Entre Ciencia e Ingeniería, ISSN 1909-8367
Año 10 No. 19 - Primer Semestre de 2016, página 40-49

Análisis y clasificación de atributos de mantenibilidad del software: una revisión comparativa desde el estado del arte[1]

Analysis and classification of software maintainability attributes: a comparative review from the state of the art

Análise e classificação de atributos de manutenção do software: uma revisão comparativa desde o estado da arte

J.D. Erazo, A.S. Florez y F.J. Pino

Recibido Septiembre 08 de 2015 – Aceptado Febrero 19 de 2016

***Resumen*-- Actualmente el mantenimiento de software es la etapa más costosa del ciclo de vida de dicho producto. Identificar los atributos que influyen sobre la mantenibilidad de software es un aspecto importante para conocer qué factores se podrían incluir durante el proceso de desarrollo con el fin de conseguir un producto altamente mantenible. En este sentido, el presente artículo ofrece una vista integral de diferentes atributos de mantenibilidad obtenidos a partir de la literatura y propone una clasificación de los mismos teniendo en cuenta: (i) las sub-características de mantenibilidad de ISO/IEC 25010 sobre las que influye, y (ii) el flujo de trabajo del desarrollo de software de RUP (*Rational Unified Process*) en los que se presenta. Como resultado de la investigación realizada se obtuvieron un total de 18 atributos clasificados de acuerdo a los criterios mencionados anteriormente, los cuales describen diferentes aspectos que se deben considerar cuando se pretende desarrollar un producto altamente mantenible. Los atributos de mantenibilidad y su clasificación, obtenidos en esta investigación han sido utilizados en la realización de un modelo de referencia de procesos que apoya la inclusión de sub-características de mantenibilidad al producto software durante el proceso de desarrollo.**

***Palabras clave*----atributos de mantenibilidad, mantenibilidad de software, sub-características de mantenibilidad.**

***Abstract*--- Nowadays software maintenance is the most expensive stage in the life cycle of a software product. Identifying the attributes that influence software maintainability is an important aspect that will be useful when knowing the factors to be included during the development process in order to achieve a highly maintainable product. In this sense, this paper provides a comprehensive overview of the different maintainability attributes based on the literature, and proposes their classification taking into account the following items: (i) the maintainability sub-characteristics from the ISO/IEC 25010 standard influenced by it, and (ii) the software development workflow of RUP (Rational Unified Process) in which it is presented. As a result of this research, a total of 18 attributes were obtained and classified, based on the criteria previously mentioned, which describe the different aspects to be considered when trying to develop a highly maintainable product. The maintainability attributes and the classification obtained in this research have been used to create a process reference model which supports the inclusion of maintainability sub-characteristics to the product during the software development process.**

***Key words*----maintainability attributes, software maintainability, maintainability sub-characteristics**.

[1]Producto derivado del proyecto de grado en la modalidad de investigación "Modelo de referencia para la inclusión de sub-características de mantenibilidad al producto software durante el proceso de desarrollo". Trabajo adscrito al Grupo de Investigación y Desarrollo en Ingeniería de Software de la Facultad de Ingeniería Electrónica y Telecomunicaciones de la Universidad del Cauca en Popayán – Cauca (Colombia).

J.D. Erazo es Ingeniera de Sistemas de la Universidad del Cauca, Popayán - Cauca (Colombia); e-mail: jderazo@unicauca.edu.co

A.S. Florez es Ingeniero de Sistemas de la Universidad del Cauca, Popayán - Cauca (Colombia); e-mail: asflorez@unicauca.edu.co

F.J. Pino es profesor titular en la facultad de Ingeniería Electrónica y Telecomunicaciones de la Universidad del Cauca, Popayán - Cauca (Colombia); e-mail: fjpino@unicauca.edu.co

***Resumo* – Atualmente a manutenção de software é a etapa mais custosa do ciclo de vida do produto. Identificar os atributos influencia a capacidade de manutenção de software é um aspecto importante para conhecer que o fator poderia incluir durante o processo de desenvolvimento com o fim de conseguir um produto altamente manutenível. Nesse sentido, o presente artigo oferece uma vista integral de diferentes atributos de manutenção obtida as partir da literatura e propõe uma classificação dos mesmos tendo em conta: (i)as sub-características de manutenção de ISSO/IEC 25010 sobre as que influem e (ii) o fluxo de trabalho do desenvolvimento de software de RUP (Rational Unified Process) nos que se apresenta. Como resultado da investigação realizada se obteve um total de 18 atributos classificados de acordo aos critérios mencionados anteriormente, os quais descrevem diferentes aspectos que devem considerar quando se pretende desenvolver um produto altamente manutenível. Os atributos de manutenção e sua classificação, obtidos nessa investigação foram utilizados na realização de um modelo de referência de processos que apoiam a inclusão das sub-características de manutenção do produto software durante o processo de desenvolvimento.**

***Palavras chave:* atributos de manutenção, manutenção de software, sub- características de manutenção.**

I. Introducción

La mantenibilidad es uno de los atributos de calidad esenciales, ya que las tareas de mantenimiento consumen una gran proporción del esfuerzo total gastado en el ciclo de vida del software [1]. El costo de esta etapa consume entre el 50% y el 80% de los recursos del proyecto [2] y el 66% de los costos del ciclo de vida del software son invertidos en el mantenimiento del producto [3]. Además, el 61% del tiempo que dedican los programadores al desarrollo es invertido en la etapa de mantenimiento, y sólo el 39% es empleado en nuevos desarrollos [4]. Lo anterior refleja que la etapa de mantenimiento: (1) requiere el mayor porcentaje de los costos del ciclo de vida del software, (2) incrementa el esfuerzo realizado, (3) impide que una gran parte del tiempo sea utilizado para nuevos desarrollos.

Dado que la etapa de mantenimiento genera altos costos durante el ciclo de vida del desarrollo de software, para facilitar su ejecución es conveniente tener en cuenta la característica de mantenibilidad de software. ISO/IEC 25010 [5] define la mantenibilidad como el grado de efectividad o eficiencia con la que un producto o sistema puede ser modificado. La mantenibilidad de software se descompone en una serie de atributos que pueden ser considerados durante diferentes etapas del proceso de desarrollo. ISO/IEC 25010 define atributo como propiedad inherente o característica de una entidad que puede ser distinguida cualitativa o cuantitativamente por medios humanos o automatizados.

Estudios previos han demostrado que una de las causas de los problemas en el mantenimiento del software es que a menudo, la mantenibilidad no es una consideración importante durante las etapas de diseño e implementación de software [3]. Realizar más esfuerzo durante el ciclo de vida de desarrollo para hacer que el software sea mantenible puede reducir significativamente el total de los costos del software [6]. Por esto, es importante conocer los atributos de software que afectan a la mantenibilidad y relacionarlos con la(s) etapa(s) del proceso de desarrollo mediante la(s) cual(es) dichos atributos pueden ser incorporados en el producto software. De esta forma se podría lograr incluir las sub-características de mantenibilidad al producto software para conseguir un producto altamente mantenible. Esto apoyaría a reducir el esfuerzo de mantenimiento, lo que permitiría usar estos mismos recursos para realizar más cambios o lograr los mismos cambios con menos recursos [7].

En este sentido, el resultado principal de este artículo es ofrecer una vista integral de diferentes atributos de mantenibilidad obtenidos a partir de la literatura y una clasificación de los mismos teniendo en cuenta: (i) las sub-características de mantenibilidad de ISO/IEC 25010 sobre las que influye: capacidad para ser analizado, modularidad, capacidad para ser modificado, reusabilidad y capacidad para ser probado; y (ii) el flujo de trabajo del desarrollo de software de RUP (Rational Unified Process) en los que se presenta: Modelo de Negocio, requisitos, análisis y diseño, implementación, pruebas y despliegue. El objetivo es ofrecer a las entidades desarrolladoras de software los aspectos básicos que se deben tener en cuenta durante el proceso de desarrollo de software para incrementar la mantenibilidad del producto software. Este trabajo ha sido utilizado como base para la construcción de un modelo de referencia de procesos (MANTuS) que pretende apoyar la inclusión de sub-características de mantenibilidad al producto software durante el proceso de desarrollo.

Además de la presente introducción, este artículo muestra en la sección 2 la estrategia de investigación utilizada para la clasificación de los atributos de mantenibilidad y los trabajos relacionados, en la sección 3 la unificación de conceptos relacionados con mantenibilidad, en la sección 4 la clasificación de atributos realizada, la sección 5 una vista general del Modelo de referencia de procesos y finalmente en la sección 6 conclusiones y trabajo futuro.

II. Antecedentes

A. Estrategia de investigación

Para identificar los atributos software que influyen en la mantenibilidad del producto, se realizó una búsqueda en la literatura consultando bases de datos electrónicas, primero se seleccionaron las bases de datos para cumplir con los objetivos de la búsqueda acerca de la mantenibilidad de software. La biblioteca digital utilizada para realizar la investigación fue Google Scholar. Para realizar la búsqueda automática en la biblioteca digital seleccionada, se utilizó la cadena de búsqueda "*software maintainability" AND "attribute"*. La cadena contiene dos partes que incluyen

los conceptos necesarios para tener un gran alcance en la investigación acerca de los atributos de mantenibilidad del producto software. La primera parte está relacionada con los estudios que hacen referencia a la mantenibilidad del producto software, la segunda parte corresponde a las investigaciones que realizan estudios de atributos de mantenibilidad. Como resultado de esta búsqueda, se obtuvieron inicialmente diferentes factores, algunos de ellos mencionados en varios estudios, por lo que fue necesario asociar los conceptos encontrados para obtener un único concepto. Se encontraron 61 aspectos (factores, métricas, atributos) que influyen sobre esta característica. Después de realizar un análisis de la definición de estos aspectos, se observó que algunos de ellos no eran atributos de producto y otros estaban definidos como métricas, por lo cual no se tienen en cuenta en esta investigación. Por lo anterior, en total se encontraron 18 atributos software que se consideran en la clasificación final realizada.

Posteriormente, estos atributos fueron clasificados teniendo en cuenta dos aspectos: las sub-características de mantenibilidad sobre las que influyen y los flujos de trabajo en los que se presenta. Para clasificarlas en cada una de las sub-características de mantenibilidad se realizó un análisis semántico teniendo en cuenta las definiciones tanto de los atributos encontrados como de las sub-características definidas en la norma ISO/IEC 25010 (modularidad, reusabilidad, capacidad para ser analizado, capacidad para ser modificado y capacidad para ser probado). Además, para clasificar estos atributos en la etapa de desarrollo en la que se presentan, se tuvieron en cuenta los flujos de trabajo definidos en RUP: Modelo de Negocio, requisitos, análisis y diseño, implementación, pruebas y despliegue. Para tener un criterio de clasificación se buscaron estudios donde se evidencia la relación entre el atributo con la sub-característica de mantenibilidad y el flujo de trabajo al que ha sido asignado.

B. Trabajos relacionados

Algunos trabajos relacionados con la descripción de atributos de mantenibilidad que son relevantes en la literatura se presentan a continuación:

- **Factores claves que afectan la mantenibilidad de software.** Kumar [6] afirma que la mantenibilidad de un producto software es afectada por principios de diseño y arquitectura, describe varios factores propuestos por diferentes investigadores en modelos de mantenibilidad de software, los cuales son de gran importancia en las evaluaciones de mantenibilidad. Entre estos factores se destacan: la estabilidad, facilidad de cambio, facilidad de análisis y facilidad de prueba, establecidos por la norma ISO 9126, y además de esto Kumar describe otros factores planteados por diferentes autores en [8-10], como: modularidad, documentación, facilidad de lectura, consistencia, simplicidad, capacidad de expansión, instrumentación, estandarización, lenguaje de programación, nivel de validación y pruebas, complejidad, trazabilidad, algunas propiedades estructurales o de código y habilidades de equipo de mantenimiento. Anda [8] realiza un estudio empírico a partir del cual define diversos factores que afectan la mantenibilidad del software, entre los cuales se encuentran: elección de clases y nombre, diseño, patrones de diseño, arquitectura, componente, encapsulamiento, herencia, librerías, simplicidad, comentarios y plataforma técnica. Los factores mencionados en estos estudios son de vital importancia para esta investigación, ya que permiten conocer los aspectos primordiales que debe tener un producto software para lograr un alto nivel de mantenibilidad.

- **Impacto de atributos internos del producto software sobre la mantenibilidad.** Kozlov et. al. [11] evalúan el impacto que tienen algunos atributos internos del producto software sobre la característica de mantenibilidad, obteniendo como resultado que el número de líneas de comentario y número de módulos influyen sobre la misma y además se encuentran otros factores que influyen negativamente sobre esta, tales como: número de líneas de código, tipos de datos globales y locales, número total de ítems de entrada y salida, y complejidad de interfaz de los métodos de clase. El conocimiento acerca de la influencia que tienen estos atributos de calidad internos sobre la mantenibilidad puede ser utilizado a favor durante el proceso de desarrollo. Estos resultados sirven como base para mejorar la mantenibilidad de software en etapas tempranas del proceso de desarrollo de software, lo cual presenta un gran aporte para esta investigación.

- **Modelo de atributos de mantenibilidad de software.** Hashim y Key [9] proponen un modelo que resalta la necesidad de mejorar la calidad del producto software de tal forma que el mantenimiento sea eficiente. En el artículo se destacan problemas asociados con el mantenimiento los cuales se relacionan con las deficiencias de la forma en la que los sistemas son desarrollados, tales como: falta de trazabilidad, falta de documentación, mal diseño e implementación, herramientas de desarrollo y técnicas y lenguajes de programación inapropiados. Además se presentan algunos atributos de mantenibilidad, que coinciden con los expuestos en [6]. El principal aporte de este estudio es que expone los problemas más importantes asociados al mantenimiento, los cuales deben ser tenidos en cuenta en esta investigación para buscar soluciones y así aumentar la facilidad de mantenimiento del producto software.

Aunque se puede ver que hay algunos trabajos relacionados con atributos de mantenibilidad de software, no se han encontrado estudios que realicen una clasificación de los mismos de acuerdo a las sub-características de mantenibilidad y el flujo de trabajo, el cual es el principal objetivo de esta investigación.

TABLA I.

DEFINICIÓN UNIFICADA DE LOS ATRIBUTOS DE MANTENIBILIDAD IDENTIFICADOS.

Atributo	Descripción	Fuente
Acoplamiento	Es una medida del grado de interdependencias que existe entre módulos de software. Los componentes bien acoplados son más resistentes al cambio.	[12-17]
Anidación	Profundidad máxima de anidamiento del módulo, mide el porcentaje de módulos anidados.	[16]
Capacidad de expansión	Un cambio físico a la información, funciones computacionales, o almacenamiento de datos puede ser realizado fácilmente.	[6]
Cohesión	Grado de comunicación entre los métodos y variables miembros de una clase.	[12, 13, 15, 16, 18]
Comentarios	Relación existente entre el número total de líneas de código y el número de líneas de comentarios.	[6, 8, 10-12, 16, 18-20]
Complejidad	Dificultad en la comprensión o el mantenimiento de los códigos. La medición de la complejidad de un módulo implica medir el flujo de control del módulo, el flujo de datos e incluso las estructuras de datos utilizadas.	[9, 12-17, 19, 20]
Consistencia	Indica que el producto software asocia y contiene simbología, terminología y notación uniforme.	[6, 16]
Documentación	Contiene explicaciones detalladas del diseño de software.	[9, 12, 16, 21]
Duplicación	Es el grado en que una secuencia de código fuente ocurre más de una vez. La excesiva duplicación es perjudicial para la mantenibilidad ya que hace que un sistema sea más largo de lo que necesita ser.	[14, 17, 19]
Encapsulamiento	Hacer privadas las variables que son innecesarias para el tratamiento del objeto pero necesarias para su funcionamiento, así como las funciones que no necesitan interacción del usuario o que solo pueden ser llamadas por otras funciones dentro del objeto.	[8, 12, 14, 16]
Estandarización	Conjunto de estándares de programación (paquetes, clases, métodos y variables) que deben estar disponibles para actuar como una guía en la escritura de código y que evita la idiosincrasia entre los programadores.	[9, 12]
Facilidad de lectura	Grado al cual un lector puede entender fácil y rápidamente el código fuente.	[9, 21]
Facilidad de entendimiento	Intenta estimar la correlación entre el estilo de la escritura de código fuente y la documentación correspondiente.	[21]
Herencia	Se pueden crear nuevas clases partiendo de una clase o de una jerarquía de clases preexistente.	[8, 12, 15]
Polimorfismo	Es utilizado en la programación orientada a objetos para realizar un ligado en tiempo de ejecución a una clase entre varias otras clases de la misma jerarquía. Ayuda a procesar instancias de clases, de acuerdo a su tipo de dato o clase.	[15]
Simplicidad	Relacionado con el tamaño y la complejidad de un sistema. Toma más tiempo identificar una clase específica cuando hay muchas clases. La presencia de muchas clases casi vacías es una señal de código poco mantenible.	[8, 12]
Tamaño	Cantidad de código del sistema.	[14, 16, 17, 19, 20]
Trazabilidad	Habilidad para trazar la representación de diseño o los componentes actuales del programa hacia los requerimientos.	[9]

III. Unificación de conceptos de atributos de mantenibilidad

En los estudios revisados, que se encuentran referenciados en la tercera columna de la Tabla I, se halló que algunos de ellos presentaban atributos de mantenibilidad similares, aunque no coincidían totalmente en el nombre o definición de los mismos, por lo que fue necesario estructurar un único concepto para cada atributo. Para la unificación de los conceptos se analizaron las definiciones de cada atributo expuestas en los artículos y posteriormente se tuvo en cuenta lo más relevante de cada uno de ellos, logrando así agruparlos en un único concepto. Como resultado de este análisis se obtuvieron 18 atributos con su respectiva definición, los cuales se presentan en la Tabla I. Incluir estos atributos al producto durante el proceso de desarrollo permite potenciar las sub-característica de mantenibilidad de dicho producto y mejorar las medidas de las métricas asociadas a las sub-características.

IV. Clasificación de atributos por sub-características y flujos de trabajo.

Los atributos identificados anteriormente se clasificaron de acuerdo a dos aspectos: sub-características de mantenibilidad sobre las que influye y los flujos de trabajo en los que se presenta.

Para realizar la clasificación por sub-caracteristicas de mantenibilidad fue necesario hacer una comparación entre las sub-características de mantenibilidad presentadas por las normas ISO/IEC 9126-1 [22] e ISO/IEC 25010 [5]. La mantenibilidad de software presenta diferentes sub-características, de acuerdo a la norma ISO/IEC 9126-1 esta se divide en cinco sub-características: capacidad para ser analizado (*Analysability*), capacidad para ser cambiado (*Changeability*), estabilidad (*Stability*), capacidad para ser probado (*Testability*) y cumplimiento de la mantenibilidad (*Compliance*). Por otra parte la norma ISO/IEC 25010 divide la mantenibilidad en cinco características: modularidad (*Modularity*), reusabilidad (*Reusability*), capacidad para ser analizado (*Analysability*), capacidad para ser modificado (*Modifiability*) y capacidad para ser probado (*Testability*). Debido a que la norma ISO/IEC 25010 se deriva de la ISO/IEC 9126, al realizar la comparación entre las sub-características definidas por estas dos normas se encuentra que la capacidad para ser analizado y la capacidad para ser probado se presentan en las dos normas. Además la sub-característica capacidad para ser modificado presentada en ISO/IEC 25010 es la combinación de las sub-características capacidad para ser cambiado y estabilidad, presentadas en la norma ISO/IEC 9126-1. Por último, la norma ISO / IEC 25010 incluye dos nuevas sub-características a la mantenibilidad: modularidad y reusabilidad. Es por esto que las sub-características que se tuvieron en cuenta para realizar la clasificación de los atributos son las definidas en la norma ISO/IEC 25010.

Con el fin de clasificar estos atributos en cada una de las sub-características de mantenibilidad se realizó un análisis semántico teniendo en cuenta las definiciones tanto de los atributos encontrados como de las sub-características definidas en la norma ISO/IEC 25010. Además, para clasificar estos atributos en la etapa de desarrollo en la que se presentan, se tuvieron en cuenta los flujos de trabajo definidos en RUP: Modelo de Negocio, requisitos, análisis y diseño, implementación, pruebas y despliegue. Los resultados obtenidos se presentan en la Tabla II.

Para tener un criterio de clasificación se buscaron estudios donde se evidencia la relación entre el atributo con la sub-característica de mantenibilidad y el flujo de trabajo al que ha sido asignado. Debido a que la relación de algunos atributos con las sub-características de mantenibilidad no fue encontrada en la literatura revisada, para estos casos se hizo un análisis por parte de los investigadores basándose en las definiciones tanto de los atributos como de las sub-características. A continuación se presenta como ejemplo el análisis realizado para algunos atributos (acoplamiento, cohesión, documentación, estandarización y trazabilidad). El análisis de los demás atributos de mantenibilidad se encuentra en [23].

Acoplamiento: En [24] se afirma que el acoplamiento en los sistemas software tiene un fuerte impacto negativo en la calidad de software y por lo tanto se debe mantener al mínimo durante la etapa de diseño. En [25] el autor se refiere al acoplamiento como propiedad de diseño. En [26] el autor se refiere al acoplamiento como un concepto integral de diseño. En [27] se indica que el acoplamiento ha sido identificado como uno de las propiedades básicas de la calidad del diseño del software. En [28] se afirma que el acoplamiento es un factor de calidad muy importante para el diseño y la implementación orientada a objetos. En [29] se evidencia que el acoplamiento debe ser considerado durante el diseño, porque se afirma que es una de las características importantes que brinda eficiencia al diseño en la orientación a objetos. En este estudio también se indica que al aumentar el acoplamiento también aumenta la complejidad del diseño, lo que hace que se necesite mayor esfuerzo para realizar las pruebas reduciendo así la capacidad para ser probado del diseño, lo cual demuestra que hay relación entre el acoplamiento y esta sub-característica. En [30] se demuestra que existe relación entre el acoplamiento con la capacidad para ser analizado y la reusabilidad del software, porque se dice que un bajo acoplamiento mejora estas dos sub-características. En [31] se afirma que el acoplamiento excesivo entre las clases de un sistema afecta la modularidad del mismo y también que la medida del acoplamiento es un buen indicador de la capacidad para ser probado. En [32] se realiza un estudio donde se concluye que la métrica de acoplamiento entre objetos presenta correlación con la sub-característica de mantenibilidad capacidad para ser modificado. En [33, 34] se muestra que el acoplamiento se relaciona con la capacidad para ser modificado, la capacidad para ser probado y la reusabilidad del sistema, ya que se

dice que cuando el acoplamiento aumenta la reusabilidad disminuye y se hace más difícil modificar y probar el sistema.

Cohesión: En [29] se afirma que la cohesión es una característica importante que ayuda a la eficiencia del diseño. En [25] el autor se refiere a la cohesión como propiedad de diseño. En [35] el autor habla de la cohesión como un atributo de diseño más que de código y un atributo que puede ser usado para predecir las propiedades de implementación como "facilidad de depuración, facilidad de mantenimiento y facilidad de modificación". En [36] se dice que la cohesión es una de las propiedades de diseño más importantes. En [26] se argumenta que en el diseño orientado a objetos la cohesión en un gran beneficio, además el autor se refiere a la cohesión como un concepto integral de diseño. En [27] se indica que la cohesión ha sido identificada como uno de las propiedades básicas de la calidad del diseño del software. En [28] se afirma que la cohesión es un factor de calidad muy importante para el diseño y la implementación orientada a objetos. En [30] se afirma que una alta cohesión aumenta la capacidad para ser modificado y la modularidad del sistema. En el estudio [32] se concluye que la métrica falta de cohesión en métodos presenta correlación con la sub-característica de mantenibilidad capacidad para ser modificado. En [31, 37] se evidencia que la cohesión influye en la reusabilidad, ya que se dice que cuando hay alta cohesión los componentes tienden a tener alta mantenibilidad y reusabilidad. En [38] se afirma que cuanto menor es la cantidad de acoplamiento y la complejidad de los componentes y más alta sea la cohesión más fácil será la capacidad para ser analizado, la estabilidad y la capacidad para ser probado del producto software. En [39] indican que se ha demostrado que las métricas de la cohesión son buenos predictores de la capacidad para ser probado, al encontrar una correlación clara entre ellas.

Documentación: en [40] se afirma que los documentos de diseño son una fuente de información importante, especialmente cuando los sistemas entran en la fase de mantenimiento. En [41] se dice que las metodologías típicas del desarrollo orientado a objetos requieren que se documente el análisis, el diseño arquitectural o a alto nivel y el diseño a bajo nivel. En [3] se dice que en la industria del software es necesario mantener una trazabilidad bidireccional durante todo el ciclo de vida de los sistemas software y también se afirma que es un aspecto importante para la facilidad de entendimiento y capacidad de ser modificado del software. En [30] se indica que la documentación contribuye a capacidad para ser analizado, la capacidad para ser modificado y la reusabilidad del software. En [42] se dice que la documentación es un factor importante para la sub-característica capacidad para ser probado ya que en las pruebas es importante tener documentación clara de los requerimientos y especificaciones.

Estandarización: ya que este atributo implica tener un conjunto de estándares de programación definidos se evidencia que es importante considerarlo durante la etapa de implementación, además este puede mejorar la facilidad de lectura del código fuente, la facilidad de entendimiento del mismo y con esto la capacidad para ser analizado y modificado.

Facilidad de lectura: en [43] se afirma que debido a que la facilidad de lectura puede afectar la calidad del software, los programadores deben preocuparse por ella. En [44] se dice que se debe inspeccionar la facilidad de lectura del código fuente para asegurar la mantenibilidad, portabilidad y reusabilidad del software. En [45] se enuncia que la facilidad de lectura afecta la capacidad para ser analizado del software, ya que simplifica el trabajo de identificar las modificaciones que se requieren hacer al sistema. En [46] se expone que la facilidad de lectura mejora la facilidad para ser entendido de un componente, lo cual mejora la capacidad para ser analizado y también la capacidad para ser cambiado del mismo.

Trazabilidad: en [47, 48] se define la trazabilidad como una característica que se debe tener en cuenta en la especificación de los requisitos del software e indican que esta área permanece como un problema ampliamente reportado debido a que no hay un análisis de las fuentes de requerimientos utilizadas en los desarrollos de software. En [49] se afirma que en el proceso de desarrollo de software, la trazabilidad y evolución de los requisitos es uno de los factores más relevantes para lograr software fiable y exacto. En [40] se señala que un diseño es considerado de mayor calidad siempre que se mantenga la trazabilidad con el código. Como el diseño representa una abstracción de la implementación, se espera que las clases en el diseño estén representadas en el código. En [50] se indica que los vacíos en la trazabilidad, como características de información ausentes reducen la capacidad para ser modificado y la variabilidad de los componentes. Además de esto en el estudio se encuentra que varias características como la capacidad para ser analizado dependen de la trazabilidad de artefactos para análisis de impacto o comprensión del programa.

A continuación se presenta la Tabla II donde se encuentra la síntesis del análisis y clasificación realizados de los atributos de mantenibilidad. Esta tabla contiene el nombre de los atributos de mantenibilidad, la fuente de donde se obtuvieron, las cinco sub-características definidas por la norma ISO/IEC 25010 [5] con las que se relacionan, que se pueden identificar con (+) el cual indica que la relación está justificada por la literatura y con (-) el cual indica que la relación es justificada por un análisis realizado por los investigadores basándose en las definiciones tanto de los atributos como de las sub-características y por último los nombres de los flujos de trabajo definido por RUP en los que se presenta.

Tabla II

Clasificación de los atributos por sub-características de mantenibilidad y flujos de trabajo

Sub-características / Atributos	Fuentes	Capacidad para ser Analizado	Modularidad	Capacidad para ser Modificado	Capacidad para ser Probado	Reusabilidad	Flujo de trabajo
Acoplamiento	[12-17]	+	+	+	+	+	Análisis y Diseño (+)
Anidación	[16]	+		+	+		Análisis y Diseño (-) Implementación (+)
Capacidad de expansión	[6]			+			Despliegue (+)
Cohesión	[12, 13, 15, 16, 18]	+	+	+	+	+	Análisis y Diseño (+)
Comentarios	[6, 8, 10-12, 16, 18-20]	+		+		+	Implementación (+)
Complejidad	[9, 12-17, 19, 20]	+	-	+	+	+	Análisis y Diseño (-) Implementación (-)
Consistencia	[6, 16]	-		-			Requisitos (+) Análisis y Diseño (+) Implementación (+) Pruebas (+) Despliegue (+)
Documentación	[9, 12, 16, 21]	+		+	+	+	Requisitos (+) Análisis y Diseño (+) Implementación (+) Pruebas (+) Despliegue (+)
Duplicación	[14, 17, 19]	+		+	-		Implementación (+)
Encapsulamiento	[8, 12, 14, 16]	+	+	+	+		Análisis y Diseño(+)
Estandarización	[9, 12]	-		-			Implementación (-)
Facilidad de lectura	[9, 21]	+		+	-	+	Implementación (+)
Facilidad de entendimiento	[21]	+		-	-	+	Análisis y Diseño (+) Implementación (+)
Herencia	[8, 12, 15]	-		+	+	+	Análisis y Diseño (+)
Polimorfismo	[15]	-		-		+	Análisis y Diseño (+)
Simplicidad	[8, 12]	+	-	-	+	+	Requisitos (-) Análisis y Diseño (-) Implementación (-)
Tamaño	[14, 16, 17, 19, 20]	-		+	+	+	Implementación (+)
Trazabilidad	[9]	+		+			Requisitos (-) Análisis y Diseño (+) Implementación (+) Pruebas (-)

(+) Indica que la relación es justificada por la literatura
(-) la relación es justificada por un análisis de los investigadores.

V. Modelo de referencia mantus

El trabajo realizado previamente ha sido la base para la creación de un modelo de referencia de procesos denominado MANTuS, el cual pretende especificar, en términos de sus propósitos y sus resultados, un conjunto de procesos que permitan incluir atributos de mantenibilidad al producto software con el fin de potenciar esta característica y que sean aplicables en el contexto de empresas desarrolladoras de software. Debido a restricciones de espacio, a continuación se presenta una vista general de este modelo de referencia. El modelo completo y en detalle se puede encontrar en [23].

El Modelo de Referencia MANTuS asume que la mantenibilidad de software es un aspecto fundamental del producto que debe ser considerado desde el inicio del ciclo de vida del mismo. Es por esto que se establecen los siguientes seis procesos (ver Fig. 1): Análisis de requisitos

de software desde la perspectiva de mantenibilidad, diseño arquitectural de software desde la perspectiva de mantenibilidad, Diseño detallado de software desde la perspectiva de mantenibilidad, Construcción de software desde la perspectiva de mantenibilidad, Pruebas de evaluación de software desde la perspectiva de mantenibilidad, Gestión de la documentación desde la perspectiva de mantenibilidad. En la Fig. 1 se presenta la vista general de los procesos que conforman el modelo de referencia, se observa que estos procesos son independientes y que el proceso de gestión de la documentación desde la perspectiva de mantenibilidad es transversal a los otros, ya que éste es un proceso de soporte que puede ser ejecutado durante los otros cinco procesos. La vista general del modelo de referencia combina y sintetiza las prácticas base definidas en los procesos de todas las sub-características de mantenibilidad.

Vi. Conclusiones y trabajo futuro

En este estudio se han identificado un total de 18 atributos que se involucran en la mantenibilidad del software, los cuales han sido clasificados de acuerdo a las sub-características de mantenibilidad sobre las que influyen y el flujo de trabajo en los que se presenta. Esto con el fin de facilitar la identificación de los factores que se deben incluir en el producto software durante el proceso de desarrollo para conseguir un producto con alta mantenibilidad. Además de esto, de acuerdo a la clasificación obtenida se observa que la mayoría de los atributos identificados se presentan durante el flujo de trabajo: Análisis y diseño. También se observó que los atributos que influyen sobre el mayor número de sub-características de mantenibilidad son: acoplamiento, cohesión, complejidad, documentación, encapsulamiento, facilidad de entendimiento, herencia, simplicidad y tamaño. De igual forma, se puede notar que los atributos consistencia, documentación y trazabilidad se presentan en todos los flujos de trabajo, por lo cual deben ser considerados durante todo el proceso de desarrollo.

Teniendo en cuenta la clasificación de atributos de mantenibilidad presentada en este artículo, se construyó el Modelo de Referencia MANTuS el cual presenta procesos generales donde se agrupan todas las prácticas base definidas para cada una de las sub-características de mantenibilidad. Con este modelo de referencia se podrían disminuir los costos de mantenimiento durante el proceso de desarrollo y además las empresas podrían utilizarlo con el fin de obtener la certificación en mantenibilidad realizado por AQC Lab [51].

Agradecimientos

Los autores agradecen al Programa de Ingeniería de Sistemas de la Universidad del Cauca. Francisco J. Pino agradece a la Universidad del Cauca donde trabaja como profesor titular.

Referencias

[1] J. M. Conejero, E. Figueiredo, A. Garcia, J. Hernández, and E. Jurado, "On the relationship of concern metrics and requirements maintainability," *Information and Software Technology,* vol. 54, pp. 212-238, 2012.

[2] C. E. H. Chua, S. Purao, and V. C. Storey, "Developing maintainable software: The Readable approach," *Decision Support Systems,* vol. 42, pp. 469-491, 2006.

[3] J.-C. Chen and S.-J. Huang, "An empirical analysis of the impact of

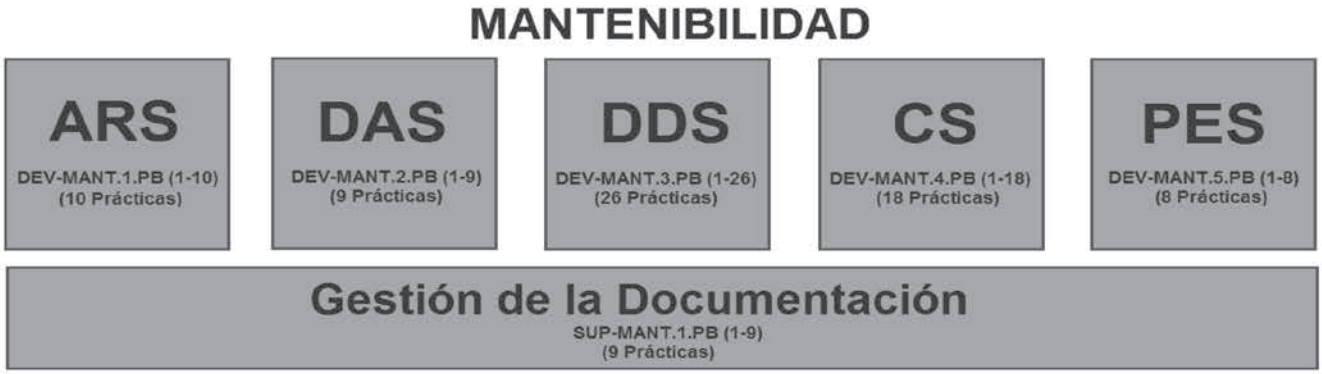

Fig. 1. Vista general del modelo de referencia (ARS= Análisis de requisitos de software, DAS= diseño arquitectural de software, DDS= diseño detallado de software, CS= construcción de software, PES= pruebas de evaluación de software)

software development problem factors on software maintainability," *Journal of Systems and Software,* vol. 82, pp. 981-992, 2009.

[4] F. J. Pino, F. Ruiz, F. García, and M. Piattini, "A software maintenance methodology for small organizations: Agile MANTEMA," *Journal Of Software Maintenance And Evolution: Research And Practice,* vol. 24, pp. 851-876, 2011.

[5] I. S. Organization, "ISO/IEC 25010," in *Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and Software Quality Models*, ed, 2011.

[6] B. Kumar, "A survey of key factors affecting software maintainability," in *2012 International Conference on Computing Sciences*, 2012, pp. 261-266.

[7] E. B. Swanson, "IS "maintainability": should it reduce the maintenance effort?," *ACM SIGMIS Database,* vol. 30, pp. 65-76, 1999.

[8] B. Anda, "Assessing software system maintainability using structural measures and expert assessments," in *Software Maintenance, 2007. ICSM 2007. IEEE International Conference on*, 2007, pp. 204-213.

[9] K. Hashim and E. Key, "A software maintainability attributes model," *Malaysian Journal of Computer Science,* vol. 9, pp. 92-97, 1996.

[10] K. Aggarwal, Y. Singh, P. Chandra, and M. Puri, "Measurement of software maintainability using a fuzzy model," *Journal of Computer Science,* vol. 1, p. 538, 2005.

[11] D. Kozlov, J. Koskinen, M. Sakkinen, and J. Markkula, "Assessing maintainability change over multiple software releases," *Journal of Software Maintenance and Evolution,* vol. 20, pp. 31-58, 2008.

[12] A. Yamashita and L. Moonen, "Do code smells reflect important maintainability aspects?," in *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*, 2012, pp. 306-315.

[13] M. Perepletchikov, C. Ryan, K. Frampton, and Z. Tari, "Coupling metrics for predicting maintainability in service-oriented designs," in *Proceedings of the 2007 Australian Software Engineering Conference (ASWEC'07)*, 2007, pp. 329-338.

[14] R. Baggen, J. P. Correia, K. Schill, and J. Visser, "Standardized code quality benchmarking for improving software maintainability," *Software Quality Journal,* vol. 20, pp. 287-307, 2012.

[15] S. C. Misra, "Modeling design/coding factors that drive maintainability of software systems," *Software Quality Journal,* vol. 13, pp. 297-320, 2005.

[16] P. Oman and J. Hagemeister, "Metrics for assessing a software system's maintainability," in *Software Maintenance, 1992. Proceerdings., Conference on*, 1992, pp. 337-344.

[17] W. Hordijk and R. Wieringa, "Surveying the factors that influence maintainability: research design," in *ACM SIGSOFT Software Engineering Notes*, 2005, pp. 385-388.

[18] M. Riaz, E. Mendes, and E. Tempero, "A systematic review of software maintainability prediction and metrics," in *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*, 2009, pp. 367-377.

[19] I. Heitlager, T. Kuipers, and J. Visser, "A practical model for measuring maintainability," in *Proceedings of the 6th International Conference on Quality of Information and Communications Technology*, 2007, pp. 30-39.

[20] K. D. Welker, "The software maintainability index revisited," *CrossTalk,* pp. 18-21, 2001.

[21] K. K. Aggarwal, Y. Singh, and J. K. Chhabra, "An integrated measure of software maintainability," in *Reliability and maintainability symposium, 2002. Proceedings. Annual*, 2002, pp. 235-241.

[22] I. S. Organization, "ISO/IEC 9126-1," in *Software Engineering – Product Quality – Part 1: Quality Model*, ed, 2001.

[23] "Modelo de referencia para la inclusión de sub-características de mantenibilidad al producto software durante el proceso de desarrollo," Pregrado, Ingenieria de sistemas, Universidad del Cauca, Popayán, 2014.

[24] F. Brito e Abreu and W. Melo, "Evaluating the impact of object-oriented design on software quality," in *Software Metrics Symposium, 1996., Proceedings of the 3rd International*, 1996, pp. 90-99.

[25] M. Sefika, A. Sane, and R. H. Campbell, "Monitoring compliance of a software system with its high-level design models," in *Proceedings of the 18th international conference on Software engineering*, 1996, pp. 387-396.

[26] R. Subramanyam and M. S. Krishnan, "Empirical analysis of ck metrics for object-oriented design complexity: Implications for software defects," *Software Engineering, IEEE Transactions on,* vol. 29, pp. 297-310, 2003.

[27] E. Capra, C. Francalanci, and F. Merlo, "An empirical study on the relationship between software design quality, development effort and governance in open source projects," *Software Engineering, IEEE Transactions on,* vol. 34, pp. 765-782, 2008.

[28] L. Etzkorn and H. Delugach, "Towards a semantic metrics suite for object-oriented design," in *Technology of Object-Oriented Languages and Systems, 2000. TOOLS 34. Proceedings. 34th International Conference on*, 2000, pp. 71-80.

[29] S. Khalid, S. Zehra, and F. Arif, "Analysis of object oriented complexity and testability using object oriented design metrics," in *Proceedings of the 2010 National Software Engineering Conference*, 2010, p. 4.

[30] F. Zhang, A. Mockus, Y. Zou, F. Khomh, and A. E. Hassan, "How does Context Affect the Distribution of Software Maintainability Metrics?," in *Software Maintenance (ICSM), 2013 29th IEEE International Conference on*, 2013, pp. 350-359.

[31] M. Badri, L. Badri, and F. Touré, "Empirical Analysis of Object-Oriented Design Metrics: Towards a New Metric Using Control Flow Paths and Probabilities," *Journal of Object Technology,* vol. 8, pp. 123-142, 2009.

[32] M. Ajrnal Chaumun, H. Kabaili, R. K. Keller, F. Lustman, and G. Saint-Denis, "Design properties and object-oriented software changeability," in *Software Maintenance and Reengineering, 2000. Proceedings of the Fourth European*, 2000, pp. 45-54.

[33] P. S. Sandhu and H. Singh, "A Reusability Evaluation Model for OO-Based Software Components," *International Journal of Computer Science,* vol. 1, pp. 259-264, 2006.

[34] P. S. Sandhu, P. Blecharz, and H. Singh, "A Taguchi approach to investigate impact of factors for reusability of software components," *Transactions on Engineering, Computing and Technology,* vol. 19, pp. 135-140, 2007.

[35] J. M. Bieman and B.-K. Kang, "Measuring design-level cohesion," *Software Engineering, IEEE Transactions on,* vol. 24, pp. 111-124, 1998.

[36] H. Kabaili, R. K. Keller, and F. Lustman, "Cohesion as changeability indicator in object-oriented systems," in *Software Maintenance and Reengineering, 2001. Fifth European Conference on*, 2001, pp. 39-46.

[37] K. Sirbi and P. J. Kulkarni, "Impact of Aspect Oriented Programming on Software Development Quality Metrics," *Global Journal of Computer Science and Technology,* vol. 10, 2010.

[38] G. Shahmohammadii and S. Jaliliii, "A Quantitative Evaluation of Maintainability of Software Architecture Styles."

[39] M. Ó. Cinnéide, D. Boyle, and I. H. Moghadam, "Automated refactoring for testability," in *Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference on*, 2011, pp. 437-443.

[40] G. Antoniol, B. Caprile, A. Potrich, and P. Tonella, "Design code traceability for object oriented systems," *Annals of Software Engineering,* vol. 9, pp. 35-58, 2000.

[41] L. C. Briand, "Software documentation: how much is enough?," in *Software Maintenance and Reengineering, 2003. Proceedings. Seventh European Conference on*, 2003, pp. 13-15.

[42] M. Bruntink and A. Van Deursen, "Predicting class testability using object-oriented metrics," in *Source Code Analysis and Manipulation, 2004. Fourth IEEE International Workshop on*, 2004, pp. 136-145.

[43] D. Posnett, A. Hindle, and P. Devanbu, "A simpler model of software readability," in *Proceedings of the 8th Working Conference on Mining Software Repositories*, 2011, pp. 73-82.

[44] R. P. Buse and W. R. Weimer, "Learning a metric for code readability," *Software Engineering, IEEE Transactions on,* vol. 36, pp. 546-558, 2010.

[45] A. B. Al-Badareen, Z. Muda, M. A. Jabar, J. Din, and S. Turaev, "Software quality evaluation through maintenance processes," in *NAUN Conference of CONTROL*, 2010.

[46] M. Ilyas, M. Abbas, and K. Saleem, "A Metric Based Approach to Extract, Store and Deploy Software Reusable Components Effectively," *IJCSI International Journal of Computer Science Issues,* vol. 10, pp. 257-264, 2013.

[47] L. Antonelli and A. Oliveros, "Fuentes Utilizadas por desarrolladores

de Software en Argentina para Elicitar Requerimientos," in *WER*, 2002, pp. 106-116.
[48] R. L. Antonelli and A. Oliveros, "Traceability en la Etapa de Elicitación de Requerimientos," in *II Workshop de Investigadores en Ciencias de la Computación*, 2000.
[49] O. C. Gotel and A. C. Finkelstein, "An analysis of the requirements traceability problem," in *Requirements Engineering, 1994., Proceedings of the First International Conference on*, 1994, pp. 94-101.
[50] R. Brcina, S. Bode, and M. Riebisch, "Optimisation process for maintaining evolvability during software evolution," in *Engineering of Computer Based Systems, 2009. ECBS 2009. 16th Annual IEEE International Conference and Workshop on the*, 2009, pp. 196-205.
[51] (22-03-2014). *AQC Lab: Laboratorio de Evaluación de la Calidad Software*. Available: http://www.alarcosqualitycenter.com/index.php/laboratorio-evaluacion-calidad-software.

Jennifer Erazo Martínez nació en Popayán, Cauca, Colombia. Es Ingeniera de Sistemas de la Universidad del Cauca.

Andrés Flórez Gómez nació en Popayán, Cauca, Colombia. Es Ingeniero de Sistemas de la Universidad del Cauca.

Francisco J. Pino, Nació en Almaguer, Cauca, Colombia. Es ingeniero en Electrónica y Telecomunicaciones de la Universidad del Cauca. Obtuvo su título de especialista en Redes y Servicios Telemáticos también en la Universidad del Cauca. Obtuvo su doctorado en Tecnologías Informáticas Avanzadas en la Universidad Castilla-La Mancha, de Ciudad Real, España. Actualmente se desempeña como profesor de planta en la facultad de Ingeniería Electrónica y Telecomunicaciones de la Universidad del Cauca. Pertenece grupo al grupo de investigación IDIS y su área de investigación es mejora de procesos de software en pequeñas organizaciones y métodos de investigación cualitativos para la ingeniería de software.