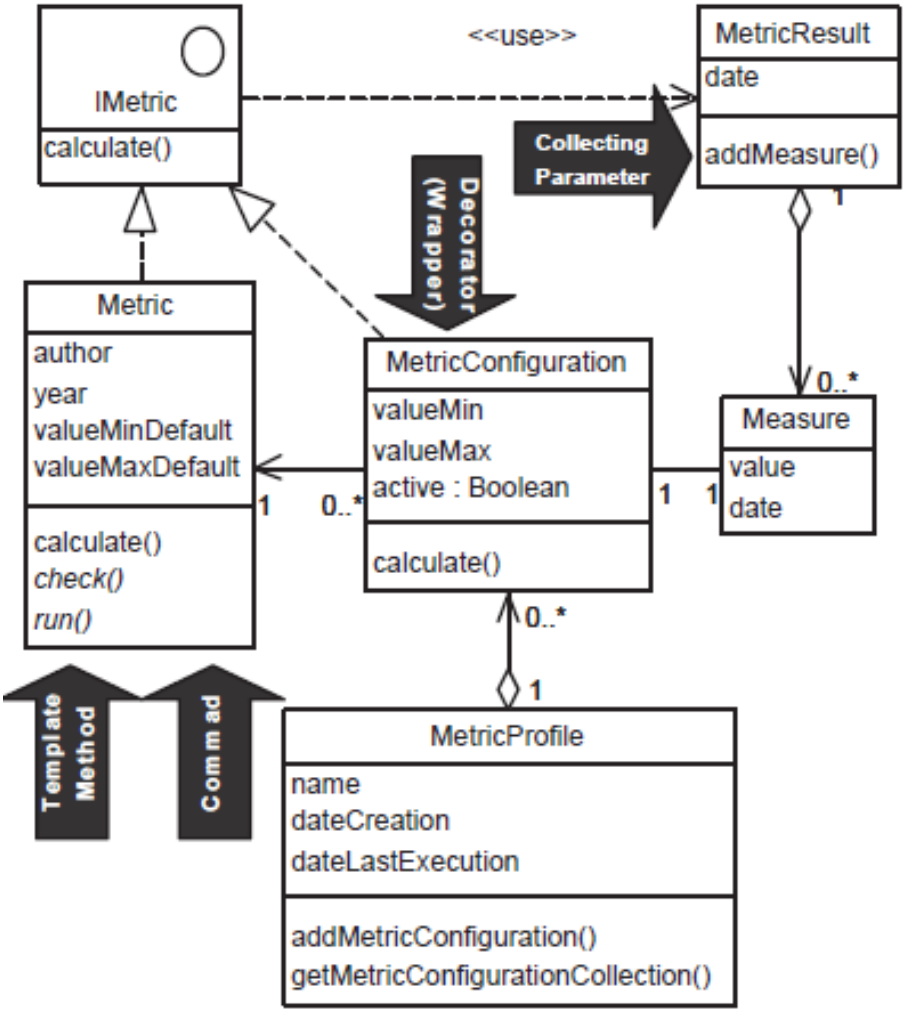

Figura 3.3: Diagrama del Framework para el cálculo de métricas con perfiles.

Técnicas y herramientas

4.1. Herramientas utilizadas

- Eclipse Java EE IDE for Web Developers. Version: 2018-09 (4.9.0)
- Apache Maven v3.6.0
- Apache Tomcat v9.0.13
- Java SE 11 (JDK)
- gitlab4j-api v4.8.56
- JUnit 5 v5.3.1

Aspectos relevantes del desarrollo del proyecto

Trabajos relacionados

- Activiti-Api
- Soporte de Métricas con Independencia del Lenguaje para la Inferencia de Refactorizaciones
- Software Project Assessment in the Course of Evolution - Jacek Ratzinger

Conclusiones y Líneas de trabajo futuras

Bibliografía

[1] Steve Berczuk and Brad Appleton. *Software Configuration Management Patterns: Effective Teamwork, Practical Integration*. ADDISON WESLEY LONGMAN INC DIV PEARSON SUITE 300, Boston, edición: 01 edition, November 2002.

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

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

[4] I. Sommerville. *Software Engineering, Global Edition*. Pearson Education Limited, 2016.

[5] 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.