Gestión de cambios en proyectos software con evaluación de calidad de código fuente

Abstract. En un mundo globalizado y moderno, donde el software es cada vez más complejo y de gran tamaño, llevar adelante un proceso de desarrollo que finalice dentro de los parámetros acordados, a la vez que se entrega un producto de calidad no es una tarea trivial, así como tampoco lo es mantener esa calidad posteriormente. Para contribuir con la mejora de la calidad del proceso, vinculada particularmente con la mantenibilidad, en este trabajo se propone una herramienta para la gestión de proyectos de software que integra la asignación y seguimiento de pedidos de cambios con la evaluación de métricas aplicadas al código fuente. La información que la misma ofrece permitirá a los responsables y participantes del proyecto tener una visión amplia del estado del proyecto, y a la vez, un panorama de la calidad del mantenimiento correctivo y adaptativo en función de la medición del código fuente y los valores umbrales definidos. El análisis del código estático se basa en métricas conocidas que miden las características del código orientado a objetos, implementado en los lenguajes de programación Java y PHP, dos de los lenguajes de programación más utilizados en la industria del software.

Keywords: Gestión de Proyectos, mantenimiento de software, métricas orientadas a objetos, gestión de cambios, asignación de recursos.

1 Introducción

Coordinar las tareas dentro de cualquier equipo independientemente de la industria es una tarea fundamental para que el producto final se mantenga dentro de los parámetros establecidos, como el tiempo, costo o calidad. Los proyectos de desarrollo de software no son la excepción, pero al poseer la característica de entregar un producto abstracto, aplicable a problemas y ámbitos tan diversos, aspectos claves como la planificación, asignación de recursos y seguimiento de la calidad resulta muy difícil y variable para los distintos proyectos.

Por otra parte, es frecuente en las tareas de mantenimiento de software la asignación y resolución de pedidos de cambio, por ejemplo, para resolver un problema o introducir una mejora en el software. La búsqueda de la persona idónea, estando el problema sin asignar o asignado erróneamente no resulta trivial, repercutiendo en tiempo y esfuerzos perdidos. La tarea se hace cada vez más difícil a medida que el proyecto se incrementa en tamaño, recursos humanos o tiempo invertido.

La mantenibilidad del software es un atributo importante de calidad, pero es difícil de estimar. Por tanto, las mediciones de código pueden aportar información para mejorar la predicción del esfuerzo de mantenibilidad. También resulta difícil determinar qué y cuándo el software es de “calidad”, y si esos cambios introducidos repercuten en mejora o detrimento del mismo. El producto software debería tener, así como los de otras industrias, métricas que permitan predecir problemas y afirmar como se desenvolverá en el transcurso

de su desarrollo y posteriormente a su entrega, como por ejemplo, si será difícil de mantener.

En este contexto, la herramienta que se propone mejorará la gestión de proyectos de software, integrando la asignación y seguimiento de pedidos de cambios con el análisis de métricas aplicadas al código fuente. La información que brinda permitirá a los responsables y participantes del proyecto (gerentes, líderes y desarrolladores) tener una visión amplia del estado del proyecto, y a la vez, un panorama de la calidad del mantenimiento correctivo y adaptativo en función de la medición del código fuente y los valores umbrales definidos.

1.1 Objetivo

Contribuir a la mantenibilidad del software mediante una herramienta de gestión de proyectos de desarrollo, que integra la asignación de recursos y el seguimiento de pedidos de cambio con la evaluación de métricas al código fuente para monitorear la calidad del producto.

2 Calidad del software

La calidad del software es una compleja combinación de factores, que varían entre diferentes aplicaciones. Diversos autores como Pressman [1], McCall [2], y estándares tales como ISO 9126 [3] han tratado de determinar y categorizar los factores que afectan a la calidad del software. Para garantizar la calidad del software, se requiere medir los atributos que la definen.

Por su esencia, compleja y subjetiva, la calidad del software se basa en modelos y métricas que se aplican a la medición y evaluación de los distintos aspectos que afectan el proceso de desarrollo y el producto software. Por lo tanto, es necesario analizar las mediciones con que se evalúa la calidad del producto mientras se diseña o construye. Estas medidas de atributos internos del producto proporcionan a los desarrolladores una aproximación en tiempo real de la eficacia de los modelos de análisis, diseño y codificación, y de la calidad general del software que se construye.

2.1 Gestión cuantitativa de proyectos de software, mantenibilidad y medición

La gestión cuantitativa de proyectos [4] proporciona una visión del grado de cumplimiento de metas, así como de las causas que explican desviaciones significativas en procesos o productos. El propósito de la gestión es dirigir un proyecto basado en un conocimiento cuantitativo, es decir medible, determinable, de los aspectos de mayor relevancia que puedan afectar el logro de los objetivos. La medición permite modificar aquellos factores que aportan una mayor eficacia en el proceso productivo, haciendo a las organizaciones más eficientes y permitiendo una ventaja estructural frente a sus competidores [5].

2.2 Métricas Orientadas a Objetos (OO)

La utilización de métricas para sistemas OO ha progresado con mucha más lentitud que los demás métodos OO [6]. Sin embargo, a medida que esta metodología es cada vez más utilizada, resulta fundamental que los desarrolladores de software dispongan de mecanismos cuantitativos para estimar la calidad de los diseños y de los programas OO.

Las métricas para sistemas OO deben ajustarse a las características que los distinguen del software convencional. Hacen hincapié en los conceptos básicos del paradigma, tales como encapsulamiento, herencia, complejidad de clases y polimorfismo. Como en todas las métricas, el objetivo principal es comprender mejor la calidad del producto, estimar la efectividad del proceso y mejorar la calidad del trabajo realizado a nivel del proyecto [7].

Las métricas conocidas como métricas CK fueron propuestas por Chidamber y Kemerer y son las más difundidas en OO [8]. Este conjunto de métricas, definidas a nivel de clases, se muestra en la tabla 1.

Tabla 1. Suite Métricas Chidamber y Kemerer

Peso de Los Métodos por Clase (WMC)	Es el número de métodos en cada clase, ponderado por la complejidad de cada método. Un método simple puede tener una complejidad de 1, y un método grande y complejo tendrá un valor mucho mayor. Cuanto más grande sea el valor para esta métrica, más compleja será la clase. Es más probable que objetos complejos sean difíciles de entender.
Profundidad del árbol de Herencia (DIT)	Se trata de la cuenta directa de los niveles de la jerarquía de herencia, considerando que en el nivel cero de la jerarquía se encuentra la clase raíz. DIT se considera como una métrica del número de clases antecesoras que una clase podría potencialmente afectar.
Número de Hijos (NOC)	Es el número de clases subordinadas a una clase en la jerarquía, es decir, la cantidad de subclases que pertenecen a una clase. NOC (número de hijos) es un indicador del nivel de reutilización, de la posibilidad de haber creado abstracciones erróneas, y del nivel de pruebas requerido.
Acoplamiento entre Objetos (CBO)	Se dice que una clase esta acoplada con otra si los métodos de una clase utilizan métodos o atributos de otra y viceversa.
Respuesta Por Clase (RFC)	RFC (respuesta de una clase) es una medida del número de métodos que potencialmente podrían ejecutarse en respuesta a un mensaje recibido por un objeto de dicha clase. RFC por lo tanto se calcula contando las ocurrencias de llamadas a otras clases de una clase particular. Nuevamente, esta métrica se relaciona con la complejidad: cuanto más alto sea el valor para RFC, más compleja será una clase y, por ende, es más probable que incluya errores.
Falta de Cohesión en los Métodos (LCOM)	Esta métrica establece en qué medida los métodos hacen referencia a los atributos. Se calcula como el número de pares de funciones sin variables compartidas de instancia menos el número de pares de

	funciones con variables de instancia compartidas. Un valor alto en LCOM implica falta de cohesión, es decir, escasa similitud entre los métodos siendo siempre deseable un alto grado de cohesión en los métodos de una clase.

Entre los años 70 y 80, Maurice Halstead desarrolla métricas basadas en el cálculo de palabras reservadas y variables. Su teoría está basada en una simple cuenta, muy fácil de automatizar, de operadores y operandos en un programa [9]:

- Los operadores son las palabras reservadas del lenguaje, tales como IF-THEN, READ, FOR, etc; los operadores aritméticos +, -, *, etc, los de asignación y los operadores lógicos AND, EQUAL TO, etc.
- Los operandos son las variables, literales y las constantes del programa.

Halstead distingue entre el número de operadores y operandos únicos y el número total de operadores y operandos. Define las métricas que se muestran en la tabla 2:

Tabla 2. Métricas Halstead

Volumen	La medida de longitud, N, es usada en la estimación de tamaño de Halstead llamada Volumen. Mientras que la longitud es una simple cuenta (o estimación) del total de operadores y operandos, el volumen da un peso extra al número de operadores y operandos únicos.
Nivel de un Programa	El nivel de un programa da una idea del nivel de detalle con que ha sido codificado. Se entiende que cuanto más código se usa para una función dada, de más bajo nivel es. En el límite, las llamadas a función tienen el nivel más alto, ya que su volumen real coincide con el potencial.
Inteligencia Contenida en el Programa o Contenido Inteligente	Se calcula a partir del volumen y el nivel del programa. Este valor se correlaciona bastante bien con el tiempo total de programación y depuración. Es por tanto una métrica que sirve para estimar la complejidad del programa desde estos dos puntos de vista.
Esfuerzo	El esfuerzo necesario para producir una porción de software está relacionado sobre todo con la dificultad de entenderlo e implementarlo.
Bugs por Clase	El número de errores entregados se correlaciona con el total de complejidad del software.

Otras métricas importantes resultan ser las definidas por McCabe [10] para calcular la complejidad y por Omán y Hagemeister [11] para la mantenibilidad, que se muestran en la tabla 3:

Tabla 3. Métricas de McCabe, Omán y Hagemeister

Complejidad ciclomática	Se basa en el diagrama de flujo determinado por las estructuras de control de un determinado código. El resultado define el número de caminos independientes dentro de un fragmento de código, y determina la cota superior del número de pruebas que se deben realizar

	para asegurar que se ejecuta cada sentencia al menos una vez.
Peso de los comentarios (CW)	Se cree que esta fórmula da una mejor medida de la contribución de las líneas de comentarios a la mantenibilidad. Este valor se utiliza dentro del Índice de Mantenibilidad para calcular la dificultad de mantener un sistema.
Índice de Mantenibilidad con comentarios (MIcw)	El objetivo principal de la métrica es determinar qué tan fácil será mantener un cuerpo de código particular.

Otras métricas usadas con frecuencia son LLOC y CLOC, que se muestran en la tabla 4:

Tabla 4. Métricas adicionales

Líneas de código lógicas (LLOC)	Cuenta la cantidad de líneas que representan instrucciones a la computadora y que no son comentarios.
Líneas de código comentadas	Cuenta la cantidad de líneas de comentarios.

2.3 Defectos del software

A diferencia de lo que ocurre con el software, pocos productos de otra índole acaban llegando a los consumidores con tantos defectos que afecten a su funcionalidad y a la seguridad o fiabilidad. Sin embargo, hay entornos donde es crucial que el software no falle. Se conocen casos en que por errores en el software se han producido grandes pérdidas económicas, o incluso la pérdida de vidas humanas [12]. Es por ello que, realizar las pruebas pertinentes y eliminar la complejidad es vital para que un sistema cumpla los objetivos para los que fue diseñado.

Ante este escenario, es necesario encontrar formas de llevar adelante revisiones de un producto de software dirigidas a las secciones de código que sean más propensas a errores. Una manera de dirigir este esfuerzo es mediante el uso de predictores para anticiparse a las secciones o módulos más propensos a defectos. Existen diferentes enfoques en cuanto a los predictores, pudiendo ser a partir de herramientas de análisis estático, usando métricas de cambio o métricas de código [13].

2.4 Pedidos de cambio

Una vez identificados los problemas que tiene el código, corregirlos no resulta una tarea trivial, sobre todo en proyectos grandes. Es esencial y deseable una gestión adecuada de los pedidos de cambio que atraviesa un proyecto, desde el momento que es reportado, asignado hasta que se considera finalizado.

El proceso de asignación de recursos humanos de la manera más eficiente posible puede tener un efecto directo en el cumplimiento de parámetros establecido, evitando pérdidas de tiempo, esfuerzo y calidad del producto.

Encontrar el desarrollador apropiado para un cambio solicitado es una tarea fundamental para su resolución de la manera más eficiente. Sin embargo, resulta una tarea difícil de automatizar y que consume tiempo. Requiere de diferentes habilidades cognitivas, comunicación con miembros del equipo y conocimiento del proyecto, tanto de los aspectos técnicos como organizacionales. Además, la reasignación puede resultar un esfuerzo duplicado [14], en cuanto significa tener que asignar más de una vez el mismo pedido de cambio.

3 Metodología

3.1 Desarrollo de la herramienta

Se decidió utilizar la metodología convencional o "cascada" para el desarrollo de la herramienta, basada en la descripción del modelo hecho por Ian Sommerville [15].

Se seleccionó esta metodología por ser apropiada al contexto particular de requerimientos estables, ya que el producto fue desarrollado por una sola persona, además de que esta metodología está orientada a la documentación, algo muy valorado en este trabajo final.

- Análisis y definición de requerimientos: Se definen los servicios, restricciones y metas del sistema. Éstos constituyen la especificación del sistema.
- Diseño del sistema y del software: Se establece una arquitectura completa del sistema. El diseño del software identifica y describe las abstracciones fundamentales del sistema software y sus relaciones.
- Implementación y prueba de unidades: Durante esta etapa, el diseño del software se lleva a cabo como un conjunto de unidades o módulos. La prueba de unidades implica verificar que cada una cumpla su especificación.
- Funcionamiento y mantenimiento: Por lo general esta es la fase más larga del ciclo de vida. El sistema se instala y se pone en funcionamiento.

3.2 Especificación de Requerimientos de acuerdo a la IEEE830

En la Fig. 1 se muestra una versión simplificada de los diagramas de casos de uso y en la Fig. 2 se muestra el modelo de la base de datos.

3.3 Diseño e Implementación

Para el diseño e implementación de esta herramienta se planteó como arquitectura de software el patrón de diseño Modelo-Vista-Controlador (MVC) el cual separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones. Para ello MVC propone la construcción de tres componentes distintos que son el modelo, la vista y el controlador, es decir, por un lado, define componentes para la representación de la información, y por otro lado para la

interacción del usuario. Este patrón de arquitectura de software se basa en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento. El framework que se utilizó, Symfony, permite diferenciar fácilmente estos componentes.

3.4 Herramientas de Medición y métricas

Él software desarrollado prevé la medición de código escrito en Java o PHP. Se utilizaron herramientas externas para las mediciones, Java con **Chidamber and Kemerer Java Metrics (ckjm)** y PHP **con PhpMetrics**, adaptándolas para su uso como librerías dentro de la aplicación, incorporando los resultados en informes propios y registrando sus valores en la base de datos. De las métricas ofrecidas por las herramientas externas se seleccionaron las descritas en la tabla 5.

Tabla 5. Métricas Implementadas

Java	PHP
• Métodos ponderados por clase • Profundidad en el árbol de herencia • Número de hijos inmediatos en el árbol de herencia • Acoplamiento entre clases • Respuesta para una clase • Falta de cohesión en los métodos	• Número de líneas lógicas de código • Líneas de Código Comentadas • Contenido inteligente • Volumen • Peso de los comentarios • Complejidad ciclomática • Falta de cohesión en los métodos • Índice de Mantenibilidad

También se registran los siguientes promedios y máximos, que se muestran en la tabla 6:

Tabla 6. Métricas Implementadas Promedio

Java	PHP
• Métodos ponderados por clase promedio • Profundidad en el árbol de herencia promedio • Número de hijos inmediatos en el árbol de herencia promedio • Acoplamiento entre clases promedio • Respuesta para una clase promedio • Falta de cohesión en los métodos promedio	• Complejidad ciclomática Promedio • Promedio de Bugs por Clase (Halstead) • Máxima Complejidad ciclomática por Método

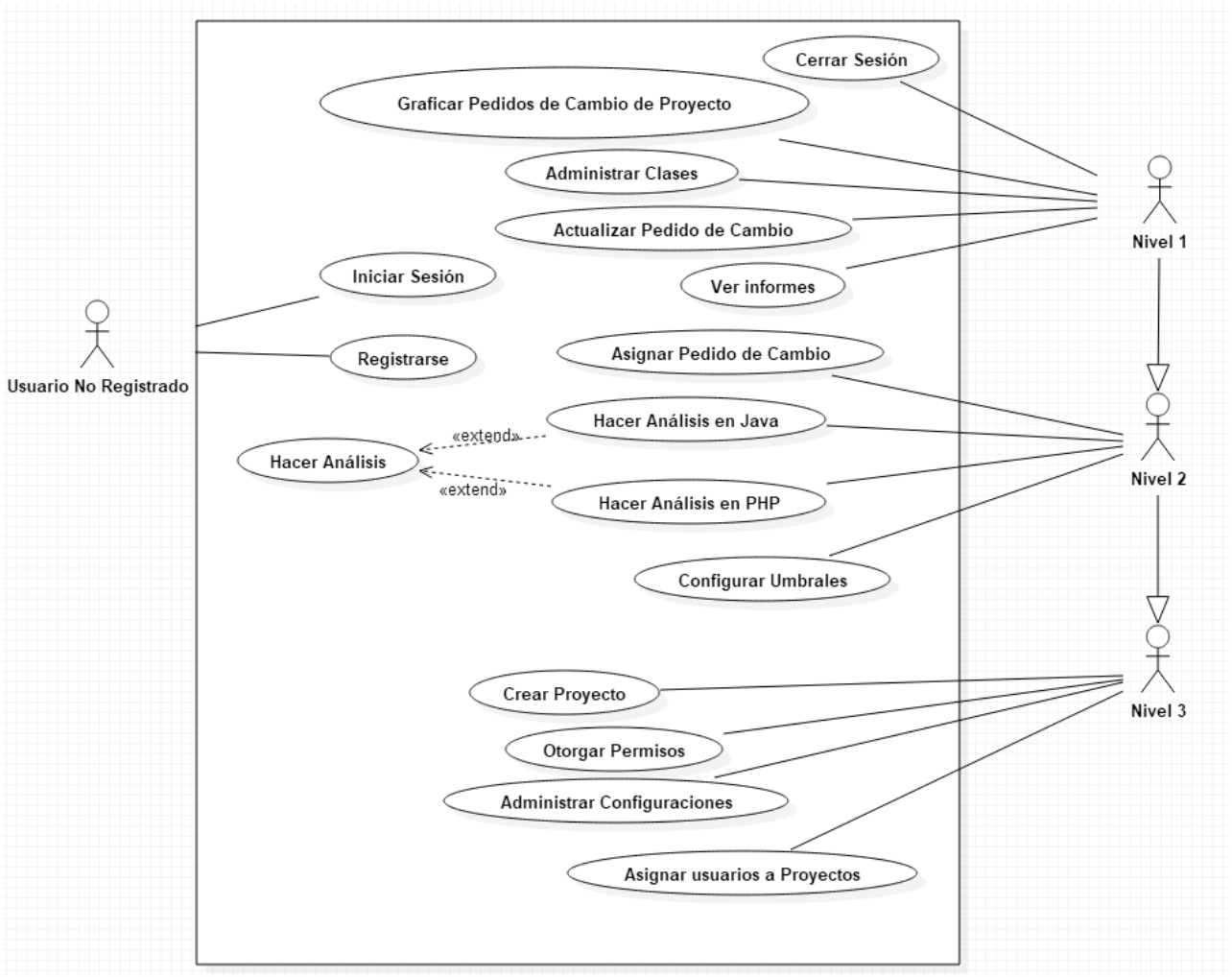

Fig. 1. Diagrama de casos de usos de la aplicación.

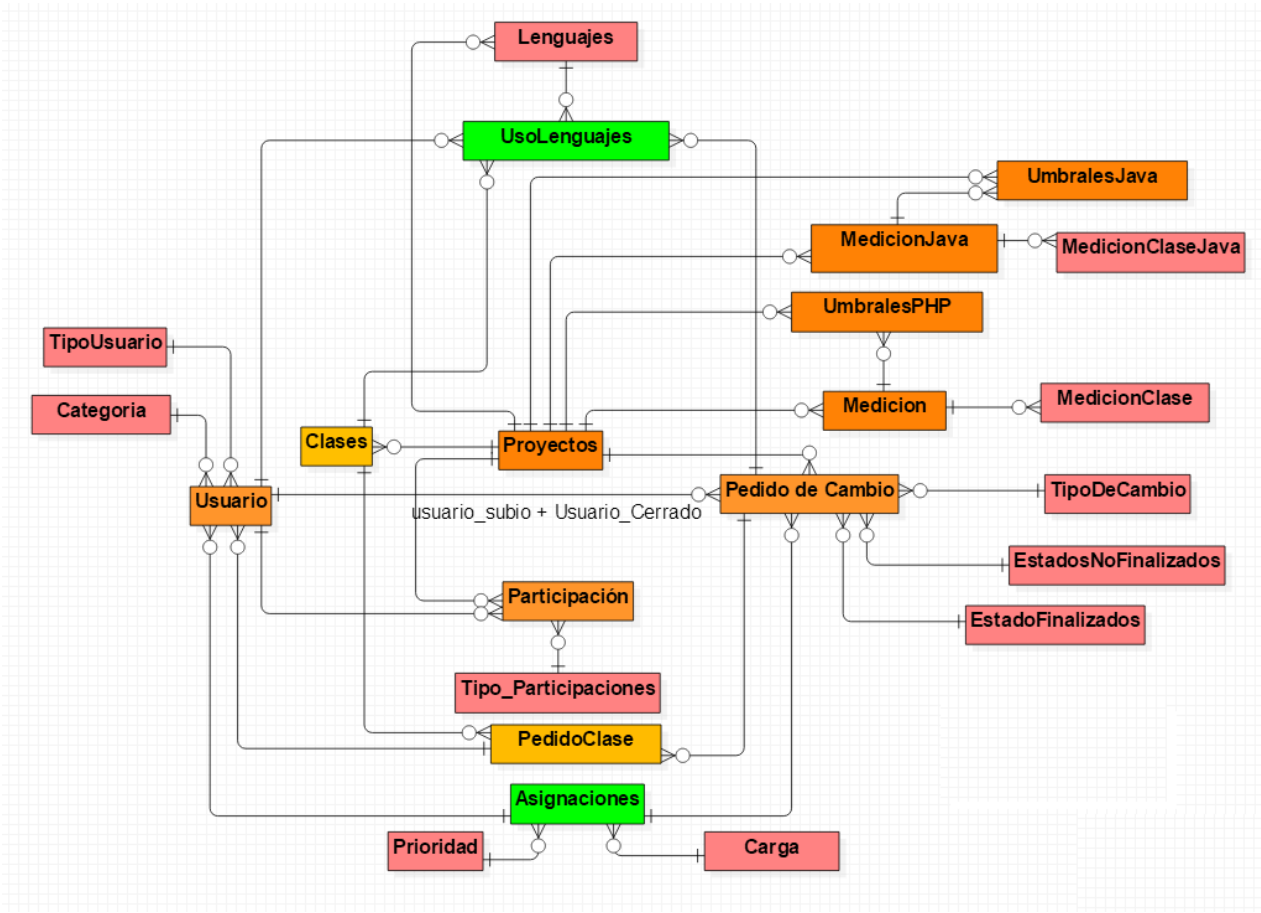

Fig. 2. Diagrama Entidad-Relación de la Base de Datos.

4 Descripción de la herramienta

La aplicación integra la administración de usuarios de la organización, pedidos de cambio realizados en un proyecto y análisis del código estático.

La pantalla es simple, con una cabecera donde figura el nombre del usuario y la opción de Logout. En la parte izquierda, una barra de acceso rápido con las opciones “Proyectos”, “Pedidos de Cambio”, “Usuarios” y “Configuraciones”.

Los proyectos son creados por usuarios con permiso de Gerente, donde se registra el nombre, fecha de inicio y fecha de entrega. El mismo u otro usuario Gerente pueden asignar usuarios al proyecto.

Cualquier usuario dentro del equipo puede generar pedidos de cambio, luego el líder del proyecto asignará el mismo a un usuario dentro del equipo. Al principio, no hay ninguna clase asociada a un pedido de cambio, pero se puede agregar la cantidad que sea necesaria.

Cuando ese pedido de cambio se considera solucionado, se lo marca como “Finalizado”, teniendo que actualizar también el estado en el que terminó.

El sistema permite realizar mediciones de código estático en Java y en PHP, teniendo cada uno su propio set de métricas. Se requiere únicamente como paso previo que estén configurados los umbrales para cada lenguaje. Cada proyecto tiene una sección de análisis donde se pueden acceder a mediciones realizadas previamente.

4.1 Interfaz general de Proyectos.

En la Fig. 3 se muestra una de los proyectos registrados por el programa. Desde aquí se puede ver información importante de los mismos organizada en cuadros desplegables, editar la información guardada al momento de crear y dar de baja si es necesario (baja lógica, no física).

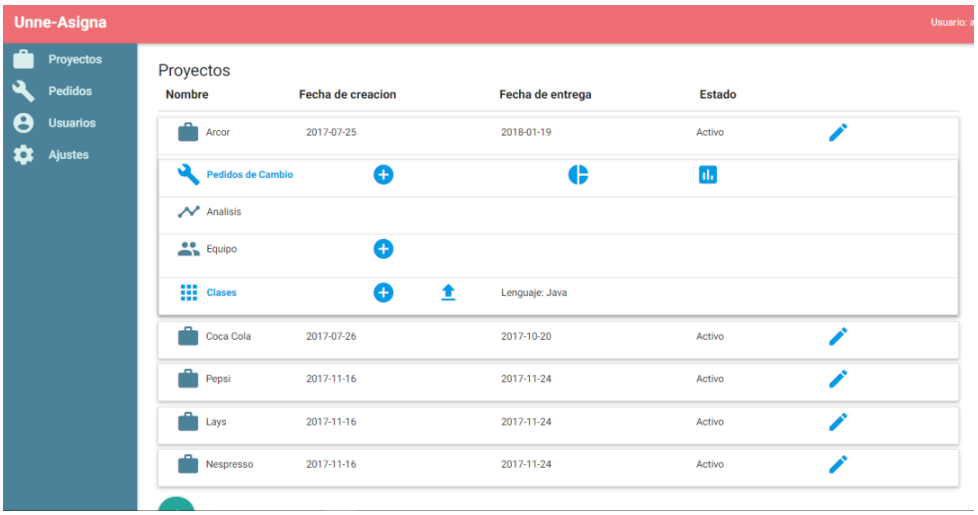

Fig. 3. Vista de todos los proyectos

4.2 Pedidos de Cambio

En la Fig. 4 se muestra el listado de Pedidos de Cambio. Se pueden mostrar todos los pedidos de cambio que el usuario puede acceder de acuerdo a sus permisos al seleccionar el botón “Pedidos de Cambio” de la barra lateral. Con los permisos necesarios, se pueden ver todos los registrados por el programa. Si se desea ver los específicos de un proyecto se puede acceder desde el botón “Pedidos de Cambio” en la sección del Proyecto.

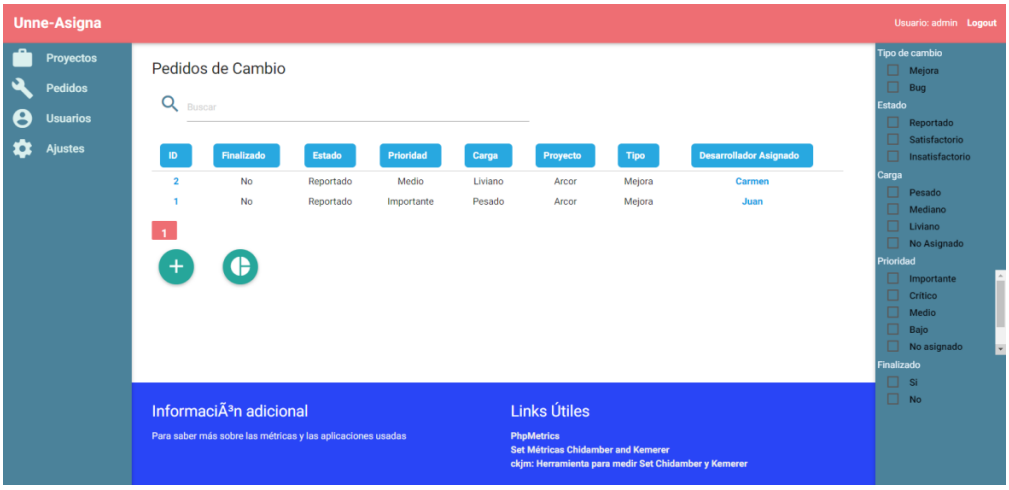

Fig. 4. Gráficos de Pedidos de Cambio.

En la Fig. 5 se muestra toda la información con respecto a un pedido de cambio. Se pueden ver las clases involucradas, editar el pedido, darlo de baja de manera lógica y marcarlo como finalizado.

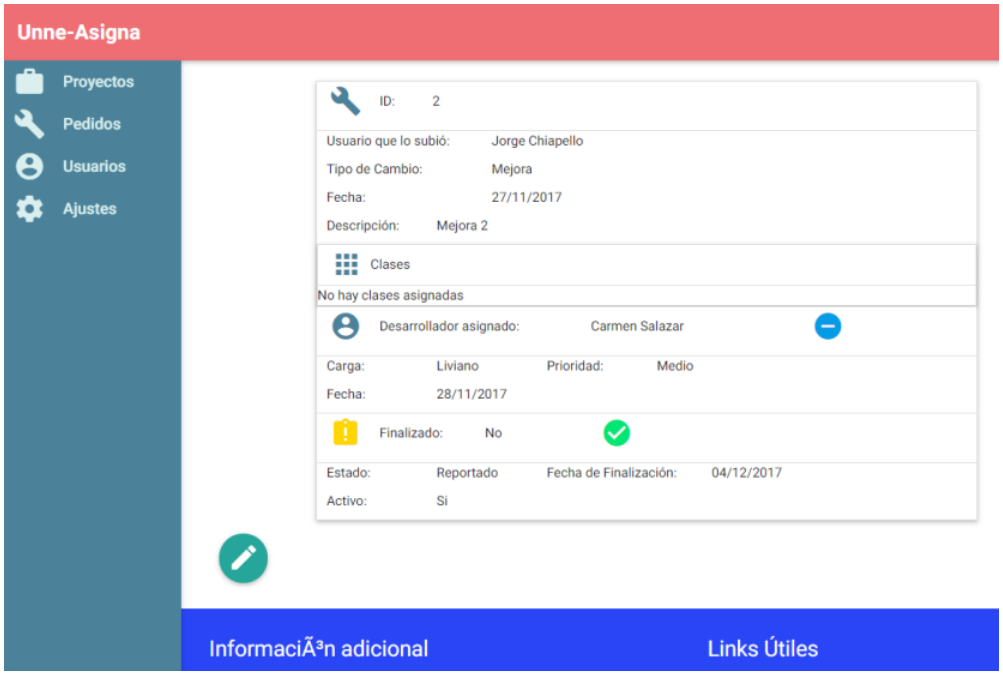

Fig. 5. Pedidos de Cambio.

4.3 Analizar Proyecto

Para realizar un análisis de un proyecto, se deben arrastrar los archivos con el código fuente de cada clase a la zona drop, todos de una sola vez. Se pueden arrastrar carpetas también, en este caso, lo registrará con el nombre de la ubicación del archivo, por ejemplo "carpeta1/Clase1.php". Hay dos analizadores para cada lenguaje, aceptando solamente archivos con extensión. php en PHP o .class en Java.

Una vez terminado el análisis, el sistema devolverá los cálculos realizados, indicando cada vez que un valor de la medición exceda los umbrales, ya sea por encima o por debajo, y mostrando un recuento final en el valor "Número de Violaciones", como se muestra en la figura 6.

Fig. 6. Informe de Análisis de Proyecto.

4.4 Gráfico Histórico Pedidos de Cambio y Graficar Pedidos de Cambio Activos

El sistema genera un resumen gráfico de la incidencia de pedidos de cambio a lo largo del tiempo del proyecto como se muestra en la figura 7. Están ordenados por mes y se puede avanzar o retroceder por años con las flechas izquierda y derecha del teclado. En la barra lateral de la derecha se encuentra los filtros de acuerdo a la carga de los pedidos de cambio y las leyendas del significado de cada barra.

Por otro lado, se puede acceder a un resumen del Estado, Prioridad y Carga de todos los Pedidos de Cambio sin finalizar, seleccionando el botón "Graficar" en la opción "Proyecto > Pedidos de Cambio", como se muestra en la figura 8.

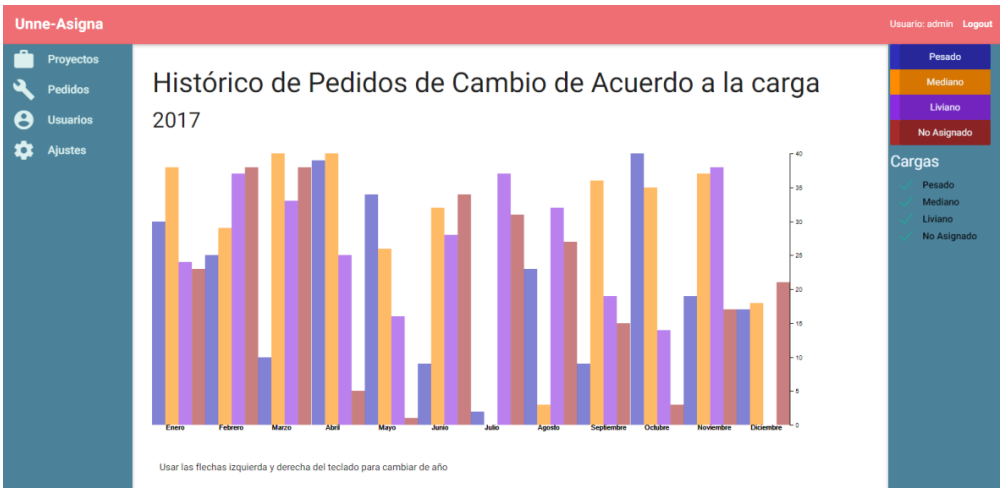

Fig. 7. Gráficos de Pedidos de Cambio.

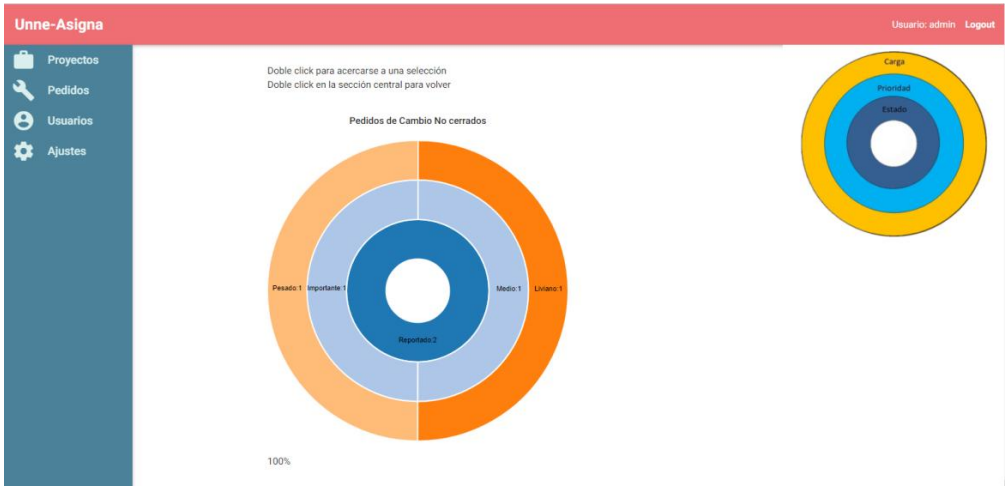

Fig. 8. Gráficos de Pedidos de Cambio.

5 Conclusiones y futuros trabajos

Producir programas funcionales (que hagan lo que se supone deben hacer), correctos (sin defectos) y mantenibles (susceptibles de evolucionar ante los cambios manteniendo las dos cualidades anteriores) y que cumplan con los estándares de calidad en programación, constituye un importante desafío en el desarrollo de software.

Pero, lo cierto es que los cambios en un programa son inevitables a lo largo del ciclo de vida del desarrollo, entonces, una cuestión importante es cómo gestionarlos adecuadamente manteniendo un cierto nivel de calidad.

Para contribuir a mejorar el proceso de desarrollo y mantenimiento de software, en este trabajo se describe una aplicación orientada al soporte de la asignación efectiva y eficiente de pedidos de cambio, apoyada en tres pilares: los pedidos de cambio, las personas que participan en el mantenimiento, y las métricas aplicadas al código que aportarán información sobre la calidad del diseño y de la programación.

El principal aporte de esta herramienta es la integración de los tres aspectos mencionados, ofreciendo información que permitirá a los responsables y participantes del proyecto (gerentes, líderes y desarrolladores) tener una visión amplia del estado del proyecto, y a la vez, un panorama de la calidad del mantenimiento correctivo y adaptativo en función de la medición del código fuente y los valores umbrales definidos, contribuyendo de este modo a la gestión cuantitativa de los proyectos de software y a la mejora de la calidad.

El producto permite aplicar métricas en proyectos PHP y Java. No permite almacenar documentos (Word, Excel, etc) como Jira, aspecto que resulta interesante pero no fundamental, al menos en una primera etapa, razón por la cual no se agregó esta funcionalidad.

Como trabajo futuro, se propone incrementar el número de informes y la modalidad de presentación de la información al usuario para la toma de decisiones. Asimismo, registrar mayor información sobre los pedidos de cambio, tal como los distintos estados que atraviesan los mismos, desde que son reportados hasta su finalización.

Por último, concretar la realización de una prueba en un contexto real de desarrollo para evaluar su comportamiento y comprobar el potencial de la herramienta.

6 Referencias

1. R. Pressman, Ingeniería de Software. Un Enfoque Práctico, Sexta ed., MCGraw Hill, 2005.
2. J. A. McCall, P. K. Richards y G. F. Walters, Factors in Software Quality, 1997.
3. International Organisation for Standardisation, ISO/IEC 9126, Geneva, Switzerland, 2001.
4. L. Gou, Q. Wang, J. Yuan, Y. Yang, M. Li y N. Jiang, «Quantitative defects management in iterative development with Bi defec, » de Sotfware Process Improvement and Practice, 14 ed., 2009, pp. 227-241.
5. Hernan, J. F. Hernandez Ballesteros y J. M. Minguet Melián, «La Medida de la Calidad del Software como Necesidad y Exigencia en Modelos Internacionales (CMMI, ISO 15504, ISO 9001),» [En línea]. Available: http://www.issi.uned.es/CalidadSoftware/Noticias/PonIng2005.rtf. [Último acceso: 12 12 2017].

6. L. A. Laranjeira, «Software Size Estimation of Object-Oriented Systems,» IEEE Transactions on Software Engineering, vol. 16, nº 5, pp. 510-522, 1990.
7. H. Gonzalez Doria, «Las Métricas de Software y su Uso en la Región,» Departamento de Ingeniería en Sistemas Computacionales, Escuela de Ingeniría, Universidad de las Américas Puebla, 2001.
8. S. Chidamber y C. Kemerer, «A metrics suite for object oriented design,» Software Engineering, IEEE Transactions, vol. 20 , nº 6, pp. 476 - 493, 1994.
9. H. H. Halstead, Elements of Software Science, Nueva York: Elsevier Science Inc., 1977.
10. T. J. McCabe, «A Complexity Measure, » IEEE Transactions On Software Engineering, vol. 2, nº 4, p. 308, 1976.
11. P. Oman y J. Hagemeister, «Metrics for assessing a software system's maintainability,» de Conference on Software Maintenance, Orlando, 1992.
12. T. Herrero, «PC World Digital,» 2002. [En línea]. Available: http://www.idg.es/pcworld/estructura/VersionImprimir.asp?idArticulo=141458.
13. M. M. Lehman, J. F. Ramil y D. E. Perry, «Metrics and Laws of Software Evolution - The nineties View,» de Proc. 4th International Software Metrics Symposium (METRICS '97), Albuquerque, NM, 1997.
14. Y. C. Cavalcanti, P. A. da Mota Silveira Neto y Ivan do Carmo Machado, «Towards Understanding Software Change Request Assignment: a survey with practitioners,» de Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering, Porto de Galinhas, Brazil, 2013.
15. I. Sommerville, Ingeniería del Software, Madrid: Pearson Educación, 2005.