UNIVERSIDAD TÉCNICA PARTICULAR DE LOJA

La universidad católica de Loja

ÁREA TÉCNICA

TITULO DE INGENIERO EN SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

Desarrollo de un plugin de recolección de datos sobre el comportamiento de estudiantes de programación para Netbeans.

TRABAJO DE TITULACIÓN.

AUTOR: Cárdenas Cabrera, Ronald Francisco

DIRECTOR: López Vargas, Jorge Afranio, Ing.

LOJA – ECUADOR

2017

Septiembre, 2017

APROBACIÓN DEL DIRECTOR DEL TRABAJO DE TITULACIÓN

Ing.

Jorge Afranio López Vargas

DOCENTE DE LA TITULACIÓN

De mi consideración:

El presente trabajo de titulación: Desarrollo de un plugin de recolección de datos sobre el comportamiento de estudiantes de programación para Netbeans realizado por Cárdenas Cabrera Ronald Francisco, ha sido orientado y revisado durante su ejecución, por cuanto se aprueba la presentación del mismo.

Loja, febrero 2017.

f).....

Ing. Jorge Afranio López Vargas

DECLARACIÓN DE AUTORÍA Y CESIÓN DE DERECHOS.

"Yo Cárdenas Cabrera Ronald Francisco declaro ser autor del presente trabajo de titulación: **Desarrollo de un plugin de recolección de datos sobre el comportamiento de estudiantes de programación para Netbeans**, de la Titulación Ingeniería en Sistemas Informáticos y Computación, siendo Jorge Afranio López Vargas director del presente trabajo; y eximo expresamente a la Universidad Técnica Particular de Loja y a sus representantes legales de posibles reclamos o acciones legales. Además, certifico que las ideas, conceptos, procedimientos y resultados vertidos en el presente trabajo investigativo, son de mi exclusiva responsabilidad.

Adicionalmente declaro conocer y aceptar la disposición del Art. 88 del Estatuto Orgánico de la Universidad Técnica Particular de Loja que en su parte pertinente textualmente dice: "Forman parte del patrimonio de la Universidad la propiedad intelectual de investigaciones, trabajos científicos o técnicos y tesis de grado o trabajos de titulación que se realicen con el apoyo financiero, académico o institucional (operativo) de la Universidad"

f)……
Autor: Cárdenas Cabrera Ronald Francisco
Cedula. 1105550295

DEDICATORIA

Este trabajo está dedicado:

- A mis padres especialmente a mi madre, quien ha sido pilar fundamental de mi vida, guiándome por el camino correcto en mi formación personal, llenándome de valores para ser de mi un hombre de bien.
- A mis abuelitos que con su sabiduría me han brindado buenos consejos que han definido lo que hoy soy como persona, gracias por su apoyo incondicional, por sus oraciones y por sus buenas enseñanzas.
- A mis tíos que me han apoyado en todo sentido para alcanzar una meta importante de mi vida.
- A mis hermanas por sus palabras motivadores a la hora de afrontar este reto, les agradezco por estar a mi lado apoyándome.

Y a todas aquellas personas que de algún modo pusieron su granito para ser parte de lo que hoy soy como persona.

Ronald Cárdenas

AGRADECIMIENTO

Primeramente le doy gracias a Dios por haberme dado salud y fuerza para afrontar los retos de la vida. Estoy agradecido con la universidad y los docentes quienes a través de sus enseñanzas me han formado académicamente para ser útil a la sociedad.

A todas a las personas, amigos, compañeros y profesores que formaron parte de mi formación en la universidad, por creer y ser parte de la consecución de un logro importante de mi vida.

Ronald Cárdenas

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

El presente trabajo describe el diseño e implementación de una herramienta para la recolección de datos acerca del comportamiento de estudiantes de programación, el diseño y el conjunto de datos que la herramienta captura se ha formado a partir de una investigación sobre trabajos relacionados con el análisis del aprendizaje en cursos de programación y una revisión de las métricas de software existentes.

La herramienta permite recolectar datos de forma automática y transparente de las actividades en tareas de programación, conforme el alumno codifica, construye y ejecuta un programa. Se utiliza técnicas como análisis estático de código, transformaciones a nivel de bytecode, algoritmos de diferencia de código, sincronización de datos, seguimientos de eventos del sistema de archivos y visualizaciones estadísticas. Además se ha desarrollado tomando en cuenta conceptos de extensibilidad, para dar soporte a nuevos lenguajes de programación y nuevas métricas, minimizando las dependencias sobre los entornos de desarrollo integrado.

Palabras clave: Java, IDEs, Sincronización de datos, Recolección de datos para Learning analytics, Rest Service, Web Development

ABSTRACT.

The present work describes the design and implementation of a tool to collect data about the behavior of programming students, the design and the data set that the capture tool has been formed from research on works related to the analysis of the Learning in programming courses and a review of existing software metrics.

The tool allows automatic and transparent data collection of activities in programming tasks, as the student codifies, builds and executes a program. It uses techniques such as static code analysis, bytecode level transformations, code difference algorithms, data synchronization, file system event tracking, and statistical visualizations. It has also been developed taking into account concepts of extensibility, to support new programming languages and new metrics, minimizing dependencies on integrated development environments.

Keywords: Java, IDEs, Data Synchronization, Data Collection for Learning analytic, Rest Service, Web Development

"Lo que no se define no se puede medir lo que no se mide no se puede mejorar. Lo que no se mejora se degrada siempre."

(Lord Kelvin, 1885)

INTRODUCCIÓN.

Desde tiempos inmemoriales el hombre ha sentido la necesidad de medir, la razón de este acto radica en la búsqueda de expandir su conocimiento sobre su alrededor, su situación y las oportunidades de mejora. Este acto de cuantificar atributos ha ido evolucionando a lo largo del tiempo hasta formar y presentar métricas específicas en una cada de las áreas en donde se desenvuelve el ser humano.

Hasta la fecha se han propuesto una gran cantidad de métricas para evaluar la calidad, predecir eventos, crear estimaciones… Tanto en el área del desarrollo software y del análisis del aprendizaje; la información que estas aportan puede ser de utilidad para aplicar procesos de análisis por parte de las instituciones educativas, especialmente en la educación virtual en donde es difícil hacer un seguimiento y ofrecer soluciones que se apeguen a la realidad del estudiante. De esto se habla en mayor detalle en el capítulo II.

Este trabajo de titulación contribuye con un estudio, selección e implementación de las métricas de software y/o medidas de análisis del aprendizaje, en una herramienta para la recolección de la información que estas puedan aportar, sobre las actividades de los estudiantes de programación al resolver sus tareas. Se espera que este tipo de herramientas sean de utilidad en el ámbito educativo especialmente para la universidad, permitiendo que la misma pueda llevar a cabo investigaciones futuras en el área de análisis del aprendizaje y también de utilidad para el estudiante proporcionando retroalimentación a través de gráficos estadísticos sobre sus actividades en tareas de programación.

El principal entregable del proyecto es un plugin para Netbeans, en el cual se implementan las métricas identificadas que proporcionan información útil. Los datos recolectados por este plugin serán almacenados de forma remota y local, el almacenamiento remoto permitirá a los docentes contar con información real generada a partir de las acciones del estudiante

Para el éxito del proyecto se debe cumplir con los siguientes objetivos:

- Seleccionar las métricas de software y/o medidas de análisis del aprendizaje que se pueden aplicar para obtener información que permitan mejorar el proceso de enseñanza/aprendizaje de materias de programación.

- Implementar visualizaciones de la información local, para el estudiante que le ayuden a desarrollar las competencias que necesita.
- Desarrollar un plugin para NetBeans que permita recolectar información sobre el comportamiento de los estudiantes de la UTPL en el desarrollo de programas Java.
- Definir un modelo de datos que permita almacenar información de manera local y remota y que garantice la consistencia entre ambos modelos.

La estructura de este trabajo se describe a continuación:

Capítulo I: En este capítulo se presenta al lector el contexto del proyecto, se habla sobre el alcance, los objetivos y los resultados en la ejecución de este trabajo.

Capítulo II: En este capítulo se profundiza en la problemática, se presenta los avances realizados hasta la fecha sobre: herramientas de recolección de datos de programación, investigaciones relacionadas con el análisis del aprendizaje en cursos de programación, con la finalidad de: entender, identificar y seleccionar los datos que podrían ser útiles para este tipo de investigaciones.

Capítulo III: Se presenta los datos a recolectar en las siguientes áreas de interés: análisis del aprendizaje, métricas de código y datos de comportamiento.

Capítulo IV: Se presenta el desarrollo de la aplicación, utilizando el proceso de desarrollo para describir cada una de las etapas críticas en el desarrollo de la solución, comprende desde el análisis hasta las pruebas y despliegue de los componentes.

Secciones Se presenta las conclusiones de este trabajo, las recomendaciones para proyectos similares y el trabajo a futuro para dar continuidad a este proyecto.

Anexos: Apartado en el cual se organizan los materiales y recursos útiles en la exposición y desarrollo de este trabajo.

CAPÍTULO I
CONTEXTO DEL PROYECTO

1.1. Contexto.

En la educación moderna la gran cantidad de datos que se generan producto de la interacción de los estudiantes con los entornos de aprendizaje virtual ha sido el principal desencadenante en el desarrollo de nuevas áreas de investigación: el análisis de aprendizaje y la minería de datos educativa son áreas de estudio que han desarrollo y propuesto métricas para evaluar y dar seguimiento a los estudiantes con el objetivo de mejorar la calidad de la educación. Por otro lado, la ingeniería de software conforme ha ido evolucionando ha desarrollado métricas para evaluar: la calidad de los programas, la productividad de los programadores, los procesos y la gestión de proyectos.

Nuevas técnicas y métodos de recolección y análisis de datos podrían ayudar a revelar nuevos caminos en el aprendizaje de los estudiantes, ofreciendo la posibilidad de proporcionar retroalimentación en tiempo real para que los profesionales adapten sus enseñanzas (Blikstein et al., 2014, p. 2). Es aquí donde adquiere importancia este trabajo, en el cual busca desarrollar una herramienta para recolectar la información que las métricas en estas áreas puedan aportan, con el fin de que en proyectos futuros se cuente herramientas que faciliten la recolección de estos datos.

Se debe tener claro que en la practica la programación es llevada a cabo utilizando ambientes de desarrollo integrado (IDEs) o mediante editores simple de texto ya sean estos de escritorio o basados en ambientes Web. Las prácticas, ejercicios y tareas se efectúan interactuando con estas herramientas, este tipo de interacciones genera datos que no son visibles a simple vista como: la cantidad de tiempo que un estudiante invierte en sus tareas, los problemas encontrados, el tiempo dedicado para practicar y adquirir habilidades de programación, la calidad de las soluciones etc.

Estos datos aún no han sido aprovechados por la UTPL, debido a que no dispone de herramientas que ayuden a capturar estos datos de forma transparente, las existentes solamente ofrecen plugins dejando la data en servidores remotos, dificultando el acceso a los datos para aplicar procesos de análisis.

Al no disponer de estos datos es complejo de dar respuesta a interrogantes que ayuden a comprender el proceso de aprendizaje de quienes inician en la programación:

- ¿Cuáles son los errores o problemas comunes que se presentan a quienes inician en esta área?
- ¿Cuáles son los hábitos de un buen programador?
- ¿Cuánto tiempo están dedicando los estudiantes a programar?

- ¿La calidad de las soluciones (programas) está acorde al nivel de estudio?
- ¿El tiempo dedicado por los estudiantes de programación cumple los requisitos mínimos solicitados por los componentes para adquirir las habilidades?
- ¿Los problemas de compilaciones y ejecuciones captados durante el ciclo de programación están correlacionados con el rendimiento del alumno?

1.2. Justificación.

Investigaciones actuales están encaminadas al análisis e identificación de patrones de comportamiento de las personas a partir de grandes volúmenes de datos resultantes de sus actividades en el día a día (Gurrin, Smeaton, & Doherty, 2014).

La programación de computadoras es una actividad humana que no está exenta a procesos de análisis; sin embargo, este tipo de actividades requiere de datos históricos y de herramientas que faciliten la recolección de los mismos. Este trabajo se enfoca en el desarrollo de un plugin para la recolección de estos datos, el cual se lo da a conocer bajo el nombre de “CODELOGS”.

1.3. Objetivos.

1.3.1. Objetivo General

- Seleccionar las métricas de software que se pueden aplicar para obtener información que permitan mejorar el proceso de enseñanza/aprendizaje de materias de programación.

1.3.2. Objetivos Específicos

- Desarrollar un plugin en Netbeans que permita recolectar información sobre el comportamiento de los estudiantes de la UTPL al desarrollo de programas en Java.
- Definir un modelo de datos que permita almacenar información de manera local y remota y que garantice la consistencia entre ambos modelos.
- Implementar visualizaciones de la información local para el estudiante que le ayuden a desarrollar las competencias que necesita.

1.4. Resultados esperados

Con el desarrollo de este trabajo se espera conseguir los siguientes resultados:

- Aplicación independiente del entorno de desarrollo (IDE), la funcionalidad como la captación de actividad del estudiante al momento de programar podrá ser ejecutada sin importar el IDE usado.
- Aplicación de métricas de código para evaluar la calidad de los programas desarrollados los estudiantes.
- Servidor de datos utilizando servicios REST para consumo y almacenamiento de datos.
- Aplicación de algoritmos de sincronización de datos, para evitar que los datos recolectados se pierdan ante una falla de conexión a internet.
- Un medio de almacenamiento de datos recolectados de forma estructurada tanto local como remota

CAPITULO II
ESTADO DEL ARTE

2.1. Introducción.

Investigaciones relacionadas con el análisis del aprendizaje en cursos de programación parten del supuesto problema de las “elevadas tasas de fallo y deserción estudiantil en estos componentes educativos”; sin embargo, no proporcionan información detallada sobre este problema, por esta razón en este capítulo se presenta esta problemática en mayor detalle, proporcionando al lector datos estadísticos que ayuden a comprender e identificar de mejor manera la utilidad de los datos a recolectar, seguido de esto se profundiza en el estudio de las investigaciones publicadas que intentan dar solución a la problemática.

En este capítulo se presenta un estudio de trabajos relacionados con el análisis del aprendizaje en cursos de programación, herramientas similares y una revisión sobre las métricas de software, la finalidad es la dar respuesta a las siguientes interrogantes: ¿Qué tipo datos son útiles?, ¿Existe algún conjunto métricas predefinido que aporte con datos relevantes en esta área?, ¿Existen herramientas o librerías que faciliten esta captura de estos datos?

2.2. Deserción estudiantil en cursos de programación.

La investigación más reciente sobre este problema fue llevada por (Watson & Li, 2014), en ella los investigadores parten de un conjunto de información relacionada con las tasas de abandono y deserción en cursos introductorios de programación de quince países. Los resultados relevan que existe una media 33.3% de estudiantes que fracasan o abandonan estos componentes educativos, para los autores este valor no es alarmante; sin embargo, se debe tener claro que este es una media y que depende de otros factores: el país, el idioma, la cantidad de alumnos por clase etc.

Para nuestra realidad (Ecuador) aún no se ha publicado una investigación seria sobre el fracaso estudiantil en estos componentes; no obstante, se puede formar una idea de nuestra condición tomando como referencia un país vecino. A nivel de Latinoamérica el único país tomado en cuenta en la investigación citada fue Brasil, el cual tiene un porcentaje de aprobación estudiantil del 45%, es decir menos de la mitad aprueban los cursos de introductorios de programación.(Watson & Li, 2014) también exponen que el número de alumnos por clase influye en este problema, en la Tabla 1 se muestra un fragmento de los resultados de la investigación.

Tabla 1 Fragmento de resultados éxito estudiantiles por clase

Tipo	**Estudio Watson & Christopher**		**Estudio Bennedsen**	
	Cursos	% existo	Cursos	% éxito
Universidad	145	66.4%	50	66%
Clases pequeñas	10	80.1%	15	82%
Clases de gran tamaño	91	65.4%	48	69%

Recuperado de : (Watson, Li, & Godwin, 2013)

Tanto la investigación llevada por (Bennedsen & Caspersen, 2007) y la investigación Watson & Christopher presentan resultados similares, llegando a la conclusión de que las clases con gran tamaño tienen un menor porcentaje de estudiantes exitosos en cuanto a la aprobación del componente. En la educación virtual la cantidad de alumnos por componente puede llegar a ser mayor que en la modalidad presencial, lo cual dificulta el dar seguimiento al grupo de estudiantes.

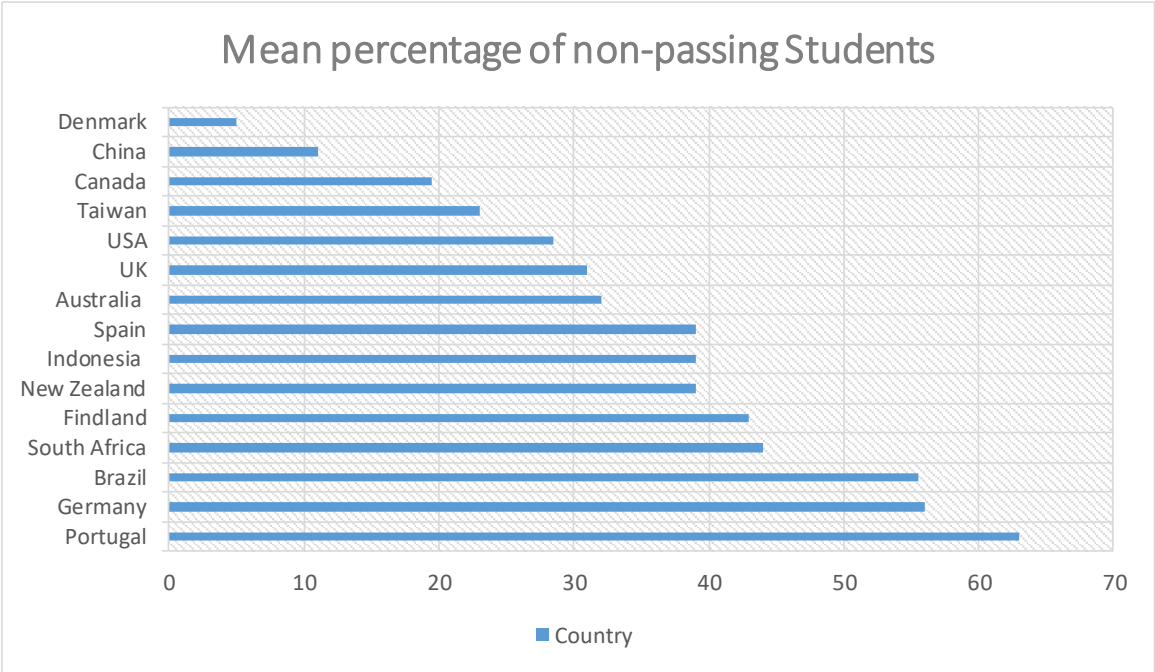


Figura 1 Promedio de estudiantes que fracasan.
Fuente : (Watson & Li, 2014)

El promedio de estudiantes que fracasan en los componentes de programación se muestra en la Figura 1, en el contexto latinoamericano Brasil es el tercer país con mayor cantidad de estudiantes que reprueban, se debe tener claro que el estudio se basó en un limitado conjunto de datos y no puede ser generalizado.

Sobre el tipo de datos a capturar la selección de estos es una etapa crítica, ya que según (Pérez, 2015) responsable de e-Learning de Gradiant - Centro Tecnológico de

Telecomunicaciones de Galicia, lo más importante es la identificación de un conjunto mínimo de información, debido a que esta debe ser de utilidad al profesor para permitirle tener una visión global del alumno y no darle un exceso de datos que lo lleve a la "parálisis por análisis".

2.3. Trabajo relacionados.

Una vez expuesto el problema, en esta sección se habla acerca de los trabajos en los cuales se han identificado y planteado métricas que según los autores podrían proporcionar utilidad en LA, esta revisión ayudara a comprender cuanto se ha avanzado permitiendo identificar los datos que formaría parte del conjunto de recolección, así como las características que la herramienta debería tener.

2.3.1. Análisis del aprendizaje.

La analítica del aprendizaje (LA) es definida por (Gros Salvat, 2013) como: “La interpretación de un amplio rango de datos producidos y recogidos acerca de los estudiantes para orientar su progresión académica, predecir actuaciones futuras e identificar elementos problemáticos” (2013, p. 7). Esta área se ha desarrollado rápidamente apoyándose de otras disciplinas, enfocándose en aprovechar la tecnología y los datos para mejorar áreas dentro de la educación.

Esta rama de investigación se enfoca en tres aspectos de la educación: aprendizaje personalizado, aprendizaje adaptivo, intervención educativa y se basa en dos actividades fundamentales, la recolección y el análisis de los datos resultantes de las acciones de los estudiantes.

Se debe tener en cuenta temas de ética y privacidad, en la actualidad estos temas no están bien definidos ya que dependen del contexto y tipo de datos a recolectar, algunos autores recalcan que es importante educar a los estudiantes para que estén convencidos de que este tipo de investigaciones busca mejorar su aprendizaje sin inmiscuirse en su intimidad (Drachsler et al., 2015, p. 3).

2.3.2. Análisis del aprendizaje en cursos de programación.

La universidad de Stanford a través de su laboratorio de investigación de tecnologías del aprendizaje ha publicado varios proyectos sobre esta área. La investigación más reciente fue llevada a cabo por (Blikstein et al., 2014) la cual tuvo por objeto de estudio la predicción del rendimiento de los alumnos e identificación de patrones en el aprendizaje. Los datos que se recolectaron en la investigación se resumen en la Tabla 2, estos datos fueron capturados conforme los estudiantes codifican la solución a los problemas planteados por el docente,

por cada estudiante se mantuvo un registro del código (capturas completas / versiones de código) que se almacenaban cuando los eventos de guardar o compilar ocurrían.

Tabla 2: Medida de actualización diferencial.

Medida actualización diferencial		
Mediciones	Nombre dado por autor	Detalles
Líneas Añadidas	Actualización diferencial	Para recolectar estos datos se enviaban capturas completas de código (varias versiones) a un servidor tomando como disparador la acción de guardar o compilar.
Líneas Eliminadas		
Líneas Modificadas		
Caracteres Añadidos		
Caracteres Modificados		
Caracteres Eliminados		

Fuente: El Autor.
Elaboración El autor.

Tabla 2 proporciona una visión general de los datos útiles en la investigación, estos pueden resumirse en el estudio de la métrica de líneas de código (LOC). Los autores parten del supuesto de que “Pequeñas y frecuentes actualizaciones pueden representar el estado del alumno como: “corrigiendo código”, en cambio grandes y menos frecuentes modificaciones pueden representar el estado de “planificación de la solución al problema”. Estos datos pueden ser muy útiles en especial si se conoce los errores o causas que hagan que esas modificaciones se lleven a cabo.

Anteriormente Blikstein ya había trabajado en proyectos similares, en el 2011 realizo una investigación con el fin de evaluar el comportamiento de los estudiantes en tareas de programación, partiendo de un conjunto de datos acerca de los programas en los cuales se contemplaba:

- El tamaño de programa.
- Numero de errores.
- Frecuencia de compilaciones correctas/incorrectas.
- Análisis de capturas de código.

En su estudio se puede apreciar que cada estudiante tiene su estilo y su forma particular de resolver los problemas de programación, (Blikstein, 2011) expone dos ventajas principales de contar con este tipo de información.

- Facilitar material de estudio que se adapte a los estilos y perfiles de los estudiantes,

de esta forma los principiantes podrían beneficiarse de contar con material con ejemplos que se adapten fácilmente a sus necesidades.

- Conocer las temáticas en la que mayor grado de dificultad se les presentan a los alumnos.

Los autores también exponen que es preferible el proceso de construcción de un programa en lugar del producto final. Aunque no exista un proceso en el desarrollo de software formal en los estudiantes, este tipo de datos brindaría información útil para los docentes.

En estos proyectos se han propuesto algunas métricas que pueden aportar con información relevante, existen otras métricas propias de la ingeniería del software que pueden ser empleadas para proporcionar datos significativos acerca de los programas desarrollados por los alumnos, en las secciones posteriores se presentara un breve estudio sobre este tema "métricas de software", de forma que permitan formar una base sólida para la selección del conjunto de métricas a implementar en la herramienta.

2.3.3. Estudio del comportamiento.

Siguiendo la misma línea de investigación, se han tomado como base de estudio el comportamiento de las compilaciones, a partir de esto se han desarrollado algoritmos que hacen uso de datos como : si un programa compilo o no, la investigación más significativa y que ha servido de base para el desarrollo de nuevas investigaciones y variantes en cuanto a algoritmos de este tipo es la de (Matthew C., 2006), en ella el autor propone un algoritmo denominado "Error quotient", el cual toma como entradas únicamente los resultados de las compilaciones.

Investigaciones realizadas en el año 2015 toman en consideración otros aspectos como errores de tiempo de ejecución, así como los tipos de errores. La aplicabilidad de estos algoritmos va desde predictores de rendimiento hasta el estudio del comportamiento de los estudiantes.

- **Error quotient (EQ).**

Este fue el primer algoritmo de este tipo desarrollo por (Matthew C., 2006), el autor propone utilizar los resultados de las compilaciones de un programa para cuantificar este proceso, según el autor existe una correlación significativa entre el valor EQ y las notas de grado.

EQ es una medida de rendimiento que ayuda a determinar cuan eficiente es un estudiante para encontrar y resolver los problemas de sintaxis encontrados en sus programas. El valor EQ de uno significa que el estudiante todo el tiempo se ha encontrado con este error y que no ha podido resolverlo. En el 2015 (Matthew C & Dorn, 2015) presentan una revisión de este algoritmo con conjunto de datos producido por las actividades de programación de

27.698 usuarios, en esta nueva investigación los autores concluyen que EQ puede todavía servir como medida alternativa de rendimiento.

- **Watwin Score**

Este algoritmo toma en cuenta otros factores que EQ no tiene en consideración como: el tiempo entre compilación, la línea de error y generalización de errores. Según los autores este algoritmo presenta mejores resultados que EQ.

- **Tasa de confusión.**

Este es un modelo de aprendizaje automático propuesto por (Lee, Rodrigo, Baker, Sugay, & Coronel, 2011). Este modelo permite juzgar si un conjunto de datos procedentes de compilación representan la confusión del estudiante. El modelo juzga con 1 en caso de un subconjunto (clip) muestre confusión y con 0 indicando que ningún estudiante del conjunto (clip) ha mostrado confusión.

2.3.4. Utilidad de las métricas del software en la educación.

Las métricas de software son usadas para medir la calidad del software desarrollado de forma profesional; aunque, estas también han sido aplicadas en el campo educativo. (Cardell-Oliver, 2011) trata de dar respuesta a la interrogante de: "¿Cómo pueden las métricas de software ayudar a programadores principiantes?", la investigadora señala que las métricas tienen un papel importante en el desempeño formativo y de diagnóstico de los aprendices.

En la investigación se aplica las métricas para determinar la calidad de los programas desarrollados por los estudiantes, para de esta forma proporcionar retroalimentación y mejorar las habilidades de programación. El conjunto de métricas que la investigadora selecciono se resume en la siguiente tabla:

Tabla 3 Métricas utilizadas en la investigación ¿Cómo pueden las métricas de software ayudar a programadores principiantes?

Métricas empleadas en la investigación		
Clasificación métrica	Métrica	Detalles-herramientas
Estilizadas	Convenciones de código	Uso de herramientas PMD CheckSyle
	Estilo de código	
Calidad	Pruebas	Junit
Métricas del producto Estáticas	Longitud del código	Análisis captura del código
	Numero de campos	Análisis captura del código
Métricas del producto dinámicas	Errores ejecución	
	Errores compilación	

Fuente: (Cardell-Oliver, 2011)
Elaborado por: Autor.

Las métricas de la Tabla 3, fueron empleadas para proporcionar retroalimentación al estudiante, la autora expone que las ventajas de estas son:

- Identificación de las dificultades individuales de los estudiantes en su aprendizaje.
- Retroalimentación para los estudiantes para mejorar el nivel de compresión.
- Medir si los estudiantes han alcanzado los niveles requeridos.

2.4. Métricas del software.

En este subapartado se busca exponer un estudio de las métricas más populares, especialmente de aquellas mencionadas en las investigaciones citadas. (Sommerville, 2012) define a una métrica como "Una característica del software, documentación de sistema o proceso de desarrollo que puede medirse de manera objetiva" (2012, p. 668) el mismo autor manifiesta que estas pueden clasificarse en métricas de control y métricas de predicción.

La importancia de estas se resume en cuatro aspectos: caracterizar, mejorar, evaluar y predecir, dependiendo del contexto en el cual sean aplicadas pueden ser catalogadas en distintas áreas. Somerville clasifica las métricas en dos áreas: predicción y control; pese a que esta es una clasificación general, es una clasificación valida debido a que resumen las principales áreas en las cuales estas han sido aplicadas.

Durante las últimas décadas investigadores, ingenieros de software y programadores han puesto como objetivo obtener software de calidad, para lograr esto se han propuesto métricas para cada área de estudio; existen métricas que permiten realizar análisis desde la complejidad y el diseño de un programa, hasta métricas tan simples como el tamaño. La selección de estas métricas está en función de las necesidades, objetivos y metas de un proyecto - "Qué es lo que se busca controlar".

(Pressman & Ph, 2007) expone las siguientes áreas de aplicación de las métricas de software:

- Nivel de código fuente.
- Estudio de diseño arquitectónico: Métricas que buscar dar un valor estimado de la arquitectura de un programa.
- Estudio de diseño orientado a objetos: Métricas que tratan estimar los atributos propios de los programas desarrollados bajo el enfoque orientado a objetos :
    - Acoplamiento
    - Tamaño
    - Cohesión

    - Primitivismo
- Modelado de requerimiento.
- Métricas orientadas a pruebas.
- Métricas de clase.

La amplia variedad de métricas hace difícil un estudio profundo de todas ellas, por esta razón en las siguientes etapas el estudio se enfoca en las métricas de código fuente, siendo estas calculadas a partir de análisis estático de código.

2.4.1. Revisión métricas de código.

Muchas de las métricas de software propuestas a lo largo de la historia no brindan apoyo en la práctica, esto es debido a la complejidad para comprender que es lo que miden o su método de aplicación en la mayoría de los proyectos comunes, esto no quiere decir que no sean válidas, lo que se busca exponer es que estas deben ser útiles y fácil de entender.

Los autores citados en este capítulo concuerdan en que las métricas deben cumplir las siguientes características:

- Simple y calculable.- Su cálculo no debe demandar esfuerzo o tiempo excesivo y debe ser fácil aprender cómo derivar la métrica y.
- Independiente del lenguaje de programación.- No debe depender de la sintaxis o de la semántica del lenguaje de programación. Debe basarse en el modelo de diseño o la estructura del programa en sí.
- Empírica e intuitivamente convincente.- Debe satisfacer las nociones intuitivas del ingeniero acerca del atributo de producto que se elabora (por ejemplo, una métrica que mide la cohesión del módulo debe aumentar en valor conforme aumenta el nivel de cohesión)
- Congruente y objetiva.- Debe producir resultados no ambiguos. Una tercera parte independiente debe poder derivar el mismo valor de métrica usando la misma información acerca del software.

A continuación se expone las métricas de código más relevantes, discutidas en foros[1] y blogs[2] de herramientas de análisis de código actualmente existentes.

[1] http://docs.sonarqube.org/display/SONAR/Metric+Definitions
[2] https://blogs.msdn.microsoft.com/zainnab/2011/05/26/code-metrics-maintainability-index/

2.4.1.1. Métricas de complejidad.

La complejidad Ciclomatica fue propuesta en 1976 por Thomas McCabe, es una métrica ampliamente utilizada y presente en la mayoría de los IDE de desarrollo, no está ligada a ningún tipo de lenguaje de programación en específico, proporciona una medida cuantitativa del grado de calidad de diseño de una clase. Conocer este tipo de información permite determinar cuan costoso es mantener un sistema a lo largo del tiempo. La Figura 2 proporciona una representación visual del cálculo de esta métrica.

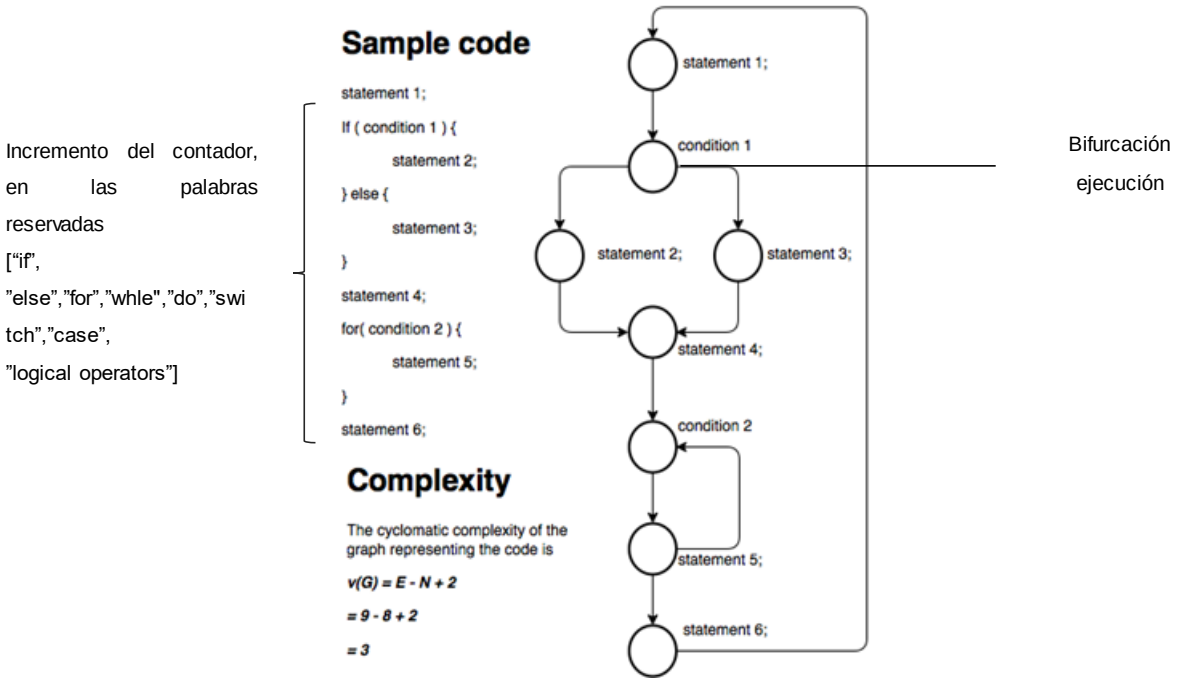


Figura 2: Complejidad ciclomatica.
Fuente : Autor

En donde cada estructura de control representa un unidad en el contador de complejidad, se debe tener claro que esta métrica no mide la complejidad en cuanto la interacción entre varios objetos en un sistema; en lugar de esto proporciona una estimación de la complejidad lógica de un programa tomando como referencia el numero sentencias de decisión.

(McCabe, Watson, & Wallace, 1996) en su trabajo muestra cómo aplicar la teoría de grafos para determinar este valor, mostrando el flujo de un programa como un grafo en donde los resultados de las sentencias condicionales son representados como aristas.

$$V = e - n + 2p$$

- E = Al número de aristas

- N = Numero de nodos
- P = El número de componentes conectados, es decir el número de subrutinas a las que se llama desde el método principal.

Existen varias formas de calcular la complejidad Ciclomatica, que van desde solamente contar predicados de decisión hasta el uso de herramientas ya existentes como Sonar Qube en donde la forma de cálculo se basa en el la formula $V = NC + 1$ en donde NC es el número de condiciones.

El problema con el valor de esta métrica en la programación orientada a objetos (POO) se basa en que CC no muestra la complejidad en cuanto relación entre clases sino más bien la complejidad lógica de un programa, por esta razón (Garg, 2014) propone un nuevo valor para complejidad ciclomatica, combinando dos conceptos por un lado el acoplamiento entre objetos (CBO) y por otra parte el concepto de complejidad ciclomatica tradicional, en la siguiente formula se resume este nuevo valor.

$$NewCyclomaticC = Cyclomatic + CBO$$

Tabla 4: Formula alternativa complejidad en POO.

Comparativa CM McCabe y NNCM Garg				
T	Clases	CM	CBO	NNCM
1	org.apache.bcel.classfile.AccessFlags	4	0	4
2	org.apache.bcel.classfile.Attribute	18	21	39
3	org.apache.bcel.classfile.AttributeReader	1	2	3
4	org.apache.bcel.classfile.ClassFormatException	1	0	1
5	org.apache.bcel.classfile.ClassParser	17	6	23
6	org.apache.bcel.classfile.Code	20	7	27
CM	Complejidad ciclomatica			
CBO	Acoplamiento entre objetos			
NCCM	Nuevo valor de complejidad ciclomatica			

Fuente : (Garg, 2014)

La tabla anterior es un fragmento de la tabla original expuesta por (Garg, 2014) en ella se puede observar que el nuevo valor de la complejidad ciclomatica difiere solamente cuando existe acoplamiento entre objetos, es decir cuando el grado de relación de una clase con otras aumenta.

Este nuevo valor de complejidad ciclomatica según el autor tiene mejor relación con el significado de programación orientada a objetos.

(Vahdat, Oneto, Anguita, Funk, & Rauterberg, 2015, p. 355) en su investigación llega a la conclusión que el valor de esta métrica puede ser usado como:

- Como medida alternativa a las notas de grado (según los autores esta esta correlacionada)
- Identificar grupos de estudiantes con mayor cantidad de problemas.
- Puede ser usada como predictor de dificultad de las tareas de programación.
- Proporcionar información a los instructores para adaptarse a las necesidades de cada estudiante.

2.4.1.2. Métricas halstead.

El conjunto de métricas propuesto por Maurice Halstead como parte de su ciencia del software es uno de los conjuntos de métricas más antiguos y estudiados de la ingeniería, esta métrica está basada en la cuenta de operando y operadores de un programa.

Autores como (Kasto & Whalley, 2013) han utilizado estas métricas en el estudio sobre la dificultad en la compresión en los cursos introductorios de programación, llegando a la conclusión que las métricas de software pueden llegar ser una herramienta realmente útil en predicción de este tipo de actividades.

En resumen, estas métricas son la longitud y el volumen como alternativa al conteo de líneas de código, el vocabulario como forma de cálculo de la complejidad para entender el código desarrollado, y el esfuerzo como medida del trabajo requerido para desarrollar un programa.

Se debe tener en cuenta que estas métricas se desarrollaron cuando aún no existían los lenguajes orientados a objetos, pero aun así en la actualidad varias herramientas de análisis de código continúan brindando soporte para estas métricas.

2.4.1.3. Métricas orientadas a objetos.

El conjunto propuesto por Chidamber & Kemerer en 1994 es ampliamente usado en el estudio de la calidad del diseño de una aplicación, estos conjuntos de métricas se basan principalmente en el estudio de herencia y la cohesión.

Métodos ponderados por clase (WMC).

Esta es una de las seis métricas originales propuestas por (Chidamber & Kemerer, 1994) en su trabajo sobre una suite de métricas para diseño orientado a objetos , el valor de esta métrica indica el costo de desarrollar y mantener una clase.

El valor de esta métrica es calculado como el total de métodos de la clase o la sumatoria de la complejidad de cada método pudiendo ser este valor de complejidad:

- Complejidad Ciclomatica de McCabe.
- El número de líneas de código.
- Asignación de 1 (Métodos no ponderados).

Tabla 5 Métodos ponderados por clase

WMC resumen	
Medición	
WMC alto	Indica que una clase es más específica, influye directamente en la capacidad de reutilización de código.
	Una clase con alto WMC es propensa a mayor número de errores y mayor dificultad para el mantenimiento.
	Una clase con alto WMC impacta directamente a las clases hijas , ya que heredan todos los métodos

Fuente: autor
Elaborado por : Autor

Acoplamiento entre objetos (CBO).

Esta métrica es usada para medir el acoplamiento entre clases, indica el número de otras clases a la cual una clase está ligada. Se tiene que tener en cuenta que eliminar el grado de acoplamiento de una clase es muy difícil, lo que se busca realmente es mantener los niveles de acoplamiento en menor grado posible(Reynolds, 2002, p. 91). Sonar Qube calcula esta métrica usando un contador, para cada identificador de una clase del proyecto el contador aumenta en 1.

Tabla 6 Acoplamiento entre objetos

CBO resumen	
Medición	Acoplamiento
CBO alto	Indica que la solución propuesta, no sigue un diseño modular.
	Una clase altamente acoplado es susceptible a cambios, ya que está fuertemente ligada a clases externas.
	Una clase con CBO alto es más propensa a errores y difícil de mantener, ya que sus funciones dependen fuertemente de otras clases.
	Indica un bajo grado de reutilización para una clase, debido a que esta no es totalmente independiente.

Fuente: El autor.
Elaborado por: El autor.

Profundidad del árbol de herencia (DIT).

La herencia es un tipo de relación entre clases que permite a los programadores reutilizar objetos previamente definidos incluyendo variables y operadores (Rosenberg & Hyatt, 1997).El uso de herencia sin embargo puede afectar el mantenimiento del programa, para controlar esto se utiliza la métrica DIT la cual calcula la distancia de una clase en relación a la clase de más alta jerarquía, permite ver la complejidad de una clase desde el punto de vista de la carga de métodos heredados. Entre mayor sea el número de métodos heredados más difícil es predecir y entender el comportamiento de una clase, esto afecta a la calidad del diseño.

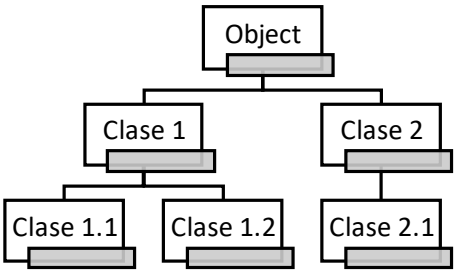


Figura 3: Ejemplo herencia.
Fuente: El autor.
Elaborado por: El autor.

En la Figura 3: Ejemplo herencia. La clase Object tiene un valor DIT de 1 puesto que de esta heredan todas las clases, las interfaces y clases tienen un valor de profundad de 1 más el valor de la cual heredan.

La utilidad de esta métrica se resume en la Tabla 7 Profundidad del árbol de herencia.

Tabla 7 Profundidad del árbol de herencia

DIT resumen	
Medición	Herencia
DIT alto	Indica que existe complejidad en el diseño.
	Un aspecto negativo es alta complejidad en clases más profundas, se torna más difícil entender el comportamiento.
	Indica que se está abusando de la herencia, Sonar establece que 5 es la profundidad máxima aceptable.
	El lado positivo de un DIT alto, es la capacidad de reutilización de

	código.

Fuente: El autor.
Elaborado por: El autor.

Número de hijos (NOC).

Esta métrica mide el número de hijos de una clase, a diferencia de DIT que se enfoca en la profundidad, además busca medir la amplitud de una clase, aunque se pueden relacionar estos valores para determinar si es necesario modificar el diseño.

Tabla 8 Número de hijos

NOC resumen	
Medición	Acoplamiento
NOC alto	Si una clase tiene un gran número de clases hijas, se debe tener cuidado a la realización un cambio puede afectar a las clases hijas.
	Indica un mayor grado de reutilización de la clase base, pero aumenta el riesgo de usar incorrectamente la herencia.

Fuente: El autor.
Elaborado por: El autor.

Respuesta para una clase (RFC).

Esta métrica mide el acoplamiento, contando el número de métodos de la clase, el número de métodos remotos directamente llamados por la clase, la fórmula para calcular el RFC es la siguiente:

$$RFC = MC + MR$$

Si existe múltiples llamadas para un método remoto este solo se cuenta una sola vez, la utilidad de esta merita se resume en la siguiente tabla.

RFC resumen	
Medición	Acoplamiento
RFC alto	Indica que una clase es más difícil de entender y mantener.

Fuente: El autor.
Elaborador por: El autor.

Es importante tener en cuenta que no se puede eliminar el acoplamiento, lo que se busca es mantener un bajo nivel de este.

2.4.1.4. Métricas de mantenibilidad

Esta métrica es utilizada para evaluar la calidad del código que se está desarrollando, esta métrica hace uso de la complejidad ciclomatica, el volumen de Halstead y líneas de código para determinar el grado de mantenibilidad, cuando más bajo es este valor mayor es la dificultad de realizar un cambio en el código y de que este sea entendido por otros programadores, la fórmula original para calcular este atributo toma como entrada la información de otras tres métricas:

- Complejidad ciclomatica.
- Líneas de código.
- Porcentaje de comentarios.

$$\text{Maintainability} = 171 - 5.2 \text{ x } \ln(\text{aveVol}) - 0.23 \text{ x ave V(g')} - 16.2 \text{ x } \ln(\text{aveL0C}) + (50 \text{ x } \sin(\text{d2.46 x perCM})$$

Sin embargo, en IDEs de desarrollo como visual estudio emplean esta métrica adaptándola a sus necesidades:

$$\underline{IM = (\text{MAX}(0{,}171 - 5.2 * \ln(\text{HV}) - 0.23(\text{CC}) - 16.2 * \ln(\text{LOC}) * 100/171)}$$

Es necesario acotar que no existe un plugin para netbeans que calcule esta métrica, por esta razón también he creído conveniente que forme parte del plugin de métricas.

2.4.1.5. Métricas orientadas al tamaño.

Esta métrica es ampliamente usada y se han propuesto diferentes esquemas para el conteo de líneas de código, cada una de estas variantes tiene su valides, en la siguiente tabla se presenta un resumen de las variantes de LOC.

- CLOC. - Esta variante cuenta solamente las líneas de código comentadas.
- SLOC. - Número de líneas de código, esta variante solamente toma en cuenta las líneas de código fuente.
- LOC.- Líneas de condigo, la forma más básica de esta métrica incluye los comentarios y las líneas de código, pero no los espacios en blanco.

Esta métrica permite medir la evolución de un programa sobre el tiempo, (Blikstein et al., 2014) en su investigación hicieron uso de esta métrica, para determinar el comportamiento de los alumnos, tiempo planeando o tiempo corrigiendo, partiendo del supuesto de que grandes modificaciones representaban que el alumno planificó la solución y pequeños cambios que el alumno estuvo corrigiendo o realizando mejoras.

En el Anexo 1 se puede visualizar de forma resumida las métricas identificadas durante las fases de investigación, este anexo describe los niveles en los cuales estas métricas pueden ser aplicadas.

2.5. Revisión de plugins similares.

En cuanto a los plugins existentes para netbeans, existen muy pocos en el repositorio oficial de la plataforma, si se ingresa el termino de búsqueda “software metrics o code metrics”, el número de resultados solamente es de dos (simple code metrics y Source code metrics), si bien existen otras herramientas propiamente del análisis de la calidad del software como PMD, SonarQube o CheckSyle; este trabajo no intenta replicar dichas herramientas, este trabajo se orienta más a la recolección de datos, dicho esto se exponen los siguientes plugins

2.5.1. Wakatime plugin.

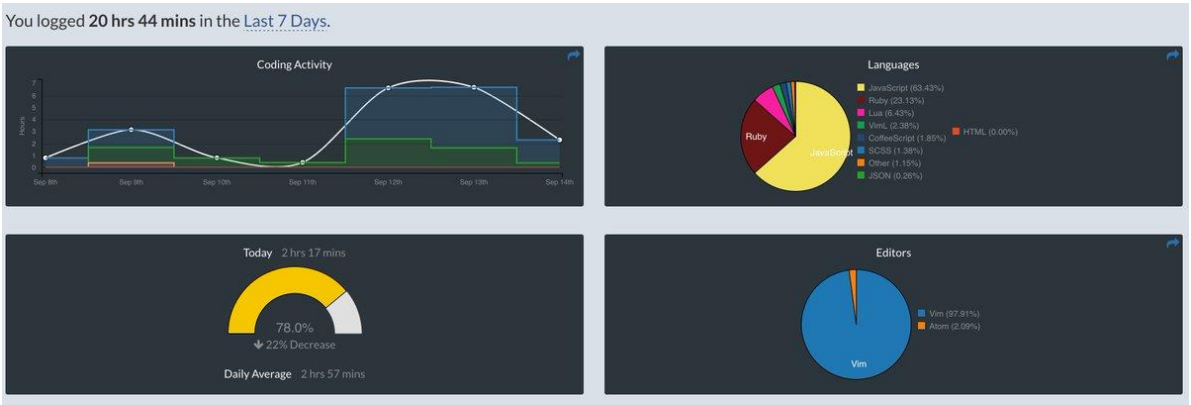


Figura 4: Plugin Wakatime
Fuente: Recuperado de : https://goo.gl/KEwORg
Elaborado por : Wakatime org

El tiempo es un recurso valioso, los ambientes de educación virtual[3] utilizan el tiempo como fuente de información para analizar el aprendizaje de los estudiantes. Por ejemplo: El tiempo que invierten revisando un recurso, al resolver un ejercicio, mantener sesión abierta etc.

Wakatime es un servicio que permite a los programadores obtener información sobre el tiempo y sus actividades de desarrollo, este servicio ofrece plugins para una amplia variedad de IDEs. Se hace mención de este ya que los datos que este capta pueden ser útiles en el campo del análisis del aprendizaje. Un ejemplo sobre el estudio de actividades a partir de datos sobre el tiempo puede ser revisado en el trabajo de (Martin & Whitmer, 2016).

[3] https://docs.moodle.org/dev/Learning_Analytics_Specification

2.5.2. Source code metrics.

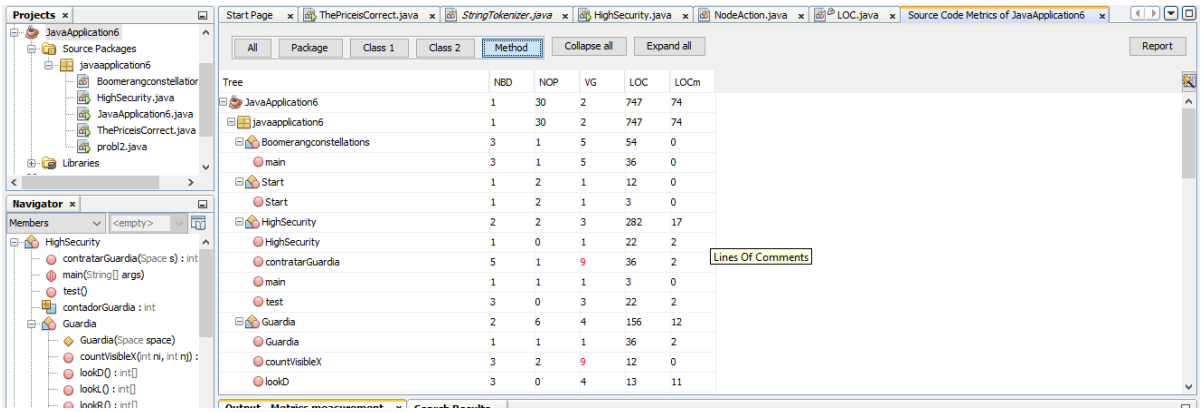


Figura 5 Plugin Source code metrics
Fuente: Autor

Este plugin presenta un amplio repertorio de métricas, proporciona información en los tres niveles típicos de un proyecto java; a pesar de esto, presenta dos inconvenientes, por un lado, para quienes inician en la programación esta información no es entendible a simple vista, y por el otro lado la ejecución del mismo es manual, no permite llevar un historial de las métricas con el fin de poder contar con información que muestre la evolución del desarrollo.

Este plugin trabaja haciendo uso de Java API compiler, en donde lo que se hace es parsear el código fuente en un árbol sintáctico AST para posteriormente extraer las métricas.

Las investigaciones estudiadas anteriormente demuestran que se necesita de conjunto mínimo de información para poder realizar análisis precisos; En este TT lo que se busca es solventar estos dos problemas extrayendo la información localmente usando métricas de software, ejecutando el plugin según las acciones del usuario y presentando la información haciendo uso de estadística.

2.5.3. Simple code metrics.

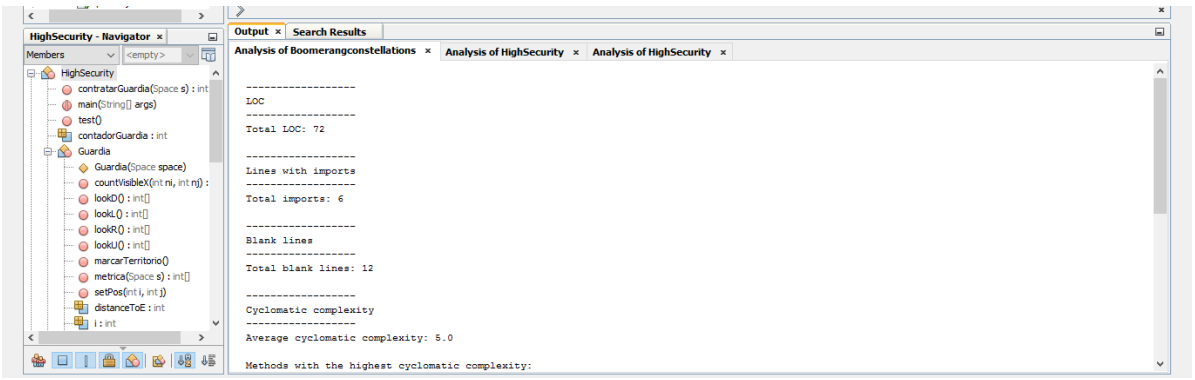


Figura 6: Plugin simple code metrics
Fuente: Autor

Otro de los plugins existentes es Simple code Metrics su repertorio se basa en 4 métricas, a diferencia de la anterior este plugin permite al programador identificar aquellos métodos con mayor complejidad.

2.6. Tecnología.

Hasta ahora se ha hablado de sobre los trabajos de investigación relacionados, las métricas de código más renombradas en las investigaciones citadas y los plugins de código abierto existentes que tienen relación a este TT.

La intención fue la de formar un conocimiento base para el desarrollo de la solución. En este sub apartado se explora los aspectos de categoría técnica, en donde se exponen las últimas técnicas en cuanto: recolección de métricas, sincronización de datos y el desarrollo del plugin.

2.6.1. Modelamiento de proyectos.

Las métricas estáticas se basan en el análisis por inspección del código fuente, muchas de las herramientas de análisis de código hacen uso del árbol de sintaxis abstracta[4] para la ejecución de cálculos, especialmente aquellas que están relacionadas con el diseño orientado a objetos.

En ciencias de la computación, un árbol de sintaxis abstracta es una representación en formato de árbol de la estructura código fuente escrito en algún lenguaje de programación, en donde cada nodo del árbol representa una parte del programa.

Cada IDE de programación implementa su propio enfoque de representación, los IDEs más populares proporcionan las siguientes librerías:

- Java Development Tools (JDT - Eclipse)
- Java Infrastructure Tools (JIT - Netbeans)
- Program Structure Interface (PSI - IntellJidea)

Todas estas implementaciones están en función de las características y objetivos de los entornos de desarrollo, existen librerías ligeras que ayudan a generar un AST utilizando menos recursos entre las más populares en la aplicación de análisis de código están:

- Byte Code engineering library.
- Object Web ASM.
- Java parser.

[4] http://www.eclipse.org/articles/Article-JavaCodeManipulation_AST/

Las dos primeras toman como entrada Bytecode, código compilado por Javac haciendo que sea más complejo analizar el código, debido a que este ha sido compilado y optimizado por el compilador, se ha optado por utilizar Java parser el cual toma como entrada ficheros .java permitiendo analizar el programa de forma exacta de como el estudiante lo escribió.

En la Figura 7 se proporciona un ejemplo de la representación de una clase llamada HelloUtpl en su correspondiente AST.

```
public class HelloUtpl {

    public static void main(String[] args) {
        // Prints "Hello, World" to the terminal window.
        System.out.println("Hello, World");
    }
}
```

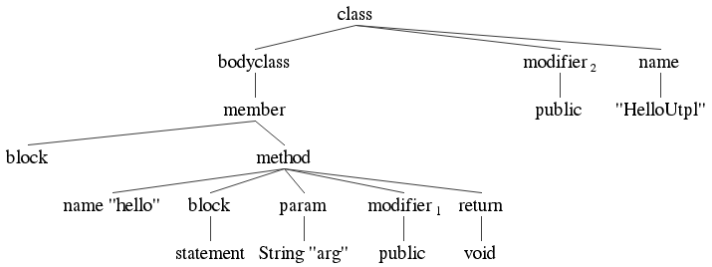


Figura 7 : Ejemplo AST generación
Fuente: El Autor.
Elaborado por: El Autor.

Cada elemento del programa (palabras reservadas, bloques, parámetros, modificadores de acceso etc.) forman parte del árbol, este tipo de estructuras permite navegar entre los elementos del programa para analizar y recolectar métricas.

La unión de estos árboles forma estructuras mucho más complejas, permitiendo generar modelos de representación de un proyecto completo, el grupo de Investigación de Ingeniería de Software[5] proporciona una aproximación para modelar un proyecto independiente de lenguaje.

[5] https://sewiki.iai.uni-bonn.de/research/jtransformer/api/java/prologast

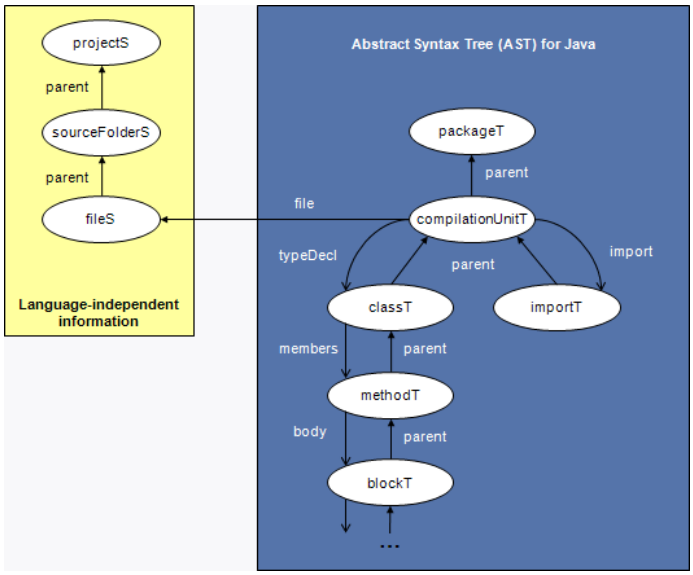


Figura 8. Representación independiente del lenguaje.
Fuente: https://goo.gl/U6ghbP
Elaborado por: Software Engineering Research

La Figura 8 muestra como se integran los AST a una representacion independientemente de su sintaxis formando una estructura comúnque de directorios y archivos, esta información es catalogada como elementos independientes debido a que no necesariamente deben estar ligados a un AST específico.

Por ejemplo eclipse tiene su propia forma de modelar la estructura de un proyecto, la figura 9 muestra los elementos que forman parte de su modelo.

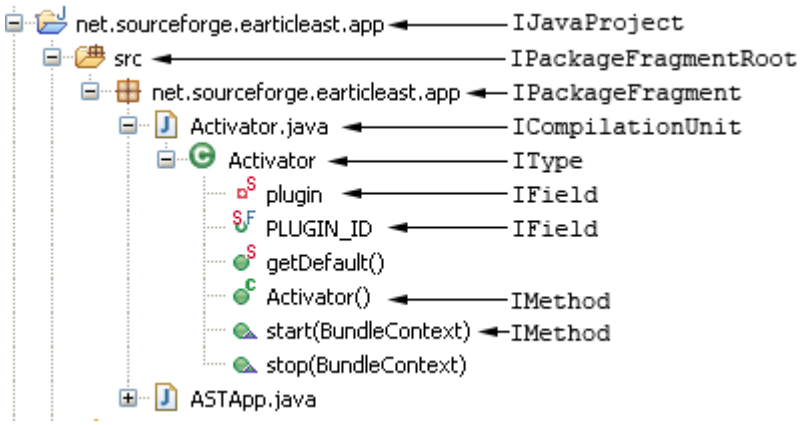


Figura 9. Eclipse modelo de proyectos.
Fuente: https://goo.gl/78puj8
Elaborado por : Organización Eclipse

La Figura 9 proporciona una representación abstracta que resume en gran medida todos los conceptos abordados en donde cada nivel del proyecto corresponde a un nodo del AST. Netbeans al igual que la mayoría de los IDE de programación java proporcionan un soporte nativo para este tipo de representaciones, proporcionando acceso al modelo del proyecto,

cada IDE presenta su propia forma de representación por lo cual este enfoque minimiza la portabilidad del código a otros entornos de desarrollo.

Por esta razón se ha creído conveniente implementar una representación propia del proyecto independiente del IDE de programación, dando lugar a la posibilidad de extraer métricas en cualquier entorno o editor que utilice el estudiante.

Para obtener este beneficio de portabilidad entre entornos se considera clave tener conocimiento en los siguientes puntos.

- Representación de código (AST).
- Definición de modelo proyecto.
- Monitoreo del código.

2.6.1.1. Neo4j como motor de representación y modelamiento.

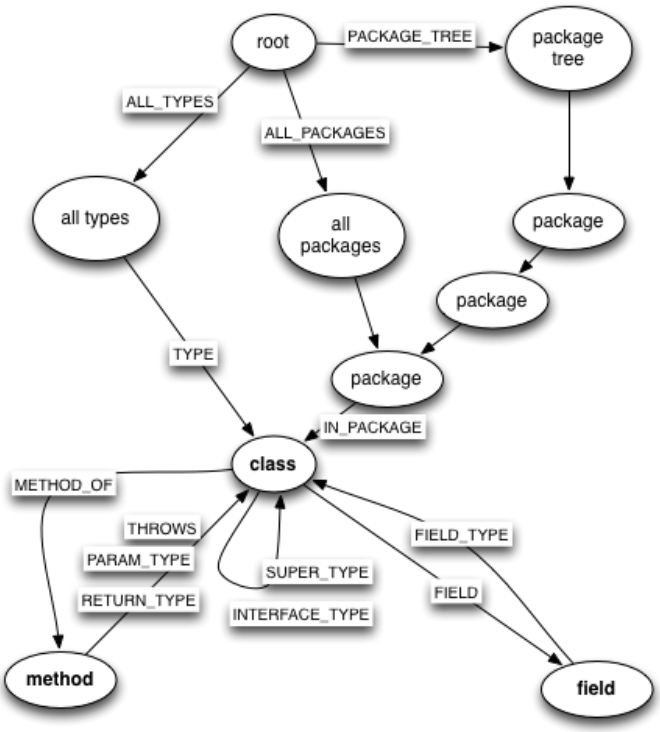


Figura 10. Grafo modelo de un proyecto.
Fuente: https://goo.gl/L8cnfc
Elaborado por: Michael Hunger

Neo4j es un proyecto de código abierto de bases de datos, esta base de datos se caracteriza por ser desarrollada bajo la plataforma java permitiendo almacenar datos en grafos en lugar de tablas. Según los desarrolladores de Neo4j puede ser empleada para

calcular métricas de software[6]. Modelando el proyecto como un grafo y ejecutando consultas mediante cypher.

(Urma & Mycroft, 2015) exponen este nuevo enfoque, siendo interesante la utilidad de este tipo de base de datos, abriendo las puertas para el desarrollo de modelos más flexibles y poderosos en cuanto a consultas sobre la estructura del código. Un ejemplo de cómo se explora la estructura del proyecto utilizando este enfoque es siguiente:

```
START root=node:types(class="java.lang.Object")
MATCH chain = (root)<-[:EXTENDS*]-(leaf)
RETURN extract(class IN nodes(chain) : class.name) AS classes,
```

La consulta extrae todos los nodos de tipo clase, no hay duda de que este enfoque es interesante, no obstante se debe tener en cuenta aspectos como el uso de memoria.

2.6.2. Transferencia de datos.

Uno de los objetivos de este proyecto es almacenamiento de datos tanto local como de forma remota, en el capítulo 1 contexto del proyecto se puede revisar en mayor detalle estos objetivos.

Para mantener estas capacidades es necesario diferenciar entre dos conceptos, replicación y sincronización términos que pueden ser intercambiables pero conceptualmente en la práctica manejan problemas totalmente diferentes.

La replicación es usada para mantener copias completas de un conjunto de datos, teniendo en mente la alta disponibilidad y rendimiento, comúnmente implementada entre servidores; mientras que, en la sincronización una base de datos central asume el rol de servidor y las bases de datos en los dispositivos móviles asumen el rol del cliente, formando un ecosistema cliente servidor. La base de datos central mantiene los datos de todos los dispositivos y los clientes solamente almacenan información limitada, propia de cada usuario[7].

Se debe tener en cuenta que en la sincronización intervienen dos tipos de acciones: push sincronización y pull sincronización, para enviar y recibir datos del servidor. Para tratar los errores de consistencia y conflicto en los datos se aplican diferentes tipos de algoritmos agrupados en dos categorías: pesimista y optimista[8].

[6] https://neo4j.com/blog/graph-databases-and-software-metrics-analysis/
[7] https://msdn.microsoft.com/en-us/library/dn589787.aspx
[8] http://airccse.org/journal/ijdms/papers/3411ijdms07.pdf

Las estrategias de tipo optimista permiten las lecturas y escrituras sin restricciones asumiendo que los errores se presentan raramente. En contraste, las estrategias pesimistas bloquean el acceso a los recursos mientras se actualizan y son liberados una vez que este proceso termina.

La figura 11, expone los tres tipos de sincronización:

- Flujo de datos unidireccional: En ese tipo de sincronización los datos son enviados en una sola dirección pudiendo ser esta desde el cliente al servidor o viceversa.
- Flujo de datos Bidireccional: En este tipo de sincronización el intercambio se da en dos direcciones, desde el servidor al cliente y viceversa.
- Flujo de datos Múltiples: Este tipo de sincronización involucra múltiples usuarios en múltiples direcciones, en este tipo de sincronización es más propensa a errores de consistencia.

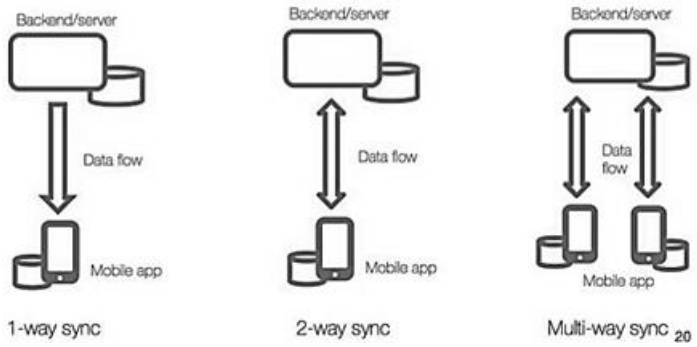


Figura 11. Tipos de sincronización.
Fuente : Recuperado de https://goo.gl/Y2Tl1A
Elaborado por: Niko Nelissen el dic 31, 2014

Patrones de sincronización.

(Mccormick & Schmidt, n.d.) Exponen tres patrones que intentan dar solución a esta problemática, todos estos están basados en máquina de estados finitos (FSM), en este trabajo se busca aplicar tanto el patrón full y último registro utilizando árboles de comportamiento como una alternativa a las FSM, las ventajas de estas estructuras pueden ser exploradas en el anexo 4.

Un árbol de comportamiento es una estructura que sirve para modelar y definir comportamientos, la idea de tras de esta estructura es la de afrontar los problemas que presentan las tradicionales máquinas de estados (FSM), en cuanto al crecimiento y escalabilidad en número de transiciones, la forma en cómo logra esto es eliminando las transacciones entre estos, definiendo mecanismos intermedios para decidir la acción a ejecutar, retornando a los invocadores mensajes de estado.

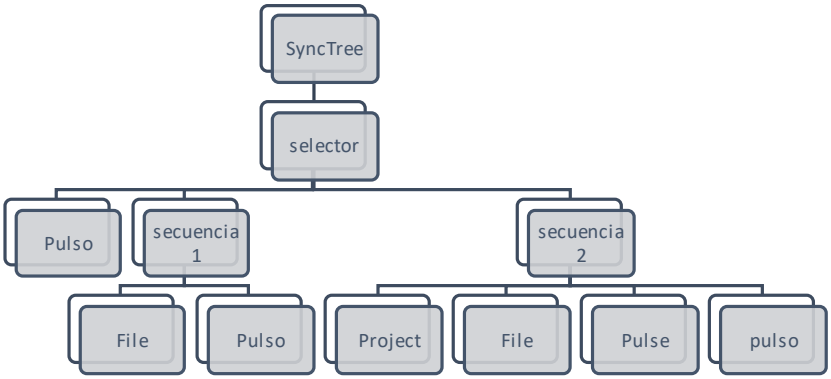


Figura 12. Árbol de sincronización.
Fuente :
Elaborado por: El autor.

- El nodo selector indica que se ejecutaran los nodos hijos hasta cuando uno retorne una respuesta de éxito.
- El nodo secuencia, indica que todas las tareas deben ejecutarse en un orden específico y terminar correctamente para retornar un código de éxito.
- Los nodos hojas representan la lógica como tal, es aquí donde se implementa las acciones a ejecutar.
- La sincronización se efectúa invocando el método step, del árbol, este método ejecuta el comportamiento y resolución de problemas en caso de existir alguno.

2.6.3. Netbeans apis.

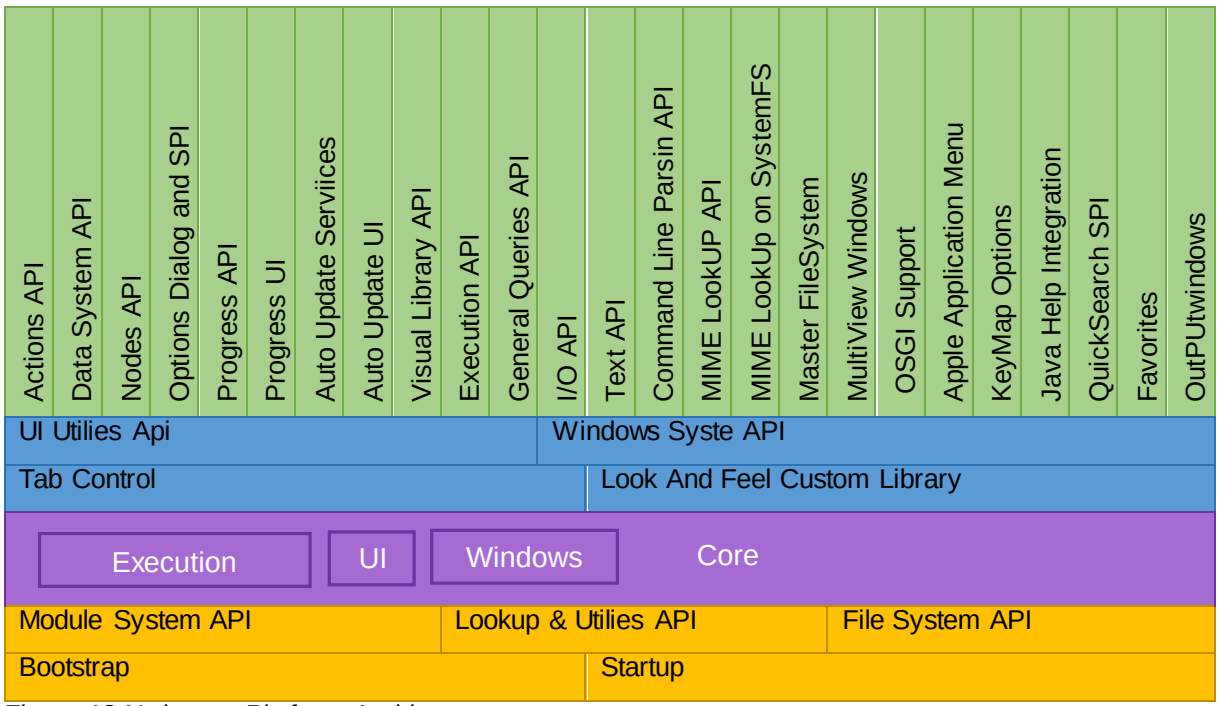


Figura 13 Netbeans Platform Architecture
Fuente: (Bock, Böck, Glick, & Konecny, 2012)

La plataforma Netbeans forma parte base de Netbeans IDE, las apis proporcionadas se resumen en gran medida a la figura 13, de estas las más relevantes y útiles para el proyecto son:

I/O Api	Dentro de este grupo están las API para presentar datos en consola
Actions API	Para ejecutar acciones bajo un contexto determinado.
Options Dialog	Utilizada para proporcionar ventanas de ajustes y opciones del monitor
Nodes API	Utilizada para escuchar eventos
Progess API& UI	Utilizada para mostrar el progreso de análisis.
Module System API	Utilizada para ejecutar acciones cuando el IDE es abierto o cerrado.
Master FileSytem	Utilizada como fuente de eventos alternativo al VFS de apache, minimiza el uso de recursos reutilizando la implementación existente en el IDE.

CAPITULO III
CONJUNTO DE DATOS DE RECOLECCIÓN

3.1. Introducción

En este capítulo se presenta los datos y métricas a recolectar, estos han sido seleccionados y agrupados en base a la utilidad identificada en los proyectos abordados en el capítulo 2.

Conforme se describe cada elemento de los conjuntos se incluirá una referencia a aquellos proyectos en los cuales, los datos han sido de utilidad para el análisis del aprendizaje en caso de existir.

3.2. Medida de actualización diferencial.

Medida de análisis del comportamiento propuesta (Blikstein et al., 2014), según los autores los resultados de su investigación indican que “existe una conexión entre las notas de grado de los estudiantes y la cantidad de cambio en sus patrones de programación” (Blikstein et al., 2014, p. 19).

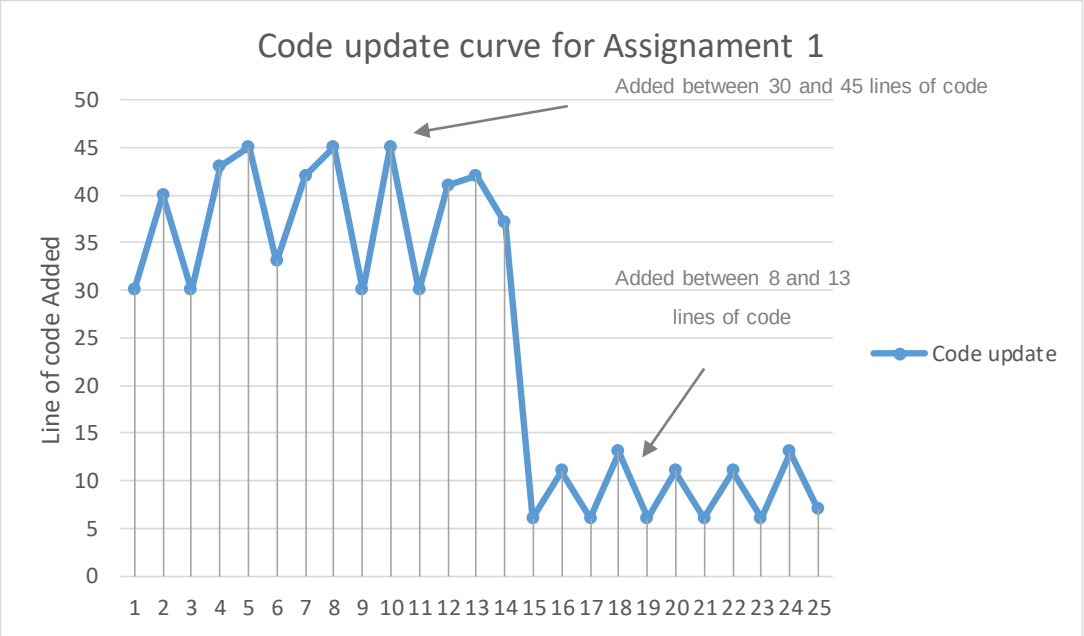


Figura 14: Code update curve
Fuente : (Blikstein et al., 2014)
Elaborado por: El autor.

La Figura 14 representa la curva de actualización de código, esta medida engloba otros datos relacionadas con las métricas estáticas:

- Líneas de código añadidas.
- Líneas de código eliminadas.

- Caracteres eliminados.
- Caracteres añadidos.

Para resumir la información sobre caracteres eliminados o añadidos en este proyecto se propone utilizar la distancia de Levenshtein, esta es la medida de distancia de edición o distancia entre palabras, representa el número mínimo de operaciones requeridas para transformar una cadena de caracteres en otra, este valor es ampliamente usado en la teoría de la información y ciencias de la computación para determinar cuan similares son dos cadenas de texto, su aplicación en este proyecto es para determinar cuantitativamente el esfuerzo de un cambio.

3.3. Datos ambiente y comportamiento.

En el análisis del aprendizaje, los datos son generados y recolectados producto de la interacción del estudiante con los recursos en línea, la mayor parte de esta información es facilitada por los sistemas de gestión del aprendizaje, la calidad y el volumen de datos ayudan a la aplicación de modelos predictivos y estadísticos para la predicción del rendimiento del estudiante.

(Martin & Whitmer, 2016) exponen que el comportamiento de los estudiantes es muy variable y para llevar a cabo estudio del mismo, proponen hacer uso de medidas relacionados con el tiempo:

- Tiempo invertido leyendo recursos.
- Recolección de tiempos sobre sesiones de estudio.
- Información de Logins.
- Tiempo apertura y cierre de recursos.

Se debe recalcar que la investigación llevada por (Martin & Whitmer, 2016) está enfocada al estudio del análisis del aprendizaje utilizando plataformas de educación virtual. Dicho esto los datos que se pretenden capturar siguen las mimas pautas pero enfocándolos en el uso de IDE de programación.

3.4. Recoleccion tiempos.

Medidas generales de analisis del aprendizaje se basan en el estudio del tiempo que un estudiante invierte en una asigunatura, revisando recursos, participando en foros etc. El tiempo es un recurso valioso y puede ser un indicador del esfuezo del estudiante para resolver un problema, por ejemplo el tiempo que ha invertido un estudiante para resolver una tarea de programacion.

Herramientas como wakatime proporcionan este tipo de información (enfocándose en métricas de líneas de código y métricas de tiempo); wakatime está basado en la recolección de datos por eventos, el principal evento para la captura de datos es cuando se guarda el código, el problema con este enfoque es que para dar soporte a una amplia familia de entornos de programación se necesita de programar un cliente del api ofrecidas en cada uno de los IDE en el caso de exista alguna.

Una mejor alternativa es implementar este soporte a nivel de sistema de archivos lo cual garantiza y facilita la escucha de este tipo de eventos minimizando las dependencias con el IDE.

Datos Logins y ambiente de trabajo.

Este conjunto proporciona información sobre el ambiente y rutinas de trabajo del estudiante.

- Inicio de sesión: Captura fechas, basadas en zonedatetime, cuando el estudiante edita código.
- Usuario de la maquina: Username de la máquina para determinar el propietario de la máquina.
- Network: Información sobre conexión de red, para determinar si está o no conectado al momento de trabajar en sus proyectos.
- UptimeMachine: Tiempo total que la maquina ha estado encendida, antes de que el estudiante empiece a programar.
- RAM total: La capacidad total de RAM de la máquina del estudiante
- RAM disponible: La RAM disponible de la maquina
- CPU: Numero de núcleos de la máquina

Adicionalmente a estos datos se agrega información relacionada con el tiempo de edición, de forma que ayude a determinar cuánto tiempo le tomo al estudiante efectuar el cambio.

Líneas añadidas: LOC líneas de código añadidas

Tiempo: El instante de tiempo en cual se llevó a cabo la modificación.

Archivo: El fichero de código sobre el cual está trabajando el estudiante.

Tabla 9 Ejemplo trama de captura de datos

INSTANTE	DATE	LÍNEAS AÑADIDAS	LEVENSHTEIN
INSTANTE 1	9:10 AM	10	40
INSTANTE 2	9:12 AM	90	320
INSTANTE 3	9:15 AM	12	50
INSTANTE 4	9:22 AM	5	8

Fuente: El autor.
Elaborado por: El autor.

La Figura 15 muestra un comportamiento inusual identificado, un estudiante no sería capaz de programar 90 líneas en 2 minutos, la métrica Levenshtein, está relacionada con las modificaciones, entre más significativos sean los cambios el valor de este será mayor, este valor intenta proporcionar datos relevantes sobre cuánto ha cambiado el código y que esfuerzo es necesario en cuanto a sustituciones y agregaciones de caracteres para volver al código inicial.

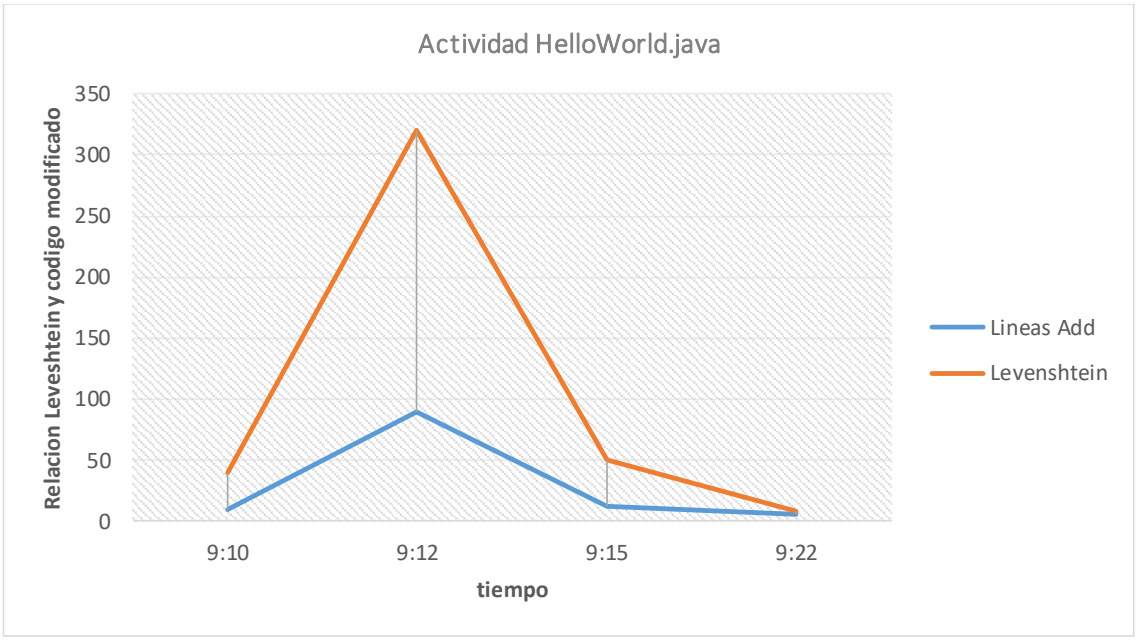


Figura 15 Representación Gráfica de la actividad
Fuente : Autor

3.5. Métricas de código.

- **Complejidad ciclomatica.**

Si bien esta métrica ha sido diseñada para medir cuan fácil es probar y mantener un programa, existen también estudios que revelan que esta métrica puede llegar a ser útil en el campo de la educación - Learning analytic. (Vahdat et al., 2015) manifiesta que el valor de esta métrica puede bridar información valiosa a los instructores de curso, proporcionando

información para identificar aquellos estudiantes que se están saliendo del camino óptimo de aprendizaje.

- **Métricas de Hasltead.**

Propuestas en 1977 por Maurice Halstead, sus métricas se basan en análisis de operandos y operadores de un programa, la extracción de estas métricas al igual que la anterior es el resultado de un análisis estático del código fuente. En la actualidad está métrica ha sufrido modificaciones (en cuanto a la definición de operando y operadores) para adaptarse a los lenguajes de programación orientados a objetos. Originalmente fueron propuestas para medir el esfuerzo de programación.

Las métricas a implementar se muestran en la siguiente lista:

- Longitud Programa.
- Vocabulario.
- Volumen.
- Esfuerzo.

- **Índice de mantenibilidad.**

Métrica usada para medir la calidad de las soluciones, indica cuan mantenible es el código que se ha desarrollado.

$$IM = (\mathrm{MAX}(0{,}171 - 5.2 * \ln(\mathrm{HV}) - 0.23(\mathrm{CC}) - 16.2 * \ln(\mathrm{LOC}) * 100/171)$$

- **Líneas de código.**

La métrica líneas de código es ampliamente usada en ingeniería de software, forma parte fundamental de la investigación de (Blikstein et al.), es usada como patrón de actualizaciones de código, el autor recalca que esta información podría ser valiosa y computacionalmente no costosa para detectar el progreso de un estudiante.(2014, p. 34)

- Radio de comentarios.
- Líneas de comentarios.
- Líneas de código fuente.

- **Número de métodos.**

Métrica usada para calcular la media de operaciones por clase, una clase debe tener un número equilibrado de operaciones, varias herramientas utilizan un valor de 3 a 7 operaciones por clase.

- **Métodos ponderados por clase.**

Esta métrica proporciona una visión de la complejidad de una clase y puede servir de referencia para determinar la capacidad de reutilización de la misma, ya que clases con más métodos son más específicas y por tanto tiene menos posibilidad de reutilización.

El factor de ponderación usado para los métodos será la complejidad ciclomatica.

3.6. Comportamiento compilaciones.

En la siguiente tabla se muestra que algoritmos hacen uso de este tipo de datos, el objetivo no es exponer su implementación o decir que se van a implementar en este TT, sino más bien es el de mostrar al lector la relevancia de estos datos (Errores de compilación, errores de ejecución, tipos de errores, tiempos corrección).

Este tipo de datos tiene utilidad cuando se aplican los algoritmos señalados en la Tabla 10 Algoritmos puntuación , estos algoritmos pretenden proporcionar medidas alternativas de rendimiento a las tradicionales notas de grado.

Tabla 10 Algoritmos puntuación

Dato	Uso identificado
Compilaciones exitosas	Algoritmo “Error Quotient” y WatWin
Compilaciones Fallidas	Algoritmo “Error Quotient” y WatWin
Tipos de errores	Algoritmo “Error Quotient” y WatWin
Tiempo de corrección	Watwin
Tiempo entre compilación	Watwin
Errores de ejecución	Normalized Programming State Model (NPSM)

Fuente: El autor
Elaborador por El autor.

La recolección de estos datos se basa en la aplicabilidad en el campo de Learning Analytic, el algoritmo propuesto por (Watson, Li, & Godwin) , trabaja con datos de errores de compilación, tiempos de corrección con el fin de predecir el rendimiento de los estudiantes.

Anteriormente a este trabajo (Matthew C.) propuso un algoritmo similar llamado “Error Quotient” sirviendo de base para algoritmos y variantes con el mismo enfoque, pese a que en su momento no pudo aseverar la credibilidad de su algoritmo debido a la falta de datos, Recientemente en el año 2015 (Matthew C & Dorn B) pusieron a prueba este con un conjunto de datos, resultado de las compilaciones de 27,698 usuarios, en este nuevo

estudio los autores manifiesta que este algoritmo puede servir como un sustituto a las medidas tradicionales de rendimiento.

Es importante tener en cuenta que tanto EQ como Watwin se basan en errores de sintaxis y no contemplan errores de tiempo de ejecución, es por esto que posteriormente (Carter, Hundhausen, & Adesope, 2015) proponen un nuevo algoritmo (NPSM) en el cual toman en cuenta los errores en tiempo de ejecución.

CAPITULO IV
PROCESO - DESARROLLO
DE CODELOGS

4.1. Introducción.

En este capítulo se presenta el desarrollo de la aplicación “CODELOGS”, la exposición es guiada tomando como referencia el ciclo de vida de desarrollo, en cada una de las etapas se muestra el trabajo realizado, los problemas y los resultados obtenidos, la exposición es guiada tomando en cuenta las etapas de la Figura 16.

4.2. Metodología de desarrollo.

Tradicionalmente las metodologías de desarrollo fueron diseñadas para un contexto en donde el trabajo es realizado en equipo, este proyecto es ejecutado por una sola persona en donde un solo actor debe desempeñar los diferentes roles que intervienen en el desarrollo de software. Según (Mogollón Afanador & Alberto Esteban Villamizar, 2011), cuando se trata de desarrollo individual es mejor adaptar y acoplar las existentes a las necesidades del desarrollador.

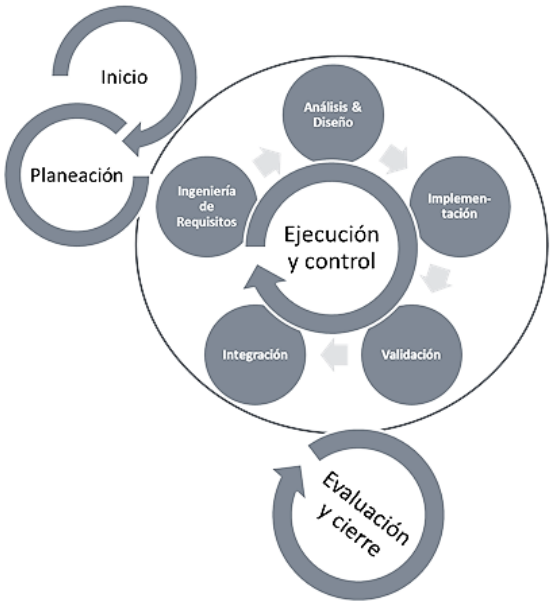


Figura 16. Ciclo de desarrollo
Fuente: Autor

Muchas de las metodologías están basadas en 4 fases fundamentales: planeación, análisis, diseño codificación y pruebas. En este proyecto se sigue un modelo de desarrollo iterativo en donde se planifica bloques temporales para el desarrollo de la funcionalidad del sistema, en cada iteración se analizaba e identificaba los elementos problemáticos que requerían de

un rediseño para dar soporte a los nuevos componentes que un principio no se tomaron en consideración.

4.3. Requisitos.

Por definición el levantamiento requisitos involucra la participación de usuarios a través entrevistas, talleres, conversatorios etc.; sin embargo, la naturaleza de este proyecto hace que sea complejo aplicar estos métodos, debido a que la mayor parte de estos provienen de los resultados de las investigaciones citadas y de la guía proporciona por el tutor.

En el contexto de este proyecto las investigaciones abordadas en el capito II revelan las características deseadas en este tipo de programas, así como las limitaciones. La descripción del TT proporciona una base sólida, la cual puede ser revisada en la documentación del proyecto tome como referencia el Anexo 9 de repositorios de proyecto.

4.3.1. Requisitos funcionales.

A. RF-01

	Especificación de requerimientos funcionales		
Código	Nombre	Fecha	Prioridad
RF-01	Autenticación del monitor	10/04/2016	Alta
Descripción	Se debe implementar un mecanismo que permita al servidor reconocer a que estudiante corresponden los datos que se están capturando.		
Entrada	Proceso	Salida	Restricciones
Token de acceso al servidor	El monitor de trabajo recolecta y envía información al servidor.	El servidor responde con los códigos HTTP estándar, 403 200,201 etc.	El token debe ser válido.
Funcionalidad	La herramienta debe permitir fácilmente identificar a que estudiante corresponden las tramas de información recibidas.		
Efectos	Ninguno		

A. RF-02

	Especificación de requerimientos funcionales		
Código	Nombre	Fecha	Prioridad
RF-02	Visualizaciones de datos	10/04/2016	Alta
Descripción	Los datos recolectados deberán ser representados mediante visualizaciones utilizando gráficos estadísticos.		
Entrada	Proceso	Salida	Restricciones
Petición de visualización.	El servidor carga las opciones visualización: Líneas de tiempo de programación. Tiempos, proyectos, modificaciones	Gráficos estadísticos. Barras, Líneas de tiempo	Contar con datos en el servidor.
Funcionalidad	Generación de gráficos estadísticos de los datos recolectados del estudiante.		
Fuente:	Ninguno		

B. RF-03

	Especificación de requerimientos funcionales		
Código	Nombre	Fecha	Prioridad
RF-03	Sincronización de datos	10/04/2016	Alta
Entrada	Proceso	Salida	Restricciones
Evento de modificación de código	Conforme el estudiante programa, Los datos deben ser almacenados localmente y sincronizados con el servidor cuando exista conexión con el servidor.	Base local actualizada con llave otorgada por el servidor.	ninguna
Funcionalidad	Sincronizar los datos locales con el servidor		
Efectos	Ninguno		

C. RF-04

	Especificación de requerimientos funcionales		
Código	Nombre	Fecha	Prioridad
RF-04	Crear sesión de trabajo	10/04/2016	Alta
Entrada	Proceso	Salida	Restricciones
Cuando se abre el IDE y se empieza a trabajar en un proyecto nuevo o existente.	El IDE notifica al monitor de trabajo que el estudiante ha empezado a trabajar en un proyecto, el monitor recibe esta petición y prepara las estructuras para el seguimiento	Creación de estructuras temporales, creación de registros locales.	El proyecto no este actualmente registrado, en la lista de proyectos abiertos.
Funcionalidad	Recuperar datos de cuando un estudiante empieza a programar.		
Efectos	Ninguno		

D. RF-05

	Especificación de requerimientos funcionales		
Código	Nombre	Fecha	Prioridad
RF-04	Cerrar sesión de trabajo	10/04/2016	Alta
Entrada	Proceso	Salida	Restricciones
Evento de modificación de código	El IDE notifica al monitor de trabajo que el estudiante ha dejado de trabajar en un proyecto, el monitor recibe esta petición y prepara las estructuras para el seguimiento	Eliminación de directorios temporales y estructuras en memoria RAM.	
Funcionalidad	Sincronizar los datos locales con el servidor		
Efectos	Ninguno		

E. RF-06

Especificación de requerimientos funcionales			
Código	Nombre	Fecha	Prioridad
RF-06	Seguimiento de	10/04/2016	Alta

	actividades		
Descripción	El programa debe ser capaz de escuchar los eventos de medicación del código para recuperar datos.		
Entrada	Proceso	Salida	Restricciones
Directorio de un proyecto	El IDE notifica al monitor de trabajo que el estudiante ha abierto y editado un proyecto, el monitor recibe esta petición y prepara las estructuras para el seguimiento	Creación de directorios temporales en OS del estudiante para análisis local del código.	
Funcionalidad	El monitor de trabajo recibe órdenes del IDE, indicándole que el estudiante ha comenzado a trabajar en un proyecto, el monitor prepara en ambiente de seguimiento para el proyecto.		
Efectos	Ninguno		

4.3.2. Requerimientos no funcionales.

Los requisitos no funcionales definen las restricciones de operación de un sistema.

Usabilidad

- La aplicación de visualización debe contar con un diseño responsivo de forma que se puede acceder a los gráficos estadísticos desde cualquier dispositivo.
- Las configuraciones deben ser mínimas para el usuario.

Rendimiento

- El recolector de datos deberá consumir menos de 98mb de memoria RAM.
- El servidor deberá operar con mínimo de 512mb RAM.

4.3.3. Casos de uso.

Los diagramas de caso de uso son una herramienta poderosa para mostrar las funciones del sistema de cara al usuario, este diagrama permite obtener una visión global de las principales funciones y de los actores que intervienen.

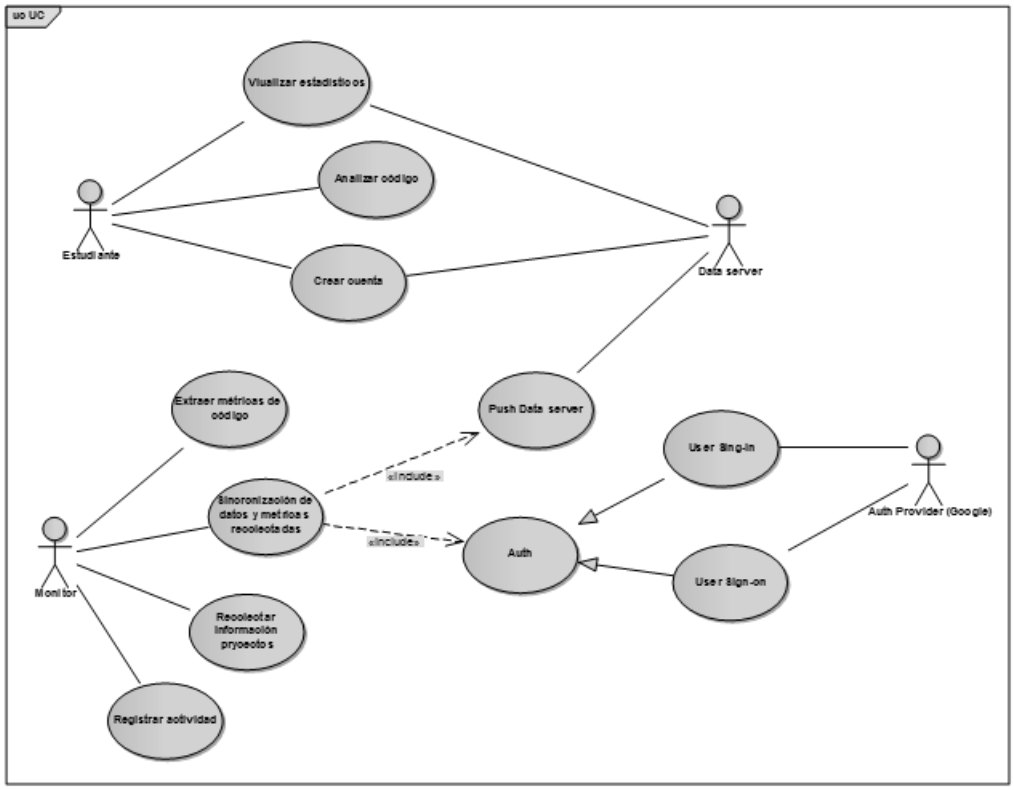


Figura 17. Diagrama de casos de uso.

Fuente: Autor.

Especificación de casos de USO

La especificación de casos de uso proporciona detalles puntuales de la funcionalidad, este sub apartado se presenta la especificación de los de casos de uso más importantes, la recolección, visualización de datos y sincronización de datos recolectaos.

Id UCS	**UCS-01**
Nombre UC	Visualizar datos
Actores	Estudiante, Server
Disparadores	El estudiante solicita un resumen de su actividad.
Pre condiciones	1. El estudiante debe contar con datos 2. El estudiante debe tener una cuenta 3. El estudiante debe tener los datos sincronizados.

Post Condiciones	La aplicación de visualización debería mostrar componentes de visualización.
Flujo Base.	1. El servidor valida al usuario 2. El estudiante solicita resumen 3. El servidor responde los datos en formato JSON 4. Si el estudiante no cuenta con datos, se redirección a una página alterna. 5. Caso contrario se procesa y renderiza los datos en gráficos estadísticos.
Flujo Alterno.	1. Usuario no valido. 2. Sin respuestas del servidor.

Id UCS	**UCS-02**
Nombre UC	Sincronización de datos.
Actores	Monitor, Server
Disparadores	Modificación de código.
Pre condiciones	1. El proyecto debe estar mapeado. 2. El monitor debe contar con un token de acceso valido. 3. El monitor debe estar ejecutándose en segundo plano.
Post Condiciones	Datos almacenados localmente y sincronizados con el servidor.
Flujo Base.	1. El monitor detecta actividad de programación. 2. El monitor aplica técnicas para recolectar datos. 3. El monitor almacena los datos localmente y se inicia la sincronización de los datos no sincronizados. 4. El servidor responde con UUIDs, o claves otorgadas a los datos.
Flujo Alterno.	3. Token no valido. 4. Sin respuestas del servidor.

4.4. Diseño.

En esta etapa se diseñó la arquitectura de la aplicación, permitiendo estructurar y modelar la solución, identificar los componentes que se desarrollaran en función de los objetivos y necesidades.

La Figura 18 representa la arquitectura seleccionada, conocida como cliente-servidor, en el Anexo 3 se proporciona la descripción arquitectónica (SAD), el cual proporciona esta información de forma más amplia utilizando el estilo 4 + 1 para representar los aspectos más importantes de la arquitectura.

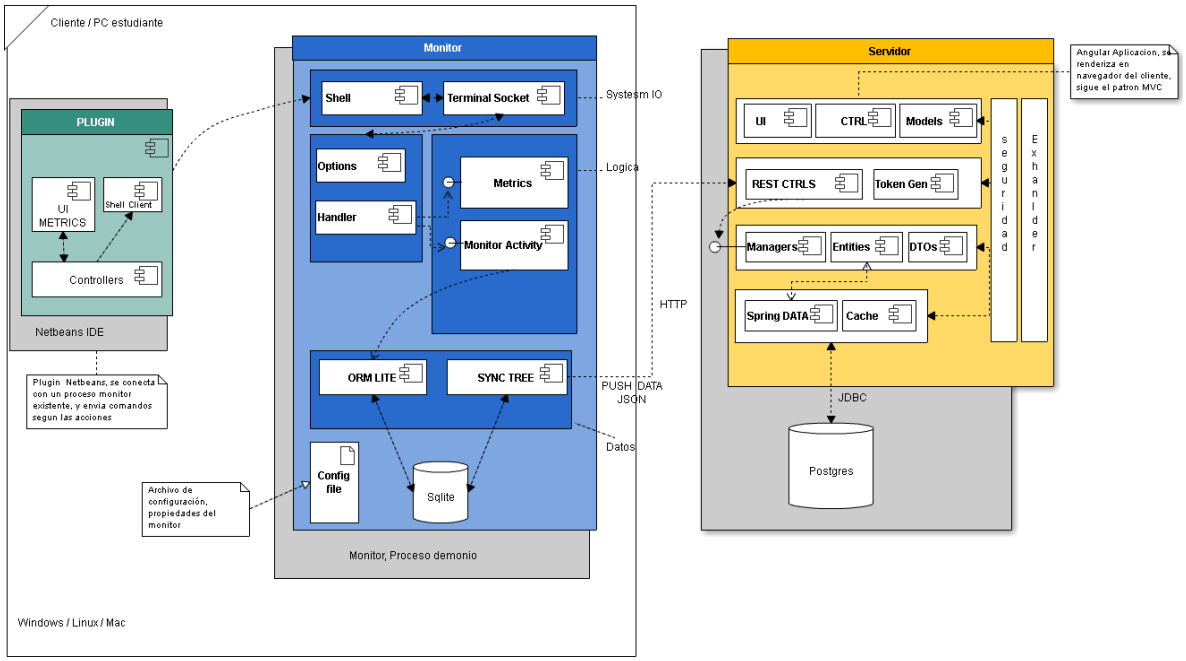


Figura 18 Visión de alto nivel de la arquitectura.
Fuente: El autor.
Elaboración: El autor.

El planteamiento de esta arquitectura responde a las necesidades identificadas, dividiendo el desarrollo en tres bloques, cada uno de estos bloques tiene su propia división de trabajo.

- Servidor.- El cual es encargado de almacenar los datos recolectados por el monitor, además de proporcionar una aplicación web para visualizar los datos.
- Monitor.- Programa encargado de aplicar las métricas de software, recolectar información de las actividades del estudiante y sincronizar los datos con el servidor.
- Plugin.- Programa cliente del monitor, el cual utiliza las API y comandos proporcionados por el monitor.

A. Diseño del Servidor.

REST es la forma en cómo interactúa las aplicaciones que residen en el nivel del cliente con el servidor, las acciones sobre los recursos son invocadas utilizando los métodos HTTP descritos en la especificación RFC7231[9].

Se han implementado 4 métodos básicos para interactuar con el servidor:

- POST Indica una acción de almacenamiento de un nuevo recurso.
- PUT Indica al servidor que un recurso debe ser almacenado, en caso de existir este debe ser actualizado.
- GET Recuperación de datos sin ningún efecto sobre estos.
- DELETE Elimina un recurso especificado.

4.4.1. Diseño API REST

Una vez que se liberan las Apis es difícil aplicar cambios sobre el diseño de estas[10], debido a al crecimiento de dependencias de los clientes, por esta razón el diseño forma parte fundamental de este bloque de trabajo, actualmente no existe un estándar adoptado por la comunidad para el diseño de API REST, lo que sí existe es una amplia variedad de enfoques y recomendaciones para el diseño de estas.

En esta sección se presenta:

- El diseño de las URL de los recursos.
- El tipo de autenticación para acceder a los recursos.
- Los formatos de intercambio de datos aceptados.

[9] https://tools.ietf.org/html/rfc7231#page-24
[10] https://elbauldelprogramador.com/buenas-practicas-para-el-diseno-de-una-api-restful-pragmatica/

4.4.1.1. Visualización recursos.

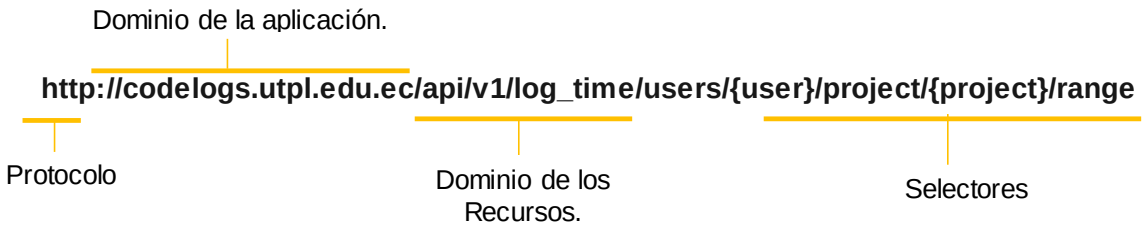


Figura 19: REST consumo URL.
Fuente: El autor.
Elaborado por: El autor.

La Figura 19 ilustra el formato de las URL utilizadas para el consumo de recursos, los demás puntos REST siguen el mismo esquema, las Apis de visualización son:

- Skills — Habilidades adquiridas por el programador siguiendo la idea (Gladwell, 2012) expone en su obra sobre la cantidad de tiempo que se tiene que invertir para adquirir habilidades en un campo.
- Loggtime — Tiempo invertido por el estudiante en prácticas de programación.
- Time spent — Tiempo invertido por recursos
- Time State — Punto para acceder a datos sobre el comportamiento del estudiante, cantidad de tiempo inactivo y cantidad tiempo de programación activa.
- Update pattern — Punto de acceso para visualización de patrones de comportamiento.[11]
- Time point — Punto de acceso para obtener datos de un punto en la línea de tiempo.
- Lang Usage — Punto de acceso para obtener datos de los lenguajes de programación más utilizados por el estudiante.
- Heart beat — Punto de acceso de los pulsos de código, resultantes de la codificación del estudiante.

[11] http://www.tandfonline.com/doi/abs/10.1080/10508406.2014.954750

4.4.1.2. Monitor.

En el caso del monitor los datos recolectados son enviados como parte del cuerpo de la petición, utilizando el formato Json, los puntos de acceso para el monitor siguen el siguiente esquema.

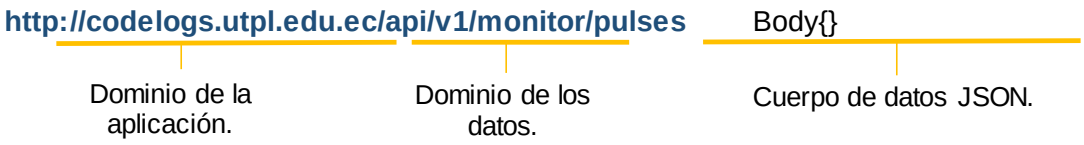


Figura 20: REST monitor URL.
Fuente: El autor.
Elaborado por: El autor.

- analysis — Punto de acceso para registrar resultados de análisis de código.
- file-code — Punto de acceso para registrar datos sobre los archivos de código.
- project — Punto de acceso para registrar datos sobre el proyecto
- work-sessions — Punto de acceso para registrar datos de trabajo, horarios tiempos etc.
- heart-beat — Punto de acceso para registrar actividad de programación.

4.4.1.3. Autenticación para uso de APIs.

La seguridad es gestionada utilizando JWT tokens los cuales siguen la especificación RFC7519[12], estos token son auto contenidos minimizando la cantidad de recursos para la capa transversal de seguridad.

En este proyecto se implementan dos tipos de token:

- JWT monitor
- JWT webapp.

La estructura de estos tokens sigue el siguiente formato:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwicm9sIjpbImFkbWluIiwidXNlciJdfQ.cqypy2kdDashrBf4azE9u5aP4ns0eG-cSyQ3tdQj0_Q

Los tokens están formados por tres partes (Parte 1), un encabezado el cual indica el algoritmo de cifrado, (Parte 2) un payload el cual puede contener datos del usuario como roles de acceso, correos etc. (pate 3) la signatura usada para validar el token.

[12] https://tools.ietf.org/html/rfc7519

La cadena de caracteres expuesta corresponde a la siguiente estructura

header	**{"alg":"HS256","typ":"JWT"}**

payload	{
	"sub": "1234567890",
	"name": "Ronald",
	"rol": ["admin","user"]
	}

signature	HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload))

Flujo

La Figura 21 representa el flujo tanto para el monitor como para la aplicación web, en el caso del monitor los token son generados dentro de la aplicación web y deben ser copiados en la sección de ajustes del plugin.

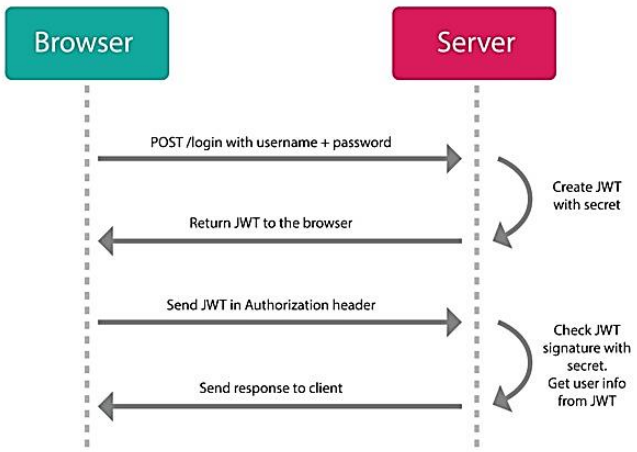


Figura 21. Flujo JWT
Fuente: https://jwt.io/introduction/
Elaborado por: JWT.io

Los token son utilizados en cada petición, formando parte del encabezado de seguridad de las peticiones HTTP, desde la especificación RFC1945[13] los campos de autorización han sido definidos de la siguiente forma:

```
Authorization  = "Authorization" ":" credentials
```

[13] https://tools.ietf.org/html/rfc1945#page-38

Para el proyecto se han definido dos tipos de autorización

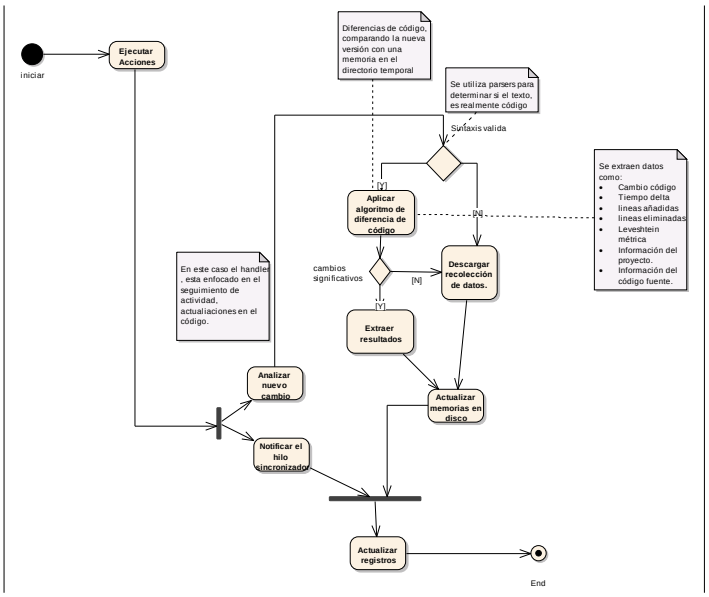
- Bearer : Utilizada en la aplicación web, para acceder a las APIs de análisis por el usuario.
- KWTBeaerer: Utilizada para el monitor para enviar datos e identificar al usuario.

4.4.1.4. Formato de representación.

El formato de intercambio de datos utilizando es la Notación de Objetos de JavaScript (JSON) un mecanismo de intercambio ligero de datos[14] , tanto el monitor como la aplicación de visualización utilizan este medio para interactuar.

Diseño del monitor

El monitor es un programa que se ejecuta como un servicio a nivel del sistema operativo, este enfoque permite minimizar el esfuerzo de desarrollo de nuevos plugins, proporcionando mecanismos para conectarse al servicio, permitiendo incluso ejecutar ordenes desde navegadores web o IDEs desarrollados sobre javascript.



Figuran 22 Eventos de actividad.
Fuente: Autor.

La recolección de la actividad del estudiante está basada en los eventos de actualización, modificación y creación de código, cuando un estudiante comienza a editar código en un proyecto se aplica las acciones descritas en la Figuran 22 .

[14] https://tools.ietf.org/html/rfc7159

4.4.2. Patrones de diseño.

Los patrones de diseño proporcionan soluciones a problemas frecuentes, estos patrones han sido identificados conforme se desarrollaba la aplicación, refinando diseño y codificando los cambios.

Según (Guerrero, Suárez, & Gutiérrez, 2013, p. 113) "Usar un patrón de Diseño requiere de años de experiencia, a veces se puede leer y leer sobre un Patrón y no entenderlo y otras veces aplicarlo mal". No obstante se tiene que ganar experiencia con la práctica y para lograr esto en este proyecto se han identificado y adaptado patrones a los problemas que se identificaron en fases exploratorias y de experimentación.

En el Anexo 5 se proporciona un grafo de los módulos con sus dependencias, el grafo permite tener una visión general de los módulos y librerías desarrolladas en donde se aplicaron los siguientes patrones de diseño:

Tabla 11: Tabla de patrones

Nombre	Uso	Modulo aplicado
Patrón decorador	Es usado a nivel del sistema de archivos para crear filtros complejos de búsqueda.	FileSystem
Patrón tuberías y filtros	Es usado por el monitor para conectar tareas y ejecutar transformación y completado de campos de información.	Monitor
Patrón estrategia	Patrón usado en la generación de AST, dependiendo del lenguaje de programación se selecciona el algoritmo capaz de generar el AST.	MetricasBase
Patrón compuesto	Aplicado en la generación de AST del código, se modela el proyecto y los principales componentes.	MetricasBase
Patrón visitor	Usado para explorar un compuesto, a nivel de exploración de un AST.	MetricasBase
Patrón inyección de dependencias	Patrón de diseño utilizado en todo el proyecto para resolver dependencias entre objetos del sistema.	Todos

Patrón comando.-	Patrón de diseño usado en la shell y el monitor para indicar acciones a ejecutar.	Monitor
Patrón Memento	Patrón memento es usado a nivel de memorias (recuerdos) para un archivo de código, para analizar las diferencias entre dos cambios, no se usa la memoria RAM para mantener todo el contenido de un archivo ya que esto sería muy costoso; en lugar de esto se utiliza referencias a una memoria en disco en un directorio temporal del SO.	Monitor

Fuente: El autor.
Elaborador por: El autor.

4.4.3. Modelo de cálculos.

En la actualidad los computadores cuentan con más de un procesador, para aprovechar estos recursos se busca definir un modelo de procesamiento que aproveche estos recursos[15] utilizando el framework fork/join, si bien los cálculos pueden implementarse secuencialmente (en un solo hilo) se ha optado por implementar un modelo de procesamiento basado en grafos, para acelerar los procesos y adquirir nuevo conocimiento sobre temas relacionados con multiprocesamiento y modularidad, la solucion al estar basada en modulos brinda la posiblidad y flexibiblidad de reutilizar implementaciones en tiempo de ejecucion y evitar recalculos de metricas de forma inecesaria. Para lograr este objetivo se ha propuesto el siguiente modelo de calculo.

El modelo de paralelismo implementado en este proyecto se basa en el uso de grafos, un grafo es definido como un conjunto de vértices (V) y aristas(A) para representar relaciones entre elementos; existen dos categorías principales: por un lado, se tiene los grafos no dirigidos los cuales se caracterizan por no tener sentido en las relaciones entre sus nodos por otro se tiene los grafos dirigidos en donde el flujo de datos tiene dirección, en la Figura 23 se presenta el primer enfoque de multiprocesamiento para el cálculo de métricas.

[15] https://docs.oracle.com/javase/tutorial/essential/concurrency/forkjoin.html

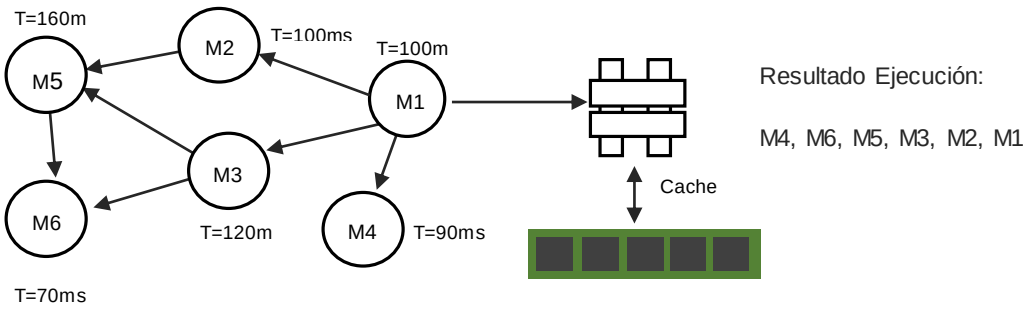


Figura 23: Enfoque 1 multiprocesamiento
Fuente: Autor.
Elaborado por: El autor.

Grafo dirigido no recursivo.

El uso de este tipo de estructuras permite garantizar que el cliente no proporcione métricas con dependencias recursivas - en primera instancia se debería verificar que la nueva métrica cumpla con una serie de requisitos mínimos:

- No existan dependencias recursivas o para sí misma.
- Que se satisfaga los requerimientos mínimos para un módulo (formulas y definiciones).

Para poner en claro la utilidad de este modelo se toma en cuenta los tiempos asignados a cada uno de los nodos del grafo, estos datos corresponden a una métrica (nodo) junto con el tiempo de procesamiento para obtener el resultado de una métrica.

- Cada nodo representa una métrica, una métrica involucra costo tanto de tiempo de procesamiento como uso de memoria. Las métricas de código son un tipo de cálculo el cual se basan en la inspección del código fuente escrito por el estudiante para obtener medidas.
- Cada nodo tiene asociado un valor de tipo T, el cual representa el tiempo que tarda en procesar dicho nodo.

Consideración 1: “Programa “Hello.java””

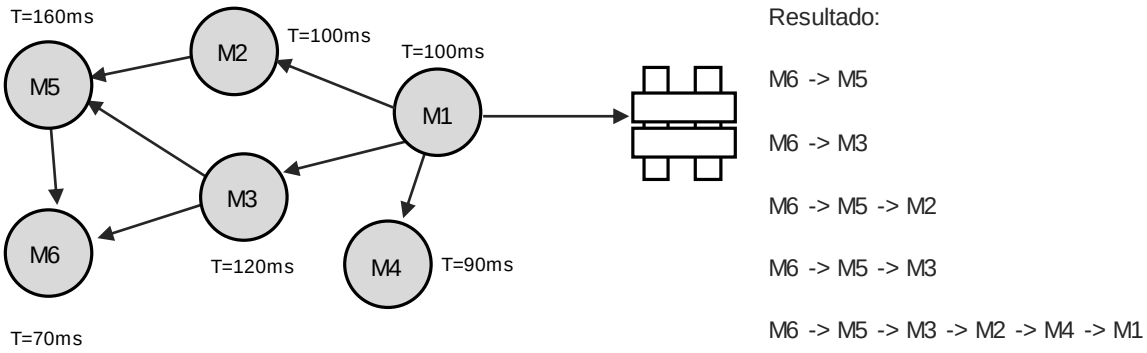


Figura 24: Modelo de procesamiento Enfoque 2
Autor: El autor.
Elaborador por: El autor.

Un modelo de paralelismo solamente enfocado en grafos no lo es todo, un mayor uso de los procesadores involucra un mayor consumo de memoria RAM, debido a que si solamente se tienen en consideración este enfoque sin un mecanismo que evite el cálculo de dependencias de forma innecesaria el resultado terminaría en alto uso de CPU.

Considere el hecho en el cual el nodo M5 se procesa antes que M2 ya que esta es una dependencia, luego de terminar los cálculos para M2 se pasa a calcular la métrica M3 cuya dependencia es M5, el cálculo se volvería a ejecutar nuevamente lo cual involucra gasto incensario de recursos.

Ventajas de este enfoque

- Reutilización de funcionalidades y multiprocesamiento en un solo punto.
- En el caso de una métrica de tipo compuesta (Radio de comentarios) = CLOC / LOC, simplemente reutilizaría el resultado utilizado para calcular CLOC y LOC en la nueva métrica.
- Aprovechamiento de Hardware.

Problemas de este enfoque:

- Las tareas aún se siguen recalculando.

Consideración 2

Para simplificar y demostrar la utilidad de este modelo de procesamiento solo se tendrá un hilo, por lo cual las métricas se ejecutarán una tras otra, sin posibilidad de ejecutar varias al mismo tiempo.

La ejecución del (nodo – M1)

Cuando se procesa un nodo con dependencias el procesador siempre explora el grafo hasta llegar a los nodos terminales.

Tabla 12 Resultados de procesamiento sin cache.

Step	Dependencia Objetivo	Métrica	Nodo AST	Tiempo Total
1	M6	M3	Hello.java	M6 + M3 = 190
2	M6	M5	Hello.java	M6 + M5 = 230
3	M6 M5	M2	Hello.java	… 330
4	M6 M5 M3 M2 M4	M1	Hello.java	… 640ms

Fuente: Autor.
Elaborador por: El autor.

En el paso 5 la ejecución de la métrica M2 involucra un incremento considerable de tiempo de procesamiento debido a que tiene que resolver las dependencias, para poder llevar los cálculos propios de M2.

Implementación de una cache.

Supongamos que el tiempo de acceso a la cache es de 5ms, los resultados de este nuevo enfoque son los siguientes:

Tabla 13. Tabla procesamiento con cache.

Step	Métrica	Dependencias	Nodo AST	Tiempo Total
1	M3	M6	Hello.java	70 + 120 = 190
2	M5	M6	Hello.java	1*cache + 160 = 165
3	M2	M6 M5	Hello.java	2*cache + 100 = 110
4	M1	M6 M5 M3 M2 M4	Hello.java	5*cache + 100 = 175

Fuente: Autor.
Elaborado por: El autor.

La idea es que con este enfoque, los resultados se reutilicen sin tener que ejecutar nuevamente cálculos debido a dependencias.

4.4.4. Métricas Dinámicas.

Las métricas dinámicas son todas aquellas que son recolectadas de un programa en ejecución; para entender los problemas entorno a este tipo de métricas se debe tener claro los siguientes conceptos:

4.4.5. Proceso

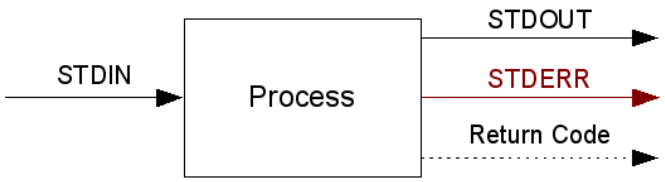


Figura 25 Proceso
Fuente: Autor.

A nivel de sistema operativo un proceso es definido como el conjunto de instrucciones ejecutados por el procesador, cada programa para poder ser ejecutado necesita ser gestionado por el sistema operativo, y esto se logra a través de la creación de procesos los cuales contienen información a nivel de registros, variables, contador de programa, memoria de trabajo etc.

Por lo general los IDE programación cuentan con herramientas como ant, maven, gradle etc. estas herramientas facilitan la ejecución de las tareas del ciclo de vida de desarrollo de un programa, dentro de estas tareas está incluido la ejecución y la compilación del programa. Por esta razón para poder obtener información del resultado de la ejecución o compilación de un programa se tiene que utilizar medios como la recolección de datos a través de las salidas de un proceso.

Todo proceso en ejecución tiene por lo general una descripción de fichero asociando a una entrada, una salida y una salida con errores, estos son representados como stdin, stdout y stderr respectivamente (solo como una forma de representación), la idea es poder redirigir la salida de error estándar del programa desarrollado por el estudiante para identificar los errores más comunes, así como el tiempo que tarda en resolverlo.

Netbeans por defecto en las aplicaciones simples de java utiliza ant para la simplificación de las tareas, para la recolección de esta información se ha identificado 2 estrategias.

a) Basado en las salidas del proceso.
b) Basado en los logs de las herramientas (ant, maven etc.).

Ant basa su configuración en ficheros XML, este permite modificar el archivo de construcción para re direccionar la salida estándar de la ejecución de un programa, esta técnica permitirá al plugin recuperar datos procedentes de la ejecución de un programa de un estudiante.

La primera estrategia no es muy recomendada puesto que interfiere con las acciones del IDE de programación, si se da el caso de redirigir la salida de una tarea en ant, los mensajes en consola de netbeans no se mostrarán, la estrategia (b) es la que en menor medida interfiere con el IDE, esta estrategia hace uso de las etiquetas

```
<record name="records-simple.log" action="start" />
<record name="ISO.log" loglevel="verbose" action="stop" />
```

4.4.6. Complementos maven.

Otro enfoque para recolectar los problemas de compilación y ejecución es mediante el uso plugins maven, en este proyecto se utiliza maven core extensions[16] junto con el uso de agentes java. Una agente java no es más que un programa que permite modificar otro programa java durante la ejecución, para lograr esto se hace uso del bytecode del programa, instrucciones a nivel de JVM (Java Virtual Machine), estas instrucciones son generadas producto de instrucción javac. Internamente todo programa tiene un área métodos, heap, pilas, classLoader, la figura 26 ilustra estas partes

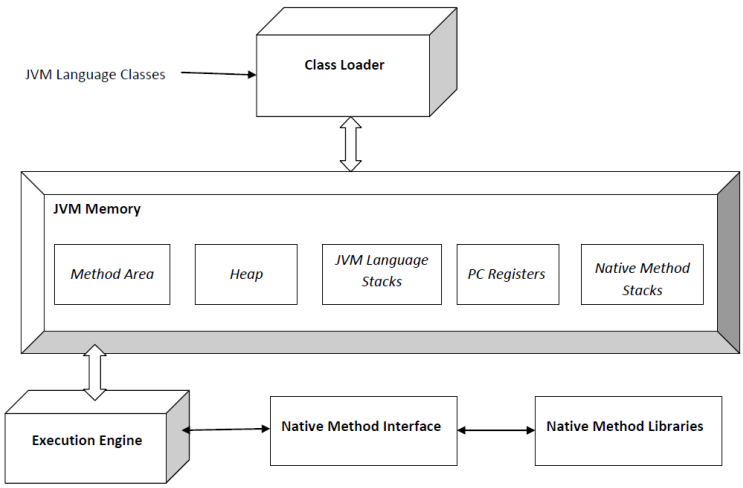


Figura 26 Especificación JVM base.
Fuente: https://goo.gl/5oue3n
Elaborado por: Michelle Ridomi

[16] El tema se expone en mayor detalle en siguiente enlace http://takari.io/2015/03/19/core-extensions.html

Al modificar el código del estudiante con esta técnica se requiere añadir el conjunto de librerías desarrolladas en el classPath del programa del estudiante, la razón es que el agente inyecta código externo con dependencias que aún no han sido agregadas al proyecto del alumno.

Para modificar el bytecode se hace uso de ASM,

```
224         mv.visitVarInsn(LSTORE, 5);
225         mv.visitLabel(l3);
226         mv.visitLineNumber(36, l3);
227         mv.visitMethodInsn(INVOKESTATIC, "java/lang/System", "currentTimeMillis",
228         mv.visitVarInsn(LSTORE, 1);
229         Label l17 = new Label();
230         mv.visitLabel(l17);
231         mv.visitLineNumber(37, l17);
232         mv.visitMethodInsn(INVOKESTATIC, CLASS_NAME, "codeLogsProgramMethod_1_0_0
233         mv.visitLabel(l4);
```

Figura 27. Bytecode Modificación.
Fuente : https://goo.gl/PkjKNP
Elaborado por: El autor.

La Figura 27 muestra fragmento de cómo se lleva a cabo la modificación del bytecode mediante ASM en los agentes java[17]. Por otro lado las extensiones de maven sirven como un medio o disparador para escuchar eventos de compilación, ejecución y mediante esto automáticamente inyectar el código necesario para capturar datos.

Se debe tener en cuenta que se requiere tener las dependencias codelogs en el classpath del proyecto del estudiante, esto es debido a que el código inyectado requiere de las librerías desarrolladas en este proyecto.

[17] https://goo.gl/yVpnRs Implementación de un agente java.

4.5. Implementación.

En esta fase el objetivo es plasmar el diseño en código, se utilizó java para programar los componentes diseñados, el Anexo 6 proporciona mayor detalle sobre esta etapa, siendo la etapa a la que se destinó más tiempo.

En total se programaron 10 artefactos correspondientes a los 3 bloques generales de trabajo, entre los cuales están incluidas las librerías y el núcleo de la aplicación, todos los componentes desarrollados pueden ser visualizados en la sección de anexos.

4.5.1. Programación de monitor.

El monitor, programa que es independiente de la plataforma fue construido utilizando intellJidea como IDE, este proporciona herramientas para medir la cantidad del trabajo en los diferentes componentes.

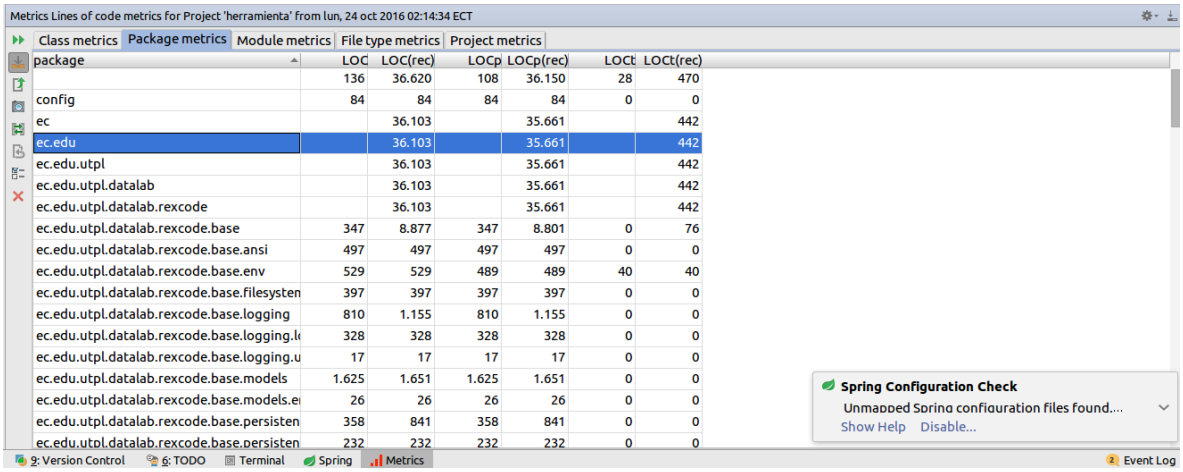


Figura 28: Líneas de código por paquetes del monitor
Fuente: El autor.
Elaborado por: El autor.

La Figura 28 indica la cantidad de código desarrollado en los diferentes paquetes del proyecto, el valor de 36.103 es el resultado del conteo de líneas de código tanto comentadas como de código fuente, el total de líneas código solamente de funcionalidad son alrededor de 23.453.

4.5.2. Programación del servidor.

Para la construcción del servidor se utilizó Jhipster, un stack de tecnología que agrupa los siguientes frameworks y librerías:

- Spring boot
- Gtaling test
- AngularJs

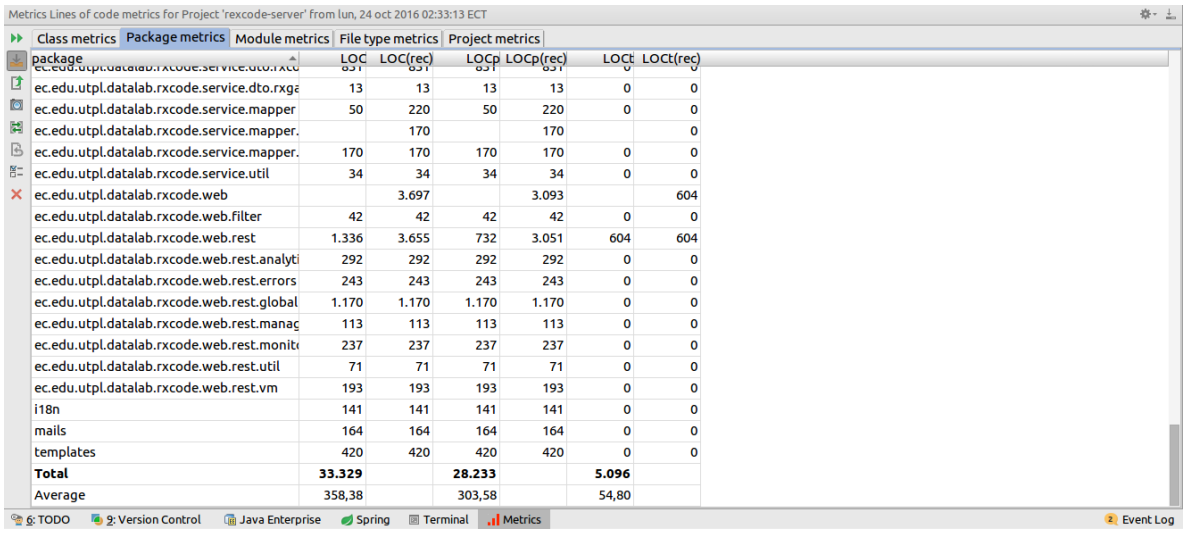


Figura 29: Líneas de código servidor.
Fuente: El autor.
Elaborado por: El autor.

En la Figura 29 se puede observar la cantidad de código desarrollado para el lado del servidor, dando un total de 28.233 líneas de código.

4.5.3. Programación del plugin.

Figura 30 Líneas de código Plugin.
Fuente: El autor.
Elaborado por: El autor.

En la figura 30 se muestra la cantidad de código desarrollado para el plugin Netbeans tiene alrededor de 7.786 líneas de código, este plugin es un enlace entre el monitor de trabajo (proceso en background) y el IDE de programación.

4.6. Pruebas.

La etapa de pruebas por si misma involucra su propio ciclo de vida, según los expertos es recomendable aplicar las pruebas una vez que se ha obtenido una versión estable del sistema. Las pruebas que se han aplicado en este proyecto son:

4.6.1. Pruebas servidor.

Las pruebas de carga tratan de crear simulaciones en un entorno de producción, sobre el comportamiento del servidor bajo ciertas circunstancias; Una prueba de carga permite medir los tiempos de respuesta, tasas de rendimiento, y los niveles de utilización de recursos para identificar el punto de ruptura de la aplicación, en el supuesto de que el punto de ruptura se produce por debajo de la condición de carga máxima.

Precondiciones

Las pruebas serán aplicadas con dos conjuntos de 30 y 100 usuarios conectados simultáneamente, este tipo de pruebas pretende mostrar los tiempos de respuesta de todas las transacciones importantes de la aplicación.

Las pruebas se ejecutaron utilizado gatling, además se debe tener en cuenta que los resultados dependen del ambiente de despliegue de la aplicación; en nuestro caso la aplicación fue ejecutada en nodo Heroku con las siguientes características:

- Nodo 512 MB RAM
- Ancho de banda limitado 1 Web Worker.

Resultados

Los resultados indican un rango aceptable de estabilidad del servidor, pese a que estas pruebas fueron ejecutadas utilizando un ambiente gratuito de Heroku .Existen dos modos de despliegue: modo producción y modo desarrollo, en el Anexo 8 se proporciona mayor información sobre cómo utilizar estos modos para la aplicación de las pruebas.

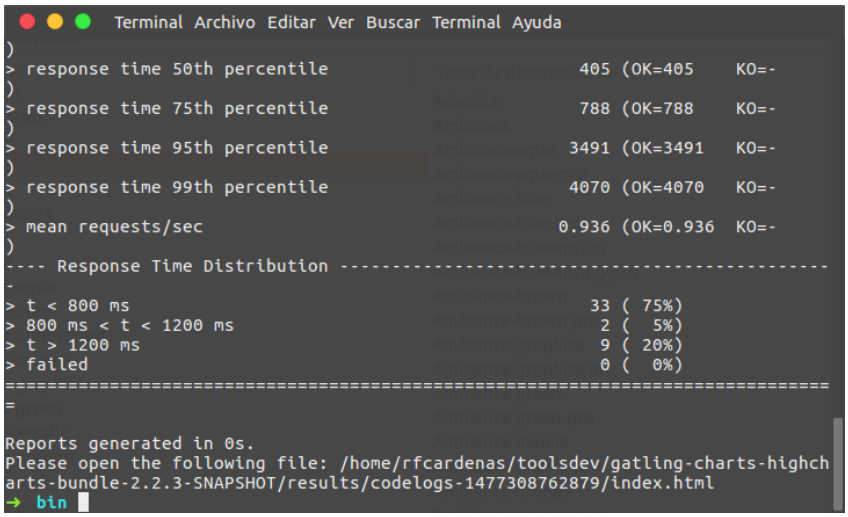


Figura 31 pruebas de carga 1.
Fuente: Autor.
Elaborado por: El autor.

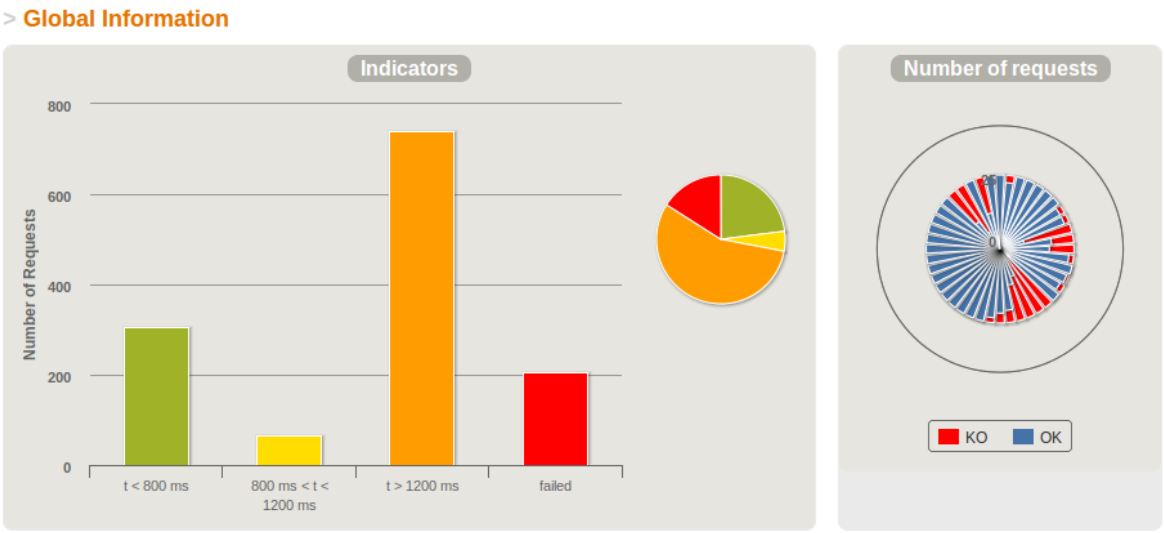


Figura 32 Load test Indicadores.
Fuente: Autor.
Elaborado por: El autor.

La Figura 32 muestra los tiempos de respuesta del servidor distribuidos en rangos de tiempo, la mayoría de las peticiones están en un tiempo de respuesta superior a los 1.2 segundos, en inicio la carga del servidor es mínima y las peticiones son respondidas en un tiempo inferior a los 800ms.

De las 1320 peticiones el 16% (207) fallaron, al inspeccionar en mayor detalle los resultados ofrecidos por gatling se encontró que estas correspondían a las peticiones de librerías como jquery.

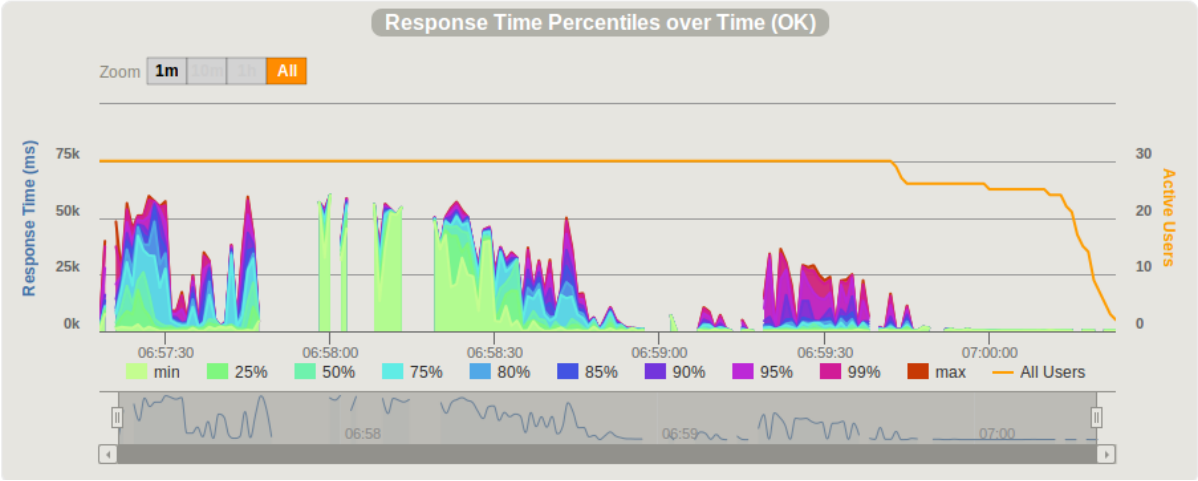


Figura 33 Load Test RTPT – Response Time Percentiles over time.
Fuente: Autor.
Elaborado por: El autor.

La Figura 33 muestra cómo se comporta la aplicación con una carga sostenida durante un intervalo de 6 minutos con 30 usuarios conectados simultáneamente, los elementos del conjunto estadístico son solo de respuestas 200 (OK) y derivadas del servidor.
La variedad de tiempos de respuesta indica que con 30 usuarios la aplicación tiene un tiempo de respuesta inferior a los 65k ms.

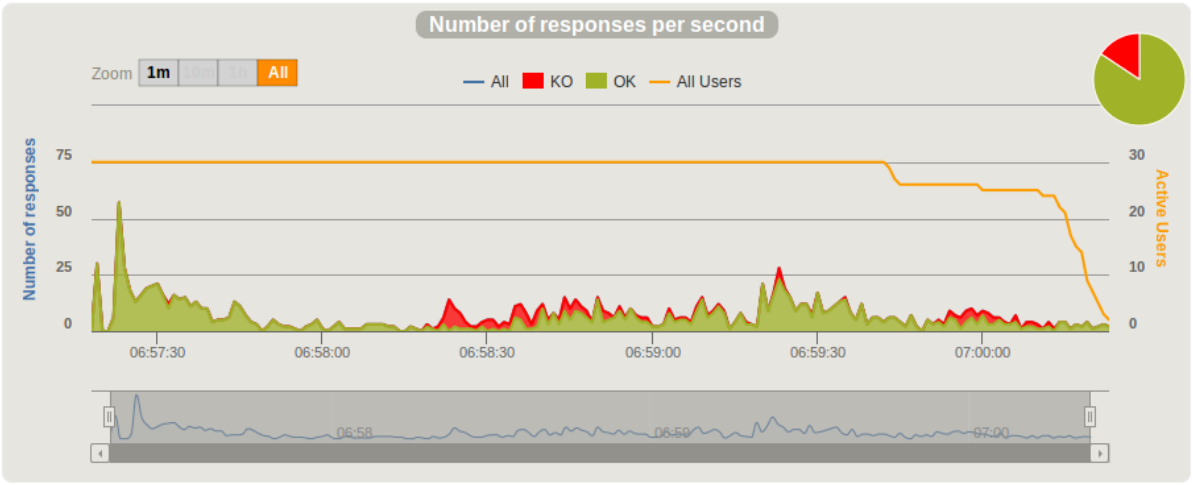


Figura 34: Load test RPS – Request per second
Fuente: Autor.
Elaborado por: El autor.

La Figura 34 muestra el número de respuestas recibidas sobre el tiempo, durante los primeros bloque tiempo las peticiones fueron 57, las primeras pruebas indican que todas las respuestas retornaron un código ok, sin embargo en el tercer periodo de tiempo, los fallos

empezaron a presentarse, la razón puede ser debido a que Heroku, pone restricciones de recursos a las cuentas gratuitas.

4.6.2. Pruebas monitor.

La ejecución de pruebas de monitor se llevó a cabo utilizando Junit, este es un conjunto de bibliotecas diseñadas para ejecutar pruebas sobre el código desarrollado, en este proyecto es utilizado para determinar si los componentes tienen un resultado adecuado.

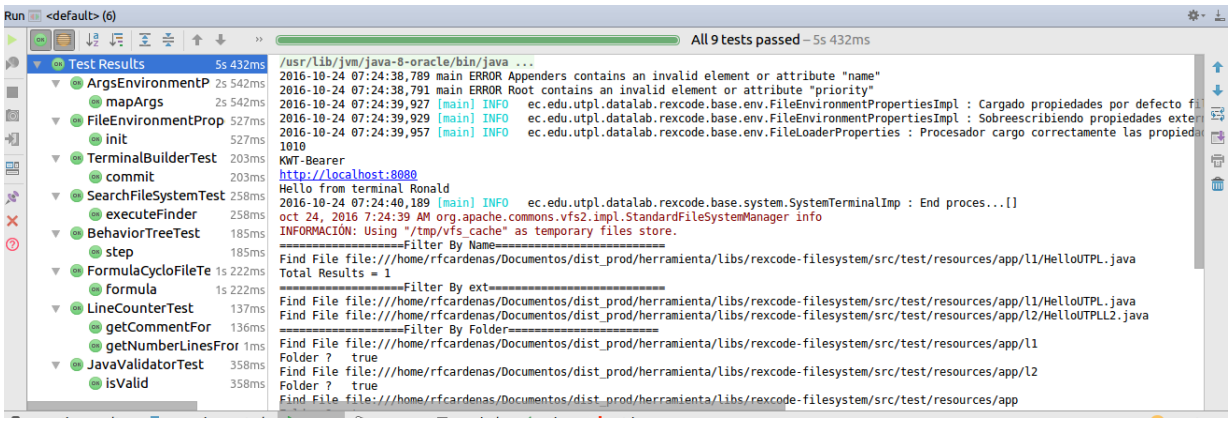


Figura 35 Test monitor.

Figura 35 expone las pruebas de los componentes más importantes del monitor, estas pruebas indican cómo se comportan las partes importantes de forma individual, se evalúa cada parte con la finalidad de conocer si los resultados son los esperados.

Las pruebas a este nivel:

- Variables de ambiente.
- Sistema de archivos.
- Terminal de comandos.
- Formulas y métricas.
- Validadores de lenguaje.

4.7. Integración continúa.

En el ciclo de vida de este proyecto, la integración continua [18] forma parte esencial del desarrollo, siendo deseable que la comunidad aporte con el desarrollo de nuevas requisitos de funcionalidad o la corrección de posibles bugs.

La integración continua es una forma de hacer integraciones automáticas en un proyecto de manera frecuente, permitiendo detectar problemas tan pronto como se da la integración. Para lograr esto se utilizó GitLab-CI[19], Docker[20] y Heroku[21]. La ejecución de pruebas, compilación y empaquetado de aplicación puede ser configurada utilizando las variables de ambiente de gitlab.

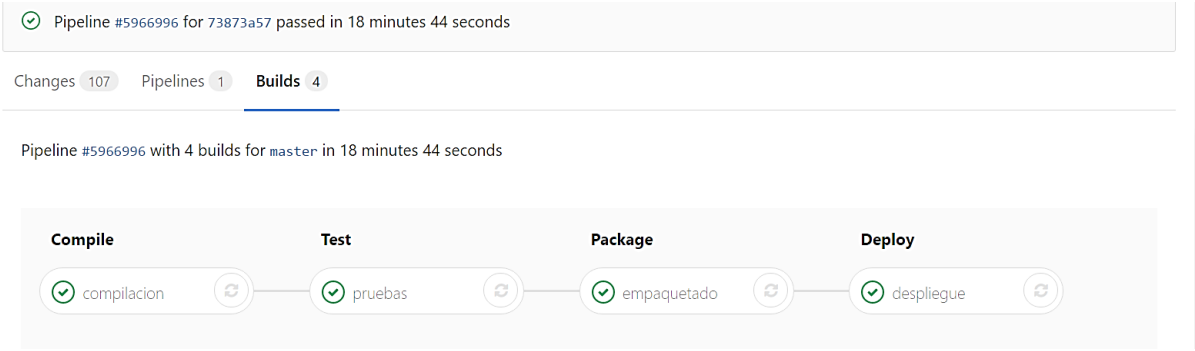


Figura 36 Integración continúa fases
Fuente: Autor.
Elaborador por: Autor.

Los artefactos son compilados, probados empaquetados y desplegados, las librerías están disponibles en un repositorio maven, alojado en gitlab.

La Figura 36 muestra como tan pronto cuando se realiza un commit en la rama master, gitlab empieza a ejecutar la compilación del proyecto, para esto baja la imagen en docker (Creada específicamente para este proyecto), una vez que la descarga, configura heroku para hacer un push de la aplicación y que los cambios se reflejen para los usuarios. Todo de forma automática.

[18] Martin Fowler: Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day.
[19] https://docs.gitlab.com/ce/ci/quick_start/README.html
[20] https://docs.docker.com/engine/getstarted/
[21] https://devcenter.heroku.com/articles/getting-started-with-java#introduction

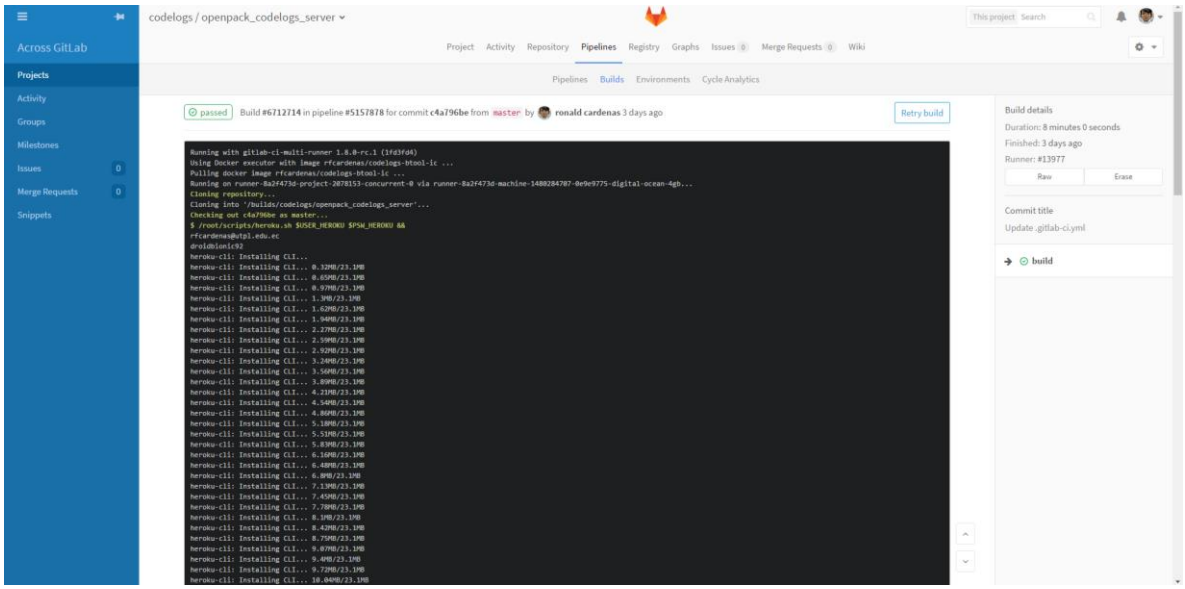


Figura 37. GitLab CI Codelogs
Fuente: Autor.
Elaborado por: El autor.

Las ventajas se resumen en:

- Cualquier cambio de código en la rama master del servidor es reflejado en la aplicación en tiempo real, para lograr esto gitlab-ci compila, empaqueta y ejecuta el server.war en heroku de forma automatizada.
- Control automático de las dependencias obsoletas.
- Minimización de tiempos de compilación, empaquetado y de carga de la aplicación a la plataforma heroku utilizando servidores dedicados para estas tareas.

4.8. Despliegue.

En esta etapa se contemplaron las siguientes actividades:

- Instalación y configuración del servidor en producción.
- Demostración de la utilidad del proyecto.

En este punto se busca que los interesados sean capaces de instalar y ejecutar el proyecto en diferentes ambientes, para esto el anexo 8 “Despliegue” proporciona una guía de como configurar y empaquetar la aplicación en diferentes entornos.

Existen dos modos de despliegue tanto a nivel del monitor como del servidor.

- Modo desarrollo.
- Modo producción.

4.8.1. Despliegue en heroku.

Para la demostración se ha hecho uso de heroku, gracias a la integracion continua implementada en este proyecto cualquier cambio efectuado en código es refleado para todos los usarios. Heroku nos proporciona un nodo gratuino asignandonos una IP y un puerto para que la aplicación pueda ser accedida desde cualquier lugar.

La Figura 38 muestra como la aplicación se ha levantado correctamente; sin embargo, el nodo tiene algunas limitaciones como: almacenamiento limitando a 10gb utilizando el complemento heroku postgres, debido a que la cuenta es gratuita.

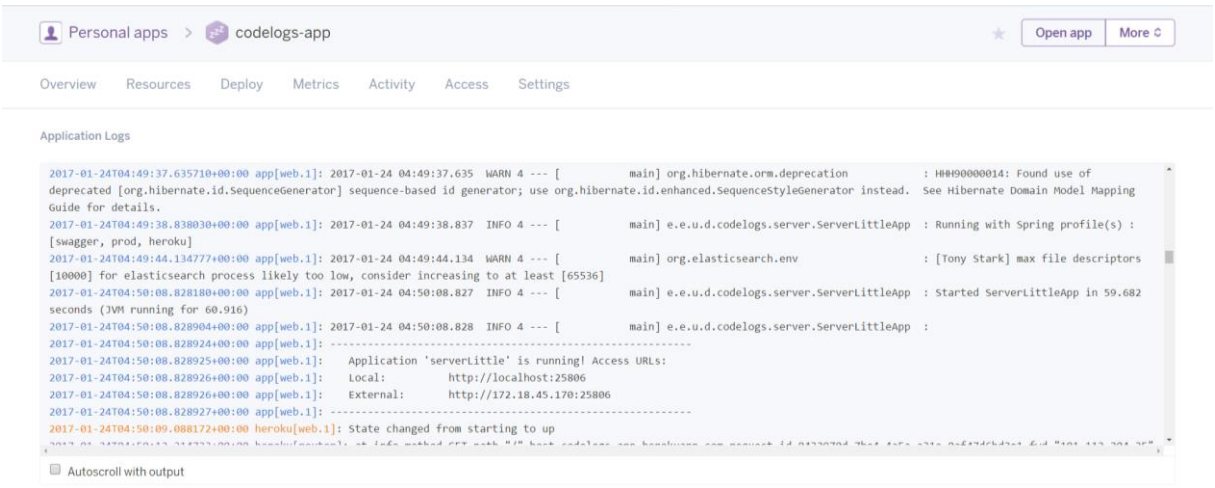


Figura 38. Despliegue en Heroku.
Fuente: El autor.
Elaboración: El autor.

La Figura 39 indica como un usuario puede generar un Token JWT para configurar su plugin.

Autenticación y autorización de usuarios.

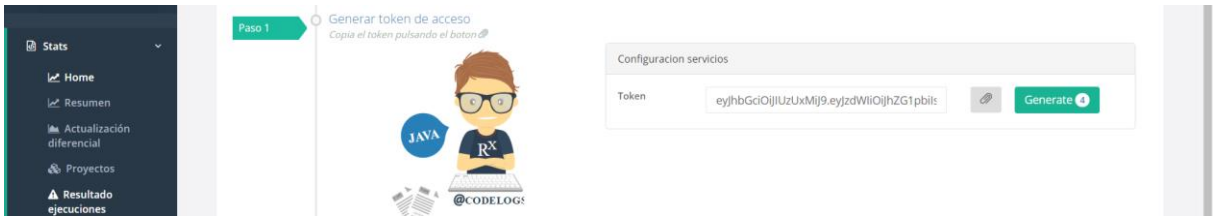


Figura 39. Generación de Tokens
Fuente: Autor.
Elaboración: El autor.

El primer punto que se cumplió es la autorización y autenticación de usuario, en esta pantalla el estudiante es capaz de generar el Token JWT para identificar al monitor, este Token le indica al servidor a que estudiante corresponde la data que se está sincronizando, no se requiere usuario ni contraseñas del lado del estudiante, simplemente se copia el token en las configuraciones del monitor y este tendrá acceso a las APIs de sincronización.

4.8.2. Visualización de datos:

En la Figura 40 se muestra algunos estadísticos, entre los más relevantes están las gráficas de líneas de tiempo y habilidades, se han definidos rangos basado en el tiempo tratando de seguir la idea que (Gladwell, 2012) expone en su libro “los fueras de serie”.

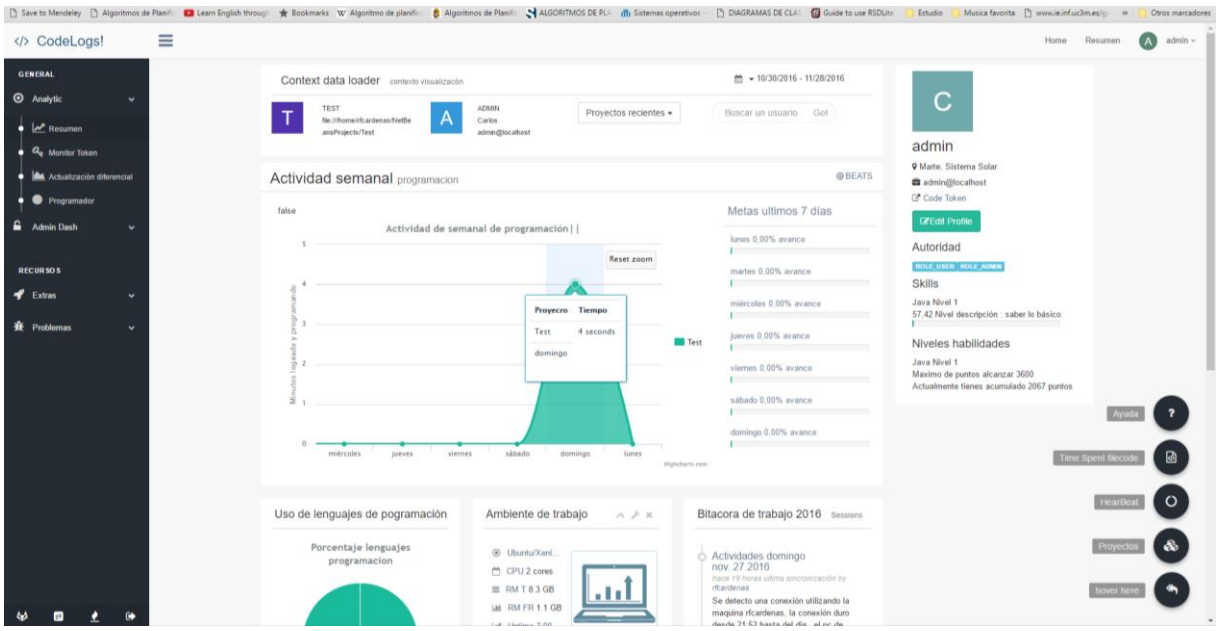


Figura 40. Visualización de estadísticos.
Fuente: El autor.
Elaboración: El autor.

Tabla 14. Niveles de habilidades.

Niveles de habilidades		
1 hora de programación.	Nivel 1 conocimiento básico	Color
10 horas de programación.	Nivel 2 un conocimiento más amplio	
100 horas de programación.	Nivel 3 un conocimiento de nivel medio.	
1000 horas de programación.	Nivel 4 conocimiento de especialista.	
10.000 horas de programación.	Nivel 5 Habilidades de tipo maestro.	

Fuente : Autor

Los rangos definidos en la Tabla 14 de habilidades son totalmente personalizables, por defecto los rangos en la base de datos son 5; no obstante estos niveles pueden extenderse. Las gráficas son generadas a partir de los últimos 7 días sobre el tiempo invertido en los diferentes proyectos, por ejemplo en la Figura 40 indica que el estudiante Carlos solamente programo 4 segundos el día domingo en un proyecto llamado test. La UI brinda de posibilidad de escoger entre rango de fechas.

4.8.3. Buscador de estudiantes:

Se ha implementado un buscador de estudiantes utilizando Elastic search, las búsquedas son ejecutadas utilizando los campos de:

- Username, first name , last name , email

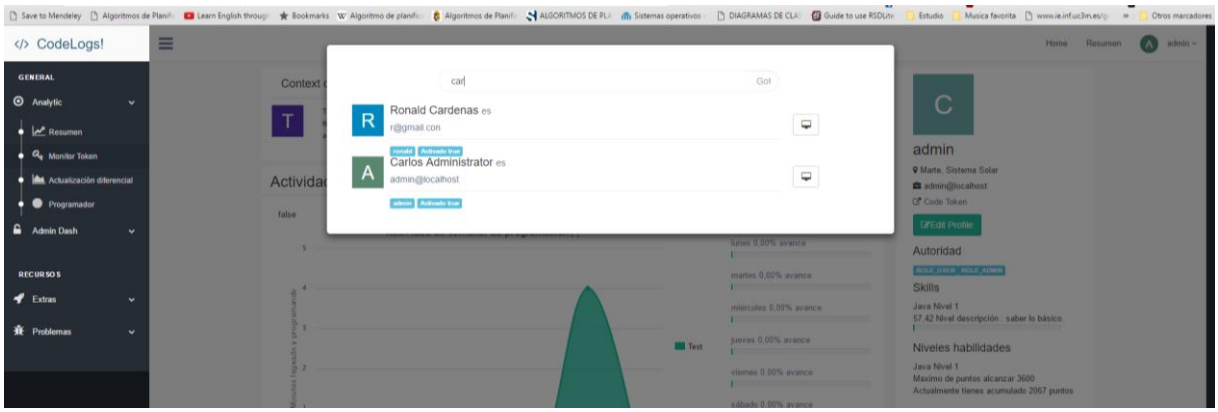


Figura 41. Buscador de estudiantes.
Fuente: El autor.
Elaboración: El autor.

La Figura 42 representa un extracto de las API REST disponibles para el consumo. A continuación se presenta el ejemplo del uso del API.

4.8.4. Consumo de APIS.

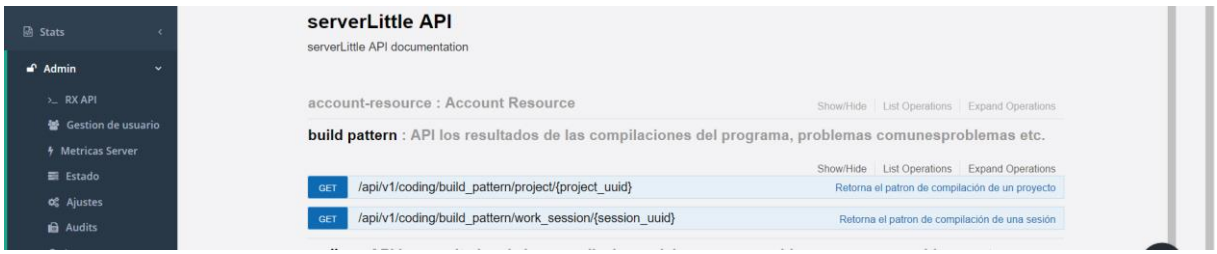


Figura 42: Extracto de APIS REST disponibles.
Fuente: El autor.
Elaboración: El autor.

En la Figura 43 se muestra la forma de invocar las Apis del proyecto. Cualquier punto de acceso del servidor requiere de un Token JWT, por esta razón se añade el encabezado Autorización.+ JWT, en este ejemplo nosotros queremos saber qué días el estudiante se ha conectado a programar, por defecto el servidor interpreta que se está solicitando la data correspondiente al Token del encabezado de la petición; sin embargo se puede solicitar datos de un usuario en específico añadiendo algunas variables en la petición.

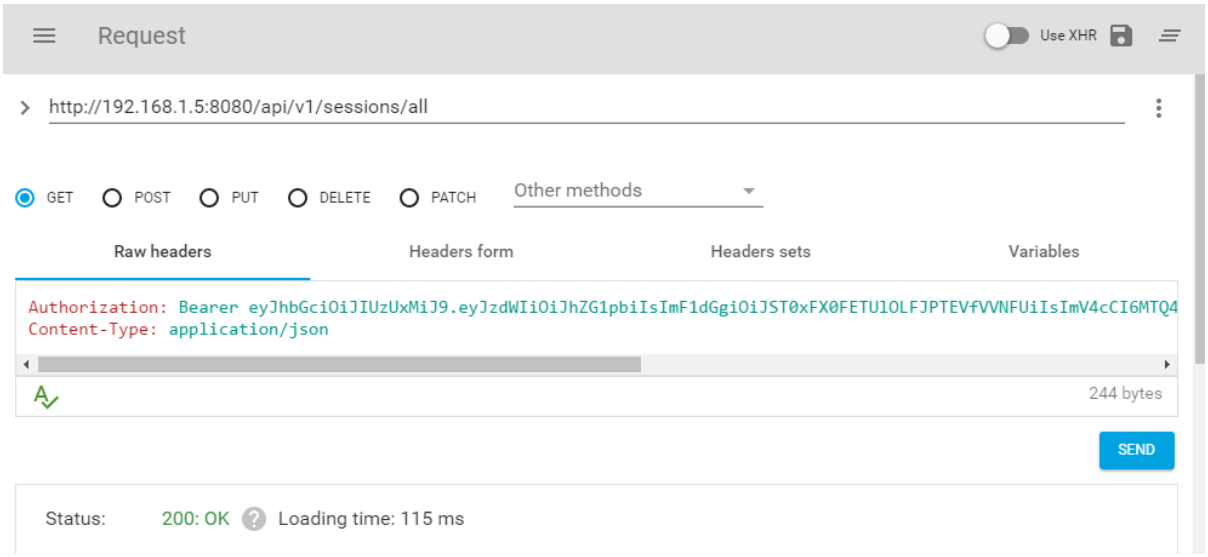


Figura 43: Petición GET API de sesiones.
Fuente: El autor.
Elaboración: El autor.

La ejecución de la operación anterior retorna un código 200, con la siguiente información en formato JSON. Este tipo de información es útil para seguimiento de la actividad.

Cabe recalcar que las sesiones de trabajo están fuertemente ligadas a un proyecto y son creados únicamente cuando el estudiante ha comenzado a modificar el código.

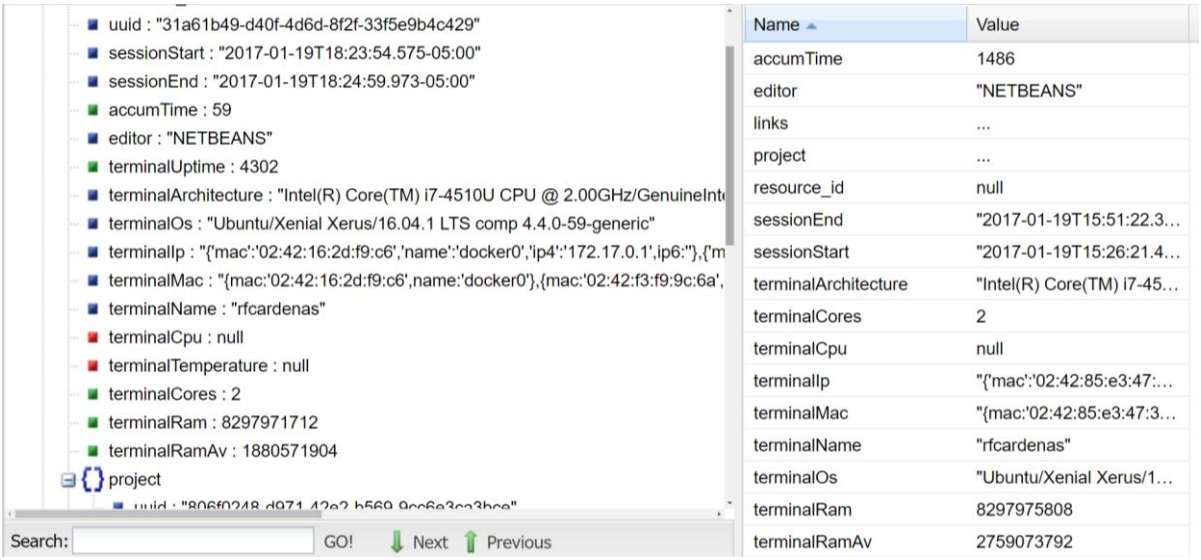


Figura 44: Respuesta petición GET API de sesiones.
Fuente: El autor.
Elaboración: El autor.

Patrón de comportamiento.

La Figura 45 representa el patrón de comportamiento en los diferentes segmentos, esta grafica permite identificar de forma visual comportamientos extraños como copias de código.

En total se tiene tres patrones:

- Actualización.
- Eliminaciones.
- Distancia Leveshtein.

Patrón de comportamiento, es la implementación grafica del patrón descrito por (Blikstein et al., 2014)

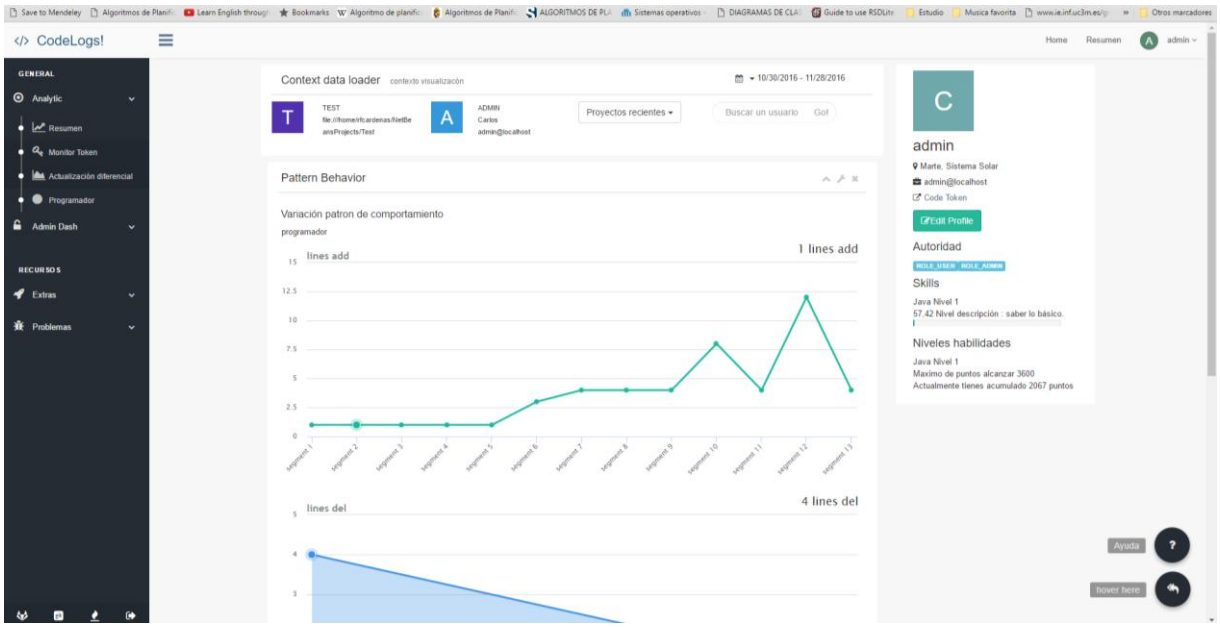


Figura 45: Patrón de actualización
Fuente: El autor.
Elaboración: El autor.

Problemas encontrados.

La Figura 46 muestra los problemas encontrados a lo largo del desarrollo de un proyecto, conforme el estudiante hace cambios, compila y ejecuta se recolectan estos resultados, es aquí donde intervienen las extensiones maven descritos en las secciones anteriores.

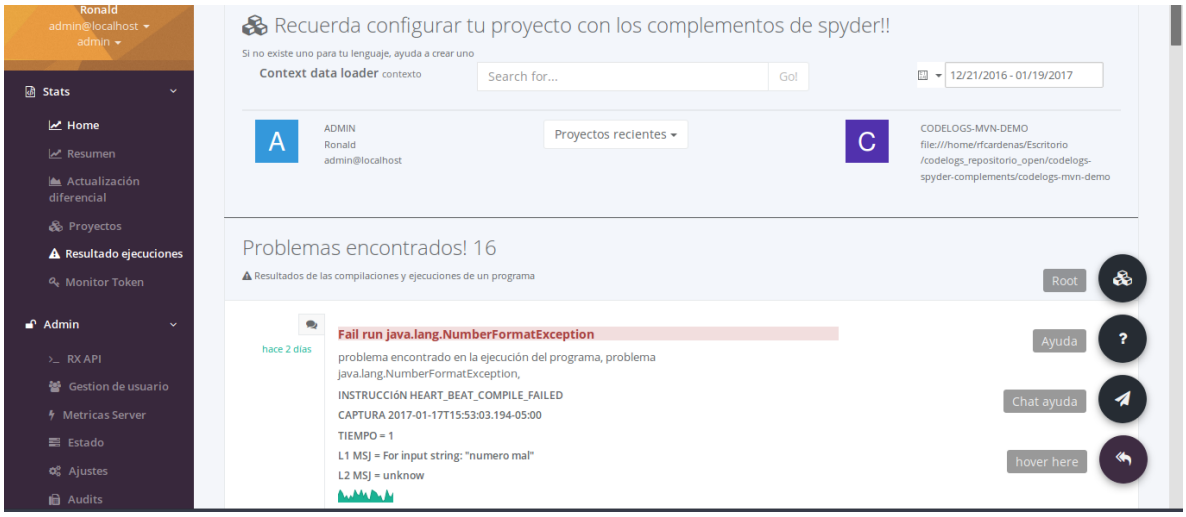


Figura 46: Problemas encontrados.
Fuente: El autor.
Elaboración: El autor.

Tiempo invertido

La Figura 47 muestra una gráfica acerca de los estados del programador:

- Programación Activa (tiempo que se mantuvo programando y haciendo modificaciones).
- Programación Inactiva (tiempo que esto distraído o haciendo otras cosas, menos programación).

También se proporciona información sobre el tiempo invertido en los archivos de código, este tipo de visualización permite identificar en que parte el estudiante ha dedicado la mayor parte de sus recursos (tiempo) al momento de desarrollar sus tareas.

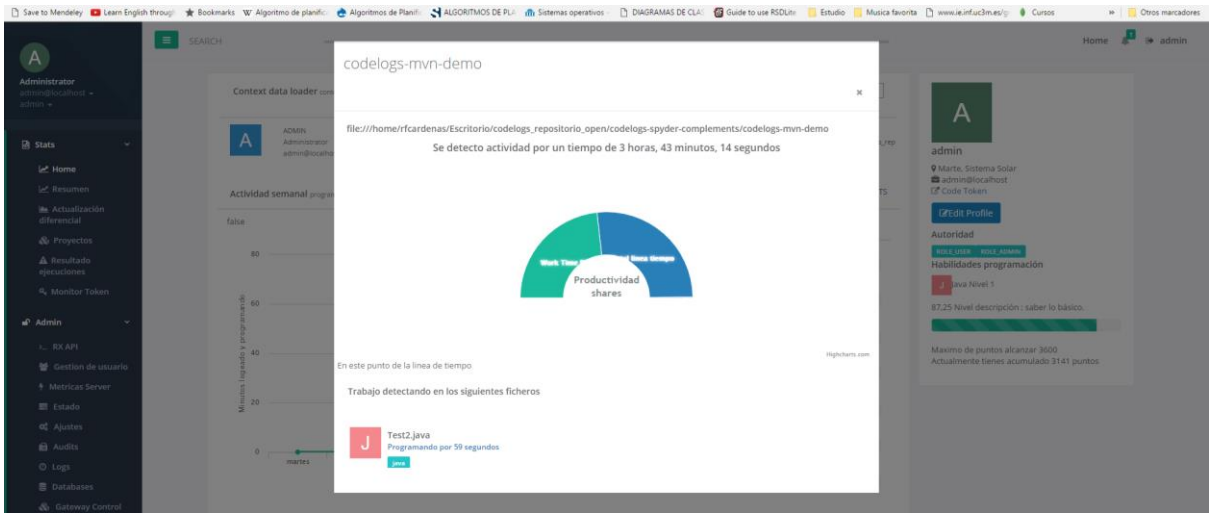


Figura 47 Detalle de un intervalo de actividad.
Fuente: El autor.
Elaboración: El autor.

Análisis de código.

Los datos capturados producto del análisis estático de código pueden ser consumidos mediante llamadas REST, como ya se explicó anteriormente se puede combinar servicios para generar graficas complejas.

La Figura 48 muestra la respuesta de este servicio, para ello se ha utilizado la interfaz swagger que permite testear las APIS.

Figura 48: Análisis de código.
Fuente: El autor.
Elaboración: El autor.

La Figura 49 muestra como al abrir Netbeans el IDE se conecta automáticamente con el proceso Spyder - Monitor, el cual es el encargado de recortar datos y sincronizarlos.

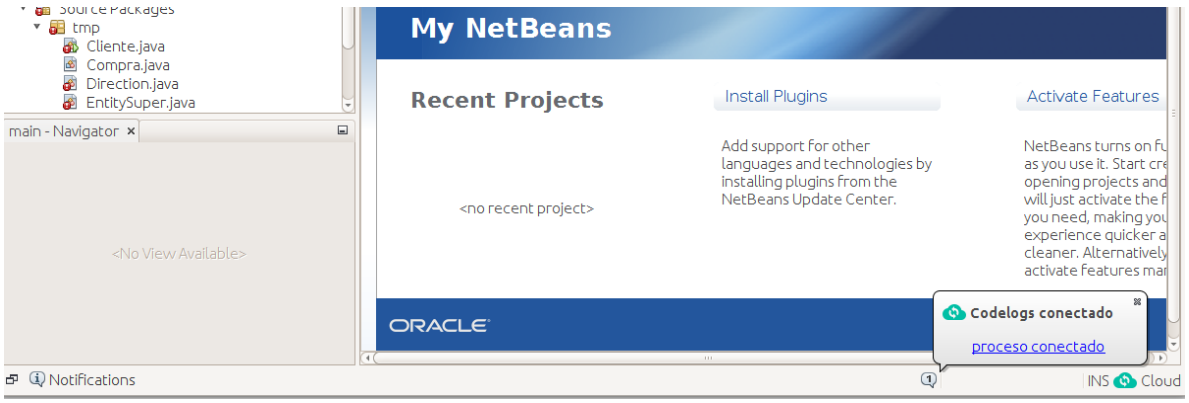


Figura 49: Netbeans conectado.
Fuente: El autor.
Elaboración: El autor.

El proceso puede ser levantando de varias maneras, a continuación se muestra la Shell, la cual permite también levantar un proceso monitor sin la necesidad de un IDE, en la Figura 50 se ilustra la Shell.

Cabe recalcar que si un estudiante programa en block de notas o cualquier otro editor, la actividad igualmente es captada debido a que se basa en los eventos del sistema de archivos del sistema operativo (Os).

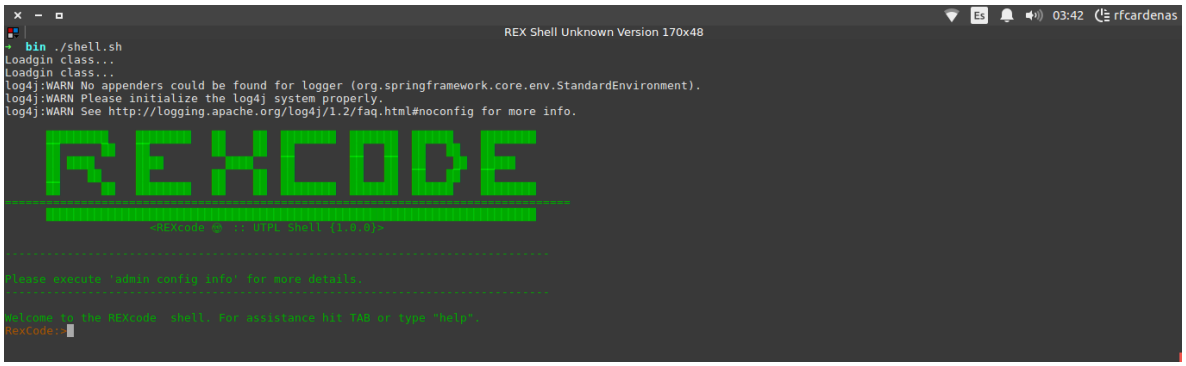


Figura 50: Servicio codelogs.
Fuente: El autor.
Elaboración: El autor.

La Siguiente figura el ilustra cómo trabaja el monitor internamente, primeramente crea la capa de persistencia, instala el soporte de archivos y crea la tuberías para el procesamiento de los eventos, una vez que esto se cumple esto, el monitor iníciala los gestores de eventos de tiempo y de sistema de archivos.

Para fines de demostración se habilitado la depuración en todos los niveles.

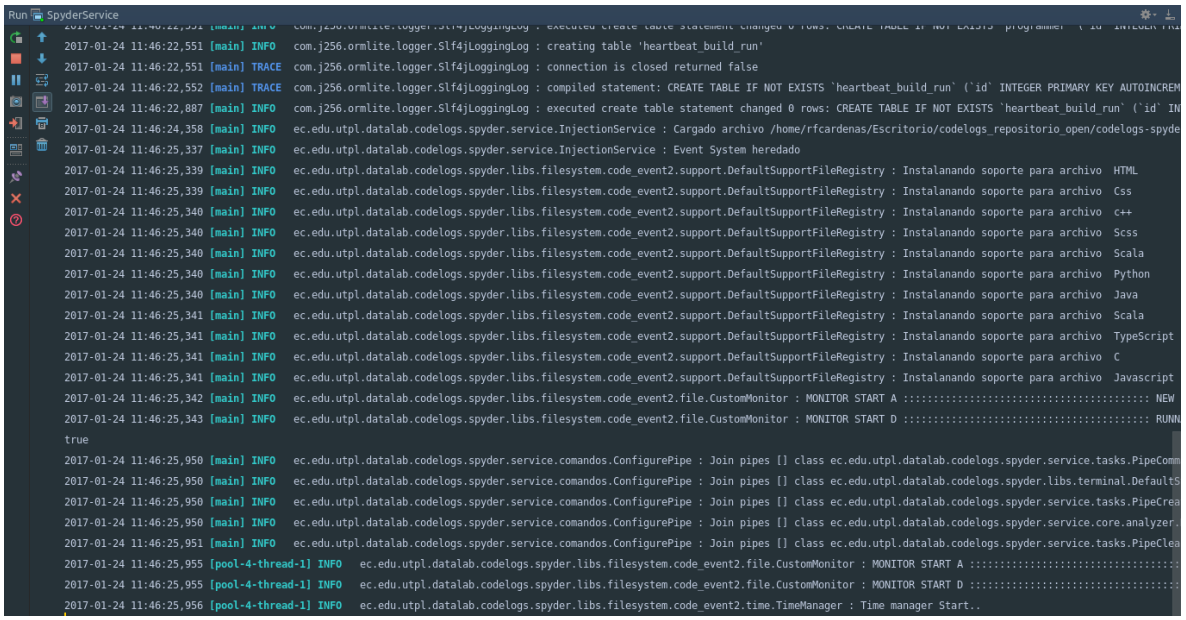


Figura 51: Demostración del proceso monitor.
Fuente: El autor.

CONCLUSIONES.

En este apartado se expone las conclusiones y los resultados obtenidos tras la finalización de del Trabajó de Titulación.

- Con el desarrollo de este proyecto se ha identificado y definido un conjunto de datos y métricas útiles para el análisis del aprendizaje enfocados en los cursos de programación, cabe recalcar que las investigaciones que han servido de base para este proyecto han sido llevadas por Instituciones de renombre a nivel global, los programas desarrollados en este trabajo se basan en las características identificadas en los proyectos citados.

- El dar seguimiento a los estudiantes en cursos de programación es una tarea compleja, debido a la falta de soporte y herramientas que faciliten esto al nivel descrito en este TT. El conjunto de datos formado en este proyecto está orientado al seguimiento de la actividad de un estudiante mientras codifica, proporciona los medios para automatizar la recolección de:
  - Problemas. recolecta datos sobre compilaciones y ejecuciones exitosas y fallidas de un programa, utilizando extensiones para núcleo de maven.
  - Evolución del código Se utiliza métricas como: líneas de código (Líneas añadidas, eliminadas), medida levenshtein para conocer cómo va creciendo el código del estudiante, estas son útiles para identificar patrones como copias de código, los estado del alumno como lo ha demostrado (Blikstein et al., 2014).
  - Comportamiento: Se utiliza mecanismos internos para recolectar datos sobre el tiempo, horarios de trabajo y sesiones garantizando que estos se recolecten en base a la actividad de codificación y no simplemente por el inicio o cierre una sesión.
  - Calidad de la solución: Se utiliza métricas como el índice mantenibilidad del código, radio de comentarios, métricas de tamaño, complejidad ciclomatica y conjunto de métricas Halstead para captar datos sobre la calidad de la solución.

- Las técnicas de modificación de bytecode de un programa java puede ser una tarea difícil sin las herramientas adecuadas, sin embargo este tipo técnicas ha sido de gran utilidad para capturar errores presentados en tiempo de ejecución de un programa. Datos que son útiles para aplicar algoritmos similares al descrito por (Matthew C., 2006). Además estos han sido objeto de investigaciones recientes para aplicar técnicas de

aprendizaje automático con la finalidad de predecir la tasa de confusión en grupos de estudiantes (Lee et al., 2011).

- El estudio de métricas de software por otra parte contribuye al dar conocer la amplia variedad de estas, los modelos matemáticos que las sustentan, su utilidad en el campo de la ingeniería del software y su utilidad en la educación.

- Investigadores como (Blikstein et al., 2014) afirman que: *el desarrollo nuevos métodos de recolección y análisis de datos podrían proporcionar nuevos caminos en el aprendizaje del estudiante*. Codelogs se presenta como una solución para recolección automática de datos, permitiendo el trabajo tanto offline como online ya que utiliza técnicas de sincronización de datos, independiente del lenguaje de programación y el entorno de desarrollo integrado, brinda la posibilidad de extender el comportamiento para analizar diferentes lenguajes de programación que se enseñan durante la carrera universitaria.

- Codelogs proporciona su propio servidor de datos, implementando una API Rest con la posibilidad de implementarlo sobre Cloud Heroku, gracias a su poco consumo de memoria inferior a los 512 RAM, sin embargo esto depende de la cantidad del tráfico generado por los programadores.

- Codelogs proporciona su propia Shell interactiva para monitorear el desarrollo de aplicaciones, brindando la posibilidad de ampliar el rango de IDE soportados e incluso los lenguajes de programación, esto es gracias a que ha sido diseñado tomando en cuenta temas de extensibilidad.

RECOMENDACIONES.

Tomando en cuenta los puntos más importantes de este proyecto, las dificultades y problemas encontrados se han formulado las siguientes recomendaciones de forma que minimice estos errores en proyectos similares o proyectos que busquen dar continuidad a esta TT.

- Si alguien está interesado en probar esta aplicación en entornos reales, en el repositorio del proyecto hay blog desarrollado con Hugo, este contiene algunas entradas para demostrar cómo implementar esta herramienta. Incluso se ha publicado algunos videos en YouTube para demostrar el funcionamiento del proyecto, cabe recalcar que hasta este punto de redacción de la tesis el proyecto está en la versión 1.0.0, una versión estable del proyecto.
  Si desea implementar es recomendable utilizar Linux o mac, en el caso de Windows el rendimiento del monitor del sistema de archivos no es óptimo debido a librerías externas, sin embargo si se dispone de un IDE, se puede reutilizar las API del editor.

- La amplia variedad de métricas en el desarrollo de software hace que sea complejo hacer un estudio profundo de cada una de ellas, debido a las restricciones de tiempo, ya que no solamente consiste en llevar un estudio sino también en la aplicación de estas para la recolección, por esta razón es recomendable de que futuros proyectos se tome en cuenta las métricas orientadas a objetos, que proporcionan información relevante en estas áreas y ayudarían a evaluar los conocimientos de nivel avanzados como, acoplamiento, cohesión, herencia, polimorfismo etc.

- Si desea ampliar el conjunto de métricas de código, se recomienda revisar algunos de los ejemplos disponibles en el repositorio del proyecto. Este proyecto tiene mucho que ofrecer y es deseable que haya una mejora continua del mismo, por ello también se recomienda utilizar las ventajas tecnológicas que ofrece GitLab como el soporte de integración continua, el cual permite automatizar varias de las fases del ciclo de vida de desarrollo, minimizando tiempos de espera para la construcción y deploy, ofreciendo retroalimentación sobre la calidad del software.

- Las métricas orientadas a objetos por su dificultad no pudieron implementarse, debido a que requieren modelos complejos de la representación de un programa, sin embargo para mitigar estos problemas codelogs está diseñado para que nuevos conjuntos de métricas puedan ser integrados ya que trabaja con comandos utilizando Json como intercambio y transferencia de datos.

- Al momento de ejecutar las pruebas a nivel de servidor se debe tener en cuenta que estas deben ser aplicadas en un entorno de producción, no tendría sentido ejecutar pruebas de carga cuando la aplicación está corriendo localmente. Además utilizar marcos de trabajo como Gatling, el cual proporciona métricas detalladas sobre el comportamiento del servidor

- Al momento de trabajar con docker y construir imágenes es preferible utilizar Dockerfiles junto con git para automatizar y acelerar el proceso de construcción y carga de las imágenes en el repositorio de docker. Además es recomendable utilizar variables de ambiente para minimizar la reconstrucción de las imágenes, lo cual puede consumir mucho tiempo.

TRABAJOS FUTUROS.

Este proyecto aún tiene mucho potencial y camino por recorrer, siempre se puede plantear nuevas metas a alcanzar, siendo deseable que más personas con intereses similares se integren y aporten con nuevas mejoras y características funcionales. La intención es que este proyecto siga creciendo y se convierta en algo genial que ayude a los programadores a optimizar sus recursos.

Actualmente hasta la fecha en cuanto a visualizaciones se han enfocado en datos relacionados con el tiempo (el recurso más valioso del hombre), sin embargo existen datos como las métricas de código que aún faltan por visualizar pero que se aclara los mecanismos de recolección están implementados.

Para trabajos futuros se ha considerado implementar lo siguiente:

- En el último análisis de la aplicación se encontró que los token de autorización basados en JWT consumen un mayor uso de recursos de red debido a la extensión del mismo, las ventajas de este son innegables, sin embargo repercute en el rendimiento de la red debido a que tiene mayor tamaño, para trabajos futuros se ha considerado implementar un nuevo token de autorización, que sea más ligero basado en UUID o algún método Hash.
- Trabajar en una versión personalizada de Swagger (Documentación Api REST), de forma que se adapte al diseño de la aplicación web.
- Añadir perfiles de usuarios de forma que pueda revisar rápidamente las habilidades del programador.
- Crear nuevos plugins para otros editores como sublime text, actualmente es relativamente fácil crear nuevos plugins simplemente se tiene que conectar con el Websockets del monitor y enviar los comandos necesarios descritos en los anexos.
- Proporcionar nuevas formas de registro y acceso a la aplicación, actualmente se soporta Google se pretende ampliar el uso de redes sociales como Facebook y Twitter.

BIBLIOGRAFÍA

Bennedsen, J., & Caspersen, M. E. (2007). Failure rates in introductory programming. *ACM SIGCSE Bulletin*, *39*(2), 32. https://doi.org/10.1145/1272848.1272879

Blikstein, P. (2011). Using learning analytics to assess students ' behavior in open-ended programming tasks. *Learning*, (November), 110–116. https://doi.org/10.1145/2090116.2090132

Blikstein, P., Worsley, M., Piech, C., Sahami, M., Cooper, S., & Koller, D. (2014). Programming Pluralism: Using Learning Analytics to Detect Patterns in the Learning of Computer Programming. *Journal of the Learning Sciences*, *23*(4), 561–599. https://doi.org/10.1080/10508406.2014.954750

Bock, H., Böck, A., Glick, J., & Konecny, D. (2012). *The definitive guide to NetBeans Platform 7*. Apress.

Cardell-Oliver, R. (2011). How can software metrics help novice programmers? *13th Australasian Computing Education Conference*, *114*(ACE 2011), 55–62. Retrieved from http://dl.acm.org/citation.cfm?id=2459943

Carter, A. S., Hundhausen, C. D., & Adesope, O. (2015). The Normalized Programming State Model. In *Proceedings of the eleventh annual International Conference on International Computing Education Research - ICER '15* (pp. 141–150). New York, New York, USA: ACM Press. https://doi.org/10.1145/2787622.2787710

Chidamber, S. R., & Kemerer, C. F. (1994). A metric Suit for Object Orient Desing. *Tse*.

Drachsler, H., Cooper, A., Hoel, T., Ferguson, R., Berg, A., Scheffel, M., … Chen, W. (2015, March 16). Ethical and privacy issues in the application of learning analytics. Retrieved from http://oro.open.ac.uk/42347/1/p390-drachsler.pdf

Garg, A. (2014). An approach for improving the concept of Cyclomatic Complexity for Object-Oriented Programming. Retrieved from http://arxiv.org/abs/1412.6216

Gladwell, M. (2012). Outliers: The Story of Success.

Gros Salvat, B. (2013, June 6). Retos y tendencias sobre el futuro de la investigación acerca del aprendizaje con tecnologías digitales. Servicio de Publicaciones, Universidad de

Murcia. Retrieved from http://diposit.ub.edu/dspace/handle/2445/44090

Guerrero, C. A., Suárez, J. M., & Gutiérrez, L. E. (2013). Patrones de Diseño GOF (The Gang of Four) en el contexto de Procesos de Desarrollo de Aplicaciones Orientadas a la Web. *Información Tecnológica*, *24*(3), 103–114. https://doi.org/10.4067/S0718-07642013000300012

Gurrin, C., Smeaton, A. F., & Doherty, A. R. (2014). Lifelogging: Personal big data. *Foundations and Trends in Information Retrieval*, *8*(1), 1–125.

Kasto, N., & Whalley, J. (2013). Measuring the difficulty of code comprehension tasks using software metrics, 59–65. Retrieved from http://dl.acm.org/citation.cfm?id=2667199.2667206

Kruchten, P. (1995). Architectural Blueprints—The " 4+1 " View Model of Software Architecture. *IEEE Software*, *12*(6), 42–50.

Lee, D. M. C., Rodrigo, M. M. T., Baker, R. S. J. D., Sugay, J. O., & Coronel, A. (2011). Exploring the Relationship Between Novice Programmer Confusion and Achievement. In *Proceedings of the 4th International Conference on Affective Computing and Intelligent Interaction - Volume Part I* (pp. 175–184). Berlin, Heidelberg: Springer-Verlag. Retrieved from http://dl.acm.org/citation.cfm?id=2062780.2062803

Martin, F., & Whitmer, J. C. (2016). Applying Learning Analytics to Investigate Timed Release in Online Learning. *Technology, Knowledge and Learning*, *21*(1), 59–74. https://doi.org/10.1007/s10758-015-9261-9

Matthew C., J. (2006). Methods and tools for exploring novice compilation behaviour. In *Proceedings of the 2006 international workshop on Computing education research - ICER '06* (p. 73). New York, New York, USA: ACM Press. https://doi.org/10.1145/1151588.1151600

Matthew C, J., & Dorn, B. (2015). Aggregate Compilation Behavior. In *Proceedings of the eleventh annual International Conference on International Computing Education Research - ICER '15* (pp. 131–139). New York, New York, USA: ACM Press. https://doi.org/10.1145/2787622.2787718

McCabe, T. J., Watson, A. H., & Wallace, D. R. (1996). Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric. *NIST Special Publication*, *500*(235), 1–114. https://doi.org/800.638.6316

Mccormick, Z., & Schmidt, D. C. (n.d.). DATA SYNCHRONIZATION PATTERNS IN MOBILE

APPLICATION DESIGN.

Mogollón Afanador, J., & Alberto Esteban Villamizar, L. (2011). INDIVIDUAL WORK DEVELOPMENT OF SOFTWARE PROJECTS: A REALITY WITHOUT METHOD EL DESARROLLO INDIVIDUAL DE PROYECTOS DE SOFTWARE: UNA REALIDAD SIN MÉTODO.

Pérez, R. M. (2015). Learning Analytics: impacto, alcance, buenas prácticas y tecnologías. Retrieved November 11, 2015, from http://www.americalearningmedia.com/edicion-038/428-tendencias/6429-learning-analytics-impacto-alcance-buenas-practicas-y-tecnologias

Pressman, R. S., & Ph, D. (2007). *Ingeniería del software, 7ma Ed*.

Reynolds, R. G. (2002). *Software Development Metrics*. *Encyclopedia of Software Engineering*. Retrieved from http://onlinelibrary.wiley.com/doi/10.1002/0471028959.sof601/full

Rosenberg, L., & Hyatt, L. (1997). Software quality metrics for object-oriented environments. *Crosstalk Journal, April*, *10*(4), 1–6. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.104.813&rep=rep1&type=pdf

Sommerville, I. (2012). *Ingeniería de software*. (luis M. Cruz Castillo, Ed.) (9th ed.). Pearson Educación.

Urma, R.-G., & Mycroft, A. (2015). Source-code queries with graph databases—with application to programming language usage and evolution. *Science of Computer Programming*, *97*(P1), 127–134. https://doi.org/10.1016/j.scico.2013.11.010

Vahdat, M., Oneto, L., Anguita, D., Funk, M., & Rauterberg, M. (2015). A Learning Analytics Approach to Correlate the Academic Achievements of Students with Interaction Data from an Educational Simulator. In *Design for Teaching and Learning in a Networked World* (pp. 352–366). Springer.

Watson, C., & Li, F. W. B. (2014). Failure rates in introductory programming revisited. In *Proceedings of the 2014 conference on Innovation & technology in computer science education - ITiCSE '14* (pp. 39–44). New York, New York, USA: ACM Press. https://doi.org/10.1145/2591708.2591749

Watson, C., Li, F. W. B., & Godwin, J. L. (2013). Predicting Performance in an Introductory Programming Course by Logging and Analyzing Student Programming Behavior. In *2013 IEEE 13th International Conference on Advanced Learning Technologies* (pp.

319–323). IEEE. https://doi.org/10.1109/ICALT.2013.99

GLOSARIO DE TÉRMINOS

A

Arquitectura de la aplicación
Nivel de diseño enfocado en aspectos "más allá de los algoritmos y estructuras de datos de la computación 62

C

CODELOGS
Nombre clave otorgado al prooyecto 18, 54, 55

compilaciones
Proceso de traducción de código fuente (escrito en un lenguaje de programación de alto nivel) a lenguaje máquina (código objeto) para que pueda ser ejecutado por la computadora. 25

F

fork/join
Framework implementación de executor Service que ayuda a tomar ventaja de múltiples procesadores, está diseñado para trabajos que pueden dividirse en pequeñas piezas de forma recursiva. (“Fork/Join (The JavaTM Tutorials > Essential Classes &gt 69

G

gatling
Framework de codigo abierto para ejecutar pruebas de carga en un servidor. 78

gitlab
Aplicación opensource que nos permite administrar repositorios en git mediante una interfaz web 82, 83, 135

H

Heroku
Heroku es una plataforma como servicio de computación en la Nube que soporta distintos lenguajes de programación. 78, 81, 82, 95, 106, 133

I

IDE
Entorno de desarrollo integrado, es un entorno de programación empaquetado como un programa, el cual integra herramientas útiles para un programador. 19

J

JSON
Acrónimo de JavaScript Object Notation, es un formato de texto ligero para el intercambio de datos. JSON es un subconjunto de la notación literal de objetos de JavaScript aunque hoy, debido a su amplia adopción como alternativa a XML 61, 67, 88

L

logs
Un log es un registro de actividad de un sistema, que generalmente se guarda en un fichero de texto, al que se le van añadiendo líneas a medida que se realizan acciones sobre el sistema. 74

M

maven
Es una herramienta de software para la gestión y construcción de proyectos Java creada por Jason van Zyl, de Sonatype, en 2002. 73, 74, 75, 82, 94, 135

metodologías
Marcos de trabajo usado para estructurar, planificar y controlar el proceso de desarrollo 55

métricas **12, 14, 15, 17, 18, 19, 21, 23, 25, 26, 27, 28, 29, 31, 32, 35, 36, 37, 38, 39, 41, 42, 47, 49, 51, 62, 69, 70, 72, 73, 81, 94, 95, 96, 97, 104, 111, 112, 115, 120, 124**

O

Os **92**

P

plugin
Una aplicación que aporta nueva funcionalidad a un programa existente, interactuando por medio de una API. ii, iii, 14, 18, 35, 36, 37, 38, 66, 74, 77, 106, 109, 129, 130, 131

R

requisitos
Condición o capacidad que un usuario necesita para poder resolver un problema o lograr un objetivo (IEEE). 18, 56, 59, 70, 82, 108, 112

REST **19, 63, 98**

S

SAD
Documento de descripción arquitectonica. 106
Documento de descripción arquitectónica. 62

T

TT
Sigla que hace referencia al trabajo de titulación. 37, 38, 52, 56, 96

ANEXOS

ANEXO 1. TABLA DE MÉTRICAS IDENTIFICADAS.

Este anexo resume las métricas identificadas durante las fases de investigación.

PR – Proyecto PK – Paquete C – Clase M – Métodos

Resumen de métricas						
Tipo	Métricas	Detalles	PR	PK	C	M
Métricas elementales	Líneas de código (LOC)	Métrica orientada al tamaño se relaciona con la complejidad, esta métrica es una de las principales usadas dentro Learning analytic (revisión proyectos existentes).	x	x	x	x
	Líneas de comentarios (CLOC)	Métrica cuantifica la cantidad de líneas comentadas en un proyecto	x		x	x
	Duplicaciones del código	Proporciona información sobre el código duplicado.	x			
	Densidad de comentarios (DC)	Muestra que tan bien comentado esta un programa.	x			
	Numero de métodos	Numero de métodos es una métrica para calcular el promedio de todas las operaciones por clase			x	
	Numero de paquetes.	Es una medida del número de paquetes conforman un programa está estructurado	x			
	Numero de atributos (NOA)	Es una medida de complejidad, es el número de métodos públicos o protected en una clase				x
	Complejidad ciclomatica.	Esta métrica permite medir el código respecto a la dificultad de mantener sin embargo estudios llevados por (Whalley & Kasto, 2014) la han utilizado para identificar la complejidad y compresión del código.	x			x
	Acoplamiento entre objetos (CBO)	Lo que intenta medir es el grado de acoplamiento, dependencias que una clase tiene para su correcto funcionamiento.			x	
Métricas Chidamber & Kemerer	Métodos ponderados por clase (WMC)	Esta métrica brinda un punto de vista de la complejidad de clase, también puede ser usada para medir el nivel de reutilización de clase.			x	
	Falta de cohesión de métodos (LCOM)	Esta métrica ha sido fuertemente criticada, por ello se han propuesto algunas variantes en total 5, intenta medir la cohesión de una clase, entre más al valor menos cohesión tiene una clase.			x	
	Profundidad en el árbol de herencia (DIT)	Permite medir la complejidad de una clase, tomando como punto de referencia el nivel de profundidad (herencia).			x	
	Respuesta de una clase (RFC)	Es una métrica de acoplamiento, también mide la complejidad de una clase basada en el número de métodos ejecutados en respuesta a una acción.			x	
	Número de hijos (NOC)	Cuenta el número de decientes de una clase.			x	

	Esfuerzo de desarrollo (E)	Estima el esfuerzo de desarrollo de un programa				
Métricas Halstead	Longitud (N)				x	
	Vocabulario (n)				x	
	Bugs (B)				x	
	Volumen (V)				x	

ANEXO 2: HERRAMIENTAS Y LIBRERÍAS UTILIZADAS.

Tabla 15 Herramientas utilizadas en el proyecto.

Etapa por iteración	Herramienta / artefacto	Versión	Detalles
Análisis y diseño	Enterprise Architect	9	Modelado de diagramas UML
	Modelio	3.5	Herramienta de diagramado y generación de código
Desarrollo	Maven	3.3.3	Principal herramienta de desarrollo, presente durante todo el ciclo.
	Java	JDK_8_U 55	Lenguaje de programación
	IDE IntellJ Idea	15 – 2016.2	IDE Usado para el desarrollo de la herramienta independiente del lenguaje y el Servidor de datos.
	IDE Netbeans	8.1	IDE desarrollo del plugin pruebas e instalación.
	H2 base	1.4	Base de datos utilizada en la etapa de desarrollo del servidor.
	SQlite	3	Base de datos embebida, almacenamiento local de datos.
	Postgresql	9.6	Base de datos utilizada en el servidor
	LiquidBase	3.5	Versionamiento de base de datos, mejoras en cada iteración.
	Jhipster	3.6.5	Stack tecnología para el lado del servidor incluye: ▪ Spring Boot ▪ Angularjs. ▪ Hazelcast
Pruebas calidad	Junit	4	Proporciona
Deploy	Heroku Cloud		Deploy de la aplicación en el servidor
	Netbeans		
Fuete : Autor			

Anexo 3. Documento de descripción arquitectónica SAD.

DESARROLLO DE SOFTWARE

PLUGIN DE RECOLECCIÓN DE DATOS SOBRE EL COMPORTAMIENTO DE ESTUDIANTES DE PROGRAMACIÓN.

Documento de Arquitectura de Software

Versión 1.0

Historial de Revisiones

Actualización.	Versión	Descripción	Autor
23/06/2016	1.0	Descripción arquitectónica de la solución.	Cárdenas Cabrera Ronald Francisco.

En la arquitectura de software, el diseño es tratado como un punto en el cual la aplicación es diseñada tomando un enfoque de alto nivel, dando como resultado la identificación y diseño de varios componentes que satisfacen la funcionalidad y requisitos del sistema tanto funcionales como no funcionales.

Puesto que es muy complejo capturar todo el diseño de software en un solo diagrama, (Kruchten, 1995) propone un modelo de representación arquitectónico denominado 4 + 1, dividiendo las áreas de interés en 5 partes.

I. Vista lógica
II. Vista de procesos
III. Vista de desarrollo
IV. Vista física
V. Casos de uso

Propósito

Este documento tiene como finalidad dar a conocer la arquitectura de sistema a desarrollar, este documento permite tener visión tanto al cliente como al desarrollador de las partes críticas y forma parte fundamental de línea base de desarrollo.

Alcance

En este documento se presenta la arquitectura de software a implementar, identificando los componentes principales que forman parte de la solución, así mismo identificando la criticidad y prioridad para el desarrollo de estos.

Arquitectura.

El siguiente diagrama proporciona una visión global de los componentes físicos y de software implementados en el proyecto.

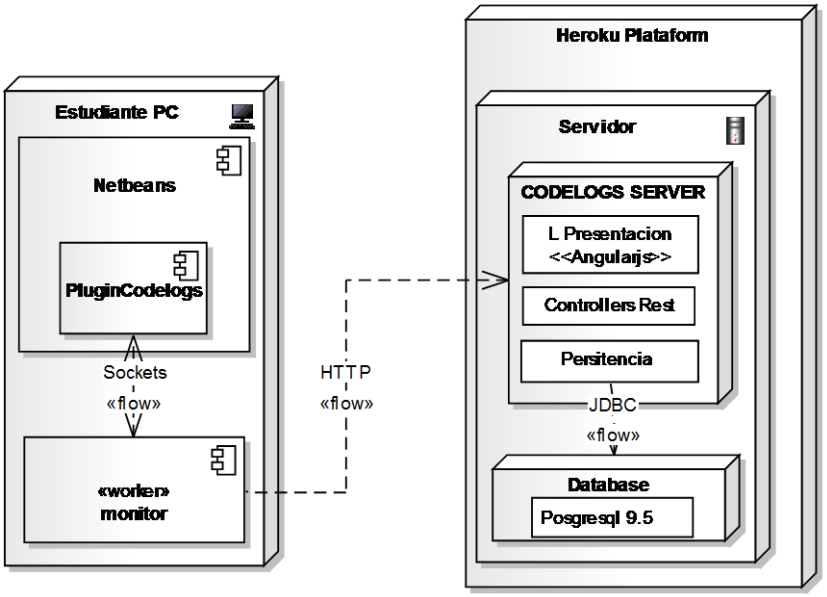


Figura 52 Vista de implementación.

Fuente: Autor.

Para los fines del proyecto se ha considerado implementar una arquitectura cliente/ servidor, también conocida como arquitectura de dos niveles, en el nivel 1 (cliente) se tiene el monitor de trabajo, este programa es el encargado de recolectar información de las activadas del estudiante al programar un asignamiento.

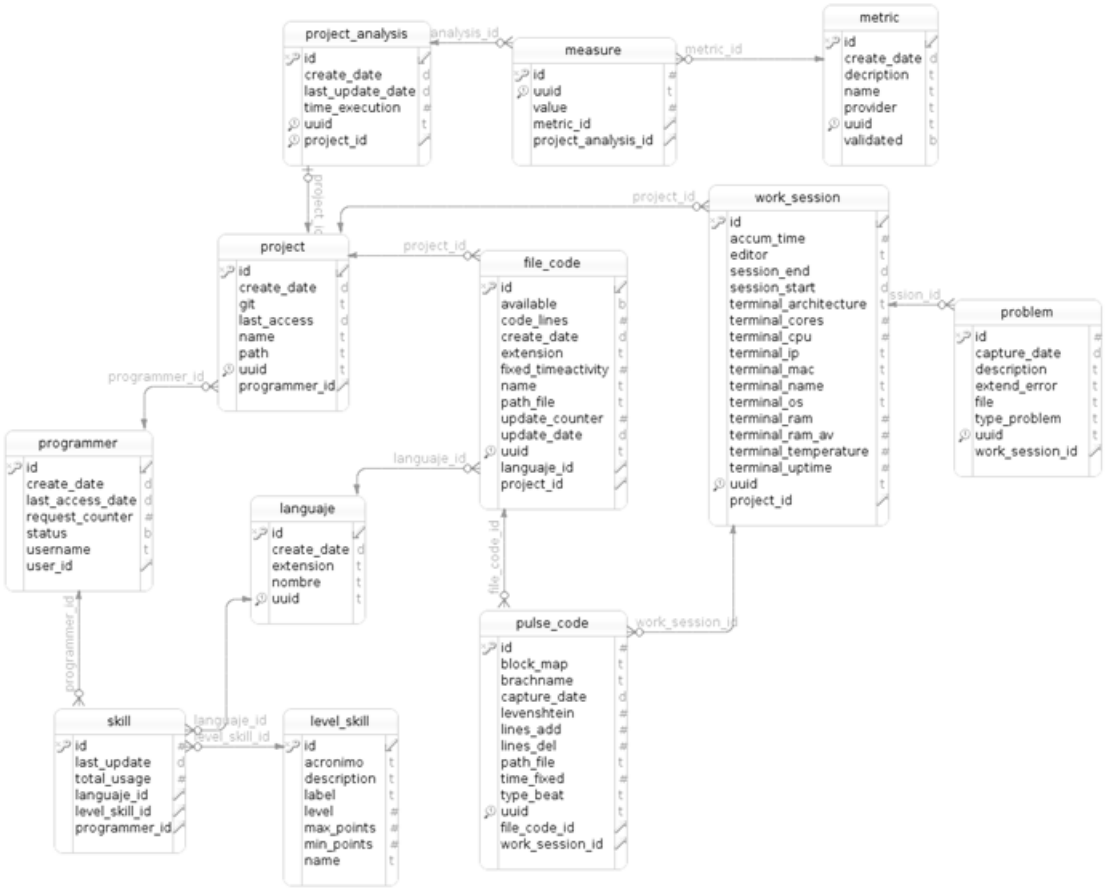
El plugin el cual es instalado en Netbeans, sirve como un enlace entre el monitor y el IDE, siendo dos procesos separados el método de comunicación entre estos más factible son los sockets, los cuales trabajan bajo un protocolo TCP.

En el nivel 2 se tiene el servidor, el servidor es el encargado de receptar las peticiones procedentes del monitor, la mayor parte de estas peticiones son métodos POST y PUT, para crear y actualizar contenido en el servidor.

Vista lógica

Esta vista describe la estructura del sistema, a través de diagramas de clase, Diagramas Entidad relación.

Diagrama ER



Vista de desarrollo

La vista de desarrollo, es la vista más útil para el programador se enfoca en la organización de los módulos software en el entorno de desarrollo. El software es empaquetado en pequeños trozos (librerías de programa, subsistemas, componentes, etc). En esta visa se ha creído conveniente utilizar el diagrama de componentes para proporcionar una visión global del desarrollo.

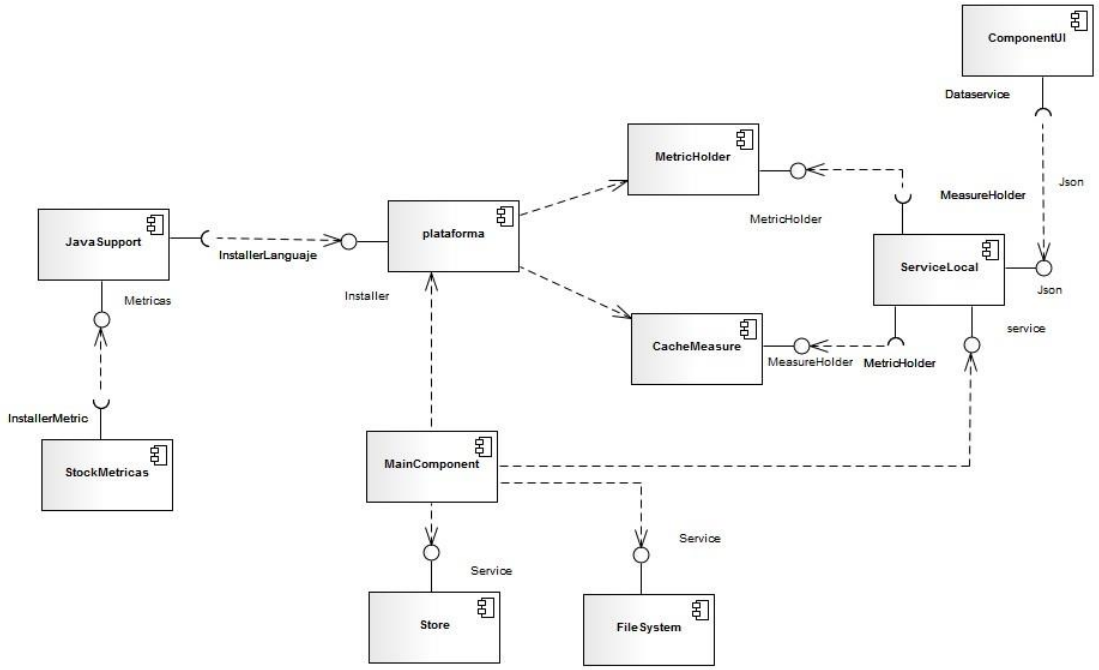


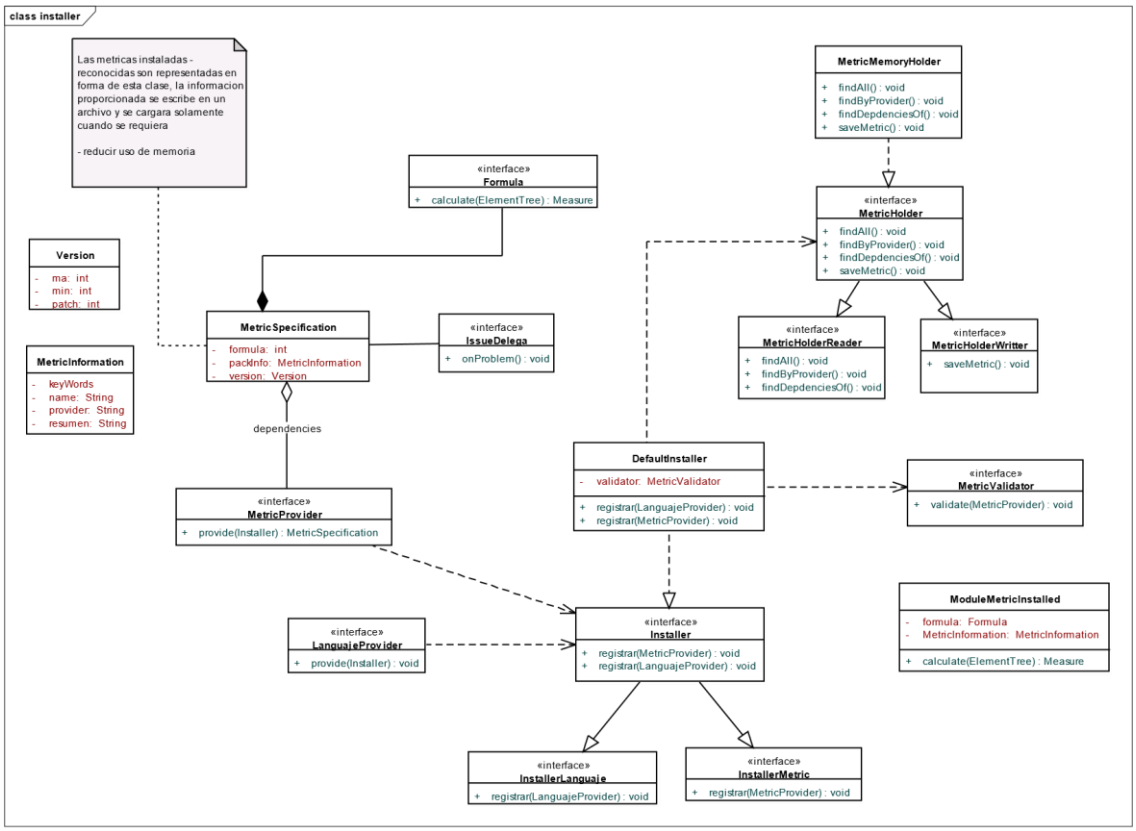
Figura 53 Diagrama de componentes.

Modulo Core métricas

Este módulo tiene la responsabilidad de extraer métricas basadas en el código, además expone una serie de interfaces de extensión del soporte de lenguajes y métricas; Agrupa la lógica relacionada con el procesamiento, recolección de datos y almacenamiento en cache de las métricas.

Para lenguajes el modulo proporciona una estructura de datos de tipo árbol para el mapeo y representación de un proyecto, proporciona utilidades para realizar búsquedas sobre el árbol de código. Otro punto que se debe tomar en cuenta es que los Parser o generadores de árboles AST, se encuentran registrados a este nivel.

En cuanto el soporte para métricas, se provee utilidades para la construcción, enlazado de métricas con sus dependencias, y ejecución concurrente. El proporcionar facilidad de extensión para cualquier desarrollador interesado en implementar sus propias métricas, las ventajas de este enfoque se resumen en: mejoras constantes mediante la reutilizar y mejorar las mismas. Este es un factor clave dentro de este proyecto, la gran variedad de métricas de código hace que sea difícil analizarlas todas debido a las limitantes de tiempo.



El núcleo proporciona medios para extender el suporte de métricas, este soporte debe garantizar que se cumplan los siguientes requisitos para el reconocimiento de un nuevo módulo de tipo métrica:

Una métrica no puede tener dependencias cíclicas

- Una métrica debe estar ligada a una especificación (Fórmula de cálculo, descripción y opcionalmente manejo de errores)
- Una métrica debe estar asociada a un único tipo de nodo específico para el análisis.

Módulo Sistema de archivos.

Este módulo es el encargado de proporcionar un API de simplificación del sistema archivos del sistema operativo, proporciona facilidades tanto para búsqueda de archivos y escucha de eventos.

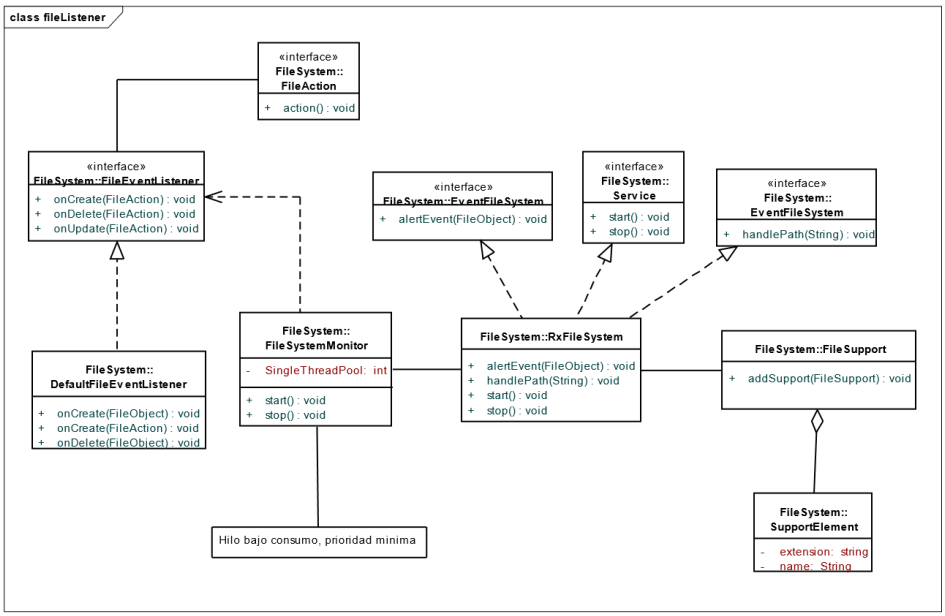
Las funcionalidades de escucha de eventos tienen soporte para tres tipos de acciones:

- Modificación de un archivo de código.
- Creación de un archivo de código.
- Eliminación de código.

Dentro de este módulo también se proporciona utilidades para detectar la inactividad de programación del estudiante, es decir médiate su API se puede crear monitores que indican cuanto tiempo el estudiante ha estado trabajando en un archivo determinado.

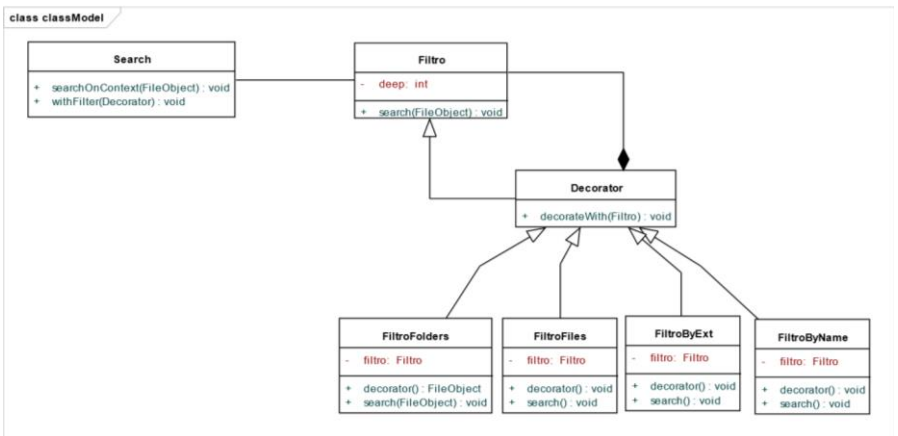
Sistema de archivos detector de eventos.

Este componente está basado sobre apache VFS, se han introducido una serie de mejoras para minimizar el uso de memoria (el monitor por defecto proporcionado por VFS escucha eventos de todos los archivos encontrados en un path), la modificación está orientada al trabajo con archivos de tipo “Código Fuente”



Sistema de archivos buscador.

El buscador está basado en el patrón de diseño decorador, esto permite al cliente añadir varios filtros de forma dinámica, para realizar búsquedas sobre el sistema de archivos requiere conocer el contexto (“Root Folder”) y la profundidad a alcanzar por el buscador.



Sistema de archivos detector de inactividad.

Este módulo tiene como función la detección de inactividad de código, así como el tiempo invertido por el programador en cada uno los archivos del programa. Este módulo proporciona un API de acceso para la especificación del tiempo mínimo para el cálculo del Detal Time por archivo.

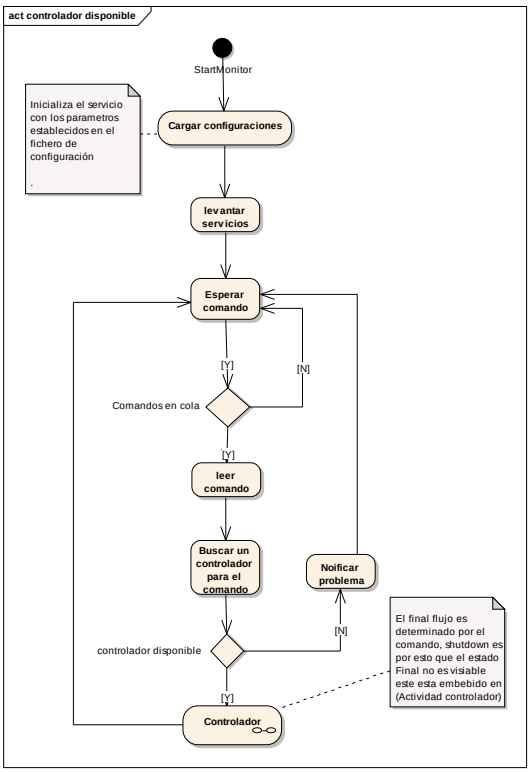
Modulo Almacenamiento.

Modulo encargado del almacenamiento local de datos, este módulo es una implantación del patrón repositorio, de manera que la aplicación no está altamente ligada al tipo de base de datos.

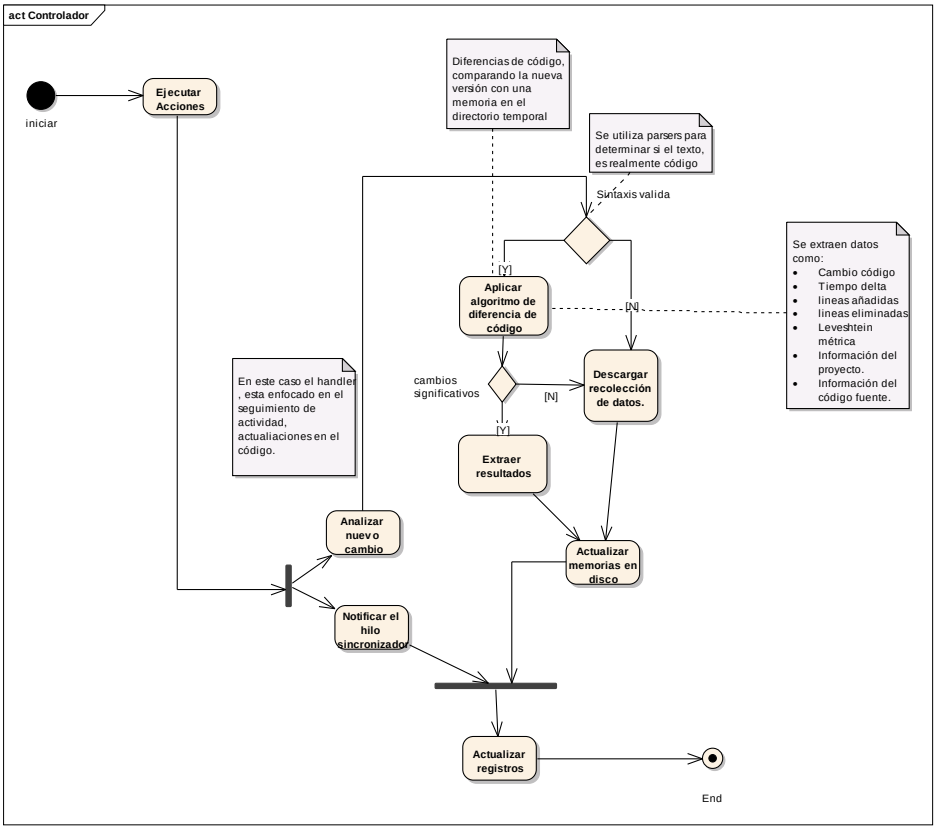
Vista de proceso.

Trata aspectos dinámicos de un sistema, esta vista es enfocada en el comportamiento del sistema en tiempo de ejecución. Para la recolección de métricas, el proceso está plasmado en el siguiente diagrama de actividad:

Este diagrama muestra el flujo general del monitor, este diagrama muestras las principales actividades cual el monitor inicia.



Este diagrama contempla el controlador de las acciones, este diagrama las actividades que se ejecutar para analizar el código del estudiante.



ANEXO 4. LIBRERÍAS DESARROLLADAS

Arboles de comportamiento [BT]

Un árbol de comportamiento es una estructura que ha sido desarrollada con el objetivo de modelar y definir comportamientos , la idea de tras de esta estructura es la de afrontar los problemas que presentan las tradicionales máquinas de estados (FSM),en cuanto al crecimiento y escalabilidad en número de transiciones, la forma en cómo logra esto es eliminando las transacciones entre estos, definiendo mecanismos intermedios para decidir la acción a ejecutar, retornando a los invocadores mensajes de estado.

Los arboles de comportamiento a diferencia de las máquinas de estado permiten reusar el comportamiento de un nodo en distintas formas bajos distintas circunstancias, lograr esto con las máquinas de estados es muy complejo, los BT están basados en tareas, como nodos raíz, los tipos más básico son:

- Tareas simples
- Tareas compuestas

Los componentes implementados en las librerías de este proyecto son:

Selector	Es similar al operador if y se ejecuta una sola vez cuando uno de sus nodos hijos retorna un código de éxito.
Secuencia	Representa un conjunto de acciones que se han de ejecutar en un orden prestablecido
Hojas	Representa una operación final, la cual no puede tener descendientes.

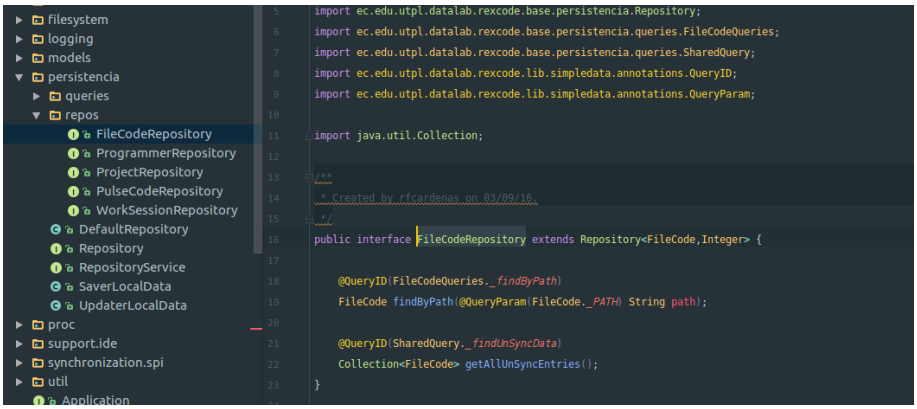
Arboles de comportamiento frente a las máquinas de estado

A diferencia de las máquinas de estado, los arboles de comportamiento permiten reutilizar funcionalidad en mayor medida, en las máquinas de estado se puede reutilizar los estados sin embargo esto es más complicado debido a que se debe tener cuidado con las transiciones entre estos y no romper el modelo o caer en bucles indefinidos.

Librería de persistencia.

Este proyecto nació de la necesidad de contar con librerías que automáticamente la implementen de modelos de acceso a datos en ORMLite, esta librería es un top sobre una ya existente, ayuda a minimizar la cantidad de código requerida para la persistencia de los datos.

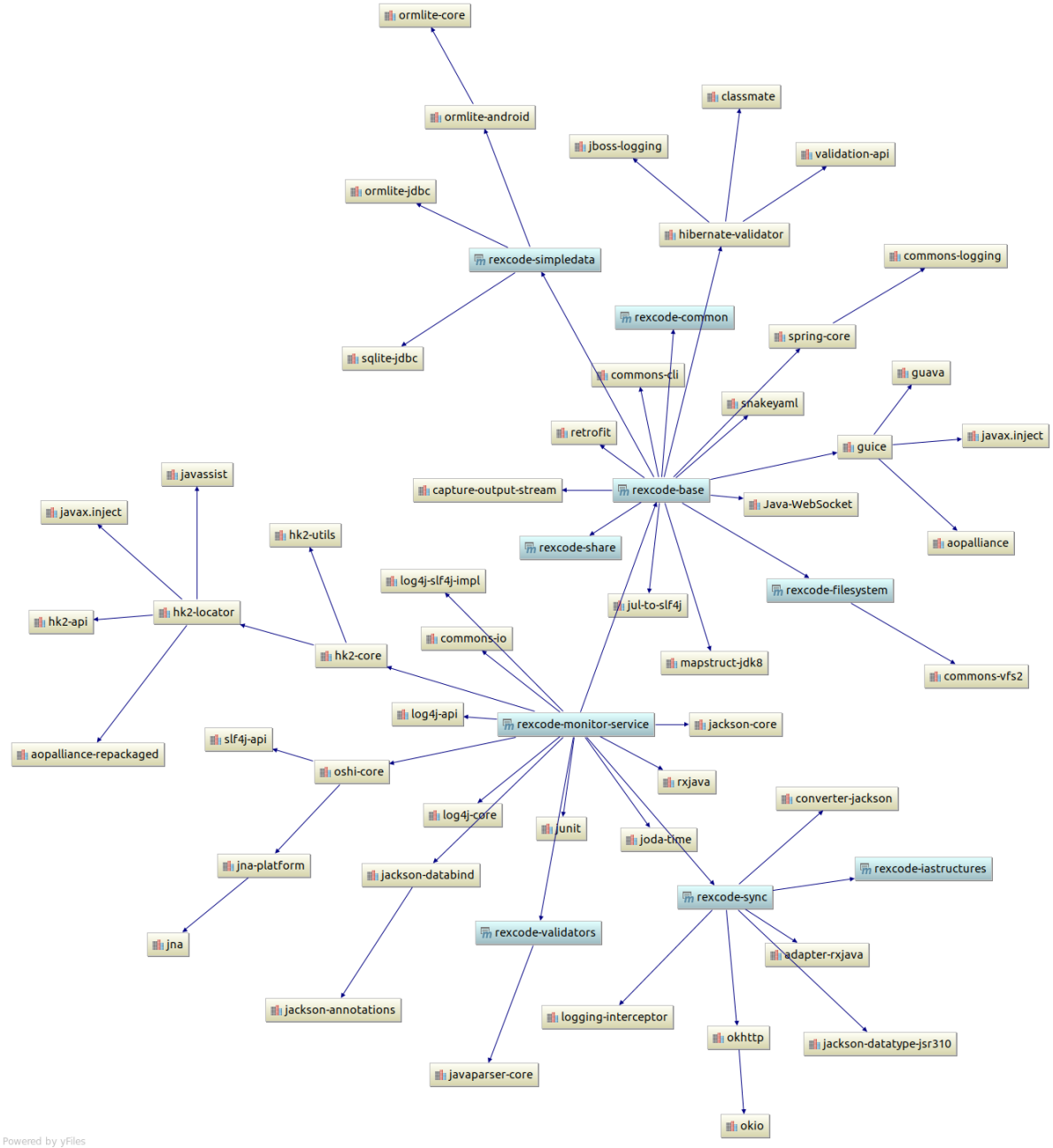
Actualmente la librería trabaja usando proxies dinámicos para la generación de instancias, así como la funcionalidad de los métodos definidos en una interfaz



¿Por qué se lo desarrollo?

Un Hibernate es algo muy pesado para algo que va ser ligero, la idea es generar algo que facilite la capa de persistencia, pero consumiendo menos memoria, esta librería es válida para las necesidades del proyecto.

ANEXO 5. GRAFO DE MÓDULOS Y SUS DEPENDENCIAS



ANEXO 6. ANEXO DE DESARROLLO.

Para quien esté interesado en seguir añadiendo funcionalidad al monitor, en este anexo se describe las principales componentes que se pueden extender. La idea es mostrar y poner a disposición del desarrollador una guía rápida para familiarizarse con los componentes desarrollados así como el uso de los mismos.

Artefacto	Versión	funcionalidad
Libs	V 1.0 pom.xml	Proporciona librerías desarrolladas y personalizadas útiles en el proyecto, las más relevantes son: • rexcode-filesystem • rexcode-iastructures • rexcode-simpledata • rexcode-share
rexcode-base	V 1.0 pom.xml	Este artefacto proporciona la base de todo el proyecto, proporciona los principales contratos y utilidades. En este nivel se implementan los diferentes environments del monitor
rexcode-metrics-base	V 1.0 pom.xml	Artefacto que proporciona el núcleo de la funcionalidad para la recolección de métricas mediante análisis estático del código.
rexcode-metrics-java	V 1.0 pom.xml	Artefacto que proporciona la implementación y soporte de métricas seleccionada para programas JAVA.
rexcode-shell	V 1.0 pom.xml	Artefacto que engloba la shell del monitor, el núcleo y los comandos disponibles en formato autocomplete.
rexcode-sync	V 1.0 pom.xml	Artefacto centrado en la lógica de sincronización de datos, es una implementación de los contratos ofrecidos por rexcodebase.
rexcode-monitor	V 1.0 pom.xml	Artefacto monitor de trabajo, el principal artefacto que enlaza la funcionalidad de los demás componentes.
rexcode-validators	V 1.0 pom.xml	Artefacto que separa los validadores

Convenciones de redacción

Tipo	Color
Interfaz	
Clase Abstracta	
Clase	

Terminal del monitor.

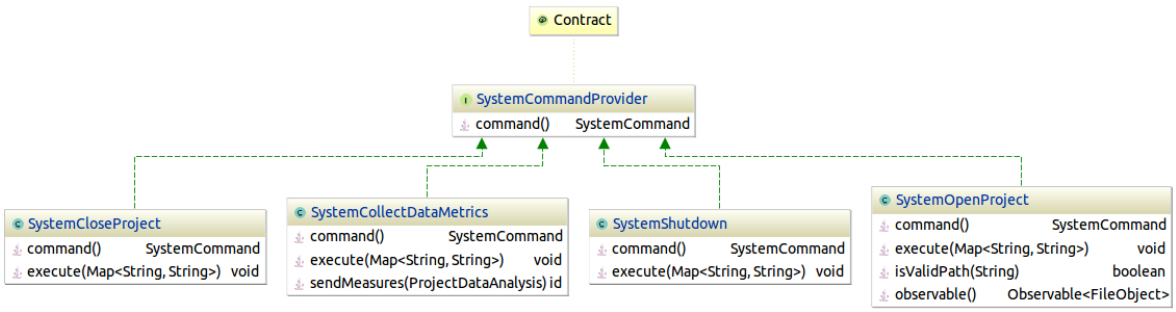
El monitor, procesa órdenes tomando como entrada las acciones implementadas en la terminal, cada comando implementa la interfaz SystemComandProvider, estos comandos son registrados en el Servicio de inyección de dependencias (InjectionService.java).

Los comandos en formato texto arriban al ProcessChannel, existen varias implementaciones para la comunicación entre procesos,

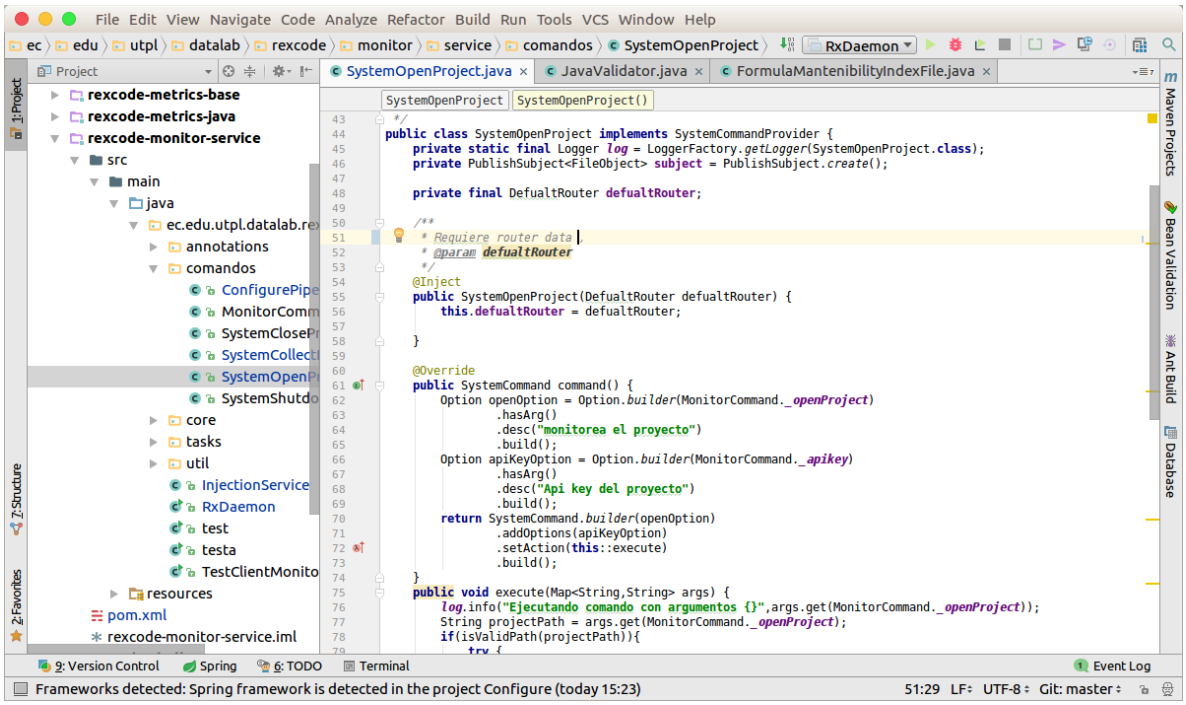
La comunicación puede ser llevada a cabo mediante:

- Websockets: es la opción preferida y con mayor soporte debido a que para futuros trabajos soporta la integración con IDEs basados en javascript.
- Memory File Share: Esta opción es la más limitada y su uso se basa en archivos compartidos.
- RMI: Java proporciona un mecanismo para invocación de métodos de forma remota

Aunque se puede implementar nuevas formas de comunicación de procesos extendiendo la clase abstracta AbstractProcessChannel



Un ejemplo de como un implementar un comando para el monitor se ilustra en la siguiente imagen:



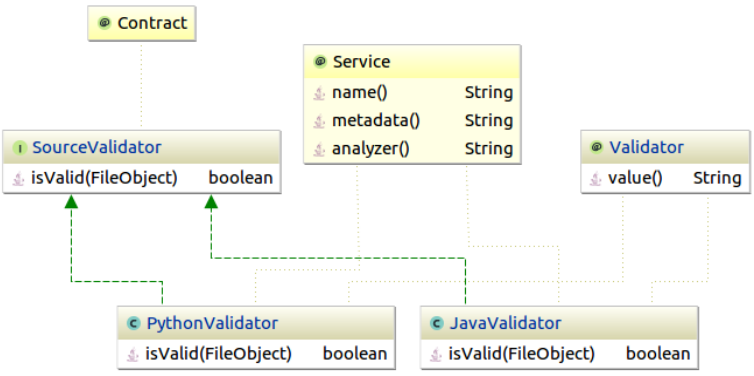
Validador lenguaje

El validador de lenguaje, limita los pulsos de código del programador se envían al servidor cuando el código este bien escrito (la implementación por defecto valida que el código sea parseable, sin embargo se puede implementar validadores más complejos como determinar si el cambio solamente fue a nivel comentario para evitar envíos innecesarios.)

```java
@Service
@Validator("java")
public class JavaValidator implements SourceValidator{
    @Override
    public boolean isValid(FileObject fileObject) throws Exception {
        boolean valid = true;
        InputStream content  = fileObject.getContent().getInputStream();
        try {
            CompilationUnit javaParser = JavaParser.parse(content);
        }catch (Exception e){
            valid = false;
        }finally {
            content.close();
        }
        return valid;
    }
}
```

El validador debe ser anotado con la anotación @Validator el cual tiene la extensión del lenguaje que tiene que procesar.

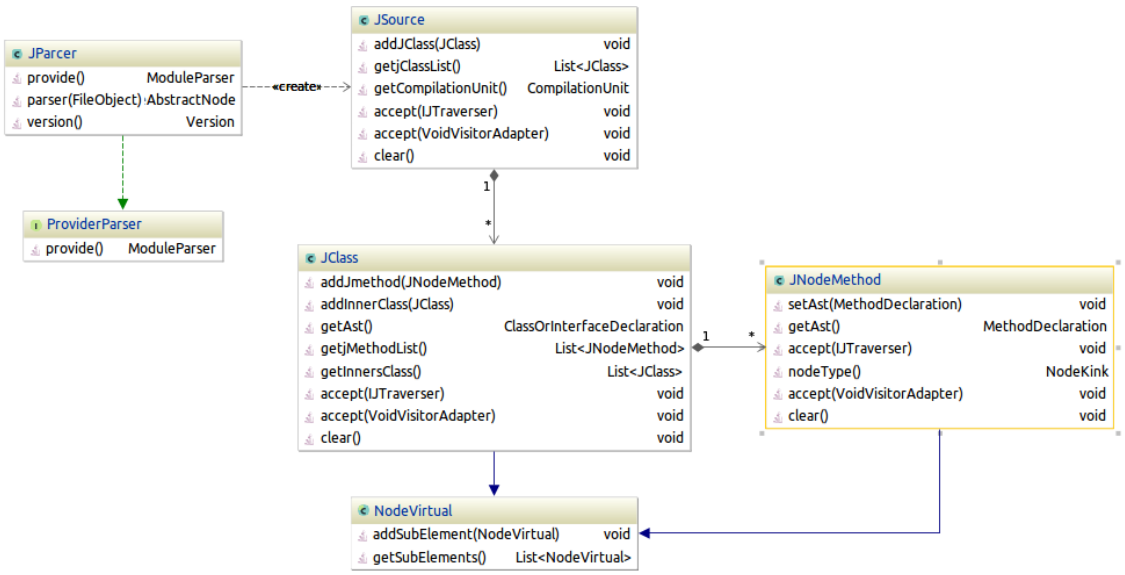
El api para acceder a memorias existentes de código es proporcionado por la interface MemoryBlock la correspondiente implementación es DefaultMemoryBlock

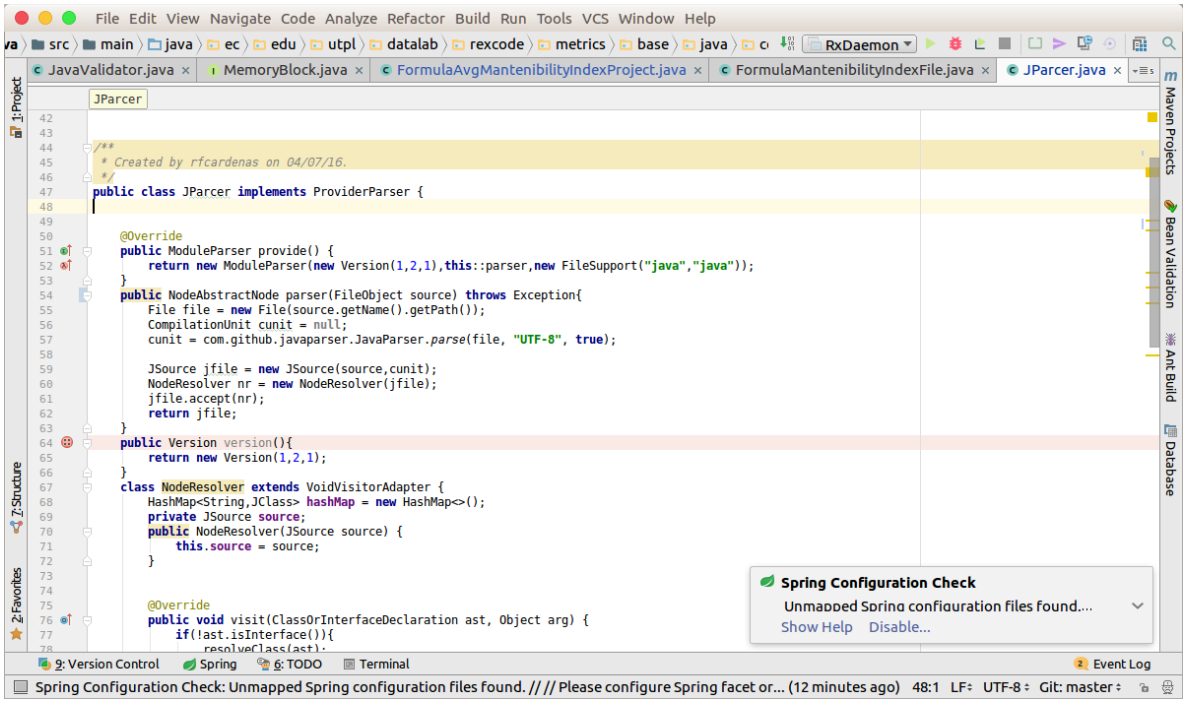


Soporte nuevos lenguajes

Para dar soporte a nuevos lenguajes de programación se tiene que implementar la interfaz ProviderParser esta interface retorna un nodo abstracto que representa el AST del código del estudiante, se puede utilizar librerías de generación AST existentes para distintos lenguajes de programación, en esta versión se ha empleado JavaParser para la generación del árbol java.

El patrón visitor es el patrón ideal para realizar operaciones distintas y no relacionadas en los diferentes niveles de un compuesto, sin embargo, proporcionamos un API para facilitar este trabajo utilizando el patrón iterator, específicamente spliterator de java 8





Para atravesar un conjunto de nodos se puede utilizar TreeTypeSpliterator, el cual recibe como parámetros el nodo root desde el cual comenzar el recorrido y la clase objetivo que se quiere alcanzar.

```
TreeTypeSpliterator<JSource> typeSpliterator = TreeTypeSpliterator.create(fragment,JSource.class);
totalLines = StreamSupport.stream(typeSpliterator,false)
        .filter(jSource -> jSource.getCompilationUnit()!=null)
        .mapToInt(jSource -> {
            try {
                CommentVisitor commentVisitor = new CommentVisitor();
                jSource.accept(commentVisitor);
```

Existen varias implementaciones de recorridos del árbol.

- TreeTypeSpliterator: Para atravesar nodos específicos según el tipo.
- DeepFirstSpliterator: Para atravesar nodos de un árbol por profundidad.
- BreadFirstSpliterator: Para atravesar nodos desde raíz.

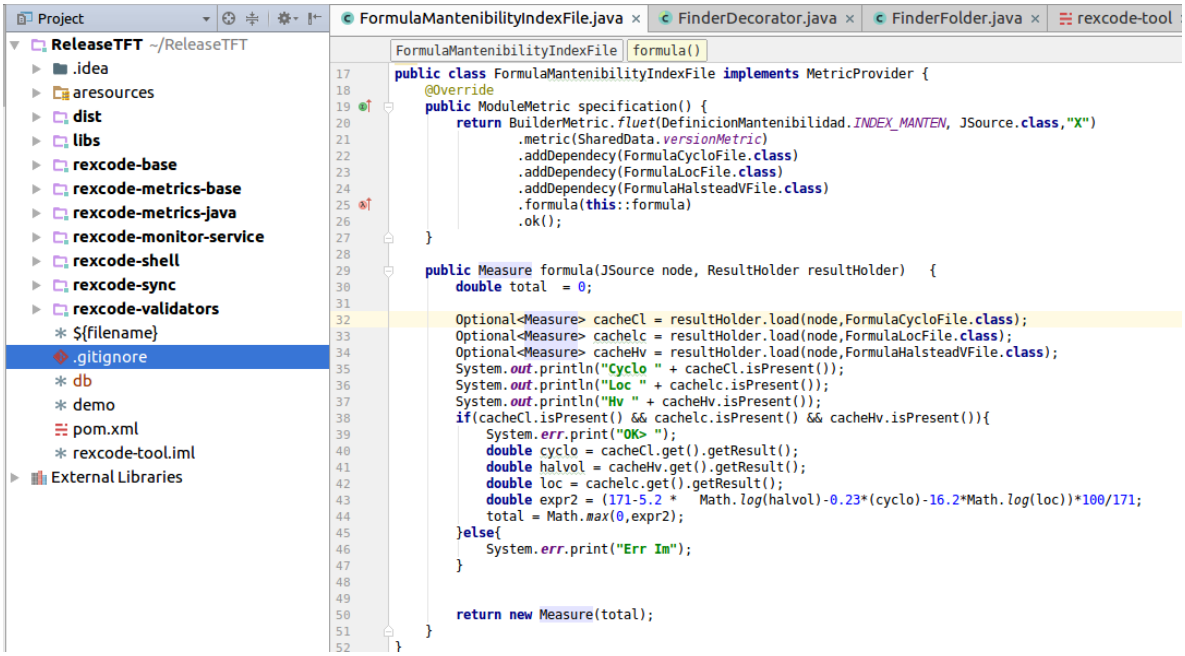
1. Extendiendo el conjunto de métricas.

Para extender el conjunto de métricas existentes, se debe hacer uso de la interfaz MetricProvider, esta interfaz representa el contrato base para proveer una métrica, en la figura se muestra un ejemplo de la métrica “índice de mantenibilidad”, la cual requiere de varias dependencias (líneas 22-24):

- Complejidad ciclomatica
- Líneas de código

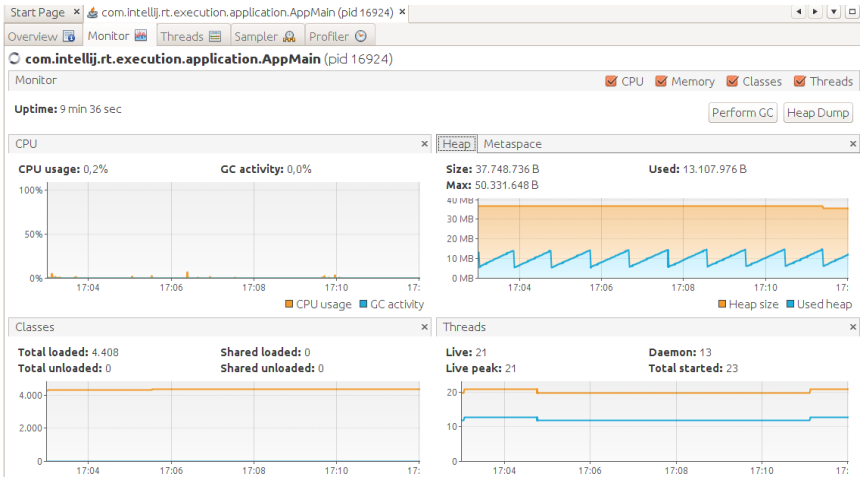
- Volumen Halstead

Al definir estas dependencias el núcleo genera y valida un grafo de la métrica, para determinar la secuencia de acciones a ejecutar. Además se valida de que estas dependencias sea validas, evitando problemas cíclicos.



Gestión de memoria

La siguiente figura representa un patrón dentado, el cual indica que el monitor devuelve todos recursos que utilizada, cada vez que GC es ejecutado JVM.



En memoria fisica RAM usada el monitor usa 98M, aunque la cantidad dependen de cuan grandes es el proyecto el maximo por defecto al trabajar en pruebas con spring boot fue de 160M

Diagramas de implementación.

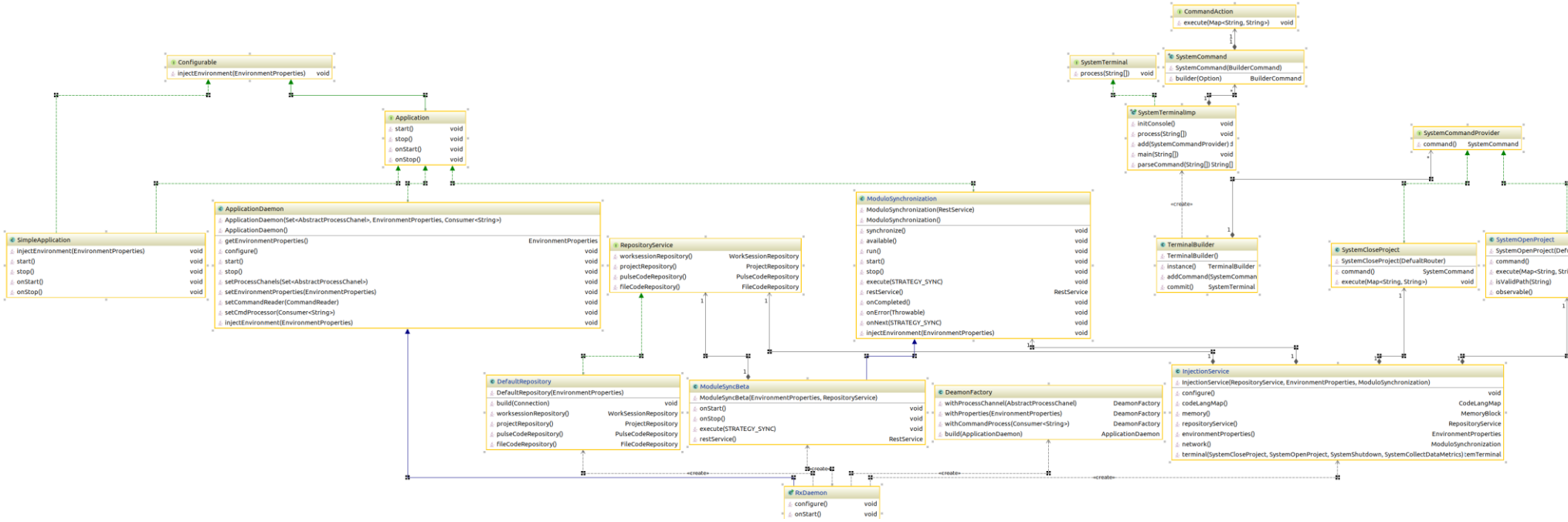
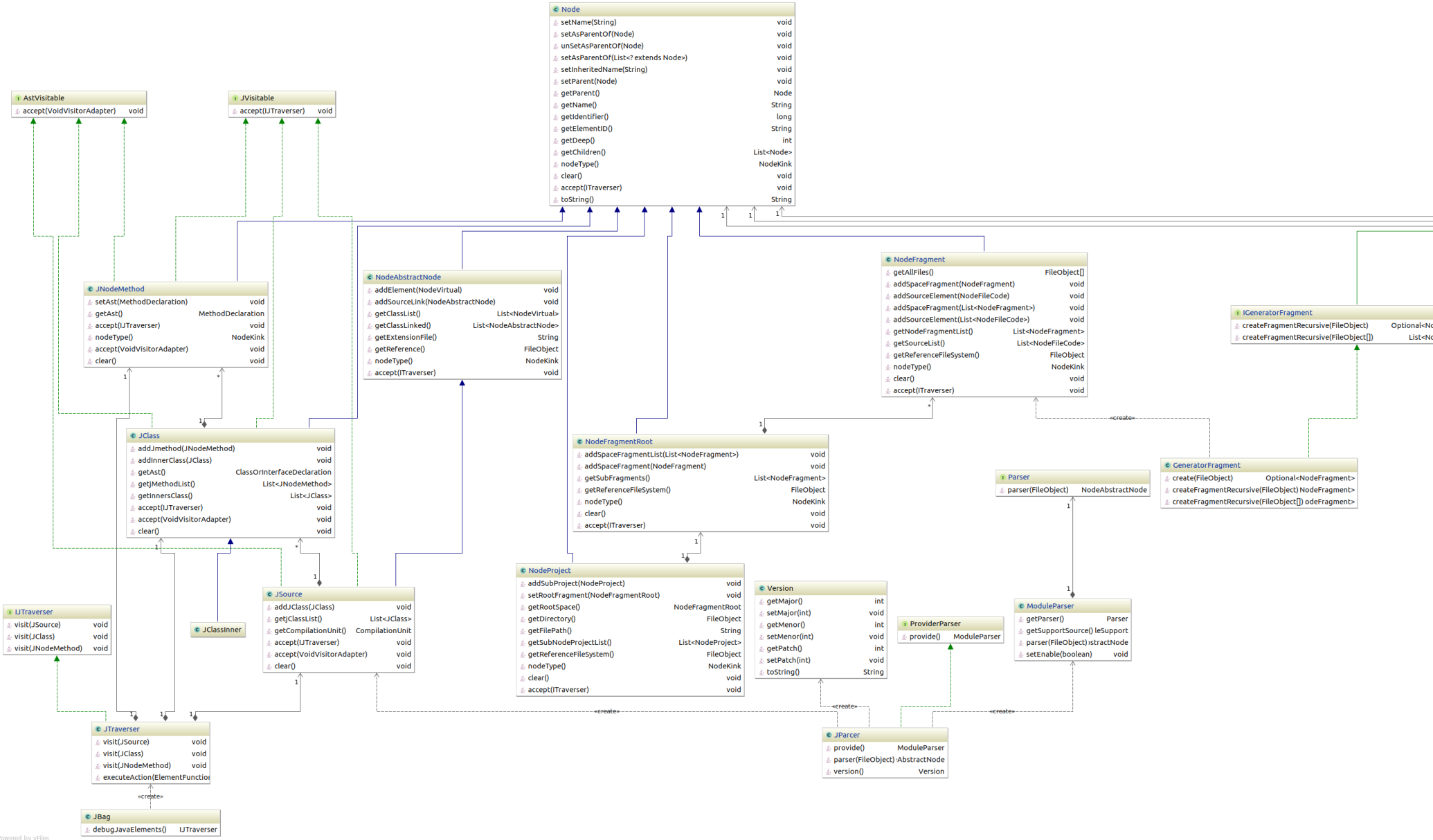
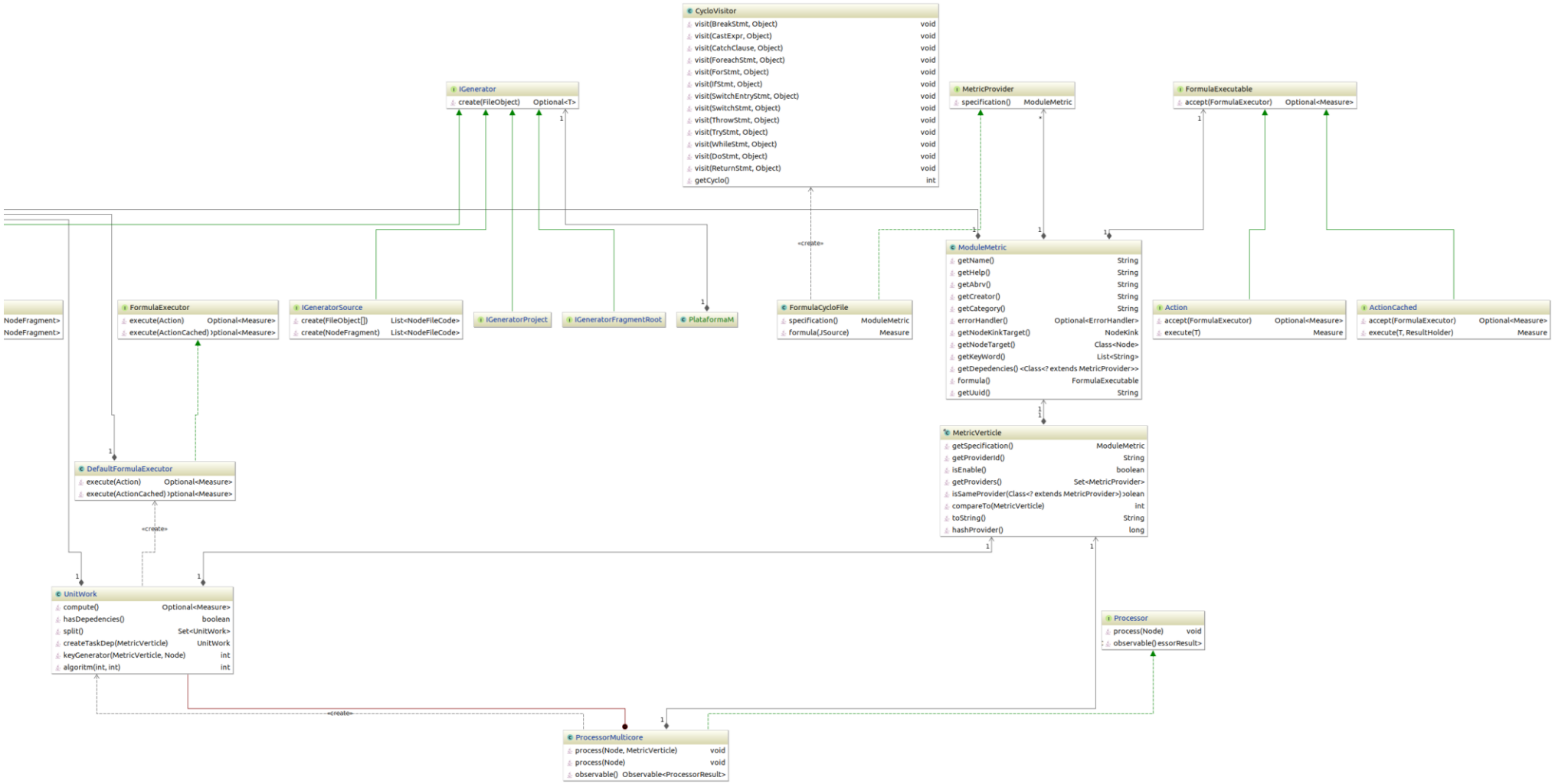


Figura 54 Diagrama de clases.

Fuente: Autor.


Node
setName(String) void
setAsParentOf(Node) void
unSetAsParentOf(Node) void
setAsParentOf(List<? extends Node>) void
setInheritedName(String) void
setParent(Node) void
getParent() Node
getName() String
getIdentifier() long
getElementID() String
getDeep() int
getChildren() List<Node>
nodeType() NodeKink
clear() void
accept(ITraverser) void
toString() String
AstVisitable
accept(VoidVisitorAdapter) void
JVisitable
accept(IJTraverser) void
JNodeMethod
setAst(MethodDeclaration) void
getAst() MethodDeclaration
accept(IJTraverser) void
nodeType() NodeKink
accept(VoidVisitorAdapter) void
clear() void
NodeAbstractNode
addElement(NodeVirtual) void
addSourceLink(NodeAbstractNode) void
getClassList() List<NodeVirtual>
getClassLinked() List<NodeAbstractNode>
getExtensionFile() String
getReference() FileObject
nodeType() NodeKink
accept(ITraverser) void
NodeFragment
getAllFiles() FileObject[]
addSpaceFragment(NodeFragment) void
addSourceElement(NodeFileCode) void
addSpaceFragment(List<NodeFragment>) void
addSourceElement(List<NodeFileCode>) void
getNodeFragmentList() List<NodeFragment>
getSourceList() List<NodeFileCode>
getReferenceFileSystem() FileObject
nodeType() NodeKink
clear() void
accept(ITraverser) void
IGeneratorFragment
createFragmentRecursive(FileObject) Optional<No
createFragmentRecursive(FileObject[]) List<No
JClass
addJmethod(JNodeMethod) void
addInnerClass(JClass) void
getAst() ClassOrInterfaceDeclaration
getjMethodList() List<JNodeMethod>
getInnersClass() List<JClass>
accept(IJTraverser) void
accept(VoidVisitorAdapter) void
clear() void
NodeFragmentRoot
addSpaceFragmentList(List<NodeFragment>) void
addSpaceFragment(NodeFragment) void
getSubFragments() List<NodeFragment>
getReferenceFileSystem() FileObject
nodeType() NodeKink
clear() void
accept(ITraverser) void
Parser
parser(FileObject) NodeAbstractNode
GeneratorFragment
create(FileObject) Optional<NodeFragment>
createFragmentRecursive(FileObject) NodeFragment>
createFragmentRecursive(FileObject[]) odeFragment>
«create»
IJTraverser
visit(JSource) void
visit(JClass) void
visit(JNodeMethod) void
JClassInner
JSource
addJClass(JClass) void
getjClassList() List<JClass>
getCompilationUnit() CompilationUnit
accept(IJTraverser) void
accept(VoidVisitorAdapter) void
clear() void
NodeProject
addSubProject(NodeProject) void
setRootFragment(NodeFragmentRoot) void
getRootSpace() NodeFragmentRoot
getDirectory() FileObject
getFilePath() String
getSubNodeProjectList() List<NodeProject>
getReferenceFileSystem() FileObject
nodeType() NodeKink
clear() void
accept(ITraverser) void
Version
getMajor() int
setMajor(int) void
getMenor() int
setMenor(int) void
getPatch() int
setPatch(int) void
toString() String
ProviderParser
provide() ModuleParser
ModuleParser
getParser() Parser
getSupportSource() leSupport
parser(FileObject) ›stractNode
setEnable(boolean) void
JTraverser
visit(JSource) void
visit(JClass) void
visit(JNodeMethod) void
executeAction(ElementFunctio
«create»
JParcer
provide() ModuleParser
parser(FileObject) ›AbstractNode
version() Version
JBag
debugJavaElements() IJTraverser
Powered by yFiles


CycloVisitor
visit(BreakStmt, Object) void
visit(CastExpr, Object) void
visit(CatchClause, Object) void
visit(ForeachStmt, Object) void
visit(ForStmt, Object) void
visit(IfStmt, Object) void
visit(SwitchEntryStmt, Object) void
visit(SwitchStmt, Object) void
visit(ThrowStmt, Object) void
visit(TryStmt, Object) void
visit(WhileStmt, Object) void
visit(DoStmt, Object) void
visit(ReturnStmt, Object) void
getCyclo() int
IGenerator
create(FileObject) Optional<T>
MetricProvider
specification() ModuleMetric
FormulaExecutable
accept(FormulaExecutor) Optional<Measure>
«create»
ModuleMetric
getName() String
getHelp() String
getAbrv() String
getCreator() String
getCategory() String
errorHandler() Optional<ErrorHandler>
getNodeKinkTarget() NodeKink
getNodeTarget() Class<Node>
getKeyWord() List<String>
getDepedencies() <Class<? extends MetricProvider>>
formula() FormulaExecutable
getUuid() String
Action
accept(FormulaExecutor) Optional<Measure>
execute(T) Measure
ActionCached
accept(FormulaExecutor) Optional<Measure>
execute(T, ResultHolder) Measure
NodeFragment>
NodeFragment>
FormulaExecutor
execute(Action) Optional<Measure>
execute(ActionCached))ptional<Measure>
IGeneratorSource
create(FileObject[]) List<NodeFileCode>
create(NodeFragment) List<NodeFileCode>
IGeneratorProject
IGeneratorFragmentRoot
PlataformaM
FormulaCycloFile
specification() ModuleMetric
formula(JSource) Measure
MetricVerticle
getSpecification() ModuleMetric
getProviderId() String
isEnable() boolean
getProviders() Set<MetricProvider>
isSameProvider(Class<? extends MetricProvider>))olean
compareTo(MetricVerticle) int
toString() String
hashProvider() long
DefaultFormulaExecutor
execute(Action) Optional<Measure>
execute(ActionCached))ptional<Measure>
«create»
UnitWork
compute() Optional<Measure>
hasDepedencies() boolean
split() Set<UnitWork>
createTaskDep(MetricVerticle) UnitWork
keyGenerator(MetricVerticle, Node) int
algoritm(int, int) int
Processor
process(Node) void
observable() essorResult>
«create»
ProcessorMulticore
process(Node, MetricVerticle) void
process(Node) void
observable() Observable<ProcessorResult>

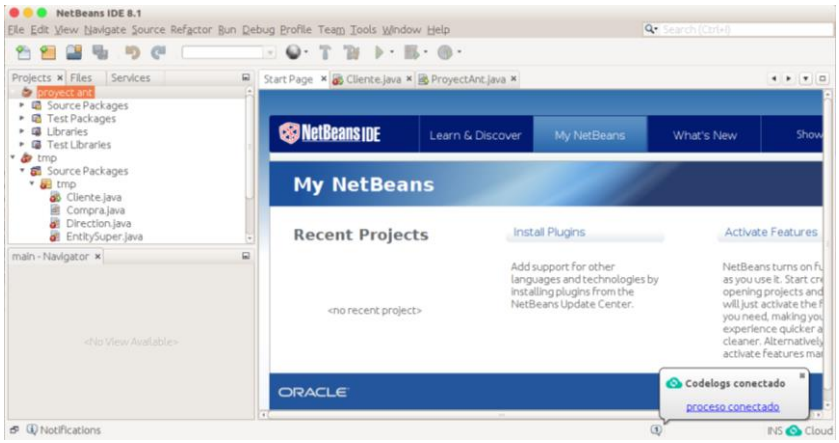
ANEXO 7. ANEXO DE USUARIO.

Plugin.

Como ya se mencionó, este plugin permite formar un vínculo entre el IDE y monitor desarrollado, se debe tener presente los siguientes iconos ya que estos indican el estado del enlace entre estos componentes.

	Icono de conexión.	Conectado al proceso monitor, el cual sincroniza la data con el servidor.
	Icono perdido de conexión.	India que el proceso monitor, no está corriendo y que el plugin no pudo conectarse para enviarle comandos.
	Icono sincronización.	Icono que indica que el monitor está trabajando

Cuando se inicia el IDE, se presenta una notificación indicando el estado del enlace entre el IDE y el proceso monitor.

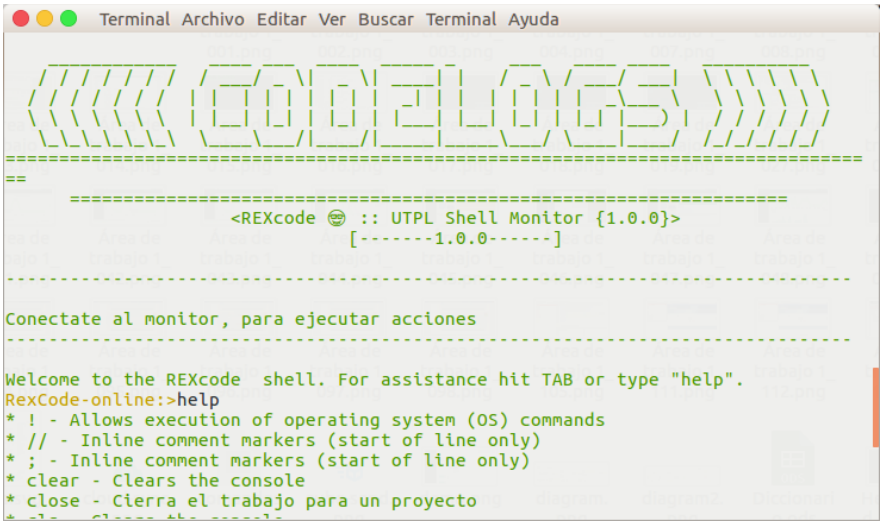


Si el IDE no encuentra ningún proceso monitor corriendo, lo que hace es levantar un proceso nuevo, pasándole como argumentos las siguientes configuraciones:

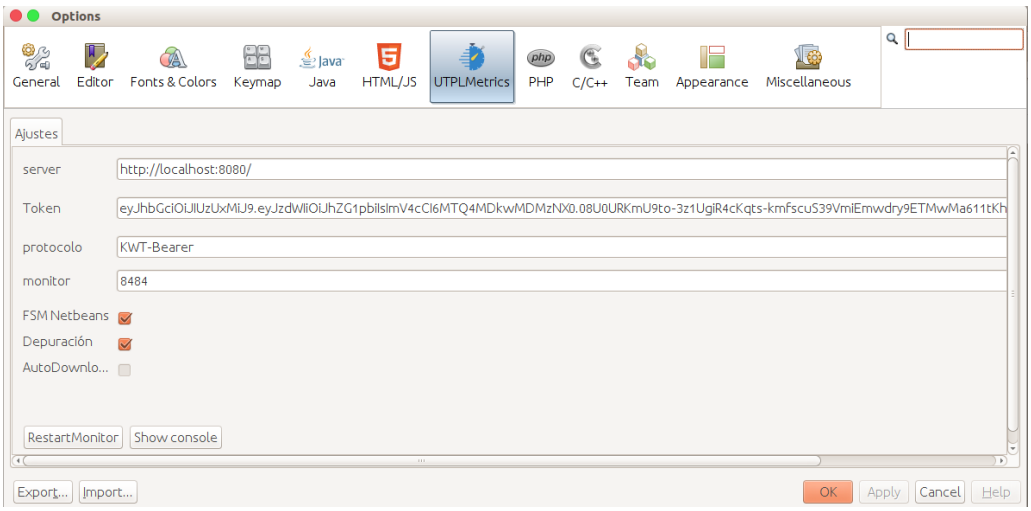
monitor.max.inactive = 1000
monitor.cpu.step = 1000
monitor.scan.path = src
monitor.cmd.sck = 9090
monitor.cmd.file = /tmp/rxcode/rxproc.proc
monitor.scan.support.file = /home/rfcardenas/ReleaseTFT/config/monitor_support.json

sync.webSocketProcessChannel.host	=	"http://localhost:8080"
sync.webSocketProcessChannel.sync.protocol	=	"KWT-Bearer"
sync.webSocketProcessChannel.sync.apikey	=	"eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJhZG1pbiIsImV4cCI6MTQ3NTg2NTMxNH0.7WN0KHghur7zDdFZk2W4afxihQB_ypkMQBRWD2AWlTyw7A7Fd2CSSyyr8w8yl2edwWmgZWYLl8lofuxlfHCOmQ"

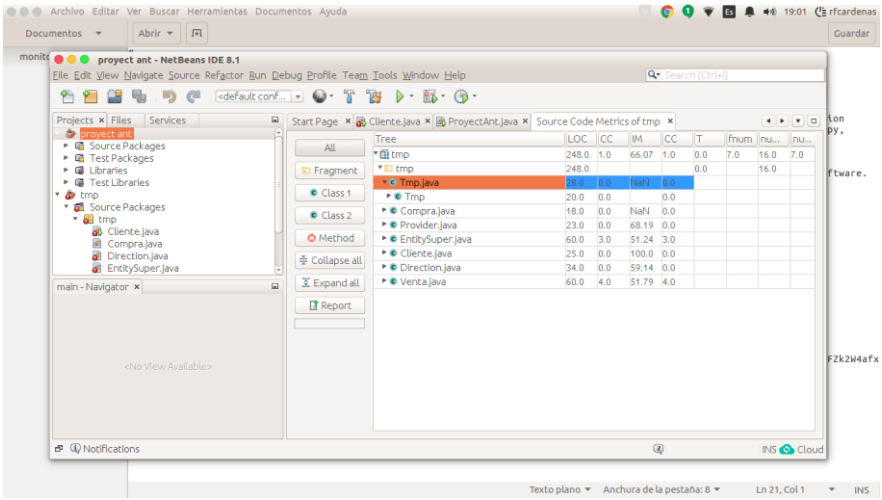
Si para el IDE que el estudiante utiliza no existe un plugin, se brinda una Shell para ejecutar órdenes sobre los proyectos, como por ejemplo indicar que proyecto monitorear, sincronizar datos etc. Todas opciones se puede visualizar invocando el comando help.



Dentro de Netbeans, se puede personalizar los ajustes del monitor navegando a las opciones del plugin : Tools> Options > UTPL Metrics



El análisis de código es ejecutando mediante opciones contextuales, puesto que este tipo de cálculos es muy costo se ejecuta solamente bajo demanda del usuario.



El modo de distribución del monitor es en formato comprimido (tar.gz y zip), estas distribuciones son independientes del IDE de programación, permitiendo que en futuros trabajos se implementen plugins para nuevos IDE de programación.

Aplicación web.

Para que el monitor pueda sincronizar los datos, necesita un token valido, para generar este se debe ir a la aplicación web y copiarlo en los ajustes del plugin.

El registros puede ser llevado utilizando las redes sociales, asi que el acceso a una cuenta es relativamente sencillo.

ANEXO 8. INSTALACIÓN.

Despliegue servidor modo desarrollo

Primero se debe empaquetar la aplicación utilizando el perfil dev, seguido de esto se puede ejecutar la aplicación normalmente, este perfil la configuración está diseñada para trabajar con base de datos embebida h2.

```
→  herokumod git:(master) ✗ ./mvnw package -Pdev -DskipTests
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building Herokumod 0.0.1-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-resources-plugin:3.0.1:copy-resources (default-resources) @ her
okumod ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 13 resources
[INFO] Copying 13 resources
[INFO]
```

Despliegue servidor modo producción

Para ejecutar la aplicación en modo producción se tiene que empaquetar con el perfil prod, este perfil requiere de configuraciones en el archivo de application-prod.yml

Seguido de esto se pude utilizar herokutobelt para cargar y ejecutar la aplicación en el servidor.

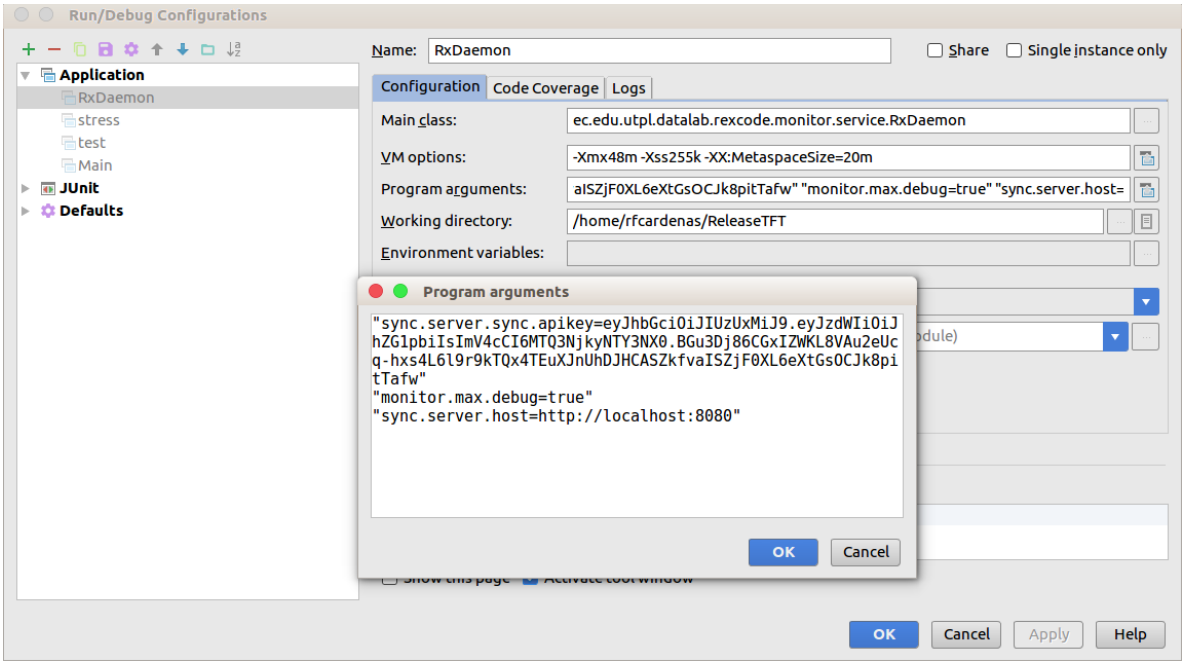
```
Email: rfcardenas@utpl.edu.ec
Password (typing will be hidden):
Logged in as rfcardenas@utpl.edu.ec
→  config git:(master) ✗ heroku ps
Free dyno hours quota remaining this month: 510h 22m (92%)
For more information on dyno sleeping and how to upgrade, see:
https://devcenter.heroku.com/articles/dyno-sleeping

=== web (Free): java  -jar target/*.war --spring.profiles.active=prod,heroku --s
erver.port=$PORT (1)
web.1: idle 2016/10/24 12:47:43 -0500 (~ 12h ago)
```

Instalación en Heroku

Heroku es una plataforma como servicio de computación en la nube que soporta distintos lenguajes de programación. Nació el 2007 y es una pionera en el campo de plataformas de computación en la nube. La aplicación a nivel del servidor está totalmente diseñada y configurada para correr fácil sin complicaciones, su consumo de memoria está dentro los límites establecidos por heroku 512 RAM para cuentas gratuitas.

Despliegue monitor



Para ejecutar el monitor en modo desarrollo se necesita pasar como argumento la propiedad del ambiente: monitor.max.debug = boolean value o se puede utilizar el archivo de configuración.

ANEXO 9. REPOSITORIOS.

El proyecto está versionado en gitlab, bajo el dominio Codelogs, este grupo contiene los diferentes proyectos que forman parte de este TT todos estos están bajo la licencia MIT.

Incluir el código fuente de la aplicación es poco práctico debido a la extensión del mismo, una mejor manera de exponer el trabajo es a través de los repositorios oficiales de este proyecto

Repositorio	Utilidad	URL
codelogs-backend	Repositorio del servidor, aplicación Spring boot construcciones y despliegue automático.	https://gitlab.com/codelogs/codelogs-backend
codelogs-frontend-appstats	Appstats, aplicación angular para visualización estadística de los datos recolectados, utilizando la API REST del server.	https://gitlab.com/codelogs/codelogs-frontend-appstats
codelogs-frontend-public	Repositorio aplicación publica, acceso a usuario sin autorización.	https://gitlab.com/codelogs/codelogs-frontend-public
codelogs-spyder-complements	Repositorio complementos maven, permiten recolectar datos de compilaciones y ejecuciones de un programa.	https://gitlab.com/codelogs/codelogs-spyder-complements
codelogs-spyder-plataforma	Código fuente de spyder, programa encargado de aplicar métricas de código, sincronización y datos de comportamiento.	https://gitlab.com/codelogs/codelogs-spyder-plataforma
codelogs-spyder-netbeans	Código fuente del plugin de Netbeans, enlace entre el servicio Spyder y el IDE	https://gitlab.com/codelogs/codelogs-spyder-netbeans
codelogs-open-repositories	Repositorio en el cual es utilizando como repositorio maven, en este repositorio se almacenan las liberáis desarrolladas, las librerías de este repositorio pueden ser utilizadas como cualquier dependencia en el pom.xml.	https://gitlab.com/codelogs/codelogs-open-repositories
Repositorio docker	Docker imagen basada en alpine , útil para integración continua en Gitlab -CI, aplicaciones del lado del cliente .	https://gitlab.com/codelogs/codelogs-min-dev
codelogs-btool-ic	Docker imagen, ambiente de desarrollo para el servidor, CI & deploy -> heroku.	https://gitlab.com/codelogs/codelogs-btool-ic
blogdev	Blog del proyecto, las entradas publicadas están relacionadas con: desarrollo, resultados y pruebas del proyecto.	https://gitlab.com/codelogs/blogdev