La refactorización de software basada en valor: Revisión sistemática de la literatura

Emanuel Irrazábal[1] , Cristina Greiner[1], Gladys Dapozo[1]

[1] Departamento de Informática. FaCENA.
Universidad Nacional del Nordeste
Av. Libertad 3400, Ciudad de Corrientes, Corrientes
{eirrazabal,cgreiner,gndapozo}@exa.unne.edu.ar

Resumen. El mantenimiento del producto software es una de las fases que más costos tiene a lo largo de la vida de una aplicación software. La refactorización del código fuente es una de las técnicas utilizadas para intentar mejorar la mantenibilidad. Actualmente la refactorización se basa en un enfoque de "valor neutro", siendo difícil priorizar la gran cantidad de opciones de refactorización que puede tener un producto software. Para cubrir estas carencias, una nueva rama de la ingeniería está emergiendo, la Ingeniería del Software Basada en Valor, estableciendo que las funcionalidades de un sistema tienen diferente grado de importancia, y que algunas de ellas aportan más "valor" que otras. Este artículo presenta una revisión sistemática de la literatura sobre la refactorización basada en valor. Se identificaron 15 artículos primarios de un total de 2300. Los trabajos analizados muestran el uso de diferentes técnicas para abordar el problema de la refactorización y el valor que pueden aportar al software que será refactorizado, existiendo opiniones opuestas sobre cómo afecta las refactorizaciones a la mantenibilidad. Como conclusión de la revisión se tiene que: ningún artículo habla directamente del valor ni se cuantifica el valor que puede aportar una refactorización al software sobre el que va a ser aplicada. Finalmente, la mayoría de los estudios sugieren que las refactorizaciones semiautomáticas o asistidas son las más adecuadas para llevar a cabo la refactorización.

Keywords: Revisión Sistemática, Refactorización, Valor, Mantenibilidad

1 Introducción

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. Aunque la calidad puede describirse desde diferentes perspectivas, cuando se trata de calidad software está tradicionalmente relacionada con la calidad del producto y la calidad del proceso [2]. En cuanto a la calidad del producto software, la mantenibilidad ha sido reconocida históricamente como una de

las características más relevantes debido a su impacto directo sobre el costo de desarrollo y mantenimiento del software. 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 costo de la fase de desarrollo [3][4][5][6]. Por ejemplo, según [5], el tiempo que un programador invierte en mantenimiento es alrededor del 61% frente al 39% dedicado al desarrollo.

Una vez conocida la mantenibilidad del producto software, el siguiente paso es intentar mejorarla mediante prácticas de Ingeniería del Software. Actualmente la mayoría de estas prácticas se basan en un enfoque de "valor neutro", tratándose con igual importancia y sin tener en cuenta el valor que aportan cada uno de estos elementos al negocio. 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 [7]. Podemos definir al "valor" es la cuantificación de la importancia que un determinado artefacto o tarea tiene para todos los implicados en ese sistema [8]. En este sentido, un campo donde podría resultar interesante aplicar los principios de la ISBV y todavía no se aplica es en la refactorización de software, priorizando las mejoras de acuerdo con el valor que aporte al sistema. El término refactorización es introducido por primera vez en [9], definiéndose como una transformación parametrizada de un programa preservando su comportamiento que automáticamente modifica el diseño de la aplicación y el código fuente subyacente. 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" [10]. Por lo tanto, la refactorización puede verse tanto como una técnica para aumentar la mantenibilidad del software [11].

Finalmente, 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 consigue aumentar la mantenibilidad del software rápidamente, ayudando a la organización a construir un mejor software y reduciendo costos de mantenimiento, que por otra parte tiene una gran influencia en la evaluación del costo de los sistemas [12]. Estos costos estarán también relacionados con el tipo de refactorización: automática, asistida o manual.

En el siguiente trabajo se realiza una revisión sistemática de la literatura acerca de la relación entre las actividades de refactorización, su priorización basada en el valor y la evolución de la manteniblidad respecto de las refactorizaciones; con el objeto de conocer qué se ha realizado hasta el momento y obtener conclusiones. El artículo está dividido en 4 secciones. Además de la introducción, en la sección 2 se detalla el proceso de revisión sistemática. En la sección 3 se presentan los resultados y en la sección 4 se desarrollan las conclusiones.

2 El Método de Revisión Sistemática

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 [13]. Su elaboración implica una serie de pasos bien estructurados y definidos. A continuación se describe detalladamente estos pasos para la revisión sistemática realizada.

2.1 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 [13]. Para esta revisión sistemática se han definido las siguientes cuestiones de investigación, teniendo en cuenta lo comentado en la introducción:

- **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 costos 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 costos asociados con una refactorización.

2.2 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 [13] y [14]. Se realiza una búsqueda preliminar para encontrar otras revisiones sistemáticas y conocer el volumen esperado de la búsqueda. A partir de estas pruebas se identifican sinónimos y alternativas a los términos de búsqueda; y se construye la cadena de búsqueda. El resultado ha sido el siguiente: ((value OR valued OR value-based) AND (refactoring OR refactor) AND (maintainability OR software maintenance) AND (software OR application software OR software system)).

El proceso de búsqueda consiste en consultar las diferentes fuentes de datos seleccionadas que se enumeran en la Tabla 1, 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 [15][16] y contienen publicaciones que se consideran relevantes para el área de interés. 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 1 se muestra el resumen de los resultados obtenidos. Además de los recursos electrónicos que muestra la Tabla 1, 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.

El criterio de inclusión para la revisión sistemática considera estudios que dirijan su investigación hacia la refactorización guiada por el valor y permitan estudiar cómo identificar puntos de refactorización en el software para aumentar su valor, así como las técnicas a utilizar para su realización. Traten sobre la característica de la

mantenibilidad y especifiquen cómo una refactorización puede mejorarla. También se incluyen estudios que realicen afirmaciones sobre cómo influye la refactorización en la mantenibilidad del software.

Tabla 1. Resultados de la búsqueda (12/03/2015).

Base de datos	Total	Repetidos	Relevantes	Incluidos
ACM Digital Library	1022	0	12	7
Springer Link	265	146	3	3
Science Direct	184	109	1	1
ProQuest Computing	2	2	1	0
IEEE Computer Society	762	34	8	4
IEEE Xplore	143	27	1	0
Current Contents	6	14	1	0
Total	2384	332	27	15

Tras una primera búsqueda se identificaron 2384 estudios, de los cuales 332 aparecían por duplicado y fueron excluidos. Tras esto, los artículos fueron siendo eliminados mediante la lectura del título y el resumen. Los restantes 27 a formar parte del siguiente paso, la lectura completa de artículos. Tras la lectura detallada de los 27 trabajos, se seleccionaron 13 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.

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. Las preguntas han sido elaboradas a partir de los trabajos de Crombie [17], Fink [18] y Kitchenham [13]. Esta evaluación de la calidad se ha tenido en cuenta para la obtención de las diferentes conclusiones sobre los estudios, dando más relevancia a los estudios que han obtenido una mejor calidad. No se ha incluido el detalle del estudio por motivos de espacio.

Tabla 2. Lista de estudios seleccionados

ID	Ref.	Título
S1	[19]	Refactoring--Does It Improve Software Quality?
S2	[20]	Economics-Driven Software Mining
S3	[21]	Assessing the Maintainability Benefits of Design Restructuring Using Dependency Analysis
S4	[22]	A Quantitative Evaluation of Maintainability Enhancement by Refactoring
S5	[23]	A Survey of Software Refactoring

ID	Ref.	Título
S6	[24]	The effects of design pattern application on metric scores
S7	[25]	Understanding the Economics of Refactoring
S8	[26]	Evaluating Architectural Stability with Real Options Theory
S9	[27]	Search-Based Refactoring: an empirical study
S10	[28]	Applying ArchOptions to value the payoff of Refactoring
S11	[29]	Search-Based Refactoring for Software Maintenance
S12	[30]	Empirical investigation of refactoring effect on software quality
S13	[31]	Designing Systems for Adaptability by Means of Architecture Options
S14	[32]	A Quantitative Evaluation of Maintainability Enhancement by Refactoring
S15	[33]	Prioritizing Code Clone Detection Results for Clone Management

3 Resultados

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" o "bad smells" 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.

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 costos 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 costo de las refactorizaciones, sino solamente la maximización de la mejora en la calidad dependiente de un modelo de medición determinado.

El artículo (S15) por su parte prioriza las tareas de refactorización de código duplicado. Desarrolla un modelo de medición para ello, teniendo en cuenta el costo de no hacer la refactorización (teniendo en cuenta métricas de calidad interna del código fuente y de pronóstico sobre la estabilidad de dicho código). En este caso se centra en los costos, y no desarrolla la justificación del modelo.

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

3.1 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 sub cuestiones 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, aun así, no menciona la ISBV ni desarrolla las técnicas necesarias. (S4 y S14) por su parte utilizan los “malos olores” 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 costo 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 [34], 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 [35]. Por lo tanto la priorización se basa solamente en la mejora de calidad, sin tener en cuenta aspectos de valor. Esto mismo sucede con (S15), afirmando que en todos los casos la refactorización es positiva y priorizando por los costos de la misma.

A continuación se resume en la Fig. 1 el número de trabajos que tienen en cuenta los costos y beneficios económicos de una refactorización respecto de los que solo tienen en cuenta aspectos de mejora en las características de calidad.

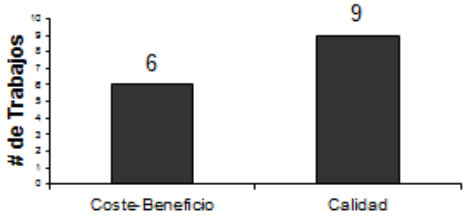

Fig. 1. Enfoque para la cuestión 1 de los trabajos encontrados

3.2 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 [36]. 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 costos de mantenimiento a partir de las refactorizaciones. En (S15) no existe una afirmación explícita, pero relaciona la baja mantenibilidad con la realización de refactorización; se podría concluir que la refactorización mejora la mantenibilidad. A continuación se resume en la Fig. 2 el número de trabajos que afirman el aumento (+) o disminución (-) de la mantenibilidad con las refactorizaciones. Y se indica el número de artículos que no realizan afirmación (=).

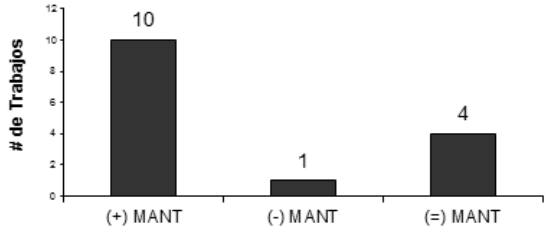

Fig. 2. Enfoque para la cuestión 2 de los trabajos encontrados.

3.3 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, S14 y S15) 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 Fig. **3** el número de trabajos que tienen en cuenta las refactorizaciones manuales, asistidas, automáticas o que por el contrario no tratan sobre esta tipología.

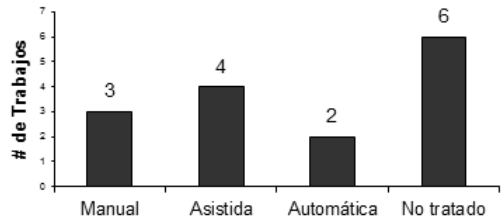

Fig. 3. Enfoque para la cuestión 3 de los trabajos encontrados.

4 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 [37][38][39][40][41][42][45][46] 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 [39][40][43][44], o también conocido como *Extract Method*. En resumen:

- 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 no se ha abordado de manera sistemática la refactorización de ese punto de vista.
- 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.

Para trabajos futuros, se propone la realización de un estudio demostrando que las refactorizaciones conllevan un aumento de la mantenibilidad, indicando el valor que las diferentes refactorizaciones pueden aportar a la mantenibilidad del software.

Referencias

[1] Rifkin, S. (2009) Guest Editor's Introduction: Software Measurement. IEEE Software, 26, p. 70.

[2] Ebert C. (2009) Guest Editor's Introduction: How Open Source Tools Can Benefit Industry. IEEE Software, 26, p: 50-51.

[3] Frazer, A. (1992) Reverse engineering- hype, hope or here? Software Reuse and Reverse Engineering in Practice, p: 209-243.

[4] Pressman, R., (2002) Ingeniería del Software: un enfoque práctico, Madrid: McGraw-Hill.

[5] Janice, S. (1998) Practices of Software Maintenance, in Proceedings of the International Conference on Software Maintenance, IEEE Computer Society.

[6] Harrison, W. (2005) What Do Software Developers Need to Know about Business? IEEE Software, 22(5), p: 5-7.

[7] Boehm, B. (2005) Value-Based Software Engineering: Overview and Agenda in Value-Based Software Engineering S. Biffl, et al., Editors, Springer, p: 3-14.

[8] Daniel Cabrero Moreno (2009) Construcción y Evolución del Software Basados en Valor, in Departamento de Tecnologías y Sistemas de la Información, p: 262.

[9] Opdyke, W. (1992) Refactoring Object Oriented Frameworks, in Computer Science, Illinois: Urbana-Champain.

[10] Fowler, M., et al. (2000) Refactoring: Improving the Design of Existing Code. 1st edition, Addison-Wesley Professional.

[11] IEEE, IEEE Standard 610.12–1990 IEEE Standard Glossary of Software Engineering Terminology. 1990, Institute of Electrical and Electronics Engineers.

[12] Wiederhold, G. (2006) What is your software worth? Communications of the ACM, 49(9), p: 65-75.

[13] Kitchenham B. and Charters S. (2007) Guidelines for performing Systematic Literature Reviews in Software Engineering, Keele University and Durham University Joint Report, Technical Report.

[14] 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.

[15] 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.

[16] Dybå T, Dingsøyr T. (2008). Empirical studies of agile software development: A systematic review. Inf Softw Technol, 50,9-10, pp:833-859.

[17] Crombie, I.K. (1996) The Pocket Guide To Critical Appraisal, BMJ Books ISBN: 978-0727910998.

[18] Fink, A. (2005) Conducting Research Literature Reviews: From the Internet to Paper, Sage Publications Inc., ISBN: 978-0761909057.

[19] Stroggylos, K. and Spinellis, D. (2007) Refactoring--Does It Improve Software Quality?. In Proceeding 5th International Workshop on Software Quality, IEEE Computer Society, 10.

[20] 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.

[21] 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.

[22] 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.

[23] Mens, T.; Tourwe, T., "A survey of software refactoring," Software Engineering, IEEE Transactions on , vol.30, no.2, pp.126,139, Feb 2004.

[24] Huston, B. (2001) The effects of design pattern application on metric scores"; Journal of Systems and Software, 58, 3, pp: 261-269.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] O'Keeffe M. Search-based refactoring for software maintenance. J.Syst.Software, 81, 4, pp: 502-516.

[30] Alshayeb M. (2009) Empirical investigation of refactoring effect on software quality. Information and Software Technology, 9;51, 9, pp:1319-1326.

[31] Engel A, Browning TR. (2008) Designing systems for adaptability by means of architecture options. Systems Engineering, 11, 2, pp: 125-146.

[32] 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.

[33] Venkatasubramanyam, R. D., Gupta, S., & Singh, H. K. (2013, May). Prioritizing code clone detection results for clone management. In Proceedings of the 7th International Workshop on Software Clones (pp. 30-36). IEEE Press.

[34] 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.

[35] Bansiya J, Davis C. A Hierarchical Model for Object-Oriented Design Quality Assessment. IEEE Transactions on Software Engineering 2002;28(1):4-17.

[36] 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.

[37] 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.

[38] 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.

[39] 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.

[40] 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.

[41] Chahal, K.K. and Singh, H. (2009) Metrics to study symptoms of bad software designs, SIGSOFT Softw Eng Notes, 34, 1, pp: 1-4.

[42] 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.

[43] 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.

[44] 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.

[45] 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.

[46] 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.