Universidad Rey Juan Carlos

Escuela Técnica Superior de Ingeniería Informática

Departamento de Lenguajes y Sistemas Informáticos II

Construcción de un Entorno para la Medición Automatizada de la Calidad de los Productos Software

Autor: Emanuel Irrazábal

Director de Tesis: Dr. Javier Garzás Parra

Dr. Juan Manuel Vara Mesa

Móstoles, Febrero 2012

Universidad
Rey Juan Carlos

El Dr. D. Javier Garzás Parra, Profesor Titular de Universidad Departamento de Lenguajes y Sistemas Informáticos II de la Universidad Rey Juan Carlos de Madrid y el Dr. D. Juan Manuel Vara Mesa, Profesor Titular de Universidad Departamento de Lenguajes y Sistemas Informáticos II de la Universidad Rey Juan Carlos de Madrid, directores de la Tesis Doctoral: "Construcción de un Entorno para la Medición Automatizada de la Calidad de los Productos Software" realizada por el doctorando D. Emanuel Irrazábal,

HACEN CONSTAR QUE:

esta tesis doctoral reúne los requisitos para su defensa y aprobación

En Madrid, a 7 de Febrero de 2012

Fdo.: Javier Garzás Parra

Fdo.: Juan Manuel Vara Mesa

Papis, hermanos y Mali

Resumen

En la presente Tesis Doctoral se propone un entorno basado en software libre, para implantar un sistema de medición de la mantenibilidad software a nivel operativo, táctico y estratégico. Y por otro lado, un soporte metodológico para la evaluación de la mantenibilidad del producto software capaz de calcular el valor y el retorno de inversión de las refactorizaciones necesarias para mejorar dicha mantenibilidad.

En el desarrollo software actual, la medición es una pieza básica a la hora de controlar la calidad del producto, más aún si el desarrollo es realizado por equipos externos, donde el receptor del software debe establecer sus propios sistemas de control de la calidad. Es en este punto donde ha surgido el desarrollo de esta Tesis Doctoral y la construcción del entorno KEMIS, como una solución para aquellas empresas que desean realizar una medición de la calidad de sus productos software. Especialmente aquellas que han externalizado el desarrollo de dichos productos y necesitan un análisis de calidad automatizado, planificable e intuitivo.

El uso de la herramienta en un entorno real ha dado la pauta del valor agregado al utilizar KEMIS. Ha recibido una amplia aceptación avalada por buenos resultados y se ha podido comprobar la dificultad que tienen las empresas al intentar mejorar el código fuente manualmente. Esta necesidad ya había sido identificada en los estudios previos y los trabajos futuros abordarán esta problemática.

Así mismo, las empresas de desarrollo arriesgan continuamente gran cantidad de recursos al priorizar de manera informal las mejoras del software. El entorno KEMIS también ayuda a resolver el riesgo que supone esta toma de decisión sin tener suficiente información o un mecanismo sistemático a seguir. Una de las mayores contribuciones de esta Tesis Doctoral es el modelo de medición del valor para priorizar las refactorizaciones del software. Uno de los aspectos por donde continuar avanzando en el modelo de medición es en los beneficios particulares de cada mejora propuesta. Por último otro aspecto a tener en cuenta en los trabajos futuros es el grado de automatización de los indicadores de valor.

Extended Abstract

In this Thesis proposes an environment based on free software to implement a system of software maintainability measurement at operational, tactical and strategic level. On the other hand, we proposed a methodological support for the evaluation of product quality software to calculate the value and ROI solutions necessary to improve maintainability.

In the current software development, measurement is a basic part in controlling product quality, especially if the development is carried out by external equipment, where the software receiver must establish their own quality control. At this point we have been started the development of this thesis and the building of KEMIS environment as a solution for companies wishing to make a measurement of the quality of their software products. Especially those that have outsourced the development of software products and require automated quality analysis, schedulable and intuitive.

The use of the tool in a real environment has given us the measure about the value added by using KEMIS. It has been widely accepted and supported by good results and have been shown the difficulty that companies have to try to improve the source code manually. This need had been identified in previous studies and future works will address this problem.

Also, development companies continually risking large amounts of resources to informally prioritize software enhancements. KEMIS environment also helps solve the risk involved in taking this decision without sufficient information or a systematic mechanism to follow. One of the major contributions of this thesis is the measurement model to prioritize the value of software refactorings. One of the aspects where further progress in the measurement model of the benefits of each proposed improvement. Finally another aspect to consider in future work is the degree of automation of the indicators of value.

Agradecimientos

En general, quiero agradecer a todos los que están leyendo esta Tesis. Especialmente a los que lo hacen movidos por el afecto, incluso aunque no lean esta oración. **A vos que estás leyéndola, gracias.**

A papá y mamá (María del Carmen, Ramón Ismael), hermanos (Ringo y Yango) y sus familias, sobrinos (Juan Marcos, Ignacio), Mali (vales oro) y su familia, tíos, primos, amigos de Argentina y de España, compañeros de trabajo, y, por supuesto, mis amigos de San Miguel. Ellos han aportado en lo que soy, y, por lo tanto, en lo que hago. Gracias.

A Javier, Juancho y Esperanza, por la dirección de la Tesis, y, en general, la dirección de mi carrera profesional estos últimos años. El apoyo, el trabajo, el aliento y la crítica constructiva han estado siempre allí. Gracias.

A todos los que me olvido. Te aseguro que al escribir la Tesis me he acordado de ti. Gracias.

Índice

Lista de Figuras

Lista de Tablas

Introducción

En la presente Tesis Doctoral se propone un entorno basado en software libre, para implantar un sistema de medición de la mantenibilidad software a nivel operativo, táctico y estratégico. Y por otro lado, un soporte metodológico para la evaluación de la mantenibilidad del producto software capaz de calcular el valor y el retorno de inversión de las refactorizaciones necesarias para mejorar dicha mantenibilidad.

En la sección 1.1 se presentan las motivaciones que han llevado a tomar la decisión de realizar este trabajo. En la sección 1.2 se establece la hipótesis principal y los objetivos directamente derivados de ella. En la sección 1.3 se describe el marco de investigación seguido durante el desarrollo de esta tesis. En la sección 1.4 se describe el contexto en que se desarrolla este trabajo, haciendo referencia a los proyectos de investigación en los que ha participado el doctorando. En la sección 1.5 se resume el método de investigación utilizado. Y finalmente, en la sección 1.6 se proporciona una visión general del resto del documento.

1.1 Planteamiento del Problema y Enfoque

Ante un mercado cada vez más competitivo y en constante desarrollo, la calidad del software está tomando mayor importancia en las organizaciones, y con ello, la medición software [1] ha adquirido mayor relevancia como principal herramienta para medir la calidad del software. La calidad software no sólo tiene influencia en los costes finales, sino que es también un factor clave para mejorar la imagen frente a los clientes y como elemento diferenciador de la competencia [2].

Aunque la calidad puede describirse desde diferentes perspectivas, la calidad software está tradicionalmente relacionada con la calidad del producto y la calidad del proceso [3]. Esto es, la calidad en el proceso de desarrollo para obtener el producto (actividades, tareas, etc. del desarrollo y mantenimiento del software) y la calidad del propio producto software. Debido a ello, se ha incrementado la tendencia hacia la institucionalización de buenas prácticas de calidad en organizaciones de desarrollo software. Dicha tendencia se ha visto reflejada en un aumento constante del número de organizaciones que tienen la intención de alinear sus procesos con modelos de referencia de proceso ampliamente conocidos, como CMMI-DEV [4] o ISO/IEC 12207 [5].

Sin embargo, hasta ahora la principal crítica a la evaluación de la calidad de los procesos ha sido la poca evidencia existente sobre el hecho de que cumplir un modelo de procesos de referencia asegure la calidad del producto [6]. De hecho, la estandarización de los procesos garantiza la uniformidad en la salida de éstos, lo que puede incluso institucionalizar la creación de malos productos. En esta línea, Maibaum y Wassyng [7], comentaban que las evaluaciones de calidad deberían estar basadas en evidencias extraídas directamente de los atributos del producto, no en evidencias circunstanciales deducidas desde el proceso. Sintetizando, un proceso estándar no necesariamente concluye con un producto de calidad.

Como respuesta, han aparecido diferentes modelos de referencia para la calidad del producto. Sin embargo, es difícil llevar a la práctica la implantación de estos modelos. La principal barrera viene del carácter abierto de estas normas. Esto es, no muestran detalles sobre qué métricas utilizar o cuáles son las más adecuadas; cómo agrupar los valores de las métricas para obtener métricas de más alto nivel; o cuáles son los valores umbrales recomendados para cada métrica.

Con la intención de paliar los efectos de esta situación la Organización Internacional para la Estandarización (ISO) ha publicado una serie de estándares relacionados con la calidad del producto. Uno de estos estándares o normas es la ISO/IEC 9126:1991 Software Engineering – Product Quality [8]. El principal objetivo de esta norma es proporcionar un marco de referencia para definir y evaluar la calidad del producto software. Para ello la norma especifica un modelo de calidad identificando un conjunto de características de calidad. Así mismo, en 2005 se publica la serie de estándares ISO/IEC 25000. Esta serie, también denominada Software product Quality Requirements and Evaluation (SQuaRE) [9], la conforman un conjunto de normas internacionales orientadas a la calidad del producto; teniendo como objetivo organizar y unificar los estándares que especifican los requerimientos de calidad del software y su evaluación. Más recientemente, en marzo del 2011 se publica la norma ISO/IEC 25010 [10], que reemplaza a la norma ISO/IEC 9126 y define un modelo de calidad para los productos software.

Una de las características definidas en el modelo de calidad del producto de la norma ISO/IEC 25010 es la mantenibilidad[1]. Históricamente la mantenibilidad ha sido reconocida como una de las características más relevantes del software

[1] La palabra “mantenibilidad” no existe oficialmente en la lengua española, pero es utilizada masivamente como españolización de la palabra inglesa “maintainability”. Esta y otras características de calidad españolizadas deben ser interpretadas como “la capacidad de …”. En este caso, mantenibilidad: “capacidad de mantenimiento”.

debido a su impacto directo sobre el coste de desarrollo y el propio mantenimiento. De hecho, estudios previos señalan la fase de mantenimiento como la fase donde más recursos se invierten en el ciclo de vida del software, implicando dos veces el coste de la fase de desarrollo [11][12][13][14][15]. Por ejemplo, según [14], el tiempo que un programador invierte en mantenimiento es alrededor del 61% frente al 39% dedicado al desarrollo. Y en ocasiones la proporción de coste que supone el mantenimiento ronda el 90% [17].

A pesar de que el modelo de calidad del producto descrito en la norma ISO/IEC 25010 no proporciona un conjunto consensuado de medidas para calcular la mantenibilidad, sí establece las propiedades de tales medidas. Éstas han de ser objetivas, reproducibles, independientes, expresadas mediante escalas válidas y suficientemente precisas para realizar comparaciones fiables. Teniendo estas premisas en mente, y el hecho de que en la actualidad los productos software suelen tener un gran tamaño, con lo que el número de medidas a obtener es muy elevado, parece que el enfoque más conveniente sería **disponer de herramientas que facilitaran la obtención automática de tales medidas.**

Una vez identificada la mantenibilidad de un producto software, el siguiente paso es intentar mejorarla. Actualmente la mayoría de las prácticas de Ingeniería del Software centradas en la mejora de la mantenibilidad adoptan un enfoque de "valor neutro", tratándose con igual importancia y sin tener en cuenta el valor que aporta cada elemento al negocio. Como lo explica Barry Boehm en [16] "…tiempo atrás, cuando las decisiones que afectaban al software tenían relativamente poca influencia en costes, planificación y valor de las compañías, una aproximación neutral en valor podía ser razonable, pero en la actualidad estas decisiones tienen gran repercusión…". Para cubrir estas carencias, una nueva rama de la ingeniería está emergiendo, la Ingeniería del Software Basada en Valor (ISBV). La ISBV establece que las funcionalidades de un sistema tienen diferente grado de importancia y que algunas de ellas aportan más "valor" que otras, entendiendo el valor como "la cuantificación de la importancia que un determinado artefacto o tarea tiene para todos los implicados en ese sistema" [17].

En este sentido, un campo donde podría resultar interesante aplicar los principios de la ISBV es en la refactorización de software, priorizando las mejoras de acuerdo con el valor que aporten al sistema. El término refactorización fue introducido por primera vez en [19], definiéndose como una transformación parametrizada de un programa preservando su comportamiento que automáticamente modifica el diseño de la aplicación y el código fuente subyacente [20][21]. Para Fowler refactorizar es "cambiar la estructura interna del software

para hacerlo más fácil de entender y más económico de modificar sin cambiar su comportamiento visible" [22]. Por lo tanto, la refactorización puede verse como una técnica para aumentar la mantenibilidad del software [23] o como una forma de mantenimiento preventivo, cuyo objetivo es disminuir la complejidad del software en anticipación a los incrementos de complejidad que los cambios pudieran traer [24].

Por lo tanto, para que dicho mantenimiento se realice siguiendo las normas de la ISBV, se debería identificar el valor aportado por las posibles refactorizaciones. De esta manera, se conseguiría aumentar la mantenibilidad del software rápidamente, ayudando a la organización a construir un software mejor y reduciendo los costes de mantenimiento, que por otra parte, como ya hemos mencionado, tienen una gran influencia en la evaluación del coste de los sistemas [25].

Es en este punto donde surge el entorno de calidad KEMIS (Kybele Environment Measurement Information System), como una solución que alinea con la norma ISO/IEC 9126 e ISO/IEC 25010 y la ISBV para aquellas organizaciones que desean realizar una evaluación y mejora de la calidad de sus productos software. La idea de desarrollar KEMIS surge al observar los retrasos que aparecen en el ciclo de vida de desarrollo de un producto software, y en concreto en la etapa de mantenimiento debido a la poca calidad del código fuente. En este sentido, las métricas son un buen medio para analizar, guiar, monitorizar y predecir el desarrollo de un producto software y los proyectos de mantenimiento asociados [26]. Sin embargo, las métricas tradicionales no son prácticas para comparar cuantitativamente la complejidad entre diferentes productos [27]. A partir de esta necesidad surge la idea de desarrollar KEMIS, que permitirá obtener indicadores de calidad del producto, a partir de métricas básicas extraídas mediante diferentes herramientas de software libre, proporcionando un soporte metodológico y tecnológico [28] para la medición de la calidad del producto.

En este sentido, la herramienta KEMIS se centrará en la mantenibilidad del producto software, definiendo un modelo de medición adaptable. Así, realizará una serie de mediciones sobre el producto a evaluar valiéndose de herramientas de software libre. Y a partir del modelo de medición definido, calculará las características definidas por la norma ISO/IEC 9126. Por otro lado, KEMIS calculará el valor y el retorno de inversión de las refactorizaciones necesarias para mejorar la mantenibilidad y resolver los problemas encontrados.

Los valores obtenidos se corresponden con las características y subcaracterísticas especificadas en la norma ISO/IEC 9126, y más adelante en la norma ISO/IEC 25010.

De este modo, se obtienen una serie de ventajas:

- Permitir que las mediciones se realicen de manera periódica y frecuente, lo que ayudará a detectar desviaciones en la calidad del producto lo más pronto posible, permitiendo abaratar los costes de modificación y corrección de deficiencias en el producto.
- Permitir que las mediciones se realicen durante todo el ciclo de vida del proyecto, desde el comienzo del desarrollo hasta el final de la etapa de mantenimiento.
- Minimizar los errores de cálculo en la obtención de las métricas, mejorando la exactitud de los resultados obtenidos.
- Posibilidad de ajustar la exigencia de las mediciones a las necesidades del proyecto mediante la construcción de un modelo de medición configurable.
- Informar sobre la mantenibilidad del producto de una manera sencilla y eficiente, que permita centrarse en el análisis e interpretación de los resultados, olvidándose del proceso de adquisición de los mismos.
- Presentar los valores obtenidos de la medición por niveles de abstracción.

Además, KEMIS proporcionará una interfaz Web que soportará el acceso remoto al cuadro de mando donde se muestran los resultados de las mediciones. Para ello ofrece distintos modos de visualización seleccionados para mostrar la información recopilada sobre el proyecto de una manera útil e intuitiva para el usuario.

1.2 Hipótesis y Objetivos

La **hipótesis** planteada en esta Tesis Doctoral es la siguiente: "*es factible la definición de un entorno que facilite la evaluación de la mantenibilidad del producto software, calculando el valor y el retorno de inversión de las posibles refactorizaciones necesarias para mejorar dicha mantenibilidad*".

El **objetivo principal** de este trabajo de investigación, derivado directamente de la hipótesis, es: "*la definición de un entorno que facilite la evaluación de la mantenibilidad del producto software, calculando el valor y el*

retorno de inversión de las posibles refactorizaciones necesarias para mejorar dicha mantenibilidad".

Para la consecución de este objetivo se han planteado los siguientes objetivos parciales:

O1. Análisis y evaluación de trabajos relacionados con los modelos de medición de la calidad del producto software, las refactorizaciones y su priorización basada en aspectos del valor; incluyendo propuestas metodológicas, herramientas, etc.

O2. Desarrollo de un modelo de calidad, donde se permitan configurar los diferentes elementos del modelo para personalizar la medición de la mantenibilidad entre proyectos. Dicho entorno estará compuesto de:

- **O2.1.** Una infraestructura básica que integra la ejecución de herramientas que generan informes con métricas básicas de calidad.
- **O2.2.** Una infraestructura avanzada que, a partir de la lectura de métricas básicas, genere y persista el valor de subcaracterísticas y características de calidad.

O3. Evaluación de la mantenibilidad del producto software:

- **O3.1.** Evaluación de la mantenibilidad del producto software a distintos niveles de abstracción, proporcionando un cuadro de mando de indicadores, métricas derivadas y métricas básicas.

O4. Desarrollo de un modelo de medición del valor. Dicho modelo permitirá:

- **O4.1.** Determinar el retorno de inversión de realizar una refactorización en el software.
- **O4.2.** Priorizar las diferentes refactorizaciones.

O5. Validación del entorno. Dicha validación se realizará desde dos puntos de vista. Como prueba de concepto de la herramienta, se realizarán:

- **O5.1.** Una infraestructura Web para centralizar las evaluaciones y las mediciones del valor.
- **O5.2.** Una prueba piloto de la herramienta y la evaluación de los resultados.

1.3 Áreas de Conocimiento Implicadas

Con el objetivo de aumentar el grado de estructuración de este trabajo se toma como marco de referencia el documento Software Engineering Body of Knowledge (SWEBOK) [29], creado por la Software Engineering Coordinating Committee y promovido por la IEEE Computer Society. El documento se define como una guía del conocimiento en el área de la Ingeniería del Software y supone un paso esencial hacia el desarrollo de la profesión, porque representa un amplio consenso respecto a los contenidos de la disciplina, que ordena todo el conocimiento existente en una ontología acordada en el seno de la IEEE. Su última versión ha sido liberada en el año 2004, con cambios menores realizados en el año 2008. Se espera una nueva versión actualizada a partir del segundo trimestre del año 2012.

Según el SWEBOK, el conocimiento en Ingeniería del Software está organizado en 10 Áreas de Conocimiento: Requisitos, Diseño, Construcción, Pruebas, Mantenimiento, Gestión de la Configuración, Gestión de proyectos, Ingeniería de Procesos, Herramientas y Métodos, y Calidad. En la última actualización (todavía no publicada) se han propuesto dos nuevas Áreas de Conocimiento: Medición y Seguridad.

Tal como ilustra la Figura 1-1, de acuerdo con el SWEBOK este trabajo se encuentra enmarcado dentro de dos áreas de conocimiento:

- En primer lugar, se enmarca en el Área de “Software Quality” (Calidad del Software), en la subárea “Practical Considerations” (Consideraciones Prácticas) y en las secciones denominadas “Software Quality Management Techniques” (Técnicas de Gestión de la Calidad del Software) y “Software Quality Measurement” (Medición de la Calidad del Software). Así mismo, este trabajo está relacionado con la subárea “Software Quality Fundamentals” (Fundamentos de la Calidad del Software). En particular con las secciones denominadas “Value and Costs of Quality” (Valor y Costes de la Calidad) y “Quality Improvements” (Mejoras en la Calidad).
- Por otro lado, este trabajo se enmarca dentro del Área “Software Maintenance” (Mantenimiento del Software), en la subárea “Key Issues in Software Maintenance” (Cuestiones Claves en el Mantenimiento Software) y en las secciones denominadas “Maintenance Cost Estimation” (Estimación del Coste de Mantenimiento) y “Software Maintenance Measurement” (Medición del Mantenimiento del Software).

Es interesante resaltar la similitud entre las áreas de conocimiento del SWEBOK y las áreas de conocimiento basado en valor definidas por Boehm, y que son presentadas en detalle en la sección 2.3.1. De las 10 y 8 áreas definidas respectivamente, seis son comunes a ambas clasificaciones. Se puede decir que se trata de dos formas distintas pero similares de clasificar todo el conocimiento existente en Ingeniería del Software, lo que implica que la ISBV es un área de conocimiento transversal, que no puede ser encuadrada en ninguna sub-área concreta de las identificadas en el SWEBOK.

Los conceptos y técnicas globales de orientación al valor aplican a todas las áreas de conocimiento del SWEBOK. No obstante, cada una de las distintas partes que compone la ISBV sí pueden ser mapeadas a áreas concretas de conocimiento definidas en el SWEBOK. En particular, el Diseño Basado en Valor aplicaría al área de diseño, el Mantenimiento Basado en Valor al área de mantenimiento, y así sucesivamente.

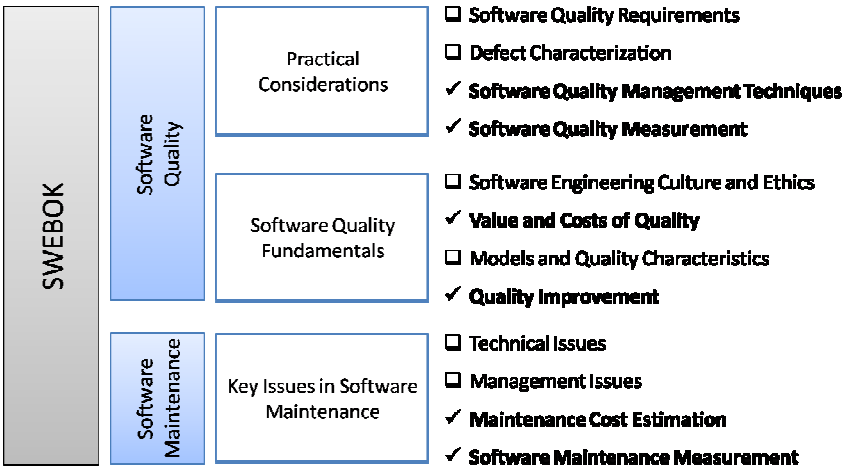


Figura 1-1. Árbol de tópicos del área "Software Quality" y "Software Maintenance" de SWEBOK

La Figura 1-2 ilustra esta correspondencia entre áreas de conocimiento. En la parte derecha se sitúan las áreas de conocimiento de ISBV definidas por Boehm, y en la parte izquierda las definidas en el SWEBOK. Por ejemplo, la "arquitectura" es definida por Boehm como un área diferenciada, mientras que en el SWEBOK forma parte del área de diseño. Por el contrario, el diseño y la construcción definidos como áreas diferenciadas en el SWEBOK, son vistos como una sola área por Boehm.

Figura 1-2. Relación entre áreas de conocimiento

1.4 Marco de Investigación

Las aportaciones presentadas en este trabajo se han desarrollado en el marco del proyecto KEMIS de I+D+i (IDI-20090175) financiado por el Centro para el Desarrollo Tecnológico Industrial (CDTI). Adicionalmente, se ha colaborado con la empresa Kybele Consulting para el desarrollo del marco tecnológico y con importantes empresas españolas para la realización de pruebas piloto y validación de la herramienta y el entorno metodológico.

1.4.1 Kybele Consulting

La empresa Kybele Consulting es una spin-off de capital 100% español que integran investigadores y profesionales en Tecnologías de la Información, Sistemas Informáticos e Ingeniería del Software

Nace del grupo de investigación Kybele (Universidad Rey Juan Carlos), al que se han unido después profesionales con amplia experiencia en unidades de desarrollo, fábricas software, calidad del producto software y mejora de procesos.

1.4.1.1 El proyecto I+D+i CDTI

El Centro para el Desarrollo Tecnológico Industrial (CDTI) es una Entidad Pública Empresarial, dependiente del Ministerio de Ciencia e Innovación, que promueve la innovación y el desarrollo tecnológico de las empresas españolas. Desde el año 2009 es la entidad del Ministerio de Ciencia e Innovación que canaliza las solicitudes de financiación y apoyo a los proyectos de I+D+i de empresas españolas en los ámbitos estatal e internacional. El proyecto KEMIS ha sido financiado por el CDTI (IDI-20090175) y su objetivo es desarrollar un entorno basado en software libre que permite, tanto a empresas que subcontratan el desarrollo software como a los propios departamentos o fábricas de desarrollo, evaluar y mejorar la calidad de las aplicaciones de forma simple, repetible, económica y eficiente.

1.5 Método de Investigación

El método de investigación que se sigue en esta tesis es la adaptación del propuesto por Marcos & Marcos en [30] para la investigación en Ingeniería del Software. Dicho método se basa en el hipotético-deductivo propuesto por Bunge [31] y se compone de una serie de pasos (ver Figura 1-3) lo suficientemente generales como para ser aplicado a cualquier tipo de investigación.

En el capítulo 2 se introducirán las bases teóricas en las se enmarca esta tesis (cuerpo del conocimiento), a partir de las cuales se ha realizado la definición del problema. Para ello, se ha definido la hipótesis, como punto de partida de esta investigación, así como el conjunto de objetivos que nos hemos marcado para alcanzar la solución (Sección 1.2).

Como se puede ver en la Figura 1-3 la definición del método de investigación es un paso más del mismo método. Los autores [30] recomiendan definirlo de esta manera ya que, la naturaleza de cada investigación tiene sus propias características y por lo tanto, no sería conveniente aplicar un único método universal de investigación.

A continuación se resume el método específico usado en esta tesis para las etapas de documentación, resolución y validación.

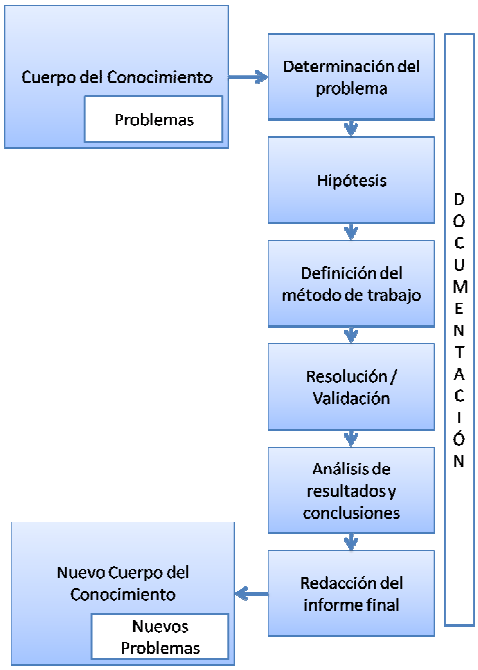


Figura 1-3. Método de investigación

1.5.1 Etapa de documentación

Para el desarrollo de esta etapa se ha utilizado el método de revisiones sistemáticas propuesto en [32]. Este método sirve para identificar, evaluar, e interpretar toda la información relativa a un tema de investigación en particular, de un modo sistemático y replicable. Surge de la investigación en el campo de la medicina y se ha convertido en una metodología confiable, rigurosa y auditable. La aplicación de las revisiones sistemáticas en el ámbito de la Ingeniería del Software permite dar un valor científico a la revisión de la literatura que se hace, definir una estrategia de búsqueda de los trabajos a evaluar y obtener finalmente una hipótesis a favor o en contra de dicha literatura.

Para el desarrollo de este trabajo de investigación, se ha tomado como referencia la adaptación del método de revisiones sistemáticas para Ingeniería del Software presentado en [33]. En dicho documento se propone una nueva aproximación en la cual el proceso de revisión sistemática está compuesto por tres grandes fases: *planificación*, *ejecución* y *análisis de los resultados* obtenidos (Figura 1-4).

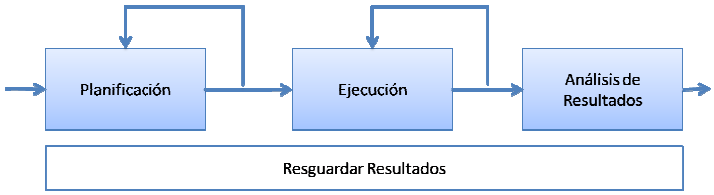


Figura 1-4. Proceso de método de revisiones sistemáticas

En la fase de **planificación** se debe establecer claramente el objetivo de la investigación y definir el protocolo de revisión a utilizar. Es decir, definir un protocolo, estableciendo el método que será utilizado a lo largo de la realización de la revisión. Además, se deben identificar los criterios de inclusión y exclusión que se seguirán para determinar las fuentes de investigación y los estudios (o documentos) a seleccionar.

En la fase de **ejecución** se lleva a cabo lo planificado en la etapa anterior, por lo que en primer lugar se debe determinar el conjunto de estudios a revisar. Estos estudios se seleccionan a través de la evaluación, para cada uno de ellos, de los criterios de inclusión y exclusión determinados anteriormente. Como paso final en esta fase se extrae la información más relevante de cada estudio.

En la fase de **análisis de resultados**, se debe sintetizar y evaluar la información extraída de cada estudio.

Por último, se debe mencionar que la fase de **resguardo de resultados** se realiza durante todo el proceso de la revisión sistemática, ya que a medida que se ejecutan cada una de las fases, el resultado de las mismas debe ser registrado.

Como se puede ver en la Figura 1-4, existen puntos de verificación a lo largo del proceso. El primer punto garantiza que la planificación realizada es la adecuada; para ello, se debe evaluar el protocolo definido y, si hubiera algún problema o incongruencia, se debería volver a la fase anterior, la planificación. El segundo punto de verificación debe realizarse una vez acabada la fase de ejecución; de la misma manera que en el punto anterior, si hubiera algún error en los resultados de esta etapa se debería ejecutar de nuevo la selección y/o extracción de información. En la sección 2, se presentará la revisión sistemática realizada conforme al proceso que se acaba de describir.

1.5.2 Etapa de resolución y validación

Para la etapa de resolución y validación se utilizará el método Investigación-Acción. El término “Investigación-Acción” proviene del autor Kart

Lewin [34] que lo utilizaba para describir una forma de investigación que podía enlazar el enfoque experimental de las ciencias sociales con programas de acción social que respondieran a los problemas sociales principales de entonces. Mediante la Investigación-Acción, Lewin argumentaba que se podían lograr de forma simultánea avances teóricos y cambios sociales. A pesar de que la primera propuesta de Investigación-Acción fue introducida en 1985 [35], en los últimos años ha obtenido una gran atención y aceptación por parte de la comunidad investigadora en Sistemas de Información [36][37].

En realidad, Investigación-Acción no se refiere a un método de investigación concreto, sino a una clase de métodos que tienen en común las siguientes características:

- Orientación a la acción y al cambio
- Focalización en un problema
- Un modelo de proceso "orgánico" que engloba etapas sistemáticas y algunas veces iterativas y
- Colaboración entre los participantes.

Se define este método de investigación como "la forma que tienen los grupos de personas de preparar las condiciones necesarias para aprender de sus propias experiencias y hacer estas experiencias accesibles a otros" [38].

La Investigación-Acción tiene una doble finalidad: generar un beneficio al "cliente" de la investigación y, al mismo tiempo, generar "conocimiento de investigación" relevante [39]. Por tanto, Investigación-Acción es una forma de investigar de carácter colaborativo, que busca unir teoría y práctica entre investigadores y profesionales mediante un proceso de naturaleza cíclica.

La Investigación-Acción está orientada a la producción de nuevo conocimiento útil en la práctica, que se obtiene mediante el cambio y/o búsqueda de soluciones a situaciones reales que le ocurren a un grupo de profesionales [36]. Esto se consigue gracias a la intervención de un investigador en la realidad del mencionado grupo. Los resultados de esta experiencia deben ser beneficiosos tanto para el investigador como para los profesionales.

En el campo de los Sistemas de Información, el cliente de una investigación es normalmente una organización a la cual el investigador suministra servicios tales como consultoría, ayuda para cambiar o desarrollo de software, a cambio de tener acceso a datos de interés para la investigación y, en muchos casos, de recibir financiación [39]. En cualquier caso, el investigador que utiliza Investigación-Acción en Sistemas de Información (IA-SI) sirve a dos

entidades diferentes: el cliente de la investigación y la comunidad científica de Sistemas de Información. Las necesidades de ambos suelen ser muy diferentes y, a veces, opuestas entre sí. Intentar satisfacer ambas demandas es el principal desafío que todos los investigadores de IA-SI tienen que enfrentar.

En este caso se aplicará el método de Investigación-Acción como resultado de la colaboración entre la actividad investigadora del grupo de investigación Kybele de la Universidad Rey Juan Carlos, y su aplicación en el proyecto de I+D+i KEMIS (IDI-20090175), financiado por el CDTI. Los roles identificados en Investigación-Acción [40] son el investigador, el objeto investigado, el grupo crítico de referencia (GCR) y los beneficiarios.

La Tabla 1-1 muestra los roles asociados a nuestro caso concreto. El rol de investigador lo tienen el doctorando y el grupo de investigación Kybele, siendo el objeto de investigación el identificado en el apartado 1.2. El grupo crítico de referencia está compuesto por los profesionales de desarrollo software usuarios de la herramienta KEMIS (tanto en pruebas piloto como en las pruebas internas realizadas).

La Figura 1-5 muestra la relación de propuesta, validación y refinamiento de propuestas, involucrando al grupo investigador, el grupo crítico de referencia y los beneficiarios. El proceso consta de una fase de propuesta de técnicas, que será llevada a cabo por el grupo investigador (Kybele) en conjunción con las pruebas piloto realizadas. Posteriormente, se contrasta el modelo a través de las validaciones realizadas por el doctorando, con los datos de proyectos reales suministrados por clientes de la empresa Kybele Consulting, quien participa en el desarrollo técnico de la herramienta KEMIS. Finalmente, se refinan las propuestas iniciales utilizando las mediciones y la experiencia de uso por parte de los clientes.

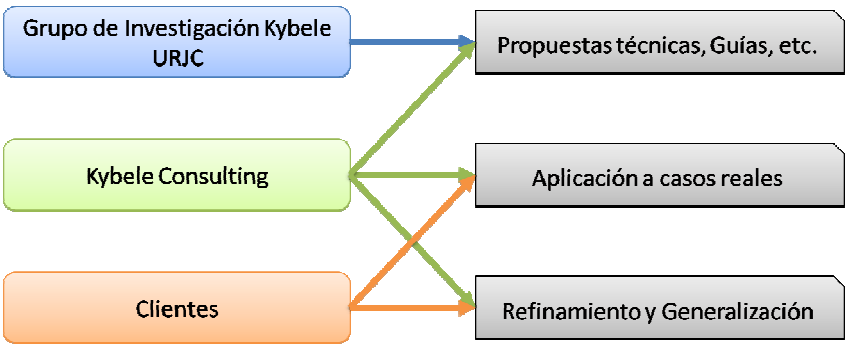


Figura 1-5. Método de investigación. Fases

Rol	Descripción	Rol identificado en nuestro trabajo
Investigador	El individuo o grupo que lleva a cabo de forma activa el proceso investigador	El doctorando y el grupo de investigación Kybele de la Universidad Rey Juan Carlos.
Objeto Investigado	El problema a resolver	Definir un entorno que facilite la evaluación de la mantenibilidad del producto software, calculando el valor y el retorno de inversión de las posibles refactorizaciones necesarias para mejorar dicha mantenibilidad
Grupo Crítico de Referencia (GCR)	Aquel para quien se investiga en el sentido de que tiene un problema que necesita ser resuelto y que también participa en el proceso de investigación.	El conjunto de los ingenieros de software, así como los gestores de proyectos de tecnologías de la información de las empresas donde se han realizado pruebas piloto.
Beneficiario	Aquel para quien se investiga en el sentido de que puede beneficiarse del resultado de la investigación, aunque no participa directamente en el proceso	Cualquier cliente potencial de un sistema de información, tanto empresas que subcontratan el desarrollo software, como los propios departamentos TIC o las fábricas de software.

Tabla 1-1. Roles identificados en la Investigación-Acción

1.6 Estructura de la Tesis

El resto de los capítulos de esta tesis, se organizan de la siguiente manera:

- El capítulo 2 ofrece una visión detallada del estado del arte de la Ingeniería del Software Basada en Valor y de las refactorizaciones basadas en valor. Se identifican carencias en las propuestas presentadas hasta el momento, y se proponen líneas de investigación abordadas en el resto del trabajo.
- En el capítulo 3 se describe el modelo de medición de la mantenibilidad implementado en KEMIS. El objetivo del proyecto KEMIS es ofrecer

indicadores de las características de calidad del producto que aporten visibilidad durante el desarrollo, la adquisición y el mantenimiento de un producto, proporcionando una infraestructura de medición automática, que se basa en métricas que se pueden obtener con herramientas de software libre, y un modelo de medición de calidad del producto software basado en el que propone la norma ISO/IEC 25000.

- En el capítulo 4 se describen los indicadores de valor y se construye un modelo de medición del valor. Se presenta una aproximación de la estructura del modelo de medición propuesto para el cálculo del valor de los componentes software desde el punto de vista de la mejora de su mantenibilidad y se discuten los resultados más significativos.
- En el capítulo 5 se discute la arquitectura de la herramienta KEMIS desde un punto de vista tecnológico.
- El capítulo 6 presenta la validación de este trabajo, mediante la aplicación o uso de KEMIS, tanto en casos de estudio como en proyectos reales.
- Finalmente, el capítulo 7 presenta las conclusiones del trabajo, analizando la consecución de objetivos, el contraste de resultados y las líneas de trabajo futuras.

Adicionalmente, se presentan una serie de anexos que han sido separados de la disertación principal con objeto de mejorar la comprensión de la exposición y cuyo contenido se detalla a continuación:

- Apéndice A: Requisitos de KEMIS. Este capítulo describe el conjunto de requisitos y casos de uso iniciales elaborados durante la especificación del entorno KEMIS.
- Apéndice B: Impacto de las refactorizaciones. En este apéndice se analiza el impacto que tienen las refactorizaciones sobre las métricas de calidad del producto descritas en el capítulo 3. El objetivo es identificar el efecto que una refactorización tendría sobre la mantenibilidad.
- Apéndice C: Entorno Visual de KEMIS. En este apéndice se describe en detalle la interfaz Web proporcionada por la herramienta KEMIS.
- Por último se incluyen dos apartados finales de acrónimos y referencia.

Estado del Arte

En este capítulo se describe el estado del arte de la Ingeniería de Software Basada en Valor, las refactorizaciones y los modelos de referencia de medición de calidad del producto software. Se realiza una breve reseña de las normas ISO/IEC 9126 e ISO/IEC 25010 de calidad del producto software, así como la descripción que hacen del concepto de mantenibilidad. Se describen los fundamentos de la Ingeniería de Software Basada en Valor, se detallan los resultados de una revisión sistemática de la literatura respecto de la refactorización basada en valor y se enumeran los indicadores de valor encontrados en otras disciplinas. Por último, se analizan un conjunto de herramientas que ayudan a la medición de la calidad del producto software.

2.1 La Calidad en el Desarrollo Software

De acuerdo con Parnas [41], en la Ingeniería de Software se trabaja mayormente con la "construcción de aplicaciones software multiversión". Por lo tanto, muchas de las actividades asociadas con una aplicación software provocan revisiones para mejorar la funcionalidad o para corregir errores. Esto mismo aseguraba Lehman en sus leyes [42], haciendo referencia al incremento de la complejidad y el crecimiento continuo. En la Tabla 2-1 se detalla una revisión de las últimas descripciones de las leyes de Lehman sobre la evolución del software. Estas leyes hacen referencia a los sistemas del tipo E, que se describen como los sistemas software construidos para solucionar un problema o implementar una aplicación computacional en el mundo real.

Por lo tanto, se puede concluir que las aplicaciones software tienden a crecer, tanto en complejidad como en funcionalidad y que es necesario realizar un mantenimiento periódico para, por lo menos, conservar su calidad actual (séptima ley de Lehman: “disminución de la calidad” [42]).

En este sentido, la calidad es un concepto complejo y multidimensional [6], es decir, puede ser descrita desde diferentes perspectivas. Garvin [43] ha hecho un intento de ordenar las diferentes perspectivas de calidad.

- Perspectiva trascendental: donde la calidad es reconocida, pero no descrita. Se realizan definiciones subjetivas y no cuantificables.
- Perspectiva del usuario: la calidad está directamente relacionada con la satisfacción de las necesidades del usuario. Características como la fiabilidad, el rendimiento o la eficiencia son las tenidas en cuenta en esta perspectiva.

- Perspectiva de la fabricación o del proceso: se centra en la conformidad con las especificaciones y la capacidad para producir software de acuerdo con el proceso de desarrollo implementado en la empresa. En este caso se tienen en cuenta características como la tasa de defectos o los costes de re-trabajo.
- Perspectiva de producto: especifica que las características de calidad del producto se definen a partir de las características de sus partes. Por ejemplo líneas de código, complejidad, diseño.
- Perspectiva basada en aspectos económicos: miden la calidad de acuerdo a costes, precios, productividad, etc.

En el caso del desarrollo software, la calidad puede estudiarse desde el punto de vista de la calidad del producto software y la calidad del proceso de desarrollo software [44].

Año	Nombre y explicación
1974	Cambio continuo. Un software de tipo E que sea utilizado deberá ser adaptado continuamente, sino este se volverá progresivamente menos satisfactorio.
1974	Incremento de la complejidad. La evolución de un programa conlleva un incremento en su complejidad si no se realiza un trabajo para mantener o reducir este nivel de complejidad.
1974	Autorregulación. El proceso de evolución del software se encuentra autorregulado con una distribución semejante a la distribución normal que presentan las mediciones de atributos de procesos y producto.
1978	Conservación de la inestabilidad organizacional (ratio de trabajo invariante). El promedio del ratio de trabajo global efectivo en un sistema que evoluciona es invariante respecto a su tiempo de vida.
1991	Conservación de la familiaridad. Durante el tiempo de vida activo de un sistema que evoluciona, el contenido de releases sucesivas es estadísticamente invariante.
1991	Crecimiento continuo. El contenido funcional de un programa debe ser continuamente incrementado para mantener la satisfacción del usuario.
1996	Disminución de la calidad. La ausencia de un mantenimiento riguroso y una respuesta adaptativa a los cambios en el entorno operacional se percibirá como una disminución de la calidad del software de tipo E.
1971 1996	Sistema de realimentación. Los software de tipo E constituyen sistemas de realimentación multi-bucle y multi-nivel y deben ser tratados de esa manera para asegurar modificaciones y mejoras exitosas.

Tabla 2-1. Leyes de Lehman, formulación del año 1996 [42]

2.1.1 Modelos para la medición de la calidad

Bosch define a los atributos de calidad orientados al desarrollo como aquellos de particular relevancia desde la perspectiva de la Ingeniería del Software [45] y pone como ejemplos a la mantenibilidad y la reutilización entre otros. A continuación se comentarán los modelos de calidad software más comunes [46] que hagan referencia a atributos de calidad orientados al desarrollo. En [46] se hace referencia a los modelos de calidad de McCall, Boehm e ISO/IEC 9126 como los más importantes. A su vez, la norma ISO/IEC 25010, publicada en marzo de 2011 y que reemplaza a la norma ISO/IEC 9126, puede ser tenida en cuenta en este grupo.

Los modelos antes citados difieren en varios aspectos. El modelo de McCall fue creado en 1977 y consiste en un árbol jerárquico donde los factores externos de calidad se relacionan con los criterios de calidad del producto. Los tres factores externos tenían que ver con la operación, revisión y transición de los productos software.

El modelo de Boehm fue publicado en 1978 y difiere del modelo de McCall ya que agrega características de rendimiento del hardware. Así mismo, el modelo de Boehm categoriza atributos de acuerdo con una perspectiva de utilidad. La norma ISO/IEC 9126 fue publicada por primera vez en 1991, con la intención de unificar los criterios de los modelos anteriores. Este modelo, al igual que los de McCall y Boehm es jerárquico, conectando los atributos de calidad a las características. En este caso, la relación de herencia es estricta, separando las características y las subcaracterísticas (por lo que una subcaracterística no se relaciona con más de una característica).

Finalmente, dos son los modelos más destacados en el área de la calidad del producto software. Estos se corresponden con la norma ISO/IEC 9126 y su sucesora, la norma ISO/IEC 25010. A continuación se detallan ambas normas.

2.1.2 La norma ISO/IEC 9126

El norma ISO/IEC 9126, ahora englobado en el proyecto SQuaRE para el desarrollo de la norma ISO/IEC 25000, establece un modelo de calidad en el que se recogen las investigaciones de multitud de modelos de calidad propuestos por los investigadores durante los últimos 30 años para la caracterización de la calidad del producto software.

Este estándar propone un modelo de calidad que se divide en tres aproximaciones: interna (calidad del código), externa (calidad en la ejecución) y

en uso. Estas tres aproximaciones se influyen entre sí (tal y como se puede ver en la Figura 2-1). Así, la calidad interna influye a la externa y esta a la calidad en uso.

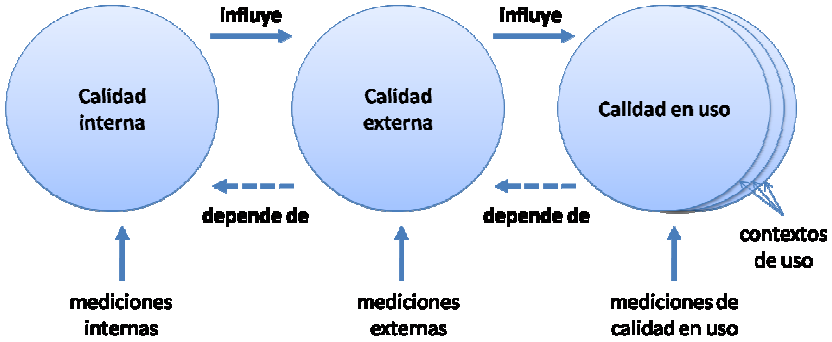


Figura 2-1. Las tres perspectivas de calidad de ISO/IEC 9126

La evaluación de la calidad de un producto software requiere de un modelo de calidad definido. Existen muchos ejemplos de modelos de calidad [47][48][49][50][51]. Un modelo de calidad está compuesto de características, que se dividen en subcaracterísticas. Finalmente, las subcaracterísticas se componen de atributos, que obtienen sus valores de mediciones en sobre el software.

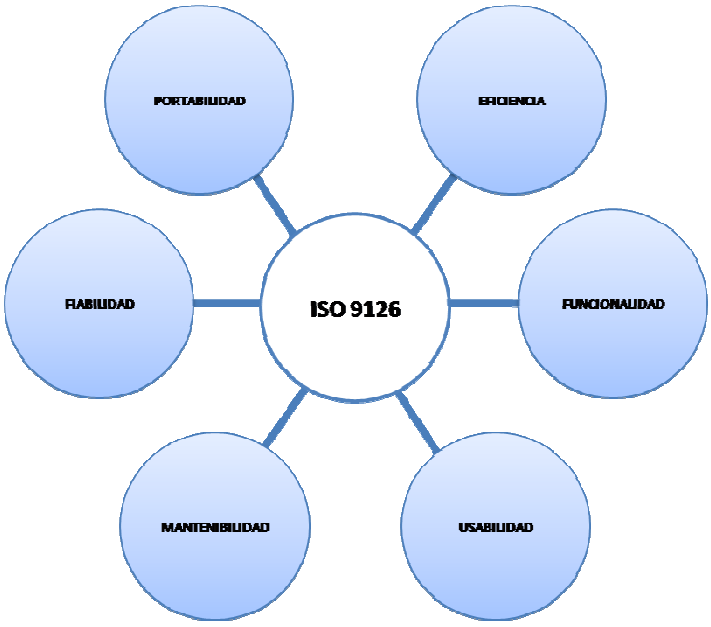


Figura 2-2. Características de calidad según la norma ISO/IEC 9126

En este caso, el modelo establece diez características, seis de ellas comunes a las vistas interna y externa y las otras cuatro propias de la vista en uso.

Las características que definen las vistas interna y externa se muestran en la Figura 2-2 y son:

- Funcionalidad, capacidad del software de proveer los servicios necesarios para cumplir con los requisitos funcionales.
- Fiabilidad, capacidad del software de mantener las prestaciones requeridas del sistema, durante un tiempo establecido y bajo un conjunto de condiciones definidas.
- Eficiencia, relación entre las prestaciones del software y los requisitos necesarios para su utilización.
- Usabilidad, esfuerzo requerido por el usuario para utilizar el producto satisfactoriamente.
- Mantenibilidad, esfuerzo necesario para adaptarse a las nuevas especificaciones y requisitos del software.
- Portabilidad, capacidad del software para ser transferido de un entorno a otro.

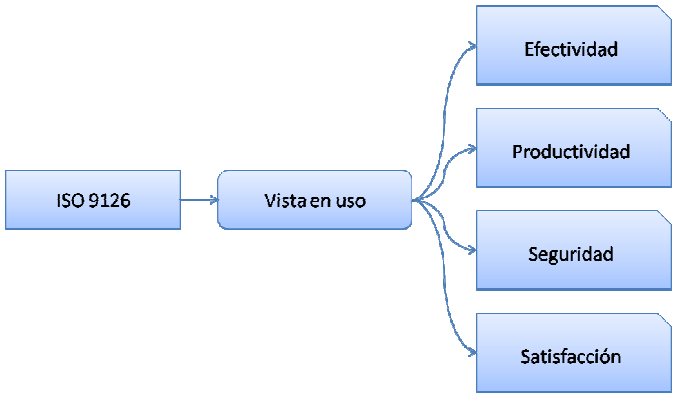


Figura 2-3. Características de la vista en uso según la norma ISO/IEC 9126

Por otro lado, las cuatro características propias de la vista en uso se muestran en la Figura 2-3 y son las siguientes:

- Efectividad, capacidad del software de facilitar al usuario alcanzar objetivos con precisión y completitud.

- Productividad, capacidad del software de permitir a los usuarios gastar la cantidad apropiada de recursos en relación a la efectividad obtenida.
- Seguridad, capacidad del software para cumplir con los niveles de riesgo permitidos tanto para posibles daños físicos como para posibles riesgos de datos.
- Satisfacción, capacidad del software de cumplir con las expectativas de los usuarios en un contexto determinado.

Como se ha comentado anteriormente, la mantenibilidad es una de las características más importantes de la calidad de un producto software. Este atributo está intrínsecamente asociado con el proceso de mantenimiento, el cuál ha sido ampliamente reconocido por representar una gran parte de los costes en el Ciclo de Vida Software (CVS) [51][52][53]. Por lo tanto, la mantenibilidad de un producto software impacta significativamente en los costes. Y por consiguiente, es importante predecir la mantenibilidad del producto software para gestionar con eficacia los costes.

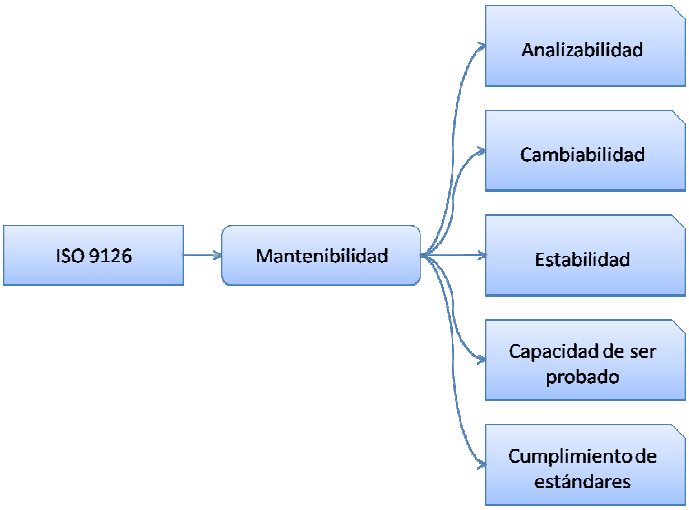


Figura 2-4. La mantenibilidad según la norma ISO/IEC 9126

La mantenibilidad, según el modelo de calidad recogido por la norma ISO/IEC 9126 está formada por las siguientes subcaracterísticas (ver Figura 2-4):

- Analizabilidad, facilidad para analizar el software en busca de deficiencias e identificar sus componentes y artefactos.
- Cambiabilidad, capacidad de permitir modificaciones en el producto software.

- Estabilidad, capacidad de evitar efectos inesperados tras la realización de modificaciones en el software.
- Capacidad de ser probado, capacidad para validar los cambios en el software.
- Adherencia a las normas, cumplimiento de los estándares y convenciones de mantenibilidad. Hace referencia a todas las anteriores.

2.1.3 La norma ISO/IEC 25010

Recientemente se ha publicado una nueva versión de la norma ISO/IEC 9126, la norma ISO/IEC 25010. Esta nueva norma se encuentra englobada en la serie de normas ISO/IEC 25000 SQuaRE (Software Product Quality Requeriments and Evaluation).

El objetivo de esta serie es guiar el desarrollo de productos software siguiendo una especificación y evaluación de requisitos de calidad. Así mismo, establece criterios para la especificación de requisitos de calidad de productos software, sus métricas y su evaluación. Se definen las siguientes divisiones de normas dentro de la serie 25000:

- ISO/IEC 2500n. Las normas que forman esta división definen todos los modelos comunes, términos y referencias a los que se alude en las demás divisiones de SQuaRE. También proporcionan una guía para crear una función de soporte responsable de la gestión de la especificación y de la evaluación del producto software.
- ISO/IEC 2501n. División del modelo de calidad. Las normas que conforman esta división presentan un modelo de calidad detallado, incluyendo características para la calidad interna, externa y en uso.
- ISO/IEC 2502n. División de mediciones de calidad. Las normas pertenecientes a esta división incluyen un modelo de referencia de calidad del producto software, definiciones matemáticas para las mediciones de calidad y una guía práctica para su aplicación. Presenta aplicaciones de métricas para la calidad externa, interna y en uso del producto software.
- ISO/IEC 2503n. División de requisitos de calidad. Las normas que forman esta división ayudan a especificar los requisitos de calidad. Estos requisitos pueden ser usados en el proceso de especificación de requisitos de calidad para un producto software que va a ser desarrollado o como entrada para un proceso de evaluación. El proceso de definición de requisitos se guía por el establecido en la norma ISO/IEC 15288.

- ISO/IEC 2404n. División de evaluación de la calidad. Estas normas proporcionan requisitos, recomendaciones y guías para la evaluación de un producto software, tanto si la llevan a cabo evaluadores, clientes o desarrolladores.
- ISO/IEC 25050-25099. Normas de extensión SQuaRE. Incluyen requisitos para la calidad de productos software "Off-The-Self" y para el formato común de la industria para informes de usabilidad.

La serie 25000 no establece los niveles de calidad deseables para cada proyecto, pero si recomienda que los requisitos de calidad deban ser proporcionales a las necesidades de la aplicación y lo crítico que sea el correcto funcionamiento del sistema implementado. Por lo tanto, será trabajo del jefe de proyecto determinar correctamente el nivel de calidad final que un producto software deberá alcanzar.

El principal objetivo de la norma ISO/IEC 25010 es proporcionar un marco para definir y evaluar la calidad del software. Para lograr esto, la norma define dos modelos. El primero es el modelo de la calidad del producto que proporciona un conjunto de características de calidad relacionadas con las propiedades estáticas y las propiedades dinámicas del software. Para ello, identifica ocho características que componen los modelos para evaluar la calidad del producto a diferencia de la norma ISO/IEC 9126 que definía seis): idoneidad funcional, eficiencia en el rendimiento, compatibilidad, usabilidad, confiabilidad, seguridad, mantenibilidad y portabilidad. La característica de mantenibilidad se mantiene. De hecho, desde el modelo de McCall es una de las pocas características que se van repitiendo a lo largo de todos los modelos.

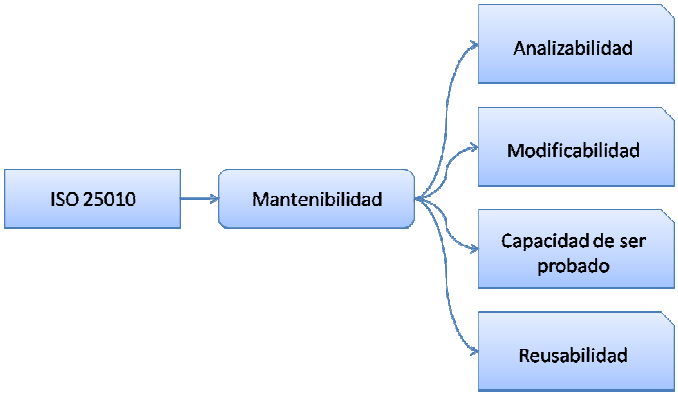


Figura 2-5. La mantenibilidad según la norma ISO/IEC 25010

Para la norma 25010, las subcaracterísticas que componen la mantenibilidad son:

- Analizabilidad: facilidad para analizar el software en busca de deficiencias e identificar sus componentes y artefactos.
- Modificabilidad: grado en el cual un producto software puede ser modificado con eficiencia y eficacia sin introducir defectos o degradar su calidad.
- Capacidad de ser probado: los atributos de software que tienen que ver el esfuerzo necesario para validar el software modificado.
- Reutilización: grado en el que un producto software puede ser utilizado en más de un sistema o como parte de otro producto software.

Por último, se enumeran los cambios en la característica de mantenibilidad entre la norma ISO/IEC 25010 e ISO/IEC 9126 (ver Figura 2-6):

- Hay dos subcaracterísticas nueva: la reutilización y la modificabilidad.
- La subcaracterística de modificabilidad combina dos subcaracterísticas de la norma ISO/IEC 9126: cambiabilidad y estabilidad.
- El cumplimiento de estándares, que es una subcaracterística en ISO/IEC 9126, está ahora fuera del alcance del modelo de calidad en ISO/IEC 25010.

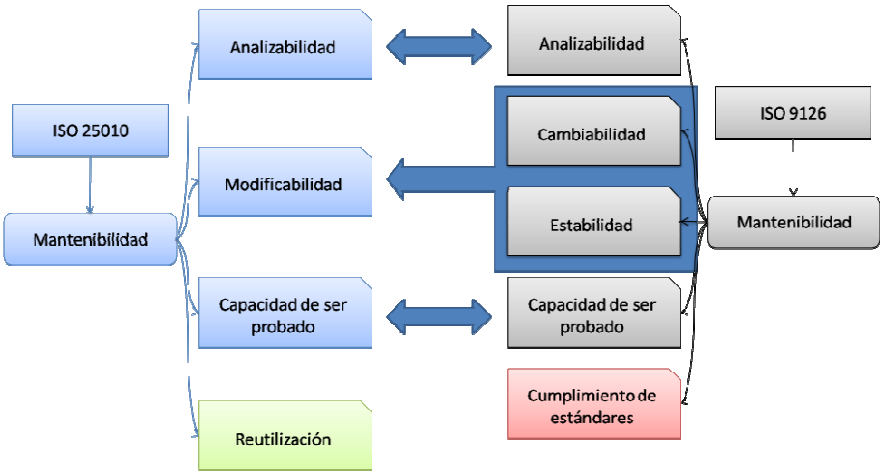


Figura 2-6. Evolución de la mantenibilidad en las normas ISO/IEC 25010 e ISO/IEC 9126

2.2 Los Modelos de Medición de la Mantenibilidad

En esta sección se analizarán los principales modelos para medir la mantenibilidad de un producto software, evaluando su amplitud y capacidad de ser llevado a la práctica mediante herramientas automáticas.

Para evaluar la calidad de un producto software son necesarios dos pasos. El primero, la definición de un modelo de calidad abstracto que relacione diferentes características de calidad. En ese sentido, una posible elección son las normas ISO/IEC 9126 e ISO/IEC 25010, las cuales descomponen el concepto abstracto de la mantenibilidad en atributos de calidad más manejables. Aún así, estos atributos son todavía muy abstractos como para poder ser medidos directamente en el producto software.

El segundo paso es la selección de herramientas de análisis de código, con las cuales obtener la información necesaria del código fuente, como por ejemplo métricas de acoplamiento, cohesión, complejidad o violaciones de buenas prácticas al codificar. Uno de los problemas de este tipo de herramientas es su grado de relación con los atributos abstractos definidos por los modelos abstractos de calidad.

Una métrica software [54][55] es un mapeo numérico de una parte del software que cuantifica uno o más atributos software [56]. Estas medidas son comúnmente considerados como factores importantes que reflejan la naturaleza del producto software, incluyendo, entre otras características, las propiedades de complejidad y mantenibilidad [57][58][59]. Las métricas de software tradicionalmente incluyen la cantidad de líneas de código fuente, la media de líneas de código por método o clase, el porcentaje de comentarios, la cantidad de decisiones (por ejemplo sentencias *if*), la cantidad de operadores o parámetros, etc.

Para obtener las métricas que nos permitirán calcular la calidad de los productos software, lo más común y eficiente es utilizar herramientas que nos permitan realizar estas mediciones y nos ofrezcan informes con los resultados. Estas herramientas pueden clasificarse en dos tipos:

- Herramientas de análisis dinámico: aquellas herramientas que realizan el análisis del software ejecutando el código fuente de dicho software. Un ejemplo claro de una herramienta de este tipo será la que nos permita conocer el número de pruebas que han fallado durante la ejecución de las mismas.
- Herramientas de análisis estático: aquellas herramientas que llevan a cabo el análisis sin necesidad de ejecutar el software bajo estudio. Estas herramientas

realizan inspecciones sobre el código fuente y ofrecen datos del mismo, tales como el número de líneas de código, complejidad básica.

El desafío actual es la unión de ambos elementos, sumando los resultados heterogéneos de herramientas de análisis del código fuente junto con un modelo de evaluación de la calidad del producto software.

En ese sentido, uno de los estudios más interesantes es el realizado por Riaz en el año 2009 [92]. En el estudio se lleva a cabo una revisión sistemática de las métricas de mantenibilidad y su capacidad de predicción. Para esta Tesis Doctoral se han revisado los artículos indicados en ese trabajo y se ha realizado nuevamente la búsqueda indicada por Riaz. A continuación se comentan los artículos que detallan un modelo de medición de la mantenibilidad y a la vez profundizan en las métricas básicas necesarias para construir el modelo.

Rüdiger Lincke indica en [60] un modelo para medir la mantenibilidad, basado en la norma ISO/IEC 1926 y algunas métricas asociadas. El objetivo principal del trabajo es demostrar que algunas métricas básicas teóricas tienen implementaciones diferentes dependiendo la herramienta. Por lo tanto, no profundiza en el modelo de medición de la mantenibilidad y su relación con las métricas básicas.

Por su parte, Heitlager [61] describe un modelo de medición de la mantenibilidad en el que empareja las subcaracterísticas de la mantenibilidad de acuerdo con la norma 9126, con propiedades del código fuente: volumen, complejidad, duplicación, tamaño de la unidad de código y pruebas. No identifica las herramientas para realizar las mediciones.

Mouchawrab describe en [62] un framework genérico para medir la capacidad con la que cuenta un sistema para ser probado. Presenta una gran cantidad de métricas, algunas de ellas asociadas a métricas del código fuente, aunque no detalla un modelo con sumarizar estas métricas.

Ioannis Samoladas [63] presenta un modelo de calidad jerárquico que evalúa el código fuente desde el punto de vista de la mantenibilidad, la seguridad y la confiabilidad. En este caso, la definición de métricas se basa en otros estudios pero sin especificar el mecanismo de la elección.

En [64], Don Coleman define un modelo de medición de la mantenibilidad polinómico, detallando una fórmula que asocia cuatro métricas: líneas de código, número de módulos, volumen y esfuerzo. En el artículo no se define con precisión el mecanismo de elección de las métricas, aún así comprueba su validez mediante casos de estudio. Así mismo, en el trabajo de Yuming Zhou [65] se indican un grupo de métricas de mantenibilidad asociadas con el código fuente.

En [66] Klaus Lochmann define una herramienta, denominada ConQAT para construir cuadros de mando de calidad y relaciona herramientas de análisis estático de código con diferentes características de calidad. Por el contrario, no define un modelo de medición específico ni realiza una selección justificada de las herramientas con las cuales obtener las métricas básicas.

Se puede concluir que es posible medir el nivel de calidad de un producto si se conoce el valor de sus atributos [69]. Sin embargo, aunque las normas ISO/IEC 9126 e ISO/IEC 25010 definen las características y subcaracterísticas del modelo de calidad, no define los atributos necesarios para calcularlas. Como se puede ver en la Tabla 2-2, los modelos se centran principalmente en la norma ISO/IEC 9126 y ningún modelo detalla a la vez un entorno metodológico (relacionado con un modelo de calidad abstracto) y un entorno tecnológico (integrando herramientas para obtener las métricas básicas).

Ref.	Herramientas	Métricas	Característica de calidad	Modelo
[60]	Si, al azar	Si, al azar	Subcaracterísticas de mantenibilidad ISO/IEC 9126	No
[61]	No	Si, justificadas	Subcaracterísticas de mantenibilidad ISO/IEC 9126	Si
[62]	No	Si, justificadas	Capacidad de de ser probado	No
[63]	No	Si, parcialmente justificadas	Subcaracterísticas de mantenibilidad ISO/IEC 9126	Si
[64]	No	Si, parcialmente justificadas	Obtiene un único valor asociado directamente a la mantenibilidad	No
[65]	No	Si, parcialmente justificadas	Obtiene un único valor asociado directamente a la mantenibilidad	No
61	Si	No	Flexible	No

Tabla 2-2. Modelos de medición de la mantenibilidad

En la herramienta KEMIS se ha trabajado en la definición de estos atributos, describiéndose cálculos y ponderaciones sobre métricas de calidad del

código fuente. Esto permite otorgar un valor de calidad a la característica de la mantenibilidad del producto software.

2.3 El Concepto de Valor

La Ingeniería del Software convencional estudia la relación entre lo que debe hacer un sistema y lo que realmente hace. Dicho de otra manera, estudia cómo transformar los requisitos de un sistema en código verificado a través de distintas técnicas y procesos, la mayoría de ellos con un uso extendido en la industria.

En la actualidad, la mayoría de las prácticas de la Ingeniería del Software utilizan un enfoque de valor neutro, es decir, cada requisito, caso de uso, objeto o prueba se trata con igual importancia. Sin embargo, no todos los elementos aportan el mismo valor al negocio: "Tiempo atrás, cuando las decisiones que afectaban al software tenían relativamente poca influencia en costes, planificación y valor de las compañías, una aproximación neutral en valor podía ser razonable, pero en la actualidad estas decisiones tienen una gran repercusión" [16].

Para cubrir las carencias del enfoque tradicional, está emergiendo una nueva rama en oposición a la Ingeniería del Software convencional. Se trata de la Ingeniería del Software Basada en Valor (ISBV), o Value-Based Software Engineering (VBSE) en su terminología inglesa. La ISBV establece que no todas las funcionalidades son igualmente importantes, ya que algunas de ellas aportan más "valor" que otras. Por lo tanto, es necesario proveer de mecanismos que gestionen esta diferenciación de los distintos artefactos software. El principal precursor de una definición común de las técnicas basadas en valor es Barry Boehm. Sus aportaciones más importantes en este sentido son presentadas en esta sección.

El término "valor" en la Ingeniería del Software es bastante confuso. Chikofsky aseguraba en la presentación inaugural de la primera conferencia de Ingeniería del Software Basada en Valor que "el primer objetivo que tenía por delante este campo de la investigación era que todos los allí presentes tuvieran una idea común de definición de valor" [70]. Tras la revisión efectuada en este capítulo, coincidimos plenamente con dicha afirmación. Bajo el término ISBV se han encontrado trabajos referidos al valor con sentidos completamente opuestos. Existen trabajos puramente económicos que cuantifican valores de inversión de cartera de proyectos (que nada tienen que ver con prácticas de Ingeniería del Software), técnicas de cuantificación de esfuerzos, e incluso propuestas de

técnicas supuestamente basadas en valor que ignoraban por completo a los implicados. Sin embargo, muchos de ellos se clasifican a sí mismos como trabajos de ISBV.

La ISBV trata de completar la visión que proporciona la Ingeniería del Software tradicional. Para ello sugiere la priorización de las tareas y artefactos que proporcionen valor a los distintos implicados, a través de distintas técnicas y métodos. En resumen, pretende orientar los recursos hacia donde más puedan aportar a los implicados. En palabras de Rick Kazman, "la ISBV trata de predecir dónde deben ser invertidos el tiempo y los recursos".

Por otro lado, también se han encontrado técnicas de priorización de esfuerzos claramente orientados a maximizar el valor de cara a los usuarios, que no nombran en ningún momento la ISBV ni el concepto de valor, aunque emparejen prácticas de Ingeniería del Software con la percepción de los implicados.

Como se ha indicado anteriormente, no existe una definición de valor ampliamente reconocida. Para esta Tesis Doctoral se mantiene la definición indicada en [17]: **el "valor" se define como la cuantificación de la importancia que un determinado artefacto o tarea tiene para todos los implicados en ese sistema**.

2.3.1 Áreas de la ISBV

La ISBV define las siguientes áreas:

- Ingeniería de requisitos basada en valor, para la identificación de principios y prácticas que identifican el valor de los requisitos.
- Arquitectura basada en valor, para reconciliar los objetivos de negocio con la arquitectura establecida en los sistemas para apoyar la construcción de software.
- Diseño y desarrollo basados en valor, para alinear el diseño y construcción del software con los objetivos de los distintos implicados.
- Validación y verificación basadas en valor, para desarrollar técnicas de verificación y validación del software que prioricen esfuerzos respecto a objetivos de valor.
- Control y planificación basados en valor, para evolucionar las técnicas más tradicionales sobre planificación, coste y control y que incluyan el concepto del valor que aportan.

- Gestión de riesgos basada en valor, para priorizar, mitigar, identificar y analizar riesgos desde la perspectiva del impacto que pueden tener en el valor.
- Gestión de la calidad basada en valor, para la priorización de los factores de calidad deseables con respecto al valor que aportan.
- Gestión de personal basada en valor, para la gestión del equipo y recursos humanos.

2.3.2 Definición de fundamentos de la Ingeniería del Software Basada en Valor

La primera iniciativa de agrupación y ordenación de todos los métodos basados en valor, que ayudasen a alinear los objetivos de negocio con la Ingeniería del Software, fue llevada a cabo por Barry Boehm. En sus trabajos iniciales [71][16] se propone una agrupación y clasificación de todas las prácticas centradas en valor en distintas áreas. Desde luego, no es el primer trabajo que maneja el concepto de "valor" y "priorización" para llevar a cabo alguna tarea de ingeniería, pero sí es el primero que identifica una línea de investigación global, y que propone dividir todos estos métodos en distintas áreas. Adicionalmente, Boehm propone definir los "fundamentos" de la ISBV en torno a siete elementos críticos, que define como el "punto de partida" para avanzar en la "Agenda de la ISBV", que ayuden a las organizaciones a implantar esta nueva aproximación:

- **Benefits Realization Approach (BRA)**: Se trata de una técnica introducida por Thorp en DMR [72] que tiene por objetivo coordinar las actividades del departamento de IT con los objetivos del resto de la organización, en lo que llama la "Cadena de Resultados". Se trata de elaborar una cadena que conecte gráficamente las iniciativas de trabajos de IT y sus contribuciones de cara a la organización.
- **Reconciliar objetivos de distintos implicados** mediante técnicas de grupo y de gestión de expectativas. En este sentido, Boehm presenta la "Teoría Win-Win", para ayudar a las empresas a asegurar el éxito de sus inversiones. Según esta propuesta, si los implicados ganan con el software propuesto, se puede afirmar el éxito del sistema.
- **Análisis de Casos de Negocio**. Propone utilizar un modelo de retorno de inversión (más conocido por las siglas ROI, del término inglés *Return On Investment*) para cuantificar los costes y beneficios de las distintas iniciativas. Este será uno de los fundamentos del modelo de medición del valor, definido en el capítulo 0.

$$ROI = \frac{Beneficio - Coste}{Coste}$$

Figura 2-7. Fórmula genérica del ROI

- **Gestión del riesgo y de oportunidades de negocio**. Desde el punto de vista de los sistemas software, cuanto antes se identifiquen los riesgos, mayor será el valor aportado por los sistemas. Boehm propone hacerlo usando técnicas como el prototipado, encuestas, etc. Para algunos casos concretos propone incluso identificar a una persona para monitorizar el nicho de mercado tecnológico en el que se trabaja.
- **Desarrollo concurrente** (en lugar de secuencia) de los distintos elementos y proyectos, monitorización del valor aportado usando la técnica BRA, y uso de metodologías ágiles para algunos proyectos para tener resultados antes y mejorar la competitividad.
- **Monitorización y control basado en valor.** Boehm indica que es muy difícil conocer el valor ganado por un proyecto en caso de cambios rápidos en la planificación. En lugar de ello, indica que para realizar una correcta monitorización y control basados en valor es necesario seguir dos pasos. Como primer paso se debe utilizar el análisis de casos de negocio, discutido anteriormente. Como siguiente paso, es necesario monitorizarlo con la aproximación indicada por Thorp y referenciado como el primer elemento clave de esta lista.
- **El cambio como oportunidad.** Muchas veces las tareas de adaptación al cambio son consideradas como costes a ser minimizados. Esto hace que los proyectos y organizaciones estén en contra de los cambios. Actualmente los cambios son cada vez más frecuentes, tanto en las tecnologías como en las necesidades de los implicados. Y por lo tanto, la organización que mejor se adapte a los cambios tiene una ventaja competitiva frente a la que no lo hace. En lo referente al software, desarrollar un diseño modular con posibilidad de anticipación al cambio puede ser considerado un buen ejemplo de inversión para disminuir los costes de cambios futuros.

Como puede observarse, dichos fundamentos se centran en su mayor parte en la gestión de proyectos, y en su valoración económica. Se trata de una propuesta de alto nivel que analiza los proyectos de IT dentro de las organizaciones. Es un primer paso para sentar las bases de la aplicación del valor a

la Ingeniería del Software, pero no entra en detalle más allá de hacer una revisión de trabajos relacionados.

Boehm presenta además en un caso de de estudio [73] la aplicación de la técnica BRA de Thorp, la estimación coste-beneficio, y la monitorización y control de los beneficios sobre un ejemplo ficticio de encargos on-line.

2.3.3 Evaluación de coste de mantenimiento

Tratándose de sistemas software, muy pocas líneas de código pueden aportar increíbles beneficios, del mismo modo que millones de líneas de código pueden aportar muy poco. La estimación del valor del software es una tarea muy compleja para los economistas, ya que debe de tener en cuenta no sólo que se trata de un intangible, más difícil de cuantificar que un tangible como un coche o una vivienda, sino también que su modelo de mantenimiento (que debe ser considerado para estimar el coste de una opción) es especialmente complejo, ya que el software tiende a crecer y a deteriorarse con el tiempo de un modo particular (ver sección 2.1).

Por un lado, es necesario cuantificar el valor que el software aportará, y por otro el coste que tendrá construirlo y mantenerlo. Para tener en cuenta el esfuerzo en mantenimiento que será requerido a lo largo del tiempo, se han desarrollado modelos de crecimiento del software [25], que analizan modelos de mantenimiento correctivo, adaptativo y perfectivo. Utiliza datos recogidos de diversos estudios para estimar qué porcentaje del código será necesario modificar y qué porcentaje de código nuevo aparecerá en cada versión sucesiva.

Wiederhold señala que los modelos de crecimiento son un factor indispensable a tener en cuenta en el cálculo de valor de los proyectos, ya que el mantenimiento consume la mayor parte de los recursos del ciclo de vida de los proyectos de éxito.

Otro aspecto que conviene resaltar es la escasa cantidad de propuestas de actividades de mantenimiento basadas en valor. Más aún, ni siquiera hay un área de conocimiento planteada (no existe el concepto de “Mantenimiento Basado en Valor”), aún teniendo el mantenimiento una gran influencia sobre la evaluación del coste de los sistemas [25], y que es una de las áreas de conocimiento de Ingeniería del Software identificadas por el proyecto SWEBOK. Uno de los únicos trabajos que hablan del tema es la Tesis Doctoral de Daniel Cabrero [17], quien propone la aplicación práctica del mantenimiento basado en valor como una nueva área de la ISBV.

2.3.4 *Relación entre las distintas áreas de la ISBV*

En general, la mayoría de las técnicas presentadas comienzan por extraer el valor relativo de cada requisito a desarrollar. Posteriormente se utiliza dicha priorización para trasladar la importancia relativa de los requisitos a técnicas y artefactos de otras áreas.

En la Figura 2-8 se representan las distintas áreas de conocimiento basado en valor y la relación existente entre dichas áreas. Las flechas entre las distintas áreas representan el flujo de valor. Por ejemplo, una flecha desde los requisitos a pruebas simboliza la posibilidad de aprovechar el valor relativo de los requisitos para priorizar las pruebas relacionadas con ellos.

Las flechas negras representan los flujos de valor que se han implementado en los trabajos identificados durante la revisión. La información del valor de los requisitos se utiliza en trabajos de calidad basada en valor [74][75][76][77][78][79][80], validación y verificación [81], arquitectura [82][83][84], y gestión de proyectos [85]. Como puede observarse en la Figura 2-8, se han realizado trabajos de aplicación del valor de los requisitos a la priorización de cartera de proyectos, arquitectura, calidad, y validación y verificación.

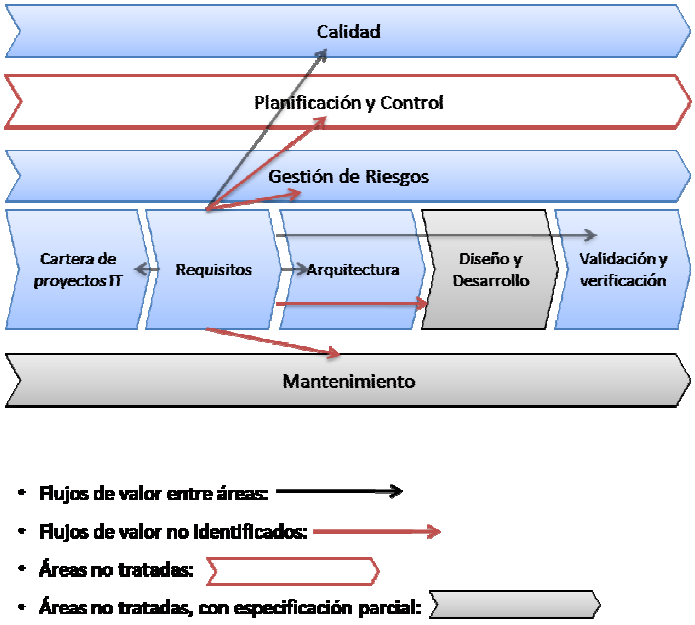


Figura 2-8. Relación entre las distintas áreas de ISBV

Las flechas rojas representan los flujos de valor que podrían investigarse, pero sobre los que no se ha encontrado ninguna propuesta hasta la fecha. Dicho de otro modo, representan las oportunidades de investigación detectadas a lo largo de en esta revisión. Se han detectado posibilidades de mejora en la relación entre los requisitos y la gestión de riesgos, diseño y desarrollo, y planificación (además del área de mantenimiento que no estaba definida).

Respetando la terminología definida por Boehm, el área de diseño y desarrollo (área representada gris) comprende el análisis, diseño de microarquitectura, y programación de los sistemas. Hasta la fecha no existe ningún trabajo sobre dicha área, por ese motivo se representa en rojo. Esta área, junto con el área de mantenimiento basado en valor ha sido abordada hasta el momento solamente por la Tesis Doctoral de Daniel Cabrero [17].

2.3.5 Indicadores de valor

Como se ha comentado, para la aplicación de una técnica basada en valor se debe asignar una prioridad a los artefactos. Dicha prioridad suele estar calculada en función de uno o varios indicadores, bien a través de una fórmula, bien a través de valores tabulados mediante tablas.

A continuación se aporta una definición de "Indicador de Valor" encontrada en [17].

- **Indicadores de valor:** estimaciones o evaluaciones de atributos que capturan algún aspecto que puede tener influencia en la percepción de los implicados de un proyecto software.
- **Indicadores de valor subjetivos:** aquellos indicadores que han sido estimados en función de las opiniones de los implicados.
- **Indicadores de valor objetivos:** aquellos indicadores que han extraídos mediante técnicas automatizadas o estadísticas del proyecto.

Otro aspecto común a todas las propuestas centradas en valor es la decisión sobre el grado de ejecución de una tarea en función del valor de un artefacto concreto. El grado de definición de las tareas varía entre propuestas, al igual que sucedía en el caso de los indicadores de valor.

Un ejemplo de tarea priorizable que típicamente se realiza en los desarrollos software es la ejecución de las pruebas de regresión. Las pruebas de regresión abarcan únicamente un subconjunto de la funcionalidad, y son realizadas antes de liberar una versión que ha sufrido cambios para asegurar un mínimo de funcionamiento. La no ejecución de este tipo de prueba para la mayoría de los

requisitos implica una priorización en función de algún criterio. En el marco de la Tesis Doctoral de Daniel Cabrero [17] se propone considerar priorizable la aplicación de cualquier tarea, buena práctica, o incluso la construcción misma de un subsistema. La definición de "tarea priorizable" realizada en [17] es la siguiente: **Una "tarea priorizable" es aquella que puede ser ejecutada de diferentes modos (o no ser ejecutada) en función del valor relativo de los artefactos involucrados en su ejecución. Una tarea priorizable es, en definitiva una tarea "prescindible" en algún grado bajo determinadas condiciones.**

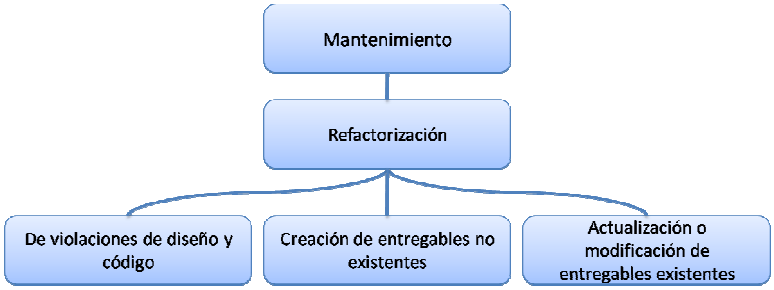


Figura 2-9. Árbol de tareas priorizable de codificación

Las tareas realizables en fase de mantenimiento es otro conjunto de actividades que no han sido contempladas anteriormente desde la óptica de la ISBV. La Figura 2-9 presenta la propuesta de tareas priorizables de [17]. En dicho trabajo se señala la importancia de definir tareas priorizables para la mayoría de las actividades a realizar en la gestión, desarrollo o mantenimiento de software.

2.4 La Refactorización

Una de las tareas principales, identificadas en fase de mantenimiento es la refactorización. El término refactorización se atribuye a Opdyke, que lo introdujo por primera vez en 1992 [19]. Una refactorización consiste en una transformación parametrizada de un programa preservando su comportamiento, que automáticamente modifica el diseño de la aplicación y el código fuente subyacente [20][21]. Para Martin Fowler [22] refactorizar se define como un cambio realizado a la estructura interna del software para hacerlo más fácil de entender y más barato de modificar sin cambiar su comportamiento visible". O lo que es lo mismo, es una técnica para aumentar la mantenibilidad del software, la facilidad con la que el software puede ser modificado [23]. La refactorización posibilita la transformación de programas preservando su comportamiento, modificando sólo

su estructura interna. También puede verse como una forma de mantenimiento preventivo, cuyo objetivo es disminuir la complejidad del software en anticipación a los incrementos de complejidad que los cambios pudieran traer [24].

La aplicación de refactorizaciones permite, partiendo de un mal diseño, reprocesarlo y convertirlo en un código bien diseñado. Los orígenes de esta práctica provienen del desarrollo software. Es por esto que es una práctica que está muy orientada al código fuente y no tanto a los principios clásicos de la Ingeniería del Software. De hecho, la refactorización está asociada al desarrollo de software ágil, en particular a la metodología de desarrollo Extreme Programming (XP) [87]. Una de las características principales de esta metodología de desarrollo ágil es la aplicación de refactorizaciones. Así, un desarrollo ágil basado en XP comenzará con un prototipo de la aplicación que se va a desarrollar, aplicando posteriormente refactorizaciones para mejorar el producto y prepararlo para la introducción de nuevas características.

La aplicación de refactorizaciones puede realizarse para diversos propósitos [22]:

- Para mejorar el diseño del software. A medida que se introducen cambios en el software, su diseño empeora. La aplicación de refactorizaciones ayuda a la mejora de la estructura de los programas. Un buen diseño permitirá que el software pueda modificarse más fácilmente, por lo que aumentará su mantenibilidad.
- Para hacer que el software sea más fácil de comprender. La aplicación de refactorizaciones hace que el programa sea más entendible para los seres humanos.
- Para encontrar errores. Al hacer el software más entendible, se conseguirá localizar más fácilmente errores en el software que en un mal diseño podrían pasar desapercibidos.
- Para programar más deprisa. Al ser más entendible y contar con un mejor diseño, el desarrollador podrá desarrollar más deprisa.

Uno de los problemas clásicos con los que se encuentra la refactorización es la detección de “oportunidades de refactorización”, es decir, lugares en el software que deberían ser refactorizados [20]. Entre las propuestas para la detección de estas oportunidades de refactorización, destaca la propuesta por Kent Beck [88] . Beck propone la utilización de lo que denomina “malos olores” (*bad smells* en terminología inglesa) en el código. Estos malos olores son partes del código fuente que presentan estructuras que la experiencia ha demostrado que son propensas a la aparición de problemas. Por ejemplo, algunos malos olores típicos

son la aparición de código duplicado, la existencia de métodos largos o clases largas, una excesiva cantidad de parámetros pasada a un método, la aparición de sentencias *switch*, o problemas más complejos de detectar, como las cadenas de mensajes entre objetos o la envidia entre objetos, es decir, la existencia de muchas llamadas desde un objeto de una clase a métodos de otra clase. Al detectar uno de estos malos olores, debería procederse a refactorizar para solucionar el problema, aplicando una refactorización adecuada al problema que va a tratarse.

Existen muchas refactorizaciones que pueden ser aplicadas sobre el software. Fowler [22] presenta un catálogo de 72 refactorizaciones que pueden ser aplicadas ante diferentes problemas encontrados en el código fuente. Este catálogo de refactorizaciones sigue manteniéndose y actualizándose, pueden encontrarse más de 90 refactorizaciones en el sitio Web oficial *www.refactoring.com*.

2.5 La Refactorización Basada en Valor

Desde el punto de vista de la Ingeniería del Software Basada en Valor, no todas las refactorizaciones son igualmente importantes, ya que algunas de ellas aportan más “valor” que otras. Por ello, es necesario proveer de mecanismos que gestionen esta diferenciación y priorizar las refactorizaciones en función del valor que pueden aportar, utilizando distintas técnicas y métodos.

Priorizar las refactorizaciones en función del valor que éstas aportan al software puede ayudar a las organizaciones a decidir qué refactorizaciones deben ser llevadas a cabo en primer lugar, y en qué partes del software deben ser aplicadas. Así, se consigue aumentar la calidad del software rápidamente ayudando a la organización a construir un mejor software y reduciendo costes de desarrollo y mantenimiento.

La refactorización basada en valor encaja en dos áreas de la ISBV. Por un lado, encaja en el área de diseño y desarrollo basado en valor, que pretende alinear el diseño y construcción del software con los objetivos de los implicados. Dado que varias metodologías de desarrollo software, como las metodologías ágiles o la programación extrema [87] se basan en el uso intensivo de la refactorización, deberíamos ser capaces de priorizar las refactorizaciones de acuerdo al valor que aportan si queremos que el proceso de desarrollo siga los principios de la ISBV. Por otro lado, la refactorización basada en valor se alinea con el mantenimiento basado en valor, que busca la evolución y mejora del software basándose en el valor que las posibles mejoras pueden aportar. De hecho, la refactorización en sí misma no es más que una forma de mantenimiento perfectivo. Por lo tanto, para

que dicho mantenimiento se realice siguiendo las normas de la ISBV, deberíamos ser capaces de identificar el valor aportado por las posibles refactorizaciones.

Para profundizar en la refactorización basada en valor se ha realizado una revisión de la literatura. Los resultados se detallan en la siguiente sección.

2.6 Revisión de la Literatura de la Refactorización Basada en Valor

Barbara Kinchenham sugirió en 2004 por primera vez la necesidad de la Ingeniería del Software basada en Evidencias [89]. Esta rama de la Ingeniería del Software trata de aplicar una aproximación basada en evidencias a investigación en la Ingeniería del Software tradicional.

El principal método utilizado en la práctica en la investigación basada en evidencias es la revisión sistemática de la literatura. Una revisión sistemática de la literatura proporciona una manera de identificar, evaluar e interpretar toda la investigación disponible sobre una cuestión de investigación, área o fenómeno de interés [90]. Las revisiones sistemáticas son una revisión metodológicamente rigurosa de los resultados de investigación, cuyo objetivo final es servir a los investigadores para proporcionar soluciones de Ingeniería del Software apropiadas en un contexto específico.

Antes de comenzar a realizar la revisión sistemática, es necesario comprobar que no se existe otra revisión de temática similar, para evitar la realización de un trabajo que ya ha sido realizado. Tras consultar las bases de datos que se muestran en la Tabla 2-3 se puede comprobar que esto no es así, por lo que se procede a comenzar el proceso de revisión sistemática.

La elaboración de una revisión sistemática de la literatura implica una serie de pasos bien estructurados y definidos. A continuación se describe detalladamente estos pasos para la revisión sistemática realizada.

2.6.1 Cuestiones de investigación

El primer paso en una revisión sistemática es definir claramente las cuestiones de investigación de interés, de cara a encontrar la información que ayude a identificar y enfocar futuras actividades de investigación en torno a dichas cuestiones. Este es el paso más importante en la revisión pues el resto del proceso será dirigido por estas cuestiones de investigación [90]. Para esta revisión sistemática se han definido las siguientes cuestiones de investigación:

- **Cuestión 1**: ¿Qué estudios se han realizado sobre el valor del software y las refactorizaciones?
    - **Cuestión 1a**: ¿Existen evidencias de que una refactorización puede aumentar el valor del producto?
    - **Cuestión 1b**: ¿Existen trabajos que aúnen la Ingeniería del Software basada en valor y las refactorizaciones?
- **Cuestión 2**: ¿Qué indicios existen de que la realización de refactorizaciones mejoren la mantenibilidad del software?
    - **Cuestión 2a**: ¿Existen refactorizaciones que empeoren la mantenibilidad del producto?
- **Cuestión 3**: ¿Qué método de refactorización es más utilizado en la práctica? ¿Es mejor realizar la refactorización automáticamente o sólo indicar los puntos en los que puede refactorizarse y la técnica a aplicar, y que sea el usuario el que los realice (refactorización asistida)?
    - **Cuestión 3a**: ¿Desaparecen los costes de refactorizar al realizar refactorizaciones automáticas?
    - **Cuestión 3b**: ¿Existen razones por las cuales una refactorización no puede realizarse completamente automática?

Respecto a la cuestión de investigación 1, con ella se pretende encontrar una respuesta a la pregunta de cómo se ha relacionado el concepto de valor con las refactorizaciones hasta ahora, así como de qué manera una refactorización puede afectar al valor del producto.

En cuanto a la cuestión de investigación 2, con ella se intenta obtener más información del grado en el que una refactorización afecta a la mantenibilidad del software, y de qué manera lo hace. Esta revisión se centra sólo en la característica de la mantenibilidad de la calidad del software, por lo que no se considera la cuestión de cómo afectan las refactorizaciones a otras características de la calidad, como pueden ser la usabilidad o la eficiencia.

Por último, la cuestión de investigación 3 aborda el problema de las refactorizaciones automáticas. Con ella se quiere conocer qué tipo de refactorización es actualmente la más recomendable, así como de qué manera puede afectar apoyarse en herramientas a los costes asociados con una refactorización.

2.6.2 Estrategia de búsqueda utilizada

El siguiente paso es definir una estrategia de búsqueda. Para el desarrollo de la estrategia seguida, este trabajo se basa en los pasos descritos en [90] y [91]:

- Realización de una búsqueda preliminar para encontrar una revisión sistemática similar a la que va a ser realizada y conocer el volumen de resultados que pueden obtenerse relacionados sobre el tema de la revisión.
- Realización de diferentes búsquedas de prueba para encontrar términos relacionados que puedan ser incluidos en la búsqueda.
- Identificación de sinónimos y alternativas a los términos de búsqueda encontrados, para aumentar los resultados obtenidos.
- Utilización de operadores booleanos AND y OR entre los distintos términos, para optimizar las búsquedas.

Así, a partir de las cuestiones de investigación enumeradas en la sección 2.6.1, la cadena de búsqueda que se ha utilizado en los distintos buscadores consultados es la siguiente:

((value OR valued OR value-based) AND (refactoring OR refactor) AND (maintainability OR software maintenance) AND (software OR application software OR software system))

2.6.3 Proceso de búsqueda

El proceso de búsqueda consiste en consultar las diferentes fuentes de datos seleccionadas que se enumeran en la Tabla 2-3 (bases de datos online, motores de búsqueda, revistas electrónicas y actas de conferencia, utilizando la cadena de búsqueda definida en la sección anterior. La selección de estas fuentes es debido a que han sido utilizadas en anteriores revisiones sistemáticas sobre Ingeniería del Software [92][93] y contienen publicaciones que se consideran relevantes para el área de interés.

En este caso no se ha establecido un umbral temporal para la selección de trabajos, sino que se han tenido en cuenta todos los estudios publicados hasta agosto de 2011 que se han encontrado en las bases de datos seleccionadas.

Utilizando el término de búsqueda indicado, se obtienen una gran cantidad de resultados, pero muchos de ellos se consideraron irrelevantes para el objeto de este estudio. En la Tabla 2-3 se muestra el resumen de los resultados obtenidos.

En ocasiones la consulta a bibliotecas digitales no suele ser suficiente para recuperar todos los trabajos relacionados con el objeto del estudio y es preciso acudir a recursos más específicos que se espera que contengan más trabajos relacionados. Por ejemplo, en este caso además de los recursos electrónicos que muestra la Tabla 2-3, se han estudiado las siguientes conferencias:

- Las 8 ediciones del "Workshops on Economics-Driven Software Engineering Research (EDSER)", desde el año 1999 hasta el año 2006.
- La conferencia "Computer Society Conference on Exploring Quantifiable Information Technology Yield (EQUITY)", del año 2007.

Base de datos	Total	Repetidos	Relevantes	Incluidos
ACM Digital Library	915	0	12	6
Springer Link	383	92	3	2
Science Direct	203	69	1	1
ProQuest Computing	2	2	1	0
IEEE Computer Society	100	24	8	3
IEEE Xplore	57	17	1	0
Current Contents	6	4	1	0
Total	**1666**	**208**	**27**	12

Tabla 2-3. Resultados de la búsqueda (1/11/2011)

2.6.4 Criterios de inclusión y exclusión

El criterio de inclusión para la revisión sistemática considera estudios que:

- Dirijan su investigación hacia la refactorización guiada por el valor. Por ejemplo, se incluirán estudios que estudien la relación existente entre la realización de una refactorización y el valor aportado por la misma.
- Permitan estudiar cómo identificar puntos de refactorización en el software para aumentar su valor, así como las técnicas a utilizar para su realización. Queda incluido cualquier trabajo que identifique candidatos a la refactorización en el software siempre que se justifique el aumento del valor que aporta su aplicación.
- Traten sobre la característica de la mantenibilidad y especifiquen cómo una refactorización puede mejorarla. Se incluyen estudios que realicen

afirmaciones sobre cómo influye la refactorización en la mantenibilidad del software.

Quedarán excluidos los estudios que:

- Traten otras características de la calidad diferentes, como puede ser la usabilidad. Este estudio se centra en la característica de la mantenibilidad de la calidad del software, un campo suficientemente extenso como para considerarlo independientemente del resto de las características de calidad. Por ello, los resultados del resto de características de la calidad quedarán excluidos.
- Presenten técnicas de refactorización que no influyan en el valor del producto software. Todos los estudios resultantes que tengan relación con la refactorización, pero que no identifiquen cómo influirá ésta en el valor del software.
- No tengan ninguna relación con el tema tratado. Todo resultado que no esté relacionado con el tema en el que se centra el estudio quedará excluido.

2.6.5 Selección de estudios

En la Figura 2-10 puede observarse el proceso de selección y el número de estudios restantes tras la aplicación de cada uno de los pasos.

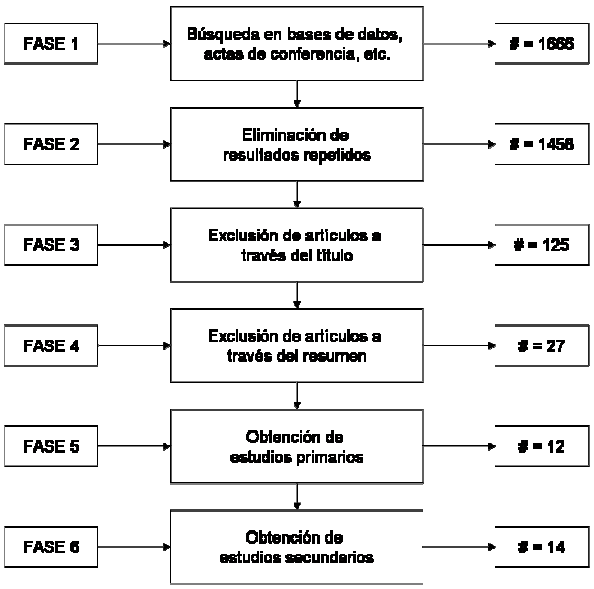


Figura 2-10. Fases del proceso de selección de estudios

Tras una primera búsqueda se identificaron 1666 estudios, de los cuales 208 aparecían por duplicado y fueron excluidos. Tras esto, 1333 estudios fueron eliminados debido a que el título no encajaba con los criterios de selección. En el cuarto paso 98 artículos fueron excluidos tras la lectura de los resúmenes, pasando los restantes 27 a formar parte del siguiente paso, la lectura completa de artículos. En la Tabla 2-4 puede observarse la lista de artículos seleccionados y que pasaron a la fase de obtención de estudios primarios.

Ref.	Título
[20]	A Survey of Software Refactoring
[24]	Understanding the Economics of Refactoring
[94]	A quantitative approach for evaluating the effectiveness of refactoring in software development process
[95]	Metrics based refactoring
[96]	Drivers for software refactoring decisions
[97]	A novel approach to optimize clone refactoring activity
[98]	Metrics to study symptoms of bad software designs
[99]	"Program, enhance thyself!": demand-driven pattern-oriented program enhancement
[100]	Which warnings should I fix first?
[101]	Scheduling of conflicting refactorings to promote quality improvement
[102]	Breaking the barriers to successful refactoring: observations and tools for extract method
[103]	How we refactor, and how we know it
[104]	Analysing refactoring dependencies using graph transformation
[105]	A Model to Identify Refactoring Effort during Maintenance by Mining Source Code Repositories
[106]	Refactoring Effect Estimation Based on Complexity Metrics
[107]	A study of reasoning processes in software maintenance management
[108]	Refactoring--Does It Improve Software Quality?
[109]	Economics-Driven Software Mining
[110]	Assessing the Maintainability Benefits of Design Restructuring Using Dependency Analysis
[111]	A Quantitative Evaluation of Maintainability Enhancement by Refactoring
[112]	The effects of design pattern application on metric scores
[113]	Evaluating Architectural Stability with Real Options Theory

Ref.	Título
[114]	Search-Based Refactoring: an empirical study
[115]	Applying ArchOptions to value the payoff of Refactoring
[116]	Search-Based Refactoring for Software Maintenance
[117]	Empirical investigation of refactoring effect on software quality
[118]	Designing Systems for Adaptability by Means of Architecture Options
[119]	A Quantitative Evaluation of Software Quality Enhancement by Refactoring Using Dependency Oriented Complexity Metrics
[120]	An expert system for determining candidate software classes for refactoring
[121]	Digging the development dust for refactorings

Tabla 2-4. Lista preliminar de estudios seleccionados

Tras la lectura detallada de los 27 trabajos, se seleccionaron 12 de ellos para proceder a comenzar con la fase de selección de estudios secundarios, en la que se analizan las referencias incluidas en los artículos seleccionados para determinar si esos trabajos pudieran ser incorporados como trabajos relevantes. Se han incorporado dos trabajos más a la revisión [122][123]. La lista de trabajos finalmente seleccionados se enumera en la Tabla 2-5.

ID	Ref.	Título
S1	[108]	Refactoring--Does It Improve Software Quality?
S2	[109]	Economics-Driven Software Mining
S3	[110]	Assessing the Maintainability Benefits of Design Restructuring Using Dependency Analysis
S4	[111]	A Quantitative Evaluation of Maintainability Enhancement by Refactoring
S5	[20]	A Survey of Software Refactoring
S6	[112]	The effects of design pattern application on metric scores
S7	[24]	Understanding the Economics of Refactoring
S8	[113]	Evaluating Architectural Stability with Real Options Theory
S9	[114]	Search-Based Refactoring: an empirical study
S10	[115]	Applying ArchOptions to value the payoff of Refactoring
S11	[116]	Search-Based Refactoring for Software Maintenance
S12	[117]	Empirical investigation of refactoring effect on software quality
S13	[118]	Designing Systems for Adaptability by Means of Architecture Options

S14	[119]	A Quantitative Evaluation of Maintainability Enhancement by Refactoring

Tabla 2-5. Lista de estudios seleccionados

2.6.6 Evaluación de calidad de los estudios

Para la evaluación de calidad de los estudios, se ha creado una lista de verificación que permite evaluar la calidad de los estudios seleccionados y su relevancia de cara a responder a las cuestiones de investigación planteadas.

Para la elaboración de la lista de verificación se seleccionaron 16 preguntas contra las que se evalúa cada estudio. Las preguntas han sido elaboradas a partir de los trabajos de Crombie [124], Fink [125] y Kitchenham [90].

Se ha utilizado la siguiente escala de puntuación: “S” si el estudio cumple lo que indica la pregunta, “P” si lo cumple parcialmente y “N” en caso de que el estudio no cumpla con el criterio que lo evalúa. En la Tabla 2-6 pueden observarse los resultados de evaluar la lista de verificación sobre los estudios seleccionados. Finalmente, asignando valores a las puntuaciones anteriores (S=1, P=0,5, N=0) se obtiene el indicador de calidad de cada estudio en relación con la revisión sistemática que se lleva a cabo.

Como puede verse, la calidad de los estudios es dispar. Esta evaluación de la calidad se tendrá en cuenta para la obtención de las diferentes conclusiones sobre los estudios, dando más relevancia a los estudios que han obtenido una mejor calidad.

2.6.7 Extracción y análisis de datos

Durante el proceso de extracción de datos se han repartido los trabajos seleccionados entre los autores de la revisión, de manera que cada autor ha extraído la información para realizar una posterior puesta en común. Se ha optado por seguir este método de trabajo porque, aún cuando no es el método estandarizado [89] ha demostrado ser de utilidad en la práctica [126].

Entre los datos recabados para cada artículo, se encuentran datos sobre el mismo (título, autores, año de publicación, revista, país, etc.), datos utilizados para responder a las preguntas de investigación y los datos para poder realizar el estudio cuantitativo que se muestra en la sección anterior.

Descripción	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14
¿Son claras las cuestiones de investigación?	S	S	S	S	P	S	S	S	S	S	S	S	S	P
¿Se construye el estudio con estudios previos?	S	S	S	S	S	P	S	S	S	S	S	P	S	P
¿Están los métodos de derivación y construcción completamente definidos?	S	N	S	S	S	S	P	P	P	P	P	P	S	N
¿Son los métodos utilizados los más relevantes?	P	P	P	P	S	S	P	S	S	S	S	S	S	S
¿Están correctamente descritos los métodos de recolección de datos?	S	P	S	N	N	P	S	P	P	P	P	N	S	S
¿Justifica los métodos estadísticos?	N	N	P	S	N	S	P	S	S	S	S	P	S	S
¿Es claro el propósito del análisis de datos?	S	P	S	S	S	S	S	S	S	S	S	S	S	P
¿Se presentan los indicios negativos?	N	N	N	N	P	N	N	S	S	S	S	S	S	P
¿Se discute sobre los problemas de la validez/fiabilidad de los resultados?	P	N	N	N	P	S	N	N	N	N	N	P	N	N
¿Puede el estudio replicarse?	S	N	P	P	S	S	P	S	S	S	S	S	S	S
¿Está definido el diseño de la investigación?	S	S	S	S	N	S	S	P	P	P	P	P	P	P
¿Es apto el diseño de la investigación para llevar a cabo el estudio?	P	S	S	S	P	S	S	P	P	P	P	P	P	P
¿Son creíbles los resultados?	P	S	S	S	S	S	S	S	S	S	S	S	S	S
¿Está el proceso de investigación descrito?	S	P	S	S	N	S	P	P	P	P	P	P	P	P
¿Están claros los enlaces entre los datos, la interpretación y las conclusiones?	P	S	S	S	S	S	S	S	S	S	S	S	S	S
¿Es el artículo claro y coherente?	S	S	S	S	S	S	S	S	S	S	S	S	S	S
Puntuaciones totales obtenidas:	**11.5**	**9**	**12.5**	**12**	**10**	**14**	**11.5**	**12.5**	**12.5**	**12.5**	**12.5**	**11.5**	**13.5**	**10.5**

Tabla 2-6. Evaluación de los artículos seleccionados

2.6.8 Resultados

La Tabla 2-7 muestra un resumen de los estudios seleccionados, indicando la técnica de investigación aplicada y la estrategia que utilizan para unir el valor con la refactorización. De este modo, la tabla resume los resultados del proceso de extracción de datos frente a las cuestiones de investigación que cada trabajo cubre.

Las investigaciones recogidas muestran el uso de diferentes técnicas para abordar el problema de la refactorización y el valor que pueden aportar al software que será refactorizado. Sin embargo, mientras que en (S1, S2, S4, S5, S6 y S14) el objetivo principal es verificar si la refactorización conlleva un aumento de la calidad y, por extensión, de la mantenibilidad, en (S3 y S7) se intenta cuantificar el valor que aportará la refactorización que va a ser realizada. Es de reseñar que, aunque los estudios traten de comprobar o cuantificar la mejora que aporta una refactorización, en ninguno de ellos aparece ninguna referencia a la ISBV.

Como se ha mencionado, en estos trabajos se describen diferentes técnicas utilizadas para abordar el problema de la refactorización. En (S3, S4, S7 y S14) se utilizan los "malos olores" propuestos en [88] para identificar candidatos a refactorización dentro del código. En estos estudios, las oportunidades de refactorización son revisadas para conseguir un "plan detallado de reestructuración", esto es, el orden de aplicación de las refactorizaciones. En (S1, S6, y S14) se propone la extracción de métricas del código antes y después de la refactorización para conocer los efectos que tiene la refactorización en las mismas. Así, (S1) utiliza métricas como LCOM (falta de cohesión en los métodos) o CBO (acoplamiento entre objetos) para concluir que las refactorizaciones no mejoran la calidad, mientras que (S6) utiliza métricas diferentes, como COF (factor de acoplamiento en el sistema) para alcanzar la conclusión contraria. (S14), por su parte, referencia métricas generales de acoplamiento para evaluar los efectos de las refactorizaciones *Move Method*, *Extract Method* y *Extract Class*. En ese sentido S12 también utiliza una serie de métricas extraídas directamente del código, pero en este caso para comprobar si la refactorización afectará positiva o negativamente a la calidad. Aunque este no es un estudio relacionado directamente con el valor, se encuentra relacionado con la segunda cuestión de investigación.

ID	Investigación realizada	Estrategia para unir valor y refactorización
S1	Análisis de métricas antes y después de refactorización.	Aumento de calidad del producto con las refactorizaciones. No existe correlación entre aumento de calidad y refactorización.
S2	Minería de repositorios como técnica para tomar decisiones económicas	El valor añadido otorgado por la refactorización proviene de sus "calidades" y dependencias.
S3	Identificación de oportunidades de refactorización mediante "malos olores".	ROI = ahorro de costes / coste de refactorización. Ahorro de costes asociado al ahorro en las pruebas de regresión.
S4	Identificación de candidatos mediante "malos olores"	Evaluación de los efectos de la refactorización. Coste de refactorización Mejora de la mantenibilidad
S5	Metodología para refactorizar. Evaluación de los efectos	Clasificación de refactorizaciones según las características de calidad que afectan.
S6	Evaluación del efecto de los patrones sobre las métricas	Las refactorizaciones se utilizan para mejorar y aumentar el valor.
S7	Análisis del código fuente para conocer el ROI.	Cálculo del ROI en función de las pruebas de regresión.
S8	Utilización de la teoría de opciones reales para calcular el valor aportado por una refactorización desde el punto de vista de la estabilidad arquitectural	Teoría de opciones reales, aplicado a la arquitectura. Utiliza la refactorización como una manera de mejorar la estabilidad de la arquitectura. Tiene en cuenta el valor del cambio y el coste de realizarlo.
S9	Compara varias técnicas de búsqueda para encontrar puntos de refactorización en el código.	No trata el problema del valor aportado por la refactorización, si no de la identificación de refactorizaciones.
S10	Utilización de la teoría de opciones reales para calcular el valor aportado por una refactorización desde el punto de vista de la flexibilidad arquitectural ante futuros cambios.	Teoría de opciones reales, aplicado a la arquitectura. Utiliza la refactorización como una manera de mejorar la flexibilidad de la arquitectura. Tiene en cuenta el valor del cambio y el coste de realizarlo.

ID	Investigación realizada	Estrategia para unir valor y refactorización
S11	Compara técnicas de búsqueda para ver cuál es la más adecuada para automatizar la refactorización del código.	No trata el problema del valor que aporta una refactorización, sino que trata de la automatización de refactorizaciones.
S12	Intenta comprobar si las refactorizaciones mejoran la calidad del código. Se basa en el impacto visible de la calidad externa de diferentes sistemas antes y después de la realización de refactorizaciones.	Comprueba la variación de calidad debido a la refactorización.
S13	Al igual que S8 y S10 utiliza la teoría de opciones reales para calcular el valor de un sistema durante todo su ciclo de vida, desde el punto de vista de la adaptabilidad.	No define de forma explícita una relación entre la refactorización y el valor, aunque sigue la misma línea de investigación que los artículos S8 y S10. Define una aproximación para calcular el grado de adaptabilidad de un sistema a lo largo del ciclo de vida y elegir la arquitectura que maximice el valor del sistema. Define al valor de un sistema como la facilidad con la que se satisfacen las necesidades de los usuarios
S14	Se propone un método de evaluación cuantitativo para medir la mejora de la mantenibilidad de acuerdo con las refactorizaciones realizadas.	El estudio se focaliza el estudio de los efectos de determinadas refactorizaciones. Se menciona la importancia costes de refactorización, pero no se indica nada acerca de los beneficios económicos. No se menciona directamente al valor. Se utilizan métricas de acoplamiento para evaluar los efectos de la refactorización

Tabla 2-7. Resumen de artículos seleccionados

El estudio (S5) puede verse como un resumen extenso de las investigaciones hechas sobre refactorizaciones. Entre los temas que trata, analiza la necesidad de conocer qué características de la calidad del software son afectadas por una refactorización, y de qué manera. Para ello, es necesario analizar cada una de las refactorizaciones teniendo en cuenta su propósito y los efectos que tiene. Por último, (S2) propone la técnica de la "minería de repositorios" para apoyar la toma de decisiones económicas en cuanto a las refactorizaciones. Los datos extraídos tras la aplicación de esta técnica pueden ser analizados y utilizados para apoyar la toma de decisiones de inversión relacionadas con el desarrollo y evolución de un sistema software.

Los artículos (S8, S10 y S13) utilizan la teoría de opciones para calcular el valor. Esta teoría, típicamente económica, establece una opción como la capacidad de ejecutar una acción en una fecha futura, en función de que se cumplan unas determinadas condiciones. De esta manera, (S8) y (S10) utilizan la teoría aplicándola a arquitectura para estimar el valor de refactorizar. En función de los beneficios estimados, puede realizarse o no una refactorización. (S13), también aplica la teoría a la arquitectura, utilizándola para estimar la arquitectura óptima de un sistema desde el punto de vista de la adaptabilidad de su arquitectura. Así, a través de los stakeholders del sistema calculará un valor deseado y lo comparará con el valor actual del sistema. Si los costes de la mejora son menores que el valor que ha perdido el sistema con el tiempo, será necesario aplicar esa mejora. Este artículo menciona la necesidad de estudiar con mayor detalle la cuantificación de los beneficios al realizar mejoras en el sistema.

Los artículos (S9) y (S11) presentan diferentes técnicas de búsqueda para identificar partes del código que pueden ser refactorizadas, teniendo en cuenta aspectos de inteligencia artificial y maximizando una función relacionada con métricas de diseño. De esta manera, consiguen mejorar la automatización de refactorizaciones, identificando el algoritmo óptimo para realizar la búsqueda. Cabe recalcar que no se tiene en cuenta en las búsquedas los aspectos del coste de las refactorizaciones, sino solamente la maximización de la mejora en la calidad dependiente de un modelo de medición determinado.

Las siguientes secciones analizan en detalle la información recuperada tras el análisis de los trabajos en relación con cada cuestión de investigación planteada.

2.6.9 Cuestión de investigación 1

La cuestión de investigación 1 trata de identificar la relación entre el trabajo analizado y el tema de interés (valor aportado por una refactorización). Además, esta cuestión incluye dos subcuestiones dirigidas a identificar si el trabajo establece mecanismos para reconocer el valor aportado por una refactorización.

Respecto a los resultados, un ejemplo significativo de lo inmaduro del estado de la cuestión de investigación lo representan los trabajos (S1) y (S6). Aunque ambos se centran en estudiar si las refactorizaciones pueden relacionarse con un aumento de la calidad y la mantenibilidad, llegan a conclusiones diametralmente opuestas. Para (S6) las refactorizaciones mejoran la calidad del software, y lo justifica en virtud de la variación que se produce solo en una métrica (COF). Sin embargo, para (S1) la refactorización suele acarrear una pérdida de calidad, basando esta afirmación en la variación surgida en otras métricas (LCOM y CBO). El trabajo (S12) concluye que no puede demostrar la relación directa de las refactorizaciones respecto del aumento de la calidad y aconseja estudiar con mayor profundidad el tema.

Por otro lado, tanto (S2), como (S4) y (S5) parten de la premisa de que es necesario conocer el valor que puede aportar una refactorización como paso previo a la toma de decisiones respecto a su aplicación. Para ayudar en esta toma de decisiones, cada artículo plantea una técnica para conocer el valor que la refactorización puede aportar al software. (S2) propone utilizar la técnica de la minería de repositorios para extraer datos que ayuden a conocer el valor aportado por una refactorización y así poder tomar decisiones como consecuencia de los resultados obtenidos, aún así, no menciona la ISBV ni desarrolla las técnicas necesarias. (S4 y S14) por su parte utilizan los “malos olores” definidos en [88] para identificar candidatos a la refactorización, posibilitando así la creación de un plan de aplicación de las refactorizaciones. Por su parte, en (S5) propone el análisis de las refactorizaciones para conocer a qué característica de la calidad afectan y poder establecer así una clasificación de refactorizaciones.

Finalmente, (S3 y S7) profundizan más en la priorización de refactorizaciones y su relación con el ROI. Según estos trabajos, este valor del ROI será calculado en función del coste de la refactorización y el ahorro en el mantenimiento futuro que puede obtenerse al aplicarla. Así, una vez identificadas todas las posibles refactorizaciones, pueden priorizarse en función del valor del ROI obtenido para cada una de ellas.

Como puede observarse, en los estudios seleccionados no aparece ninguna referencia explícita a la ISBV. Sin embargo, todos tratan el problema del valor tratando de cuantificarlo o de priorizar las mejoras para aumentar el valor. De hecho, es comúnmente aceptado cuantificar el valor en función del ROI obtenido [127], tal como se hace en (S3) y (S7).

En cuanto a los siguientes artículos, existen dos grupos bien diferenciados. Por un lado están los trabajos (S8, S10 y S13) que aplican una teoría económica, la teoría de opciones reales, para cuantificar el valor que aporta un cambio. En los tres casos se tratan aspectos de la arquitectura del sistema. De esta manera, los artículos evalúan el retorno en el tiempo de una inversión en la mejora de la flexibilidad, estabilidad o adaptabilidad de la arquitectura. El resultado final se evalúa en unidades monetarias por lo que facilita comparaciones, aunque los modelos no sean del todo detallados y no tengan en cuenta diferentes tipos de refactorizaciones. Por otro lado los trabajos (S9 y S11) se centran en la búsqueda y priorización de refactorizaciones maximizando funciones de desempeño (en inglés fitness function). Las funciones de desempeño tienen en cuenta, en ambos casos, los resultados del modelo de medición de calidad QMOOD [128]. Por lo tanto la priorización se basa solamente en la mejora de calidad, sin tener en cuenta aspectos de valor.

A continuación se resume en la Figura 2-11 el número de trabajos que tienen en cuenta los costes y beneficios económicos de una refactorización respecto de los que solo tienen en cuenta aspectos de mejora en las características de calidad.

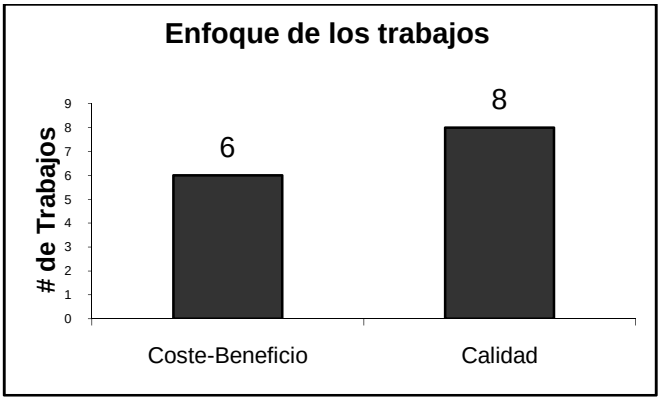


Figura 2-11. Enfoque para la cuestión 1 de los trabajos encontrados

En la Tabla 2-8 se resume la relación entre valor y refactorización encontrado en cada estudio.

ID	Relación entre valor y refactorización
S1	Búsqueda de relación entre refactorización y aumento de calidad. Utiliza métricas extraídas del código para comprobarlo. No encuentra la relación.
S2	Es necesario conocer los costes y el beneficio que añade una refactorización para saber si merece la pena realizarla. Propone la minería de repositorios como técnica para conocerlo. Aproximación de alto nivel.
S3	ROI = ahorro de costes de mantenimiento / coste de refactorización. Ahorro de costes de mantenimiento: ahorro en el coste de las pruebas de regresión. Coste de refactorización: no indicado. Refactorizaciones a nivel de clase.
S4	Identificación de la necesidad de evaluar los efectos de refactorizar para tomar decisiones económicas. Coste de refactorización. Aumento de la mantenibilidad.
S5	Análisis de las refactorizaciones para conocer a qué característica de la calidad afectan. Sin adecuarse a ningún modelo de calidad.
S6	Las técnicas de refactorización mejoran la calidad del software. Refactorizaciones estudiadas a nivel de diseño.
S7	ROI en función de las pruebas de regresión. El ROI indica el valor que aporta refactorizar una clase. Refactorizaciones a nivel de clase y método.
S8	Aplicación de la teoría de opciones reales a la elección de arquitectura. Identificación de refactorización con mejora en la flexibilidad, y cálculo del valor en función de la mejora en la flexibilidad de la arquitectura.
S9	No trata el tema
S10	Aplica la teoría de opciones reales para cuantificar el valor aportado de la aplicación de una refactorización.
S11	No trata el tema
S12	Busca la relación entre refactorización y aumento de calidad.
S13	No trata la refactorización como tal, aunque si propone la utilización de la teoría de opciones para estimar en qué momento debe realizarse una mejora, basándose en el valor del producto.
S14	Propone un método de evaluación cuantitativo para medir la mejora de la mantenibilidad basadas en las refactorizaciones, enfocándose en métricas de acoplamiento. Indica la importancia de la maximización del coste-efecto pero no indica nada acerca del cálculo de coste.

Tabla 2-8. Cuestión de investigación 1: evidencias de la relación entre refactorización y aumento de valor

2.6.10 Cuestión de investigación 2

La cuestión de investigación 2 se centra en la búsqueda de evidencias que relacionen la refactorización con un aumento de la mantenibilidad del software. Mientras que en (S1) se llega a la conclusión de que las refactorizaciones no mejoran la mantenibilidad del software apoyándose para ello en métricas extraídas del software, en el resto de artículos que también tratan el tema (S3, S4, S5, S6, S7 y S14) se llega a la conclusión contraria. Estos artículos defienden que las refactorizaciones mejoran la mantenibilidad del software. Más concretamente, en (S4) se indica que hay pocos estudios cuantitativos que demuestren que la mantenibilidad es influida positivamente por las refactorizaciones. A lo largo del artículo realiza un estudio cuantitativo llegando a la conclusión de que la realización de una refactorización mejora la mantenibilidad. Parar realizar esta afirmación estudia el acoplamiento entre los diferentes métodos del software, utilizando para ello métricas extraídas directamente del mismo y analizando los valores obtenidos antes y después de la aplicación de una refactorización. En (S7) se indica que las refactorizaciones son de gran utilidad durante el mantenimiento perfectivo, el tipo de mantenimiento en el que más esfuerzo se necesita [129]. Finalmente, en (S3) se afirma que un buen diseño es menos costoso de mantener que uno malo, por lo que la mejora del diseño derivada de una refactorización implica un aumento de la mantenibilidad del software. Un caso especial es (S12), que al calcular los efectos que tiene sobre la calidad una refactorización, calcula también el efecto sobre la característica de mantenibilidad. Como resultado, concluye que no puede decirse que las refactorizaciones mejoren o empeoren la mantenibilidad. En (S8, S10 y S13) se relacionan las refactorizaciones y en general las mejoras del sistema con un aumento en la flexibilidad, estabilidad y adaptabilidad de la arquitectura, lo que va a ayudar al mantenimiento futuro, recalcando la tendencia natural de las aplicaciones a evolucionar. En (S9 y S11) se analiza el impacto en la calidad que tiene la aplicación de refactorizaciones en función de diferentes métodos de búsqueda que son utilizados para refactorizar. Aunque no se trata la mantenibilidad como característica independiente, si se indica como objetivo general la reducción de costes de mantenimiento a partir de las refactorizaciones.

A continuación se resume en la Figura 2-12 el número de trabajos que afirman el aumento de la mantenibilidad con las refactorizaciones (+) o su

disminución (-). También se indica el número de artículos que no realizan ninguna afirmación (=).

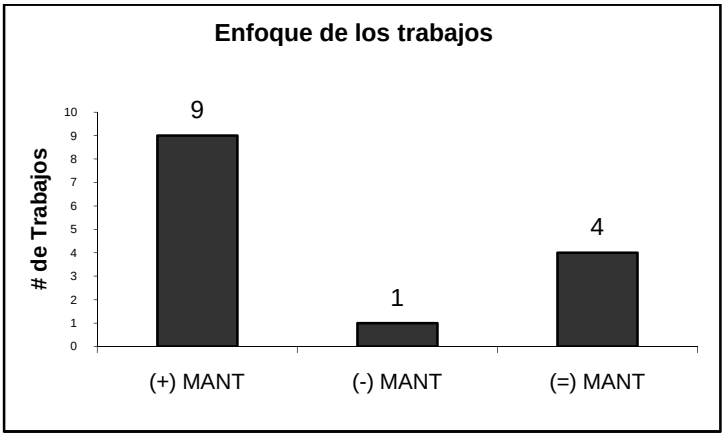


Figura 2-12. Enfoque para la cuestión 2 de los trabajos encontrados

En la Tabla 2-9 se resume la relación encontrada entre refactorización y aumento de la mantenibilidad en los artículos analizados.

ID	Relación entre refactorización y aumento de la mantenibilidad
S1	No mejoran la calidad, afectando negativamente entre otras a la mantenibilidad.
S2	No realiza ningún estudio o afirmación sobre si la refactorización aumenta o no la mantenibilidad.
S3	Relación mantenibilidad – mejora en el diseño del software. Las refactorizaciones mejoran la calidad del software.
S4	Estudio cuantitativo que demuestra que las refactorizaciones mejoran la mantenibilidad.
S5	Se refactoriza para mejorar la mantenibilidad, reduciendo así costes de mantenimiento.
S6	Las refactorizaciones mejoran la calidad. Destaca la mantenibilidad y reutilización.
S7	Las refactorizaciones ayudan al mantenimiento perfectivo, el más costoso de todos.
S8	No identifica la relación directa entre refactorización y mantenibilidad, aunque si relaciona la refactorización con un aumento en la flexibilidad, lo que lleva a mejorar el mantenimiento.
S9	No indica nada.

ID	Relación entre refactorización y aumento de la mantenibilidad
S10	Relaciona la refactorización con la mantenibilidad. Utiliza las refactorizaciones para modificar el software y posteriormente calcula el valor como la variación en la mantenibilidad.
S11	Al probar los diferentes algoritmos de búsqueda, analiza el aumento de calidad que se produce con la aplicación de la refactorización. Sin embargo, no habla de mantenibilidad, sino de calidad (flexibilidad, reusabilidad e inteligibilidad)
S12	Comprueba el efecto de la refactorización en la mantenibilidad, entre otras características de la calidad. Indica que no existe una relación directa entre refactorización y aumento de la mantenibilidad en todos los casos.
S13	No indica nada, aunque cita la mantenibilidad entre los atributos necesarios para calcular el valor.
S14	El artículo solo muestra resultados de aumento de la mantenibilidad respecto de las refactorizaciones.

Tabla 2-9. Cuestión de investigación 2: evidencias de la relación entre refactorización y aumento de mantenibilidad

2.6.11 Cuestión de investigación 3

La cuestión de investigación 3 se centra en el problema de la automatización de refactorizaciones. En (S1 y S7) se establece que lo más complicado es la localización de oportunidades de refactorización. Por ello proponen la búsqueda manual de áreas problemáticas potencialmente refactorizables. En este sentido (S3) se puede contemplar como un refuerzo de la afirmación anterior, ya que aunque reconoce que la refactorización manual es muy costosa, es la más fiable ya que se realiza a partir del conocimiento del desarrollador. Por otro lado (S4, S5 y S14) se inclinan más por las refactorizaciones asistidas, es decir, utilizar herramientas de soporte que ayuden a llevar a cabo el proceso de refactorización, pero dejando siempre que sea el desarrollador quien tome la decisión de refactorizar. Para respaldar esta propuesta, (S5) expone que la refactorización asistida o semiautomática es actualmente la mejor opción y la más utilizada en la práctica, ya que la mayor parte del conocimiento requerido para realizar la refactorización reside en el propio desarrollador y que dicho conocimiento no puede ser extraído directamente del software a refactorizar.

Tanto (S9) como (S11) analizan diferentes técnicas de refactorización automática. (S9) analiza 3 técnicas diferentes de refactorización basada en búsqueda para identificar posibles puntos a refactorizar. Por lo tanto, propone técnicas para automatizar la refactorización. (S11) analiza igualmente 4 técnicas diferentes para automatizar las refactorizaciones. Estas técnicas se basan en diferentes algoritmos. Se intenta comprobar cuál de las técnicas es más efectiva, concluyendo que todas son efectivas.

A continuación se resume en la Figura 2-13 el número de trabajos que tienen en cuenta las refactorizaciones manuales, asistidas, automáticas o que por el contrario no tratan sobre esta tipología.

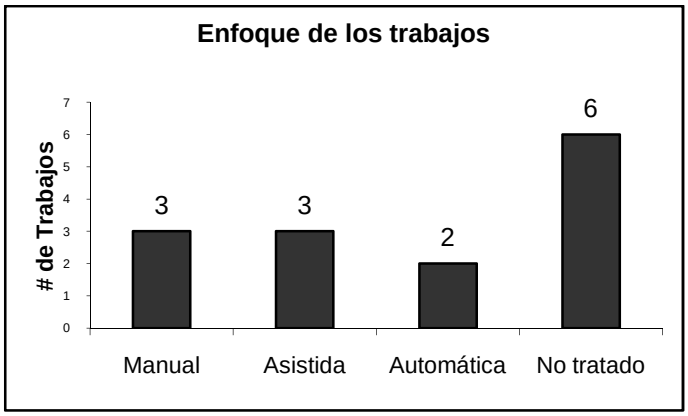


Figura 2-13. Enfoque para la cuestión 3 de los trabajos encontrados

En la Tabla 2-10 se resume el grado de automatización de las refactorizaciones tenidas en cuenta en los artículos.

ID	Automatización de la refactorización y valor
S1	Búsqueda manual. Complejidad de la identificación de áreas problemáticas.
S2	No trata sobre qué tipo de refactorización es mejor.
S3	Refactorización manual más costosa y fiable.
S4	Refactorización asistida. Uso de herramientas de ayuda.
S5	Refactorización semiautomática es la más utilizada en la práctica. El conocimiento requerido está implícito en el desarrollador.
S6	No trata sobre qué tipo de refactorización es mejor.
S7	Búsqueda manual de oportunidades de refactorización.
S8	No trata el tema.

ID	Automatización de la refactorización y valor
S9	Propone la automatización de refactorizaciones, comparando 3 métodos diferentes de búsqueda.
S10	No trata el tema.
S11	Analiza 4 técnicas de búsqueda para la refactorización automática, basándose en diferentes algoritmos, para comprobar cuál de ellas es más efectiva
S12	No trata el tema.
S13	No trata el tema.
S14	Realiza y aconseja refactorizaciones asistidas, en este caso particular debido a que el método necesita de pasos manuales.

Tabla 2-10. Cuestión de investigación 3: evidencias sobre la refactorización automática

2.6.12 Discusión

Los resultados de la revisión sistemática llevada a cabo demuestran que la refactorización basada en valor está muy poco estudiada, ya que no aparece reflejada explícitamente en ninguno de los artículos seleccionados. Sin embargo, en los estudios aparecen intentos de demostrar qué efectos pueden aportar al software las refactorizaciones. Así, en ellos se estudia si la realización de refactorizaciones aumenta la mantenibilidad del software y, por tanto, su calidad. Pese a que en (S1) y parcialmente en (S12) se llegue a la conclusión contraria, parece haber un consenso generalizado en que la realización de refactorizaciones trae consigo un aumento de la mantenibilidad del software. Sin embargo, en nuestra opinión una carencia recurrente de todos los trabajos estudiados es que las propuestas o afirmaciones que contienen no han sido validadas o respaldadas con datos cuantitativos reales provenientes de proyectos industriales reales, como, por ejemplo en [105]. El único trabajo que hace un esfuerzo en esta dirección es (S4), que realiza un estudio cuantitativo de los beneficios que tiene la refactorización en la mantenibilidad. No obstante, aunque el estudio realizado se considera correcto, su aportación es limitada, en tanto que no se realiza en un proyecto industrial, sino en un proyecto experimental.

Para realizar el estudio del efecto sobre la mantenibilidad en el software, los autores muestran soluciones muy limitadas. Así, proponen diferentes técnicas para identificar si existe una mejora de la mantenibilidad, como puede ser el uso de métricas extraídas directamente del código. La utilización de métricas para medir la mantenibilidad es una buena solución, pero esas métricas deben ser

adecuadas a un modelo de medición, como puede ser el propuesto en la norma ISO/IEC 25010, descrito en la sección 2.1.3. En caso de utilizar métricas que no se adapten a un modelo estándar, los resultados pueden ser engañosos y las conclusiones extraídas pueden ser erróneas. En resumen, aunque parece que la mayoría de los estudios apoyan la teoría de que las refactorizaciones mejoran la mantenibilidad del software, los trabajos estudiados adolecen de falta de evidencias reales que sirvan para respaldar esta afirmación.

En cuanto al valor aportado por una refactorización, aunque la mayoría de los estudios coinciden en la necesidad de conocer qué va a aportar una determinada refactorización antes de tomar la decisión de realizarla, sólo cinco de los artículos (S3, S7, S8, S10 y S13) intentan identificar categóricamente el ROI que la aplicación de una refactorización puede aportar. De hecho los artículos (S8, S10 y S13) hacen referencia a (S3 y S7) en ese sentido. Estos trabajos concluyen que para el cálculo del ROI hay que tener en cuenta dos factores: el coste de la refactorización y el ahorro en los costes de mantenimiento futuros que implica la refactorización. El principal problema de estos trabajos gira en torno a cómo se calcula el segundo de los factores. (S3 y S7) relacionan el ahorro en los costes de mantenimiento con el ahorro que se obtiene en los costes de las pruebas de regresión. Esta forma de calcular el impacto de la refactorización en los costes de mantenimiento es, cuanto menos, limitada, ya que una refactorización afecta al software en más puntos que en las pruebas de regresión. Por eso, aunque puede considerarse como una primera aproximación hacia el cálculo del ahorro de los costes de mantenimiento, es preciso continuar el trabajo en esta línea para identificar un mecanismo que permita evaluar de forma correcta y completa la disminución en los costes de mantenimiento que implica la refactorización.

En el caso de (S8, S10 y S13) solamente se aportan soluciones de alto nivel para mejoras de la arquitectura y no se tienen en cuenta, por ejemplo, diferentes cálculos dependiendo el tipo de refactorización u otro tipo de refactorizaciones.

Por último, la mayoría de los estudios sugieren que la mejor forma de realizar refactorizaciones es manualmente pero contando con la ayuda de alguna herramienta, esto es, realizando una refactorización asistida. Así, aunque todos los autores coinciden en que la refactorización manual resulta un proceso más lento y costoso, reconocen que es también el método más fiable, ya que la refactorización automática, aunque es factible, presenta todavía muchas deficiencias, como la posibilidad de que el código sea incluso más complicado de mantener [130]. Por lo tanto, la técnica más recomendada pasa por utilizar herramientas de soporte para identificar refactorizaciones candidatas, seleccionar manualmente las refactorizaciones a llevar a cabo y de nuevo utilizar herramientas para llevar a

cabo las refactorizaciones seleccionadas. Esta forma de proceder ayuda a aumentar la productividad drásticamente en la aplicación de refactorizaciones [21].

En síntesis se destacan los siguientes hallazgos:

- Existen opiniones opuestas (S1, S4, S12) sobre si las refactorizaciones traen consigo una mejora de la mantenibilidad del software.
- Ningún artículo habla de Ingeniería del Software Basada en Valor, por lo que puede concluirse que existe una carencia de conocimiento en este campo.
- No se cuantifica el valor que puede aportar una refactorización al software sobre el que va a ser aplicada.
- Es una práctica habitual en los estudios seleccionados limitar el número de métricas estudiadas, obviando los efectos que pueden tener sobre las métricas no consideradas. Sin embargo, esto se considera una limitación, ya que es complicado identificar si están estudiando la métrica adecuada. Por esto, aunque se considera una buena solución la utilización de métricas para saber en qué medida aumenta la mantenibilidad, es difícil encontrar las métricas adecuadas.
- La mayoría de los estudios sugieren que las refactorizaciones semiautomáticas o asistidas son las más adecuadas para llevar a cabo la refactorización.

2.6.13 Conclusiones

Los resultados muestran que la refactorización basada en valor es un campo en el que apenas se ha investigado. La mayoría de autores coinciden en que la mantenibilidad del software mejora con la aplicación de refactorizaciones, pero no son capaces de cuantificar esta mejoría. Se han realizado aproximaciones al cálculo de un ROI para la aplicación de una refactorización, pero siempre limitando el estudio hacia métricas concretas y no hacia la mejora de la mantenibilidad en general. De hecho, ningún artículo indica de manera objetiva y general los beneficios que aporta una refactorización al software. En cambio, se han encontrado varios artículos [94][95][96][97][98][101][105][106][131] en donde se detectan y priorizan refactorizaciones para maximizar la calidad del software, de acuerdo con distintos modelos de calidad compuestos principalmente por métricas del código fuente.

Lo que si se observa en los estudios es una tendencia a pensar que las refactorizaciones semiautomáticas o asistidas son las más apropiadas en estos

momentos para realizar una refactorización, ya que permiten automatizarla sin que su utilización sea demasiado compleja, y siguen basándose en la pericia del desarrollador a la hora de encontrar un artefacto susceptible de ser refactorizado. Así mismo se ha observado que una de las refactorizaciones más comunes es la Extracción de Método [96][97][103], o también conocido como *Extract Method*.

Para trabajos futuros, se propone la realización de un estudio que demuestre que las refactorizaciones conllevan un aumento de la mantenibilidad, indicando el valor que las diferentes refactorizaciones pueden aportar a la mantenibilidad del software. También se propone la creación de una herramienta que permita identificar y priorizar en función de su valor las posibles refactorizaciones que puedan realizarse sobre un determinado software. Por último, se propone también la realización de un estudio cuantitativo que demuestre, utilizando datos de proyectos reales en la industria, que la aplicación de refactorizaciones tiene un impacto positivo sobre la mantenibilidad del software.

2.7 Indicadores para la Mejora de Productos Software Basada en Valor

El proyecto KEMIS se centra en la mantenibilidad del producto software. Así mismo, implementa un soporte metodológico para la evaluación de la calidad del producto software capaz de calcular el valor y el retorno de inversión de las refactorizaciones necesarias para mejorar dicha calidad y resolver los problemas encontrados. En esta sección del estado del arte se recopilan los indicadores de valor encontrados en la literatura.

2.7.1 Indicadores de valor

Como primer paso se han recopilado los indicadores de valor encontrados en artículos donde se mida el valor de manera sistemática, en cualquier disciplina de la Ingeniería del Software. Se ha buscado fundamentalmente en cuatro fuentes:

- Los artículos de las diferentes ediciones de los workshops internacionales “Economics-Driven Software Engineering Research”.
- Los artículos de la conferencia “IEEE Computer Society Conference on Exploring Quantifiable Information Technology Yield” del año 2007.
- La revisión de la literatura de refactorización basada en valor realizada en la sección 2.6.

- Búsquedas de las palabras: “value” + “software” en las bases de datos digitales más representativas: ACM Digital Library, Springer Link, Science Direct, ProQuest Computing, IEEE Computer Society, IEEE Xplore, Current Content y Google Scholar.

En las siguientes tablas se resumen los indicadores de valor encontrados para priorizar tareas en distintas disciplinas de la Ingeniería del Software: gestión de requisitos, seguridad, trazabilidad, refactorización, mantenimiento y pruebas. En la primera columna se resumen las tareas que se priorizan en el artículo, en la segunda columna se indica la publicación. En la tercera columna se detallan los indicadores de valor encontrados, mientras que en la cuarta columna se presenta un posible indicador de valor para la priorización de las mejoras del software, análogo al indicador de valor de la columna anterior.

Por ejemplo, en [132] se priorizan implementaciones de requisitos en un desarrollo software. Para ello utiliza dos indicadores de valor: el coste de implementar el requisito y la importancia relativa del requisito. Realizando la analogía, tenemos dos potenciales indicadores para priorizar las mejoras de un producto software: el coste de implementar la mejora y la importancia relativa de la mejora respectos de otras mejoras. Este tipo de técnica ha sido utilizada en otros artículos, por ejemplo en [113][115][118] para inferir indicadores de valor a partir de indicadores económicos.

Tarea priorizable	**Ref.**	**Indicador de valor**	**Indicador de valor para priorizar la mejora**
Implementación de requerimientos	[132] [133]	Coste de implementar el requerimiento candidato	Coste de implementar la mejora candidata
		Importancia relativa del requerimiento, respecto de los otros requerimientos	Importancia relativa de la mejora, respecto de las otras mejoras
	[134]	Beneficio de implementar el requerimiento	Beneficio de implementar la mejora candidata
		Coste de implementar el requerimiento candidato	Coste de implementar la mejora candidata
	[135] [136]	Definido por los implicados.	Otros indicadores definidos por los implicados.

Tabla 2-11. Indicadores de valor asociados con la gestión de requisitos

Tarea priorizable	Ref.	Indicador de valor	Indicador de valor para priorizar la mejora
Evaluación de defectos potenciales	[137]	Costes de reparar los defectos	Coste de implementar la mejora candidata
		Costes de prevenir los defectos	Coste de implementar la mejora candidata
		Costes de buscar los defectos	Coste de encontrar las mejoras candidatas
		Costes de aislar y verificar los defectos	
Elección de inspecciones para detectar defectos en productos software	[138]	Beneficio esperado de encontrar un defecto en una clase	Beneficio de implementar la mejora candidata
		Defectos que se esperan encontrar en una clase	Mejoras que se pueden aplicar a un componente
		Coste indirecto de la inspección a realizar	Coste indirecto de encontrar las mejoras candidatas
		Coste directo de la inspección a realizar	Coste directo de implementar la mejora candidata
Inspecciones de Software	[74]	Importancia relativa de los componentes software.	Importancia relativa de la mejora, respecto de las otras mejoras.
Tareas de desarrollo conducido por test	[139]	Grado de eficiencia para remover defectos.	Grado de eficiencia para realizar las mejoras.
		Tiempo de desarrollo.	Tiempo de realización de la mejora.
		Tiempo de trabajo del desarrollador.	Tiempo de trabajo del desarrollador.

Tabla 2-12. Indicadores de valor asociados con la inspección de código

Tarea priorizable	Ref.	Indicador de valor	Indicador de valor para priorizar la mejora
Elección de tecnologías de seguridad	[140]	Coste de realizar la incorporación de la tecnología de seguridad	Coste de implementar la mejora candidata
		Facilidad de implementación de la tecnología de seguridad	Facilidad de implementación de la mejora
		Efectividad de la tecnología de seguridad	Efectividad de la mejora
Mejora en la seguridad de un sistema, desde el punto de vista de la disponibilidad	[140]	Pérdida anual esperada antes de realizar la mejora de seguridad en el sistema.	Pérdida anual esperada antes de realizar la mejora.
		Pérdida anual esperada después de realizar la mejora de seguridad en el sistema.	Pérdida anual esperada después de realizar la mejora.
		Coste de realizar la mejora de seguridad.	Coste de implementar la mejora candidata.
Inversión en tecnología de la información	[141]	Costes de entrenamiento al invertir en nueva tecnología.	Costes de entrenamiento para las mejoras de Interfaz de Usuario.
		Frecuencia de uso del Hardware o Software adquirido.	Frecuencia de uso de los componentes Software.

Tabla 2-13. Indicadores de valor asociados con la seguridad

Tarea priorizable	Ref.	Indicador de valor	Indicador de valor para priorizar la mejora
Actividades de refactorización	[111]	Mejora en métricas de acoplamiento, cohesión, tamaño y complejidad	Mejora de métricas de calidad
Actuaciones de mantenimiento	[107]	Importancia de la tarea de mantenimiento.	Importancia relativa de la mejora, respecto de las otras mejoras.

Tarea priorizable	Ref.	Indicador de valor	Indicador de valor para priorizar la mejora
		Urgencia de la tarea de mantenimiento.	Urgencia de realizar la mejora, respecto de las otras mejoras.
Actividades de refactorización	[24] [110]	Costes salvados de pruebas de regresión por realizar la refactorización.	Costes salvados de pruebas de regresión por realizar la refactorización.
		Costes de refactorización.	Coste de implementar la mejora candidata.
Actividades de refactorización: del diseño para mejorar la adaptabilidad de la arquitectura [113] y para mejorar la flexibilidad [115] [118].	[113] [115] [118]	Valor monetario actual del componente a ser modificado.	Valor monetario actual del componente a ser modificado.
		Valor monetario futuro del componente a ser modificado.	Valor monetario futuro del componente a ser modificado.
		Volatilidad del componente.	Volatilidad del componente.
		Coste de realizar el cambio.	Coste de implementar la mejora candidata.
		Tiempo para decidir la mejora a realizar.	Tiempo para decidir la mejora a realizar.
Estrategias de mejora del diseño	[143]	Coste de mantener el código sin mejorar el diseño.	Coste de mantener el código sin realizar tareas de mejora.
		Coste implementar el diseño candidato.	Coste de implementar la mejora candidata.
		Coste de mejorar el código a partir del diseño.	Coste de actualizar documentación y código fuente asociado.
		Probabilidad de que los artefactos de diseño cambien en el futuro.	Probabilidad de cambio de los componentes software.

Tarea priorizable	Ref.	Indicador de valor	Indicador de valor para priorizar la mejora
		Importancia relativa del artefacto de diseño, respecto de los otros artefactos de diseño.	Importancia relativa de la mejora, respecto de las otras mejoras.
Actividades de refactorización del diseño	[114] [116]	Mejora en métricas asociadas al diseño.	Mejora de métricas de calidad.

Tabla 2-14. Indicadores de valor asociados con la el mantenimiento y la refactorización

Tarea priorizable	Ref.	Indicador de valor	Indicador de valor para priorizar la mejora
Aplicación de trazabilidad	[75]	Probabilidad de cambio,	Probabilidad de cambio de los componentes Software
		Análisis de riesgos.	Análisis de riesgos de la mejora a realizar
		Capacidad para realizar la trazabilidad.	Disponibilidad para implementar la mejora.
	[77]	Importancia relativa de las tareas de trazabilidad.	Importancia relativa de la mejora, respecto de las otras mejoras.
	[78]	Importancia relativa de las tareas de trazabilidad.	Importancia relativa de la mejora, respecto de las otras mejoras.
		Análisis de riesgos.	Análisis de riesgos de la mejora a realizar.
		Coste de implementar la trazabilidad.	Coste de implementar la mejora candidata.

Tabla 2-15. Indicadores de valor asociados con la trazabilidad

Tarea priorizable	Ref.	Indicador de valor	Indicador de valor para priorizar la mejora
Revisiones	[79]	Coste de fallar en la revisión.	Coste de fallo en la realización de la mejora.

Tarea priorizable	Ref.	Indicador de valor	Indicador de valor para priorizar la mejora
		Importancia relativa de los componentes a revisar.	Importancia relativa de la mejora, respecto de las otras mejoras.
Pruebas funcionales	[81]	Importancia relativa del requerimiento al que está asociado la prueba.	Importancia relativa de la mejora, respecto de las otras mejoras.
		Complejidad del requisito al que está asociado la prueba.	Facilidad de implementación de la mejora.
		Volatilidad del requisito al que está asociado la prueba.	Probabilidad de Cambio de los componentes Software.
		Propensión al fallo de los componentes a ser probados.	Propensión al fallo de los componentes a ser mejorados.
	[144]	Definido por los implicados,	Otros indicadores definidos por los implicados.

Tabla 2-16. Indicadores de valor asociados con las pruebas

2.7.2 Indicadores de valor para priorizar una mejora en el producto software

Una vez interpretados y resumidos los diferentes atributos de valor en otras disciplinas se ha obtenido la lista de indicadores para la priorización de mejoras. Se ha analizado también su comportamiento respecto del valor de la mejora. Por ejemplo, la importancia relativa de una mejora tiene una relación directa respecto del valor: a mayor importancia relativa de una mejora, mayor es el valor que se obtiene al realizarla. Como resultado se tienen Tabla 2-17 y la Tabla 2-18. Se marca la columna M (mejora) si el indicador es afectado por el tipo de mejora, por ejemplo el coste de implementar una mejora no será el mismo si se trata de cambiar el nombre de una clase o si por el contrario se planifica romper una dependencia cíclica. La columna C, de forma análoga, señala si el indicador de valor es afectado por el tipo de componente, por ejemplo la importancia de un componente depende solamente del componente en sí y no de la mejora que se practique sobre el mismo.

Indicador de valor	M	C
Importancia relativa de la mejora, respecto de las otras mejoras	X	
Importancia relativa del componente, respecto de otros componentes.		X
Beneficio de implementar la mejora candidata	X	X
Cantidad de mejoras que se pueden aplicar a un componente	X	X
Facilidad de implementación de la mejora	X	
Capacidad de mejorar de las métricas de calidad	X	
Urgencia de realizar la mejora, respecto de las otras mejoras.	X	
Urgencia de mejorar el componente, respecto de otros componentes.		X
Rendimiento esperado de la mejora	X	
Disponibilidad para implementar la mejora	X	
Grado de eficiencia para realizar la mejora	X	X
Propensión al fallo de los componentes a ser mejorados		X
Pérdida anual esperada antes de realizar la mejora		X
Coste de mantener el código sin realizar tareas de mejora		X
Frecuencia de uso de los componentes Software.		X
Valor monetario futuro del componente a ser modificado.		X
Volatilidad del componente.		X

Tabla 2-17. Indicadores de valor directos para priorizar una mejora

Indicador de valor	M	C
Coste de implementar la mejora	X	X
Coste directo de encontrar las mejoras candidatas	X	X
Coste indirecto de encontrar las mejoras candidatas	X	X
Costes de entrenamiento para las mejoras de Interfaz de Usuario	X	X
Coste de actualizar documentación y código fuente asociado	X	X
Coste de actualizar las pruebas	X	
Coste de fallo en la realización de la mejora.	X	
Pérdida anual esperada después de realizar la mejora	X	
Valor monetario actual del componente a ser modificado		X
Tiempo para decidir la mejora a realizar	X	
Tiempo de realización de la mejora	X	
Análisis de riesgos de la mejora a realizar	X	
Tiempo de trabajo del desarrollador	X	
Importancia relativa de la mejora, respecto de las otras mejoras	X	
Importancia relativa del componente, respecto de otros componentes.		X
Beneficio de implementar la mejora candidata	X	X
Cantidad de mejoras que se pueden aplicar a un componente	X	X

Indicador de valor	M	C
Facilidad de implementación de la mejora	X	
Capacidad de mejorar de las métricas de calidad	X	
Urgencia de realizar la mejora, respecto de las otras mejoras.	X	
Urgencia de mejorar el componente, respecto de otros componentes.		X
Rendimiento esperado de la mejora	X	
Disponibilidad para implementar la mejora	X	
Grado de eficiencia para realizar la mejora	X	X
Propensión al fallo de los componentes a ser mejorados		X
Pérdida anual esperada antes de realizar la mejora		X
Coste de mantener el código sin realizar tareas de mejora		X
Frecuencia de uso de los componentes Software.		X
Valor monetario futuro del componente a ser modificado.		X
Volatilidad del componente.		X

Tabla 2-18. Indicadores de valor inversos para priorizar una mejora

2.7.3 Conclusiones

Al analizar diferentes modelos de medición para priorizar tareas en otras disciplinas se ha logrado el objetivo principal de esta sección, las listas de indicadores de valor para priorizar las mejoras del producto software. También se han identificado pautas para diseñar el modelo para medir el valor de una mejora:

- Es recomendable elegir una gran cantidad de indicadores de valor para realizar la priorización. Esto hace que cada indicador no tenga peso suficiente para sesgar la priorización, obteniendo mayor precisión.
- Existen diferentes niveles de abstracción en los indicadores de valor. La obtención de una medida abstracta final de valor deberá componerse de medidas de menor nivel en modelos de medición jerárquicos.
- Existe una gran variedad de dimensiones en los indicadores de valor: esfuerzo, uso del tiempo, grado de importancia, grado de urgencia, etc. Por lo que para diseñar un modelo de medición es necesario analizar detenidamente las funciones que deriven atributos más abstractos.
- Se han encontrado dos tipologías distintas de indicadores, los relacionados con el coste de realizar la mejora y los relacionados con el beneficio de realizar la mejora. Por ello es recomendable no perder de vista la fórmula genérica del ROI para encontrar el valor de una mejora (ver Figura 2-14).

$$ROI = \frac{Beneficio - Coste}{Coste}$$

Figura 2-14. Fórmula genérica del ROI

- Muchos de los indicadores están asociados con el juicio experto de los implicados en la mejora. Por lo tanto es necesario conocer los roles habituales implicados en las decisiones de mejora.

2.8 Visualización del Software

La visualización de la información es una tecnología emergente que intenta adoptar técnicas gráficas para la representación de entidades abstractas que no tienen formas concretas [152]. Este método de representación está probado como una solución efectiva para el manejo de una gran cantidad de datos.

La visualización del software (*Software Visualization*) es una adaptación de la visualización de la información aplicado al software. Engloba todo tipo de visualizaciones que puedan realizarse a partir de un software, desde la representación dinámica de un algoritmo a la tabulación del código [153]. Estas visualizaciones serán diferentes ya que tratan información sobre el software desde distintos puntos de vista. Como es lógico, la calidad del software también puede ser representada a través de visualizaciones, que deberán ser adecuadas para los datos que se deben representar.

Las visualizaciones del software pueden estar orientadas además a todo tipo de usuarios. Desde programadores hasta gerentes, todos pueden necesitar el uso de una visualización del software. Es por ello que se puede diferenciar entre diferentes tipos de visualizaciones para los mismos datos dependiendo de quién sea el usuario al que van dirigidas [153].

De este modo, uno de los principales problemas que presenta la visualización del software es el hecho de encontrar unos gráficos adecuados a los datos que serán representados y al usuario al que van dirigidos. En múltiples ocasiones las visualizaciones son tan simples y faltas de elementos visuales que no permiten al usuario interpretarlas correctamente [154]. Sin embargo, en otras ocasiones son tan complejas que tampoco tienen ningún valor real para el usuario [155].

A lo largo de la corta historia de la visualización del software, se han realizado estudios para resolver este problema. De estos estudios han surgido una

serie de propuestas de las características que debe tener una buena visualización, así como que diagramas son los más adecuados para mostrar determinados datos.

Algunas de estas características que debe tener una buena visualización son las siguientes:

- Las visualizaciones deben tener una correspondencia con el mundo real [156]. Es decir, los gráficos utilizados deben ser coherentes con los conceptos que están representando.
- Es recomendable el uso del color y, si es posible, de las tres dimensiones del espacio [152]. La utilización de estas dos características dentro de una visualización ayuda a mostrar más información. Sin embargo, el color debe ser elegido correctamente, ya que una selección errónea puede hacer que la visualización pierda su significado y pase a ser contraproducente [155] .
- Una buena visualización debe combinar una parte de presentación con una parte explorativa[157]. Esto implica que un buen gráfico debe mostrar datos y, al mismo tiempo, dar la posibilidad de descubrir otros a través de él.

Una visualización debe ser sensible a los cambios. Si dos gráficos iguales muestran datos con una pequeña diferencia entre ellos, esta diferencia debe ser apreciada en la visualización, correspondiendo siempre esta diferencia a la escala utilizada [158].

En cuanto a los diagramas de reconocido prestigio para representar ciertos datos, pueden identificarse los siguientes:

- Diagrama de Kiviat o gráfico de radar: es un gráfico que permite mostrar información multivariante en un gráfico de dos dimensiones. Representa los valores en los ejes, comenzando siempre desde el mismo punto de origen. Su utilización está muy extendida para representar la calidad de una entidad.
- Histogramas y gráficos de evolución: son representaciones que permiten visualizar de manera clara el histórico de un proyecto, así como su evolución a lo largo de un tiempo determinado.
- Gráficos dinámicos: este tipo de representaciones pueden ser utilizadas para representar cualquier tipo de información. Dado su dinamismo, ayudan al manejo de los datos que muestran. Gráficos de este tipo pueden ser desde una tabla que permite la ordenación, hasta gráficos que muestran dinámicamente la evolución de una entidad.

Teniendo todo esto en cuenta, es recomendable seleccionar correctamente las visualizaciones que van a utilizarse para mostrar los datos que se desean. Hacer buen uso de una visualización va a permitir que el usuario de la misma obtenga

información de una forma rápida, sencilla y eficaz. Sin embargo, una mala utilización de la misma puede provocar que se provoque una confusión mayor de la que ofrecen los datos antes de ser visualizados.

2.8.1 Algunas herramientas para la evaluación del software

Uno de los objetivos del proyecto KEMIS (ver sección 1.4.1.1) es construir una herramienta basada en software libre para la evaluación de los productos software.

En esta sección se analizarán las diferentes herramientas que se encuentran actualmente en el mercado. Son numerosos los trabajos en los que se ha tratado la clasificación de herramientas para la evaluación del software [145][150][151]. Para este trabajo, se seguirá la clasificación propuesta por [145] para dar una visión general de las diferentes herramientas existentes. Según esta clasificación existen dos tipos de herramientas:

Herramientas especializadas de métricas, las cuales:

- Están diseñadas para dar soporte a una función distinta de la medición, pero que, sin embargo, incorporan capacidad de medir, como son las herramientas de gestión de proyectos.
- Dan soporte a un tipo específico de métricas o grupo de métricas, como por ejemplo el tamaño del código.

Herramientas universales de métricas, diseñadas para dar soporte a las métricas que se centran en la extracción, el análisis y la realización de informes de los datos. Este tipo de herramientas deben proporcionar:

- Una interfaz de usuario flexible que permita la entrada de datos de una forma sencilla y a su vez facilite su integración con un entorno de automatización que asegure una toma de datos más exacta y eficiente.
- Dar soporte a una gran variedad de algoritmos, métricas, modelos de calidad o reglas.
- Capacidad para generar informes orientados a gráficos o a tablas, así como una serie de informes tipo que cubran las necesidades generales de las mediciones.
- Flexibilidad para poder modificar los informes y adecuarlos a las necesidades del usuario final.

2.8.2 XRadar

XRadar [146] es una herramienta de código abierto orientada a sistemas Java que integra un conjunto de herramientas de software libre de las que se obtienen las diferentes métricas.

Esta herramienta proporciona informes de los sistemas analizados en formato HTML ó SVG, en los cuales se puede apreciar el valor de cada uno de los campos analizados, siempre teniendo como referencia unos valores máximos y medios que existen en dicha herramienta por defecto. Estos informes son presentados al usuario mediante tablas y gráficos. Entre las métricas que representa XRadar pueden encontrarse las dependencias entre paquetes, el tamaño del código y su complejidad, las duplicaciones del mismo, entre otras.

En la Figura 2-15 se muestra un ejemplo de salida de XRadar.

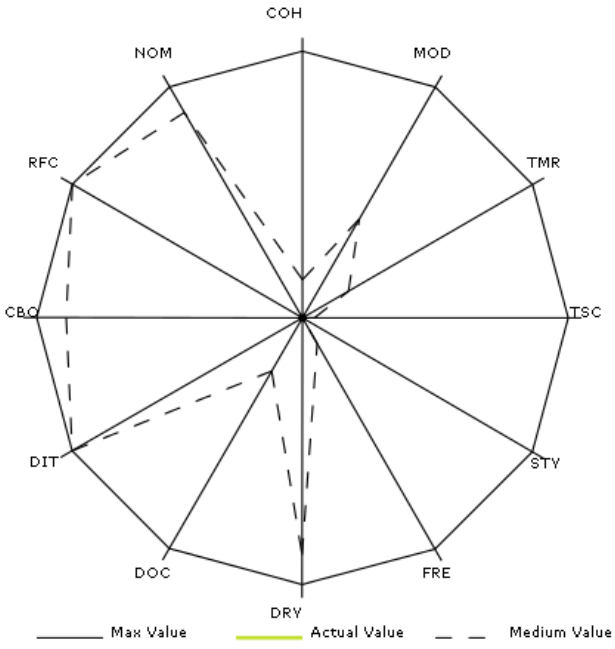


Figura 2-15. Ejemplo visualización mostrada por XRadar

Los problemas principales que posee esta herramienta es que, pese a que realiza la medición de los atributos base, no otorga valores a las características definidas por ninguna norma, como por ejemplo la norma ISO/IEC 9126.

En cuanto a la visualización, es intuitiva. Sin embargo, carece de datos que serían muy informativos. La utilización de diagramas de tipo radar está muy extendida, pero es una visualización muy simple y no es capaz de ofrecer demasiada información.

2.8.3 Sonar

Sonar [147] es una herramienta de control de la calidad distribuida bajo licencia GPL, cuyo propósito básico es unir las herramientas utilizadas para integración continua bajo un mismo entorno que permita medir la calidad del desarrollo.

Sonar recolecta los datos, los analiza y devuelve informes de los mismos, con lo que puede ser utilizada como herramienta para controlar la calidad del producto software. Esta herramienta proporciona sus propias reglas y permite crear otras nuevas. Además el informe proporcionado está clasificado según las características de la norma ISO/IEC 9126.

Un ejemplo de informe producido por Sonar es el que se muestra en la Figura 2-16. Como puede observarse el informe producido por Sonar es bastante completo, permitiendo tener una gran cantidad de datos a niveles muy bajos de profundidad.

El problema que presenta Sonar como herramienta es que su cuadro de mando muestra una gran cantidad de datos, impidiendo al usuario centrarse en los problemas más importantes detectados. Es una herramienta más centrada en el desarrollador, quien conoce el código fuente, que en el responsable de calidad del mismo. Así mismo, el modelo de calidad de referencia utilizado es el modelo SQALE [148], el cuál para el caso de la mantenibilidad solo identifica dos subcaracterísticas: la capacidad de ser leído y la capacidad de ser entendido el código fuente. Así mismo, no está probada la relación entre el modelo de calidad de referencia y las métricas seleccionadas en Sonar.

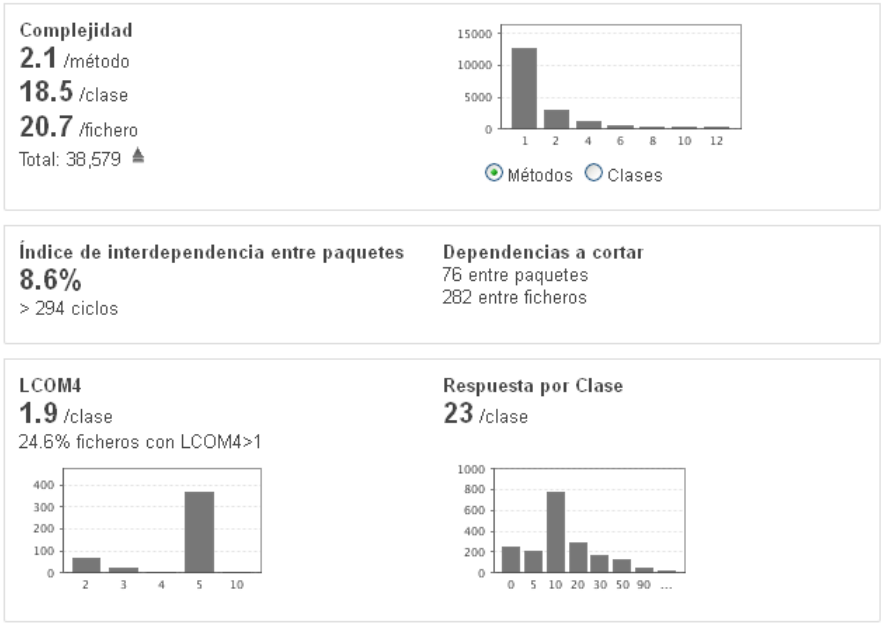


Figura 2-16. Ejemplo de informe producido por Sonar

2.8.4 Dashboard

Dashboard [149] es un plugin para herramientas de construcción software que permite centralizar en un informe los resultados obtenidos de otras herramientas de análisis estático de código fuente (por ejemplo, Cobertura, Surefire, Checkstyle, PMD, CPD, etc.).

En la Figura 2-17 puede verse el informe realizado por Dashboard para la ejecución de CheckStyle. Esta herramienta se limita a representar los informes extraídos por otras herramientas, haciéndolo además con unas visualizaciones bastante pobres. No representa las características de ningún modelo de referencia como puede ser la norma ISO/IEC 9126. En el caso de algunas herramientas carece de visualización, mostrando únicamente los datos obtenidos.

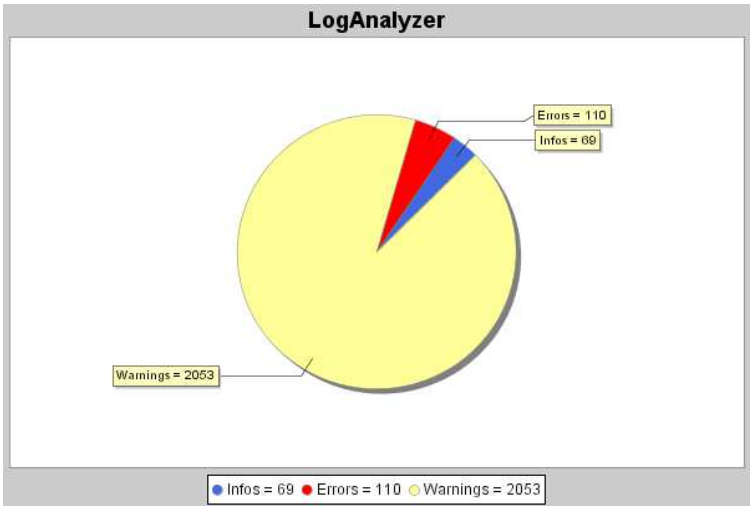


Figura 2-17. Representación que ofrece Dashboard luego de la ejecución de CheckStyle

2.8.5 Conclusiones sobre las herramientas de medición

Como hemos podido ver, existen una gran cantidad de herramientas que pueden utilizarse como entornos de medición. Todas ellas proponen una visualización de los datos obtenidos que intenta ser informativa. Sin embargo, todas estas herramientas analizadas presentan una serie de carencias, ya sea en las visualizaciones, en el modelo de calidad de referencia utilizado o en la relación entre el modelo de calidad y las métricas que obtiene (un resumen puede verse en la Tabla 2-19). El principal problema se encuentra en la relación entre las métricas básicas obtenidas de otras herramientas y su relación con un modelo de medición de calidad. Así mismo, todas las herramientas hacen uso de otros componentes de análisis estático del código fuente.

Ref.	Métricas	Gráficos	Modelo de Calidad	Relación métricas/modelo
[146]	Si	Si	No	No
[147]	Si	Si	SQALE	No
[149]	Si	Parcialmente	No	No

Tabla 2-19. Herramientas de medición de la mantenibilidad

El modelo de medición de la mantenibilidad

Uno de los objetivos de esta Tesis Doctoral es ofrecer indicadores de las características de calidad del producto que aporten visibilidad durante el desarrollo, la adquisición y el mantenimiento de un producto. Junto con ello, se busca proporcionar una infraestructura de medición automática basada en métricas obtenidas con herramientas de software libre, y un modelo de medición de calidad del producto software basado en el propuesto por la norma ISO/IEC 9126.

En este capítulo se presenta nuestra propuesta para un modelo de medición de la calidad y su especificación para la característica de mantenibilidad. Por un lado se presenta el marco metodológico y por otro lado el soporte tecnológico que se ha utilizado para el desarrollo de KEMIS, el entorno que implementa el modelo propuesto.

3.1 Soporte técnico: las herramientas para las mediciones básicas

Teniendo en cuenta las carencias observadas en la sección 2.2 al analizar los diferentes modelos de medición de la calidad y de la mantenibilidad, los cuales no cuentan con un soporte tecnológico y metodológico suficientes, se han revisado las herramientas existentes para el análisis del código fuente. El objeto de ello ha sido encontrar las mediciones básicas necesarias para construir un modelo de medición de la mantenibilidad.

En esta sección se han revisado las herramientas de software libre principales, con las que se logrará el soporte técnico para automatizar las medidas básicas de un modelo de medición de mantenibilidad.

3.1.1 La selección de las herramientas

Para la selección de estas herramientas se ha seguido una aproximación similar a la utilizada por [159], al realizar una revisión de los repositorios de herramientas libres existentes. Como primer paso se han seleccionado los repositorios a ser examinados. En el trabajo de Feitelson [160] se hace referencia a los repositorios existentes, como Codeplex, Google Code, Kenai o SourceForge. El último de ellos es el que más cantidad de proyectos contiene, con más de 314.000. Así mismo, presenta la mayor cantidad de actividad y de usuarios registrados con más de 2.000.000.

Como siguiente paso, se han identificado una serie de cadenas de búsqueda para encontrar las herramientas libres relacionadas con el objeto de la investigación. Las fuentes de esta cadena de búsqueda han sido el análisis de otros modelos de medición de la mantenibilidad, llevado a cabo en la sección 2.1.3 y el análisis de la parte 3 de la norma ISO/IEC 9126, donde se detallan las características internas del modelo de calidad. Los resultados pueden verse en la Tabla 3-1. Se han obtenido un conjunto de cadenas de búsqueda preliminares y definitivas (más simples) para abarcar mayor cantidad de búsquedas. La palabra "coverity" ha sido reemplazada por "coverage" al ser el término técnico correcto. Finalmente, estos términos han sido los utilizados para realizar la búsqueda en el repositorio.

Cadenas de búsqueda preliminares	Cadenas de búsqueda definitivas
Coverity	Coverage
Testing Tools	Test
Test Tools	
Static analysis tools	Static analysis
Static code analysis	
Static analysis	
Code analysis	Source Code Analysis
Source code analyzer	
Code analyzer	

Tabla 3-1. Cadenas de búsquedas utilizadas en SourceForge

A continuación, se ha filtrado la búsqueda a partir de los lenguajes de programación que podrían ser analizados por las herramientas de análisis del código fuente. Se han elegido los lenguajes de programación más populares de acuerdo con el índice TIOBE[2] (revisado por última vez en enero de 2011): VB.Net, C#, Java and C/C++. La Tabla 3-2 lista las principales herramientas encontradas en el repositorio SourceForge, de acuerdo con su actividad y cantidad de descargas (enero de 2011). Así mismo, se ha encontrado información adicional que refuerza la elección de algunas herramientas. Por ejemplo, las herramientas asociadas con las pruebas unitarias (también llamadas *xUnit*) están relacionadas con más de 140 proyectos; otras herramientas como FindBugs o PMD tienen más del 95% de comentarios positivos.

Para completar la selección anterior, se han incluido grupos de herramientas adicionales para los lenguajes relacionados con la tecnología .Net y

[2] http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

el análisis estático del código fuente y que no aparecían en SourceForge. Estas herramientas son FxCop y StyleCop [161]. Solo aclarar que actualmente estas herramientas están incluidas en los entornos de desarrollo .Net.

Herramienta	Actividad	Descargas	Cadena de búsqueda	Filtro
CheckStyle	99.94%	4,473,961	Source Code Analysis	JAVA
FindBugs	99.93%	606,868	Static Analysis	JAVA
PMD	99.38%	508,312	Source Code Analysis	JAVA
CCCC	92.76%	64,523	Source Code Analysis	C
JUnit	99.74%	2,900,703	Test	JAVA
NUnit	99.68%	1,908,561	Test	.NET
CPPUnit	98.97%	545,474	Test	C
EMMA	99.95%	1,487,428	Source Code Analysis Test	JAVA

Tabla 3-2. Herramientas encontradas

Las herramientas que miden el acoplamiento y la complejidad ciclomática son históricamente reconocidas para realizar la evaluación de la mantenibilidad, como se ha podido ver en la sección 2.1.3 y en gran cantidad de estudios, desde el año 1977 [47]. Por lo tanto, es necesario revisar las herramientas libres que extraen esas métricas básicas, aunque muchas de las herramientas indicadas en la Tabla 3-2 calculan estas métricas, no están enfocadas en analizar el acoplamiento y la complejidad ciclomática. En particular, se han elegido las herramientas JavaNCSS y JDepend.

Esta selección inicial ha sido reforzada con los resultados de un cuestionario respondido por jefes de departamento de calidad y desarrolladores de software.

3.1.2 Evaluación de la utilización de herramientas de análisis de código fuente en empresas españolas

Entre el año 2009 y 2010, en el marco de esta Tesis Doctoral, se ha realizado un estudio en base a las evaluaciones del proceso de desarrollo software llevadas a cabo por la empresa Kybele Consulting S.L (ver sección 1.4.1). La Figura 3-1 resume el tamaño de las más de 20 empresas evaluadas, donde el 30% eran medianas (entre 50 y 250 empleados), 55% eran pequeñas (entre 10 y 50 empleados) y 15% eran micro-empresas (hasta 10 empleados), de acuerdo con la "European Commission Recommendation 2003/361/EC".

El objetivo de las evaluaciones realizadas por Kybele Consulting ha sido la de analizar una instantánea del estado actual de los procesos de desarrollo software de la empresa, las prácticas seguidas y las herramientas utilizadas. Al final de la evaluación se les informaba a las empresas acerca de sus desviaciones, debilidades y fortalezas.

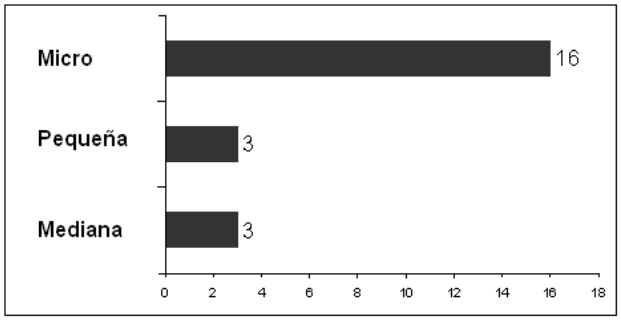


Figura 3-1. Tamaño de las empresas

Como parte de la evaluación de la empresa los integrantes del departamento de calidad y los desarrolladores software fueron entrevistados. Parte de las preguntas estaban relacionadas con la utilización de herramientas de análisis estático de código y pruebas unitarias. En la Figura 3-2 se puede ver el número de empresas que utilizan o planeaban utilizar alguna de estas herramientas.

Se han realizado las siguientes preguntas:

- P1: ¿Cuál de estas herramientas han sido utilizadas durante el último año?
- P2: ¿Cuál de estas herramientas serán utilizadas durante el siguiente año?

De acuerdo con los resultados obtenidos existen dos grandes tipos de herramientas utilizadas por las empresas evaluadas. Las primeras están asociadas con el análisis del código fuente y la búsqueda de problemas potenciales, como FxCop, PMD o FindBugs. En este sentido, las herramientas de análisis del estilo de codificación (que miden, por ejemplo, la cantidad de caracteres de una sentencia, o la cantidad de comentarios de una clase), no son ampliamente utilizadas. Esto puede ser debido a la gran cantidad de reglas por defecto que implementa este tipo de herramientas y al gran esfuerzo requerido para su personalización.

El segundo tipo de herramientas son las asociadas con las pruebas unitarias. Aunque esta práctica no esté completamente extendida en muchas empresas, el concepto del grado de cobertura de las pruebas si es tenido en cuenta.

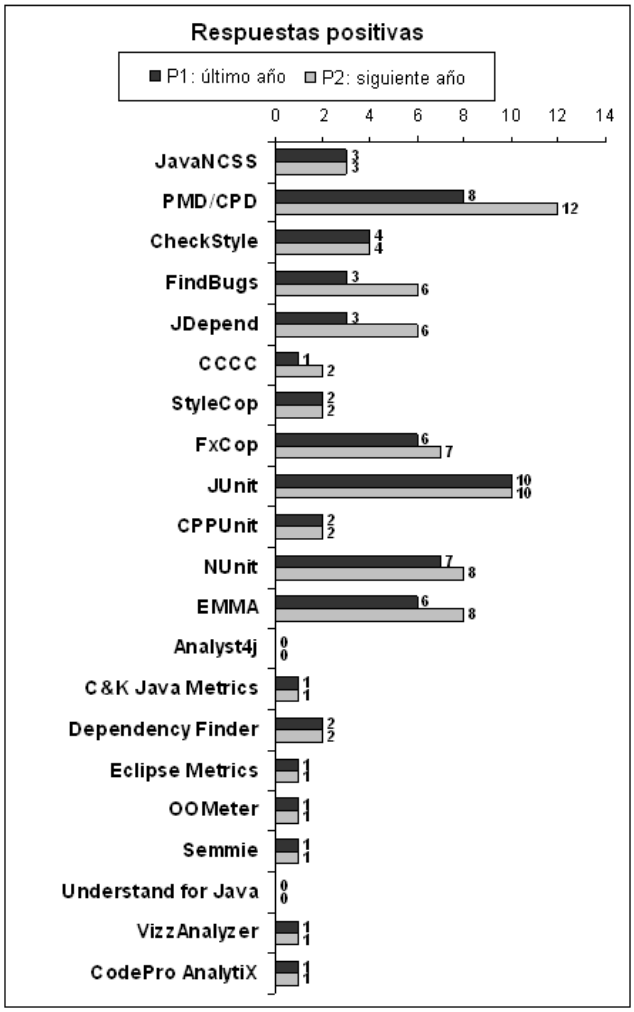


Figura 3-2. Herramientas utilizadas por las empresas

Finalmente la Tabla 3-3 enumera las principales herramientas libres de análisis del código fuente y evaluación de pruebas unitarias, de acuerdo a su popularidad y utilización en las empresas. En la primera y segunda columna se indican la referencia principal de la herramienta y su nombre respectivamente. Así mismo, se indican las métricas básicas que pueden obtenerse mediante estas herramientas. Un análisis posterior de las métricas y su utilización en el modelo de medición de la mantenibilidad se realizará en la sección 3.1.3.

Ref.	Herramienta	Métricas
[162]	JavaNCSS	Sentencias de código fuente no comentadas (NCSS)
		Complejidad ciclomática (CC)
[163]	PMD/CPD	Evaluación del código fuente (posibles errores, código no utilizado, código duplicado)
[164]	CheckStyle	Validación de estilos
[165]	FindBugs	Evaluación del código fuente (posibles errores)
[166]	JDepend	Dependencias cíclicas (CDC)
		Acoplamiento aferente, acoplamiento eferente (Ca, Ce)
[167]	CCCC	Número de módulos (NOM)
		Líneas de código (LOC)
		Líneas de comentarios (COM)
		Complejidad Ciclomática (MVG)
		Peso del método por clase (WMC)
		Fan-In, Fan-In concreteness, Fan-Out visibility, Fan-Out (…) (FI,FIc,FIv,FO,FOc,FOv)
		Líneas de código (LOC)
		Líneas de comentarios (COM)
[161]	StyleCop	Validación de estilos
[161]	FxCop	Evaluación del código fuente (posibles errores)
[168]	Junit	Número de pruebas unitarias (UT)
[169]	CPPUnit	Número de pruebas unitarias (UT)
[170]	Nunit	Número de pruebas unitarias (UT)
[171]	EMMA	Cobertura de pruebas unitarias por paquetes (CBP)
		Resumen de cobertura de pruebas unitarias (OCS)

Tabla 3-3. Selección final de herramientas y métricas básicas relacionadas

3.1.3 Relación de las herramientas libres de análisis estático de código respecto de la mantenibilidad

Como se ha mencionado anteriormente, en esta Tesis Doctoral nos hemos enfocado en revisar el grado de relación entre las herramientas seleccionadas y la mantenibilidad de acuerdo con su definición en la norma ISO/IEC 9126. Es decir, el grado de soporte técnico que ofrecen para la construcción de un modelo de medición de la mantenibilidad.

En esta sección se realiza un análisis de las métricas obtenidas por las herramientas indicadas en la Tabla 3-3 y el grado de relación con la mantenibilidad.

3.1.3.1 Java NCSS

La herramienta JavaNCSS soporta principalmente dos mediciones, NCSS y CC. De acuerdo con [92] ofrece un soporte parcial para las subcaracterísticas de analizabilidad y capacidad de ser probado y muy poco soporte a la estabilidad o cambiabilidad.

La métrica NCSS puede ser utilizada para evaluar la analizabilidad del código fuente debido a que es un buen indicador de la facilidad con la que el código fuente puede ser entendido y probado. Aún así, esta medida no se encuentra completamente alineada con estas subcaracterísticas. Por el contrario, el paradigma de desarrollo puede influenciar seriamente la analizabilidad y la capacidad de que el código fuente sea probado [173][174].

Así mismo, la relación entre la analizabilidad y la complejidad ciclomática (CC) es total. De hecho, el número de ciclos ha sido tradicionalmente aceptado como una medida para evaluar la complejidad de cualquier componente software. Por lo tanto, mientras menor sea el valor de la CC mayor será la analizabilidad del componente software [173][174]. En cuanto a la capacidad para ser probado, una CC mayor indica una mayor dificultad para realizar pruebas (mayor cantidad de caminos independientes que deben incluirse como caminos posibles) [175].

3.1.3.2 PMD/CPD

El PMD es una herramienta de análisis estático del código fuente Java que busca problemas potenciales, como, por ejemplo, bloques try/catch vacíos, código muerto o código duplicado (en este caso es el módulo CPD el encargado de realizar este análisis).

Se han analizado los diferentes grupos de reglas de la herramienta PMD y como se alinean con las subcaracterísticas de mantenibilidad de la ISO/IEC 9126. Entre un 5% y 10% de las reglas están directamente relacionadas con la cambiabilidad y capacidad de ser probado por lo que se puede considerar un emparejamiento parcial en relación con estas subcaracterísticas.

En particular, estudiando la documentación de PMD, se puede concluir que 16 de los 20 grupos de reglas están directamente relacionados con la claridad del código fuente y por lo tanto relacionados con la analizabilidad. Así mismo, tres de los grupos de reglas están directamente relacionados con la cambiabilidad (*migration*, *coupling* y *string*).

Finalmente, uno de los grupos está directamente relacionado con la característica de capacidad de ser probado, en especial con los problemas al realizar pruebas unitarias.

3.1.3.3 CheckStyle

El objeto principal de esta herramienta es ayudar al desarrollador a mejorar la calidad del estilo de codificación, automatizando un conjunto de validaciones. Por lo tanto, se puede concluir que Checkstyle se alinea a la subcaracterística de analizabilidad. Para confirmarlo se han revisado los grupos de reglas de la herramienta, encontrando que 13 de los 14 grupos están dirigidos a mejorar el análisis del código fuente.

3.1.3.4 FindBugs

FindBugs analiza el código fuente Java buscando patrones de errores bien conocidos, de manera similar que PMD. La herramienta identifica tres diferentes tipos de error: *correctness bugs*, problemas en el código fuente, *bad practices*, violaciones a buenas prácticas de codificación y *dodgies* o construcciones extrañas en el código fuente. En general, las tres categorías ayudan a mejorar la claridad del código fuente y por lo tanto podemos concluir que FindBugs está alineado con la analizabilidad.

3.1.3.5 JDepend

JDepend es otra herramienta de análisis estático de código que, entre otras métricas básicas, detecta el número de dependencias cíclicas (CDC) y el acoplamiento aferente (Ca) por agrupación de clases Java o también llamada *package*. Ambas medidas pueden ser utilizadas para medir la cambiabilidad del código fuente.

De hecho, de acuerdo con [176], las dependencias cíclicas disminuyen la habilidad de adaptación o modificación del sistema. Por lo tanto, a mayor CDC menor cambiabilidad. Así mismo, Ca indica el número de paquetes relacionados mediante llamadas entrantes (por ejemplo, al utilizar un método de una clase incluida en el paquete). Por lo tanto, a mayor Ca más complejo es el análisis necesario para entender la relación entre paquetes lo que dificulta la posibilidad de cambio.

3.1.3.6 CCCC

CCCC también es una herramienta de análisis estático de código que provee un conjunto de métricas similares a JavaNCSS o JDepend, a excepción de una métrica denominada NOM (Number of Modules) y WMC (Weighted Modules Complexity). En la Tabla 3-4 se comparan las métricas obtenidas por tres herramientas. Nótese que aunque el nombre de las métricas sea diferente el significado del valor proporcionado es el mismo.

JavaNCSS	JDepend	CCCC
NCSS	---	LOC, COM
CC	---	MVG
---	CDC; D	FO; FOc; Fov; FI; FIc; Fic

Tabla 3-4. Correspondencias entre métricas de JavaNCSS, JDepend y CCCC

Por lo tanto, lo descrito anteriormente para JavaNCSS y JDepend también es válido para CCCC. Es decir:

Las métricas LOC, COM y MVG obtenidas a partir de CCCC se encuentran ampliamente alineadas con la analizabilidad. Respecto de las pruebas, LOC y COM están parcialmente alineadas mientras que MVG se encuentra totalmente alineada. La correspondencia entre las métricas FO, FOc, Fov, FI, FIc y Fic soportadas por CCCC y las métricas CYC y D de JDepend permiten concluir que las métricas mencionadas de CCCC están alineadas con la cambiabilidad.

Por último, se analizarán las métricas para las que no se ha encontrado correspondencia. La métrica WMC se encuentra totalmente alineada con la analizabilidad al ser un buen indicador de la complejidad de cada modulo, de acuerdo con su descripción [167]. Así mismo, la métrica NOM está totalmente alineada con la cambiabilidad debido a que una gran cantidad de módulos implica una mayor complejidad al modificar el sistema.

3.1.3.7 StyleCop y FxCop

StyleCop es una herramienta para analizar el estilo del código fuente escrito en lenguaje .Net (C# y VB .Net). En cambio FxCop es una herramienta que analiza el código fuente en busca de patrones de errores bien conocidos, de manera similar a FindBug o PMD.

Por tanto, StyleCop y FxCop son herramientas similares a PMD y Checkstyle. Estudiando en profundidad el contenido de las reglas, se puede ver que estas herramientas están focalizadas en analizar buenas prácticas del código fuente lo cual mejora su analizabilidad.

3.1.3.8 Junit, CPPUnit, NUnit y EMMA

En esta sección se han agrupado las herramientas enfocadas a la realización de pruebas. En particular Junit, CPPUnit y NUnit están relacionadas con las pruebas unitarias mientras que EMMA abarca la cobertura de las pruebas unitarias (es decir cuánto código fuente es capaz de revisar el conjunto de pruebas unitarias).

Por tanto, al ser herramientas enfocadas en las pruebas estarán directamente relacionadas con la subcaracterística de “capacidad de ser probado”. Además, las pruebas unitarias siempre han estado relacionadas con la documentación del código fuente y con la evaluación de la estabilidad [61].

3.1.4 Resumen de las herramientas y su relación con las subcaracterísticas

En la Tabla 3-5 se resumen los emparejamientos detallados en la sección 3.1.3. Por lo tanto, se valoran las métricas como (T)otalmente, (A)mpliamente, (P)arcialmente o (N)o alineadas con la subcaracterística de mantenibilidad definida por la norma ISO/IEC 9126.

Herramienta	Métricas	Analizabilidad	Cambiabilidad	Estabilidad	Capacidad de ser probado
JavaNCSS	NCSS	A	N	N	A
	CC	A	N	N	T
PMD/CPD	Rules	T	P	N	P
CheckStyle	Checks	T	N	N	N
FindBugs	Bugs	T	N	N	N
JDepend	CYC	N	T	N	N
	D	N	T	N	N
CCCC	LOC	A	N	N	T
	MVG	A	N	N	T
	COM	A	N	N	N
	FO, FOc, Fov, FI, FIc, Fic	N	A	N	N
	NOM	N	A	N	N
	WMC	T	N	N	N
StyleCop	Rules	T	N	N	N
FxCop	Rules	T	N	N	N
Junit	UT	P	N	T	T
CPPUnit	UT	P	N	P	T
NUnit	UT	P	N	P	T
EMMA	UTLC	P	N	P	T
	UTBC			P	T

Tabla 3-5. Herramientas alineadas con la norma ISO/IEC 9126

En particular:

- La analizabilidad es la subcaracterística más soportada, especialmente en las herramientas que analizan el código fuente y detectan problemas de codificación. De hecho, en [66] se da un ejemplo de modelo de medición de la calidad software donde solo se tienen en cuenta este tipo de herramientas.
- La cambiabilidad puede ser computada a partir de herramientas que analizan las dependencias cíclicas, como JDepend o CCCC y por algunos grupos de reglas de PMD o FindBugs
- La estabilidad es la subcaracterística menos soportada, siendo sólo parcialmente soportada por las herramientas de pruebas unitarias.
- Finalmente, las herramientas para pruebas unitarias y cobertura de pruebas unitarias están directamente relacionadas con la medición de la capacidad de ser probado de un producto software.

Como conclusión final, se puede indicar que las herramientas de análisis estático pueden ser utilizadas para evaluar la calidad en términos del modelo de calidad del producto presentado por la norma ISO/IEC 9126. No obstante, la estructura jerárquica seguida por la norma implica la necesidad de construir mecanismos de derivación de métricas más abstractas para poder cuantificar las subcaracterísticas identificadas por el estándar. Como se comenta en la sección 2.2, no existe soporte técnico para automatizar la derivación de estas métricas, lo que hace necesaria la construcción de entornos más complejos.

Además, hay una falta de flexibilidad a la hora de utilizar estas herramientas, posibilidad de comunicación de los resultados o capacidad de persistencia de los resultados. Por todo esto, se antoja necesario construir un entorno tecnológico y metodológico para utilizar las métricas básicas obtenidas por estas herramientas. En la construcción de dicho entorno se deberá tener en cuenta:

- La reutilización de métricas básicas y reportes obtenidos por las herramientas anteriormente analizadas.
- La implementación de un modelo de medición jerárquico. Esto es, un modelo que permita obtener los valores numéricos de cada subcaracterística a partir de las métricas básicas.
- Y finalmente, tener la capacidad de incluir modelos de medición flexible con umbrales, indicadores o funciones configurables.

El objetivo es construir un modelo de medición de la calidad del producto software que sea personalizable y ampliable por las distintas organizaciones. Por

ejemplo, pueden ser asignados diferentes pesos para cada métrica básica en las funciones que deriven indicadores de mayor nivel de abstracción. De esta manera las organizaciones podrían personalizar el aporte final que realiza cada métrica al indicador.

3.1.5 Conclusion

Con las herramientas libres actuales de análisis estático de código se pueden obtener las métricas básicas necesarias para la medición de la mantenibilidad de un producto software. Una vez obtenidas estas métricas básicas es necesario seguir construyendo el modelo de medición de la mantenibilidad. Se irán componiendo dichas métricas en otras con mayor nivel de abstracción hasta llegar a las subcaracterísticas y características de calidad de acuerdo con la norma ISO/IEX 9126.

3.2 El Modelo de Medición de la Mantenibilidad

Para poder obtener valores indicadores de la calidad del producto software, es necesario establecer unos parámetros sobre los que realizar dicha medición. Para ello, el entorno metodológico de la herramienta KEMIS se basa en la norma ISO/IEC 9126 e ISO/IEC 25010, que establecen unos atributos a partir de los cuales se obtendrán valores indicadores de calidad.

Además, el modelo de referencia para la medición de la calidad del producto software de la norma ISO/IEC 25000 establece que las medidas de calidad software (*Software Quality Measures*) indican las características y subcaracterísticas de calidad del producto software, Figura 3-3.

El valor de estas medidas de calidad software se obtiene por la aplicación de una función de medida (*Measurement Function*) a los elementos de medida de calidad (*Quality Measure Elements*). Los elementos de medida de calidad pueden ser tanto medidas base como medidas derivadas obtenidas según describe el método de medición correspondiente (*Measurement Method*), de acuerdo a la norma ISO/IEC 15939 [67].

Aunque las normas ISO/IEC 9126 y 25000 establecen cuáles son las características de la calidad de un producto software y sus subcaracterísticas, no especifica qué medidas pueden utilizarse como indicador de una subcaracterística.

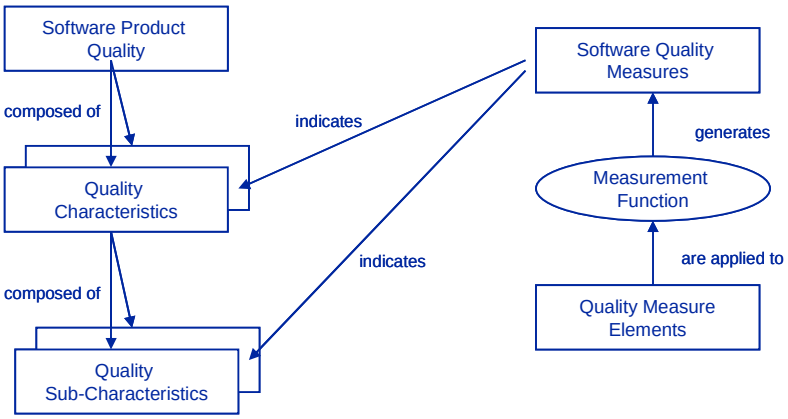


Figura 3-3. Modelo de referencia de medición de la calidad del producto software, según la norma ISO/IEC 25000

Uno de los aportes de esta Tesis Doctoral es el modelo de medición, basándose en la norma ISO/IEC 9126 e ISO/IEC 25010, para obtener valores indicadores de la calidad de un producto software. Por otra parte, el entorno KEMIS permitirá configurar los elementos del modelo de medición (medidas de calidad, funciones de medición, umbrales, etc.), adaptándolo a las características de cada organización o simplemente por la evolución natural del modelo de calidad. La Figura 3-4 muestra el modelo conceptual del modelo de medición:

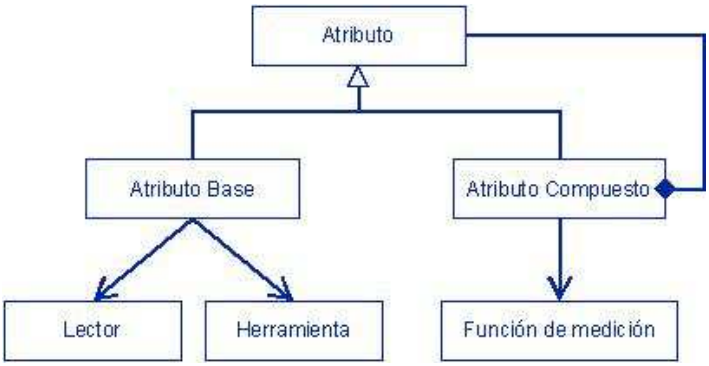


Figura 3-4. Modelo conceptual del modelo de medición de calidad del producto software

Así, un atributo representa una medida de calidad, esto es, una propiedad de calidad del propio producto software, del uso del producto software o de su proceso de desarrollo. Ejemplos de atributos son los siguientes:

- Atributos base: número de sentencias de código que no son comentario, complejidad ciclomática.
- Atributos compuestos: mantenibilidad, analizabilidad, densidad de la complejidad ciclomática

Obtener una medición significa, por tanto, obtener el valor de un atributo. Si se trata de un atributo base, su valor se obtendrá leyendo el informe proporcionado por la herramienta que lo midió, y si es compuesto se calculará a través de una función de medición.

3.3 Configuración del Modelo Medición de la Mantenibilidad

En esta sección se muestra una aplicación del modelo de calidad diseñado para el entorno KEMIS, esto es, una definición de los elementos configurables del modelo. Esta definición constituye el modelo de medición de la mantenibilidad, siendo el modelo por defecto del entorno KEMIS

El primero de los elementos configurables del modelo de medición son las medidas de calidad software a partir de las cuales se obtienen los valores indicadores de calidad de las características y subcaracterísticas. La selección de medidas de calidad para la característica de mantenibilidad se muestra en la Tabla 3-6.

Característica	Subcaracterística	Medida de calidad
Mantenibilidad	Analizabilidad	Densidad de complejidad ciclomática
		Densidad de código repetido
		Densidad de comentarios
		Densidad de defectos de la capacidad para ser analizado
	Cambiabilidad	Densidad de defectos de la capacidad para ser cambiado
		Densidad de ciclos
	Estabilidad	Densidad de defectos de estabilidad

Capacidad de ser probado	Densidad de defectos en pruebas unitarias
	Densidad de pruebas unitarias
	Cobertura de pruebas unitarias
	Tasa de fallos de pruebas unitarias
	Tasa de errores de pruebas unitarias

Tabla 3-6. Relación de medidas de calidad contempladas en KEMIS

A continuación se describen estas medidas de calidad, las funciones que permiten obtener sus valores y el escalado hacia valores indicadores para las subcaracterísticas y características.

3.3.1 Medidas de calidad

En este apartado se detallan las medidas de calidad utilizadas y su implementación en el entorno KEMIS. Recuérdese que la implementación de KEMIS tiene como objetivo la medición de desarrollos en el lenguaje de programación Java. Para otras implementaciones, como, por ejemplo, .Net o C solo es necesario modificar la configuración de la fuente de los datos (herramientas que analizarán el código fuente en el lenguaje Java, .Net o C).

La estructura de las definiciones que integran cada medida de acuerdo a la norma ISO/IEC 25020 [68] se indica en la Tabla 3-7.

Aspecto	Contenido
Nombre	Nombre asignado a la medida de calidad.
Característica	Característica/s y subcaracterística/s de calidad del modelo a la/s que hace referencia la medida de calidad.
Objetivo	Objetivo de la medida de calidad. Puede contener una pregunta a la que la medida de calidad dé respuesta.
Lenguaje	Los lenguajes de programación para los cuales la medida de calidad se encuentra disponible.
Criterio de decisión	Umbrales numéricos u objetivos para determinar la necesidad de realizar diversas acciones o investigaciones según los resultados de la medida de calidad.

Aspecto	Contenido
Visualización	Una descripción de cómo los resultados de las mediciones de calidad serán mostrados y/o comunicados a los usuarios.
Función de medición	Ecuación para el cálculo de la medida de calidad a partir de sus elementos.
Elementos de la medida de calidad	Nombre y definición de los elementos usados para calcular la medida de calidad.
Método de Medición	Descripción del método de medición de los elementos.
Fuentes de datos	Descripción de la/s fuente/s de datos de los elementos usados para calcular la medida de calidad.
Coste de obtención de los datos	Una declaración del coste de obtención de los datos a partir de las fuentes de datos.

Tabla 3-7. Estructura de las definiciones de medidas de calidad según las ISO/IEC 25020

El modelo de medición proporciona mecanismos para obtener mediciones de las características y subcaracterísticas a partir de los valores indicativos de las medidas de calidad, obteniendo así nuevas medidas de calidad.

Para ello, el modelo de medición define una serie de funciones para obtener el valor de las medidas de calidad normalizado, denominadas funciones de calidad (elemento configurable del modelo de medición de KEMIS). Esto es fundamental, ya que los valores de las medidas de calidad software que componen una subcaracterística pertenecen a distintos rangos numéricos. En el momento de componer la subcaracterística es necesario que sus valores pertenezcan a los mismos rangos, por ejemplo valores entre 0 y 100.

En este sentido, antes de pasar a componer una subcaracterística, se normaliza el valor de los atributos a través de su función de calidad. Por tanto, además de los elementos descritos anteriormente, el modelo de calidad define por cada una de las medidas de calidad una función de calidad. Estas funciones permiten normalizar los valores de las medidas de calidad (obteniendo un valor entre 0 y 100) en función de sus umbrales, obteniendo así nuevas medidas de calidad con las que poder operar para obtener otras medidas de calidad compuestas. Por lo tanto, a la tabla anterior se le añadirá un elemento más:

Aspecto	Contenido
Función de Calidad	Ecuación para la normalización de los valores de la medida de calidad en función de sus umbrales.

Los umbrales servirán para interpretar correctamente una medida y en este caso para normalizarla. Un umbral divide el espacio de valores que puede tomar una métrica en diferentes regiones, permitiendo la interpretación d los resultados. Por ejemplo, se puede indicar que un valor de complejidad ciclomática para que un método sea fácilmente analizable no puede ser mayor a 20. En ese caso todos los métodos con mediciones mayores a 20 se considerarán no analizables.

Pero, como indican Lanza y Marinescu [155], no existen los umbrales perfectos. Es necesario trabajar con los umbrales e ir perfeccionándolos mediante su uso. Existen principalmente dos fuentes para ayudar en la definición de los valores de los umbrales: la información estadística, basada en la recolección, resumen y análisis de datos; y los umbrales generalmente aceptados, que no están sustentados por un estudio estadístico pero son ampliamente reconocidos.

Para este modelo de medición se ha optado por los umbrales basados en información estadística. Siguiendo lo indicado en [155] se han analizado un conjunto de proyectos Java representativos por la cantidad de clases y la cantidad de versiones previstas (ver Tabla 3-8). Las medidas buscadas son:

- El valor típico: es decir el promedio de los valores analizados.
- El valor mayor y menor promedio.
- El valor mayor y menor extremo: un valor fuera de estos límites se considera fuera de rango.

Por último, con estas mediciones se han calculado los valores extremos de la siguiente manera:

- Valor mayor: promedio más desviación estándar.
- Valor menor: promedio menos desviación estándar.
- Valor mayor/menor extremo: 50% más/menos que el valor mayor

Proyecto	Version	Proyecto	Version
Apache MyFaces	1.2.6	Maven	2.0.9
Jboss WS-native	3.1.0	Openejb	3.1
Apache CXF	2.1.3	ow2-easybeans	1.0.2
Hibernate GA	3.3.1	Apache Struts	2.1.6
ibatis 2-core	2.3.4.726	Apache Log4j	1.2.15
Jasperreports	3.1.2	Apache openjpa	1.2.0
JetSpeed	2.1.3	Jetty	6.1.15

Tabla 3-8. Proyectos estudiados

Como ejemplo, en la Figura 3-5 se detalla el cálculo de los umbrales para la densidad de la complejidad ciclomática (una de las medidas derivadas) A continuación se describirán las diferentes medidas que componen el modelo de medición de la mantenibilidad, así como su implementación en el entorno KEMIS. Por cada medida se especifica también el valor de los umbrales, calculado de acuerdo al proceso que se acaba de describir.

Análisis estadístico para la obtención de umbrales recomendados. Densidad de Complejidad ciclomática

Proyecto	Versión	NCSS	CC	CC Densidad
Apache MyFaces	1.2.6	18360	7723	0,42
Jboss WS-native	3.1.0	51233	17504	0,34
Apache CXF	2.1.3	110909	37071	0,33
Hibernate GA	3.3.1	72527	26467	0,36
ibatis 2-core	2.3.4.726	10889	3972	0,36
Jasperreports	3.1.2	77485	25159	0,32
JetSpeed	2.1.3	75173	26411	0,35
Jetty	6.1.15	47245	18647	0,39
maven	2.0.9	13459	4735	0,35
openejb	3.1	117826	40195	0,34
ow2-easybeans	1.0.2	41304	11853	0,29
Apache Struts	2.1.6	28860	9138	0,32
Apache Log4j	1.2.15	11100	3678	0,33
Apache openjpa	1.2.0	107914	49056	0,45

	Métrica
	DenCC
Media de las densidades	0,36
Desviación std	0,04
Margen superior	0,40
Margen inferior	0,31
Valor muy alto	0,60
Valor muy bajo	0,16

Margen superior	media + desviacíon std
Margen inferior	media - desviacíon std
Valor muy alto	margen superor * 1,5
Valor muy bajo	margen superor * 0,5

Figura 3-5. Análisis estadístico de la densidad de complejidad ciclomática

3.3.2 Subcaracterísticas de la analizabilidad

A continuación se detallarán las métricas listadas en la Tabla 3-6 para la subcaracterística de analizabilidad. Para la descripción de cada métrica se utilizarán los atributos indicados en la Tabla 3-7. En todas las métricas derivadas se utilizarán funciones de densidad, calculando el ratio, entre, por ejemplo, el tamaño y la complejidad ciclomática. Esto favorece la independencia de las métricas respecto de los proyectos. Así, por ejemplo, un proyecto con 10.000 líneas de código fuente y una complejidad ciclomática acumulada de 5.000 tendrá menor densidad de complejidad ciclomática que un proyecto con 500 líneas de código y una complejidad ciclomática acumulada de 400.

Para la descripción de los umbrales se utilizarán cuatro variables:

- m: valor mayor
- n: valor menor
- u: valor mayor extremo
- v: valor menor extremo

Con estos valores se construye la función de calidad de cada métrica y un gráfico bajo cada tabla donde se ilustra la normalización obtenida de la función de calidad.

3.3.2.1 Densidad de complejidad ciclomática

La densidad de la complejidad ciclomática mide la relación entre la complejidad ciclomática de un producto y su tamaño (sin contar comentarios). De esta forma se puede obtener un indicador que es independiente del tamaño del proyecto. Esta medida es utilizada en otros trabajos, como, por ejemplo, [155][172][173].

Las aplicaciones con una mayor densidad de la complejidad ciclomática son más difíciles de entender y más difíciles de probar (tienen una mayor densidad de caminos de ejecución) y, por lo tanto, más difíciles de mantener.

Aspecto	Contenido
Nombre	Densidad de Complejidad Ciclomática

Aspecto	Contenido
Característica	Característica: • Calidad Interna – Mantenibilidad Subcaracterística: • Calidad Interna – Analizabilidad • Calidad Interna – Capacidad de ser probado
Objetivo	Evaluar la complejidad funcional del código para conocer la capacidad del mismo para ser analizado y el esfuerzo requerido para revisar dicho código. Pregunta: ¿Cómo de complejo es el código?
Lenguaje	Esta medida de calidad está disponible para: • Java. • C++.
Criterio de decisión	Umbrales recomendados[3]: m=0,16; n=0,31; u=0,40; v=0,60; Valores menores que m o mayores que v son considerados como mala calidad (nota 0 de 100). Valores mayores que n y menores que u son considerados como buena calidad (nota 10 de 100). El resto de valores serán valores intermedios (nota entre 0 y 100).
Visualización	El resultado de le medición de calidad se mostrará en el sitio Web del proyecto generado a partir de la herramienta KEMIS.
Función de medición	$$CCD = \frac{\sum CC}{NCSS} \cdot 100$$
Elementos de la medida de calidad	**CCD**: Densidad de complejidad ciclomática. **CC**: Complejidad ciclomática. **NCSS**: Sentencias de código que no son comentarios.
Método de Medición	**CC:** número de caminos linealmente independientes. **NCSS:** número de sentencias de código que no son comentarios.

[3] Obtenidos estadísticamente.

Aspecto	Contenido
Fuentes de datos	Informes de las siguientes herramientas: • JavaNCSS (Java). • CCCC (C++)
Coste de obtención de los datos	Bajo: estas medidas se obtienen automáticamente a partir de herramientas.
Escenarios de uso por cada uno de los roles	Esta medida de calidad puede ser usada por: • El equipo de desarrollo: o Los jefes de proyecto pueden conocer la calidad de los desarrollos de sus equipos, el diseño del software y el esfuerzo requerido para desarrollar las pruebas unitarias con el fin de validar el código. o Los desarrolladores pueden conocer la calidad de sus desarrollos y la cantidad de pruebas unitarias que deben desarrollar. • Los clientes del producto.
Función de calidad	$Q\_CCD = \begin{cases} 0 & , si \quad CCD \le m \\ \frac{CCD - m}{n - m} \cdot 100 & , si \quad m \le CCD \le n \\ 100 & , si \quad n \le CCD \le u \\ \frac{v - CCD}{v - u} \cdot 100 & , si \quad u \le CCD \le v \\ 0 & , si \quad CCD > v \end{cases}$

Tabla 3-9. Densidad de complejidad ciclomática

En la Figura 3-6 se ilustra la función de calidad de la métrica. En este caso si la densidad de complejidad ciclomática es mayor al valor máximo extremo (v) o menor al valor mínimo extremo el resultado es 0. Es 100 si la métrica se encuentra entre los valores máximo y mínimo promedio (n y u). Por último, para las

mediciones de densidad entre valores promedio y extremos se utiliza una función lineal.

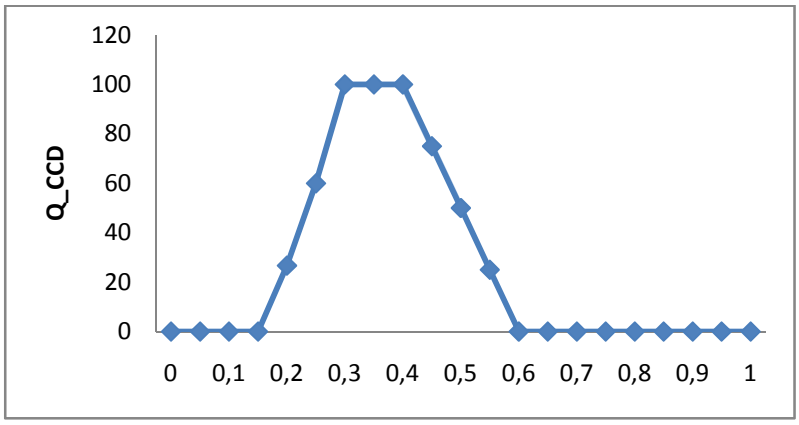


Figura 3-6. Función de densidad de complejidad ciclomática (Q_CCD)

3.3.2.2 Densidad de código repetido

La densidad de código repetido mide la relación entre la cantidad de código repetido de un producto y su tamaño (sin contar comentarios). De esta forma se puede obtener un indicador que es independiente del tamaño del proyecto.

A la hora de duplicar código, se duplican también los errores. Una densidad de código repetido alta, indica que la probabilidad de que un error esté repetido en varios sitios sea alta.

Aspecto	Contenido
Nombre	Densidad de Código Repetido
Características	Caracterísitca: • Calidad Interna – Mantenibilidad Subcaracterística: • Calidad Interna – Analizabilidad • Calidad Interna – Cambiabilidad • Calidad Interna – Estabilidad en las modificaciones

Aspecto	Contenido
Fase del ciclo de vida de la calidad de producto (objetivo de medición)	Calidad Interna (Producto en desarrollo/mantenimiento)
Objetivo	Evaluar la cantidad de código duplicado para conocer su capacidad para ser analizado, modificado y para evitar efectos inesperados a la hora de realizar una modificación. Pregunta: ¿Qué porcentaje de código se ha duplicado?
Lenguaje	Esta medida de calidad está disponible para: • Java • C++
Criterio de decisión	Umbrales recomendados: u=0,03; v=0,05; Valores mayores que v serán considerados como mala calidad (nota 0 de 100). Valores menores que u serán considerados como buena calidad (nota 100 de 100). El resto de valores serán valores intermedios (nota entre 0 y 100).
Visualización	El resultado de le medición de calidad se mostrará en el sitio Web del proyecto generado a partir de la herramienta KEMIS.
Función de medición	$DCR = \frac{DPL}{SGL} \cdot 100$
Elementos de la medida de calidad	**DCR:** Densidad de Código Duplicado **DPL:** Líneas de Código Duplicadas. **SGL:** Líneas de código significativas. Esta métrica se corresponde con la métrica NCSS. En lugar de usar NCSS se usa el número de líneas que proporciona la misma herramienta que cuenta el número de líneas duplicadas.

Aspecto	Contenido
Método de Medición	**DPL:** número de líneas de código duplicadas. **SGL:** número de líneas de código significativas.
Fuentes de datos	Informes de las siguientes herramientas: • CPD (Java) • Simian (Java / C++)
Coste de obtención de los datos	Bajo: estas medidas se obtienen automáticamente a partir de herramientas.
Escenarios de uso por cada uno de los roles	Esta medida de calidad puede ser usada por: • El equipo de desarrollo: o Los jefes de proyecto pueden conocer la calidad de los desarrollos de sus equipos, la calidad del diseño del software y el posible coste a la hora de introducir modificaciones en el código. o Los desarrolladores pueden conocer la calidad de sus desarrollos. • Los clientes del producto.
Función de calidad	$Q\_DCR) = \begin{cases} 100 & , si \quad DCR < u \\ \frac{v - DCR}{v - u} \cdot 100 & , si \quad u \le DCR \le v \\ 0 & , si \quad DCR > v \end{cases}$

Tabla 3-10. Densidad de código repetido

En la Figura 3-7 se grafica la función de calidad de la métrica. En este caso si la densidad de código repetido es mayor al valor máximo extremo (v) el resultado es 0. Es 100 si el ratio de código repetido es menor a un 3% (u:0,03). Por último, para las mediciones de densidad de código repetido entre los valores extremos se utiliza una función lineal.

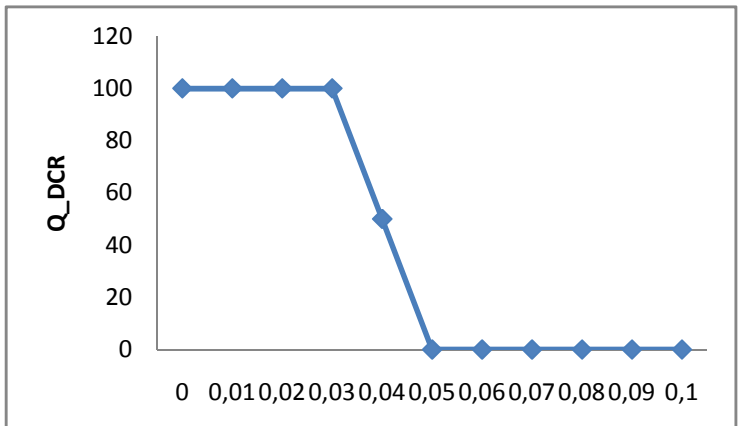


Figura 3-7. Función de densidad de código repetido (Q_DCR)

3.3.2.3 Densidad de comentarios

La densidad de comentarios mide la relación entre la cantidad de comentarios de un producto y su tamaño (obviamente, incluyendo comentarios). De esta forma se puede obtener un indicador que es independiente del tamaño del proyecto.

Aspecto	Contenido
Nombre	Densidad de Comentarios
Características	Característica: • Calidad Interna – Mantenibilidad Subcaracterística: • Calidad Interna – Capacidad para ser Analizado
Fase del ciclo de vida de la calidad de producto (objetivo de medición)	Calidad Interna (Producto en desarrollo/mantenimiento)
Objetivo	Evaluar los comentarios en el código para conocer su capacidad para ser analizado y modificado. Pregunta: ¿Qué porcentaje de comentarios posee el código?

Aspecto	Contenido
Lenguaje	Esta medida de calidad está disponible para: • Java • C++
Criterio de decisión	Umbrales recomendados: m = 0, n = 30, u = 90 y v = 100; Valores menores que m o mayores que v son considerados como mala calidad (nota 0 de 100). Valores mayores que n y menores que u son considerados como buena calidad (nota 100 de 100). El resto de valores serán valores intermedios (nota entre 0 y 100).
Visualización	El resultado de le medición de calidad se mostrará en el sitio Web del proyecto generado a partir de la herramienta KEMIS.
Función de medición	$$COMD = \frac{COML}{COML + NCSS} \cdot 100$$
Elementos de la medida de calidad	**COMD:** Densidad de Comentarios **COML:** Número de líneas de comentario y JavaDoc del código. **NCSS:** Non Commenting Source Statements, sentencias de código fuente que no son comentario
Método de Medición	**COML:** número de líneas de código que son comentario y JavaDoc del código. **NCSS:** número de sentencias de código que no son comentarios.
Fuentes de datos	Informes de las siguientes herramientas: • JavaNCSS (Java). • CCCC (C++).
Coste de obtención de los datos	Bajo: estas medidas se obtienen automáticamente a partir de herramientas.

Aspecto	Contenido
Escenarios de uso por cada uno de los roles	Esta medida de calidad puede ser usada por: • El equipo de desarrollo: o Los jefes de proyecto pueden conocer la calidad de desarrollo de sus equipos y el posible coste a la hora de introducir modificaciones en el código. o Los desarrolladores pueden conocer la calidad de sus desarrollos. • Los clientes del producto.
Función de calidad	$Q\_COMD = \begin{cases} 0 & , si \quad COMD \le m \\ \frac{COMD - m}{n - m} \cdot 100 & , si \quad m \le COMD \le n \\ 100 & , si \quad n \le COMD \le u \\ \frac{v - COMD}{v - u} \cdot 100 & , si \quad u \le COMD \le v \\ 0 & , si \quad COMD > v \end{cases}$

Tabla 3-11. Densidad de comentarios

En la Figura 3-8 se ilustra la función de calidad de la métrica. En este caso si la densidad de comentarios es mayor al valor máximo extremo (v) o menor al valor mínimo extremo el resultado es 0. Es 100 si la métrica se encuentra entre los valores máximo y mínimo promedio (n y u). Por último, para las mediciones de densidad entre valores promedio y extremos se utiliza una función lineal.

Analizando los umbrales presentados, la función de calidad devolverá un valor de 100 si el porcentaje de comentarios va desde el 30% hasta el 90%

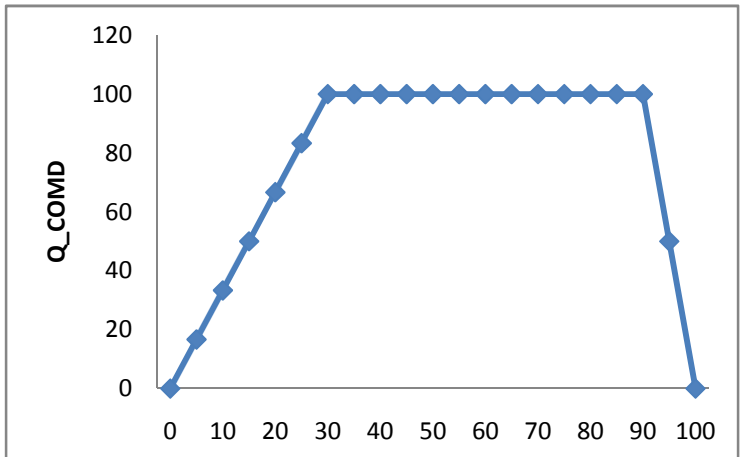


Figura 3-8. Función de densidad de comentarios (Q_COMD)

3.3.2.4 Densidad de defectos de la capacidad de ser analizado

La densidad de defectos de la capacidad de ser analizado mide la relación entre el número de defectos de analizabilidad (personalizado según los estándares de cada organismo) y el tamaño del proyecto. Este indicador nos muestra lo complejo que es el software a la hora de ser analizado. Se calcula la densidad de violaciones al realizar un análisis estático del código fuente. En este caso se utiliza la herramienta PMD comentada en la sección 3.1.3.2.

ASPECTO	CONTENIDO
Nombre	Densidad de Defectos de la Capacidad de ser Analizado
Característcas	Característica: • Calidad Interna – Mantenibilidad Subcaracterística: • Calidad Interna – Capacidad para ser Analizado
Fase del ciclo de vida de la calidad de producto (objetivo de medición)	Calidad Interna (Producto en desarrollo/mantenimiento)

ASPECTO	CONTENIDO
Objetivos	Evaluar lo defectos de la capacidad del código para ser analizado. Pregunta: ¿Cuál es el porcentaje de defectos encontrados de la capacidad del código para ser analizado?
Lenguaje	Esta medida de calidad está disponible para: • Java • C++
Criterio de decisión	Umbrales recomendados: u = 20 y v = 100; Valores menores que u serán considerados como buena calidad (nota 100 de 100). Valores mayores que v serán considerados como mala calidad (nota 0 de 100). El resto de valores serán valores intermedios (nota entre 0 y 100).
Visualización	El resultado de le medición de calidad se mostrará en el sitio Web del proyecto generado a partir de la herramienta KEMIS.
Función de medición	$ANALDEFD = \frac{\sum ANALDEFD}{NCSS} \cdot 100$
Elementos de la medida de calidad	**ANALDEFD:** Densidad de Defectos de la Capacidad para ser Analizado. **ANALDEF:** Defectos de la Capacidad para ser Analizado. Violaciones en reglas al realizar el análisis estático del código fuente. **NCSS:** Non Commenting Source Statements, sentencias de código fuente que no son comentario
Método de Medición	**ANALDEF:** número de defectos de la capacidad para ser analizado. **NCSS:** número de sentencias de código que no son comentarios.

ASPECTO	CONTENIDO
Fuentes de datos	Informes de las siguientes herramientas: • JavaNCSS (Java) • PMD (Java) • FindBugs (Java) • CheckStyle (Java) • CCCC (C++)
Coste de obtención de los datos	Bajo: estas medidas se obtienen automáticamente a partir de herramientas.
Escenarios de uso por cada uno de los roles	Esta medida de calidad puede ser usada por: • El equipo de desarrollo: o Los jefes de proyecto pueden conocer la calidad de desarrollo de sus equipos. o Los desarrolladores pueden conocer la calidad de sus desarrollos. • Los clientes del producto.
Función de calidad	$Q\_ANALDEFD = \begin{cases} 100 & ,si \quad ANALDEFD < u \\ \frac{v - ANALDEFD}{v - u} \cdot 100 & ,si \quad u \le ANALDEFD \le v \\ 0 & ,si \quad ANALDEFD > v \end{cases}$

Tabla 3-12. Densidad de defectos de la capacidad de ser analizado

En la Figura 3-9 se grafica la función de calidad de la métrica. En este caso si la densidad de defectos es mayor al valor máximo extremo (v) el resultado es 0. Es 100 si el ratio de defectos y líneas de código (en este caso obtenidos mediante la herramienta PMD) es menor a un 20%. Por último, para las mediciones de densidad de defectos entre los valores extremos se utiliza una función lineal.

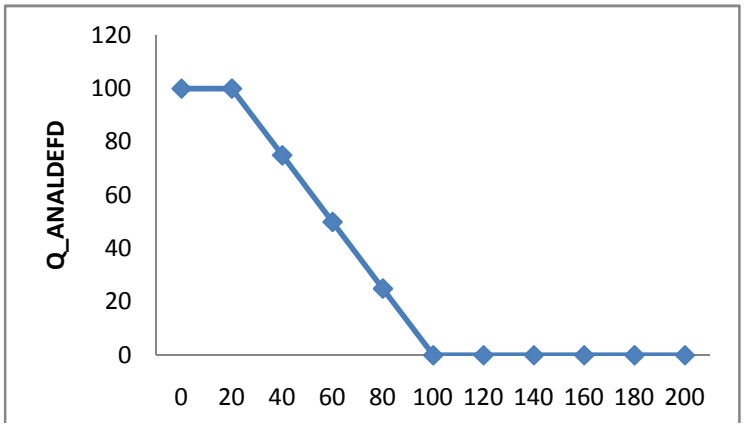


Figura 3-9. Función de densidad de defectos de la capacidad de ser analizado (Q_ANALDEFD).

3.3.3 Subcarácterísticas de la cambiabilidad

Al igual que lo realizado con la subcaracterística de analizabilidad en este apartado se detallarán las métricas listadas en la Tabla 3-6 para la subcaracterística de analizabilidad. Para la descripción de cada métrica se utilizarán nuevamente los atributos indicados en la Tabla 3-7 y funciones de densidad.

3.3.3.1 Densidad de defectos de la capacidad de ser cambiado

La densidad de defectos de la capacidad de ser cambiado un producto se mide como la relación entre el número de defectos de cambiabilidad que presenta el producto y su tamaño. Los defectos de cambiabilidad están relacionados con las herramientas de análisis estático de código comentadas en el apartado 3.1.4.

Aspecto	Contenido
Nombre	Densidad de Defectos de la Capacidad para ser Cambiado
Características	Característica: • Calidad Interna – Mantenibilidad Subcaracterística: • Calidad Interna – Capacidad para ser Cambiado

Aspecto	Contenido
Fase del ciclo de vida de la calidad de producto (objetivo de medición)	Calidad Interna (Producto en desarrollo/mantenimiento)
Objetivos	Evaluar lo defectos de la capacidad del código para ser cambiado. Pregunta: ¿Cuál es el porcentaje de defectos encontrados de la capacidad del código para ser cambiado?
Lenguaje	Esta medida de calidad está disponible para: • Java • C++
Criterio de decisión	Umbrales recomendados: u = 20 y v = 50; Valores menores que u serán considerados como buena calidad (nota 100 de 100). Valores mayores que v serán considerados como mala calidad (nota 0 de 100). El resto de valores serán valores intermedios (nota entre 0 y 100).
Visualización	El resultado de le medición de calidad se mostrará en el sitio Web del proyecto generado a partir de la herramienta KEMIS.
Función de medición	$CHGDEFD = \frac{\sum CHGDEF}{NCSS} \cdot 100$
Elementos de la medida de calidad	**CHGDEFD:** Densidad de defectos de cambiabilidad **CHGDEF:** Defectos de cambiabilidad. **NCSS:** Non Commenting Source Statements, sentencias de código fuente que no son comentario
Método de Medición	**CHGDEF:** número de defectos de la capacidad para ser cambiado. **NCSS:** número de sentencias de código que no son comentarios.

Aspecto	Contenido
Fuentes de datos	Informes de las siguientes herramientas: • PMD (Java) • FindBugs(Java) • CCCC(C++) • JDepend(Java)
Coste de obtención de los datos	Bajo: estas medidas se obtienen automáticamente a partir de herramientas.
Escenarios de uso por cada uno de los roles	Esta medida de calidad puede ser usada por: • El equipo de desarrollo: o Los jefes de proyecto pueden conocer la calidad de desarrollo de sus equipos. o Los desarrolladores pueden conocer la calidad de sus desarrollos. • Los clientes del producto.
Función de calidad	$Q\_CHGDEFD = \begin{cases} 100 & , si \quad CHGDEFD < u \\ \frac{v - CHGDEFD}{v - u} \cdot 100 & , si \quad u \le CHGDEFD \le v \\ 0 & , si \quad CHGDEFD > v \end{cases}$

Tabla 3-13. Densidad de defectos de la capacidad de ser cambiado

En la Figura 3-10 se grafica la función de calidad de la métrica. En este caso si la densidad de defectos asociados con la cambiabilidad es mayor al valor máximo extremo (v) el resultado es 0. Es 100 si el ratio de defectos y líneas de código (en este caso obtenidos mediante la herramienta PMD o FindBugs) es menor a un 20% (u:20). Por último, para las mediciones de densidad de defectos entre los valores extremos se utiliza una función lineal.

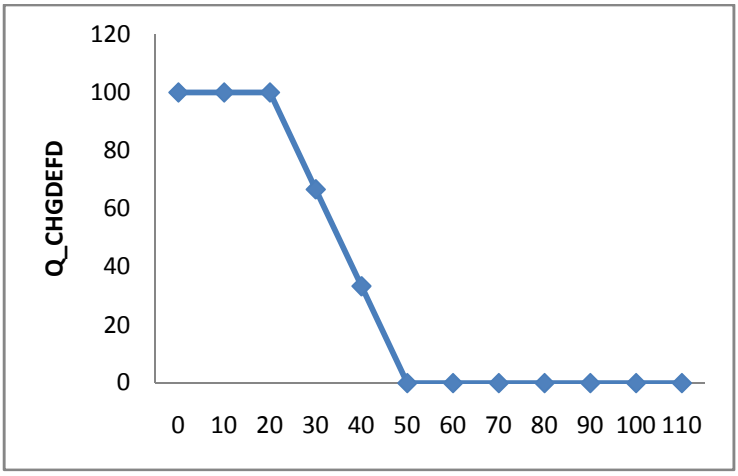


Figura 3-10. Función de densidad de defectos de cambiabilidad (Q_CHGDEFD)

3.3.3.2 Densidad de ciclos

La densidad de ciclos de un producto mide la relación entre el número de dependencias cíclicas que presenta el producto y la cantidad de paquetes.

Aspecto	Contenido
Nombre	Densidad de Ciclos
Características	Característica: • Calidad Interna – Mantenibilidad Subcaracterística: • Calidad Interna – Cambiabilidad • Calidad Interna – Reutilización • Calidad Interna – Capacidad de ser probado
Fase del ciclo de vida de la calidad de producto (objetivo de medición)	Calidad Interna (Producto en desarrollo/mantenimiento)

Aspecto	Contenido
Objetivo	Evaluar la existencia de ciclos entre paquetes para conocer las dependencias que pueden traer problemas entre dichos paquetes. Pregunta: ¿Existen ciclos entre paquetes en el software?
Lenguaje	Esta medida de calidad está disponible para: • Java • C++
Criterio de decisión	Umbrales recomendados: u=0; u=0 es el valor óptimo. No deben existir ciclos. El resto de valores serán considerados de baja calidad (nota 0 de 100).
Visualización	El resultado de le medición de calidad se mostrará en el sitio Web del proyecto generado a partir de la herramienta KEMIS.
Función de medición	$CYCD = \frac{\sum CYC}{PACK} \cdot 100$
Elementos de la medida de calidad	**CYCD:** Densidad de Ciclos. **CYC:** Ciclos entre paquetes. **PACK:** Paquetes del sistema.
Método de Medición	**CYC:** número de ciclos entre paquetes. **PACK:** número de paquetes del sistema
Fuentes de datos	Informes de las siguientes herramientas: • JDepend (Java) • CCCC (C++)
Coste de obtención de los datos	Bajo: estas medidas se obtienen automáticamente a partir de herramientas.

Aspecto	Contenido
Escenarios de uso por cada uno de los roles	Esta medida de calidad puede ser usada por: • El equipo de desarrollo: o Los jefes de proyecto pueden conocer la calidad de desarrollo de sus equipos, el esfuerzo requerido a la hora de desarrollar un conjunto fiable de pruebas unitarias y el coste de introducir una modificación en el código. o Los desarrolladores para conocer la calidad de sus desarrollos. • Los clientes del producto.
Función de Calidad	$Q\_CYCD = \begin{cases} 100 & , si \quad CYCD = 0 \\ 0 & , si \quad CYCD > 0 \end{cases}$

Tabla 3-14. Densidad de ciclos

En la Figura 3-11 se grafica la función de calidad de la métrica. En este caso si la densidad de cicloses igual o mayor a 1 el resultado es 0. Y es 100 si la cantidad de ciclos es menor que la cantidad de paquetes que contiene el proyecto.

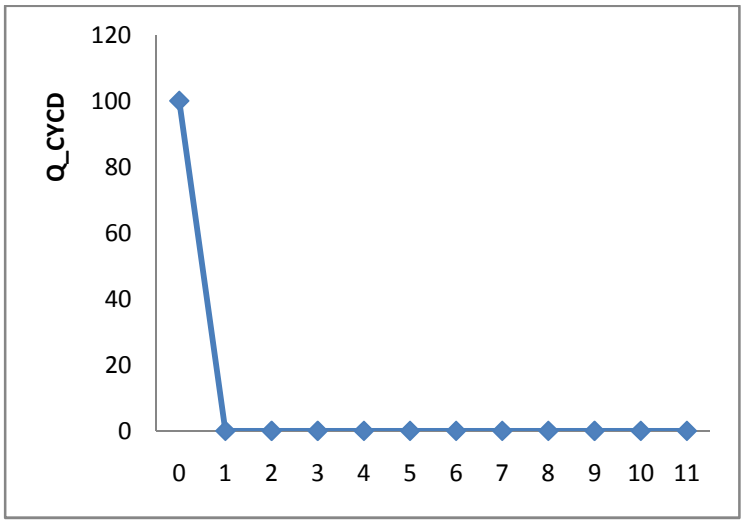


Figura 3-11. Función de densidad de ciclos (Q_CYCD)

3.3.4 Subcaracterísticas de la estabilidad

Al igual que lo realizado con la subcaracterística de analizabilidad y cambiabilidad en este apartado se detallarán las métricas listadas en la Tabla 3-6 para la subcaracterística de estabilidad. Para la descripción de cada métrica se utilizarán nuevamente los atributos indicados en la Tabla 3-7 y funciones de densidad.

3.3.4.1 Densidad de defectos de estabilidad

La densidad de defectos de estabilidad mide la relación entre el número de defectos de estabilidad que presenta el producto y su tamaño. Los defectos de estabilidad están relacionados con las herramientas de análisis estático de código comentadas en el apartado 3.1.4.

Aspecto	Contenido
Nombre	Densidad de Defectos de la Estabilidad
Características	Característica: • Calidad Interna – Mantenibilidad Subcaracterística: • Calidad Interna – Estabilidad
Fase del ciclo de vida de la calidad de producto (objetivo de medición)	Calidad Interna (Producto en desarrollo/mantenimiento)
Objetivos	Evaluar lo defectos de la estabilidad del código. Pregunta: ¿Cuál es el porcentaje de defectos encontrados en la estabilidad del código?
Lenguaje	Esta medida de calidad está disponible para: • Java • C++

Aspecto	Contenido
Criterio de decisión	Umbrales recomendados: u = 20 y v = 50; Valores menores que u serán considerados como buena calidad (nota 100 de 100). Valores mayores que v serán considerados como mala calidad (nota 0 de 100). El resto de valores serán valores intermedios (nota entre 0 y 100).
Visualización	El resultado de le medición de calidad se mostrará en el sitio Web del proyecto generado a partir de la herramienta KEMIS.
Función de medición	$MOSTDEFD = \frac{\sum MOSTDEF}{NCSS} \cdot 100$
Elementos de la medida de calidad	**MOSTDEFD:** Densidad de Defectos de Estabilidad **MOSTDEF:** Defectos de Estabilidad. **NCSS:** Non Commenting Source Statements, sentencias de código fuente que no son comentario
Método de Medición	**MOSTDEF:** número de defectos de estabilidad. **NCSS:** número de sentencias de código que no son comentarios.
Fuentes de datos	Informes de las siguientes herramientas: • PMD • FindBugs • CheckStyle
Coste de obtención de los datos	Bajo: estas medidas se obtienen automáticamente a partir de herramientas.

Escenarios de uso por cada uno de los roles	Esta medida de calidad puede ser usada por:
	• El equipo de desarrollo:
	  o Los jefes de proyecto pueden conocer la calidad de desarrollo de sus equipos.
	  o Los desarrolladores pueden conocer la calidad de sus desarrollos.
	• Los clientes del producto.

Aspecto	Contenido
Función de calidad	$Q\_MOSTDEFD = \begin{cases} 100 & ,si \quad MOSTDEFD < u \\ \frac{v - MOSTDEFD}{v - u} \cdot 100 & ,si \quad u \leq MOSTDEFD \leq v \\ 0 & ,si \quad MOSTDEFD > v \end{cases}$

Tabla 3-15. Densidad de defectos de estabilidad

En la Figura 3-12 se grafica la función de calidad de la métrica. En este caso si la densidad de defectos asociados con la estabilidad es mayor al valor máximo extremo (v) el resultado es 0. Es 100 si el ratio de defectos y líneas de código (en este caso obtenidos mediante la herramienta PMD o FindBugs) es menor a un 20% (u:20). Por último, para las mediciones de densidad de defectos entre los valores extremos se utiliza una función lineal.

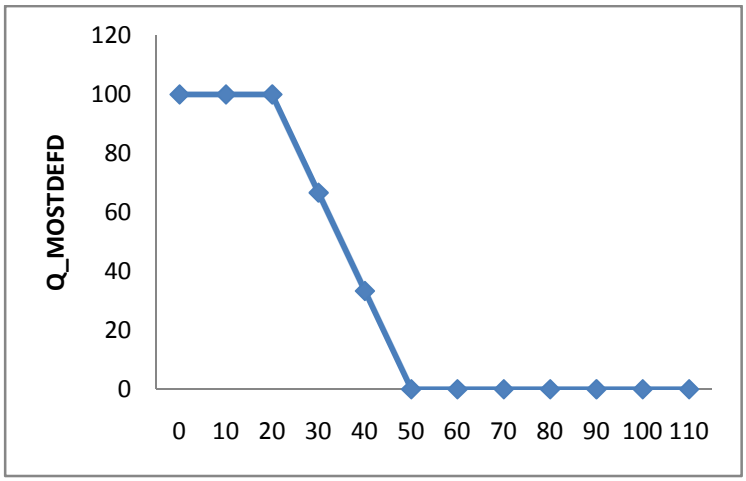


Figura 3-12. Función de densidad de defectos de estabilidad (Q_MOSTDEFD)

3.3.5 Subcaracterísticas de la capacidad de ser probado

Al igual que lo realizado con las otras subcaracterísticas en este apartado se detallarán las métricas listadas en la Tabla 3-6 para la subcaracterística de la capacidad de ser probado. Para la descripción de cada métrica se utilizarán nuevamente los atributos indicados en la Tabla 3-7 y funciones de densidad.

3.3.5.1 Densidad de pruebas unitarias

A la hora de hacer pruebas unitarias, hay que prestar especial atención a la complejidad ciclomática. Las pruebas unitarias deben recorrer todos los caminos ejecutables definidos en el código fuente. En este sentido, sería deseable que el número de pruebas unitarias fuese aproximadamente igual a la complejidad ciclomática, es decir una prueba por cada camino.

La densidad de pruebas unitarias mide la relación entre el número de pruebas unitarias y la complejidad ciclomática, es decir, el número de pruebas unitarias codificadas en relación con el número de pruebas unitarias deseable. Este es un buen indicador de la amplitud y profundidad con la que se han realizado las pruebas.

Aspecto	Contenido
Nombre	Densidad de Pruebas Unitarias
Característica	Característica: • Calidad Interna – Mantenibilidad Subcaracterística: • Calidad Interna – Capacidad de ser probado • Calidad Interna – Estabilidad en las modificaciones
Fase del ciclo de vida de la calidad de producto (objetivo de medición)	Calidad Interna (Producto en desarrollo/mantenimiento)
Objetivos	Evaluar la densidad de pruebas unitarias para conocer la capacidad de las pruebas unitarias a la hora de validar el código. Pregunta: ¿Existen suficientes pruebas unitarias para validar gran parte del código fuente del sistema?
Lenguaje	Esta medida de calidad está disponible para: • Java • C++

Aspecto	Contenido
Criterio de decisión	Umbrales recomendados: u=0,75; v=0,90; Valores menores que u serán considerados como mala calidad (nota 0 de 100). Valores mayores que v serán considerados como buena calidad (nota 100 de 100). El resto de valores serán valores intermedios (nota entre 0 y 100).
Visualización	El resultado de le medición de calidad se mostrará en el sitio Web del proyecto generado a partir de la herramienta KEMIS.
Función de medición	$UTD = \frac{\sum UT}{\sum CC}$
Elementos de la medida de calidad	**UTD:** Densidad de Pruebas Unitarias. **UN:** Número de Pruebas Unitarias. **CC:** Complejidad Ciclomática.
Método de Medición	**UTD:** número de pruebas unitarias. **CC:** número de caminos linealmente independientes.
Fuentes de datos	Informes de las siguientes herramientas: • JavaNCSS (Java) • EMMA (Java) • CppUnit • CCCC (C++)
Coste de obtención de los datos	Bajo: estas medidas se obtienen automáticamente a partir de herramientas.

Aspecto	Contenido
Escenarios de uso por cada uno de los roles	Esta medida de calidad puede ser usada por: • El equipo de desarrollo: o Los jefes de proyecto pueden conocer la calidad de las pruebas unitarias desarrolladas por sus equipos y el nivel de confianza de acuerdo a la cantidad de código que está siendo probado. o Los desarrolladores pueden conocer la calidad de las pruebas unitarias del sistema. • Los clientes del producto.
Función de Calidad	$Q\_UTD = \begin{cases} 0 & , si \quad UTD < u \\ \frac{UTD - u}{v - u} \cdot 100 & , si \quad u \leq UTD \leq v \\ 100 & , si \quad UTD > v \end{cases}$

Tabla 3-16. Densidad de pruebas unitarias

En la Figura 3-13 se grafica la función de calidad de la métrica. En este caso si la densidad de pruebas unitarias es mayor al 90% el resultado es 100. Y es 0 en caso de que la densidad de pruebas unitarias sea inferior al 75%. Por último, para las mediciones de densidad de pruebas entre los valores extremos se utiliza una función lineal.

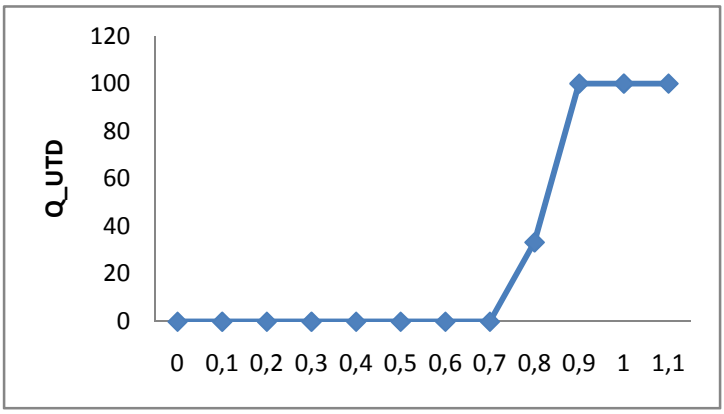


Figura 3-13. Función de densidad de pruebas unitarias (Q_UTD)

3.3.5.2 Densidad de errores en las pruebas unitarias

La densidad de errores en las pruebas unitarias mide la relación entre el número de defectos en pruebas unitarias y el tamaño del proyecto. Este indicador nos muestra lo complejo que son las pruebas unitarias y configura la tolerancia del modelo a los errores.

Aspecto	Contenido
Nombre	Densidad de Errores en las Pruebas Unitarias
Características	Característica: • Calidad Interna – Mantenibilidad Subcaracterística: • Calidad Interna – Capacidad de ser probado • Calidad Interna – Estabilidad en las modificaciones
Fase del ciclo de vida de la calidad de producto (objetivo de medición)	Calidad Interna (Producto en desarrollo/mantenimiento)
Objetivos	Evaluar la densidad de errores en las pruebas unitarias para conocer si dichas pruebas unitarias están siendo mantenidas correctamente. Pregunta: ¿Están los desarrolladores manteniendo las pruebas unitarias después de realizar modificaciones en el código?
Lenguaje	Esta medida de calidad está disponible para: • Java • C++

Aspecto	Contenido
Criterio de decisión	Umbrales recomendados: u=0; v=5; Valores mayores que v serán considerados de mala calidad (Nota 0 de 100). Valores menores que u serán considerados como buena calidad (nota 100 de 100). El resto de valores tendrán valores intermedios (nota entre 0 y 100).
Visualización	El resultado de le medición de calidad se mostrará en el sitio Web del proyecto generado a partir de la herramienta Maven.
Función de medición	$UTED = \frac{\sum UTE}{\sum UT}$
Elementos de la medida de calidad	**UTED:** Densidad de Errores en las Pruebas Unitarias. **UTE:** Número de Errores en las Pruebas Unitarias. **UT:** Número de Pruebas Unitarias.
Método de Medición	**UTE:** número de errores en las pruebas unitarias. **UT:** número de pruebas unitarias.
Fuentes de datos	Informes de las siguientes herramientas: • EMMA (Java) • CppUnit (C++)
Coste de obtención de los datos	Bajo: estas medidas se obtienen automáticamente a partir de herramientas.

Aspecto	Contenido
Escenarios de uso por cada uno de los roles	Esta medida de calidad puede ser usada por: • El equipo de desarrollo: o Los jefes de proyecto pueden conocer el nivel de mantenimiento que se está realizando en las pruebas unitarias por parte de los desarrolladores y el nivel de confianza de que el código se está probando correctamente. o Los desarrolladores pueden conocer el estado y la calidad de las pruebas unitarias del sistema. • Los clientes del producto.
Función de calidad	$Q\_UTED = \begin{cases} 100 & , si \quad UTED < u \\ \frac{v - UTED}{v - u} \cdot 100 & , si \quad UTED \leq x \leq v \\ 0 & , si \quad UTED > v \end{cases}$

Tabla 3-17. Densidad de errores en las pruebas unitarias

En la Figura 3-14 se grafica la función de calidad de la métrica. En este caso si más de un 5% de las pruebas son erróneas el resultado es 0. Y es 100 en caso de que no existan errores al ejecutar las pruebas unitarias. Por último, para las mediciones de densidad de errores en las pruebas unitarias entre los valores extremos se utiliza una función lineal.

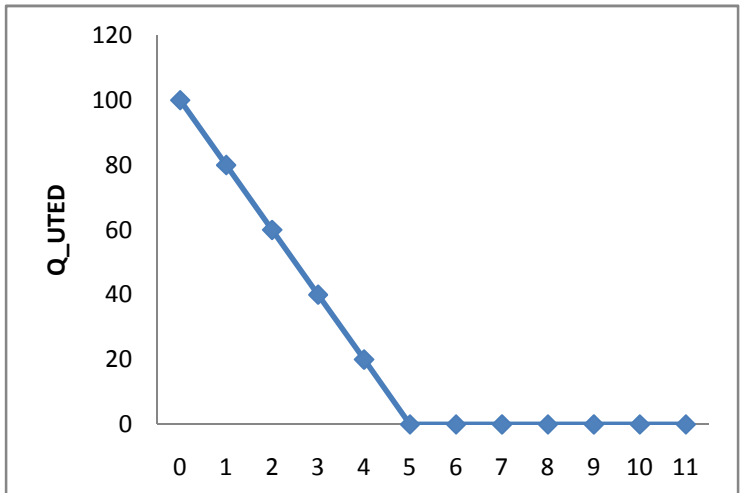


Figura 3-14. Función de densidad de errores de pruebas unitarias (Q_UTED)

3.3.5.3 Cobertura de pruebas unitarias

La cobertura de pruebas unitarias permite conocer el porcentaje de código que ha sido ejecutado al menos una vez, por las pruebas unitarias.

ASPECTO	CONTENIDO
Nombre de la medida de calidad	Cobertura de Pruebas Unitarias
Propiedades de la calidad del software	Caracterísitca: • Calidad Interna – Mantenibilidad Subcaracterística: • Calidad Interna – Capacidad de ser probado • Calidad Interna – Estabilidad en las modificaciones
Fase del ciclo de vida de la calidad de producto (objetivo de medición)	Calidad Interna (Producto en desarrollo/mantenimiento)

ASPECTO	CONTENIDO
Propósito de la medida de calidad.	Evaluar la cobertura de líneas del código que se ejecutan con las pruebas unitarias para conocer el porcentaje de código que está siendo validado. Pregunta: ¿Cuál es el porcentaje de líneas de código probadas en la ejecución de las pruebas unitarias?
Lenguajes de programación	Esta medida de calidad está disponible para: • Java • C++
Criterio de decisión	Umbrales recomendados: u=0,50; v=0,75 Valores menores que u serán considerados mala calidad (nota 0 de 100). Valores mayores que v serán considerados como buena calidad (nota 100 de 100). El resto de valores tendrán valores intermedios (nota entre 0 y 100)
Visualización	El resultado de le medición de calidad se mostrará en el sitio Web del proyecto generado a partir de la herramienta KEMIS.
Función de medición	$UTCR = \frac{\sum UTLC}{\sum NCSS}$
Elementos de la medida de calidad	**UTCR:** Ratio de Cobertura de Líneas **NCSS:** Sentencias de Código que no son comentarios. **UTLC:** NCSS ejecutadas por las pruebas unitarias.
Método de Medición	**NCSS:** número de sentencias de código que no son comentarios. **UTLC:** número de NCSS ejecutadas, al menos una vez, durante la ejecución de las pruebas unitarias.
Fuentes de datos	Informes de las siguientes herramientas: • EMMA (Java) • JUnit(Java) • CppUnit(C++)

ASPECTO	CONTENIDO
Coste de obtención de los datos	Bajo: estas medidas se obtienen automáticamente a partir de herramientas.
Escenarios de uso por cada uno de los roles	Esta medida de calidad puede ser usada por: • El equipo de desarrollo: o Los jefes de proyecto pueden conocer la calidad de las pruebas unitarias desarrolladas por sus equipos y la cantidad de código que está siendo probado. También pueden conocer el nivel de protección que proporcionan las pruebas unitarias sobre posibles futuros errores. o Los desarrolladores pueden conocer la calidad de las pruebas unitarias desarrolladas. • Los clientes del producto.
Función de calidad	$Q\_UTCR = \begin{cases} 0 & , si \quad UTCR < u \\ \frac{v - UTCR}{v - u} \cdot 100 & , si \quad u \le UTCR \le v \\ 100 & , si \quad UTCR > v \end{cases}$

Tabla 3-18. Cobertura de pruebas unitarias

En la Figura 3-15 se grafica la función de calidad de la métrica. En este caso si menos de un 50% del código fuente ha sido cubierto por las el resultado es 0. Y es 100 en caso de que la cobertura sea superior al 75%. Por último, para los valores de cobertura intermedia se utiliza una función lineal.

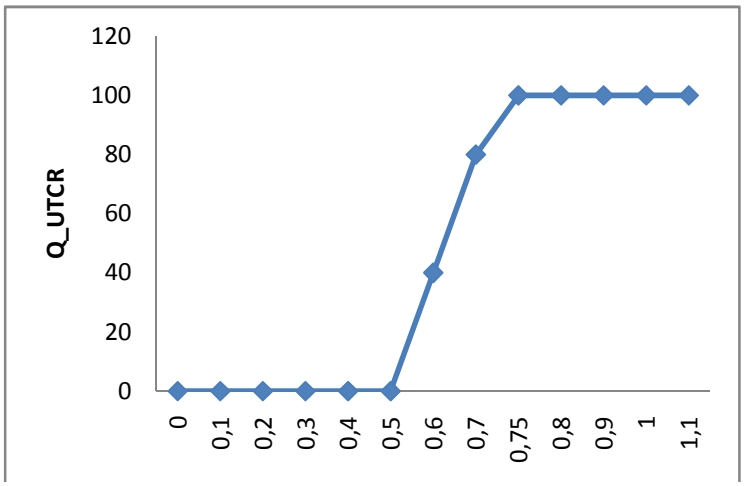


Figura 3-15. Función de cobertura de pruebas unitarias (Q_UTCR)

3.3.5.4 Densidad de defectos en pruebas unitarias

La densidad de defectos en pruebas unitarias mide la relación entre el número de defectos en pruebas unitarias y el tamaño del proyecto. Este indicador nos muestra lo complejo que son las pruebas unitarias.

Aspecto	Contenido
Nombre de la medida de calidad	Densidad de Defectos en pruebas unitarias
Propiedades de la calidad del software	Característica: • Calidad Interna – Mantenibilidad Subcaracterística: • Calidad Interna – Capacidad para ser probado
Fase del ciclo de vida de la calidad de producto (objetivo de medición)	Calidad Interna (Producto en desarrollo/mantenimiento)

Aspecto	Contenido
Propósito de la medida de calidad.	Evaluar lo defectos de las pruebas unitarias. Pregunta: ¿Cuál es el porcentaje de defectos de las pruebas unitarias?
Lenguajes de programación	Esta medida de calidad está disponible para: • Java • C++
Criterio de decisión	Umbrales recomendados: u = 0 y v = 40; Valores menores que u serán considerados como buena calidad (nota 100 de 100). Valores mayores que v serán considerados como mala calidad (nota 0 de 100). El resto de valores serán valores intermedios (nota entre 0 y 100).
Visualización	El resultado de le medición de calidad se mostrará en el sitio Web del proyecto generado a partir de la herramienta KEMIS.
Función de medición	$UTDEFD = \frac{\sum UTDEF}{NCSS} \cdot 100$
Elementos de la medida de calidad	**UTDEFD:** Densidad de defectos en pruebas unitarias **UTDEF:** Defectos en las pruebas. **NCSS:** Non Commenting Source Statements, sentencias de código fuente que no son comentario
Método de Medición	**UTDEF:** número de defectos en las pruebas. **NCSS:** número de sentencias de código que no son comentarios.
Fuentes de datos	Informes de las siguientes herramientas: • JUnit(Java) • CppUnit (C++) • EMMA (Java)
Coste de obtención de los datos	Bajo: estas medidas se obtienen automáticamente a partir de herramientas.

Aspecto	Contenido
Escenarios de uso por cada uno de los roles	Esta medida de calidad puede ser usada por: • El equipo de desarrollo: o Los jefes de proyecto pueden conocer la calidad de desarrollo de sus equipos. o Los desarrolladores pueden conocer la calidad de sus desarrollos. • Los clientes del producto.
Función de calidad	$Q\_UTDEFD = \begin{cases} 100 & ,si \quad UTDEFD < u \\ \frac{v - UTDEFD}{v - u} \cdot 100 & ,si \quad u \le UTDEFD \le v \\ 0 & ,si \quad UTDEFD > v \end{cases}$

Tabla 3-19. Densidad de defectos en pruebas unitarias

En la Figura 3-16 se grafica la función de calidad de la métrica. En este caso si más del 40% del código fuente tiene errores el resultado es 0. Y es 100 en caso de que no existan errores al ejecutar las pruebas unitarias. Por último, para las mediciones de densidad de errores en las pruebas unitarias entre los valores extremos se utiliza una función lineal.

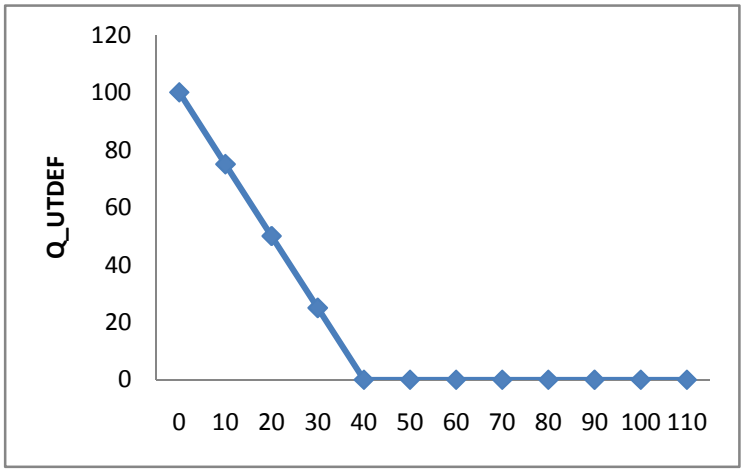


Figura 3-16. Función de densidad de defectos en pruebas unitarias (Q_UTDEFD)

3.3.5.5 Densidad de fallos de pruebas unitarias

La densidad de fallos en pruebas unitarias mide la relación entre el número de fallos en pruebas unitarias y la complejidad ciclomática. Mientras más complejidad presente el código fuente, mayor será la probabilidad de errores al ejecutar pruebas unitarias. En el caso de este ratio, si es alto, significará que la cantidad de errores es superior a la media esperada.

Aspecto	Contenido
Nombre	Tasa de fallos de pruebas unitarias
Carácterística	Característica: • Calidad Interna – Mantenibilidad Subcaracterística: • Calidad Interna – Capacidad para ser probado
Fase del ciclo de vida de la calidad de producto (objetivo de medición)	Calidad Interna (Producto en desarrollo/mantenimiento)
Objetivos	Evaluar lo defectos de las pruebas unitarias. Pregunta: ¿Cuál es el porcentaje de defectos de las pruebas unitarias?
Lenguaje	Esta medida de calidad está disponible para: • Java • C++
Criterio de decisión	Umbrales recomendados: u = 0 y v = 5; Valores menores que u serán considerados como buena calidad (nota 10 de 10). Valores mayores que v serán considerados como mala calidad (nota 0 de 10). El resto de valores serán valores intermedios (nota entre 0 y 10).
Visualización	El resultado de le medición de calidad se mostrará en el sitio Web del proyecto generado a partir de la herramienta KEMIS.

Aspecto	Contenido
Función de medición	$UTFD = \frac{\sum UTF}{CC}$
Elementos de la medida de calidad	**UTFD:** Densidad de Fallos de las Pruebas Unitarias **UTF:** Número de Fallos de las Pruebas Unitarias **CC:** Complejidad Ciclomática.
Método de Medición	**UTF:** número de fallos en las pruebas. **CC:** número de caminos linealmente independientes.
Fuentes de datos	Informes de las siguientes herramientas: • JavaNCSS(Java) • JUnit(Java) • CppUnit(C++) • EMMA(Java) • CCCC(C++)
Coste de obtención de los datos	Bajo: estas medidas se obtienen automáticamente a partir de herramientas.
Escenarios de uso por cada uno de los roles	Esta medida de calidad puede ser usada por: • El equipo de desarrollo: o Los jefes de proyecto pueden conocer la calidad de desarrollo de sus equipos. o Los desarrolladores pueden conocer la calidad de sus desarrollos. • Los clientes del producto.
Función de calidad	$Q\_UTFD = \begin{cases} 100 & ,si \quad UTFD < u \\ \frac{v - UTFD}{v - u} \cdot 100 & ,si \quad u \le UTFD \le v \\ 0 & ,si \quad UTFD > v \end{cases}$

Tabla 3-20. Densidad de fallos de pruebas unitarias

En la Figura 3-17 se grafica la función de calidad de la métrica. En este caso si el ratio entre la complejidad ciclomática acumulada y el número de pruebas que han fallado es mayor a 5 el resultado es 0. Y es 100 en caso de que no existan errores al ejecutar las pruebas unitarias. Por último, para las mediciones de

densidad de errores en las pruebas unitarias entre los valores extremos se utiliza una función lineal.

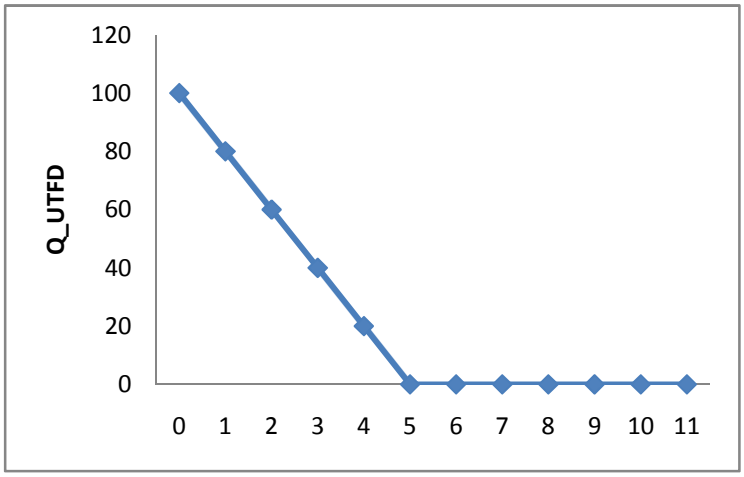


Figura 3-17. Función de densidad de fallos de pruebas unitarias (Q_UTFD)

3.3.6 Características y subcaracterísticas de calidad

Una vez definidas y normalizadas las medidas de calidad que utilizará KEMIS, el modelo de medición de la mantenibilidad define las funciones que permiten computar un valor para cada subcaracterística. Estas funciones, denominadas **funciones de derivación** permiten configurar el modelo, dando mayor importancia a unas medidas de calidad que a otras asignando pesos diferentes a cada medida.

A continuación se muestra la definición de la medición de la característica mantenibilidad y sus subcaracterísticas. Los valores obtenidos para cada subcaracterística contribuyen a una nueva función de derivación que permite obtener el valor agregado para la característica de mantenibilidad. Este es el mismo mecanismo propuesto por la norma ISO/IEC 25000 para la medición de la calidad del producto software (ver Figura 3-18).

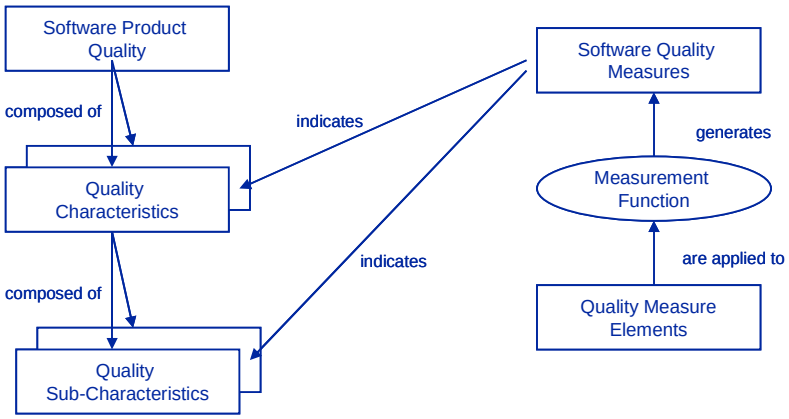


Figura 3-18. Modelo de referencia de medición de la calidad del producto software, según la norma ISO/IEC 25000

3.3.6.1 Mantenibilidad

La Tabla 4-1 muestra la función de derivación para la característica de mantenibilidad. Como se puede observar, dicha funcionalidad se limita a combinar los valores obtenidos para las subcaracterísticas de analizabilidad, estabilidad, cambiabilidad y capacidad para ser probado, asignando un peso a cada una lo cual permte una configuración posterior más flexible.

Mantenibilidad	
Nivel	Estratégico
Función de Derivación	$$MAINT = \frac{(P_{ANAL} \cdot ANAL) + (P_{CHG} \cdot CHG) + (P_{MOST} \cdot MOST) + (P_{TEST} \cdot TEST)}{P_{ANAL} + P_{CHG} + P_{MOST} + P_{TEST}}$$
Atributos	**MAINT:** Mantenibilidad del Producto **P_{ANAL} y ANAL:** Peso y Valor de la Capacidad para ser Analizado. **P_{CHG} y CHG:** Peso y Valor de la Capacidad para ser Cambiado. **P_{MOST} y MOST:** Peso y Calidad de la Estabilidad. **P_{TEST} y TEST:** Peso y Valor de la Capacidad para ser Probado.

Tabla 3-21. Mantenibilidad

3.3.6.2 Analizabilidad

La Tabla 3-22 muestra la función de derivación para la subcaracterística de analizabilidad. Como se puede observar, y al igual que con la característica de mantenibilidad dicha funcionalidad se limita a combinar los valores obtenidos para los atributos descritos en la sección 3.3.2, asignando un peso a cada una lo cual permte una configuración posterior más flexible.

Los atributos que lo conforman son:

- Densidad de complejidad ciclomática.
- Densidad de código repetido.
- Densidad de comentarios.
- Densidad de defectos de la capacidad para ser analizado.

Capacidad para ser analizado	
Nivel	Táctico
Función de Derivación	$ANAL = \frac{(P_{CCD} \cdot Q\_CCD) + (P_{DCR} \cdot Q\_DCR) \cdot (P_{COMD} \cdot Q\_COMD) + (P_{ANALDEFD} \cdot Q\_ANALDEF)}{P_{CCD} + P_{DCR} + P_{COMD} + P_{ANALDEFD}}$
Atributos	**ANAL:** Capacidad del proyecto para ser analizado. **P_{CCD} y Q_CCD:** Peso y Calidad de la Densidad de la Complejidad Ciclomática. **P_{DCR} y Q_DCR:** Peso y Calidad de la Densidad de Código Repetido. **P_{COMD} y Q_COMD:** Peso y Calidad de la Densidad de Comentarios. **$P_{ANALDEFD}$ y ANALDEFD:** Peso y Calidad de la Densidad de Defectos de la Capacidad para ser analizado.

Tabla 3-22.Capacidad para ser analizado

3.3.6.3 Cambiabilidad

La Tabla 3-23 muestra la función de derivación para la subcaracterística de cambiabilidad. Como se puede observar, y al igual que con las demás

características dicha funcionalidad se limita a combinar los valores obtenidos para los atributos descritos en la sección 3.3.3, asignando un peso a cada una lo cual permte una configuración posterior más flexible.

- Densidad de defectos de la capacidad para ser cambiado.
- Densidad de ciclos.

Capacidad para ser cambiado	
Nivel	Táctico
Función de Derivación	$CHG = \frac{(P_{CHGDEFD} \cdot Q\_CHGDEFD) + (P_{CYCD} \cdot Q\_CYCD)}{P_{CHGDEFD} + P_{CYCD}}$
Atributos	**CHG:** Cambiabilidad del Proyecto **$P_{CHGDEFD}$ y Q_CHGDEFD:** Peso y Calidad de la Densidad de defectos de la Capacidad para ser Cambiado. **P_{CYCD} y Q_CYCD:** Peso y Calidad de la Densidad de Ciclos.

Tabla 3-23. Capacidad para ser cambiado

3.3.6.4 Estabilidad

La Tabla 3-24 muestra la función de derivación para la subcaracterística de estabilidad. Como se puede observar, y al igual que con las demás subcaracterísticas dicha funcionalidad se limita a combinar los valores obtenidos para los atributos descritos en la sección 3.3.4. En este caso se ha recopilado un solo atributo por lo que no hace falta asignarle un peso.

Estabilidad	
Nivel	Táctico
Función de Derivación	$MOST = Q\_MOSTDEFD$
Atributos	**MOST:** Estabilidad **$P_{MOSTDEFD}$ y MOSTDEFD:** Peso y Calidad de la Densidad de Defectos de Estabilidad.

Tabla 3-24. Estabilidad

3.3.6.5 Capacidad para ser Probado

La Tabla 3-25 muestra la función de derivación para la subcaracterística de "capacidad para ser probado". Como se puede observar, y al igual que con las otras subcaracterísticas dicha funcionalidad se limita a combinar los valores obtenidos para los atributos descritos en la sección 3.3.5, asignando un peso a cada una lo cual permte una configuración posterior más flexible.

- Densidad de defectos en pruebas unitarias
- Densidad de pruebas unitarias
- Cobertura de pruebas unitarias
- Tasa de fallos de las pruebas unitarias
- Tasa de errores de las pruebas unitarias

Capacidad para ser probado	
Nivel	Táctico
Función de Derivación	$$TEST = \frac{\left(P_{UTDEFD} \cdot Q\_UTDEFD\right) + \left(P_{UTD} \cdot Q\_UTD\right) + \left(P_{UTCR} \cdot Q\_UTCR\right) + \left(P_{UTFD} \cdot Q\_UTFD\right) + \left(P_{UTED} \cdot Q\_UTED\right)}{P_{UTDEFD} + P_{UTD} + P_{UTCR} + P_{UTFD} + P_{UTED}}$$
Atributos	**TEST:** Capacidad para ser probado **P_{UTDEFD} y Q_UTDEFD:** Peso y Calidad de la Densidad de defectos en pruebas unitarias. **P_{UTD} y Q_UTD:** Peso y Calidad de la Densidad de Pruebas Unitarias. **P_{UTCR} y Q_UTCR:** Peso y Calidad de la Densidad de Cobertura de las Pruebas Unitarias. **P_{UTFD} y Q_UTFD:** Peso y Calidad de la Densidad de Fallos de las Pruebas Unitarias. **P_{UTED} y Q_UTED:** Peso y Calidad de la Densidad de Errores de las Pruebas Unitarias.

Tabla 3-25. Capacidad para ser probado

El Modelo de Medición del Valor

Junto a la especificación de un modelo para la medición de la mantenibilidad, otro de los objetivos fijados al comienzo de esta Tesis Doctoral es la construcción de un modelo de medición del valor a ser utilizado para el cálculo del valor de los componentes software desde el punto de vista de la mejora de su mantenibilidad.

Al igual que con el modelo de medición de la mantenibilidad, se busca proporcionar una infraestructura de medición automática basada en métricas básicas obtenidas con herramientas de software libre, y un modelo de medición del valor basado en el propuesto por la norma ISO/IEC 9126.

En este capítulo se describen los indicadores de valor que conforman el modelo propuesto para la medición del valor, se presenta dicho modelo y se discuten los resultados más significativos.

4.1 Analogía entre el Modelo de Medición de la Norma ISO/IEC 25000 y el Modelo de Medición del Valor

Como hemos descrito anteriormente, KEMIS se basa en las normas ISO/IEC 9126 y 25000 para identificar los atributos a partir de los cuales se obtendrán valores indicadores de calidad (no confundir estos valores con la definición de valor en ISBV; en este caso hablamos de valor como el resultado de una medición).

Como se ha comentado anteriormente, estas normas proponen un conjunto de características y subcaracterísticas (medidas de calidad) que componen la calidad del producto software. Los elementos de medida de calidad son medidas base o medidas derivadas obtenidas según describe el método de medición correspondiente. En la Figura 4-1 se observa el esquema antes mencionado y como se van construyendo los valores más abstractos a partir de valores concretos, hasta llegar a la calidad del producto.

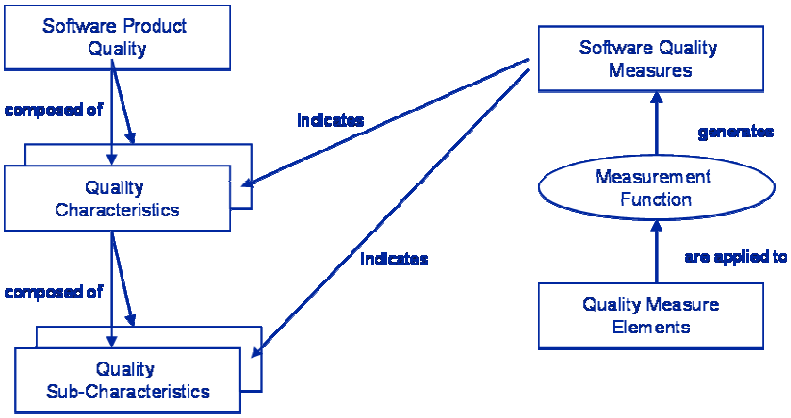


Figura 4-1. Modelo de Referencia de medición de la calidad del producto software, según la norma ISO/IEC 25000

Podemos aplicar la misma idea para definir un modelo para la definición del valor, tal como muestra la Figura 4-2. En la parte izquierda de la figura se encuentra el valor del producto (medida final más abstracta) que está compuesto de indicadores de valor (características de valor) que a su vez está compuesto de subcaracterísticas. Estas características y subcaracterísticas son en realidad indicadores (derivados o básicos) de valor.

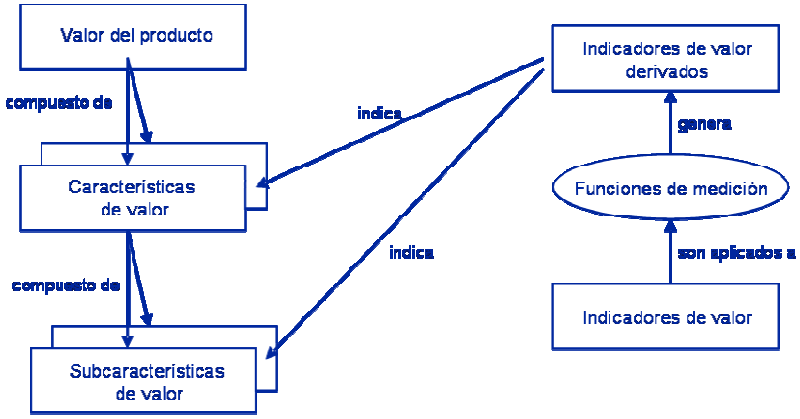


Figura 4-2. Modelo de referencia para la medición del valor basado en el modelo para la medición de la calidad del producto software

Conviene mencionar que no se ha encontrado ningún modelo que, como el que se propone, permita obtener un indicador de valor agregado a partir de indicadores de menor nivel de abstracción. Así, la propuesta consiste en construir el modelo de medición de forma jerárquica, siendo la raíz el valor del producto, y las hojas los indicadores de valor básicos. Por último se incluyen las funciones de medición que correlacionan indicadores de un nivel con un indicador para el nivel siguiente (ver Figura 4-3).

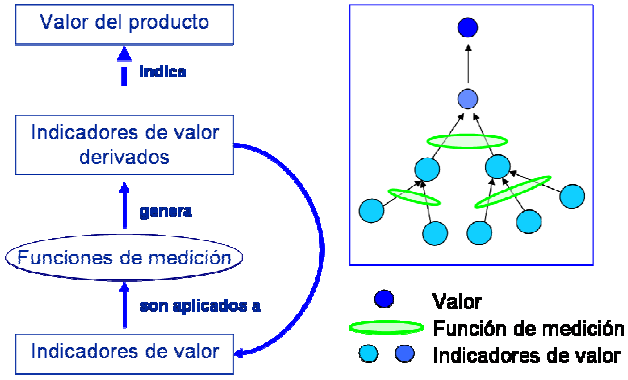


Figura 4-3. Arquitectura del modelo de medición del valor

Teniendo en cuenta lo visto en el capítulo anterior, cada componente del producto software tendrá asociado una medida de mantenibilidad obtenida aplicando el modelo de medición de la mantenibilidad, y una medida de su valor obtenida aplicando el modelo de medición del valor. Como paso final se priorizarán las refactorizaciones de acuerdo con el valor de cada componente a ser refactorizado. Para ello, tal como se ve en la Figura 4-4 existirán componentes con diferentes niveles de mantenibilidad y con una lista de mejoras o refactorizaciones asociadas a cada componente.

Como describiremos a continuación, la forma de priorizar estas refactorizaciones (a partir del valor) tendrá también en cuenta la mantenibilidad actual del componente, de tal manera que se otorgará más valor a las mejoras de un componente con menor mantenibilidad.

COMPONENTE	MANTENIBILIDAD	MEJORA
C1	85	mejora 1 mejora 2
C2	77	mejora 3 mejora 6
C3	100	
C4	35	mejora 3 mejora 6 mejora 10

Figura 4-4. Recomendaciones para aumentar la mantenibilidad.

Esta forma de priorizar las refactorizaciones está basada en el concepto de "retorno de inversión". En economía existen diversos indicadores del valor de las inversiones, que permiten clasificarlas y tomar decisiones (por ejemplo el VAN o la TIR), que ya han sido utilizadas en publicaciones relacionadas con la ISBV [82][84]. En nuestra propuesta aplicamos esta idea, teniendo en cuenta que cada una de las posibles refactorizaciones del producto es una inversión, ya que se necesitarán recursos para llevarlas a cabo, y que en las distintas publicaciones asociadas al valor se hace referencia al "retorno de inversión" o ROI [16] [17] [24], se ha propuesto al ROI como el valor del producto.

La fórmula del ROI relaciona los costes de realizar la mejora con los beneficios obtenidos (ver Figura 4-5). Esta fue una de las conclusiones a las que habíamos llegado en la sección 2.7.3, al estudiar los indicadores de valor de otras disciplinas. Analizando otros indicadores de valor se han encontrado dos tipologías distintas, los indicadores relacionados con el coste de realizar una tarea y los relacionados con el beneficio.

Pero el cálculo del ROI no necesariamente debe ser la indicada en la Figura 4-5. Analizando otros ejemplos [16] [17] [24], lo importante al calcular el ROI es medir la relación entre los costes y los beneficios.

$$ROI = \frac{Beneficio - Coste}{Coste}$$

Figura 4-5. Definición de ROI

Resumiendo, se tiene conceptualmente el modelo de la Figura 4-6, en la que el ROI es el resultado final del modelo de medición del valor y la

mantenibilidad es uno de los componentes principales, de tal manera que la refactorización asociada a un producto con menor mantenibilidad tiene más valor (relación inversa).

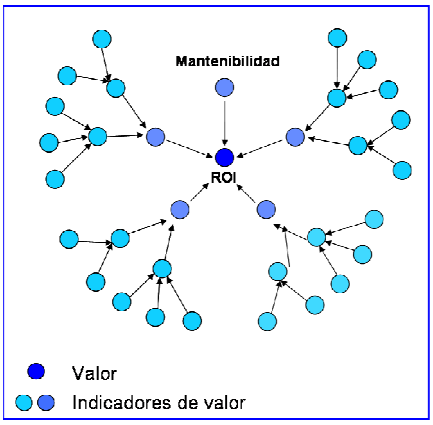


Figura 4-6. Modelo conceptual de medición de valor de KEMIS

En las siguientes secciones se desarrollará el modelo de medición del valor implementado en la herramienta KEMIS, teniendo en cuenta el objetivo del modelo de medición del valor descrito anteriormente.

4.2 El Modelo de Medición Implementado en el Entorno KEMIS

La Figura 4-7 muestra el modelo conceptual del modelo de medición del valor que será implementado en la herramienta KEMIS. El modelo se conformará mediantes atributos, que representarán los indicadores de valor del propio producto software. Si se trata de un atributo base, su valor se obtendrá leyendo el informe proporcionado por la herramienta que lo midió, y si es compuesto se calculará a través de su función de medición. Ejemplos de atributos son los siguientes:

- Atributos base obtenido: por ejemplo el grado de importancia de un componente (como indicador de valor básico).
- Atributos compuestos obtenidos: la mantenibilidad, la posibilidad de cambio, etc. (como indicadores de valor derivados).

Obtener una medición significa, por tanto, obtener el valor de un atributo.

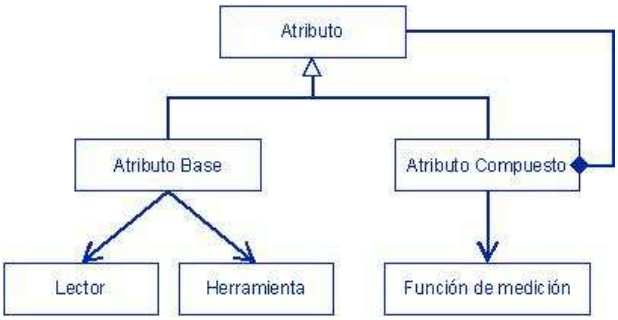


Figura 4-7. Modelo conceptual del modelo de medición del valor

4.3 Configuración del Modelo de Medición del Valor

En esta sección se muestra el desarrollo del modelo de medición del valor, esto es, una definición de los elementos configurables del modelo de acuerdo con lo descrito en el apartado 2.3.5 de esta Tesis Doctoral. Esta definición constituye el modelo por defecto del entorno KEMIS, basado en la experiencia, la investigación de los indicadores de valor utilizados en otras disciplinas (sección 2.7) y el conocimiento de los desarrolladores.

En la Tabla 4-1 se enumeran los indicadores de valor de los niveles 0, 1 y 2 del modelo de medición, de tal manera que en el nivel 0 se encuentra el ROI, que será el valor final obtenido y en los niveles 1 y 2 los indicadores de valor de menor nivel de abstracción.

Nivel 0	Nivel 1	Nivel 2
ROI	Mantenibilidad	Modelo de medición de la mantenibilidad del entorno KEMIS
	Posibilidad de cambio	Función de medición estadística de cambios históricos.
	Importancia relativa	Informe obtenido de los implicados
	Urgencia	Informe obtenido de los implicados
	Coste total de introducir la mejora	Coste de realizar la mejora
		Coste de actualizar las pruebas y la documentación

Tabla 4-1. Relación de indicadores de valor

A continuación, se presentan cada uno de los indicadores de valor y las diferentes funciones de medición que componen el modelo de medición.

4.3.1 Indicadores de Valor

En primer lugar se presenta cada indicador siguiendo la estructura que se recoge en la Tabla 4-2.

Aspecto	Contenido
Nombre del indicador de valor	(Acrónimo) - Nombre asignado al indicador de valor.
Propósito del indicador de valor	Objetivo del indicador de valor. Puede contener una pregunta a la que el indicador de valor dé respuesta.
Lenguajes de programación	Los lenguajes de programación para los cuales el indicador de valor se encuentra disponible.
Función de medición	Ecuación para el cálculo del indicador de valor a partir de sus elementos.
Elementos del indicador de valor	Nombre y definición de los elementos usados para calcular el indicador de valor.
Método de Medición	Descripción del método de medición de los elementos.
Fuentes de datos	Descripción de la/s fuente/s de datos de los elementos usados para calcular el indicador de valor.
Coste de obtención de los datos	Una declaración del coste de obtención de los datos a partir de las fuentes de datos.

Tabla 4-2. Estructura para la definición de indicadores de valor en KEMIS

4.3.2 Importancia relativa

No todos los componentes de un producto software tienen la misma importancia. Tampoco todos los implicados le otorgan el mismo grado de importancia a cada componente. El objetivo de este indicador es contemplar este factor. Para ello cada implicado deberá proporcionar la información que permita asignar una importancia a cada componente. La idea será agregar esta información proveniente de los distintos implicados. El objetivo final es priorizar la refactorización de aquellos componentes a los que se otorgue mayor importancia.

Aspecto	Contenido
Nombre del indicador de valor	**(IR)** - Importancia relative
Propósito del indicador de valor	Obtener una medida cuantitativa de la importancia de los componentes desde el punto de vista de los implicados. Pregunta: ¿Qué tan importante es que mejore la mantenibilidad de este componente respecto de los demás?
Lenguajes de programación	Independiente del lenguaje de programación
Función de medición	Fórmula para obtener la importancia relativa del **componente x** $\sum_{y=1}^{n} I_y C_x * P_y$ Debe realizarse por cada componente. Condiciones: $\sum_{y=1}^{n} P_y = 1$
Elementos del indicador de valor	**n** = cantidad de implicados. $\mathbf{I_yC_x}$ = importancia del y-ésimo implicado para el componente x. $\mathbf{P_y}$ = peso del y-ésimo implicado.
Método de Medición	Análisis de comparación por pares ponderada por implicados (ver sección 4.4.1).
Fuentes de datos	Informe XML con la importancia relativa de cada componente. Los detalles de la definición del esquema se encuentran en la sección 5.2.5.1.
Coste de obtención de los datos	Medio: los datos se obtienen directamente de los implicados, que podrán actualizarlos. La carga de los datos es manual (aunque se proporciona soporte tecnológico).

Tabla 4-3. Importancia relativa

4.3.3 Urgencia

No todos los componentes de un producto software tienen la misma urgencia de ser mejorados. Tampoco todos los implicados le otorgan el mismo grado de urgencia a cada componente. El objetivo de este indicador es contemplar este factor. Para ello cada implicado deberá proporcionar la información que permita asignar una urgencia a cada componente. La idea será agregar esta información proveniente de los distintos implicados. El objetivo final es priorizar la refactorización de aquellos componentes a los que se otorgue mayor urgencia..

Aspecto	Contenido
Nombre del indicador de valor	**(UR)** – Urgencia
Propósito del indicador de valor	Obtener una medida cuantitativa de la urgencia de mejora de los componentes desde el punto de vista de los implicados. Pregunta: ¿Qué tan urgente es que mejore la mantenibilidad de este componente respecto de los demás?
Lenguajes de programación	Independiente del lenguaje de programación
Función de medición	Fórmula para obtener la urgencia del **componente x** $\sum_{y=1}^{n} U_y C_x * P_y$ Debe realizarse por cada componente. Condiciones: $\sum_{y=1}^{n} P_y = 1$
Elementos del indicador de valor	**n** = cantidad de implicados. $\mathbf{U_y C_x}$ = urgencia del y-ésimo implicado para el componente x. $\mathbf{P_y}$ = peso del y-ésimo implicado.
Método de Medición	Análisis de comparación por pares ponderada por implicados (ver sección 4.4.1 y 4.4.3).

Aspecto	Contenido
Fuentes de datos	Informe XML con la urgencia de cada componente. Los detalles de la definición del esquema se encuentran en la sección 5.2.5.2.
Coste de obtención de los datos	Medio: los datos se obtienen directamente de los implicados, que podrán actualizarlos. La carga de los datos es manual (aunque se proporciona soporte tecnológico).

Tabla 4-4. Urgencia

4.3.4 Posibilidad de cambio

Se calculará la posibilidad de que un cambio afecte a un componente del producto. Si se estima mejorar una parte del producto que no va a cambiar o que tiene una muy baja posibilidad de que lo haga no será beneficiosa la mejora. El retorno de invertir en un componente que no cambia a lo largo del tiempo es muy bajo.

La mejora de los componentes con mayor posibilidad de cambio será más beneficiosa.

Aspecto	Contenido
Nombre del indicador de valor	**(PC)** - Posibilidad de cambio
Propósito del indicador de valor	Identificar cuáles son los componentes del producto que tienen más probabilidad de cambiar en el futuro a partir de la información histórica. Pregunta: ¿Qué tan probable es que el componente cambie?
Lenguajes de programación	Actualmente analizado para los lenguajes que soporten las herramientas de control de versiones SVN y CVS.
Función de medición	$\sum_{i=2}^{k} \left\| CH_i - CH_{i-1} \right\| * 2^{i-k}$
Elementos del indicador de valor	**CH_i:** cambios realizados en el componente, en el período i. **k :** cantidad de períodos a considerarse. El valor recomendado es 12.

Aspecto	Contenido
Método de Medición	Explicado en la sección 4.4.4.
Fuentes de datos	El log de las herramientas de control de versiones SVN y CVS.
Coste de obtención de los datos	Bajo: estas medidas se obtienen a partir del procesamiento de los logs de SVN y CVS.

Tabla 4-5. Posibilidad de cambio

4.3.5 Coste de realizar la mejora

Las mejoras propuestas por KEMIS tendrán como objetivo aumentar la mantenibilidad del producto, que previamente ha sido obtenida siguiendo el modelo de medición del entorno. Una de las partes del coste total será el de realizar la mejora. Cada componente a ser mejorado tendrá una complejidad ciclomática asociada, siguiendo con lo propuesto por [17][143] se determinará el coste de realizar la mejora en horas/hombre estimadas de acuerdo con la complejidad ciclomática del componente a mejorar. Esta información ser proporcionada de forma externa por los implicados.

La mejora de los componentes con menor coste será más beneficiosa.

Aspecto	Contenido
Nombre del indicador de valor	**(CR)** - Coste de realizar la mejora
Propósito del indicador de valor	Identificar el esfuerzo en horas/hombre de la mejora, teniendo en cuenta la complejidad ciclomática del componente a mejorar. Pregunta: ¿Cuánto me cuesta mejorar el componente?
Lenguajes de programación	Actualmente implementado para los lenguajes: Java y C++. La única medición que debe realizarse es la de la complejidad ciclomática.

Aspecto	Contenido
Función de medición	Se obtiene el esfuerzo a partir de la media de la complejidad ciclomática de los componentes a mejorar. $\frac{\sum CC}{\#componentes}$ Con este valor se obtiene el esfuerzo en horas/hombre considerado por los implicados
Elementos del indicador de valor	**CC** : Complejidad ciclomática. Fichero XML de los implicados con el coste de mejora por rango de complejidad ciclomática.
Método de Medición	Explicado en la sección 5.2.5.4.
Fuentes de datos	Informe de la herramienta que mide la complejidad ciclomática del lenguaje con el que se ha codificado el producto a evaluar. Informe proporcionado por los implicados en formato XML
Coste de obtención de los datos	Bajo: la información es fácil de obtener y no hay un procesamiento previo.

Tabla 4-6. Coste de realizar la mejora. Coste de actualizar las pruebas

Una de las partes del coste total será el de actualizar las pruebas que han quedado obsoletas por la mejora. Cada componente a ser mejorado tendrá una complejidad ciclomática asociada, siguiendo con lo propuesto por [17][143] se determinará el coste de realizar la mejora en horas/hombre estimadas de acuerdo con la complejidad ciclomática del componente a mejorar. Esta información ser proporcionada de forma externa por los implicados.

La mejora de los componentes con menor coste será más beneficiosa.

Aspecto	Contenido
Nombre del indicador de valor	**(CA)** - Coste de actualizar las pruebas

Aspecto	Contenido
Propósito del indicador de valor	Identificar el esfuerzo en horas/hombre de actualizar las pruebas unitarias del componente a mejorar Pregunta: ¿Cuánto me cuesta actualizar las pruebas unitarias del componente?
Lenguajes de programación	Actualmente implementado para los lenguajes: Java y C++. La única medición que debe realizarse es la de la cantidad de pruebas unitarias por componente.
Función de medición	Se obtiene el esfuerzo a partir de la media de la cantidad de pruebas unitarias de los componentes a mejorar. $\frac{\sum UT}{\#componentes}$ Con este valor se obtiene el esfuerzo en horas/hombre considerado por los implicados
Elementos del indicador de valor	**UT** : Cantidad de pruebas unitarias. Fichero XML de los implicados con el coste de actualización por el rango de prueba unitaria a actualizar.
Método de Medición	Explicado en la sección 5.2.5.4.
Fuentes de datos	Informe de la herramienta que mide la cantidad de pruebas unitarias por componente del lenguaje con el que se ha codificado el producto a evaluar. Informe proporcionado por los implicados en formato XML
Coste de obtención de los datos	Bajo: la información es fácil de obtener y no hay un procesamiento previo.

Tabla 4-7. Coste de actualizar las pruebas

4.3.6 Coste total de introducir la mejora

El coste total de introducir la mejora está formado por el costo de realizar la mejora y por el coste de actualizar las pruebas unitarias relacionadas con los componentes modificados

(CT) - Coste total de introducir la mejora	
Nivel	1
Función de Derivación	**CT = CA + CR**
Atributos	**CA:** coste de actualizar las pruebas unitarias **CR:** coste de realizar la mejora

Tabla 4-8. Coste total de introducir la mejora

4.3.7 Mantenibilidad

Las mejoras propuestas tendrán como objetivo aumentar la mantenibilidad del producto, que previamente ha sido obtenida siguiendo el modelo de medición del entorno. La mejora de los componentes con menor mantenibilidad será más beneficiosa.

Aspecto	Contenido
Nombre del indicador de valor	**(MA)** – Mantenibilidad
Propósito del indicador de valor	Obtener una medida cuantitativa de la mantenibilidad de los componentes a partir del modelo de medición según la norma ISO/IEC 25000 Pregunta: ¿Qué tan mantenible es el componente?
Lenguajes de programación	Actualmente implementado en la herramienta KEMIS para Java y C++
Función de medición	Indicado en el modelo de medición de la mantenibilidad
Elementos del indicador de valor	Indicado en el modelo de medición de la mantenibilidad
Método de Medición	Ejecución del modelo de medición de calidad implementado por el entorno KEMIS
Fuentes de datos	Base de datos del entorno KEMIS.

Aspecto	Contenido
Coste de obtención de los datos	Bajo: estas medidas se obtienen a partir de la ejecución del entorno KEMIS, cuyos valores se encuentran almacenados en base de datos.

Tabla 4-9. Mantenibilidad

4.4 El ROI

El resultado final del valor o ROI estará en función de los indicadores de nivel 1.

$$Valor = f(MA, CT, PC, IR, UR)$$

A continuación se estudia el aporte de estos indicadores al valor para construir la función de derivación del valor. Respecto de la mantenibilidad, puede tener los valores de 0 a 100 [0,100]:

- Si la mantenibilidad es 100% entonces el valor es 0
- Si la mantenibilidad es 0% entonces el valor es grande (no necesariamente el máximo).

Por lo que la fórmula sería la siguiente (donde MA_MAX es en este caso 100):

$$Valor = (MA - MA\_MAX)\, f(CT, PC, IR, UR)$$

La importancia relativa (IR) puede tener valores que vayan de 0 a 100 [0,100] de tal manera que mientras más importante sea el componente mayor valor tendrá mejorarlo. Lo mismo sucede con la urgencia relativa (UR).

Opción	Importante	Urgente	Resultado
1	si	si	Valor alto
2	si	no	Valor medio
3	no	si	Valor medio
4	no	no	Valor bajo

Figura 4-8. Contribución de la importancia y la urgencia al valor

Ambos parámetros son independientes entre sí y están normalizados a valores porcentuales. Si se multiplicarán sucedería que un valor bajo de urgencia influiría en la importancia, y viceversa. Las opciones 2 y 3 indican una contribución media al valor, eso depende del peso en la decisión final que tiene tanto la importancia relativa como la urgencia. En base a lo antes mencionado se propone la siguiente fórmula:

$$f(IR,UR) = IR * P_{ir} + UR.P_{ur}$$

Con la condición de que la ponderación de la importancia relativa y la ponderación de la urgencia (P_{ir} y P_{ur}) sean valores positivos y su suma sea igual a 1.

La posibilidad de cambio (PC) puede tener los valores de 0 a un valor alto indeterminado [0,infinito). Si la PC es muy grande, entonces mejorar la mantenibilidad tiene mucho valor y si la PC es 0 entonces no tiene valor el realizar la mejora. En base a esto y agregando la función descrita anteriormente:

$$Valor = (MA - MA\_MAX)(IR * P_{ir} + UR.P_{ur}f)(PC)\, f(CT)$$

El costo (CT) tiene un valor positivo, por lo que mientras más costosa sea la mejora menor será el valor final. Por lo tanto la función de derivación final es esta:

$$Valor = \frac{(MA - MA\_MAX)(IR * P_{ir} + UR.P_{ur}f)(PC)}{CT}$$

RIO – Retorno de Inversión	
Nivel	0
Función de Derivación	$\frac{(MA - MA\_MAX)(IR * P_{ir} + UR.P_{ur}f)(PC)}{CT}$

RIO – Retorno de Inversión	
Atributos	**MA_MAX:** máximo valor de la mantenibilidad **MA:** mantenibilidad **IR:** importancia relativa **P_{ir}:** ponderación de la importancia relativa **UR:** urgencia **P_{ur}:** ponderación de la urgencia **PC:** probabilidad de cambio **CT:** coste total de introducir la mejora

Tabla 4-10. ROI

4.4.1 Importancia relativa

Para conocer la importancia relativa de los componentes se realizará una evaluación por pares tal y como se describe en [177], construyéndose una matriz donde tanto las filas como las columnas serán los componentes a evaluar. En cada intersección se realizará la evaluación indicando el componente más importante y el grado de importancia relativa.

Juicio verbal	Clasificación numérica
Extremadamente más importante	9-8
Muchísimo más importante	7-6
Más importante	5-4
Moderadamente más importante	3-2
Igual de importante	1

Figura 4-9. Escala de comparación de la importancia relativa entre componentes

La escala de evaluación empieza en 0 para aquellos pares con importancia similar, hasta 9 cuando uno de los componentes es extremadamente más importante (ver la escala en la Figura 4-9).

	C1	C2	C3	C4	C5	C6
C1	1	3	6		6	
C2		1			9	1
C3		8	1	4		3
C4	4	2		1	5	
C5			3		1	2
C6	5			4		1

Figura 4-10. Matriz de comparación rellenada por el implicado

Una vez rellena la matriz se procederá a completar las intersecciones vacías. En cada intersección opuesta a una rellena se evaluará la importancia relativa como la inversa del valor de importancia relativa opuesta. Por ejemplo en la Figura 4-11 se ha evaluado a C1 con una importancia relativa de 6 respecto de C3 y se completa la matriz en la intersección de C3-C1 con el valor 1/6.

	C1	C2	C3	C4	C5	C6
C1	1	3	6	1/4	6	1/5
C2	1/3	1	1/8	1/2	9	1
C3	1/6	8	1	4	1/3	3
C4	4	2	1/4	1	5	1/4
C5	1/6	1/9	3	1/5	1	2
C6	5	1	1/3	4	½	1

Figura 4-11. Matriz de comparación completada con los valores opuestos

A partir de aquí se realizarán los pasos para obtener una síntesis de las importancias relativas. El paso 1 es sumar los valores por columna (ver Figura 4-12, fila final o “S”).

	C1	C2	C3	C4	C5	C6
C1	1	3	6	1/4	6	1/5
C2	1/3	1	1/8	1/2	9	1
C3	1/6	8	1	4	1/3	3
C4	4	2	1/4	1	5	1/4
C5	1/6	1/9	3	1/5	1	2
C6	5	1	1/3	4	½	1
S	10,67	15,11	10,71	9,95	21,83	7,45

Figura 4-12. Paso 1: sumar las columnas

Una vez obtenida la suma, se dividen los valores de la matriz por la suma que corresponde a su columna (ver Figura 4-13).

	C1	C2	C3	C4	C5	C6
C1	0,09	0,20	0,56	0,03	0,27	0,03
C2	0,03	0,07	0,01	0,05	0,41	0,13
C3	0,02	0,53	0,09	0,40	0,02	0,40
C4	0,38	0,13	0,02	0,10	0,23	0,03
C5	0,02	0,01	0,28	0,02	0,05	0,27
C6	0,47	0,07	0,03	0,40	0,02	0,13

Figura 4-13. Paso 2: dividir cada elemento por el total de cada columna

Por último se suman los valores de cada fila y se divide por la cantidad de componentes. Por ejemplo para C1 es (0.09 + 0.20 + 0.56 + 0.03 + 0.27 + 0.03)/6 = 0.1966. De esta manera obtendremos los porcentajes de la importancia relativa de cada componente, normalizados al 100% (ver Figura 4-14).

	C1	C2	C3	C4	C5	C6	Importancia relativa
C1	0,09	0,20	0,56	0,03	0,27	0,03	19,66%
C2	0,03	0,07	0,01	0,05	0,41	0,13	11,76%
C3	0,02	0,53	0,09	0,40	0,02	0,40	24,31%
C4	0,38	0,13	0,02	0,10	0,23	0,03	14,90%
C5	0,02	0,01	0,28	0,02	0,05	0,27	10,62%
C6	0,47	0,07	0,03	0,40	0,02	0,13	18,75%

Figura 4-14. Paso 3: promediar los elementos en cada fila y obtener la importancia relativa

4.4.2 Proceso de obtención de la importancia relativa

El entorno KEMIS tendrá como origen de datos para la importancia relativa un fichero en formato XML, con una matriz por implicado. El resultado de cada matriz de comparación por pares dará la importancia relativa de cada componente para el implicado que ha completado la matriz. A su vez cada implicado tendrá un peso en la toma de decisión de la importancia relativa, de tal manera que la suma de los pesos sea siempre 1. Por último se obtienen los porcentajes de importancia de cada componente teniendo en cuenta la importancia relativa de todos los implicados.

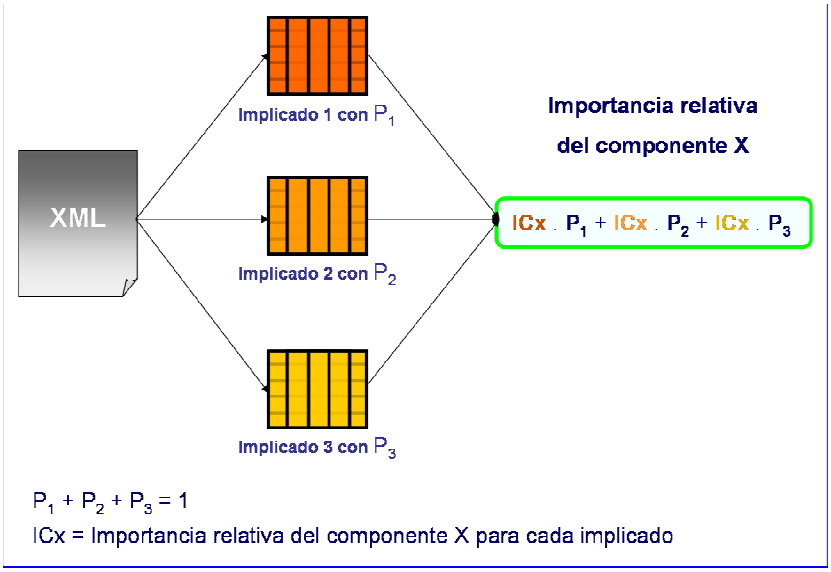


Figura 4-15. Origen y cálculo de la importancia relativa

4.4.3 Urgencia

En el caso de la obtención de la urgencia de mejora para cada componente existen dos opciones. La primera, realizarlo de forma análoga al cálculo de la importancia relativa. Esto en la práctica puede ser costoso, ya que la importancia relativa es una información de configuración estática que tiende a no cambiar en el tiempo mientras que la urgencia si variará y provocará que los implicados tengan que volver a comparar las urgencias por pares.

COMPONENTE	URGENCIA
C1	25%
C2	55%
C3	15%
C4	5%
SUMA	100%

Figura 4-16. Lista de urgencias relativas

La segunda opción propuesta es esperar directamente los valores de urgencia relativa de los componentes. Como única restricción se los espera en porcentaje y normalizados al 100%.

4.4.4 Posibilidad de cambio

Se calculará la posibilidad de que un cambio afecte a un componente del producto. Si se estima mejorar una parte del producto que no va a cambiar o que tiene una muy baja probabilidad de que lo haga no será beneficiosa la mejora. El retorno de invertir en un componente que no cambia a lo largo del tiempo es muy bajo.

Utilizaremos un método basado en información histórica de los cambios del componente para predecir la probabilidad de cambios futuros [178][179][180]. Se utilizará una fórmula simplificada de la descrita en [179] para obtener la probabilidad de cambio.

$$WE_{1..k}(H, P, w(i,k)) = \sum_{i=2}^{k} \left| P_i(H) - P_{i-1} \right| w(i,k)$$

Donde WE es una medición genérica que tiene en cuenta la evolución de una propiedad P ponderada por una función de peso w a través de la historia H. En este caso la historia se encuentra dividida en k períodos de tiempo.

WE es la suma de los cambios en la propiedad P en las diferentes versiones del sistema evaluado y donde cada cambio es ponderado con un peso de acuerdo a la versión i donde ocurre el cambio, comparado con la versión anterior.

4.4.4.1 Especializando la fórmula de posibilidad de cambio

En base a lo descrito anteriormente se ha realizado la especialización de la posibilidad de cambio. Se han seguido los siguientes pasos:

- Identificar la historia total a considerar. En este caso y siguiendo el trabajo de [181], consideramos la historia total como de 12 meses.
- Identificar los períodos de tiempo (el valor de k). De acuerdo a [181], lo definimos como de 1 mes y por lo tanto el valor de k es en este caso 12.
- Identificar la propiedad que tendremos en cuenta para definir el grado de cambio. En este caso utilizaremos la siguiente fórmula.

$$CH_i = \frac{\#La + \#Lb + \#Lm}{COML + NCSS}$$

Donde #La es la cantidad de líneas agregadas durante el mes para el componente, #Lb es la cantidad de líneas borradas, y #Lm es la cantidad de líneas modificadas. COML representa la cantidad de líneas de comentarios y NCSS la cantidad de líneas de código, por lo tanto el denominador es la cantidad total de líneas. El resultado final es el indicador de cambio (CH) de un componente.

- Indicar la función de peso. Teniendo en cuenta que los cambios más cercanos en el tiempo tienen más peso que los cambios más alejados, se utiliza una función de peso que decrece en importancia respecto del tiempo. La función utilizada será la que aparece en [179] y que tiene en cuenta las potencias negativas de base dos.

$$w(i,k) = 2^{i-k}$$

- El indicador de la probabilidad de cambio de un componente i estará dado por:

$$\sum_{i=2}^{k} \left| CH_i - CH_{i-1} \right| * 2^{i-k}$$

Arquitectura de la Herramienta KEMIS

En este capítulo se describe la arquitectura de la herramienta KEMIS con la que se han implementado los modelos de medición de la mantenibilidad y del valor. El capítulo se desarrolla en las diferentes vistas arquitecturales de la herramienta con especial hincapié en aspectos del diseño, utilización de patrones y las características de la medición del valor.

5.1 Vista Lógica

5.1.1 Macro arquitectura

La herramienta KEMIS utiliza una arquitectura cliente-servidor. En el servidor encontramos dos proyectos, KEMIS-CORE y KEMIS-PLUGIN.

- KEMIS-CORE: encargado de realizar la lectura de los diferentes reportes generados al ejecutar las herramientas de medición. A partir de estas medidas de calidad se calcularán los atributos básicos de medición y se almacenarán en la base de datos. Luego serán utilizadas para calcular los indicadores de calidad de más alto nivel (subcaracterísticas y características).
- KEMIS-PLUGIN: plugin desarrollado para la herramienta de construcción Maven, encargado de ejecutar el KEMIS-CORE luego de obtener los reportes de las herramientas de medición.
- KEMIS-WEB: una aplicación Web que permite tener acceso remoto a los resultados de las mediciones. También permite el envío de proyectos a ser evaluados automáticamente desde el servidor Web.

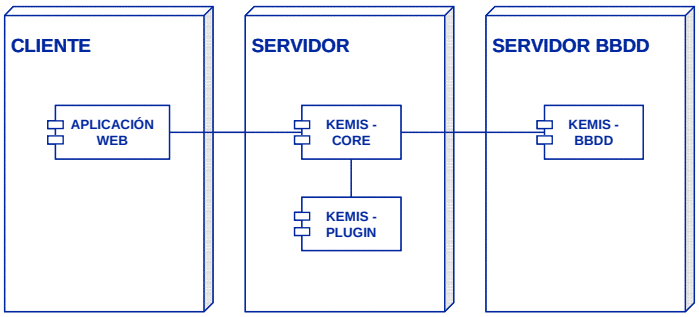


Figura 5-1. Arquitectura cliente-servidor de la herramienta KEMIS

A continuación se muestra la agrupación en paquetes de los principales elementos del sistema de KEMIS-CORE.

- MMNT: contiene el mapeo objeto-relacional de las entidades asociadas con las mediciones realizadas.
- MMMT.MMNTMODEL: realiza el mapeo objeto-relacional del modelo de medición, que consta de un conjunto de atributos básicos y compuestos, diseñados a partir de un patrón composite.
- COMMON.FUNCTION: realiza el mapeo objeto-relacional de las funciones de medición. Las funciones pueden ser unarias, binarias y n-arias. La lógica de ejecución de las funciones también se realiza aquí.
- COMMON.VALUE: contiene las entidades objeto-relaciones que representan los valores de las mediciones. Estos pueden ser valores básicos o agregado, únicos o un rango de ellos.
- MMNT.TOOL: agrupa todo lo referido las herramientas de medición básicas que ejecuta el entorno KEMIS.
- MMNT.ENTITY: realiza el mapeo objeto-relacional de las entidades medidas. Una entidad es un módulo del producto a ser evaluado. Los módulos que solo sean contenedores de otros también serán evaluados, de tal manera que se crea una relación padre-hijo entre entidades.
- MMNT.REPORTS: este módulo es el encargado de generar los reportes HTML o PDF utilizando la tecnología JasperReport.
- MMNT.READER: en este módulo se realizan las lecturas de los resultados generados por las herramientas de medición ejecutadas por KEMIS.
- VALUE_MMNT: paquete que realiza el mapeo objeto-relacional de la nueva funcionalidad.
- VALUEREADER: paquete que contiene los lectores para los distintos informes que es necesario analizar para calcular el valor de la aplicación analizada.
- STRUCTURE: contiene clases que permiten generar la estructura de módulos, paquetes y clases necesaria para generar el informe resultante.
- EXECUTION: contiene las clases propias de la ejecución de la aplicación.

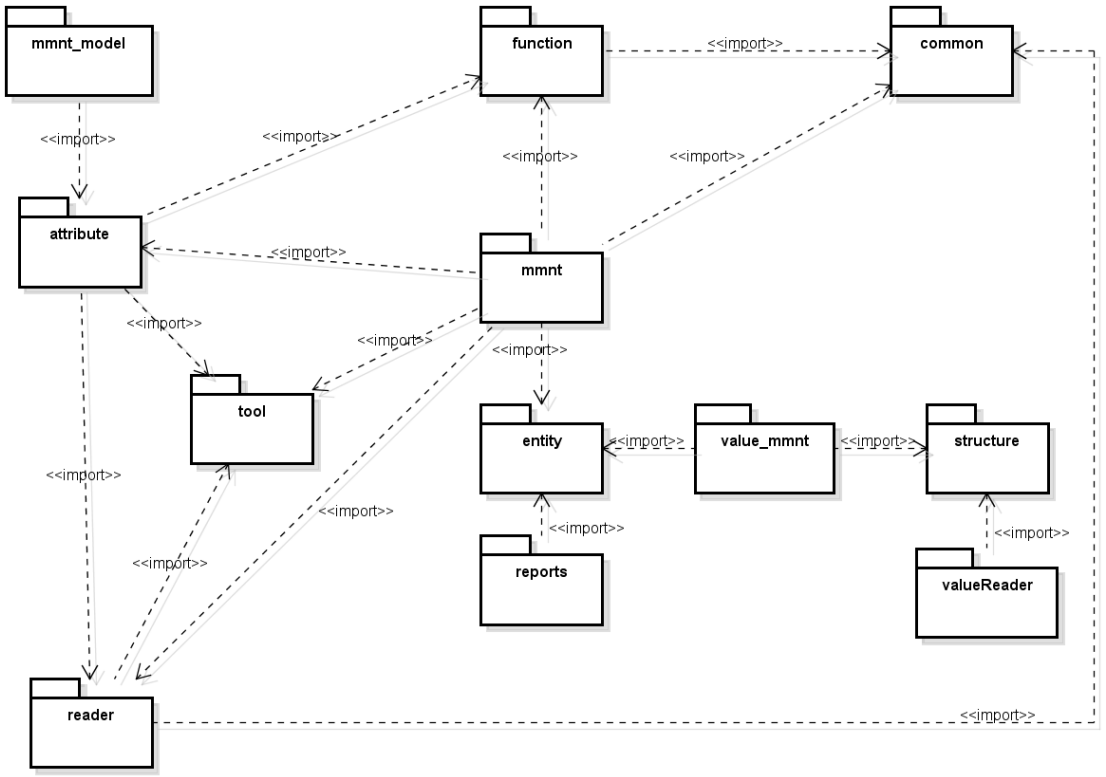


Figura 5-2. Diagrama de paquetes principales de KEMIS-CORE

Así mismo, se encuentra en fase de diseño el tratamiento de las mejoras y las refactorizaciones. Se introducirán los siguientes nuevos paquetes:

- REFACTORING: Encargado de dar soporte a las refactorizaciones.
- ACTIONS: Contiene los distintos tipos de bad smells que pueden ser identificados por el sistema.
- AST: Encargado de generar el árbol de sintaxis abstracta (Abstract Syntax Tree – AST), así como los distintos tipos de sentencias existentes en el código fuente analizado.
- ASSOCIATION: Encargado de definir y detectar las asociaciones existentes entre las diferentes partes del código.
- DECOMPOSITION: Contiene las clases necesarias para descomponer el código analizado en diferentes estructuras o statements.
- CFG: Contiene clases de soporte al paquete Decomposition para descomponer el código.
- INHERINTANCE: Encargado de definir y detectar la herencia existente dentro del código fuente.
- UTIL: Contiene clases necesarias para crear el árbol AST.
- MATH: Contiene clases de soporte al paquete Util, que permite realizar ciertas operaciones matemáticas con las clases.
- DISTANCE: Encargado de generar la matriz de candidatos a refactorización, calcular las distancias y detectar candidatos definitivos a refactorización.
- ENVIRONMENT: Encargada de simular un entorno de trabajo para poder utilizar clases propias de Eclipse.
- ENVIRONMENTUTIL: Proporciona soporte al paquete Environment.
- MANIPULATORS: Permite realizar manipulaciones sobre el árbol AST.
- PREFERENCES: Contiene preferencias y variables globales al sistema.

Estos nuevos paquetes se integran con la aplicación, como puede observarse en la siguiente figura:

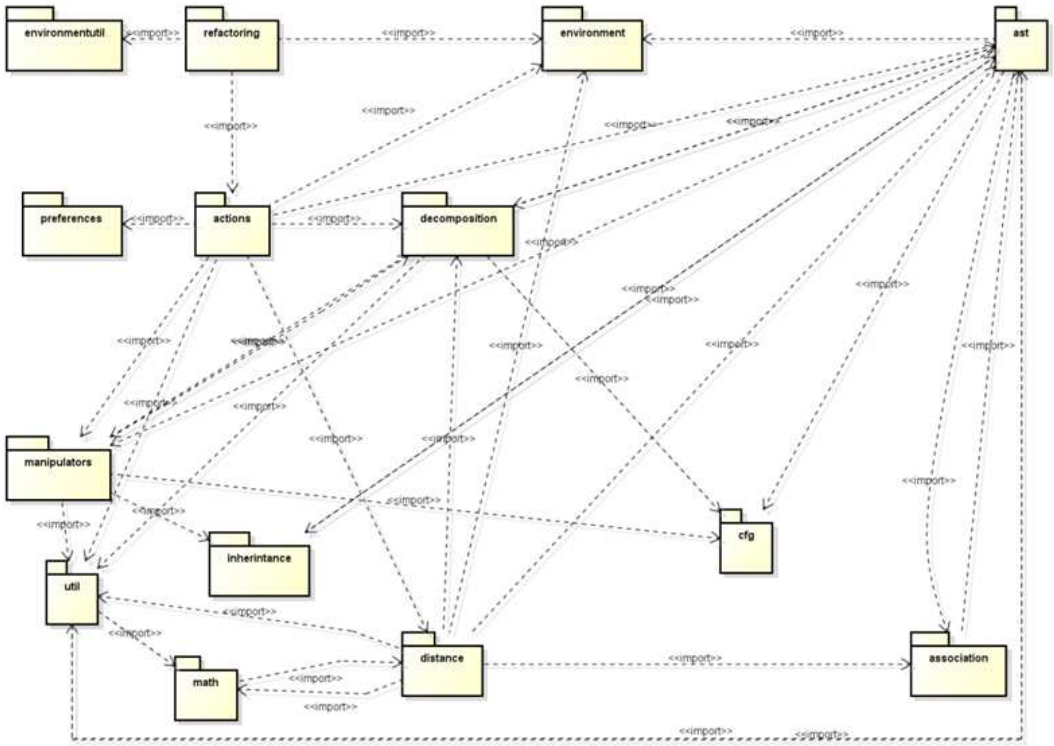


Figura 5-3. Diagrama de paquetes

5.2 Vista de Procesos

A continuación se muestran los procesos más significativos del sistema KEMIS-core, a través de diagramas de actividad.

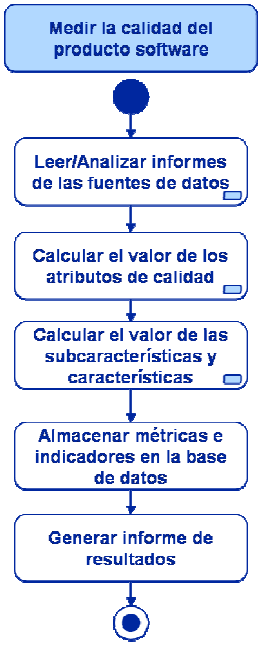


Figura 5-4. Diagrama de actividades para medir la calidad del producto software

A su vez cada uno de estos pasos se dividen en un conjunto de tareas menores, como se verá en la Figura 5-5. El objetivo principal de la herramienta KEMIS es la medición de la calidad del producto software. Como base fundamental construirá un modelo de medición genérico, capaz de ser configurado. En los apartados 3.3 y 4.3 de esta Tesis Doctoral se han descrito los diferentes modelos de medición implementados en KEMIS. Para lograr esta generalidad se ha diseñado una base de datos capaz de mantener los modelos de medición de una forma genérica (ver apartado 5.4.2). De esta manera, los pasos para realizar la medición son los siguientes:

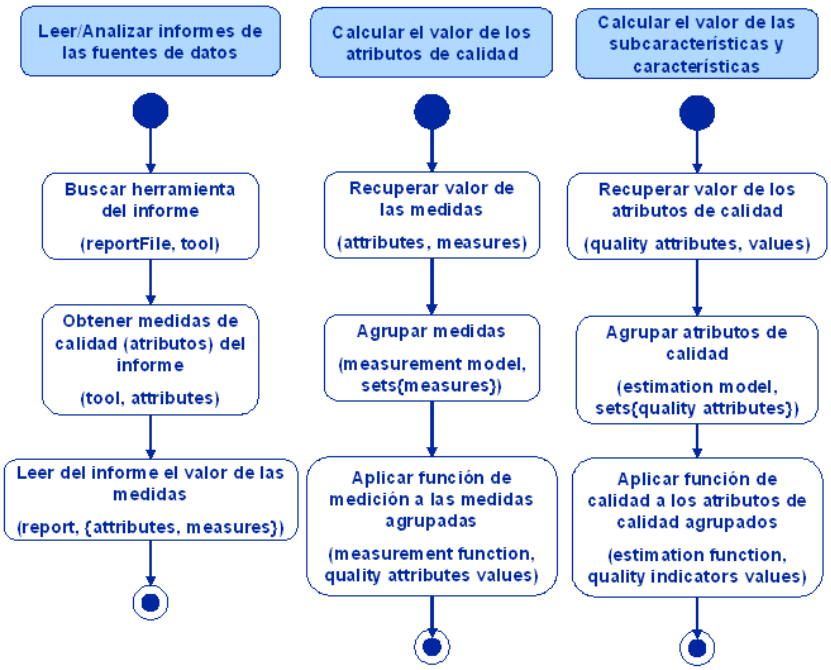


Figura 5-5. Detalle de las actividades realizadas.

5.2.1 Leer/Analizar los informes de las fuentes de datos

Una de las características más importantes es la obtención de los datos. En ese sentido, es indispensable lograr medidas objetivas, repetibles y automáticas [9]. En el caso de tratarse de mediciones al código fuente esto podrá ser realizado sin mayores problemas a partir de la lectura de los informes de las herramientas de análisis estático del código (ver apartado 3.1.3). De esta manera se obtendrán las medidas de calidad básicas. En el caso de tratarse de otro tipo de infomación en la que se necesita un componente humano (como por ejemplo la importancia de una funcionalidad) será necesario el uso de elementos intermedios que faciliten la toma de información y, que, por lo tanto, aumenten su objetividad. En este caso, se ha optado por métodos de evaluación por pares (utilizado en [17] y descrito en el apartado 4.4.2).

5.2.2 Calcular el valor de los atributos de calidad

En este caso, una vez obtenidos las medidas básicas, estas serán persistidas en el modelo de datos (ver apartado 5.4.2). Estas medidas serán recuperadas y sobre ellas se aplicará la función de medición. Para que el modelo de medición sea dinámico, es necesario que las funciones de medición también se encuentren incluidas en el modelo de datos. De esta manera, una medida derivada de otras será el resultado de la obtención de medidas básicas persistidas en base de datos aplicadas a una función contenida en la base de datos. El resultado también será persistido en el modelo de datos.

5.2.3 Calcular el valor de las características y subcaracterísticas

De la misma manera que se han calculado las medidas derivadas del apartado anterior, se calculan las subcaracterísticas y características de calidad. Estoy continúa así hasta llegar a la medida final (en el caso de esta Tesis Doctoral son la mantenibilidad y el valor). Junto con las funciones también se persisten otras características de configuración, como son los pesos y umbrales de las funciones.

5.2.4 Implementación del cálculo del valor

El valor puede definirse como la cuantificación de la importancia que un determinado artefacto o tarea tiene para todos los implicados (o stakeholders) en un sistema. La posibilidad de cuantificar este valor hace que puedan identificarse las partes del sistema que deben ser mejoradas, atendiendo al valor que pueda producir esa mejora.

El cálculo de valor se realiza en KEMIS a través de unos atributos de entrada que proporciona el usuario. Estos atributos de entrada se corresponden con los diferentes indicadores que influirán en el cálculo del valor de los diferentes elementos del código fuente.

Estos indicadores alimentarán al modelo de medición del valor, que será el utilizado para realizar los cálculos. En la siguiente tabla puede observarse el modelo de medición del valor:

Como puede observarse, el valor (ROI) es calculado a través de la composición de 5 características. Estas características pueden ser calculadas mediante el análisis del código fuente, de históricos o mediante el juicio de un experto o de un implicado. En secciones posteriores se presenta la manera en la

que se calcula cada uno de estos indicadores. Una vez que se tienen calculados estos indicadores, se realiza la composición de las características para determinar el valor que tendrá el artefacto.

	Características	Subcaracterísticas
ROI	Mantenibilidad	Modelo de medición de la mantenibilidad del entorno KEMIS
	Posibilidad de cambio	Función de medición estadística de cambios históricos.
	Importancia relativa	Informe obtenido de los implicados
	Urgencia	Informe obtenido de los implicados
	Coste total de introducir la mejora	Coste de realizar la mejora
		Coste de actualizar las pruebas y la documentación

Tabla 5-1. Relación de indicadores de valor

5.2.5 Construcción de los indicadores en KEMIS

Como se ha indicado en el punto anterior, el valor es calculado a través de una serie de indicadores. Estos indicadores pueden ser extraídos directamente desde el código, del juicio de un experto o de históricos. El cálculo de los valores para cada proyecto será recibido por la aplicación en formato XML, que será analizado para extraer los datos necesarios para el cálculo del valor de los diferentes artefactos.

A continuación se presentan los indicadores utilizados en el cálculo del valor para la herramienta KEMIS:

5.2.5.1 Importancia relativa

No todos los componentes del producto tienen la misma importancia relativa respecto a los demás. Tampoco todos los implicados le otorgan el mismo grado de importancia a cada componente. Por lo tanto, esta información puede ser extraída de los implicados. Con esta medida se pretende conseguir una cuantificación de la importancia de los componentes desde el punto de vista de los

implicados. Para la extracción de esta información de los implicados, es necesario que estos completen un XML. Este XML seguirá la siguiente estructura:

```
<xs:element name="project" msdata:IsDataSet="true"
msdata:UseCurrentLocale="true">
<xs:complexType>
<xs:choice minOccurs="0" maxOccurs="unbounded">
<xs:element name="functionality">
<xs:complexType>
<xs:sequence>
<xs:element name="id" type="xs:string" minOccurs="0" />
<xs:element name="name" type="xs:string" minOccurs="0"
/>
<xs:element name="value" type="xs:string" minOccurs="0"
/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:choice>
</xs:complexType>
</xs:element>
</xs:schema>
```

Figura 5-6. Esquema del informe de importancia

En la Figura 5-7 puede verse un ejemplo de un informe de importancia. Como puede observarse, se define la importancia en función de las funcionalidades. Estas funcionalidades pueden estar compuestas por cualquier unidad o conjunto de unidades de código, ya sean módulos, paquetes o clases. Cada una de las funcionalidades tendrá un identificador y un nombre asociado, además de un valor que indicará la importancia relativa de la funcionalidad respecto al proyecto completo que se está analizando.

Para una correcta definición de la importancia relativa se ha realizado una herramienta que automatiza su cálculo. Se utiliza una técnica denominada Proceso Analítico Jerárquico o AHP por sus siglas en inglés. De esta manera se logran los siguientes objetivos:

- Mejor comunicación respecto de los diferentes implicados
- Mayor objetividad
- Simplicidad en la realización de las comparaciones.

```
<?xml version='1.0' encoding='UTF-8'?>
<project>
	<functionality>
		<id>0</id>
		<name>Gestión de nóminas</name>
		<value>0,14</value>
	</functionality>
	<functionality>
		<id>1</id>
		<name>Gestión de usuario</name>
		<value>0,53</value>
	</functionality>
</project>
```

Figura 5-7. Ejemplo de informe de importancia

5.2.5.2 Urgencia

No todos los componentes del producto tienen la misma urgencia de ser mejorados. Tampoco es en todo momento esta información será la misma, sino que se espera que cambie con el paso del tiempo. Con esta medida se pretende obtener una cuantificación de la urgencia de los componentes desde el punto de vista de los implicados.

```
<xs:element name="project" msdata:IsDataSet="true"
msdata:UseCurrentLocale="true">
<xs:complexType>
<xs:choice minOccurs="0" maxOccurs="unbounded">
<xs:element name="functionality">
<xs:complexType>
<xs:sequence>
<xs:element name="id" type="xs:string" minOccurs="0" />
<xs:element name="name" type="xs:string" minOccurs="0"
/>
<xs:element name="value" type="xs:string" minOccurs="0"
/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:choice>
</xs:complexType>
</xs:element>
</xs:schema>
```

Figura 5-8. Esquema de informe de urgencia

De igual manera que para la importancia relativa, la urgencia se le indica a la aplicación a través de un fichero XML de entrada, que tendrá el esquema de la Figura 5-8. Los informes de importancia relativa y de urgencia siguen la misma estructura, si bien sus valores serán diferentes en función de las necesidades identificadas por los implicados. Existe la posibilidad de que la urgencia cambie con mayor velocidad en el tiempo por lo que no sea recomendable realizar un proceso analítico jerárquico.

5.2.5.3 Posibilidad de cambio

No todos los componentes tienen la misma posibilidad de que un cambio les afecte a un componente del producto. Si se estima mejorar una parte del producto que no va a cambiar o que tiene una muy baja posibilidad de que lo haga no será beneficiosa la mejora. El retorno de invertir en un componente que no cambia a lo largo del tiempo es muy bajo. Con la extracción de esta medida se pretende identificar cuáles son los componentes que tienen más probabilidad de cambiar.

Para calcular la posibilidad de cambio se hace uso de la herramienta StatSCM [182], la cual permite realizar el cálculo de líneas agregadas, modificadas y borradas por unidad de tiempo (ver apartado 4.4.4 para conocer la medición de cambio propuesta) .El informe generado por la herramienta es analizado por KEMIS y a través de él se consigue obtener la posibilidad de cambio que tendrá cada clase

5.2.5.4 Costes

La modificación de todos los componentes no tiene el mismo coste. Cuanto más costosa sea la modificación de un componente, menos valor tendrá su modificación. Es por esto que el coste debe influir en el cálculo del valor de un componente. Se distinguen dos tipos de coste:

- Coste de realizar la mejora: la propia realización de la mejora del producto para la mejora de su calidad va a tener un coste asociado.
- Coste de actualizar las pruebas: además del coste de realizar la mejora, hay que tener en cuenta el coste que tendría actualizar las pruebas que se realicen sobre el componente.

Para realizar el cálculo de los costes se tomará la complejidad ciclomática de cada clase, buscando la correspondencia en un fichero de configuración de costes. Este fichero de configuración seguirá el esquema definido a continuación:

```
<xs:element name="project" msdata:IsDataSet="true"
msdata:UseCurrentLocale="true">
<xs:complexType>
<xs:choice minOccurs="0" maxOccurs="unbounded">
<xs:element name="costs">
<xs:complexType>
<xs:sequence>
<xs:element name="cc" minOccurs="0"
maxOccurs="unbounded">
<xs:complexType>
<xs:sequence>
<xs:element name="cost" type="xs:string" minOccurs="0"
msdata:Ordinal="0" />
</xs:sequence>
<xs:attribute name="max" type="xs:string" />
<xs:attribute name="min" type="xs:string" />
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:choice>
</xs:complexType>
</xs:element>
</xs:schema>
```

Figura 5-9. Esquema del informe de costes

En la Figura 5-10 puede verse un ejemplo de un informe de costes:

```
<?xml version='1.0' encoding='UTF-8'?>
<project>
<costs>
      <cc max="10"><cost>20</cost></cc>
      <cc min="10" max="25"><cost>12</cost></cc>
      <cc min="25" max="40"><cost>25</cost></cc>
</costs>
</project>
```

Figura 5-10. Ejemplo de informe de costes

5.2.5.5 Mantenibilidad

La mantenibilidad se define como el esfuerzo necesario para adaptarse a las nuevas especificaciones y requisitos del software. Esta característica forma parte del modelo de medición del valor debido a que una modificación que aumente en mayor medida la mantenibilidad del producto tendrá más valor para la calidad del

producto software que una modificación que aumente en menor medida la mantenibilidad.

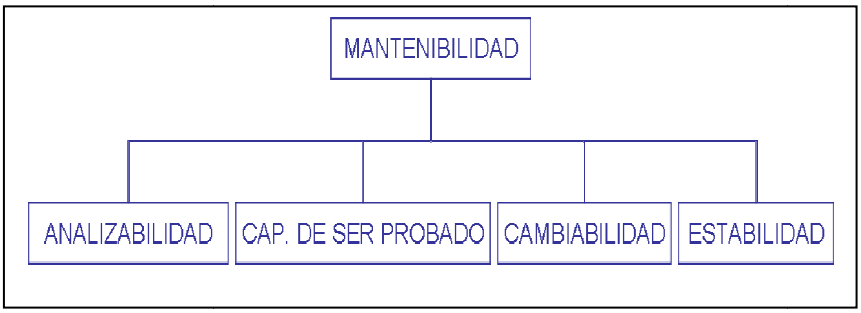


Figura 5-11. Subcaracterísticas de la mantenibilidad

Para el caso de la mantenibilidad se ha utilizado el modelo de medición de la mantenibilidad implementado en KEMIS. El valor de la mantenibilidad es calculado a nivel de módulo en la ejecución de KEMIS, y posteriormente almacenado en base de datos. De esta manera lo tendremos disponible para ser incorporado a cada clase, que heredarán sus valores en función del módulo al que pertenezcan.

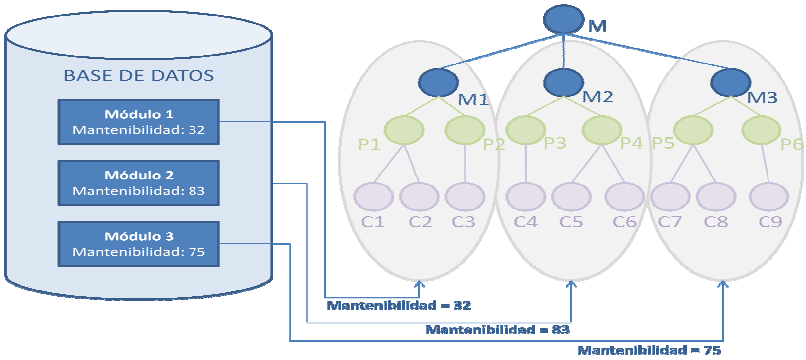


Figura 5-12.Cálculo de la mantenibilidad

5.2.5.6 Tipos de estructura de proyectos en KEMIS

Un proyecto puede estructurarse de diferentes maneras en función del punto de vista en el que se observe. Así, la estructura variará dependiendo de cuál sea ese punto de vista elegido. En el caso de KEMIS, se ha identificado la necesidad de conocer dos tipos de estructuras, una estructura anidada en función de los módulos, y otra anidada en función de los paquetes.

5.2.5.6.1 Estructura modular

La estructura en función de módulos pretende conocer la jerarquía existente entre los diferentes módulos del proyecto. Dentro de estos módulos, se identifican diferentes paquetes y clases. En el caso particular de los paquetes, es posible que un mismo paquete aparezca en dos módulos al mismo tiempo, si bien las clases no serán las mismas. Esta estructura puede observarse en la Figura 5-13.

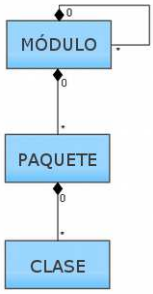


Figura 5-13. Estructura modular (UML)

Un ejemplo de una estructura de este tipo puede verse en la Figura 5-14:

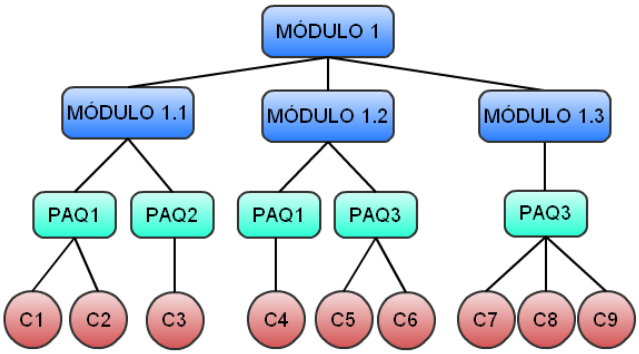


Figura 5-14. Ejemplo de estructura modular

Esta estructura presenta los proyectos en función de sus funcionalidades, agrupando las clases y los paquetes respecto a los módulos a los que pertenecen. Esta estructura permite trabajar con los proyectos de la misma manera que lo hace Maven. De esta manera se conoce la estructura que tiene cada uno de los módulos del proyecto. Hasta este momento, KEMIS-CORE solamente trabajaba hasta nivel de módulo. Con esta extensión, puede trabajarse a niveles más bajos, lo que nos ayudará a establecer el valor y las refactorizaciones posteriores.

5.2.5.6.2 Estructura de paquetes

De igual manera, los proyectos pueden clasificarse en función de sus paquetes. Así, pueden identificarse los paquetes que tiene el proyecto, así como las clases asociadas a cada uno. En la Figura 5-15 y la Figura 5-16 puede verse la estructura.

Figura 5-15. Estructura de paquetes (UML)

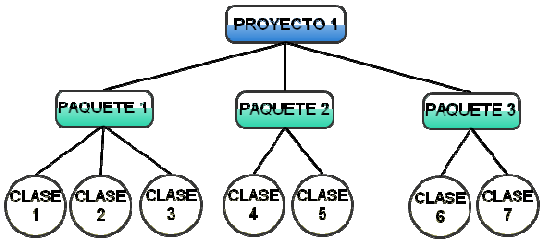


Figura 5-16. Ejemplo de estructura de paquetes

Esta estructura es típica de Java, en la cual los proyectos constan de diferentes paquetes, cada uno de ellos con sus clases correspondientes. Pero obvia los módulos del proyecto, lo que permite asignar valores a un determinado paquete y conocer de esta manera el que poseerá cada una de sus clases.

5.2.5.6.3 Extracción de la estructura del proyecto

La estructura del proyecto obtenida se representa mediante un fichero UML. Se obtienen dos ficheros XML, uno para la estructura modular y otro para la estructura de paquetes. Estos dos ficheros siguen el siguiente esquema:

```
<?xml version="1.0" encoding="UTF-8"?>
   •  <xs:schema
      xmlns:xs="http://www.w3.org/2001/XMLSchema">
      <xs:element name="project">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="module"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="package">
            <xs:complexType>
                  <xs:sequence>
      <xs:element ref="id"/>
      <xs:element ref="name"/>
      <xs:element ref="class" maxOccurs="unbounded"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="module">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="id"/>
                        <xs:element ref="name"/>
                        <xs:choice minOccurs="0">
                              <xs:element
ref="package" maxOccurs="unbounded"/>
                              <xs:sequence>
      <xs:element ref="module" maxOccurs="unbounded"/>
      <xs:element ref="package" minOccurs="0"/>
                              </xs:sequence>
                        </xs:choice>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="id" type="xs:string"/>
      <xs:element name="class">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="id"/>
                        <xs:element ref="name"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
</xs:schema>
```

Figura 5-17. XML-Schema de la estructura modular

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema">
	<xs:element name="project">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="package"
maxOccurs="unbounded"/>
			</xs:sequence>
		</xs:complexType>
	</xs:element>
	<xs:element name="package">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="id"/>
				<xs:element ref="name"/>
				<xs:element ref="class"
maxOccurs="unbounded"/>
			</xs:sequence>
		</xs:complexType>
	</xs:element>
	<xs:element name="name" type="xs:string"/>
	<xs:element name="id" type="xs:string"/>
	<xs:element name="class">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="id"/>
				<xs:element ref="name"/>
			</xs:sequence>
		</xs:complexType>
	</xs:element>
</xs:schema>
```

Figura 5-18. XML-Schema de la estructura de paquetes

5.2.6 Integración con la herramienta KEMIS

La extracción de la estructura del proyecto se integra con KEMIS, tanto en la parte del núcleo (KEMIS-CORE) como en el plugin de KEMIS para Maven 2 (MAVEN-KEMIS-PLUGIN). En la siguiente figura se presenta como se introduce la ejecución dentro del ciclo de vida de ejecución de KEMIS.

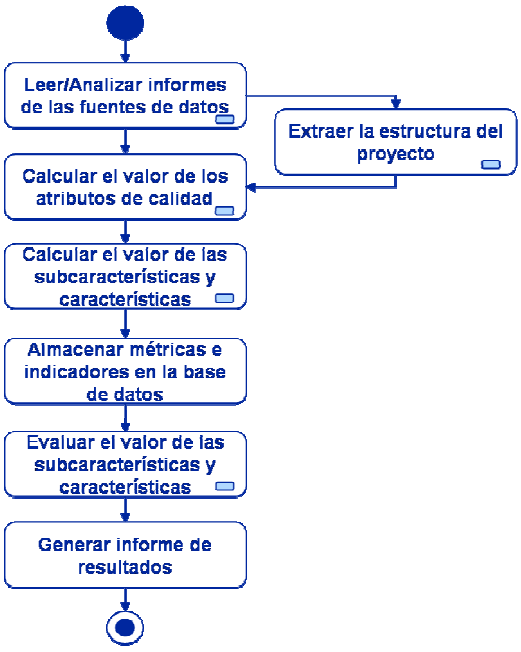


Figura 5-19. Integración de la extracción de la estructura en el ciclo de vida de KEMIS

5.2.6.1 Pasos para el cálculo del valor

Para realizar el cálculo del valor, KEMIS realiza los siguientes pasos:

1. Extracción de la estructura del proyecto: a través del análisis del proyecto, KEMIS obtiene la estructura del proyecto, ya sea la estructura modular o la estructura de paquetes. Con esta estructura crea un fichero XML (descrito en el apartado).
2. Lectura de los XML de importancia e urgencia.
3. Mapeo de las funcionalidades indicadas en los informes de importancia e urgencia con las diferentes unidades de código. Para realizar este paso, es necesario que se proporcione un fichero XML en el que se produzca el mapeo de cada funcionalidad con las unidades de código que contiene. El XML debe seguir el siguiente esquema:

```
<xs:element name="project" msdata:IsDataSet="true"
msdata:UseCurrentLocale="true">
<xs:complexType>
<xs:choice minOccurs="0" maxOccurs="unbounded">
<xs:element name="relation">
<xs:complexType>
<xs:sequence>
<xs:element name="codeunits" minOccurs="0"
maxOccurs="unbounded">
<xs:complexType>
<xs:sequence>
<xs:element name="id" nillable="true" minOccurs="0"
maxOccurs="unbounded">
<xs:complexType>
<xs:simpleContent msdata:ColumnName="id_Text"
msdata:Ordinal="0">
<xs:extension base="xs:string">
</xs:extension>
</xs:simpleContent>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
<xs:attribute name="functionality" type="xs:string" />
<xs:attribute name="id" type="xs:string" />
</xs:complexType>
</xs:element>
</xs:choice>
</xs:complexType>
</xs:element>
</xs:schema>
```

Figura 5-20. Esquema del mapeo de funcionalidades con unidades de código

Un ejemplo de un fichero de este tipo sería el siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<project>
<relation functionality="Gestión de nóminas" id="0">
	<codeunits>
		<id>0</id>
	</codeunits>
</relation>
<relation functionality="Gestión de usuario" id="1">
	<codeunits>
		<id>3</id>
	</codeunits>
</relation>
<relation functionality="Facturación" id="2">
	<codeunits>
		<id>2</id>
		<id>4</id>
	</codeunits>
</relation>
<relation functionality="Alta de usuario" id="3">
	<codeunits>
		<id>6</id>
	</codeunits>
</relation>
</project>
```

Figura 5-21. Ejemplo del mapeo de funcionalidades con unidades de código

4. Asignación de los valores de importancia y urgencia a cada una de las unidades de código.
5. Lectura y análisis del informe de posibilidad de cambio (StatSCM). Identificación de las clases modificadas.
6. Asignación de los valores de posibilidad de cambio a cada una de las clases.
7. Lectura del informe generado por JavaNCSS para identificar la complejidad ciclomática de cada clase.
8. Asignación de los valores de coste a cada clase, en función de su complejidad ciclomática.
9. Lectura de mantenibilidad de cada módulo desde la base de datos.
10. Asignación de la mantenibilidad a cada clase, utilizando la mantenibilidad del módulo.
11. Cálculo del valor para cada clase, siguiendo la fórmula descrita en el apartado 4.3.1 de esta Tesis Doctoral.
12. Cálculo del valor para las unidades de código de grado superior (paquetes, módulos), en función de la adición del valor de cada una de sus clases.

En la Figura 5-22 pueden observarse gráficamente los pasos que se siguen para el cálculo del valor.

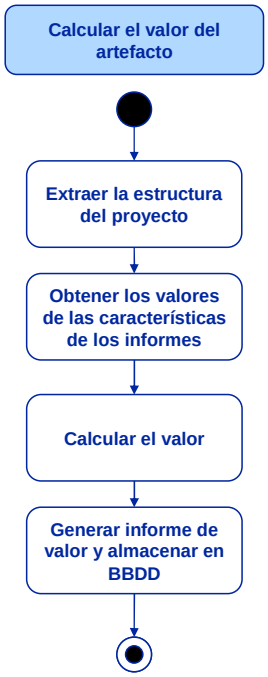


Figura 5-22. Cálculo del valor

5.3 Vista de Desarrollo

La siguiente figura muestra la descomposición de la arquitectura de KEMIS en sus diferentes componentes:

Cada uno de estos componentes, se encarga de:

- KEMIS-WEB: interfaz orientado a un usuario para la consulta de los resultados obtenidos por KEMIS.
- KEMIS-CORE, Motor de Medición de Calidad: módulo que se encarga de generar medidas de calidad e indicadores de calidad para las diferentes características y subcaracterísticas de calidad que refleja la ISO/IEC 9126 y 25000.
- KEMIS-CORE, Generador de Informes: módulo encargado de la generación de informes de calidad, que permitirá desde la generación de gráficos hasta la generación del informe completo de la calidad de producto.

- KEMIS-CORE, Módulo de Acceso a Datos: esté módulo permite la independencia de KEMIS con el Sistema Gestor de Base de Datos, permitiendo desde un principio el soporte para H2, MySQL y Oracle.
- Servidor, Base de Datos: base de datos del sistema.

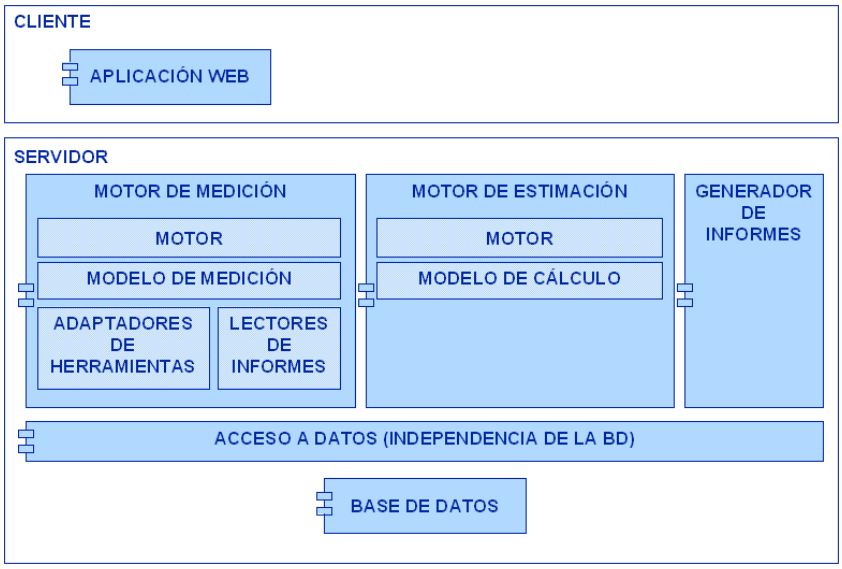


Figura 5-23. Arquitectura de KEMIS por componentes

A continuación se muestra el diseño de la implementación de KEMIS. Este dispone de:

- Un proceso de construcción definido y automatizado (p. ej. Con herramientas como Maven, Final Builder, etc.)
- Un proceso de integración continua implantado (con herramientas como Cruise Control, Continuum, etc.).
- Un sistema de control de versiones (con herramientas como CVS, Subversión, SourceSafe, etc.)

La Figura 5-24 muestra las tecnologías y módulos de bajo nivel para dicha arquitectura. De manera resumida el modelo de funcionamiento es el siguiente:

- Según la periodicidad definida para la herramienta de integración continua (p. Ej. Cotinuum), se ejecuta una construcción y una auditoría de calidad. Este proceso puede ser diario o semanal y puede que no sea necesaria durante cada ejecución de la construcción del software.

  Durante el proceso de construcción:

- Se descargan los artefactos que forman el producto, desde el repositorio de versiones (p. Ej. Subversion)
- Se construye el software.
- Se ejecutan pruebas.
- Se ejecutan las herramientas que generan métricas de calidad de bajo nivel, que generalmente producen ficheros xml.
- Posteriormente, la información de los ficheros xml, se recoge y se almacena en una base de datos.
- Los datos almacenados se utilizan para generar informes, usando por ejemplo Jasper Reports.
- De la misma manera, a partir del cliente Web es posible obtener información de los resultados y realizar una ejecución remota de KEMIS, para evaluar un producto.

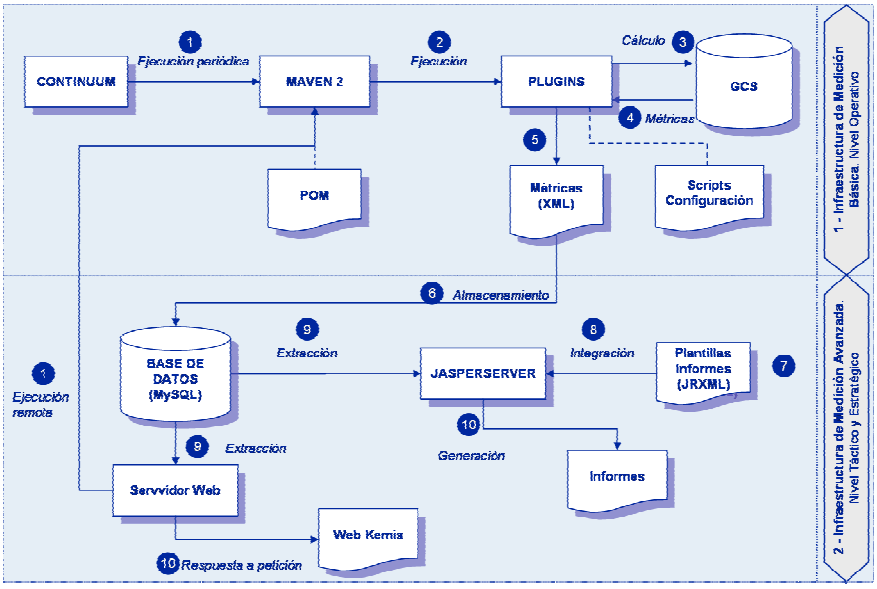


Figura 5-24 Micro-arquitectura del motor de medición y estimación de calidad

De esta manera, junto con la construcción del software se puede obtener un informe de calidad que proporcione visibilidad de la calidad del producto y facilite la toma de decisiones, como pueda ser la liberación de una versión.

5.4 Vista Física

La arquitectura propuesta sigue el paradigma MVC (Modelo Vista.Controlador). La siguiente figura, diagrama de despliegue, muestra cómo se distribuye el sistema y las interconexiones físicas entre los módulos del sistema con sistemas externos. Se verá en los siguientes gráficos las diferentes vistas de despliegue. En el caso de la Figura 5-25, vemos las capas en las que se encuentra dividida. En la Figura 5-26 se detallan los componentes internos de alto nivel de KEMIS-CORE.

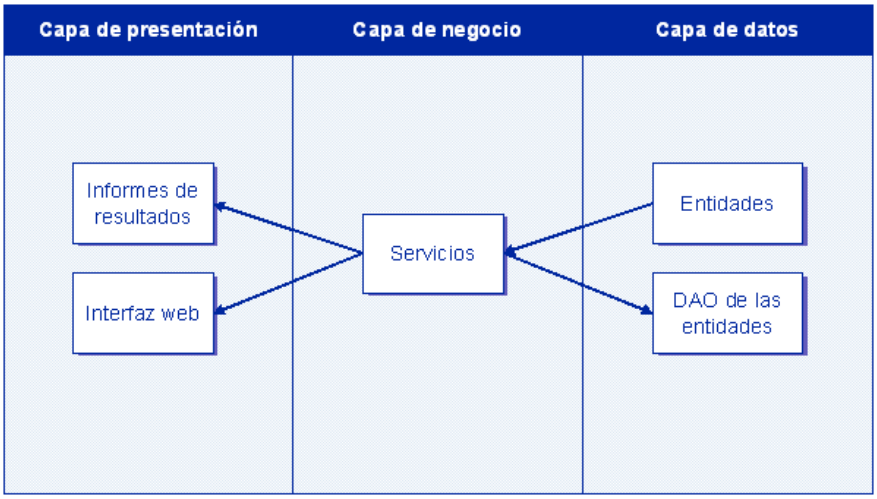


Figura 5-25. Las capas en la arquitectura de KEMIS

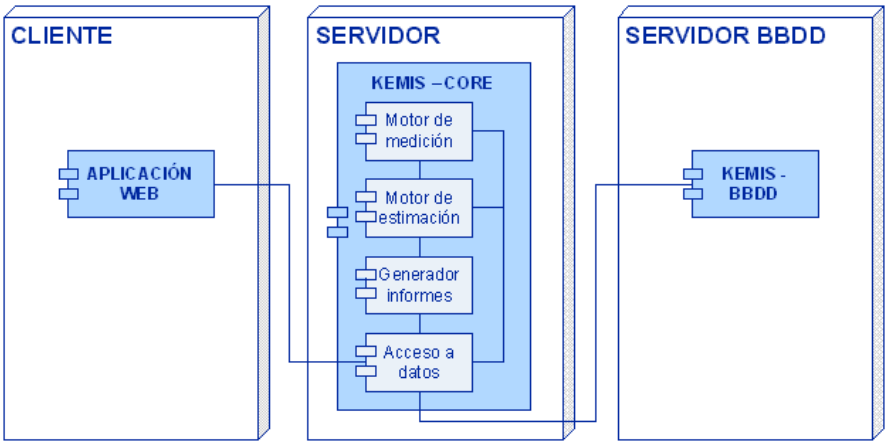


Figura 5-26. Detalle de los componentes internos

5.4.1 Especificaciones de construcción

5.4.1.1 Gestión de la configuración

La aplicación reside en un repositorio de versiones. Las ventajas que aporta un repositorio son múltiples, desde poder trabajar concurrentemente varios desarrolladores, facilidad al versionar, mantenimiento de versiones liberadas, etc.

Todos los ficheros que se generen a partir de otros no deberían ser versionados. Ejemplo son los archivos *.class o el distribuible de la aplicación (ya sea en formato zip, jar o war). Cuando se desee liberar una nueva versión se hará siguiendo las políticas de gestión de la configuración corporativa. Se versionará sobre el repositorio, y en ningún caso creando directorios de versión.

5.4.1.2 Script de construcción

Se utilizará la herramienta de construcción Maven 2.0. El Script de construcción no sólo realizará las labores de compilación, sino también ejecutará las pruebas unitarias, generará el script necesario para la creación de la base de datos y realizará el empaquetado del distribuible.

5.4.1.3 Integración continua

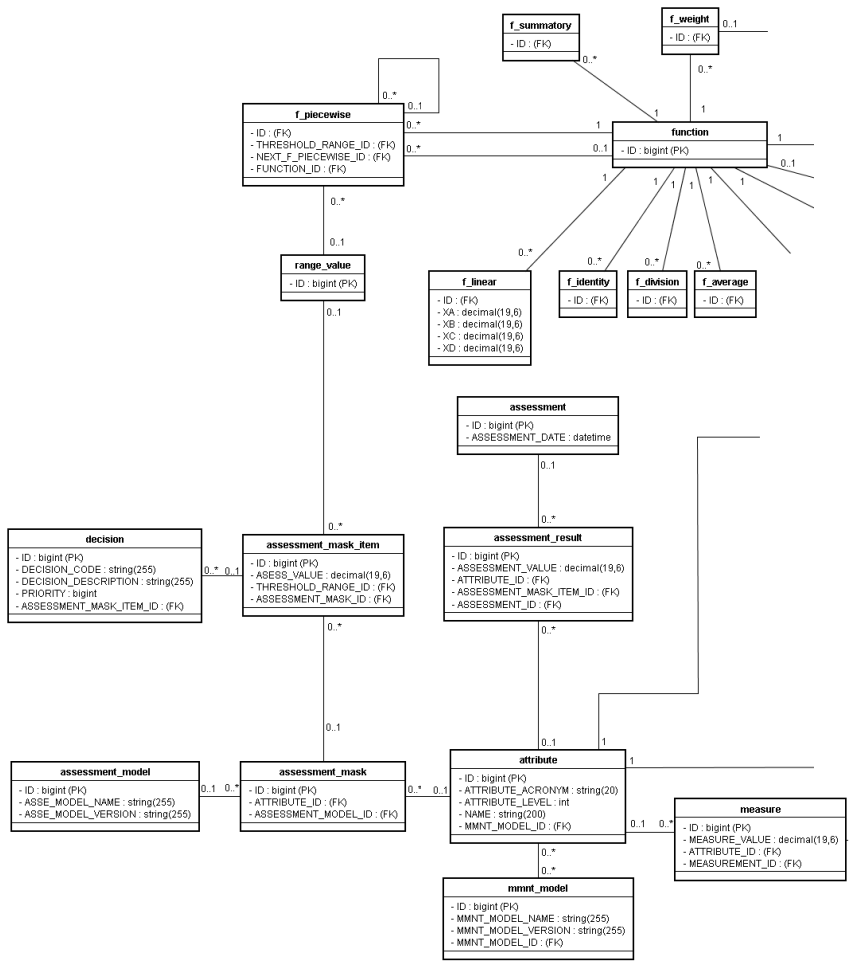
Una vez se tenga el script de construcción del sistema, se añadirá este proyecto a un servidor de integración continua, de tal forma que cualquier miembro del equipo tenga a su disposición información actualizada del progreso del proyecto, así como acceso a los últimos distribuibles y versiones liberadas.

5.4.2 Diseño físico de los datos

Aunque no se entrará en el detalle de la descripción del modelo de datos empleado para la herramienta KEMIS, se recalcarán sus principales características. Está compuesto por 34 tablas, destacándose un conjunto de tablas principales que se corresponden con la lógica de negocio implementada en la herramienta. De esta manera, se pueden ver las siguientes tablas:

- Management_model: table principal y relacionadas que representa a los modelos de medición implementados.
- Attribute: atributos básicos o derivados que conforman un modelo de medición.
- Function: es la tabla principal de un conjunto de tablas que especifican las diferentes funciones de medición de un modelo de medición particular. Se relaciona con otras entidades del modelo de datos para generar medidas derivadas.

- Entity: unidades de código de un proyecto software, incluido el mismo proyecto software.
- Value: valor de una entidad obtenida a partir del modelo de medición del valor. El modelo de medición del valor se encuentra configurado junto con el modelo de medición de la mantenibilidad en el modelo de datos actual.
- Tool: representa todas las herramientas que serán utilizadas para realizar los diferentes análisis al código fuente.
- Reader: lector de informes TXT o XML generados por las herramientas que se ejecutarán al analizar los proyectos.
- Assessment: resultados de la evaluación de un proyecto con un modelo de medición
- Measure: valor de un atributo durante una evaluación
- Measurement_weight: peso de los atributos al aplicar un modelo de medición en un proyecto. Los pesos están relacionados con la evaluación.
- Report: resultados de evaluación de un proyecto relacionado con un modelo de medición


f_summatory
- ID : (FK)
f_weight
- ID : (FK)
f_piecewise
- ID : (FK)
- THRESHOLD_RANGE_ID : (FK)
- NEXT_F_PIECEWISE_ID : (FK)
- FUNCTION_ID : (FK)
function
- ID : bigint (PK)
range_value
- ID : bigint (PK)
f_linear
- ID : (FK)
- XA : decimal(19,6)
- XB : decimal(19,6)
- XC : decimal(19,6)
- XD : decimal(19,6)
f_identity
- ID : (FK)
f_division
- ID : (FK)
f_average
- ID : (FK)
assessment
- ID : bigint (PK)
- ASSESSMENT_DATE : datetime
decision
- ID : bigint (PK)
- DECISION_CODE : string(255)
- DECISION_DESCRIPTION : string(255)
- PRIORITY : bigint
- ASSESSMENT_MASK_ITEM_ID : (FK)
assessment_mask_item
- ID : bigint (PK)
- ASESS_VALUE : decimal(19,6)
- THRESHOLD_RANGE_ID : (FK)
- ASSESSMENT_MASK_ID : (FK)
assessment_result
- ID : bigint (PK)
- ASSESSMENT_VALUE : decimal(19,6)
- ATTRIBUTE_ID : (FK)
- ASSESSMENT_MASK_ITEM_ID : (FK)
- ASSESSMENT_ID : (FK)
assessment_model
- ID : bigint (PK)
- ASSE_MODEL_NAME : string(255)
- ASSE_MODEL_VERSION : string(255)
assessment_mask
- ID : bigint (PK)
- ATTRIBUTE_ID : (FK)
- ASSESSMENT_MODEL_ID : (FK)
attribute
- ID : bigint (PK)
- ATTRIBUTE_ACRONYM : string(20)
- ATTRIBUTE_LEVEL : int
- NAME : string(200)
- MMNT_MODEL_ID : (FK)
measure
- ID : bigint (PK)
- MEASURE_VALUE : decimal(19,6)
- ATTRIBUTE_ID : (FK)
- MEASUREMENT_ID : (FK)
mmnt_model
- ID : bigint (PK)
- MMNT_MODEL_NAME : string(255)
- MMNT_MODEL_VERSION : string(255)
- MMNT_MODEL_ID : (FK)

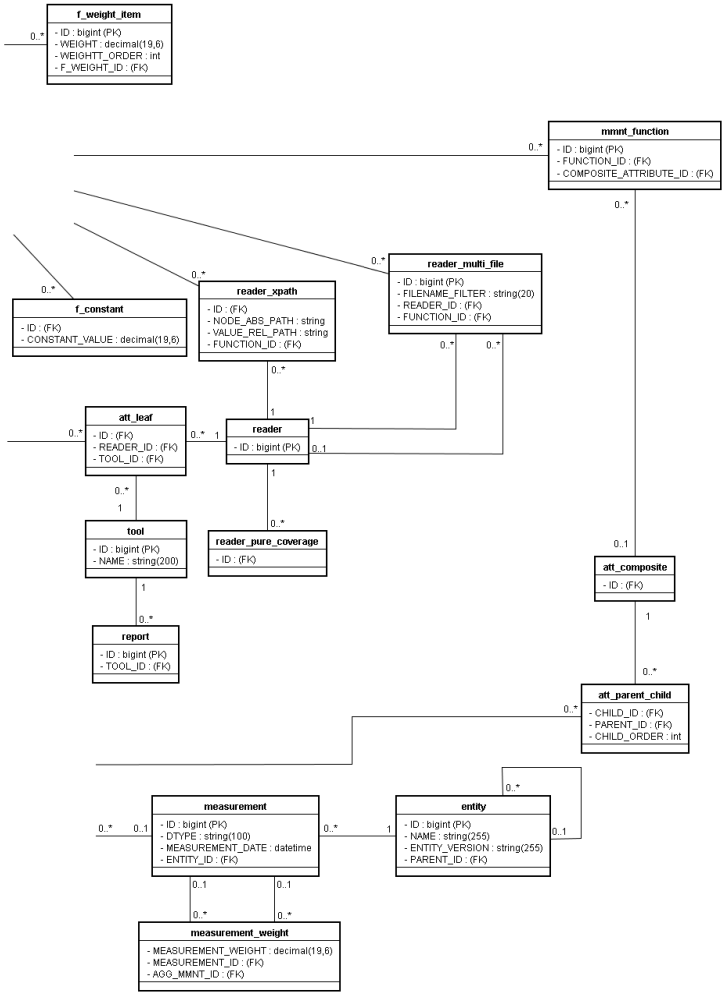


Figura 5-27. Diseño físico de datos en KEMIS

Validación

En este apartado se describirán las validaciones realizadas a los modelos de medición detallados; que se han abordado siguiendo el método de resolución y validación descrito en el apartado 1.5 de esta Tesis Doctoral. Así mismo se resumirán las verificaciones realizadas al entorno KEMIS. Se analizará un caso de éxito de la implantación de KEMIS en un importante proyecto de desarrollo de una empresa española que tiene como objetivos prestar servicios de tecnología de la información y telecomunicaciones. Finalmente, se reseñan las auditorías técnicas realizadas en el marco del proyecto KEMIS de I+D+i (IDI-20090175) financiado por el Centro para el Desarrollo Tecnológico Industrial (CDTI).

6.1 Sobre la Validación de Técnicas Basadas en Valor

Tal y como se comenta en [17] la validación de las técnicas basadas en valor están condicionadas por diferentes aspectos. Para ello se han revisado y clasificado las validaciones en trabajos previos. En el ámbito de la ISBV, los trabajos presentan fundamentalmente dos tipos de validaciones de técnicas basadas en valor.

Revisando las validaciones realizadas en el ámbito del valor, estas son validaciones de laboratorio, comparando los resultados con y sin el uso del valor. En un primer momento se planificó realizar grupos de estudiantes y realizar experimentaciones controladas, con y sin la aplicación de los modelos de medición en un proyecto de desarrollo software. Este tipo de validaciones puede verse en [74] o en [81]. Esta idea se rechazó debido a que sería necesaria una gran cantidad de tiempo e iteraciones para obtener los datos suficientes. Así mismo, otra de las amenazas de validación era la ausencia de clientes reales. Esto podía disminuir la presión en los plazos y en la cantidad de funcionalidad entregada (los cuales muchas veces son las principales causas de disminución de la calidad en el desarrollo).

El segundo tipo de validación es aquel que en lugar de utilizar diversos grupos, únicamente se ocupan de asegurar que las tareas no priorizadas dejan de consumir recursos. Por ejemplo, en [78] se demuestra que la trazabilidad basada en valor reduce en un 35% el esfuerzo a realizar con respecto al trazado total de los artefactos, es decir, se demuestra que el esfuerzo realizado en trazabilidad disminuye, pero no se evalúa de ningún modo el impacto que esta reducción de esfuerzo tiene.

6.2 Revisión de las Debilidades en Otros Planteamientos

Tomando la enumeración de [17] y tal como se indica allí, existen dudas razonables de la aplicabilidad de las técnicas a entornos reales de desarrollo y mantenimiento de software. Las carencias más relevantes de las validaciones asociadas con el valor son:

- Inexistencia de implicados de negocio reales. La ISBV tiene por objetivo alinear el negocio y las prácticas de IS. En ninguna validación planteada se extraen los indicadores que guían las prácticas de implicados reales. La aplicación de técnicas en grupos controlados de alumnos no son el mejor escenario para evaluar las directrices de valor, dado que éstas no existen o son artificialmente supuestas.
- Carencia de evaluación de las implicaciones de negocio. La no aplicación de una determinada práctica de IS tendrá posiblemente un impacto negativo sobre la calidad del producto. La evaluación de dicho impacto es fundamental en de cara a verificar propuestas basadas en valor.
- Validación mediante experimentos cortos. Para poder evaluar la validez de la alineación del negocio de los sistemas de información a largo plazo, es necesario analizar series temporales de varios meses y varios proyectos. Esto es debido a que los objetivos de negocio suelen situarse a medio o largo plazo. Por tanto, su cumplimiento no puede evaluarse en un experimento de horas de duración. Indisponibilidad de datos de mantenimiento y explotación. La evaluación de la priorización por valor de las técnicas destinadas a elaborar software de calidad debe ser contrastada teniendo en cuenta (entre otros aspectos) el comportamiento de dicho software a lo largo de periodos de mantenimiento. La disponibilidad de dichos datos hace necesario la implantación y recogida de datos durante largos períodos de tiempo, por lo que es difícilmente simulable en un experimento.

Teniendo en cuenta estos aspectos y tomando como ejemplo validaciones realizadas en otros modelos de medición, como, por ejemplo en [60][61][62] o [63] es necesario realizar validaciones sobre proyectos reales. Por otro lado, es imprescindible contrastar la evolución de indicadores del negocio como pueden ser la cantidad y tiempo de respuesta en las incidencias relacionadas con el desarrollo y mantenimiento del producto software. Debido a ello se ha planteado la realización de una validación principal tomada de la implantación y uso de la herramienta KEMIS en una gran empresa española. En el siguiente apartado se detalla el caso de estudio.

6.3 Descripción del Caso de Estudio

El entorno KEMIS se ha implantado en una gran empresa española. El sistema principal de esta empresa recibe actualmente más de 2.000.000 de transacciones semanales. El proyecto sobre el que se implanta KEMIS es uno de importantes dimensiones, desarrollado en dos lenguajes de programación: C++ y Java. El problema que presentaba la organización era la ausencia de control sobre la calidad de producto debido a:

- La falta de visibilidad provocada por el amplio número de desarrolladores presentes en el equipo: una media de 65 trabajadores, 20 de ellos externos.
- La heterogeneidad de los desarrolladores: los desarrolladores, tenían entre 1 y 10 años de experiencia, con una media de 3 años.
- Las dimensiones del proyecto: en 1 año se incorporarían 8 nuevos módulos.
- La externalización de algunos módulos del producto: es decir, parte del desarrollo iba a realizarse en un proveedor externo.

Además, para esta empresa era vital poder realizar un control más exhaustivo sobre la parte del producto que ha sido externalizada de acuerdo a penalizaciones que se podrían imponer al proveedor, siempre y cuando el nivel de calidad no se encontrase en los límites que se pueden establecer en el Acuerdo de Nivel de Servicio (ANS).

6.3.1 Actividades realizadas: Infraestructura Básica de Medición

En una primera fase se implementa y se implanta en el proyecto de la empresa una infraestructura básica de medición. Para ello se realizan las siguientes actividades:

- Diseño de dos modelos de medición: para la parte realizada en la tecnología Java (parte del proyecto externalizada) y para la parte elaborada en la tecnología C++ (parte del proyecto desarrollado en la propia empresa). El modelo ha permitido diferentes grados de control de la calidad de producto, con el fin de exigir unos niveles de calidad más altos a la parte del proyecto desarrollada por proveedores externos.
- Selección de herramientas de software libre de obtención de métricas para los desarrollos Java y C++. A partir de sus informes se obtienen los atributos de bajo nivel que suponen la base para el cálculo del resto de atributos.

- Configuración de estas herramientas de obtención de métricas (mediante plugins ya existentes) en la herramienta de construcción usada en la empresa (MAVEN).
- Automatización de la obtención de estos informes en el entorno de Integración Continua.

6.3.2 Resultados obtenidos

Obtención de métricas (atributos base), a partir de la generación de manera automatizada y periódica de informes, que permiten fácilmente el cálculo (de manera manual) de los indicadores de calidad de más bajo nivel que ya proporcionan una visibilidad significativa de los productos en desarrollo. Algunos de estos indicadores son por ejemplo: Densidad de Código Repetido, Densidad de Complejidad Ciclomática, Ratio de Cobertura de Pruebas Unitarias, etc.

6.3.3 Infraestructura Avanzada de Medición

En una segunda fase se implementa y se implanta en el proyecto de la empresa una infraestructura avanzada de medición. Para ello se realizan las siguientes actividades:

- Diseño y desarrollo de la base de datos así como el modulo de acceso a estos datos, permitiendo así la persistencia de las mediciones obtenidas y facilitando la creación de históricos.
- Diseño y desarrollo de los módulos de lectura, medición y evaluación para obtener de manera automática los atributos base a partir de los informes de herramientas de métricas y mediante funciones aplicadas sobre esos atributos base se obtienen los atributos de más alto nivel del modelo de medición.
- Carga inicial de la base de datos con los modelos de medición definidos en la primera fase.
- Generación de un plugin de KEMIS para la herramienta de construcción MAVEN, usada en el proyecto, con el fin de integrar KEMIS dentro del proyecto y facilitar su ejecución.

6.3.3.1 Primeros resultados obtenidos

Se ha integrado KEMIS en el proyecto permitiendo así la obtención de manera automatizada y periódica de todos los indicadores de calidad de los diferentes niveles del modelo de medición y una valoración de los mismos de acuerdo a unos determinados umbrales. De esta manera se ha implementado el

modelo de medición de la mantenibilidad y el modelo de medición del valor para medir el valor de cada clase o conjunto de clases.

Debido a la completa integración de KEMIS en el proyecto, los resultados de las mediciones aparecen de manera automática en el sitio Web del proyecto, haciendo que estos resultados estén accesibles para todo el equipo proporcionando así visibilidad a todos los niveles (estratégico, táctico y operativo).

6.3.4 Objetivos iniciales conseguidos con la implantación de KEMIS

Mediante la implantación de KEMIS en el proyecto se han obtenido los siguientes objetivos:

- Automatización del proceso de medición. Se ha pasado de obtener informes de calidad del producto cada 15 días y suponiendo 2-3 días de trabajo, a informes diarios y completamente integrados en el proceso de construcción.
- Visibilidad de la calidad del producto. Todos los participantes (directivos, desarrolladores, proveedores, etc.) y a todos los niveles (operativo, táctico y estratégico) tienen acceso a información precisa y actualizada con el fin de facilitar la toma de decisiones.
- Persistencia. Almacenamiento de los resultados en base de datos proporcionando así una mayor facilidad para crear históricos acerca de la calidad de producto.
- Soporte de diferentes modelos de medición. Este punto en este proyecto es especialmente necesario debido a que se está desarrollando código fuente en diferentes lenguajes. Se ha logrado un control más exhaustivo en los módulos externalizados y en cuanto al desarrollo interno, se ha obtenido una mayor visibilidad de la calidad del producto..

6.3.5 Evolución de los indicadores de calidad

A continuación se muestra una serie de gráficos con la evolución que han sufrido algunos indicadores de calidad participantes en el cálculo del nivel de calidad de la mantenibilidad, una vez que se comienza a realizar un control de la calidad del producto gracias a la implantación de KEMIS.

Estos datos son referentes al modulo/ del proyecto cuyo desarrollo fue externalizado a otra empresa. Se presentan 2 mediciones, una realizada cuando KEMIS no había sido implantado y la otra un mes después de su implantación.

En las siguientes figuras se muestra la evolución de algunos de los indicadores de calidad:

- Indicador: Densidad de Violaciones de PMD.
- Valor recomendado: Inferior a 0.2.

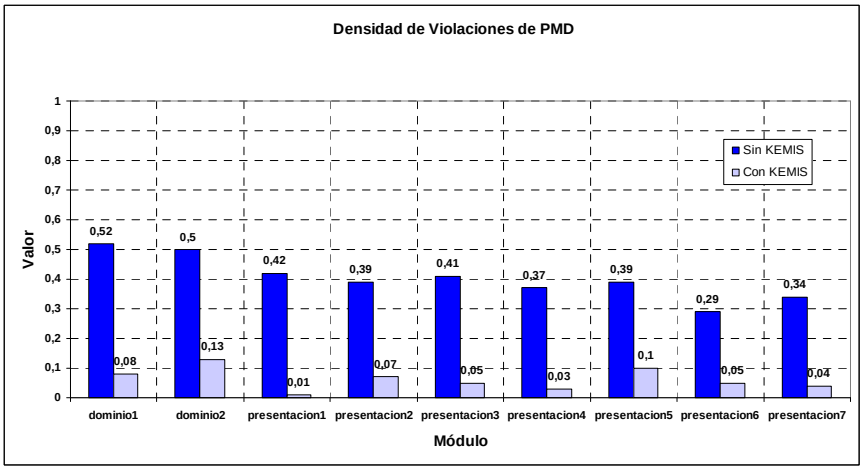


Figura 6-1. Evolución de las violaciones PMD

- Indicador: Densidad de Código Repetido.
- Valor recomendado: Inferior a 0.03.

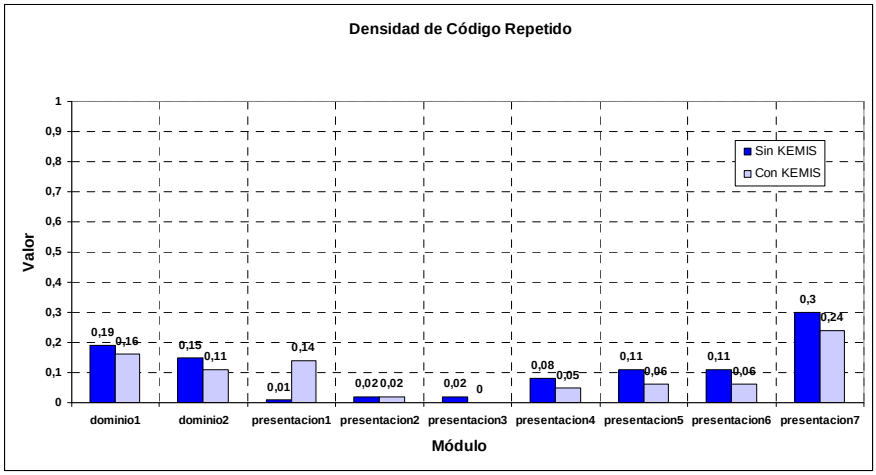


Figura 6-2. Evolución del código repetido

- Indicador: Densidad de Pruebas Unitarias.

- Valor recomendado: Superior a 0.9.

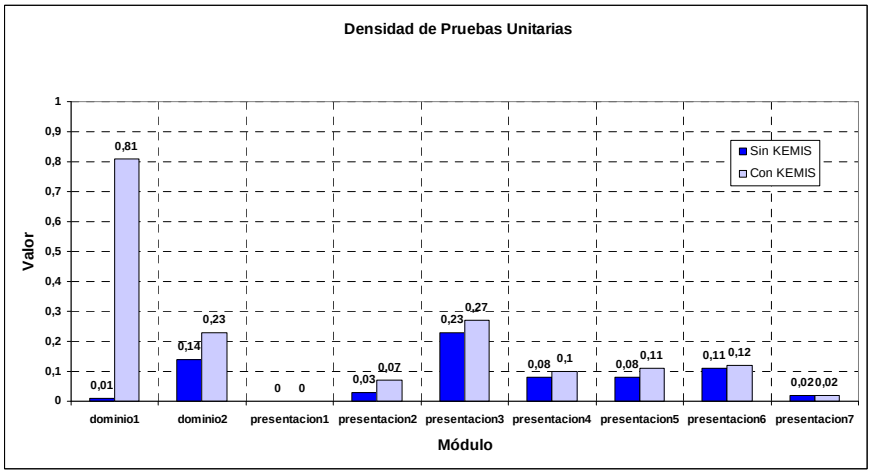


Figura 6-3. Evolución de la densidad de pruebas unitarias

Como se puede comprobar después de un mes de la implantación de KEMIS, los indicadores de calidad han mejorado de forma global en más de un 50%.

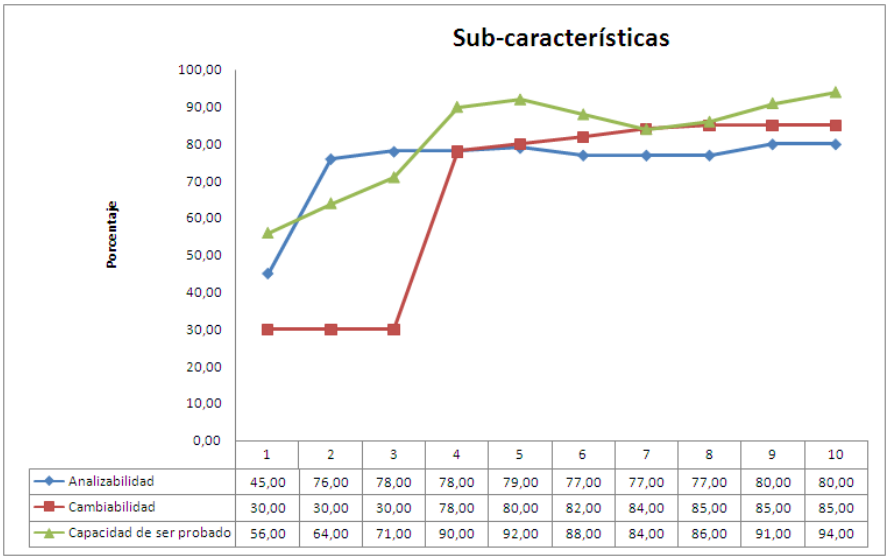
6.3.6 Análisis estadístico de los resultados al implantar KEMIS

En este apartado se analizarán los resultados en la mejora de la mantenibilidad y se lo comparará con la evolución de las incidencias detectadas durante el desarrollo de diferentes proyectos en la empresa que ha implantado KEMIS.

Como se puede observar en la Figura 6-4, durante el primer mes la analizabilidad era bastante mala (en el orden de un 30%). Esto se debía a la poca cantidad de comentarios y a la no utilización de herramientas de análisis estático del código fuente (PMD, FindBugs, CheckStyle, etc.). Se realizó un programa detallado de mejora y se planificaron formaciones para aprender a utilizar estas herramientas. Por otro lado se comunicaron los objetivos y la manera en la que el modelo de medición penalizaba la ausencia de comentarios y las violaciones de las reglas. Como resultado de estos cambios y de la comunicación constante de los resultados se pudo mejorar la analizabilidad (especialmente debido a que las herramientas como PMD o FindBugs contienen ejemplos, explicaciones y otras características didácticas para favorecer el aprendizaje de los desarrolladores.

De igual manera, la subcaracterística de cambiabilidad ha tenido un tiempo largo de mejora (unos 3 meses) hasta llegar a valores aceptables (superiores incluso al 80%) y mejores que la analizabilidad. En esos tres primeros meses se realizaron refactorizaciones para lograr romper dependencias cíclicas y disminuir la complejidad ciclomática. A partir del tercer mes los nuevos desarrollos fueron adecuándose a las nuevas características del desarrollo.

En el caso de la capacidad de ser probado el código fuente, existía ya una cultura de utilización de pruebas unitarias. En ese sentido, el problema era la cobertura de las pruebas unitarias y el tratamiento de los errores. Se consiguieron buenos resultados desde el inicio, incluso superando en los meses 4 y5 el 90%. Durante los meses 7 y 8, al incorporarse nuevos desarrollos, los recursos necesarios para la realización de pruebas se vió disminuida.



	1	2	3	4	5	6	7	8	9	10
Analizabilidad	45,00	76,00	78,00	78,00	79,00	77,00	77,00	77,00	80,00	80,00
Cambiabilidad	30,00	30,00	30,00	78,00	80,00	82,00	84,00	85,00	85,00	85,00
Capacidad de ser probado	56,00	64,00	71,00	90,00	92,00	88,00	84,00	86,00	91,00	94,00

Figura 6-4. Evolución de las mediciones. Subcaracterísticas de la mantenibilidad

Como puede verse en la Figura 6-5, la mantenibilidad alcanzó registros aceptables (mayor de 80%) a partir del cuarto mes. Y se mantuvo en la franja de entre 80% y 85% en los siguientes seis meses. También en esta gráfica se advierte una pequeña disminución de la mantenibilidad, durante los meses 6, 7 y 8 debido al inicio del desarrollo de nuevos módulos.

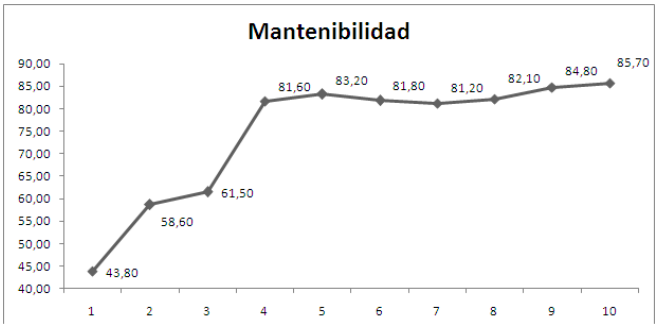


Figura 6-5. Evolución de las mediciones. Característica de mantenibilidad

Para contrastar los resultados de la mejora en la mantenibilidad, así como la priorización en las refactorizaciones de acuerdo con el valor de cada grupo de clases a ser mejorado, se han analizado aspectos del negocio. Una de las ventajas evidentes al implementar KEMIS ha sido la mejora en el nivel de conocimiento y productividad de los desarrolladores. Al final del mes quinto y décimo se realizó una encuesta donde se preguntaba el grado de satisfacción al utilizar las herramientas de calidad (KEMIS, PMD, FindBugs, CheckStyle, jUnit, etc.). Los resultados, que pueden verse en la Figura 6-6, muestran como s pasó de un 39% de desarrolladores totalmente satisfechos a un 62%. Así mismo, los desarrolladores no decididos respecto de estas buenas prácticas pasaron de un 22% a un 11%. Uno de los aspectos más importantes a destacar ha sido que el 2% de los desarrolladores estaban completamente insatisfechos con las nuevas prácticas y eso permaneció así hasta el décimo mes.

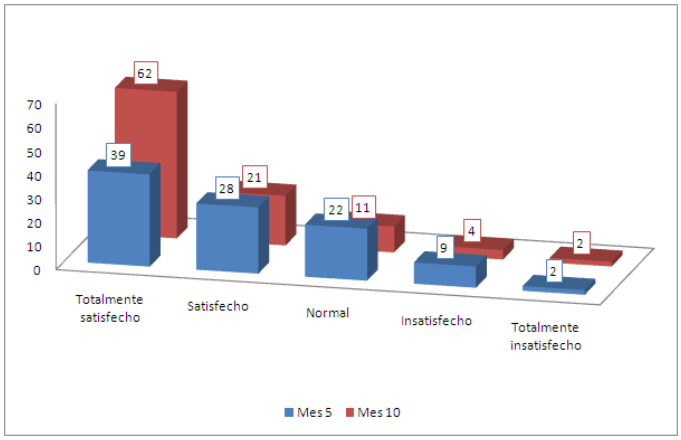
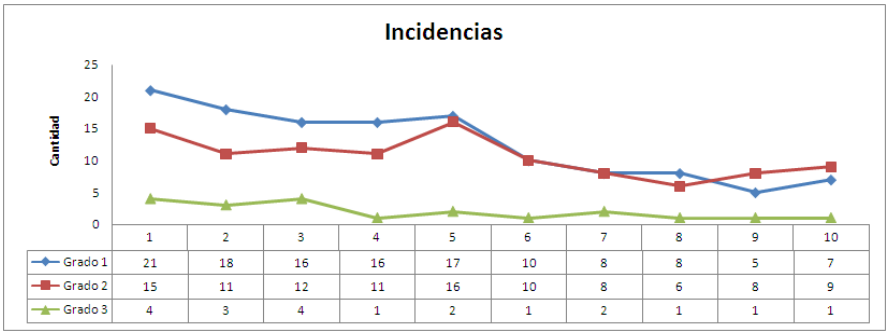


Figura 6-6. Resultado de la encuesta de satisfacción realizada a los desarrolladores

Continuando con el análisis de los resultados se recopilaron datos de las incidencias encontradas y resueltas durante los diez meses de utilización de KEMIS. En este caso se ha considerado como incidencia a cualquier error detectado en las pruebas de aceptación, ocasionado por problemas en el código fuente. Se descartan los errores que provienen de una mala interpretación de los requisitos, y que, por otra parte, estuviesen bien codificados. Los errores se califican por grado, dependiendo el tiempo en que se puede resolver y otros aspectos técnicos que serán detallados a continuación:

- Grado 1: identificados con precisión y que pueden ser corregidos y probados en el día.
- Grado 2: errores que afectan más de una capa (lógica de negocio, comunicación, base de datos, etc.) y requiere más de 1 día de trabajo.
- Grado 3: errores de grado 2 que impiden el desarrollo y/o mantenimiento de otras partes del código fuente.

A continuación, y analizando el gestor de tareas de la empresa (utilizada desde antes de la implementación de KEMIS) se graficó la evolución del número de incidencias.



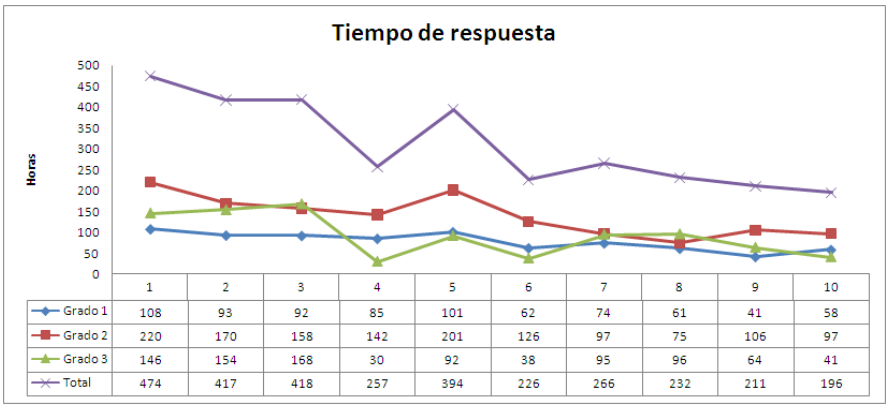
	1	2	3	4	5	6	7	8	9	10
Grado 1	21	18	16	16	17	10	8	8	5	7
Grado 2	15	11	12	11	16	10	8	6	8	9
Grado 3	4	3	4	1	2	1	2	1	1	1

Figura 6-7. Evolución de las incidencias

Como se puede ver en la Figura 6-8, la tendencia ha sido de disminución de las incidencias. Esto pudo deberse a la utilización de las herramientas de análisis estático del código fuente y a la mayor cobertura de las pruebas unitarias. Es importante recalcar aquí que no se aumentaron las incidencias en los tres primeros meses, aunque se realizaron gran cantidad de refactorizaciones para disminuir el acoplamiento entre clases. Las incidencias del tipo 1 y 2 tendieron a equipararse. Las conclusiones obtenidas con el departamento de calidad de la empresa indican

que la disminución de incidencias del tipo 1 ha posibilitado la profundización en las inspecciones, y, por lo tanto, el descubrimiento de otro tipo de problemas. Esto se ha considerado ampliamente beneficioso.

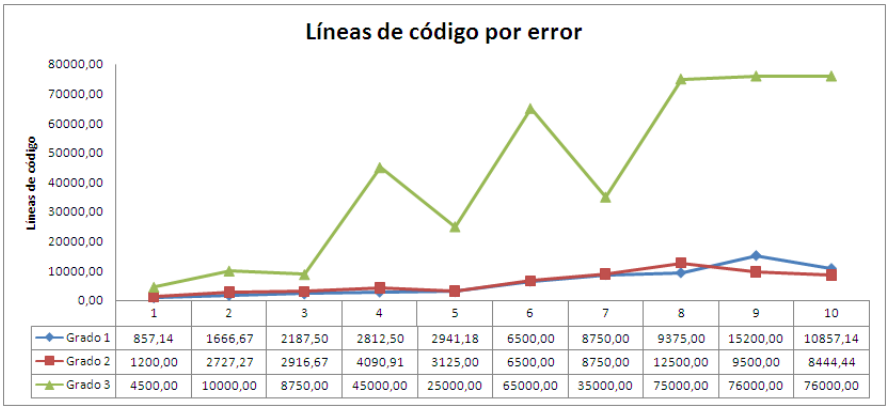
En la Figura 6-8 se indican los tiempos de respuesta totales para cada tipo de incidencia. La tendencia de los tiempos de respuesta ha sido el de disminuir. En el mes 5 se ha notado una suba considerable (acompañando la cantidad de incidencias, vistas anteriormente, con la cantidad de horas invertidas en solucionarlo). En este sentido, y analizándolo con el departamento de desarrollo de la empresa se ha concluido que las causas estaban relacionadas con la inclusión de nuevos desarrollos realizados por terceros. Estos nuevos recursos no tuvieron la misma formación inicial, sino que se realizó una formación a distancia supervisada por los desarrolladores de la empresa cliente.



	1	2	3	4	5	6	7	8	9	10
Grado 1	108	93	92	85	101	62	74	61	41	58
Grado 2	220	170	158	142	201	126	97	75	106	97
Grado 3	146	154	168	30	92	38	95	96	64	41
Total	474	417	418	257	394	226	266	232	211	196

Figura 6-8. Evolución de los tiempos de respuesta de las incidencias

En la Figura 6-9 se aprecia la evolución del promedio de líneas de código por error encontrado. Es decir, cada cuantas líneas de código fuente se encontraba un error de grado 1, 2 o 3. Aunque la cantidad de información no es suficiente para obtener una conclusión categórica, se puede observar una tendencia similar en todas las tipologías de errores. La cantidad de errores disminuye mientras que la cantidad de líneas de código aumenta. En los errores de grado 3 esto fue, por lo general cíclico. Esto tiene que ver con que nunca existieron más de 3 errores de grado 3 en un mes.



	1	2	3	4	5	6	7	8	9	10
Grado 1	857,14	1666,67	2187,50	2812,50	2941,18	6500,00	8750,00	9375,00	15200,00	10857,14
Grado 2	1200,00	2727,27	2916,67	4090,91	3125,00	6500,00	8750,00	12500,00	9500,00	8444,44
Grado 3	4500,00	10000,00	8750,00	45000,00	25000,00	65000,00	35000,00	75000,00	76000,00	76000,00

Figura 6-9. Promedio de líneas de código por error

6.3.7 Conclusiones

La implantación de KEMIS en esta empresa no sólo ha demostrado que proporciona visibilidad con sus indicadores, sino que gracias a que soporta el versionado de modelos de calidad, ha facilitado la personalización de la herramienta a las necesidades de la empresa e incluso a las necesidades de distintos proyectos dentro de la propia empresa. Las incidencias han disminuido y el trabajo de mantenimiento es menor.

Las refactorizaciones realizadas durante los diez meses, así como la capacidad de invertir nuevos recursos (por ejemplo en cobertura de pruebas unitarias) siguió la priorización sugerida por el modelo de medición del valor. En ese sentido, se compararon las importancias de las diferentes funcionalidades, se fueron analizando los cambios en las clases y la misma mantenibilidad. Como resultado de ello se pudieron focalizar los esfuerzos y los recursos en los módulos más importantes y cuyo retorno de inversión era mayor.

Durante los diez meses de utilización del entorno KEMIS, los usuarios aportaron ideas y sugerencias relacionadas con el modelo de calidad, las herramientas y reglas utilizadas, así como los umbrales de medición. Esto favoreció la construcción y mejora tanto del entorno como del modelo de medición definitivo detallado en esta Tesis Doctoral.

6.4 Verificación del Entorno KEMIS

Para la verificación interna del entorno KEMIS se han realizado pruebas manuales de aceptación. Estas pruebas se realizaron semanalmente a partir de la elaboración de la interfaz de KEMIS. Así mismo, se planificaron siete revisiones mayores a lo largo de todo el proyecto, en los que se estableció el objetivo de 100% de éxito en la ejecución de las pruebas. Para ello, se ha detallado un conjunto de pruebas escritas en la herramienta Testlink, la cual provee de una interfaz Web capaz de manejar perfiles de pruebas, diferentes ejecuciones de las mismas e informes detallados.

En la planificación y ejecución de las pruebas han participado desarrolladores de la empresa Kybele Consulting relacionados con el entorno KEMIS, así como personal que no ha tenido contacto directo con la herramienta. De esta manera se ha intentado lograr una mayor cobertura en la ejecución de las pruebas, disminuyendo el sesgo que pudiesen tener los desarrolladores del entorno. En las siguientes tablas se enumeran algunas de las plantillas utilizadas para la verificación de la interfaz. Para la realización de las pruebas se ha dividido la interfaz en diferentes aspectos, probando las características principales, como, por ejemplo: las visualizaciones, la búsqueda de proyectos, el manejo de históricos, la ventana principal o la ejecución remota de proyectos.

Tarea	Resultado
Explicación: comprobar el 1er párrafo de la explicación	OK
Explicación: comprobar el 2do párrafo de la explicación	OK
Comprobar funcionamiento combos y checkbox	OK
Clic sobre treemap	OK

Tabla 6-1. Ejemplo de tarea de verificación de KEMIS. TreeMap

Tarea	Resultado
Distribución de características	OK
Característica a NULL	OK
Característica a 0	OK
Característica mayor a 0	OK
Característica mayor a 100	OK

Tabla 6-2. Ejemplo de tareas de verificación de KEMIS. Tabla de características

Analizabilidad	Capacidad de ser probado	Cambiabilidad	Obtenido	Esperado
NULL	NULL	NULL	sin líneas	sin líneas
NULL	NULL	0	sin líneas	sin líneas
NULL	0	0	sin líneas	sin líneas
0	0	0	sin líneas	sin líneas
20	0	0	sin líneas	sin líneas
20	40	0	una línea	una línea
20	40	NULL	una línea	una línea
20	40	30	tres líneas	tres líneas
100	100	100	tres líneas	tres líneas
120	100	80	tres líneas, >100 --> 100	tres líneas, >100 --> 100
120	120	130	tres líneas, >100 --> 100	tres líneas, >100 --> 100

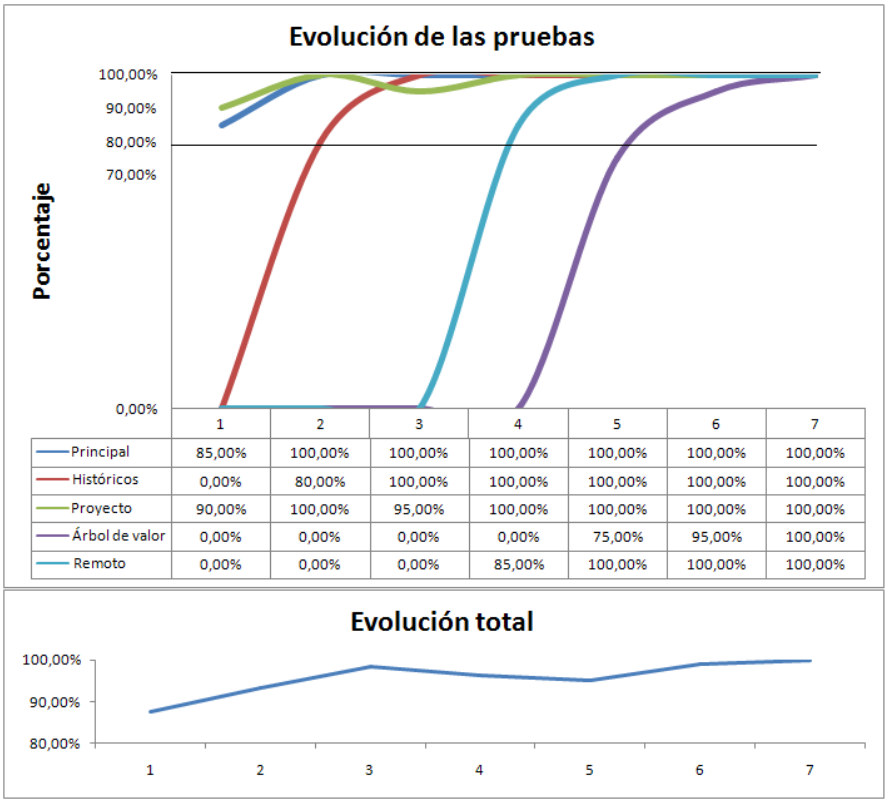
Tabla 6-3. Ejemplo de tareas de verificación de KEMIS. Kiviat

Nombre de Proyecto	Obtenido	Esperado
Vacío	Link Funcionando. Ítem Vacío	Link Funcionando. Ítem Vacío
NULL	Link Funcionando. Ítem Vacío	Link Funcionando. Ítem Vacío
Normal	Link Funcionando.	Link Funcionando.
Mayor a 50 caracteres	Link Funcionando. Nombre completo	Link funcionando. Nombre completo.

Tabla 6-4. Ejemplo de tareas de verificación de KEMIS. Búsqueda de proyectos

En la Figura 6-10 se muestra la evolución de la ejecución del conjunto de pruebas de aceptación interna realizadas en el entorno KEMIS. Se puede observar como las diferentes partes del entorno fueron evaluándose conforme eran desarrolladas. De esta manera, uno de los últimos aspectos evaluados fue el modelo de medición del valor. De hecho, revisando los errores al realizar las pruebas de aceptación, fue uno de los aspectos que más problemas y correcciones sufrió (solo pudo pasar satisfactoriamente con un 75% de las pruebas al inicio). La

interfaz principal fue una de las primeras en converger hacia el 100% de las pruebas, incluso tras agregarse nuevas características. La interfaz de proyectos individuales llegó al 100% de éxito durante la segunda revisión del entorno pero disminuyó hasta el 95% en la tercera revisión. Esto se debió al sesgo en la interpretación de algunas pruebas y a la cobertura de las mismas. Para favorecer una mayor amplitud en las pruebas se incorporaron dos técnicas, las clases de equivalencia y el análisis del valor límite [184]. De esta manera pudo mejorarse el grado de cobertura. Hacia el final de la sexta y séptima revisión se pudo lograr el 100% de éxito en la aplicación de pruebas de aceptación con un nivel de cobertura verificado internamente en las revisiones anteriores.



	1	2	3	4	5	6	7
Principal	85,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
Históricos	0,00%	80,00%	100,00%	100,00%	100,00%	100,00%	100,00%
Proyecto	90,00%	100,00%	95,00%	100,00%	100,00%	100,00%	100,00%
Árbol de valor	0,00%	0,00%	0,00%	0,00%	75,00%	95,00%	100,00%
Remoto	0,00%	0,00%	0,00%	85,00%	100,00%	100,00%	100,00%

Figura 6-10. Evolución del porcentaje de pruebas satisfactorias

6.5 Auditorías Oficiales del Centro para el Desarrollo Tecnológico Industrial

Debido a que parte de la financiación para la realización del entorno KEMIS de I+D+i (IDI-20090175) ha sido llevada a cabo por el Centro para el Desarrollo Tecnológico Industrial (CDTI), dicho organismo ha realizado auditorías técnicas periódicas. En la Tabla 6-5 se puede observar un resumen de las fechas de auditoría, las evaluaciones realizadas y los resultados obtenidos. El proyecto se ha dado por finalizado en febrero del año 2012 con resultado ha sido completamente satisfactorio, concediéndose la máxima nota técnica.

Fecha de realización	Evaluación técnica	Descripción
Octubre 2009	Correcta	Se realizó un informe técnico detallado de las tareas realizadas, seguimiento de lo planificado y resultados obtenidos. La documentación fue revisada por un auditor jefe del CDTI con un mes de antelación. Por último se realizó una presentación con los resultados técnicos alcanzados, auditándose el entorno y las implantaciones realizadas hasta el momento. El resultado fue satisfactorio, obteniéndose una evaluación sobresaliente.
Marzo 2011	Correcta	
Febrero 2012	Correcta	

Tabla 6-5. Auditorías internas CDTI

Conclusión

A modo de conclusión de esta Tesis Doctoral, en este capítulo se resumen las principales contribuciones realizadas y se analiza el cumplimiento de los objetivos establecidos en la sección 1.2. Además, se realiza un análisis de los resultados obtenidos enumerando las publicaciones que sirven para contrastarlos, tanto en foros nacionales como en foros internacionales.

7.1 Análisis de los Objetivos

Al inicio de esta Tesis Doctoral, se presentaron una serie de objetivos parciales (sección 1.2) cuyo cumplimiento implica el cumplimiento del objetivo principal de esta tesis: *la definición de un entorno que facilite la evaluación de la mantenibilidad del producto software, calculando el valor y el retorno de inversión de las posibles refactorizaciones necesarias para mejorar dicha mantenibilidad.*

A continuación se analizará el cumplimiento de cada uno de los objetivos parciales:

El objetivo 1 se definió como: *"Análisis y evaluación de trabajos relacionados con los modelos de medición de la calidad del producto software, las refactorizaciones y su priorización basada en aspectos del valor; incluyendo propuestas metodológicas, herramientas, etc."*

Como parte del trabajo necesario para la consecución de este objetivo, se ha desarrollado los modelos actuales de evaluación de calidad del producto y las herramientas para la obtención de las métricas asociadas a dichos modelos. Este estudio ha resultado clave en la especificación de los objetivos del proyecto KEMIS y ha sido fundamental para proporcionar una solución a las necesidades planteadas. En particular, en el capítulo II se ha presentado el estado del arte de la Ingeniería de Software Basada en Valor, las refactorizaciones y los modelos de referencia para la medición de la calidad del producto software. Así, se realiza una breve reseña de las normas ISO/IEC 9126 e ISO/IEC 25010 de calidad del producto software, así como su descripción de la mantenibilidad. Se describen los fundamentos de la Ingeniería de Software Basada en Valor y se detallan los resultados de la revisión sistemática de la literatura sobre refactorización basada en valor, enumerando los indicadores de valor utilizados en otras disciplinas. Por

último, como parte de los estudios previos, se analizan un conjunto de herramientas centradas en la medición de la calidad del producto software.

Aparte de los estudios del estado del arte se han identificado las características genéricas que debería reunir un modelo de medición de la calidad del producto software basado en valor, con la intención de definir un modelo genérico de análisis como punto de partida hacia la obtención de un modelo de implementación para diferentes tecnologías.

El objetivo 2 se definió como: "*Desarrollo de un modelo de calidad, donde se permitan configurar los diferentes elementos del modelo para personalizar la medición de la mantenibilidad entre proyectos*". Para la consecución de este objetivo, se han tenido en cuenta las siguientes metas:

O2.1. Una infraestructura básica que integra la ejecución de herramientas que generan informes con métricas básicas de calidad.

Para llevar a cabo la implantación de la infraestructura de la herramienta KEMIS se han realizado las siguientes tareas:

- Se han estudiado la principales herramientas de análisis del código fuente, así como las medidas que extraen (sección 3.1.1 y sección 3.1.2).
- Se han alineado las medidas básicas con las subcaracterísticas de mantenibilidad de la norma ISO/IEC 9126 (sección 3.1.3)
- Se ha realizado una tabla resumiendo las alineaciones (sección 3.1.4) y se han obtenido conclusiones del estudio que facilitaron la construcción del modelo de medición (sección 3.1.5)

Mediante la información obtenida se desarrolló una infraestructura avanzada de medición, persistiendo los resultados en un modelo de medición de la mantenibilidad con funciones de derivación y umbrales.

O2.2. Una infraestructura avanzada que, a partir de la lectura de métricas básicas, genere y persista el valor de subcaracterísticas y características de calidad.

Se definió el proceso de derivación de las subcaracterísticas y características de calidad:

- Se realizó la configuración del modelo de medición de la mantenibilidad (sección 3.3).
- Se diseñaron las características de las medidas de calidad que constituían el modelo de medición (sección 3.3.1).
- Se definieron las características de las medidas de calidad para la subcaracterística de analizabilidad que implementará la herramienta KEMIS (sección 3.3.2).
- Se definieron las características de las medidas de calidad para la subcaracterística de cambiabilidad que implementará la herramienta KEMIS (sección 3.3.3).
- Se definieron las características de las medidas de calidad para la subcaracterística de estabilidad que implementará la herramienta KEMIS (sección 3.3.4).
- Se definieron las características de las medidas de calidad para la subcaracterística de "capacidad para ser probado" que implementará la herramienta KEMIS (sección 3.3.5).
- Se definieron las funciones de derivación con las que se obtienen los valores de las subcaracterísticas (sección 3.3.6).
- Se definió la función de derivación con la que se obtiene el valor de la característica de mantenibilidad (sección 3.3.6.1).

Se definió e implementó en la herramienta KEMIS el procedimiento de lectura y procesamiento de las métricas básicas y derivadas (secciones 5.2.1, 5.2.2 y 5.2.3).

Finalmente, se diseñó la arquitectura de la herramienta KEMIS (sección 5.1), se realizó el desarrollo de la lógica de negocio de KEMIS (sección 5.4), y, en particular, de la base de datos para persistir el modelo de medición (sección 5.4.2).

El objetivo 3 se refería a la "evaluación de la mantenibilidad del producto software:" a partir de los siguientes objetivos parciales:

O3.1. Evaluación de la mantenibilidad del producto software a distintos niveles de abstracción, proporcionando un cuadro de mando de indicadores, métricas derivadas y métricas básicas.

Una vez desarrollada la infraestructura básica y avanzada de la herramienta KEMIS se ha realizado el entorno visual (ver apéndice B), en particular:

- Se han elegido un conjunto de librerías para las visualizaciones (sección B.1).
- Se han analizado las visualizaciones que más útiles de acuerdo con la estructura de los datos a mostrar (sección B.2).
- Se ha diseñado la estructura de la visualización (sección B.3).
- Se ha desarrollado el cuadro de mando de la herramienta KEMIS teniendo en cuenta los visualizadores y la arquitectura definida anteriormiente (sección B.4)

El objetivo 4 se refería al "*desarrollo de un modelo de medición del valor*" y se dividía en los siguientes objetivos parciales:

O4.1. Determinar el retorno de inversión de realizar una refactorización en el software.

Se han buscado de manera formal y sistemática los diferentes indicadores de valor relacionados con las mejoras del software. Para ello se han utilizado revisiones sistemáticas y búsquedas en congresos representativos de la Ingeniería del Software basada en Valor. Como resultado de esto, se han obtenido tablas con los indicadores potenciales, utilizando técnicas de analogía con otras disciplinas (sección 2.7) y se ha diseñado el modelo de medición del valor:

- Se ha definido un modelo de medición del valor (sección 4.2)
- Se han definido una serie de indicadores de valor (sección 4.3.1)
- Se ha diseñado una función de derivación para obtener el valor en base a los indicadores antes mencionados (sección 4.4).

Como paso posterior se han seleccionado las herramientas necesarias para obtener los indicadores de valor (sección 5.2.5). En particular se ha desarrollado una herramienta en Excel para favorecer el tratamiento de los datos de importancia relativa y urgencia de mejora (ver apartado 5.2.5.1).

Por último se ha realizado la integración del módulo de medición del valor en la herramienta KEMIS (sección 5.2.6).

O4.2. Priorizar las diferentes refactorizaciones.

Para cumplir con este objetivo se han identificado las principales refactorizaciones y se ha realizado un estudio preliminar de la influencia que tiene cada refactorización en la mantenibilidad (sección A.2.1).

En la sección A.2.9 se analizaron los resultados, identificando refactorizaciones que disminuían la mantenibilidad.

El objetivo 5 indicaba: "Validación del entorno. Dicha validación se realizará desde dos puntos de vista". Como prueba de concepto de la herramienta, se implementarán:

O5.1. Una infraestructura Web para centralizar las evaluaciones y las mediciones del valor.

Se ha decidido cuáles serán las visualizaciones que se van a introducir en la aplicación (ver Apéndice B). En un primer momento, se ha realizado un estudio previo de lo que se pretendía mostrar, promoviendo dos interfaces distintas de visualización. Por un lado, se ha realizado una interfaz principal que muestra un resumen de todos los proyectos sobre los que se ejecute la herramienta y por otro lado una interfaz individual para mostrar el resumen de cada uno de sus proyectos (sección B.4).

Una vez se tiene realizada la interfaz Web, comenzaron las pruebas de verificación y validación de la misma. Se han realizado pruebas de aceptación internas teniendo en cuenta clases de equivalencia y análisis del valor límite (sección 6.4). La interfaz se ha colocado en pre-producción para que fuese probada por personas de diferentes entornos; los fallos obtenidos han sido notificados al personal encargado de la Web, realizándose el mantenimiento correctivo.

O5.2. Una prueba piloto de la herramienta y evaluación de los resultados.

Se han evaluado los resultados de la herramienta KEMIS en entornos reales. De esta manera se contrastan los modelos de medición teóricos con la

experiencia práctica, permitiendo una mejora continua de las funciones de medición, validación de los umbrales y la obtención de información de retorno del mercado para facilitar la innovación. En el apartado 6.3 de esta Tesis Doctoral se presenta un resumen de las tareas desempeñadas en entornos reales y las conclusiones obtenidas al utilizar la herramienta.

Finalmente, parte de la herramienta KEMIS ha sido auditada por el Centro para el Desarrollo Tecnológico Industrial (CDTI), con resultado satisfactorio (sección 6.5)

7.2 Resultados Científicos

Algunos de los resultados obtenidos durante el desarrollo de la Tesis Doctoral se han publicado en diferentes congresos, tanto nacionales como internacionales. A continuación se presentan las diferentes publicaciones, agrupadas por tipo de publicación. Así mismo, se presentan dos publicaciones que se encuentran actualmente en proceso de revisión.

- **Artículos en Conferencias Internacionales**

Autores: **Irrazábal E.**, Garzás J., Marcos E.

Título: Alignment of open Source Tools With The New ISO 25010 Standard: focus on maintainability.

Congreso: International Conference on Software and Data Technologies (ICSOFT). Sevilla – España.

Año: 2011

Autores: **Irrazábal E.**, Vara JM, Garzás J, Marcos E.

Título: Alignment of Open Source Tools with ISO norms for software product quality.

Congreso: Software Measurement European Forum (SMEF). Roma - Italia.

Año: 2010.

Autores: Quevedo A., **Irrazábal E.**, Enríquez J., Vara J. M., Garzás J.

Título: Entorno de calidad KEMIS.

Congreso: International Conference on the Quality of Information and Communications Technology (QUATIC), Industrial Track. Oporto - Portugal.

Año: 2010.

- **Artículos en Conferencias Iberoamericanas**

Autores: **Irrazábal E.** , Garzás J., Vara J. M., Marcos E.

Título: Un enfoque basado en valor para la refactorización software.

Congreso: XV Jornadas de Ingeniería del Software y Bases de Datos (JISBD). Valencia - España.

Año: 2010.

- **Artículos en Conferencias Nacionales**

Autores: **Irrazábal E.**, Garzás J.

Título: Análisis de métricas básicas y herramientas de código libre para medir la mantenibilidad.

Congreso: XII Jornadas de Innovación y Calidad del Software (JICS). Madrid – España.

Año: 2010.

Autores: **Irrazábal E**., Santa Escolástica R., Garzás J.

Título: El entorno KEMIS. Aproximación al modelo de medición del valor.

Congreso: I Jornada de Jóvenes Investigadores. Cáceres – España

Año: 2010.

- **Artículos en Revisión**

Autores: **Irrazábal E.**, Vara J.M., Garzás J., Marcos E.

Título: An Approach of a Value-Based Measurement Model to Prioritize Refactorings

Revista: Special issue in Science of Computer Programming - BENEVOL 2010

Autores: **Irrazábal E.**, Vara J.M., Garzás J., Marcos E.

Título: Evaluating open source tools to measure software maintainability: a perspective based on ISO 25010

Conferencia: 16th International Conference on Evaluation & Assessment in Software Engineering (CORE A)

7.3 Conclusiones Generales y Trabajos Futuros

En base a los estudios previos se ha observado que la calidad del producto no es tenida en cuenta de forma proactiva y en algunos casos ni siquiera es tenida en cuenta de forma objetiva. En estos casos, las organizaciones reaccionan ante los problemas del producto en plena producción ocasionando pérdidas directas e indirectas (por ejemplo de imagen) aún mayores.

Actualmente existen proyectadas una serie de tareas para el futuro de la herramienta KEMIS entre las que se destacan el hecho de integrar nuevas herramientas de medición y documentación, dado el auge y frecuencia con la que hoy en día se están desarrollando dichas herramientas. También almacenar mayor cantidad de datos procedentes de los resultados de la medición que permitan definir y generar nuevos indicadores e informes.

7.3.1 Ampliar el modelo de medición de la mantenibilidad

Actualmente se están realizando pruebas piloto con nuevas herramientas de análisis estático del código. Así mismo se han realizado pruebas implementando el modelo de medición con herramientas que analizan lenguajes .Net.

7.3.2 Diseñar modelos de medición para otras características de calidad del producto software

De acuerdo al diseño de la herramienta KEMIS, esta tiene la capacidad de soportar otros modelos de medición. Por ello se está trabajando en definir otros modelos de medición asociados con la usabilidad o la seguridad.

7.3.3 Realizar un análisis estadístico de la aplicación de refactorizaciones y la evolución de la mantenibilidad

Con el uso de la herramienta KEMIS y la recolección de información será posible realizar estudios estadísticos concluyentes en relación con la evolución de la mantenibilidad. Para ello es necesario diseñar y ejecutar experimentos controlados en entornos académicos, así como aumentar la cantidad de empresas que utilizan la herramienta.

Apéndice A: Impacto de las refactorizaciones

En este apéndice se analizará el catálogo de refactorizaciones de Martin Fowler descrito en [22], identificando el impacto que tendrán sobre las subcaracterísticas de mantenibilidad del producto software. Para ello, se utiliza el modelo de medición de la mantenibilidad descrito en el capítulo 3.

A.1 Mejora de la Mantenibilidad

La realización de una modificación en el producto software tendrá más valor cuanto más consiga aumentar la mantenibilidad del producto. El aumento de mantenibilidad que se obtendrá estará determinado por dos factores: la mantenibilidad que posee el componente sobre el que se va a realizar el cambio antes de aplicarlo, y la capacidad que tendrá el cambio de mejorar la mantenibilidad. El primero de estos dos valores vendrá determinado por las características del propio proyecto. Es por esto que se decide centrar el estudio en el segundo de los factores, la capacidad que tiene la modificación de mejorar la mantenibilidad.

En el caso de la refactorización, puede determinarse el efecto que tiene sobre el código fuente, analizando el impacto que tendrá en ciertas métricas. Como ya se ha indicado en la sección 3.3 es posible, a partir de mediciones en los niveles más bajos del producto (código fuente), realizar cálculos para alcanzar en primer lugar las características de la mantenibilidad (analizabilidad, cambiabilidad, capacidad de ser probado y estabilidad) y la propia mantenibilidad.

El modelo de medición de la mantenibilidad proporcionado por KEMIS permite, partiendo de métricas extraídas directamente del código fuente, componer los valores para calcular la analizabilidad, cambiabilidad y capacidad de ser probado de un producto, y, a partir de estos, calcular la mantenibilidad. Utilizando el modelo de medición de la mantenibilidad proporcionado por KEMIS puede identificarse el impacto que tendrá una refactorización en la mantenibilidad. Por lo tanto, el análisis que se va a realizar sí tendrá en cuenta esta característica. Resumiendo, el análisis va a tratar de averiguar el impacto que tendrá una refactorización en la mantenibilidad del producto.

A.2 Identificación de las métricas

Las métricas definidas por KEMIS son las siguientes:

- Sentencias de código que no son comentarios (NCSS): esta métrica indica la cantidad de líneas de código que posee el producto o una parte del mismo. Esta métrica se asocia con la analizabilidad, ya que cuanto mayor sea un código, más complicado será su análisis.
- Complejidad ciclomática (CC): esta métrica define el número de caminos independientes dentro de un fragmento de código. Está asociada a dos características de la mantenibilidad: la analizabilidad, ya que a mayor complejidad ciclomática más difícil será analizar el software; y a la capacidad de ser probado, ya que cada camino independiente deberá ser probado.
- Violaciones de PMD (PMDv): PMD es una herramienta de software libre para Java que permite analizar el código fuente identificando problemas potenciales, como pueden ser fragmentos de código muerto, posibles bugs, código mal utilizado, etc. Las normas a ser tenidas en cuenta se definen en conjuntos de reglas (rulesets). Cada error reportado por la herramienta es considerado una violación de las reglas de PMD. Por lo tanto, esta métrica está asociada con la analizabilidad, ya que a más violaciones más complejo será el análisis del código.
- Código duplicado (CD): como su propio nombre indica, esta métrica define la cantidad de líneas que aparecen duplicadas dentro del código fuente. Esta métrica se asocia con la analizabilidad, ya que cuanto más duplicado esté un código más complejo será su análisis.
- Ciclos (CYC): esta métrica define la cantidad de ciclos que se producen entre diferentes llamadas dentro del código fuente. Un ciclo se produce cuando un componente A utiliza alguna propiedad de otro B, y este a su vez utiliza propiedades de A, bien lo haga directamente o bien a través de un componente B. Los ciclos provocan problemas al realizar cambios en el software, por lo que pueden relacionarse con la cambiabilidad del producto.
- Paquetes (PACK): esta métrica calcula la cantidad de paquetes que tiene el producto. Esta métrica se combina con los ciclos para calcular la densidad de ciclos del código. Por lo tanto, el número de paquetes afecta a la cambiabilidad.
- Pruebas unitarias (PU): esta métrica indica la cantidad de pruebas unitarias que existen en el código. Esta métrica se relaciona con la capacidad de ser probado, ya que la cantidad de pruebas unitarias indicará en qué medida puede probarse el producto.

- Errores en las pruebas unitarias (PUe): esta métrica contabiliza la cantidad de errores que se han producido en las pruebas unitarias. Está relacionada con la capacidad de ser probado, al igual que las pruebas unitarias.
- Ratio de cobertura de pruebas unitarias (PUcov): esta métrica indica el porcentaje de líneas de código que sean cubiertas por las pruebas unitarias. Por lo tanto, esta métrica se relaciona también con la capacidad de ser probado.

Una vez obtenidas estas métricas, pueden ser combinadas para alcanzar métricas más abstractas, hasta conseguir un valor para las características de la mantenibilidad y para la propia mantenibilidad. El valor de una métrica de mayor abstracción se realiza a través de métricas más concretas, ponderándolas dependiendo del impacto que tengan en el cálculo de la característica de mayor abstracción. En la Figura A-1 puede verse la composición que se produce partiendo de las métricas base hasta la característica de la mantenibilidad.

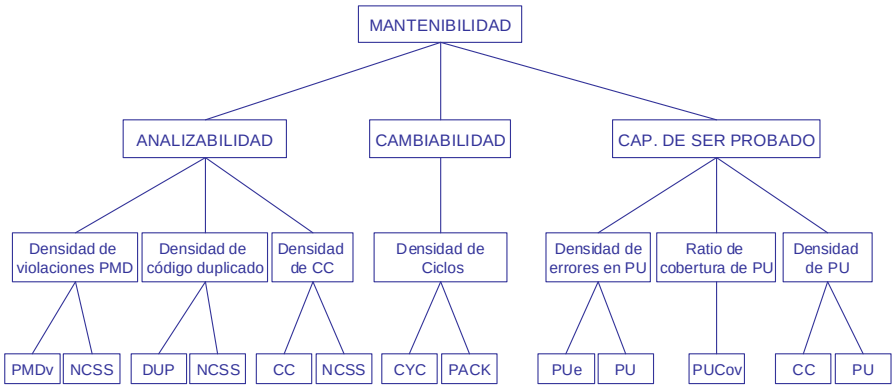


Figura A-1. Cálculo de mantenibilidad a partir de métricas del código fuente

Como se observa en la imagen, a partir de las métricas extraídas del código fuente pueden inferirse valores para la mantenibilidad del producto. Por lo tanto, estudiando el impacto que tendrán las refactorizaciones sobre estas métricas, puede analizarse el impacto que tendrán sobre la mantenibilidad total del producto y, por lo tanto, su influencia en el valor del producto.

A.2.1 Análisis de la influencia de las refactorizaciones en la mantenibilidad del producto

Como se ha indicado en el apartado anterior, a partir de métricas extraídas directamente del código fuente puede calcularse la mantenibilidad del producto

software. Gracias a esta posibilidad, puede analizarse el impacto que tendrán las refactorizaciones sobre estas métricas para identificar así el efecto que una refactorización tendrá sobre la mantenibilidad.

Para realizar este análisis, se han de seleccionar las refactorizaciones sobre las que se va a trabajar. Las refactorizaciones utilizadas son las que aparecen en el catálogo de refactorizaciones de [22]. Una vez seleccionadas, se realiza el análisis, identificando la variación que se produce en las métricas del producto software tras la aplicación de la refactorización.

El análisis se presenta siguiendo el esquema que propone Fowler en [22], donde se agrupan los diferentes tipos de refactorizaciones en función de una serie de características comunes. Cada grupo de refactorizaciones compone un capítulo del libro, por lo tanto, para nombrar al grupo se utilizará la traducción al castellano del nombre del capítulo. Al final de cada apartado se presenta una tabla en la que se muestra el aumento o disminución de la característica que produce la refactorización, siguiendo la siguiente puntuación:

Símbolo	Descripción
++	Aumento considerable del valor de la característica
+	Ligero aumento del valor de la característica
o	Ningún efecto en la característica
-	Ligera disminución del valor de la característica
--	Disminución considerable del valor de la característica

Tabla A-1. Valoración del efecto en las características

Para la valoración de la característica de la mantenibilidad, al estar compuesta por las otras tres características del modelo, su valoración es diferente. La fórmula utilizada para calcularla es la siguiente:

- Mantenibilidad = $\sum$ (nº de "+") - $\sum$ (nº de "–") de manera que si:
- Mantenibilidad ≥ 3 => ++
- Mantenibilidad = 2 ó Mantenibilidad = 1 => +
- Mantenibilidad = 0 => o
- Mantenibilidad = -1 ó Mantenibilidad = -2 => -
- Mantenibilidad ≤ -3 => --

A.2.2 Grupo de refactorizaciones 1 "componiendo métodos"

- Extract Method: esta refactorización es de las más utilizadas en la práctica [22]. Su aplicación consiste en, a partir de un método con gran cantidad de

líneas de código, extraer una parte creando un nuevo método. De esta manera consigue reducirse el número de líneas del código, haciéndolo más sencillo. Además, esta refactorización puede utilizarse para eliminar código repetido, eliminando este código repetido y sustituyéndolo por una llamada al nuevo método. Por lo tanto, puede observarse cómo la refactorización Extract Method mejora la Analizabilidad, disminuyendo las líneas de código y eliminando el código repetido. Además, al haber una disminución considerable de las líneas de código, aumenta la cobertura de las pruebas unitarias. Por lo tanto, esta refactorización también afecta a la capacidad de ser probado, mejorándola.

- Inline Method: esta refactorización se aplica cuando el cuerpo de un método es tan simple que puede sustituir a la llamada de manera que el código no se vuelve más confuso. Sin embargo, si el método es llamado más de una vez en diferentes partes del código, un cambio en la funcionalidad puede hacer que haya que realizar la modificación en esas partes, mientras que si se mantuviese el método la modificación sólo se realiza una vez. Por lo tanto, Inline Method afecta negativamente a la Analizabilidad, aumentando el código repetido.
- Inline Temp: esta refactorización se aplica cuando se tiene una variable temporal que es asignada una vez a una expresión simple. En este caso se indica que deben sustituirse todas las referencias a la variable temporal por la expresión. Esta refactorización apenas tiene efecto en la Mantenibilidad.
- Replace Temp with Query: esta refactorización consiste en sustituir las variables temporales utilizadas para almacenar el resultado de una expresión por un método en cuyo cuerpo se calcule la expresión. En este caso, las Analizabilidad se ve ligeramente afectada, ya que la sustitución puede evitar asignaciones posteriores, reduciendo en pequeña medida las líneas de código.
- Introduce Explaining Variable: esta refactorización consiste en utilizar una variable temporal para almacenar parte de una expresión, de manera que cuente con un nombre que explique su propósito de manera clara. Esta refactorización ayuda a hacer las expresiones más manejables y entendibles. Sin embargo, aumenta la cantidad de líneas de código. Por lo tanto, siguiendo el modelo de medición propuesto, esta refactorización afecta negativamente a la Analizabilidad.
- Split Temporary Variable: esta refactorización se aplica cuando a una variable temporal, utilizada sólo para almacenar el resultado de una expresión, se le asigna más de una vez. En este caso, se aconseja utilizar dos variables

temporales distintas, en vez de reutilizar la misma. Esta refactorización no tiene ninguna influencia en la Mantenibilidad según el modelo de medición utilizado.

- Remove Assignments to Parameters: esta refactorización consiste en eliminar una asignación a un parámetro cuando lo que se está asignando es otro objeto. En este caso, la refactorización propone utilizar una variable temporal. Esta refactorización pretende ayudar a eliminar errores con el paso por valor y por referencia. Por lo tanto, esta refactorización mejora ligeramente la Capacidad de Ser Probado del código, ayudando a reducir los errores.
- Replace Method with Method Object: esta refactorización se utiliza cuando se tiene una clase de gran tamaño pero no se puede aplicar la refactorización Extract Method debido a las variables temporales existentes en el método. Esta refactorización propone hacer que el método se transforme en una clase cuyas variables locales sean los atributos de la clase. De esta manera, puede utilizarse Extract Method en la clase, realizando llamadas a estos métodos para sustituir el método anterior. Esta refactorización ayuda a disminuir las líneas de código y el código repetido. Por lo tanto, tiene un impacto positivo sobre la Analizabilidad. Esta refactorización crea una nueva clase, pero al estar fuertemente ligada a la clase que lo llama por ser una refactorización de un antiguo método de ésta, es poco probable que la clase cambie de paquete y se puedan producir ciclos. Por lo tanto, la Cambiabilidad no se ve influida.
- Substitute Algorithm: esta refactorización consiste en sustituir un algoritmo por otro más claro. Que sea más claro implica que sea más corto y que sea más fácil de probar, reduciendo la cantidad de pruebas unitarias necesarias y aumentando el ratio de cobertura del código. Por los tanto, tanto la Analizabilidad como la Capacidad de Ser Probado son mejoradas con esta refactorización.

Este grupo de refactorizaciones afectan principalmente a la Analizabilidad. No todas las refactorizaciones mejoran la mantenibilidad, pero se observa que la mayoría de ellas tienen un efecto positivo. Los resultados pueden observarse en la Tabla A-2.

Refactorización	A	C.	T.	M.
Extract Method	++	o	+	++
Inline Method	-	o	o	-
Inline Temp	o	o	o	o
Replace Temp with Query	+	o	o	+
Introduce Explaining Variable	-	o	o	-

Refactorización	A	C.	T.	M.
Split Temporary Variable	o	o	o	o
Remove Assignments to Parameters	o	o	+	+
Replace Method with Method Object	+	o	o	+
Substitute Algorithm	++	o	+	++

Tabla A-2. Resultados de refactorizaciones "Componiendo métodos"

A.2.3 Grupo de refactorizaciones 2 "moviendo características entre objetos"

- Move Method: esta refactorización propone que, si un método llama o es llamado más veces por una clase externa que por su misma clase, este método debe ser movido a esa clase que lo utiliza. Esta refactorización afectará a la Cambiabilidad si el método es perteneciente a otro paquete, ya que estas llamadas pueden provocar ciclos. Por lo tanto, aplicar la refactorización mejora la Cambiabilidad. El resto de características no sufren variación.
- Move Field: esta refactorización es similar a la refactorización Move Method, pero teniendo en cuenta a los atributos de la clase. Por lo tanto, se comporta de igual manera que ella.
- Extract Class: esta refactorización se aplica cuando se tiene una clase que realiza el trabajo que deberían hacer dos. Se propone dividir la clase en dos, con funcionalidades diferentes. Esta refactorización consiste disminuir las líneas de código que poseerá la clase, aunque aparecerá una clase más. Por lo tanto, tiene un efecto positivo sobre la Analizabilidad.
- Inline Class: esta refactorización es la opuesta a la anterior. Es decir, se tiene una clase que hace poco trabajo. Lo que se propone es combinarla con otra. Por lo tanto, esta clase aumentará sus líneas de código, perjudicando a la Analizabilidad.
- Hide Delegate: esta refactorización promueve la encapsulación. De esta manera, la refactorización propone la utilización de métodos que delegan en llamadas a métodos de objetos de otra clase. De esta manera se reduce el acoplamiento entre clases. Esta refactorización afecta positivamente a la Cambiabilidad.
- Remove Middle Man: esta refactorización es opuesta a la anterior. Propone la eliminación de métodos que delegan en llamadas a métodos de objetos de otra

clase. Al ser opuesta, dificulta ligeramente los cambios, por lo tanto disminuye en pequeña medida la Cambiabilidad.

- Introduce Foreign Method: esta refactorización se aplica cuando se necesita un método en una clase, pero ésta no puede modificarse (caso común en Java, donde existe gran cantidad de clases ya realizadas). La refactorización propone la creación de un método en la clase que quiere hacer uso de ese método, que tenga como primer argumento a una instancia de esa clase que no puede ser modificada. Esta refactorización, al crear un nuevo método en la clase que puede ser utilizado más de una vez, ayuda a reducir el código repetido y, por lo tanto, mejora ligeramente la Analizabilidad. Sin embargo, la aplicación de este método puede reducir la Cambiabilidad del código, ya que al aparecer el método en una clase que "no le corresponde", puede ayudar a la aparición de ciclos.
- Introduce Local Extension: esta refactorización es similar a la anterior, pero en vez de introducir el método nuevo en otra clase ya existente, se introduce en una clase nueva, creada especialmente para contener este método. De esta manera, si el método es utilizado por distintas clases, el método está disponible desde esta clase y no debe ser introducido en más de una clase. Esta refactorización ayuda a evitar la aparición de código repetido, lo que hace que mejore la Analizabilidad. Al revés que en el caso de Introduce Foreign Method, en esta refactorización la Cambiabilidad no se ve afectada.

En este caso, las refactorizaciones afectan en su mayoría a la Analizabilidad y a la Cambiabilidad. Este grupo de refactorizaciones tiene efectos pequeños sobre la mantenibilidad, no mejorándola ni empeorándola en demasía. Los resultados obtenidos pueden verse en la Tabla A-3:

Refactorización	A	C	T	M
Move Method	o	+	o	+
Move Field	o	+	o	+
Extract Class	+	O	o	+
Inline Class	-	O	o	-
Hide Delegate	o	+	o	+
Remove Middle Man	o	-	o	-
Introduce Foreign Method	+	-	o	o
Introduce Local Extension	+	O	o	+

Tabla A-3. Resultados de refactorizaciones "Moviendo características entre objetos"

A.2.4 Grupo de refactorizaciones 3 "organizando datos"

- Self Encapsulate Field: esta refactorización propone que, si un atributo es accedido directamente desde su propia clase, se creen métodos "get" y "set" para realizar este acceso. Esta refactorización no afecta a ninguna de las características de la Mantenibilidad.
- Replace Data Value with Object: esta refactorización se aplica cuando se tiene un objeto de un tipo primitivo, pero este objeto necesita tener más comportamiento (nuevas características, métodos, etc.). Propone crear una clase que contenga la funcionalidad relativa a ese objeto. Esta refactorización afecta a la Analizabilidad, reduciendo la cantidad de líneas de código que se tendrían si este comportamiento se implementara dentro de la misma clase. De igual manera, disminuye el código repetido. Al haber una reducción en las líneas de código, también afecta a la Capacidad de ser Probado, ya que facilita la creación de pruebas unitarias aumenta el grado de cobertura de las pruebas.
- Change Value to Reference: esta refactorización propone la utilización de referencias a objetos en lugar del objeto en sí. De esta manera, puede utilizarse por toda la clase teniendo que utilizar solo un objeto para cada instancia. Esta refactorización no afecta especialmente a ninguna característica de la Mantenibilidad.
- Change Reference to Value: esta refactorización es la opuesta a la anterior. De igual manera que ésta, no tiene ningún efecto sobre las características de la mantenibilidad.
- Replace Array with Object: esta refactorización se aplica cuando se tiene un array cuyos elementos tienen distintos significados. La refactorización propone crear una clase que permita generar objetos cuyos atributos sean los diferentes campos que tenía el array. De esta manera, si se necesita tratar a los elementos del array de manera diferente, pueden ser tratados más fácilmente. Esta refactorización ayuda a disminuir ligeramente las líneas de código y en mayor medida la complejidad ciclomática, ya que es común que aparezcan comparaciones si se utiliza el array. Por lo tanto, la refactorización mejora tanto la Analizabilidad como la Capacidad de ser Probado.
- Duplicate Observed Data: esta refactorización consiste en la creación de "Observadores" para separar los datos en las capas cuando se tiene un sistema de capas. Para aplicarlo, es necesario duplicar los datos y crear un mecanismo de sincronización entre los "Observadores" creados. Al ser necesario duplicar

los datos se aumenta la cantidad de código repetido y las líneas de código. Por lo tanto, esta refactorización disminuye la Analizabilidad del código.

- Change Unidirectional Association to Bidirectional: esta refactorización se aplica cuando dos clases se utilizan mutuamente, pero solo tiene una conexión unidireccional entre ellas. Esta refactorización no tiene efectos en la Mantenibilidad ni en ninguna de sus características según el modelo de medición propuesto.
- Change Bidirectional Association to Unidirectional: esta refactorización es opuesta a la anterior. Se aplica cuando una clase no necesita características de otra, pero tienen una relación bidireccional. Al igual que la anterior, no tiene un efecto directo sobre la Mantenibilidad teniendo en cuenta el modelo de medición.
- Replace Magic Number with Symbolic Constant: esta refactorización se aplica cuando se tiene un número con un significado particular en el código. Se aconseja crear una constante que reemplace a ese valor. Esta refactorización, facilita realizar los cambios ya que, si el valor cambia, es necesario modificarlo en varios lugares, mientras que con la constante solo se modificaría una vez. Sin embargo, en el modelo propuesto esto no es tenido en cuenta, por lo que puede indicarse que su impacto en la Mantenibilidad y sus características es nulo.
- Encapsulate Field: similar a Self Encapsulate Field, pero sin tener en cuenta la clase desde la que se accede. Tampoco influye en la mantenibilidad según el modelo de medición propuesto.
- Encapsulate Collection: esta refactorización propone que, cuando un método devuelve una colección, hacer que devuelva un valor que no sea modificable y crear métodos para añadir y eliminar objetos de la colección. Esta refactorización mejora ligeramente la Analizabilidad, ya que al crear los métodos para añadir y eliminar, desaparece código repetido.
- Replace Record with Data Class: esta refactorización consiste en sustituir los registros de la programación tradicional por un objeto. No tiene ningún efecto en la Mantenibilidad.
- Replace Type Code with Class: esta refactorización consiste en, en una clase que tiene un código de tipo numérico que no afecta al comportamiento de la clase, reemplazar este tipo numérico por una clase. Tampoco tiene efecto en la mantenibilidad.

- Replace Type Code with Subclasses: esta refactorización consiste en, en una clase que tiene un código de tipo numérico que sí afecta al comportamiento de la clase, reemplazar este tipo numérico por una clase. Tampoco tiene efecto en la mantenibilidad.
- Replace Subclass with Fields: esta refactorización se aplica cuando se tienen subclases que varían sólo en los métodos que devuelven datos constantes. La refactorización propone modificar los métodos en las superclases y eliminar las subclases. Su efecto en la Mantenibilidad es prácticamente inapreciable.

Como ha podido apreciarse, estas refactorizaciones que están más encaminadas a organizar los datos afectan en pequeña medida a la mantenibilidad. Se muestra un resumen del impacto de estas refactorizaciones en la Tabla A-4.

Refactorización	A	C	T	M
Self Encapsulate Field	o	o	o	O
Replace Data Value with Object	++	o	+	++
Change Value to Reference	o	o	o	O
Change Reference to Value	o	o	o	O
Replace Array with Object	+	o	++	++
Duplicate Observed Data	--	o	o	-
Change Unidirectional Association to Bidirectional	o	o	o	O
Change Bidirectional Association to Unidirectional	o	o	o	O
Replace Magic Number with Symbolic Constant	o	o	o	O
Encapsulate Field	o	o	o	O
Encapsulate Collection	+	o	o	+
Replace Record with Data Class	o	o	o	O
Replace Type Code with Class	o	o	o	O
Replace Type Code with Subclasses	o	o	o	O
Replace Subclass with Fields	o	o	o	O

Tabla A-4. Resultados de refactorizaciones "Organizando los datos"

A.2.5 Grupo de refactorizaciones 4 "simplificando expresiones condicionales"

- Decompose Conditional: esta refactorización se aplica cuando se tiene una sentencia condicional complicada. Propone extraer métodos de la condición, de la parte del "then" y la parte del "else". Al extraer métodos, mejora la Analizabilidad. Además, al contar con una lógica condicional más sencilla, se

facilita la creación de pruebas para la condición. Por lo tanto, también mejora la Capacidad de Ser Probado.

- Consolidate Conditional Expression: esta refactorización se aplica cuando se tiene una secuencia de condicionales que devuelven el mismo resultado. Consiste en agrupar estas expresiones en una sola y extraerla posteriormente en un método. Esta refactorización ayuda a mejorar la Analizabilidad del código fuente y, de igual manera que el anterior, la Capacidad de Ser Probado.
- Consolidate Duplicate Conditional Fragments: esta refactorización se aplica cuando se tiene el mismo fragmento de código en todas las ramas de una expresión condicional. La refactorización propone sacar de la condición el método. Esta refactorización disminuye el código duplicado y las líneas de código, mejorando la Analizabilidad.
- Remove Control Flag: esta refactorización propone la eliminación de las variables que son utilizadas como un flag para abandonar un bucle o una serie de expresiones condicionales, sustituyéndolas por una expresión break o return. Esta refactorización no tiene efecto en la mantenibilidad.
- Replace Nested Conditional with Guard Clauses: esta refactorización se aplica cuando se tiene un comportamiento condicional que no deja claro el camino condicional de la expresión. Propone sustituir la expresión condicional por varios condicionales con diferentes puntos de salida. Esta refactorización ayuda a conseguir una mejor cobertura de pruebas del código fuente, al disminuir los caminos posibles al desaparecer las sentencias else.
- Replace Conditional with Polymorphism: esta refactorización propone la utilización de polimorfismo cuando se tienen sentencias condicionales de manera que eligen el comportamiento en función del tipo de un objeto. Esta refactorización mejora la Analizabilidad, al reducir las líneas de código. Además, mejora la Capacidad de Ser Probado, ya que facilita la creación de pruebas unitarias al disminuir las expresiones condicionales.
- Introduce Null Object: esta refactorización propone que, cuando se tienen comprobaciones repetidas de un valor nulo, se reemplace este valor nulo con un objeto nulo. Esta refactorización ayuda a la reducción del código duplicado, por lo que aumenta la Analizabilidad. También ayuda a eliminar comprobaciones, lo que reduce la complejidad ciclomática y facilita la Capacidad de Ser Probado.

- Introduce Assertion: esta refactorización indica que, cuando se tiene una sección de código que asume algo sobre el estado del programa, se utilicen aserciones para hacerlo explícito. De esta manera se ayuda a la depuración y a la identificación de errores. Esta aserción añade una línea de código, por lo que puede considerarse que apenas aumenta las líneas de código y, por lo tanto, disminuiría la Analizabilidad. Por lo tanto, no tiene efecto en la Mantenibilidad.

Estas refactorizaciones afectan principalmente a la Analizabilidad y la Capacidad de Ser Probado del producto. Los resultados pueden apreciarse en la Tabla A-5.

Refactorización	A	C	T	M
Decompose Conditional	+	o	+	+
Consolidate Conditional Expression	+	o	+	+
Consolidate Duplicate Conditional Fragments	++	o	o	+
Remove Control Flag	o	o	o	o
Replace Nested Conditional with Guard Clauses	o	o	+	+
Replace Conditional with Polymorphism	+	o	++	++
Introduce Null Object	+	o	+	+
Introduce Assertion	o	o	o	o

Tabla A-5. Resultados de refactorizaciones "Simplificando expresiones condicionales"

A.2.6 Grupo de refactorizaciones 5 "haciendo más simples las llamadas a métodos

- Rename Method: esta refactorización consiste en modificar el nombre de un método para hacerlo más adecuado a la función que realiza. No tiene ningún efecto en la Mantenibilidad.
- Add Parameter: esta refactorización propone la introducción de un parámetro en la llamada a un método cuando se necesita más información de quien lo llama. Esta refactorización tampoco afecta a la Mantenibilidad ni a sus características.
- Remove Parameter: esta refactorización es la opuesta a la anterior. Cuando no se vuelve a utilizar un parámetro en el cuerpo de un método, debería ser eliminado. Tampoco tiene ningún efecto en la Mantenibilidad.
- Separate Query from Modifier: esta refactorización se utiliza cuando un método que devuelve un valor además modifica el estado de un objeto. La

refactorización propone crear dos métodos, uno para devolver el valor y otro para la modificación. Esta refactorización afecta ligeramente la Capacidad de Ser Probado, ya que contar con dos métodos diferentes facilita las pruebas frente a tener un método con dos funcionalidades.

- Parameterize Method: esta refactorización propone que, cuando se tienen varios métodos que hacen cosas similares pero con diferentes valores que aparecen en el cuerpo del método, sustituirlos por un método que pase como parámetro ese valor. Esta refactorización afecta positivamente a la Analizabilidad, disminuyendo en gran medida las líneas de código.
- Replace Parameter with Explicit Methods: esta refactorización se aplica cuando se tiene un método que ejecuta un código diferente en función del valor que se le pasa en un parámetro que se le pasa y que actúa como discriminador. Propone crear un método para cada valor que puede tomar ese parámetro. Esta refactorización consigue aumentar la Capacidad de Ser Probado y la Analizabilidad, ya que ayuda a disminuir la complejidad ciclomática.
- Preserve Whole Object: esta refactorización se aplica cuando se pasa como parámetros a un método varios valores o atributos de un objeto. La refactorización propone pasar como parámetro el objeto directamente. Esta refactorización no tiene ningún efecto en la Mantenibilidad según el modelo de medición utilizado.
- Replace Parameter with Method: esta refactorización se aplica cuando se tiene un objeto que almacena el resultado de una llamada a un método, y luego este objeto es pasado como parámetro en otro método. Se propone eliminar el parámetro y hacer que el método realice la llamada al otro método. Esta refactorización no tiene efecto directo en la Mantenibilidad.
- Introduce Parameter Object: esta refactorización propone que, cuando se tiene un conjunto de parámetros que por su naturaleza tienden a ir siempre juntos, crear un objeto y que sea este el que se pase como parámetro. De igual manera, no afecta a la Mantenibilidad ni a sus características.
- Remove Setting Method: esta refactorización se aplica cuando un atributo es fijado en tiempo de creación y nunca es modificado. Propone la eliminación del método set para el atributo. Esta refactorización aumenta ligeramente la Analizabilidad, al reducir las líneas de código.
- Hide Method: esta refactorización indica que, si un método no es utilizado por otra clase, hacerlo privado. Su aplicación no tiene efecto en la Mantenibilidad.

- Replace Constructor with Factory Method: esta refactorización propone que, si es necesario tener algo más que un constructor simple, reemplazar el constructor con una factoría. En este caso, la introducción de un nuevo constructor hace que aumenten las líneas de código. Por lo tanto, puede indicarse que la refactorización disminuye la Analizabilidad.
- Encapsulate Downcast: esta refactorización se aplica cuando un método devuelve un objeto que necesita ser transformado a su tipo (mediante un cast) tras la llamada. La refactorización propone realizar el casting en el cuerpo del método, y devolver un objeto del tipo necesario. Esta refactorización no afecta a la Mantenibilidad.
- Replace Error Code with Exception: esta refactorización indica que si un método devuelve un código especial para indicar un error, lance mejor una excepción. Esta refactorización ayuda a reducir las líneas de código y a disminuir la complejidad ciclomática, al eliminar parte de las condiciones de eliminar el código. Por lo tanto, mejora la Analizabilidad y la Capacidad de Ser Probado.
- Replace Exception with Test: esta refactorización se utiliza cuando una excepción es utilizada para reemplazar a una condición que debería haber sido comprobada antes. Se propone hacer que se compruebe antes la condición, eliminando la excepción. Esta refactorización no tiene efecto en la Mantenibilidad.

Este conjunto de refactorizaciones no afecta particularmente a la Mantenibilidad del producto. Al estar centradas en que las llamadas a los métodos sean más simples, no están tan dedicadas a mejorar la Mantenibilidad tanto como a hacer el código más legible. Los resultados pueden verse en la Tabla A-6.

Refactorización	A	C	T	M
Rename Method	o	o	o	O
Add Parameter	o	o	o	O
Remove Parameter	o	o	o	O
Separate Query from Modifier	o	o	+	O
Parameterize Method	++	o	o	+
Replace Parameter with Explicit Methods	+	o	+	+
Preserve Whole Object	o	o	o	O
Replace Parameter with Method	o	o	o	O
Introduce Parameter Object	o	o	o	O
Remove Setting Method	+	o	o	+
Hide Method	o	o	o	O

Refactorización	A	C	T	M
Replace Constructor with Factory Method	-	o	o	-
Encapsulate Downcast	o	o	o	O
Replace Error Code with Exception	++	o	+	++
Replace Exception with Test	o	o	o	O

Tabla A-6. Resultados de refactorizaciones "Haciendo más simples las llamadas a métodos"

A.2.7 Grupo de refactorizaciones 6 "tratando con la generalización"

- Pull Up Field: esta refactorización implica que, cuando dos o más subclases tienen el mismo atributo, moverlo a la clase padre. Esta refactorización disminuye en muy pequeña medida las líneas de código, por lo que puede decirse que no afecta a la Mantenibilidad.
- Pull Up Method: similar a la refactorización anterior, pero con un método. Sin embargo, en este caso se elimina código repetido y se reducen las líneas de código. Por lo tanto, si aumenta la Analizabilidad. Además, permite tener que realizar menos pruebas unitarias y mejorar la cobertura de pruebas, con lo que también mejora de Capacidad de Ser Probado.
- Pull Up Constructor Body: similar a los dos anteriores, pero con el cuerpo de los constructores. En este caso se movería el cuerpo del constructor a la clase padre, realizando la llamada a éste desde los constructores de las clases hijas. Esta refactorización reduce ligeramente el código repetido y las líneas de código, por lo que mejora la Analizabilidad.
- Pus Down Method: esta refactorización se aplica cuando un método de la clase padre es sólo utilizado por alguna de sus subclases. Se recomienda en este caso introducir el método en las subclases, eliminándolo de la clase padre. Esta refactorización promueve la aparición de código repetido y que aparezcan nuevas líneas de código, por lo que empeora la Analizabilidad. De igual manera, hace que empeore la cobertura de pruebas y que sean necesarias nuevas pruebas unitarias. Por lo tanto también empeora la Capacidad de Ser Probado.
- Push Down Field: similar al anterior pero aplicada a los atributos. Aunque aumenta ligeramente las líneas de código, apenas tiene efecto en la Mantenibilidad.

- Extract Subclass: esta refactorización se aplica cuando se tiene una clase que tiene características que solo son utilizadas en determinadas instancias. En este caso, se propone crear una subclase para los objetos de este tipo. Esta refactorización no afecta a ninguna de las características de la Mantenibilidad.
- Extract Superclass: esta refactorización se aplica cuando dos clases tienen características similares. De esta manera, se propone crear una clase padre que contenga todas las características comunes. Esta refactorización ayuda a mejorar la Analizabilidad al reducir el código repetido y las líneas de código. De igual manera, mejora la cobertura de pruebas y disminuye la cantidad de pruebas necesarias.
- Extract Interface: esta refactorización se aplica cuando un conjunto de características de una clase es utilizado por varios clientes. Se propone la creación de una interfaz que permita realizar las llamadas a través de ella. Este caso es similar al Extract Superclass, pero al ser una interfaz no tendrá el código implementado. Por lo tanto, su efecto en la Mantenibilidad es prácticamente nulo.
- Collapse Hierarchy: esta refactorización se aplica cuando una clase y una subclase no son muy diferentes. En este caso se propone combinar ambas clases en una sola. Este caso se podía considerar la unión de aplicar Pull Up Method, Push Down Method, Pull Up Field y Push Down Field, por lo que en teniendo en cuenta esto no tendrá efecto en la Mantenibilidad.
- Form Template Method: esta refactorización se da cuando se tienen dos métodos en subclases que aplican los mismos pasos, pero estos pasos son aplicados en distinto orden. La refactorización propone identificar es agrupar cada grupo de pasos en métodos nuevos implementados en la clase padre, utilizándolos en las clases hijas. Esta refactorización ayuda a mejorar la Analizabilidad, ayudando a evitar la repetición de código y disminuyendo por lo tanto las líneas de código. Además, permite que las pruebas sean más sencillas, alcanzando una mayor cobertura con menos pruebas.
- Replace Inheritance with Delegation: esta refactorización se aplica cuando una subclase utiliza sólo una parte de su clase padre o no quiere heredar los datos. En este caso se propone crear un atributo del tipo de la clase padre y ajustar los métodos para delegar en los métodos de la clase padre, eliminando así la herencia. Esta refactorización tiene un efecto negativo tanto en la Analizabilidad como en la Capacidad de Ser Probado. Al tener que crear nuevos métodos aumentan las líneas de código y hace que disminuya la cobertura de pruebas.

- Replace Delegation with Inheritance: esta refactorización es la opuesta a la anterior. En este caso, la Analizabilidad y la Capacidad de Ser Probado se ven favorecidas.

Este grupo de refactorizaciones afecta principalmente a la Analizabilidad y a la Capacidad de Ser Probado. En la Tabla A-7 se muestra un resumen de los resultados obtenidos:

Refactorización	A	C	T	M
Pull Up Field	o	o	o	o
Pull Up Method	++	o	+	++
Pull Up Constructor Body	+	o	o	+
Push Down Method	--	o	-	--
Push Down Field	o	o	o	o
Extract Subclass	o	o	o	o
Extract Superclass	++	o	+	++
Extract Interface	o	o	o	o
Collapse Hierarchy	o	o	o	o
Form Template Method	++	o	+	++
Replace Inheritance with Delegation	-	o	-	-
Replace Delegation with Inheritance	+	o	+	+

Tabla A-7. Resultados de refactorizaciones "Tratando con la generalización"

A.2.8 Grupo de refactorizaciones 7 "grandes refactorizaciones"

- Tease Apart Inheritance: esta refactorización aplica cuando se tiene una jerarquía de herencia que realiza dos trabajos a la vez. Se propone crear dos jerarquías diferentes y utilizar la delegación para invocar métodos de una a otra. En este caso la Cambiabilidad se ve afectada. Al reestructurar la jerarquía, es posible que se produzca una reducción en los ciclos que aparezcan. En cuanto a la Analizabilidad, se crea una nueva jerarquía, lo que puede provocar un aumento de líneas de código. Sin embargo, al tener la funcionalidad similar a la anterior, el impacto será leve.
- Convert Procedural Design to Objects: esta refactorización se aplica cuando se tiene código escrito de manera procedural. La refactorización indica que debe aplicarse la orientación a objetos, creando los objetos y métodos necesarios. Esta refactorización mejora la Analizabilidad, ayudando a redefinir los largos métodos procedimentales, haciendo que se comporten

como objetos. Por lo mismo, mejora la Capacidad de Ser Probado, mejorando el ratio de cobertura de pruebas, reduciendo la probabilidad de que haya errores (métodos más cortos se pruebas mejor) y facilita la creación de pruebas unitarias.

- Separate Domain form Presentation: esta refactorización se aplica cuando se tiene una serie de clases de presentación que contienen lógica de negocio. La refactorización indica que debe separarse la lógica de la presentación. Esta refactorización ayuda a mejorar la Cambiabilidad, reduciendo la posibilidad de aparecer ciclos. Además, mejora también la creación de pruebas unitarias, facilitándolas.
- Extract Hierarchy: esta refactorización se aplica cuando se tiene una clase que realiza demasiado trabajo, en ocasiones a través de muchas expresiones condicionales. Se propone crear una jerarquía de clases que permitirá representar los casos especiales. Esta refactorización permite mejorar todas las características. La Analizabilidad ya que se reduce la complejidad ciclomática al desaparecer los condicionales. De igual manera se mejora la Capacidad de Ser Probado. Por último, al dividir una clase grande en varias más pequeñas es posible que se eliminen ciclos, mejorando la Cambiabilidad.

Estas refactorizaciones, como era de esperar al ser refactorizaciones de mayor volumen, tienen un gran efecto en la Mantenibilidad y en sus características. En la Tabla A-8 puede observarse el resumen de los resultados del análisis para este grupo de refactorizaciones:

Refactorización	A	C	T	M
Tease Apart Inheritance	o	+	o	+
Convert Procedural Design to Objects	+	o	++	++
Separate Domain form Presentation	o	+	++	++
Extract Hierarchy	+	+	++	++

Tabla A-8. Resultados de refactorizaciones “Grandes refactorizaciones”

A.2.9 Resultados del análisis

Tras la realización del análisis pueden extraerse diferentes conclusiones sobre los datos. En primer lugar, más de la mitad de las refactorizaciones (40 de las 72 analizadas) tienen un efecto en la Mantenibilidad. Se estas, el 80% (32) tienen un efecto positivo, mientras que 8 refactorizaciones tienen un efecto

negativo. Las 30 restantes no tienen ningún efecto en la Mantenibilidad o su efecto es tan pequeño que puede considerarse nulo. En la Figura A-1 pueden observarse los resultados obtenidos.

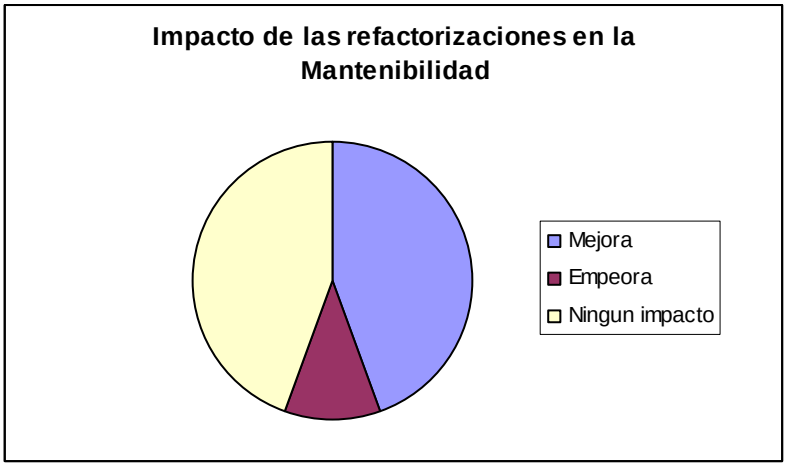


Figura A-1.- Impacto de las refactorizaciones en la Mantenibilidad

En cuanto a la Analizabilidad, de las subcaracterísticas de la Mantenibilidad es la que más se ve influida. 26 refactorizaciones mejoran la Analizabilidad, mientras que tan solo 7 la empeoran. En la siguiente figura puede observarse la distribución obtenida respecto al efecto de las refactorizaciones en la Analizabilidad.

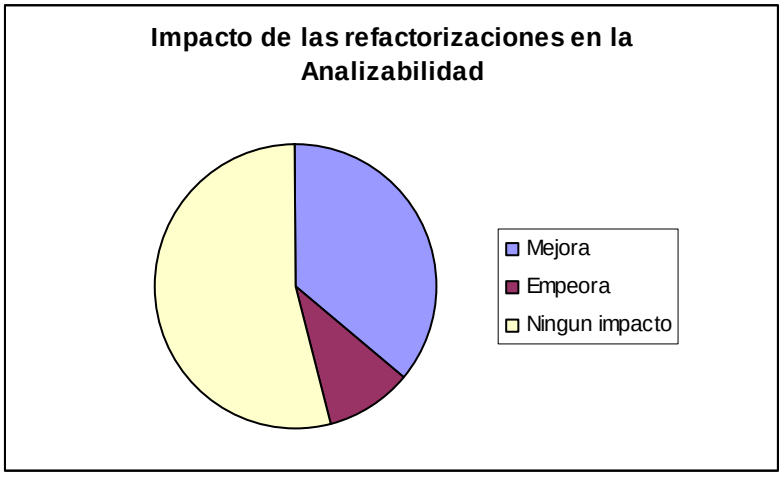


Figura A-2. Impacto de las refactorizaciones en la Analizabilidad

En cuanto a la Capacidad de Ser Cambiado, también es influida en gran medida, teniendo efecto en ella un cuarto de las refactorizaciones. De igual manera en la Figura A-3 puede verse la distribución. Por último, la Cambiabilidad se ve menos influida por las refactorizaciones, sólo teniendo impacto 8 de las refactorizaciones en su valor. Pueden verse también los resultados en la Figura A-4.

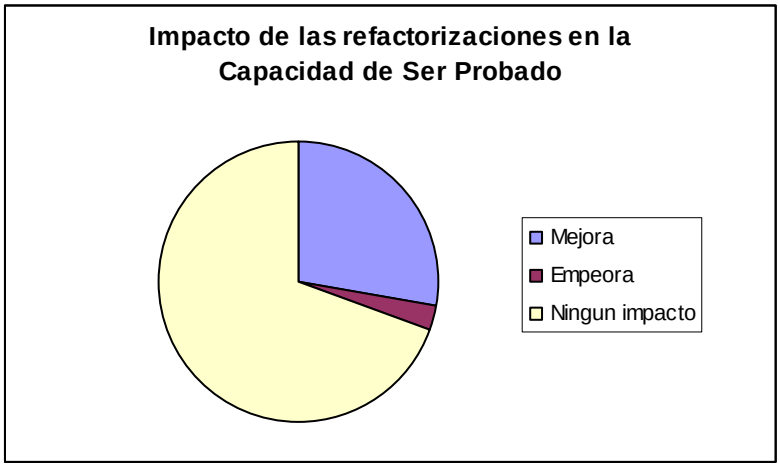


Figura A-3. Impacto de las refactorizaciones en la Capacidad de Ser Probado

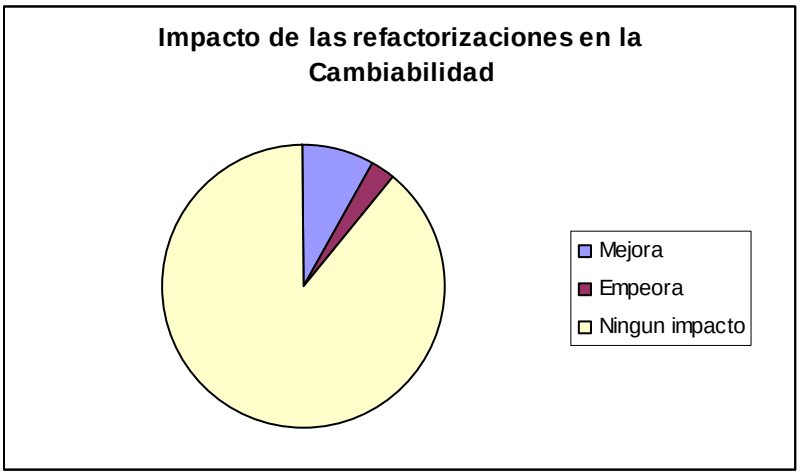


Figura A-4. Impacto de las refactorizaciones en la Cambiabilidad

En la Figura A-5 puede verse un resumen de los resultados, destacándose la proporción de las refactorizaciones que influyen para cada característica analizada.

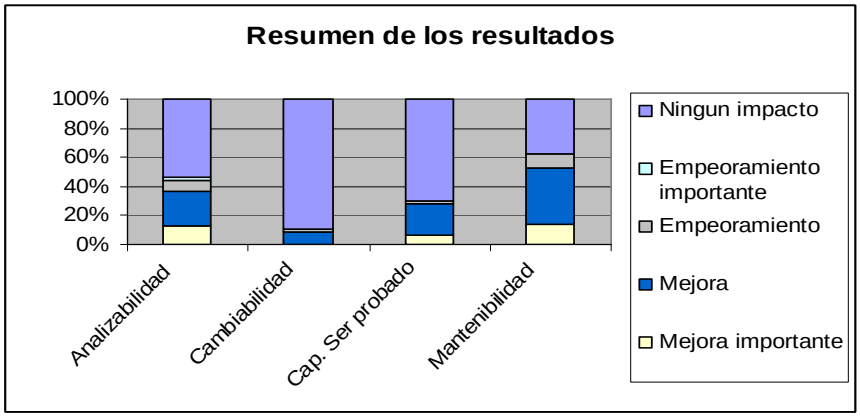


Figura A-5. Resumen de los resultados

Apéndice B: El Entorno Visual de KEMIS

En este apéndice se detallará el entorno visual del proyecto KEMIS. En la sección B.1 se comentarán las diferentes librerías analizadas para el desarrollo de visualizaciones. En la sección B.2 se realiza un análisis de las visualizaciones elegidas para el entorno KEMIS. Por último, en la sección B.3 se indicarán las decisiones de arquitectura tenidas en cuenta al desarrollar KEMIS.

B.1 Librerías para el Desarrollo de Visualizaciones

Una de los objetivos básicos que pretende alcanzar este proyecto es el de mostrar los datos que devuelve KEMIS en visualizaciones sencillas y claras, pero que a la vez sean altamente informativas.

Existen gran cantidad de librerías que permiten realizar visualizaciones de datos. Para la realización de este trabajo, se han analizado muchas de estas librerías para, gracias al conocimiento profundo de las mismas, decidir cuáles eran las más apropiadas para la aplicación que se pretende realizar.

En las siguientes secciones se muestran algunas de las librerías que se sopesaron en el momento de comenzar con la realización del proyecto.

B.1.1 Google Visualization API

La API de visualizaciones de Google [184] es una librería ofrecida como Servicio por Google. Permite elegir entre una gran cantidad de visualizaciones, entre los que destacan tablas, diagramas de barras, diagramas tipo radar, líneas del tiempo y mapas. Estas visualizaciones pueden ser integradas en una página Web que se comunicará con la API para realizar la presentación de los datos. Con una o varias de estas visualizaciones es posible crear informes o cuadros de mando que sirvan para analizar los datos mostrados. Además, permite crear, compartir y reutilizar las visualizaciones.

Su funcionamiento es muy sencillo, permitiendo obtener una visualización incluyendo únicamente código JavaScript y HTML en la aplicación Web. Es por tanto una librería recomendable en el caso de que el cuadro de mando se realice como una aplicación Web.

B.1.2 JFreeChart

JFreeChart [186] es una librería distribuida bajo licencia GNU Lesser General Public License (LGPL) [187] y desarrollada para Java, que permite introducir visualizaciones y gráficos de una manera sencilla en las aplicaciones desarrolladas en esta plataforma. Consiste en una API que permite la inclusión de gran cantidad de gráficos, como pueden ser gráficos circulares, gráficos de barras, histogramas, diagramas de Pareto y otra gran cantidad de visualizaciones.

Las visualizaciones que ofrece JFreeChart son sencillas y claras, aunque poco atractivas. Sin embargo, ofrecen una gran cantidad de información y pueden ser utilizadas de una manera fácil. Por lo tanto, es una librería recomendable en caso de querer realizar un cuadro de mando integrado en una aplicación Java. Pese a que su principal misión es integrarse con aplicaciones Java, también es sencillo integrarla en aplicaciones web,

B.1.3 The JavaScript InfoVis Toolkit

The JavaScript InfoVis Toolkit [188] es una librería desarrollada íntegramente en JavaScript y que permite realizar visualizaciones interactivas. Permite la inclusión de gran cantidad de visualizaciones, entre los que destacan treemaps, grafos radiales, grafos o árboles hiperbólicos, y otras visualizaciones. Estas visualizaciones permiten la realización de gráficos diferentes a las que se han mostrado hasta ahora. Además, estos gráficos pueden combinarse, permitiendo así conseguir unas visualizaciones mucho más informativas.

La manera de utilizar estas visualizaciones es integrándolas en una aplicación Web, por lo que es una librería recomendable en el caso de que se quieran visualizaciones atractivas e informativas, que permitan la interacción con el usuario. El principal problema que presenta esta librería es que algunas de sus visualizaciones no son soportadas por algunas versiones de los navegadores.

B.1.4 D3

D3 [189] es una librería realizada para JavaScript, anteriormente llamada Protovis. Permite la representación de datos basándose en primitivas sencillas como son barras o puntos. Estas primitivas pueden ser combinadas para generar una gran cantidad de visualizaciones, que permitirán mostrar los datos de una manera más atractiva para la persona que los esté observando. Además, la

composición de estas primitivas permitirá que las visualizaciones creadas contengan gran cantidad de información.

Esta librería también está pensada para el desarrollo de aplicaciones Web. Puesto que permite crear una gran cantidad de visualizaciones que pueden llegar a ser muy complejas, es recomendable usar esta librería en caso de querer representar gráficas avanzadas. El principal problema que presenta esta librería es que sus visualizaciones apenas permiten la interactividad con el usuario.

B.1.5 PlotKit

PlotKit [190] es una librería para JavaScript, distribuida bajo licencia BSD y que permite la realización de tablas y gráficos, que pueden ser incluidos en una aplicación Web. Entre las visualizaciones que permite realizar, se encuentran gráficos circulares, diagramas de barras o diagramas de líneas.

Tampoco permite una gran interacción con el usuario. Sin embargo, la sencillez de sus visualizaciones hace que sea una librería interesante a la hora de realizar gráficos que no requieran de una gran complicación en su representación. Además, es muy compatible con los navegadores, con lo que su integración es sencilla.

B.1.6 iGoogle Tabs

IGoogle [191] es la página personal que ofrece Google. En ella pueden introducirse gadgets en tabs. Estos tabs son cuadros en cuyo interior puede cargarse lo que se desee. La ventaja que tienen estos tabs es que pueden ser movidos, minimizados o eliminados, entre otras opciones.

iGoogle Tabs consiste en una serie de ficheros escritos en JavaScript que, combinados con una hoja CSS (Cascade Style Sheets) permiten la introducción de, entre otras muchas cosas, visualizaciones en el interior de un tab, permitiendo además que este pueda realizar las acciones arriba indicadas.

Esta librería permite gran dinamismo a la hora de mostrar las visualizaciones. Así, es posible moverlas en el cuadro de mando, minimizarlas cuando no interesen. Todo esto presentado con una interfaz sencilla e intuitiva, que es conocida y utilizada por un elevado número de usuarios.

B.2 Análisis de las Visualizaciones

Una vez llegado a este punto, es necesario seleccionar las visualizaciones que deben introducirse, así como las librerías que van a ser utilizadas para el desarrollo de las mismas.

Tras el análisis de los casos de uso, pueden distinguirse tres interfaces diferentes que han de ser realizadas:

- Una interfaz de resumen para mostrar los resultados obtenidos tras las ejecuciones de varios proyectos exitosos. Esta interfaz debe mostrar un pequeño resumen de los resultados obtenidos tras la ejecución de KEMIS sobre esos proyectos seleccionados.
- Otra interfaz que muestre todos los datos obtenidos para cualquier ejecución de KEMIS realizada sobre un proyecto, o sobre un módulo de un proyecto. Esta interfaz debe contener visualizaciones que permitan mostrar todos esos datos en una única interfaz.
- Una última interfaz que permita mostrar los datos históricos. En una primera aproximación, esta interfaz puede mostrar las mismas visualizaciones que la interfaz de resumen, aunque debería permitirse la ampliación de la interfaz con nuevas visualizaciones.

Tras analizar las interfaces necesarias, deben seleccionarse unas visualizaciones adecuadas que nos permitan mostrar la información del proyecto de una manera que sea útil para el usuario. Las visualizaciones elegidas se enumeran a continuación.

B.2.1 Treemap

El *TreeMap* es una visualización que permite la visualización de datos jerárquicos utilizando rectángulos. En ellos, el tamaño de cada rectángulo viene determinado por una característica de la entidad a la que hace referencia y relacionado proporcionalmente con el resto de entidades mostradas en él. De igual manera, los rectángulos del TreeMap pueden tener un color. De esta manera puede ser representada otra característica de la entidad que también será proporcional a las del resto de entidades. En la Figura B-1 puede verse un ejemplo de *TreeMap*.

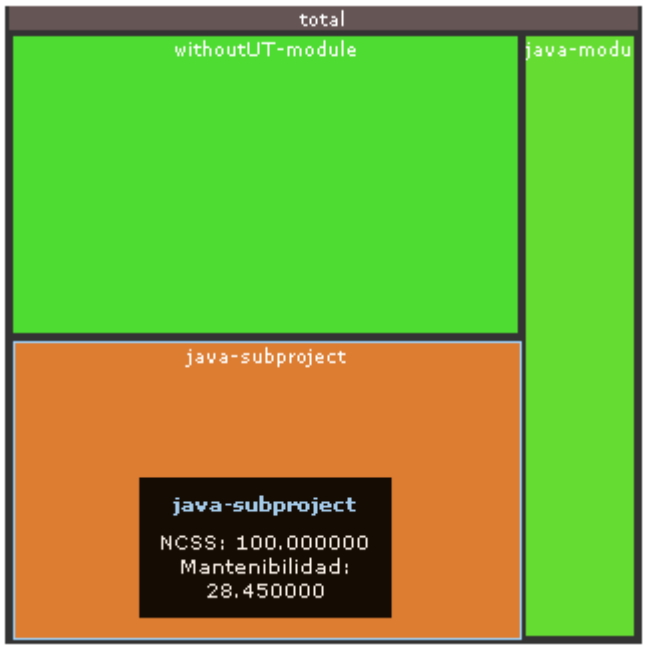


Figura B-1. Ejemplo de TreeMap utilizado

Esta visualización ha sido elegida porque representa los datos de una forma clara y muy eficaz, informando de dos características de una entidad de una manera inmediata para el usuario. Además, añadiendo a un TreeMap la capacidad de seleccionar las características van a ser representadas, permiten tener un gran conocimiento de los datos que se van a tratar sin un gran esfuerzo. Tras observar las alternativas, se ha considerado utilizar el TreeMap propuesto por la librería The JavaScript InfoVis Toolkit, descrita en la sección B.1.3.

B.2.2 Tablas

Aunque a priori sean un recurso poco atractivo visualmente, el uso de tablas está muy extendido para la representación de todo tipo de datos. Las tablas son visualizaciones sencillas, que permiten al usuario observar muchos datos sobre una determinada entidad de una forma rápida.

El problema principal que presentan las tablas es su poco dinamismo y su escasa riqueza visual. Esto puede resolverse mediante la utilización de tablas más avanzadas, que permitan la introducción de gráficos en su interior, así como de

tablas que permitan la ordenación, lo que posibilita poder realizar acciones sobre ellas.

De este último tipo es la tabla seleccionada, obtenida de la librería Google Visualization API. Esta tabla permite la ordenación de sus columnas, así como la posibilidad de formatear sus celdas o introducir código HTML en ellas. Un ejemplo de la tabla que se va a utilizar puede observarse en la Figura B-2.

Submódulo	Mantenibilidad	Analizabilidad	Cap. de ser Probado	Cambiabilidad	
withoutUT-module	82.12	78.12	70.12	90.12	Ver Historico
java-module	77.21	32.12	2.12	52.12	Ver Historico
java-subproject	28.45	12.14	25.45	65.45	Ver Historico

Figura B-2. Ejemplo de tabla utilizada

B.2.3 Diagrama de Kiviat

Como ya se ha comentado anteriormente, el diagrama de Kiviat o grafo de radar es una de las representaciones más apropiadas a la hora de mostrar la calidad de una entidad. La posibilidad de mostrar en sus ejes varias características que hacen referencia a una misma entidad hace que se adecue mucho a los datos que quieren mostrarse en la aplicación. Así, pueden representarse las subcaracterísticas de la mantenibilidad en esta visualización, permitiendo así hacerse una instantánea del estado de la mantenibilidad y de sus subcaracterísticas. Un ejemplo puede verse en la Figura B-3.

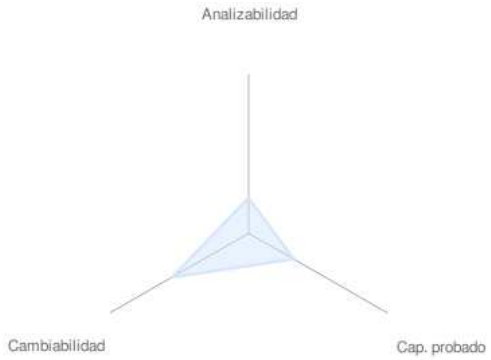


Figura B-3. Ejemplo de diagrama de Kiviat utilizado

Aunque no sea una visualización muy dinámica, el hecho de poder mostrar en ella varias características y de que esté recomendada para la visualización de la calidad hace que haya sido tenida en cuenta para la interfaz. Para representarlo, se va a hacer uso de la librería Google Visualization API.

B.2.4 Grafo radial o R-Graph

El grafo radial es un tipo de visualización radial que permite mostrar una jerarquía en árbol de manera que cada nodo del árbol se corresponde a un nodo de la visualización, mientras que las conexiones de ese nodo se corresponden con enlaces dentro del gráfico. Los nodos se colocarán en círculos concéntricos atendiendo a su nivel dentro del árbol y a las conexiones que establece. Un ejemplo de esta visualización puede verse en la Figura B-4.

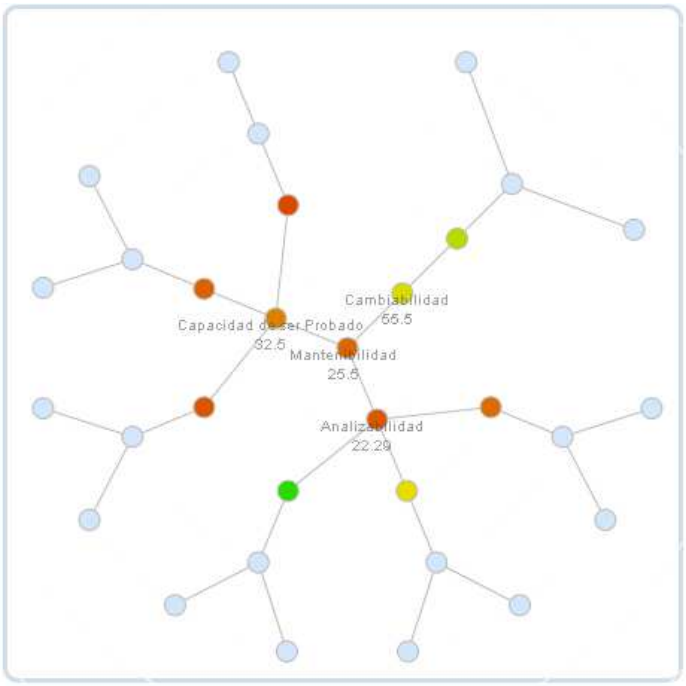


Figura B-4. Ejemplo de R-Graph utilizado

Esta visualización es especialmente útil en el caso de los datos obtenidos de KEMIS. Esto es debido a que KEMIS utiliza herramientas de software libre para extraer los atributos base del producto sobre el que se está ejecutando. Una vez

obtenidos estos datos, va infiriendo resultados de nivel superior hasta llegar a las subcaracterísticas de la mantenibilidad y, finalmente, a la mantenibilidad. Así, puede entenderse una estructura en árbol en la que la raíz sea la mantenibilidad y las hojas los atributos base, con el resto de datos calculados entre ambos. Para esta visualización se ha utilizado la librería The JavaScript InfoVis Toolkit, cuyo R-Graph permite el movimiento de los nodos, centrando la característica que se crea conveniente en cada momento.

B.3 Decisiones de Arquitectura

Según establece el modelo 4+1 [192] para la descripción del producto software, debido a la dificultad de representar un sistema en un único diagrama, es necesario describirlo mediante:

- La especificación de sus casos de uso, donde se describen los actores que interactúan con el sistema y las funcionalidades que este debe proporcionarles. Esta especificación se ha visto en la sección anterior.
- La vista de diseño, que describe las entidades presentes en el sistema, su funcionalidad y su propósito.
- La vista de implementación, que define la localización y finalidad de los ficheros que componen el sistema.
- La vista de procesos, que describe el funcionamiento del sistema desde una perspectiva de funcionalidad.
- La vista de despliegue o vista física, que indica la distribución de los diferentes componentes del sistema entre las distintas máquinas que lo forman.

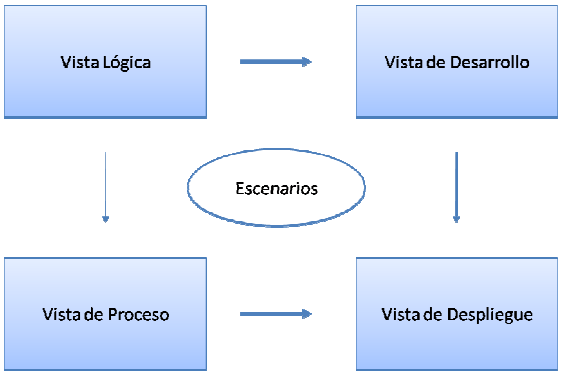


Figura B-5. Modelo 4+1 para la descripción del producto software

Las vistas de diseño, implementación, procesos y despliegue se definirán más adelante en las próximas secciones, haciendo uso del lenguaje de modelado UML v2.1 [193].

KEMIS cuenta con una capa de acceso a datos, en la que se utiliza la tecnología Hibernate [194] y una capa de negocio, en la que se hace uso de la tecnología que proporciona Spring.

Teniendo todo esto en cuenta, se desea realizar la capa de presentación de la aplicación, siguiendo para ello el patrón Modelo-Vista-Controlador (MVC) [195]. Este patrón de arquitectura promueve la separación de la aplicación en tres componentes diferentes: la vista, la lógica y los datos. De esta manera el usuario interactuará con la vista de la aplicación. El controlador gestionará y modificará el modelo según sea necesario, y finalmente delegará en la vista para la tarea de la representación de los datos en una interfaz de usuario. El esquema en el que se estructura la aplicación puede verse en la Figura B-.

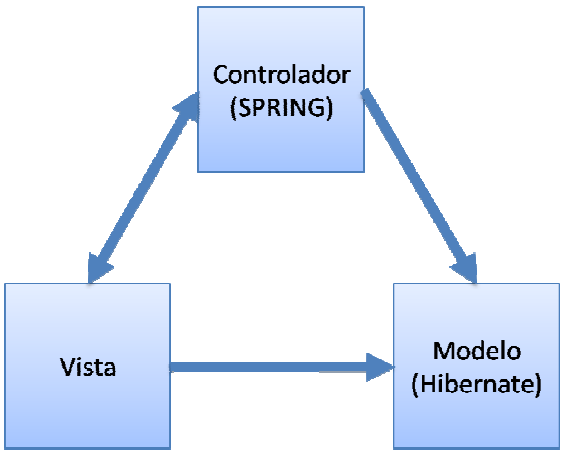


Figura B-6. Modelo arquitectura MVC

En las próximas secciones veremos qué tecnologías se han utilizado para la capa de presentación de la aplicación.

B.3.1 Vista de Diseño

Como ya se ha comentado, la arquitectura de KEMIS implica un modelo que sigue el patrón MVC, en el que deben separarse las capas de presentación, lógica de negocio y datos. En este anexo nos centraremos en la capa de presentación. Con esto en mente, puede observarse en la Figura B-7 el diagrama

que muestra los distintos elementos que implementan cada vista, haciendo especial hincapié en la capa de presentación, pero sin olvidar el resto de capas, ya que son sobre las que se sustenta la capa de presentación.

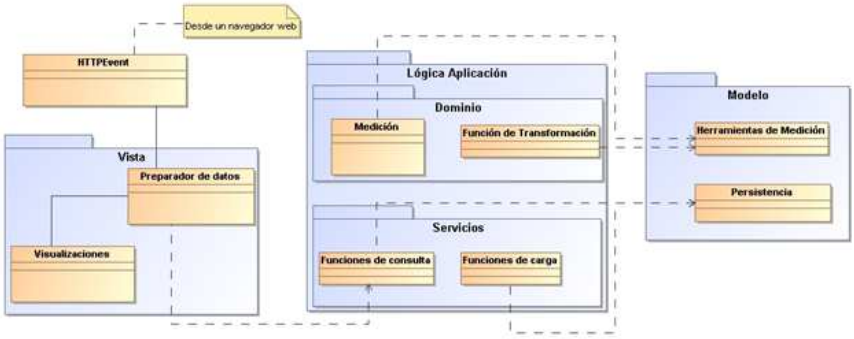


Figura B-7. Modelo lógico de la herramienta KEMIS

- **Modelo:** es la capa en la que se encuentra en Sistema Gestor de Bases de Datos (SGBD) y las herramientas necesarias para la realización de mediciones.
- **Lógica de la aplicación:** esta capa contiene la lógica de negocio de la aplicación. Pueden distinguirse dos paquetes distintos:
    - **Dominio:** es el paquete que contiene toda la lógica necesaria para la realización de la medición, así como las funciones de transformación que aparecen en ella.
    - **Servicios:** es el paquete que se ocupa de la comunicación entre el modelo y el dominio. Además, es el encargado de ofrecer servicios que serán utilizados por la capa de presentación.
- **Vista:** es la encargada de recibir las peticiones a través de un navegador Web. Dependiendo de la petición recibida, realizará las operaciones necesarias para preparar los datos que serán mostrados por las visualizaciones.

B.3.2 Vista de desarrollo

Esta sección trata da una visión sobre cómo se ha implementado la capa de presentación, así como la forma de implementar las relaciones que posee con el resto de capas de la aplicación.

En el diagrama de la Figura B-8 puede observarse el diagrama de implementación de la aplicación Web.

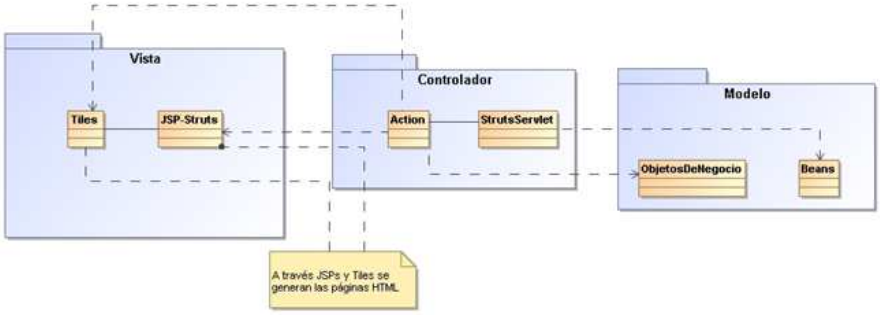


Figura B-8. Diagrama de implementación de la aplicación Web

Modelo: esta capa es la que contiene la relación entre la base de datos y la aplicación Web. Para su desarrollo se ha hecho uso de la tecnología de *beans*. Un *bean* es un componente software reutilizable que nos permite hacer uso de un objeto sin tener la necesidad de reprogramarlo. Así, convirtiendo los servicios ofertados por KEMIS en beans, es posible utilizarlos en toda la aplicación web, permitiéndonos así realizar las acciones necesarias sobre la base de datos.

Controlador: esta capa es la encargada de contener toda la lógica de negocio de la aplicación. Para su implementación, se han utilizado las ventajas que ofrece Struts2 [196] en el desarrollo de aplicaciones web. *Struts2* es un framework que permite el desarrollo de aplicaciones web de una forma sencilla, siguiendo en todo momento el uso del patrón MVC. Particularizando, Struts2 permite crear Acciones (*Actions*) que, mapeadas sobre una determinada petición a la aplicación web, realiza la ejecución de toda la lógica necesaria para dar una respuesta.

Para cada petición posible, se crea una *Action* que realizará toda la lógica perteneciente a la aplicación y, dependiendo del resultado de la misma, dirigirá al usuario hacia uno u otro resultado.

En la aplicación Web desarrollada para el cuadro de mando, se han identificado tres acciones principales, una para la petición de los resultados de las mediciones sobre proyectos exitosos, otra para la petición de los resultados de un proyecto y otra para la petición del histórico de un proyecto. Para cada una de ellas, si el resultado es exitoso se dirigirá al usuario a la página correspondiente que incluirá las peticiones. En caso contrario, se mostrará un mensaje de error indicando que no se ha podido realizar la solicitud.

Los objetivos principales de estas acciones son, en primer lugar identificar si la petición realizada es correcta y, en caso de ser así, generar los objetos con el formato necesario para que pueda ser utilizado por la aplicación.

Tras analizar las librerías utilizadas, se decide utilizar una librería que permite convertir un objeto Java en un objeto en JavaScript Object Notation (JSON) [197]. JSON es un formato ligero para el intercambio de datos. Es especialmente para poder representar objetos y arrays de datos. Al crearse un objeto en formato JavaScript, puede ser utilizado por las librerías directamente o, en caso de que estas no reciban un objeto JSON, puede transformarse de una manera sencilla en cualquier otro tipo de objeto JavaScript.

B.3.3 Interacción con el usuario

Esta capa es la encargada de la interacción con el usuario. Para su implementación, se ha utilizado la tecnología que ofrece *JavaServer Pages* (JSP) [198] y *Tiles 2* [200]. La combinación de estas dos tecnologías, unido a las etiquetas que *Struts2* ofrece para el manejo de los objetos desde la capa de presentación, hacen posible la generación del código HTML que será mostrado en el navegador Web.

Tiles 2 es una extensión que facilita el desarrollo de aplicaciones Web. En particular, permite la definición de fragmentos de código que pueden ser incluidos en cualquier página. Esta tecnología se integra con *Struts2* de manera que el código generado tras la ejecución de un tile pueda ser después transformado en código HTML.

Haciendo uso de la combinación de *Tiles 2*, *JSPs*, las etiquetas ofrecidas por *Struts2* y las librerías de visualizaciones creadas en JavaScript, se ha desarrollado la visualización del cuadro de mando. Para ello, se define en primer lugar un plantilla que seguirán todas las páginas que se van a generar. En esta plantilla se define las partes comunes para todas las páginas que se van a generar, y se permite la inclusión de tiles en las partes diferentes para cada aplicación. En concreto, para la aplicación generada se definen como partes diferentes las siguientes:

- El *header* o cabecera: cabecera de la aplicación, que contendrá la imagen y el logo de la empresa cliente de la aplicación.
- La barra de navegación: barra que se incluye debajo del header y que permite navegar entre diferentes sitios de la página. En una versión inicial, siempre contendrá un enlace a la página de inicio de la aplicación.
- Un *footer* o pie de página: que aparecerá abajo al final de cada página.

- El menú, que mostrará todas las opciones que se pueden realizar en la página. En principio, contendrá enlaces a todos los proyectos evaluados y a los históricos de los mismos.
- El contenido de la página, que será la parte en la que se incluirán las visualizaciones que se van a mostrar.
- El *head*, que permite la inclusión de todas las librerías y hojas de estilo que deben ser utilizadas por cada una de las páginas, ya que estos pueden diferir de unas páginas a otras.

Una vez que se identifican estas partes, se crea la plantilla como un jsp que contendrá etiquetas que indicarán donde debe introducirse el tile correspondiente.

Para la creación de cada una de las páginas, se cuenta con un fichero *tiles.xml*. En él se define, en formato XML, qué partes o *tiles* deben utilizarse para la generación de cada una de las páginas. Estos *tiles* serán también ficheros jsp que contendrán en su interior el código necesario para la generación de una página en concreto.

Además, en este fichero *tiles.xml*, pueden declararse preparadores. Un preparador es un código que se ejecuta antes de la inclusión del *tile*. En este caso, el uso de preparadores sirve de gran ayuda a la hora de realizar el menú, ya que permite conocer qué proyectos se mostrarán antes de realizar ninguna acción asociada a la página.

Una vez que se ha construido la página, se realiza la *Action* asociada a ella. Esta *Action* expondrá una serie de objetos para que puedan ser accedidos por el cliente Web. Haciendo uso de las etiquetas que *Struts2* permite utilizar, estos objetos y sus propiedades pueden ser accedidos desde JSP, consiguiendo así la obtención de los resultados de una petición.

Por último, una vez que se tienen los datos reales, éstos son utilizados por las librerías de visualización para generar los gráficos. Una vez que se generan todas las visualizaciones, se muestra la página al usuario con los resultados de la petición que había realizado.

La Figura B-9 muestra cómo se realiza la ejecución de una petición.

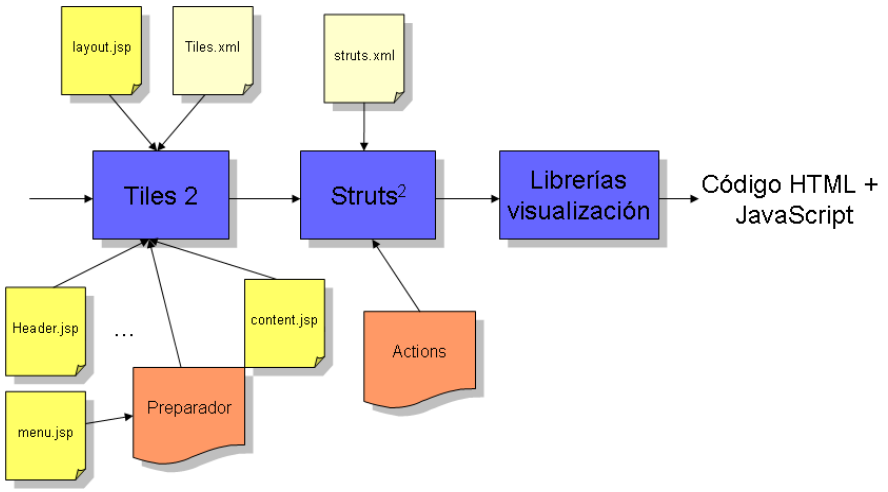


Figura B-9. Etapas para la construcción de la salida a una petición

B.3.4 Vista de procesos

La aplicación Web diseñada para el cuadro de mando cuenta con un único proceso para responder a las necesidades del usuario. El proceso identificado consiste en la generación de un cuadro de mando con los datos solicitados por parte del usuario. Independientemente de cuál sea la petición del usuario, el proceso que se sigue es el mismo.

Inicialmente se recibe la petición del usuario y, dependiendo de cuál sea, se realiza una consulta sobre la base de datos haciendo uso de los servicios ofertados por KEMIS. Una vez recibidos los resultados, se generan los datos para que puedan ser utilizados por las librerías de visualizaciones. En el momento que ya se tienen los datos de una manera adecuada, se muestra al usuario la salida correspondiente en su navegador web.

En la Figura B-10, se presenta la relación de este proceso y los demás elementos que existen dentro de KEMIS. De esta manera puede apreciarse las relaciones que existen entre los diferentes procesos y como se integra la aplicación web en el producto.

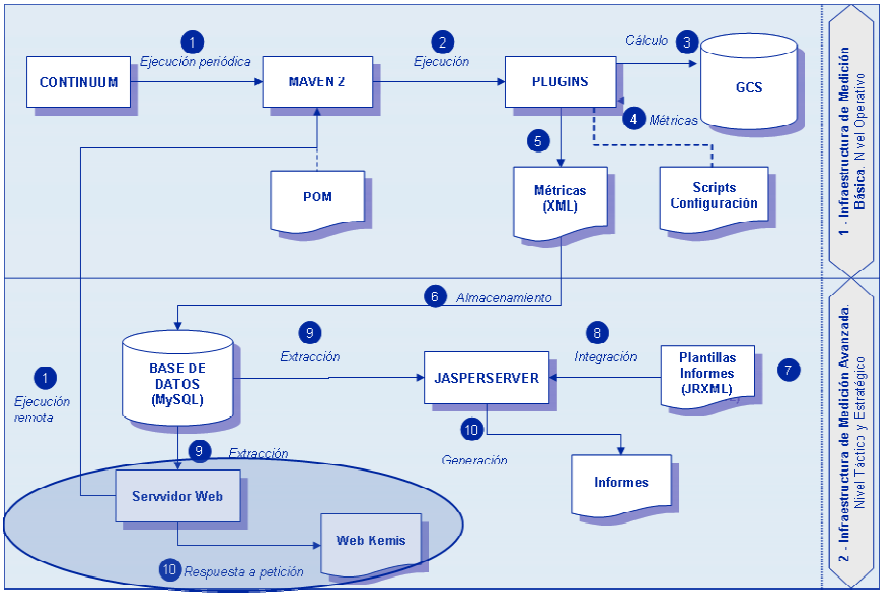


Figura B-10. Modelo de procesos KEMIS

B.3.5 Vista de despliegue

El cuadro de mando realizado se ejecuta como una aplicación distribuida entre tres nodos:

- El primero es el "Servidor de Bases de Datos", que contiene la base de datos y el modelo de datos que dará persistencia a las mediciones que se hayan realizado utilizando KEMIS. Esta base de datos puede ser de cualquier tipo que sea aceptado por KEMIS. En la actualidad, la base de datos que se utiliza es MySQL 5.0 [201].
- El segundo nodo es el "Servidor Web", que contiene el despliegue de la aplicación. En él se encuentran todos los archivos y librerías necesarios para mostrar el cuadro de mando. También se utiliza un servidor Apache Tonca 5.5 (Tomcat) como servidor de aplicaciones. Este nodo será el encargado de recibir las peticiones de un cliente y servirle las respuestas cuando estas estén disponibles.
- Por último, el nodo "Cliente" es el ordenador desde el cual el cliente accede al "Servidor Web" y comienza a realizar peticiones a la misma. Esta conexión se realiza a través del protocolo HTTP.

La distribución indicada anteriormente puede verse en la Figura B-11.

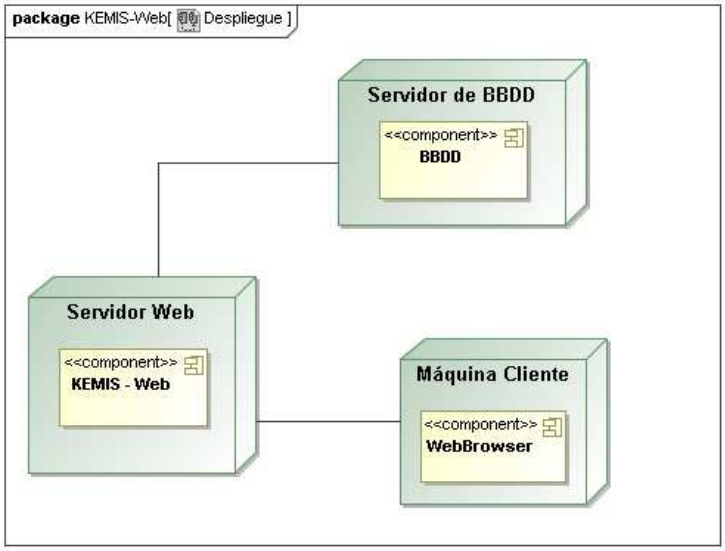


Figura B-11. Diagrama de despliegue del cuadro de mando

B.4 Resultado Final del Cuadro de Mando

El cuadro de mando implementado durante el desarrollo de esta Tesis se encuentra actualmente integrado con KEMIS, sirviéndole como la interfaz que muestra los resultados de las mediciones realizadas sobre los proyectos.

Tres son las interfaces principales, que permiten ver las mediciones realizadas sobre un proyecto y sus módulos, los datos históricos de los mismos y un resumen de los resultados de los proyectos exitosos. En las siguientes figuras pueden observarse unos ejemplos de la interfaz resultante.

En la Figura B-12 puede observarse la interfaz que recoge los resúmenes de algunos proyectos que ya han sido analizados por KEMIS. Como puede verse, en esta primera interfaz agrupa los datos en una tabla resumen que permite visualizar los datos principales obtenidos para cada proyecto. Además, permite acceder al proyecto y a su histórico.

Figura B-12. Interfaz realizada para el resumen de proyectos (tabla)

La Figura B-13 muestra la otra visualización que permite hasta ahora la página de resumen. En ella se muestran los datos de resumen de proyectos agrupados en un TreeMap. Esta forma de visualización permite identificar rápidamente las diferencias existentes entre un proyecto y otro, en la característica seleccionada. Esta característica puede ser modificada con los menús desplegables de la derecha.

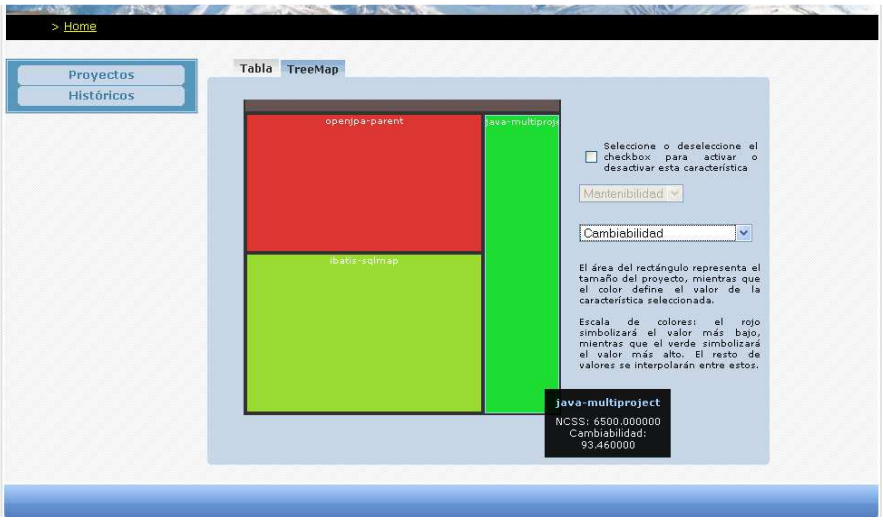


Figura B-13. Interfaz realizada para el resumen de proyectos (TreeMap)

En la Figura B-14 puede verse la interfaz realizada para la visualización del histórico de un proyecto. Como puede observarse, la tabla mostrada es similar a la de la pantalla de resumen. Sin embargo, los datos mostrados son diferentes. Con esto se consigue dotar de uniformidad a las vistas de la interfaz.

Figura B-14. Interfaz realizada para histórico de un proyecto (Tabla)

Similar es lo que ocurre con la Figura B-15, que muestra la otra visualización disponible de los proyectos. Tanto desde el TreeMap que aparece en la interfaz como desde la tabla, puede accederse a cada una de las mediciones anteriores que aparecen.

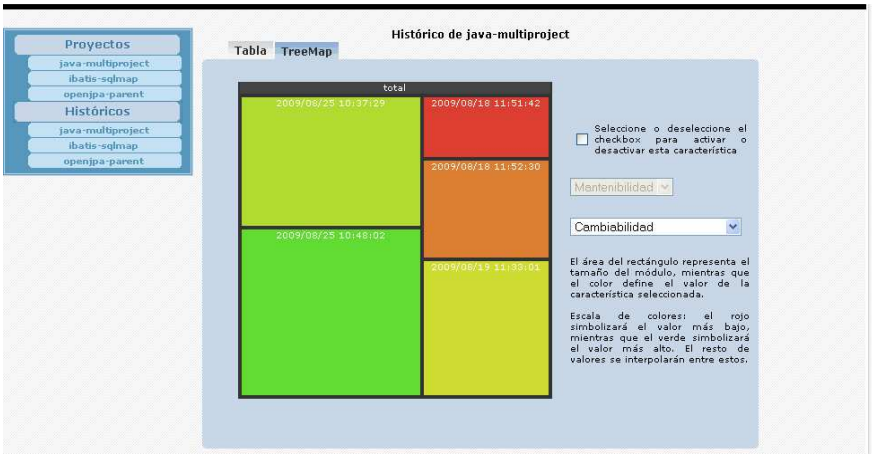


Figura B-15. Interfaz realizada para el histórico de un proyecto (*TreeMap*)

Por último, en las siguientes figuras pueden verse las visualizaciones incluidas en la interfaz que se ha realizado para la visualización de los resultados de un proyecto. Esta interfaz muestra unas visualizaciones diferentes que permiten establecerse una imagen rápida del estado actual del proyecto. Entre las visualizaciones que aparecen en esta interfaz se encuentran un R-Graph dinámico que muestra el estado de todas las métricas extraídas o calculadas del proyecto (ver Figura B-16). Este R-Graph permite mostrar en el centro el nodo que se desee. Además, cuenta también con un diagrama de Kiviat que muestra las subcaracterísticas de la mantenibilidad según la norma ISO/IEC 9126 (ver Figura B-17).

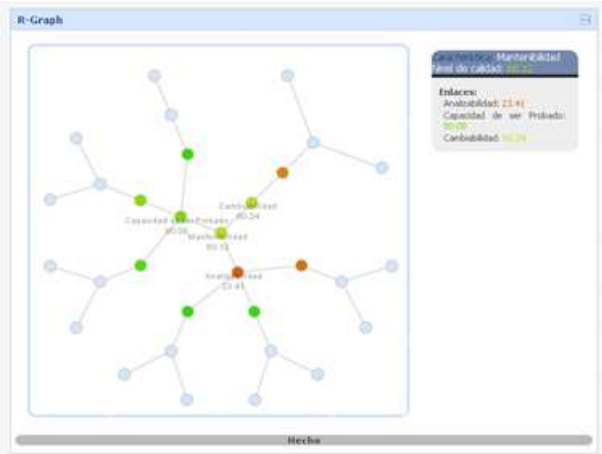


Figura B-16 R-Graph incluido en la página de visualización de un proyecto

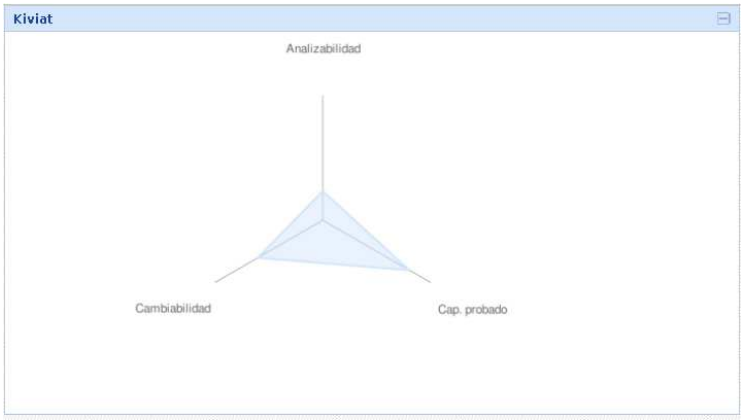


Figura B-17. Diagrama de Kiviat incluido en la página de proyecto

También se puede visualizar una tabla y un *TreeMap* con el resumen de sus módulos (ver Figura B-18). Por último, cuenta con una tabla que muestra, ordenada por niveles, todos los atributos con sus respectivos valores (ver Figura B-19). Todas estas visualizaciones se encuentran introducidas en ventanas que pueden ser minimizadas o movidas a la posición que se desee dentro de la interfaz.

Con estas visualizaciones se obtiene una visión global del estado del proyecto. Además, al ser visualizaciones tan interactivas permiten al usuario realizar acciones con ellas que permitirán que el cuadro de mando se acople a las necesidades del usuario.

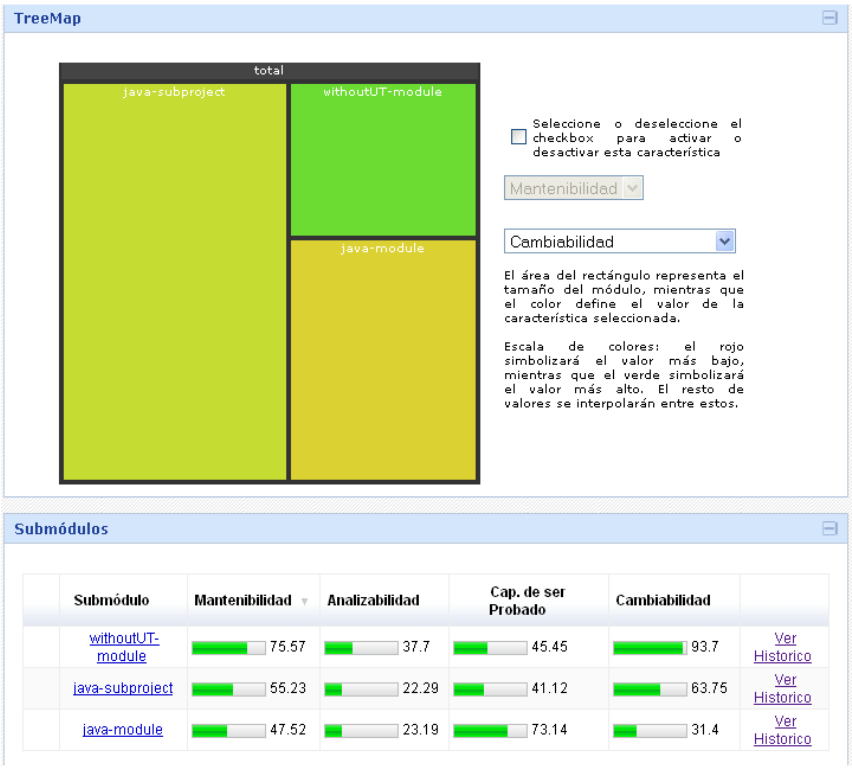


Figura B-18. *TreeMap* y tabla incluidos en la página de proyecto

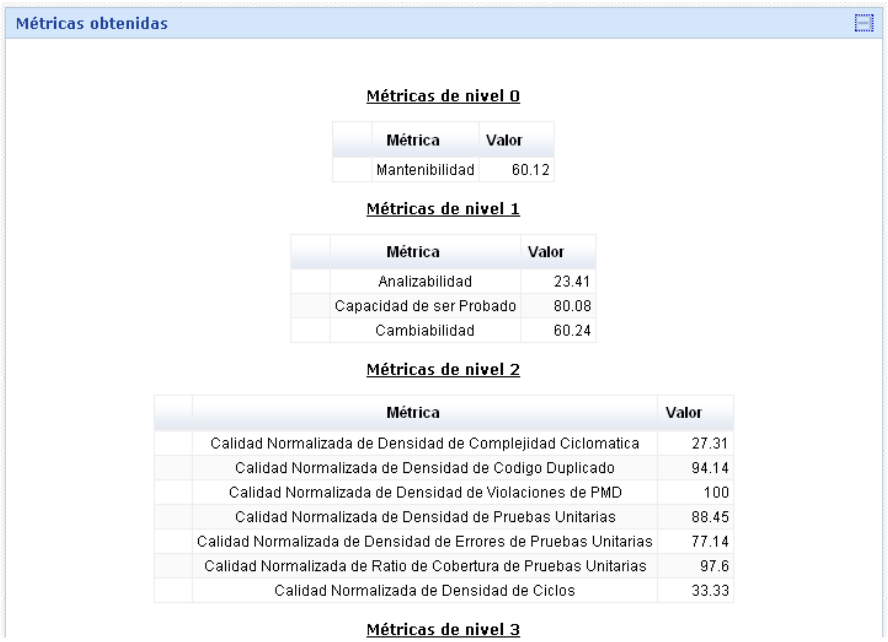


Figura B-19. Tablas resumen incluidas en la página del proyecto

B.5 Conclusiones Sobre las Visualizaciones

Como ha podido observarse, existen una gran cantidad de librerías que permiten el desarrollo de diferentes tipos de visualizaciones. Estas visualizaciones pueden ser más o menos complejas, pero todas ellas son de gran utilidad a la hora de representar datos.

Estas librerías se encuentran en continua evolución, y además tienen un uso muy extendido en aplicaciones de muy diferentes campos. Es por ello que se considera que puedan ser unos gráficos adecuados para la realización del cuadro de mando para el proyecto KEMIS.

La mayoría de las librerías antes citadas están diseñadas para la inclusión de las visualizaciones dentro de una aplicación Web.

En esta línea, la herramienta KEMIS pretende combinar algunas de estas visualizaciones de manera que, aprovechando las ventajas que ofrece cada una de ellas, permita la realización de un cuadro de mando en el cual se disponga de toda la información que sea posible, pero que a la vez esté mostrada de una manera sencilla y adaptada a las necesidades del usuario de la aplicación. Además,

permite el aprovechamiento de las ventajas derivadas de la utilización de software libre, evitando grandes costes pero obteniendo a la vez visualizaciones de una alta calidad.

Apéndice C: Requisitos del Proyecto KEMIS

En este anexo se detallan la especificación de los requisitos funcionales de alto nivel y los casos de uso del proyecto KEMIS. Las especificaciones han sido validadas por el equipo técnico del Centro para el Desarrollo Tecnológico Industrial y han sido la base para las revisiones de los hitos del proyecto.

A partir de estas especificaciones se ha trabajado en el desarrollo del entorno tecnológico de la herramienta KEMIS.

C.1 Requisitos Funcionales

A continuación se muestra una breve descripción de los requisitos funcionales del sistema de más alto nivel.

C.1.1 RF_001: Medir la calidad del software

El sistema será capaz de realizar mediciones de calidad de cualquier producto software. Para ello, se encargará de recoger información de diferentes fuentes de datos (herramientas de evaluación de código, herramientas de evaluación de diseños, herramientas de ejecución de pruebas, sistemas de integración continua, sistemas de gestión de incidencias, etc.) en diferentes formatos (xml, csv, ficheros de texto plano) y los almacenará en una base de datos.

El objetivo de este proceso es homogeneizar la información acerca de la calidad de un producto, que se proporciona por las diferentes herramientas de las que disponen las organizaciones, que distan mucho de estar integradas.

C.1.2 RF_002: Realizar un cuadro de mando sobre la calidad del software

A partir de las mediciones de calidad recogidas por el sistema, se realizará una evaluación de la calidad del producto, que proporcionará indicadores de calidad para las diferentes características de acuerdo con un modelo de calidad del producto software (p. ej. Mantenibilidad) y subcaracterísticas (p. ej. “Analizabilidad”, “Cambiabilidad”, Estabilidad, etc.).

Finalmente, a partir de los indicadores de calidad para las características y subcaracterísticas de calidad, se podrá generar un único indicador (cuadro de mando) de calidad que refleje la calidad del producto bajo un modelo de calidad definido.

C.1.3 RF_003: Gestionar modelos para estructurar la medición de la calidad

El modelo de medición de calidad recoge, principalmente, las fuentes de datos de calidad, la definición de las agrupaciones de los datos para obtener medidas de calidad, las funciones que transforman las mediciones de calidad en indicadores normalizados de calidad según unos umbrales (funciones de normalización), los propios umbrales de calidad y las funciones que agrupan indicadores para obtener los indicadores finales de las características y subcaracterísticas de calidad (funciones de calidad)..

El sistema debe ser capaz de proporcionar mecanismos para la gestión de los modelos de medición de calidad: la gestión de fuentes de datos (p. ej. una herramienta de comprobación de estilo de código), la gestión de medidas de calidad (p. ej. la densidad de la complejidad ciclomática), la gestión de las funciones de normalización de indicadores, la gestión de umbrales y la gestión de las funciones de calidad.

Además, se deberá permitir la existencia de diferentes modelos de medición de calidad, generalmente debidas a la naturaleza heterogénea de las diferentes aplicaciones a medir (diferentes arquitecturas, diferentes lenguajes de programación, etc.)

C.1.4 RF_004: Generar recomendaciones para la mejora del software

A partir de los indicadores de calidad, el sistema generará recomendaciones de cambios para aumentar la calidad del sistema. Estas recomendaciones de cambios estarán especialmente enfocadas a cambios en el código. Una de las tecnologías que potencia la resolución de cuestiones como las anteriores es la refactorización, esto es, las transformaciones a programas preservando su comportamiento, modificando sólo su estructura interna (como cambiar el nombre de una función, su interfaz para hacerla más reutilizable o eliminar código duplicado). Al finalizar los cambios, se podrá observar la mejora en la calidad del

producto, realizando una nueva medición y evaluación de calidad y comparando los resultados.

La búsqueda de recomendaciones tendrá un enfoque basado en valor, en el que dichas recomendaciones se prioricen según el beneficio que aporten al software desde el punto de vista del negocio, como se muestra en la Figura C-1.

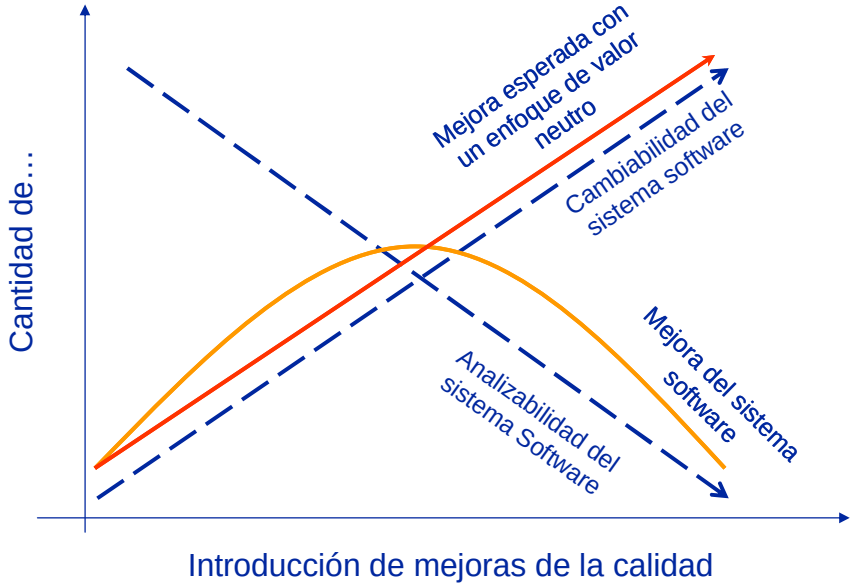


Figura C-1. Ejemplo de la necesidad de guiar por valor la mejora software

C.1.5 RF_005: Generar informes sobre la calidad del software

Tras la medición y la evaluación de calidad, el sistema generará informes acerca de las mediciones, estimaciones y las mejoras recomendadas por el sistema. Los informes se pueden referir a una auditoría de calidad o al histórico de auditorías de calidad realizadas.

Estos informes proporcionan información desde diferentes perspectivas, entre las que se destacan:

- Durante el desarrollo del sistema:
  - Directivos, Jefes de Proyecto y Desarrolladores obtienen información de la calidad del producto, según sus necesidades, a diferente nivel: estratégico, táctico u operacional.

- Durante fase de transición de desarrollo a producción:
    - Clientes: a la hora de adquirir un determinado desarrollo, es necesario tener visibilidad acerca de la calidad del sistema desarrollado.
    - Explotadores: a la hora de poner un sistema en producción, es necesario tener visibilidad acerca de la calidad del producto, para evitar errores o caídas inesperadas del sistema, alto tiempo de corrección de errores, etc. que impacten sobre los usuarios.
- Durante la fase de mantenimiento, evolutivo o correctivo:
    - Evitar cambios que degraden la calidad del sistema: con el tiempo, los sistemas software se degradan debido a los cambios que sufren. Es importante controlar la calidad del sistema, para controlar e incluso prevenir esta degradación de la calidad.

C.2 Casos de Uso

A continuación se describen los casos de uso obtenidos a partir de los requisitos funcionales.

C.2.1 CU_001: Medir la calidad del producto software

Código	CU_001
Nombre del caso de uso	Medir la calidad del producto software.
Descripción	Permite realizar una medición de la calidad software, obteniendo un cuadro de mando de indicadores y métricas de calidad.
Actores	- Herramienta de integración continua, p.ej. Cruise Control - Herramienta de construcción, p.ej. Ant - Usuario desde consola
Precondiciones	- Modelo de medición de calidad definido en el sistema.

Flujo normal	Intenciones del actor 1. Iniciar la ejecución del sistema.	Responsabilidades del sistema 2. Leer/Analizar informes de las fuentes de datos. 3. Obtener indicadores de calidad. 4. Almacenar métricas e indicadores en la base de datos. 5. Generar informe de resultados.
Flujo alternativo		
Poscondiciones	Se genera el informe de resultados de la medición de la calidad del producto software.	
Notas	Modelo de medición de calidad: recoge la definición de las agrupaciones de los datos para obtener medidas de calidad, las funciones que transforman las mediciones de calidad en indicadores normalizados de calidad según unos umbrales, los propios umbrales de calidad y las funciones que agrupan indicadores para obtener los indicadores finales de las características y subcaracterísticas.	

C.2.2 CU_002: Consultar resultados

Código	CU_002
Nombre del caso de uso	Consultar resultados.
Descripción	Permite al usuario consultar, en cualquier momento, los resultados generados en las mediciones realizadas.

Actores	- Usuario (roles: directivos, jefes de proyecto, desarrolladores, clientes, explotadores).	
Precondiciones	- Se ha ejecutado la aplicación al menos una vez satisfactoriamente, por lo que existen informes de resultados generados por la aplicación.	
Flujo normal	Intenciones del actor 1. Ver los resultados obtenidos.	Responsabilidades del sistema 2. Muestra el cuadro de mando que contiene los resultados obtenidos de las mediciones.
Poscondiciones	- Se ha mostrado el cuadro de mando con los resultados obtenidos de las mediciones de calidad del producto software.	
Notas	Cuadro de mando: medio para mostrar claramente los resultados, con un formato simple y conciso, que permita a los usuarios centrarse en las áreas que necesitan atención inmediata y seguir otras mediciones clave del producto software analizado.	

C.2.3 CU_003: Gestionar modelo de medición de la calidad

Código	CU_003
Nombre del caso de uso	Gestionar modelo de medición de la calidad.
Descripción	Permite al usuario gestionar el modelo de medición que el sistema KEMIS utilizará en sus mediciones, adaptándolo a las necesidades concretas de la empresa o producto bajo medición.

Actores	- Usuario (roles: directivos, jefes de proyecto, desarrolladores, clientes).	
Flujo normal	Intenciones del actor 3. Gestionar el modelo de medición de calidad.	Responsabilidades del sistema 4. Modificar el modelo de medición del sistema conforme a las peticiones del usuario.
Poscondiciones	- El modelo de medición de la calidad del sistema ha sido modificado conforme a las peticiones expresadas por el usuario.	
Notas	Modelo de medición de calidad: recoge la definición de las agrupaciones de los datos para obtener medidas de calidad, las funciones que transforman las mediciones de calidad en indicadores normalizados de calidad según unos umbrales, los propios umbrales de calidad y las funciones que agrupan indicadores para obtener los indicadores finales de las características y subcaracterísticas.	

C.2.4 CU_004: Definir las características y el modelo de medición del valor

Código	CU_004
Nombre del caso de uso	Definir las características y el modelo de medición del valor.
Descripción	Permite identificar cuáles serán las características que van a ser utilizadas para el cálculo del valor. Se identificarán las herramientas necesarias para generar las mediciones de las características y las fórmulas necesarias para obtener el valor final de cada componente.
Actores	Desarrollador
Flujo normal	1. Se realiza una identificación de estudios preliminares. 2. Basándose en los estudios, se identifican las características que han de tenerse en cuenta.

	3. Se identifican las herramientas necesarias para obtener los resultados de las diferentes características. 4. Se identifica en qué grado cada característica influirá en el valor. Se obtiene una fórmula de cálculo del valor
Flujo alternativo	- Flujo alternativo 1: Si no existe ninguna herramienta para calcular alguna característica: 3a. Se realiza una herramienta que permita la obtención de los resultados para la característica deseada. Seguir en 4.
Poscondiciones	- Flujo Básico y flujo alternativo 1: - Se obtiene un modelo de medición del valor
Notas	- El cálculo del valor se realizará durante la construcción del sistema con la herramienta Maven.

C.2.5 CU_005: Extraer la estructura del producto y calcular el valor de sus componentes

Código	CU_002
Nombre del caso de uso	Extraer la estructura del producto y calcular el valor de sus componentes.
Descripción	Permite conocer la estructura del producto que está siendo analizado y, a través de una serie de informes de entrada, calcular el valor de cada uno de sus componentes.
Actores	Evaluador, Maven.
Precondiciones	- Generación de los informes necesarios para la realización del cálculo del valor.
Flujo normal	1. El usuario inicia la medición del valor. 2. El sistema genera la estructura del producto. 3. El sistema analiza los informes de entrada y obtiene los resultados para cada uno de los indicadores. 4. El sistema calcula el valor de los diferentes componentes del producto. 5. El sistema genera un informe con los resultados. 6. El informe es persistido en la base de datos del sistema
Flujo alternativo	2.a, 3.a, 4.a y 5.a Si se produce un error durante la ejecución: 1. El sistema muestra el error en las trazas de ejecución del sistema.

Poscondiciones	- Flujo Básico: - Se genera el informe de resultados y se persiste. - Flujo 2.a, 3.a , 4.a y 5.a: - Se asegura la integridad de los datos en el sistema de persistencia. - No se genera el informe de resultados y, por tanto, no es posible persistirlo.
Notas	- El cálculo del valor se realizará durante la construcción del sistema con la herramienta Maven.

C.2.6 CU_006: Visualización del valor de cada componente del proyecto analizado

Código	CU_006
Nombre del caso de uso	Visualización del valor de cada componente del proyecto analizado.
Descripción	Permite visualizar los datos relativos al valor y a las características del mismo para los diferentes componentes del proyecto que ha sido analizado.
Actores	Evaluador
Precondiciones	- Generación del informe del valor del proyecto y almacenamiento del mismo en base de datos.
Flujo normal	1. El usuario accede al portal web generado por KEMIS. 2. El usuario identifica el proyecto del cual quiere visualizar el valor y solicita su visualización. 3. El sistema obtiene los datos relativos al proyecto desde la base de datos. 4. El sistema realiza cálculos en los datos para adaptarlos al visualizador seleccionado. 5. El sistema presenta los datos en el visualizador. 6. El usuario utiliza la visualización para conocer los datos relativos a cada componente.
Flujo alternativo	3.a, 4.a y 5a Si se produce un error durante la ejecución: 1. El sistema muestra una página de error, y los datos no pueden ser consultados.

Poscondiciones	- Flujo Básico: - Se genera la visualización y se presentan los datos al usuario. - Flujo 3.a , 4.a y 5.a: - No se genera la visualización ni se presentan los datos al usuario.
Notas	- Servidor Web (recomendado Tomcat), navegador web (soporte al menos para Internet Explorer, Mozilla Firefox y Google Chrome)

C.2.7 CU_007: Identificar el valor aportado por una mejora

Código	CU_007
Nombre del caso de uso	Identificar el valor que aportará la realización de una mejora.
Descripción	Permite indicar cuál es el valor que puede aportar una mejora en función de los componentes que serían modificados al aplicarla.
Actores	Evaluador.
Precondiciones	- Generación del informe del valor del proyecto y almacenamiento del mismo en base de datos.
Flujo normal	1. El usuario accede al portal web generado por KEMIS. 2. El usuario identifica el proyecto del cual quiere visualizar el valor que aportará una mejora. 3. El sistema obtiene los datos relativos al proyecto desde la base de datos. 4. El usuario selecciona los componentes que van a ser modificados con la mejora. 5. El sistema calcula el valor que aportará la mejora a las clases.
Flujo alternativo	3.a, y 5a Si se produce un error durante la ejecución: 1. El sistema muestra una página de error, y los datos no pueden ser consultados.

Poscondiciones	- Flujo Básico: - Se genera la visualización y se presentan los datos al usuario. - Flujo 3.a , y 5.a: - No se genera la visualización ni se presentan los datos al usuario.
Notas	- Servidor Web (recomendado Tomcat), navegador web (soporte al menos para Internet Explorer, Mozilla Firefox y Google Chrome)

C.2.8 CU_008: Seleccionar las partes del código en las que se desea buscar posibles refactorizaciones

Código	CU_008
Nombre del caso de uso	Seleccionar las partes del código en las que se desea buscar posibles refactorizaciones
Descripción	Permite al usuario seleccionar en qué partes del código (módulos, paquetes o clases) quiere buscar posibles refactorizaciones.
Actores	Desarrollador
Precondiciones	Es necesario que se le ofrezca al usuario el árbol de ficheros y el valor de cada uno de los componentes del sistema. Esta funcionalidad proviene de la fase anterior del proyecto, por lo que solamente es necesario integrarla con ella.
Flujo normal	5. Se ofrece al usuario el árbol de componentes del producto, indicando el valor de cada uno de ellos. 6. El usuario puede ir creando nuevas mejoras, que se van definiendo, indicando el valor de cada una de ellas. Estas mejoras consisten en un conjunto de clases y su valor asociado, constituyendo así una parte del código que el usuario quiere mejorar. 7. Una vez que el usuario ha decidido qué partes del código quiere mejorar, debe pulsar un botón para comenzar a buscar la mejora.
Flujo alternativo	Flujo alternativo 1: Si el usuario no elige ninguna mejora, el sistema no puede pasar al paso 3. En ese caso, no se ofrecerá el

	botón de buscar mejora.
Poscondiciones	- Flujo Básico: - El sistema busca refactorizaciones en el sistema.
Notas	- Servidor Web (recomendado Tomcat), navegador web (soporte al menos para Internet Explorer, Mozilla Firefox y Google Chrome).

C.2.9 CU_009: Identificar candidatos a refactorización en las partes del código seleccionadas

Código	CU_009
Nombre del caso de uso	Identificar candidatos a refactorización en las partes del código seleccionadas
Descripción	Permite, una vez que el usuario ha seleccionado las partes del código que quiere mejorar, buscar partes de ese código seleccionado que son candidatas a la aplicación de una refactorización.
Actores	Maven.
Precondiciones	- Selección por parte del usuario de las partes del código que quiere mejorar.
Flujo normal	7. El usuario inicia búsqueda de mejoras. 8. El sistema analiza el sistema en busca de las refactorizaciones definidas 9. El sistema devuelve como resultado posibles refactorizaciones dentro del sistema. 10. Las posibles refactorizaciones son mostradas en la interfaz Web.
Flujo alternativo y excepciones	- Flujo alternativo 1: el sistema no encuentra ninguna posible refactorización en el sistema (paso 2): 3. El sistema no devuelve ningún resultado. 4. La interfaz Web indica que no se han identificado mejoras 2.a Si se produce un error durante la ejecución: 1. El sistema muestra el error en las trazas de ejecución del sistema.

Poscondiciones	- Flujo Básico: - Se devuelve un conjunto de refactorizaciones, que se muestran en la interfaz Web. - Flujo Alternativo 1 y Excepción 2a: - Se indica por la interfaz Web que no ha podido encontrarse ninguna refactorización.
Notas	- La búsqueda de refactorizaciones se realizará cuando el usuario lo desee a través de la aplicación Web.
Poscondiciones	- Flujo Básico: - Se aplica la refactorización y se muestra al usuario el resultado. - Flujo alternativo 1: - No se presenta la refactorización, al no haber sido aplicada. - Flujo 3.a: - No se ofrece el código nuevo al usuario, al existir un error
Notas	- Servidor Web (recomendado Tomcat), navegador web (soporte al menos para Internet Explorer, Mozilla Firefox y Google Chrome)

C.3 Matriz de Trazabilidad

La siguiente tabla muestra la matriz de trazabilidad entre requisitos funcionales y casos de uso, esto es, indicando qué los requisitos funcionales que satisface cada caso de uso.

	RF_001	**RF_002**	**RF_003**	**RF_004**	**RF_005**
CU_001	X	X			
CU_002					
CU_003			X		
CU_004				X	
CU_005				X	
CU_006				X	
CU_007				X	
CU_008					X
CU_009					X

Tabla C-1. Matriz de trazabilidad Requisitos – Casos de Uso

Acrónimos

Acrónimo	Significado
BRA	Benefits Realization Approach
CBO	Coupling Between Object classes
CMMI-DEV	Capability Maturity Model Integration for Development
COF	Coupling Factor Metric
CVS	Ciclo de Vida Software
ISBV	Ingeniería de Software Basada en Valor
ISO	International Organization for Standardization
LCOM	Lack of COhesion Metric
ROI	Return On Investment
SWEBOK	Software Engineering Body of Knowledge
TIR	Tasa Interna de Retorno
VAN	Valor Actual Neto
VBSE	Value-Based Software Engineering
XP	eXtreme Programming
Atributos Base	
NCSS: Non Commented Source Statements COM: number of Comment Lines CC: Cyclomatic Complexity DPL: Duplicate Lines SGL: Significant Lines UT: number of Unit Tests UTE: number of Error in Unit Tests UTLC: Unit Test Line Coverage UTBC: Unit Test Branch Coverage CYC: number of Cycles. PACK: number of Packages	

PV: number of PMD Violations
Atributos Derivados
CCD: Cyclomatic Complexity Density DCD: Duplicated Code Density UTD: Unit Tests Density UTED: Unit Tests Error Density UTCR: Unit Tests Coverage Ratio CYCD: Cycles Density PVD: PMD Violations Density COMD: Comment Lines Density ANALDD: Analyzability Defects Density CHGDD: Changeability Defects Density UTDD: Unit Test Defects Density MOSTDD: Modification Stability Defects Density
Atributos Derivados Normalizados
Q_CCD: Normalized Quality Value of Cyclomatic Complexity Density Q_DCD: Normalized Quality Value of Duplicated Code Density Q_UTD: Normalized Quality Value of Unit Tests Density Q_UTED: Normalized Quality Value of Unit Tests Error Density Q_UTCR: Normalized Quality Value of Unit Tests Coverage Ratio Q_CYCD: Normalized Quality Value of Cycles Density Q_PVD: Normalized Quality Value of PMD Violations Density Q_COMD: Normalized Quality Value of Comment Lines Density ANALDD: Normalized Quality Value of Analyzability Defects Density CHGDD: Normalized Quality Value of Changeability Defects Density UTDD: Normalized Quality Value of Unit Test Defects Density MOSTDD: Normalized Quality Value of Modification Stability Defects Density

Referencias

[1] Rifkin, S. (2009) Guest Editor's Introduction: Software Measurement. IEEE Software, 26, p. 70.

[2] Piattini, M., et al. (2008) Medición y estimación del software: técnicas y métodos para mejorar la calidad y productividad del software, Ra-ma.

[3] Ebert C. (2009) Guest Editor's Introduction: How Open Source Tools Can Benefit Industry. IEEE Software, 26, p: 50-51.

[4] SEI, Process Maturity Profile (2001) CMMI v1.2, SCAMPI v1.2, Class A Appraisal Results. 2011 Mid-Year Update, Septiembre 2011, Software Engineering Institute.

[5] ISO, ISO/IEC 12207:2008. Systems and software engineering - Software life cycle processes, in International Organization for Standarization.

[6] Kitchenham, B. and S.L. Pfleeger (1996) Software Quality: The Elusive Target. IEEE Software, 20(1), p: 12-21.

[7] Maibaum T, W.A. (2008) A Product-Focused Approach to Software Certification.

[8] ISO, Software Product Evaluation–Quality Characteristics and Guidelines for their Use. ISO/IEC Standard 9126, International Organization for Standarization.

[9] ISO, ISO/IEC 25000 Software and system engineering – Software product Quality Requirements and Evaluation (SQuaRE) –Guide to SQuaRE, in International Organization for Standarization. 2005: Ginebra, Suiza.

[10] ISO, ISO/IEC 25010 Software and system engineering – Software product Quality Requirements and Evaluation (SQuaRE) – Quality model and guide. International Organization for Standarization. . 2005: Ginebra, Suiza.

[11] Frazer, A. (1992) Reverse engineering- hype, hope or here? Software Reuse and Reverse Engineering in Practice, p: 209-243.

[12] Pressman, R., (2002) Software Engineering: A Practitioner's Approach., Madrid: McGraw-Hill.

[13] Saiedian, H. and N. Carr (1997) Characterizing a software process maturity model for small organizations. , in ACM SIGICE Bulletin, p. 2-11.

[14] Janice, S. (1998) Practices of Software Maintenance, in Proceedings of the International Conference on Software Maintenance, IEEE Computer Society.

[15] Harrison, W. (2005) What Do Software Developers Need to Know about Business? IEEE Software, 22(5), p: 5-7.

[16] Boehm, B. (2005) Value-Based Software Engineering: Overview and Agenda in Value-Based Software Engineering S. Biffl, et al., Editors, Springer, p: 3-14.

[17] Erlikh L. (2000) Leveraging Legacy System Dollars for E-business. IT Professional, 2, 3, pp: 17-23.

[18] Daniel Cabrero Moreno (2009) Construcción y Evolución del Software Basados en Valor, in Departamento de Tecnologías y Sistemas de la Información, p: 262.

[19] Opdyke, W. (1992) Refactoring Object Oriented Frameworks, in Computer Science, Illinois: Urbana-Champain.

[20] Mens, T. and T. Tourwé (2004) A Survey of Software Refactoring. IEEE transactions on software engineering, 30(2), p: 126-139.

[21] Tokuda, L. and D. Batory (2008) Evolving Object-Oriented Designs with Refactorings. Automated Software Engg., 8(1), p: 89-120.

[22] Fowler, M., et al. (2000) Refactoring: Improving the Design of Existing Code. 1st edition, Addison-Wesley Professional.

[23] IEEE, IEEE Standard 610.12–1990 IEEE Standard Glossary of Software Engineering Terminology. 1990, Institute of Electrical and Electronics Engineers.

[24] Stroulia E. et al. (2003) Understanding the Economics of Refactoring, in 5th International Workshop on Economics-Driven Software Engineering Research (EDSER-5): The Search for Value in Engineering Decisions, Portland, OR, USA. p: 44-49.

[25] Wiederhold, G. (2006) What is your software worth? Communications of the ACM, 49(9), p: 65-75.

[26] Briand L., Morasca S., Basili V. 1996. Property-Based Software Engineering Measurement. IEEE Transactions on Software Engineering 22(1):68-86.

[27] Sangwan R.S., Vercellone-Smith P., Laplante P.A., (2008) Structural Epochs in the Complexity of Software over Time. IEEE Computer Society.

[28] Lavazza L. (2000) Providing Automated Support for the GQM Measurement Process. IEEE Software, 17, 3 ,pp:56-62.

[29] Bourque, P., Dupuis, R., Abran, A., Moore, J. W. & Tripp, L. L. (1999) The Guide to the Software Engineering Body of Knowledge. IEEE Software, 16, p: 35-44.

[30] Marcos, E. & Marcos, A. (1998). An Aristotelian Approach to the Methodological Research: a Method for Data Models Construction. In: Information Systems - The Next Generation. L. Brooks and C. Kimble (Eds.). McGraw-Hill.

[31] Bunge, M. (1979). La Investigación Científica. Barcelona: Ariel.

[32] Kitchenham, B. Procedures for Performing Systematics Reviews (2004) Keele University Technical Report TR/SE-0401. ISSN:1353-7776. NICTA Technical Report 0400011T.1.

[33] Biolchini, J., Gomes Mian, P., Cruz Natali, A.C., Horta Travasso, G. (2005) Systematic Review in Software Engineering. Technical Report ES 679/05.

[34] Lewin, K. (1947) Frontiers in Group Dynamics. Human Relations., 1, pp: 5-41.

[35] Wood-Harper, T. (1985) Research Methods in Information Systems: Using Action Research. En Mumford et al. (eds.). Research Methods in Information Systems. Amsterdam:North-Holland., pp: 169-191.

[36] Avison, D., Lan, F., Myers, M. & Nielsen, A. (1999) Action Research. Communications of the ACM., 42, pp: 94-97.

[37] Seaman, C. B. (1999) Qualitative Methods in Empirical Studies of Software Engineering. IEEE Transactions on Software Engineering., 25, pp: 557-572.

[38] McTaggart, R. (1991) Principles of Participatory Action Research. Adult Education Quarterly, pp: 41.

[39] Kock, N. & Lau, F. (2001) Information Systems Action Research: Serving Two Demanding Masters. Information Technology & People (special issue on Action Research in Information Systems), 14, pp: 6-11.

[40] Wadsworth, Y. 1998, What is participatory Action Research? Action Research Internation, paper 2, online.

[41] Parnas, D. L. (2001) Some software engineering principles, Structured Analysis and Design, pp.:237-247.

[42] Lehman, M. M. (1996) Laws of Software Evolution Revisited, in Proceedings of the 5th European Workshop on Software Process Technology EWSPT '96, Springer-Verlag, pp: 108-124.

[43] Garvin, D. (1984) What does product quality really mean?, Sloan Management Review, 26, pp. 25-45.

[44] Robson, C. (2002) Real world research: a resource for social scientists and practitioner-researchers, Blackwell Published Ltd.

[45] Bosch, J. (2000) Design & Use of Software Architectures, Addison-Wesley.

[46] Pfleeger, S. L. (1998) Software Engineering - Theory and Practise, Prentice-Hall.

[47] McCall, J.A., Richards, P.K., Walters, G.F.,(1977) Factors in Software Quality, RADC TR-77-369.

[48] Boehm, B.W, Brown, J.R., Kaspar, J.R., et.al. (1978) Characteristics of Software Quality, TRW Series of Software Technology, Amsterdam, North Holland.

[49] Bowen, T. P., Wigle, G. B., Tsai, J. T. (1985) Specification of software quality attributes. Tech. Rep. RADC-TR-85-37, Rome Air Development Center.

[50] Grady, R. B. (1992) Practical software metrics for project management and process improvement,Prentice Hall.

[51] Dromey, R. G. (1995) "A model for software product quality", IEEE Transactions on Software Engineering, no. 2, pp: 146-163.

[52] Bhatt P., Williams K., Shroff G., and Misra A. K. (2006) “Influencing Factors in Outsources Software Maintenance”, ACM SIGSOFT Software Engineering Notes, 31, 3, pp: 1–6.

[53] Bøegh, J., (2008) A New Standard for Quality Requirements, IEEE Software, vol. 25, no. 2, pp. 57-63.

[54] Chidamber S.R. and Kemerer C.F. (1994) A metrics suite for object-oriented design, IEEE Transactions on Software Engineering, 20, 6, pp: 476-493.

[55] Kitchenham B.A., Hughes R.T. and Kinkman S.G. (2001) Modeling Software Measurement Data, IEEE Transactions on Software Engineering, 27, 9, pp: 788-804.

[56] Fenton N.E.and Kaposi A.A. (1987) Metrics and software structure, Information and Software Technology, 29, pp:301-320.

[57] Poels G. and Dedene G. (2000) Distance-based software measurement: necessary and sufficient properties for software measures, Information and Software Technology, 42, pp: 35-46.

[58] Weyuker E.J. (1988) Evaluating Software Complexity Measures, IEEE Transactions on Software Engineering, 14, 9, pp:1357-1365.

[59] Reformat M., Pedrycz W. and Pizzi N.J. (2003) Software quality analysis with the use of computational intelligence, Information and Software Technology, 45, pp:405-417.

[60] Lincke, R., Lundberg, J. & Löwe, W. (2008) Comparing software metrics tools, in Proceedings of the 2008 international symposium on Software testing and analysis, ACM, pp: 131.

[61] Heitlager, I., Kuipers, T. & Visser, J. (2007) A Practical Model for Measuring Maintainability, in Proceeding of Quality of Information and Communications Technology, 2007 (QUATIC 2007), IEEE, pp. 30-39.

[62] Mouchawrab, S., Briand, L.C. & Labiche, Y. (2005) A measurement framework for object-oriented software testability, Information and Software Technology, 47, 15, pp: 979-997.

[63] Samoladas, I., Gousios, G., Spinellis, D. & Stamelos, I. 2008, "The SQO-OSS quality model: Measurement based open source software evaluation", Open Source Development, Communities and Quality, Vol. 275, pp. 237-248.

[64] Coleman, D. and Ash, D. and Lowther, B. and Oman, P. (1994) Using metrics to evaluate software system maintainability, IEEE Computer, pp: 44-49.

[65] Zhou, Y. and Xu, B. (2008) Predicting the maintainability of open source software using design metrics, Wuhan University Journal of Natural Sciences, 13, 1, pp:14-20.

[66] Lochmann K. and Heinemann L. (2011) Integrating quality models and static analysis for comprehensive quality assessment, in Proceeding of the 2nd international workshop on Emerging trends in software metrics (WETSoM '11). ACM, pp.:5-11.

[67] ISO (2007) ISO 15939 Software and system engineering—Measurement Process, International Organization for Standarization

[68] ISO (2007) ISO 25020 Software and system engineering--Software product Quality Requirements and Evaluation (SQuaRE)--Measurement reference model and guide, International Organization for Standarization

[69] Zelkowitz M.V. (1978) “Perspectives in Software Engineering”, ACM Comput Surv (CSUR), 10,pp: 197–216.

[70] Chikofsky, E. (2007) Engineering Management & Integration. 1st IEEE Computer Society Conference on Exploring Quantifiable Information Technology Yields. Amsterdam, IEEE Computer Society.

[71] Boehm, B. (2003) Value-Based Software Engineering. ACM SIGSOFT Software Engineering Notes, 28.

[72] Thorp, J. (1998) The Information Paradox: Realizing the Business Benefits of Information Technology, Toronto, Mcgraw-Hill Publisher Ltd., 2002.

[73] Boehm, B. & Huang, L. G. (2003) Value-Based Software Engineering : A Case Study. IEEE Computer Society, 36, pp: 33-41.

[74] Thelin, T., Runeson, P. & Wholin, C. (2003) Prioritized Use Cases as a Vehicle for Software Inspections. IEEE Software, 20, 30-33.

[75] Cleland-Huang, J., Zemont, G. & Lukasik, W. (2004) A Heterogeneous Solution for Improving the Return on Investment of Requirements Traceability. Requirement Engineering Conference, 12th IEEE International (RE'04). IEEE Computer Society.

[76] Boehm, B. (2005a) Value-based quality processes and results. International Conference on Software Engineering. Proceedings of the third workshop on Software quality. St. Louis, Missouri.

[77] Egyed, A., Biffl, S., Heindl, M. & Grünbacher, P. (2005) A value-based approach for understanding cost-benefit trade-offs during automated software traceability. 3rd international workshop on Traceability in emerging forms of software engineering Long Beach, California ACM Press.

[78] Heindl, M. & Biffl, S. (2005) A Case Study on Value-based Requirements Tracing. 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering Lisbon, Portugal ACM Press.

[79] Lee, K. & Boehm, B. (2005) Empirical results from an experiment on value-based review (VBR) processes. International Symposium on Empirical Software Engineering (ISESE). Noosa Heads, Australia.

[80] Huang, L. & Boehm, B. (2006) How Much Software Quality Investment Is Enough: A Value-Based Approach. IEEE Software, 23, 88- 95.

[81] Srikanth, H. & Williams, L. (2005) On the economics of requirements-based test case prioritization. 7th international workshop on Economics-driven software engineering research St. Louis, Missouri ACM Press

[82] Kazman, R., Asundi, J. & Klein, M. (2001) Quantifying the Costs and Benefits of Architectural Decisions. Proceedings of the 23rd International Conference on Software Engineering. Toronto, Ontario, Canada, IEEE Computer Society.

[83] Kazman, R., Asundi, J. & Klein, M. (2002) CBAM: Making Architectural Decisions. An Economic Approach (Technical Report CMU/SEI-2002-TR-035). Software Engineering Institute.

[84] Kim, J. & Kang, S. (2008) Architecture decision based on value-based software engineering concepts. India Software Engineering Conference. Hyderabad, India.

[85] Little, T. (2005) Context-Adaptive Agility: Managing Complexity and Uncertainty. IEEE Software, 22, 28-35.

[86] Pigosky, T. M. (1997) Practical Software Maintenance, New York (EEUU), John Wiley & Sons.

[87] Beck K. (1999) Extreme Programming Explained: Embrace Change, Addison-Wesley Professional.

[88] Beck K, Fowler M. (1999) Bad Smells in Code. Refactoring: Improving the Design of Existing Code: Addison Wesley.

[89] Kitchenham B.A., Dybå T., Jorgensen M. (2004) Evidence-based Software Engineering, in Proceedings of the 26th International Conference on Software Engineering, IEEE, pp:273-281.

[90] Kitchenham B. and Charters S. (2007) Guidelines for performing Systematic Literature Reviews in Software Engineering, Keele University and Durham University Joint Report, Technical Report.

[91] Kitchenham B. (2004) Procedures for Perfoming Systematic Review. Australia: Joint Technical Report, Software Engineering Group, Department of Computer Scinece Keele University, United Kingdom and Empirical Software Engineering, National ICT Australia Ltd.

[92] Riaz M, Mendes E, Tempero E. 2009. A systematic review of software maintainability prediction and metrics. Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement: IEEE Computer Society.

[93] Dybå T, Dingsøyr T. (2008). Empirical studies of agile software development: A systematic review. Inf Softw Technol, 50,9-10, pp:833-859.

[94] Usha, K., Poonguzhali, N. and Kavitha, E.A (2009) A quantitative approach for evaluating the effectiveness of refactoring in software development process, in Proceeding of International Conference on Methods and Models in Computer Science, ICM2CS, IEEE, pp: 1-7.

[95] Simon, F., Steinbruckner, F. and Lewerentz, C.(2001) Metrics based refactoring, in Proceedings of the Fifth European Conference on Software Maintenance and Reengineering, IEEE Computer Society, pp:30-38.

[96] Mäntylä M.V., Lassenius C. (2006) Drivers for software refactoring decisions, in Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering, ACM, pp: 297-306.

[97] Bouktif, S., Antoniol, G., Merlo, E. and Neteler, M. (2006) A novel approach to optimize clone refactoring activity, in Proceeding 8th annual conference on Genetic and evolutionary computation, ACM, pp: 1885-1892.

[98] Chahal, K.K. and Singh, H. (2009) Metrics to study symptoms of bad software designs, SIGSOFT Softw Eng Notes, 34, 1, pp: 1-4.

[99] Tilevich, E. and Back, G. (2008) 'Program, enhance thyself!': demand-driven pattern-oriented program enhancement, in Proceeding 7th international conference on Aspect-oriented software development, ACM, pp: 13-24.

[100] Kim, S. and Ernst, M.D. (2007) Which warnings should I fix first?, in Proceeding. 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, ACM, pp: 45-54.

[101] Liu, H., Li, G., Ma, Z. and Shao, W. (2007) Scheduling of conflicting refactorings to promote quality improvement, in Proceeding twenty-second IEEE/ACM international conference on Automated software engineering, ACM, pp: 489-492.

[102] Murphy-Hill, E. and Black, A. (2008) Breaking the barriers to successful refactoring: observations and tools for extract method, in Proceeding 30th international conference on Software engineering, ACM, pp: 421-430.

[103] Murphy-Hill, E. and Parnin, C. and Black, A.P. (2009) How we refactor, and how we know it, Proceeding 31st International Conference on Software Engineering (ICSE' 09), IEEE, pp: 287-297.

[104] Mens, T., Taentzer, G., Runge, O. (2007) Analysing refactoring dependencies using graph transformation"; Software and Systems Modeling, 6, 3, pp: 269-285.

[105] Moser, R., Pedrycz, W., Sillitti, A. and Succi, G. (2008) A Model to Identify Refactoring Effort during Maintenance by Mining Source Code Repositories, in Proceeding. 9th International Conference on Product-Focused Software Process Improvement (PROFES'08), pp: 360-370.

[106] Higo, Y. and Matsumoto, Y. and Kusumoto, S. and Inoue, K. (2008) Refactoring Effect Estimation Based on Complexity Metrics, in Proceeding Australian Software Engineering Conference, IEEE, pp: 219-228.

[107] Carr, M. and Wagner, C. (2002) A Study of Reasoning Processes in Software Maintenance Management, Inf.Technol.and Management, Kluwer Academic Publishers, 3, 1-2, pp: 181-203.

[108] Stroggylos, K. and Spinellis, D. (2007) Refactoring--Does It Improve Software Quality?. In Proceeding 5th International Workshop on Software Quality, IEEE Computer Society, 10.

[109] Bahsoon, R. and Emmerich, W. (2007) Economics-Driven Software Mining; in Proceeding First International Workshop on The Economics of Software and Computation, IEEE Computer Society, 3.

[110] Leitch, R. and Stroulia, E. (2002) Assessing the Maintainability Benefits of Design Restructuring Using Dependency Analysis, in Proceeding IEEE International Symposium on Software Metrics (METRICS'02), pp: 309-322.

[111] Kataoka Y, Imai T, Andou H and Fukaya T. (2002) A Quantitative Evaluation of Maintainability Enhancement by Refactoring, in Proceeding. International Conference on Software Maintenance (ICSM'02), IEEE Computer Society,pp: 576-585.

[112] Huston, B. (2001) The effects of design pattern application on metric scores"; Journal of Systems and Software, 58, 3, pp: 261-269.

[113] Bahsoon, R. and Emmerich, W. (2004) Evaluating architectural stability with real options theory, in Proceeding 20th IEEE International Conference on Software Maintenance (ICSM'04), Citeseer, pp: 443-447.

[114] O'Keeffe, M., Cinnéide, M.Ó. (2008) Search-based refactoring: an empirical study Journal of Software Maintenance and Evolution: Research and Practice, John Wiley \& Sons, Inc., 20, 5, pp: 345-364.

[115] Bahsoon, R. and Emmerich, W. (2004) Applying ArchOptions to value the payoff of refactoring, in Proceeding Sixth International Workshop on The Economics-Driven Software Engineering Research, Citeseer, pp: 66-70.

[116] O'Keeffe M. Search-based refactoring for software maintenance. J.Syst.Software, 81, 4, pp: 502-516.

[117] Alshayeb M. (2009) Empirical investigation of refactoring effect on software quality. Information and Software Technology, 9;51, 9, pp:1319-1326.

[118] Engel A, Browning TR. (2008) Designing systems for adaptability by means of architecture options. Systems Engineering, 11, 2, pp: 125-146.

[119] Reddy, K.N., Rao, A.A. (2009) A Quantitative Evaluation of Software Quality Enhancement by Refactoring Using Dependency Oriented Complexity Metrics, in Proceeding. Second International Conference on Emerging Trends in Engineering & Technology, IEEE Computer Society, pp: 1011-1018.

[120] Kosker, Y. and Turhan, B. and Bener, A. (2009) An expert system for determining candidate software classes for refactoring, Expert Syst Appl, Elsevier, 36, 6, pp: 10000-10003.

[121] Schofield C, Tansey B, Xing Z, Stroulia E. (2006) Digging the development dust for refactorings"; Proc. 14th IEEE International Conference on Program Comprehension (ICPC'06), IEEE Computer Society, pp: 23-34.

[122] Bahsoon, R. and Emmerich, W. (2004) Applying ArchOptions to value the payoff of refactoring. In: Proceedings of the 6th International Workshop on Economics Driven Software Engineering Research, pp: 66-70.

[123] Engel A., Browning T.R. (2008) Designing systems for adaptability by means of architecture options. Systems Engineering, 11, 2, pp:125-146.

[124] Crombie, I.K. (1996) The Pocket Guide To Critical Appraisal, BMJ Books ISBN: 978-0727910998.

[125] Fink, A. (2005) Conducting Research Literature Reviews: From the Internet to Paper, Sage Publications Inc., ISBN: 978-0761909057.

[126] Brereton P., Kitchenham B.A., Budgen D., Turner M., Khalil M. (2007) Lessons from applying the systematic literature review process within the software engineering domain. Journal of Systems and Software, 80(4), pp:571-583.

[127] Boehm B, Valerdi R, Honour E. The ROI of systems engineering: Some quantitative results for software-intensive systems. Syst.Eng. 2008;11(3):221-234.

[128] Bansiya J, Davis C. A Hierarchical Model for Object-Oriented Design Quality Assessment. IEEE Transactions on Software Engineering 2002;28(1):4-17.

[129] Polo M., Piattini M., Ruiz F. (2001) Using Code Metrics to Predict Maintenance of Legacy Programs: A Case Study. Proceedings of the IEEE International Conference on Software Maintenance (ICSM'01): IEEE Computer Society, pp: 202.

[130] Calliss F.W. (1988) Problems with automatic restructures, SIGPLAN Not.,23(3),pp:13-21.

[131] Du Bois B., Demeyer S. and Verelst J. (2004) Refactoring-improving coupling and cohesion of existing code, in Proceedings of the 11th Working Conference on Reverse Engineering, IEEE, pp: 141-151.

[132] Karlsson J, Ryan K. (1997) A Cost-Value Approach for Prioritizing Requirements. IEEE Softw., 14(5), pp: 67-74.

[133] Ho-Won J. (1998) Optimizing Value and Cost in Requirements Analysis. IEEE Softw., 15(4), pp: 74-78.

[134] Sivzattian, S. and Nuseibeh, B. (2001) Calibrating value estimates of requirements, Third international workshop on economicsdriven software engineering research. Co-located with the international conference on software engineering (ICSE), Citeseer.

[135] Azar J., Smith R.K. and Cordes D. (2007) Value-Oriented Requirements Prioritization in a Small Development Organization, IEEE Softw., 24(1), pp:32-37.

[136] Azar J, Smith R, Cordes D. (2007) Value-Oriented Prioritization: A Framework for Providing Visibility and Decision Support in the Requirements Engineering Process. Department of computer science.University of Alabama, Alabama.

[137] Raffo D., Harrison W.,Settle J. and Eickelmann N. (2000) Understanding the role of defect potential in assessing the economic value of process improvements, in Proceedings of the Second Workshop on Economics-Driven Software Engineering Research, Citeseer.

[138] Biffl, S. (2001) Hierarchical Economic Planning of the Inspection Process, in Proceedings of the Third International Workshop on Economics-Driven Software Engineering Research (EDSER-3), IEEE Computer Society Press, Citeseer.

[139] Müller, M.M. and Padberg, F. (2003) About the return on investment of test-driven development, in Proceedings of the 5th International Workshop on Economic-Driven Software Engineering Research (EDSER-5), Citeseer.

[140] Butler, S.A. and Shaw, M. (2002) Incorporating nontechnical attributes in multi-attribute analysis for security, in Proceedings of the 4th Workshop on Economics-Driven Software Engineering Research (EDSER-4), Citeseer.

[141] Tijdink, T. and Nieuwland, E. (2007) Measuring the business value of availability, in Proceedings of the 2007 IEEE International Conference on Exploring Quantifiable IT Yields, IEEE Computer Society, pp: 101-112.

[142] Mentezelou, P. and Kyriakidou, T. (2007) A Model for Measuring the Relation _Information-Value_ in Companies, in Proceedings of the 2007 IEEE International Conference on Exploring Quantifiable IT Yields, IEEE Computer Society, pp: 55-62.

[143] Cabrero D., Garzas J. and Piattini M. (2007) Choosing the best design strategy from requirement: A value-based approach, in Proceedings of the 2007 IEEE International Conference on Exploring Quantifiable IT Yields, IEEE Computer Society, pp: 87-94.

[144] Gao, J. and Zhu, J. (2009) Prioritized Test Generation Strategy for Pair-Wise Testing, in Proceedings of the 15th IEEE Pacific Rim International Symposium on Dependable Computing, IEEE, pp: 99-102.

[145] Giles A.E. and Daich G.T. (1995) Universal MetricsTools. CrossTalk(February).

[146] XRadar (2009) What is XRadar ?, http://xradar.sourceforge.net/

[147] Sonar, (2001) Features, http://www.sonarsource.org/features/

[148] Letouzey, J.L. and Coq, T. (2010) The SQALE Analysis Model: An Analysis Model Compliant with the Representation Condition for Assessing the Quality of Software Source Code, in Proceedings of the Second International Conference on Advances in System Testing and Validation Lifecycle (VALID), pp:43-48.

[149] Dashboard (2008) About Maven Dashboard Report Plugin, mojo.codehaus.org/ dashboard-maven-plugin/

[150] Kingsbury J, Dawood M. (1995) Metrics Tools: Quality - Defect Tracking. CrossTalk: The Journal of Defense Software Engineering (Mayo).

[151] Erickson D, Steadman A. (1995) Metrics Tools: Effort and Schedule. CrossTalk: The Journal of Defense Software Engineering (Marzo).

[152] Gall H, Riva C, Jazayeri M. (1999) Visualizing Software Release Histories: The Use of Color and Third Dimension, in Proceedings of Software Maintenance (ICSM'99), pp:99-108.

[153] Stasko J.T., Domingue J., Brown M.H. and Price B.A. (1998) Software Visualization—Programming as a Multimedia Experience, The MIT Press.

[154] Petre M. (1995) Why Looking Isn't Always Seeing: Readership Skills and Graphical Programming, Communications of the ACM, ACM, 38, 6, pp:33-44.

[155] Lanza M. and Marinescu R. (2006) Object-Oriented Metrics in Practice. Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems, Springer-Verlag New York Inc.

[156] Chuah M., Eick S.G.. (1998) Information Rich Glyphs for Software Management Data, IEEE Computer Graphics and Applications, 18(4), pp: 24-29.

[157] Chen C, Härdle W, Unwin A. (2008) Handbook of data visualization, Springer.

[158] Pinzger, M., Gall, H., Fischer, M. and Lanza, M. (2005) Visualizing multiple evolution metrics, in Proceedings of the 2005 ACM symposium on Software visualization, ACM, pp: 67-75.

[159] Von Wangenheim, C.G., Hauck, J. and Von Wangenheim, A., (2009) Enhancing Open Source Software in Alignment with CMMI-DEV, IEEE Software, 26(2), pp. 59-67.

[160] Feitelson, D.G., Heller, G.Z. and Schach, S.R. (2006) An Empirically-Based Criterion for Determining the Success of an Open-Source Project, in Proceedings of the Australian Software Engineering Conference (ASWEC 2006), pp.: 6.

[161] Scott Mitchell (2008) Toolbox : Static Analysis Tools For .NET, Matt Berseth's Blog. Microsoft Corporation.

[162] Lee C. C. (2008) JavaNCSS - A Source Measurement Suite for Java, http://javancss.codehaus.org/

[163] InfoEther (2011) PMD, http://pmd.sourceforge.net

[164] Checkstyle (2011) Checkstyle, http://checkstyle.sourceforge.net/

[165] University of Maryland (2011) FindBugs™ - Find Bugs in Java Programs, http://findbugs.sourceforge.net/

[166] Clarkware Consulting (2009) JDepend, http://clarkware.com/software/JDepend.html.

[167] Littlefair T. (2010) CCCC - C and C++ Code Counter, http://cccc.sourceforge.net/

[168] JUnit.org (2011) JUnit, http://www.junit.org/

[169] Lepilleur B. (2011) CppUnit - C++ port of JUnit, http://sourceforge.net/projects/cppunit/

[170] Nunit (2007) Nunit, http://www.nunit.org/

[171] Roubtsov V. (2005) EMMA: a free Java code coverage tool, http://emma.sourceforge.net/

[172] Li, W. and Henry, S. (1993) Object-oriented metrics that predict maintainability, Journal of Systems and Software. 23, 2: p: 111-122.

[173] Heitlager, I., T. Kuipers, and J. Visser (2007) A Practical Model for Measuring Maintainability, in Proceeding of the Quality of Information and Communications Technology (QUATIC), pp: 13-15.

[174] Spinellis, D. (2008) Evaluating the Quality of Open Source Software, in Proceedings of SQM 2008: Second International Workshop on Software Quality and Maintainability—12th European Conference on Software Maintenance and Reengineering (CSMR 2008) satellite event, pp: 5–28.

[175] Zhu, H., Hall, P.V. and May, J.H.R. (1997) Software unit test coverage and adequacy, ACM Computer Surveys, 29(4): pp: 366-427.

[176] Ajrnal Chaumun, M., et al. (2000) Design properties and object-oriented software changeability, in Proceedings of the Fourth European Conference in Software Maintenance and Reengineering.

[177] David Anderson, Métodos cuantitativos para los negocios. 2 ed. 2004: Paraninfo. 834.

[178] Thomas, Z., et al., Mining Version Histories to Guide Software Changes, in Proceedings of the 26th International Conference on Software Engineering. 2004, IEEE Computer Society.

[179] Tudor Gîrba, Yesterday's Weather: Guiding Early Reverse Engineering Efforts by Summarizing the Evolution of Changes. Software Maintenance, IEEE International Conference on, 2004.

[180] Annie, T.T.Y., Raymond N., and Mark C. (2004) Predicting Source Code Changes by Mining Change History. IEEE Trans. Softw. Eng., 2004. 30(9): p. 574-586.

[181] Garzás, J., Consideraciones económicas de la fabricación software, X.C.N.U.d. Oracle, Editor. 2007: Bilbao, España.

[182] StatSCM (2011), StatSCM Introduction, http://stat-scm.sourceforge.net/

[183] Fowler, M. (2000) Continuous Integration,

http://martinfowler.com/articles/continuousIntegration.html

[184] Zhu, H., Hall, P.A.V. and May, J.H.R. (1997) Software unit test coverage and adequacy. ACM Computing Surveys (CSUR), 29(4): p. 366-427.

[185] Google (2008) Introducing the Google Visualization API http://googlecode.blogspot.com/2008/03/introducing-google-visualization-api.html

[186] Glibert, D. (2009) Welcome To JFreeChart!, http://www.jfree.org/jfreechart/

[187] Free Software Foundation (2007) GNU Lesser General Public License, http://www.gnu.org/licenses/lgpl.html

[188] García N. (2011) JavaScript InfoVis Toolkit, http://thejit.org/docs/

[189] Bostock D., (2001) , Data-Drive Documents, http://mbostock.github.com/d3/

[190] Tse, A. (2010) PlotKit, JavaScript Chart Plotting, http://www.liquidx.net/plotkit/

[191] Google (2011) Sitio del desarrollador de iGoogle (Labs), http://code.google.com/intl/es/apis/igoogle/

[192] Krutchen, P.B. (1995) The 4+ 1 view model of architecture, IEEE Software Journal, Vol. 12, No. 6, pp:42-50

[193] OMG (2007) UML 2.1.1 specification, http://www.omg.org/spec/UML/2.1.1

[194] Red Hat (2011) Relational Persistence for Java and .NET, http://www.hibernate.org

[195] Reenskaug T. (1979) Thing-model-view-editor an Example from a planning system, Technical note, Xerox PARC.

[196] The Apache Software Foundation (2011) Struts2, http://struts.apache.org/2.x/

[197] JSON (2011), Introducing JSON, http://json.org/

[198] Oracle (2011), JavaServer Pages Technology,

[199] http://www.oracle.com/technetwork/java/javaee/jsp/index.html

[200] The Apache Software Foundation (2011) Apache Tiles, http://tiles.apache.org/

[201] Oracle, (2011), MySQL, http://www.mysql.com/