Mantenibilidad del Software. Consideraciones para su especificación y validación

Software Maintainability. Considerations for its specification and validation

Jenny Adones Farfán¹ Vianca Vega-Zepeda^{1*}

Recibido 15 de julio de 2020, aceptado 1 de septiembre de 2020

Received: July 15, 2020 Accepted: September 1, 2020

RESUMEN

En la actualidad todas las organizaciones dan soporte a sus procesos de negocio incorporando productos de software. Dichos productos, a lo largo de su vida útil, deben ser modificados por diferentes motivos. Mientras el producto esté en operación, requerirá que se realice mantenimiento. Dada la criticidad de la característica de mantenibilidad de los productos, es relevante que cada vez que se especifican los requisitos de un nuevo software, quede claramente establecido qué hacer para asegurar que el mismo pueda ser mantenido en el transcurso de su vida útil.

Este artículo presenta una revisión bibliográfica, cuyo objetivo es presentar algunas consideraciones importantes sobre el proceso de mantenimiento, que se deben tener en cuenta a la hora de especificar el atributo de mantenibilidad, de tal forma que, posteriormente se pueda validar que efectivamente el producto es mantenible. Para esto se presentan algunos desafíos del proceso de mantenimiento, métricas asociadas a la mantenibilidad de los productos y algunas recomendaciones para conseguir su correcta especificación.

La principal conclusión obtenida, es que las organizaciones debieran recolectar a través del tiempo, datos históricos relacionados con los ajustes y cambios realizados en el producto, que permitan a futuro evaluar la conveniencia de continuar dando mantenimiento al producto, o simplemente reemplazarlo por uno nuevo. Además, dada la relevancia del atributo de mantenibilidad, en la especificación de requisitos, se deben incluir aspectos funcionales y de proceso que permitan de manera objetiva validar posteriormente si un producto es mantenible o no.

Palabras clave: Mantenibilidad del Software, requisitos no funcionales, especificación de requisitos, validación de requisitos no funcionales.

ABSTRACT

Currently, all organizations support their business processes by incorporating software products. These products over time must be modified for different reasons. While the product is in operation, it will require maintenance. Given the criticality of the maintainability of the products, it is very important that each time the requirements of a new software are specified, it is clearly established what to do to ensure that it can be maintained over time.

This article presents an investigation whose objective is to present some important considerations of the maintenance process, which must be taken into account when specifying the maintainability attribute, in such a way that later it can be validated that the product is actually maintainable. For this, some

¹ Universidad Católica del Norte. Departamento de Ingeniería de Sistemas y Computación. Antofagasta, Chile.
E-mail: jadones01@ucn.cl; vvega@ucn.cl

* Autor de correspondencia: vvega@ucn.cl

challenges of the maintenance process, metrics associated with the maintainability of the products and some recommendations to get its correct specification are presented.

The main conclusion obtained is that organizations should collect over time, historical data related to the adjustments and changes made in the product. In addition, given the relevance of the maintainability attribute, in the specification of requirements should include functional and process aspects that allow objectively validate later if a product is maintainable or not

Keywords: Software Maintainability, non-functional requirements, requirements specification, non-functional requirements validation.

INTRODUCCIÓN

La amplia gama de productos de software disponibles en la actualidad, que resuelven problemas específicos de las organizaciones, denota la necesidad de las empresas de automatizar y optimizar sus procesos de negocios. La alta competencia y los mercados globales obligan a las empresas a estar en un constante cambio. El uso de las tecnologías de información trae consigo amplios beneficios que requieren de inversión continua. Es aquí donde los sistemas de información, se han vuelto fundamentales para el negocio. En este contexto la mantención de software debe ser vista como una ventaja competitiva que permita a los sistemas de información evolucionar al ritmo que cambian los objetivos del negocio. Esta adaptación es necesaria para que las empresas sigan siendo competitivas en los mercados globales [1].

La necesidad de utilizar un sistema de información por largo tiempo, conlleva a la decisión de someter o no el producto de software a mantenimiento. Este problema tiene varios puntos de vista, los que pueden ser analizados de manera individual. Por un lado, se encuentran los intereses de la empresa que desarrolló el producto de software, en otra perspectiva está la empresa que hace uso del software y por último, se encuentra el punto de vista de la empresa que realiza la mantención del software. Cada uno de estos actores es responsable de asegurar la continuidad del funcionamiento de un producto de software.

La característica de mantenibilidad representa la capacidad del producto software para ser modificado efectiva y eficientemente, debido a necesidades evolutivas, correctivas o perfectivas. Esta característica se subdivide a su vez en las siguientes subcaracterísticas [1]:

- **Modularidad.** Capacidad de un sistema o programa de ordenador (compuesto de

componentes discretos) que permite que un cambio en un componente tenga un impacto mínimo en los demás.

- **Reusabilidad.** Capacidad de un activo que permite que sea utilizado en más de un sistema software o en la construcción de otros activos.
- **Capacidad de ser analizado.** Facilidad con la que se puede evaluar el impacto de un determinado cambio sobre el resto del software, diagnosticar las deficiencias o causas de fallos en el software, o identificar las partes a modificar.
- **Capacidad para ser modificado.** Capacidad del producto que permite que sea modificado de forma efectiva y eficiente sin introducir defectos o degradar el desempeño.
- **Capacidad para ser probado.** Facilidad con la que se pueden establecer criterios de prueba para un sistema o componente y con la que se pueden llevar a cabo las pruebas para determinar si se cumplen dichos criterios.

Mientras se usan los sistemas, surgen nuevos requisitos y es importante mantener su utilidad al cambiarlo para acomodar las nuevas características. El software mantenible es aquel que económicamente se adapta para lidiar con los nuevos requisitos, y donde existe una baja probabilidad de que los cambios insertarán nuevos errores en el sistema [2].

En definitiva, la “Mantenibilidad” (*Maintainability*) es la facilidad con la que se modifica, mejora y/o adapta un producto software. Esta característica es identificada y definida por normas de calidad ampliamente aceptadas, que recomiendan establecer métricas para su evaluación [3].

Un Atributo de Calidad es una propiedad medible o evaluable de un sistema que se utiliza para indicar cuán bien el sistema satisface las necesidades de sus *stakeholders*. Se puede pensar de un atributo

de calidad como una medida de las “bondades” de un producto junto a algunas dimensiones de interés para un *stakeholder* [4].

La mantenibilidad es considerada un atributo de calidad por lo cual a su vez se distingue como un requisito no funcional del sistema. Y como tal, debe ser especificado al inicio de cada proyecto de desarrollo de software, cuando se define el alcance del producto a desarrollar.

La especificación de los requisitos no funcionales siempre debe ser objetiva, de manera tal que permita la posterior validación del cumplimiento del atributo especificado [5]. Es por esto que es necesario un entendimiento acabado de cómo se puede medir objetivamente si un producto de software es mantenible o no.

En este contexto, la investigación reportada en este artículo, busca responder las siguientes preguntas de investigación:

1. ¿Cómo se puede cuantificar de manera objetiva la mantenibilidad de un producto de software?
2. ¿Qué atributos o componentes se pueden utilizar para evaluar qué tan mantenible es un producto de software?
3. ¿Qué evidencias se pueden presentar para asegurar que un producto sí es mantenible?

El artículo se organiza comenzando con una visión global de los desafíos que plantea desarrollar la etapa de mantenimiento de software. Luego, presenta una revisión del estado del arte relacionada con las métricas de mantenibilidad. Continúa con un breve análisis de las recomendaciones que diversos autores entregan para la especificación de requisitos, con énfasis en los requisitos no funcionales. Finalmente, en las conclusiones se presentan los hallazgos y consideraciones que se deben tener presente cuando se desea especificar el requisito de mantenibilidad de un producto de software.

DESAFÍOS DE LA ETAPA DE MANTENIBILIDAD

Se tratarán en esta sección algunos de los desafíos que presenta la etapa de mantenimiento, los que han sido identificados a partir de una revisión de la literatura. El foco de esta revisión fue la búsqueda de propuestas de técnicas, modelos o procesos de

mantenimiento de software. También se incluyó una revisión de los atributos no funcionales, específicamente el atributo de mantenibilidad y sus métricas asociadas. Cabe destacar que muchas de las investigaciones nombran al mantenimiento como parte del ciclo de vida del software y las dificultades que este tiene, pero no se enfocan en resolver o tratar aspectos específicos de este proceso [6-7, 16, 8-15].

El primer desafío y por ende el más importante es que se debe partir de la base que el software de calidad requiere contar con la capacidad de evolucionar, es decir, el software debe contar con el atributo, “*evolvability*”, como se conoce en el idioma inglés, que determina la capacidad que este posee de acomodarse fácilmente a futuros cambios, por lo cual se hace necesario abordar la evolución de forma explícita durante todo el proceso de desarrollo situación que no siempre es considerada [17].

Como se ha mencionado anteriormente el mantenimiento en el software es gatillado por cambios en los factores de su entorno ya sea por el hardware que lo soporta, la conectividad a la que debe estar sometido, a los nuevos requisitos del cliente, a los lenguajes de programación utilizados, por nombrar algunos.

La no existencia de documentación asociada al sistema de información se torna un problema para el equipo de mantenedores cuyo trabajo es más complejo porque deben dedicar tiempo y esfuerzos a comprender el software que se presenta como una caja negra a ser descifrada a través de su código fuente [18-19].

Gestionar la arquitectura de software después de la fase de implementación es una tarea muy compleja debido a los frecuentes cambios en los requisitos y el entorno del software [20]. Existe el marco de trabajo SHEPhERd (*Software arcHitecture Evolution*) cuyo objetivo es minimizar los costos mientras se mantiene la confiabilidad y el rendimiento de la arquitectura de software [21].

Para mantenerse al día con las crecientes necesidades y deseos de los usuarios, es necesario hacer de la evolución una tarea planificada durante todo el ciclo de vida del software [22]. En la idea de anticiparse a los cambios futuros que tendrá el software es que se

plantea el uso de Campos Aleatorios Condicionales (CRF) como base matemática para proporcionar una exploración cuantitativa de los requisitos emergentes de los usuarios. La tendencia debe ser realizar un proceso de evolución de software, basado en tales enfoques de detección de intención, que permitan automatizar el mantenimiento preventivo.

El tiempo prolongado de permanencia del software en la empresa conlleva a que los datos históricos y los expertos pueden ya no estar presentes o ser imprecisos al momento de requerir el mantenimiento, por lo cual, la estimación de los esfuerzos en los que se incurrirá durante el mantenimiento es una tarea ardua que en [23] se busca resolver con la propuesta de un Modelo de Estimación de Esfuerzo de Mantenimiento de software (SMEEM), modelo que es aplicable solo para entornos de mantenimiento basados en programación ágil y extrema [24]. La estimación de costos y esfuerzos continúa hasta la fecha siendo una tarea problemática [25-30].

La forma en cómo se distribuye el equipo de trabajo que realiza el mantenimiento de software presenta también sus desafíos. Es así como la implementación utilizando código abierto [31] donde desarrolladores se encuentran separados geográficamente ha llevado al análisis de los nuevos ecosistemas de desarrollo [32-33] que fomentan la colaboración, pero agregan nuevos desafíos al proyecto [34]. En [35] se desarrolló una plataforma de intercambio de conocimiento colaborativo llamada CollabDev. El objetivo de esta herramienta es analizar las aplicaciones en múltiples idiomas y ofrecer diversas perspectivas estructurales, arquitectónicas y funcionales a las personas involucradas en el mantenimiento. En este mismo contexto, el Desarrollo Global de Software (DGS) engloba las acciones de cambio que se deben realizar para el desarrollo y por ende para el mantenimiento de software. En este tipo de desarrollo, se habla de las 3C que representan los desafíos en la comunicación, coordinación y control que presentan los proyectos con personal distribuido geográficamente, mismos desafíos que deben sortear los mantenedores al enfrentar un proyecto de mantenimiento. Estos desafíos en el DGS se acentúan debido a lo que en la literatura se denomina las 3 distancias: Distancia geográfica, Distancia Temporal y Distancia Socio-Cultural, aspectos que se pueden trasladar a las diferencias entre los equipos mantenedores y los equipos desarrolladores [36].

La reutilización de software se presenta también como un punto a evaluar debido a que se debe fomentar que los conocimientos adquiridos durante un desarrollo sean registrados dentro de la empresa, práctica que permite alcanzar el objetivo de entregar software de calidad a tiempo y dentro del presupuesto asignado y por ende facilita la etapa de mantenimiento [37-38].

La integración, entrega e implementación continua permite a las organizaciones liberar de manera frecuente y confiable nuevas características y productos de software. Estas prácticas de desarrollo también han sido aplicadas al mantenimiento [39].

En la etapa de mantenimiento se deben considerar una serie de factores que deben ser estudiados. Estos se pueden clasificar en factores internos como tamaño del sistema, madurez del sistema, número de usuario finales, experiencia del equipo del proyecto [40], metodología de desarrollo [41-2], tecnología, número de solicitudes de cambio, evaluación de los nuevos requisitos [43], además de factores externos como la actitud del cliente, el clima organizacional [44], la gestión del conocimiento [45-46] y los entornos de gestión de proyectos [47].

Las empresas desarrolladoras de software deben ser capaces de asegurar la calidad del producto de software mantenido la cual debe ser medida y evaluada [48-49].

MÉTRICAS DE MANTENIBILIDAD

Para responder la pregunta ¿Cómo se puede cuantificar de manera objetiva la mantenibilidad de un producto de software? Es necesario revisar qué métricas se han propuesto para medir qué tan mantenible es un producto de software.

Las métricas de proceso son atributos del proceso, como el tiempo que tarda en completarse una tarea. Las métricas de producto son atributos del software en sí, como el tamaño o la complejidad. Una métrica de software es una característica de un sistema de software, documentación de sistema o proceso de desarrollo que puede medirse de manera objetiva [1].

Es difícil hacer mediciones directas de muchos de los atributos de calidad del software.

La Figura 1, tomada de [1] muestra atributos externos de calidad que pueden ser medidos a través de la relación con atributos internos de calidad.

Las características del software que pueden medirse fácilmente, como el tamaño y la complejidad ciclométrica, no tienen una relación clara y consistente con los atributos de calidad como comprensibilidad y mantenibilidad. Las relaciones varían dependiendo de los procesos de desarrollo, la tecnología empleada y el tipo de sistema a diseñar [1].

Ahora bien, la evaluación de la calidad de un producto de software debe ser realizada a través de métricas del producto que se dividen en dos clases:

1. **Métricas dinámicas**, que se recopilan mediante mediciones hechas de un programa en ejecución. Dichas métricas pueden recopilarse durante las pruebas del sistema o después de que el sistema está en uso. Un ejemplo es el número de reportes de *bugs* o el tiempo necesario para completar un cálculo.
2. **Métricas estáticas**, las cuales se recopilan mediante mediciones hechas de representaciones del sistema, como el diseño, el programa o la documentación. Ejemplos de mediciones estáticas son el tamaño del código y la longitud promedio de los identificadores que se usaron.

Estos tipos de métrica se relacionan con diferentes atributos de calidad. Las métricas dinámicas ayudan a valorar la eficiencia y fiabilidad de un programa. Las

métricas estáticas ayudan a valorar la complejidad, comprensibilidad y mantenibilidad de un sistema de software o de los componentes del sistema.

Las métricas de la Figura 2 son aplicables a cualquier sistema [1].

El mantenimiento de software es considerado la etapa más costosa del ciclo de vida de desarrollo de software. Los estudios concuerdan ampliamente en que el mantenimiento del software toma una proporción más alta de presupuestos de TI que un nuevo desarrollo (casi dos tercios en mantenimiento y un tercio en desarrollo) [1, 4, 50].

Las buenas técnicas de ingeniería de software, como la especificación precisa, el uso de desarrollo orientado a objetos y la administración de la configuración, contribuyen a reducir los costos de mantenimiento [1].

Este estudio busca indagar en las diferentes características del software que indiquen si este es mantenible de manera cuantitativa, por lo cual se han recopilado métricas destinadas a responder las preguntas de investigación cuyo objetivo es identificar de manera objetiva y numérica si un software es mantenible o no.

Se mencionan a continuación métricas de proceso y de producto que guían las respuestas solicitadas en la investigación.

Fuente: Sommerville [1].

Figura 1. Relaciones entre atributos de software internos y externos.

Fuente: Sommerville [1].

Métrica de software	Descripción
Fan-in/Fan-out	Fan-in (abanico de entrada) es una medida del número de funciones o métodos que llaman a otra función o método (por ejemplo, X). Fan-out (abanico de salida) es el número de funciones a las que llama la función X. Un valor alto para fan-in significa que X está estrechamente acoplado con el resto del diseño y que los cambios a X tendrán extensos efectos dominó. Un valor alto de fan-out sugiere que la complejidad global de X puede ser alta debido a la complejidad de la lógica de control necesaria para coordinar los componentes llamados.
Longitud de código	Ésta es una medida del tamaño de un programa. Por lo general, cuanto más grande sea el tamaño del código de un componente, más probable será que el componente sea complejo y propicie a errores. Se ha demostrado que la longitud del código es una de las métricas más fiables para predecir la proclividad al error en los componentes.
Complejidad ciclomática	Ésta es una medida de la complejidad del control de un programa. Tal complejidad del control puede relacionarse con la comprensibilidad del programa. En el capítulo 8 se estudia la complejidad ciclomática.
Longitud de identificadores	Ésta es una medida de la longitud promedio de los identificadores (nombres para variables, clases, métodos, etcétera) en un programa. Cuanto más largos sean los identificadores, es más probable que sean significativos y, por ende, más comprensible será el programa.
Profundidad de anidado condicional	Ésta es una medida de la profundidad de anidado de los enunciados if en un programa. Los enunciados if profundamente anidados son difíciles de entender y propician potencialmente a errores.
Índice Fog	Ésta es una medida de la longitud promedio de las palabras y oraciones en los documentos. Cuanto más alto sea el valor del índice Fog de un documento, más difícil será entender el documento.

Figura 2. Métricas estáticas de productos de software.

Sommerville [1] identifica algunas métricas de proceso que sirven para valorar la mantenibilidad:

1. **Número de peticiones para mantenimiento correctivo.** Un aumento en el número de reportes de *bugs* y fallas indicaría que se introdujeron más errores en el programa de los que se repararon durante el proceso de mantenimiento. Esto podría revelar un declive en la mantenibilidad.
2. **Tiempo promedio requerido para el análisis del impacto.** Refleja el número de componentes de programa que se ven afectados por la petición de cambio. Si este tiempo aumenta, implica que más componentes resultaron afectados y que la mantenibilidad decrece.
3. **Tiempo promedio tomado para implementar una petición de cambio.** Éste no es el mismo que el tiempo para el análisis del impacto, aunque puede correlacionarse con él, sino más bien es la cantidad de tiempo que se necesita para modificar el sistema y su documentación, después de valorar cuáles componentes serán

afectados. Un aumento en el tiempo necesario para implementar un cambio puede indicar un declive en la mantenibilidad.

4. **Número de peticiones de cambio pendientes.** Con el tiempo, un aumento en este número implicaría un declive en la mantenibilidad.

Cabe mencionar que para la obtención de los indicadores anteriormente mencionados se requiere del registro sistemático de solicitudes de cambios, tiempos de desarrollo, etc. Es decir, para poder formular los indicadores y obtener valores correctos se debe incorporar en la cultura organizacional el registro de los datos necesarios para responder a los indicadores.

Con mayor frecuencia la complejidad de mantenimiento se hace evidente en sistemas heredados, la calidad de los mismos disminuye a medida que pasa el tiempo y se requiere de adaptaciones del sistema a los cambios en los modelos de negocio de las instituciones.

Para determinar si un producto de software debe ser sometido a mantenimiento o no, pueden ser evaluados a través de [1]:

1. **El número de peticiones de cambio del sistema.** Los cambios al sistema, por lo general, corrompen la estructura del sistema y dificultan cambios futuros. Cuanto más alto sea este valor acumulado, más baja será la calidad del sistema.
2. **El número de interfaces de usuario.** Éste es un factor importante en los sistemas basados en formas, donde cada forma puede considerarse como una interfaz de usuario separada. Cuantas más interfaces haya, más probabilidad habrá de que existan inconsistencias y redundancias en dichas interfaces.
3. **El volumen de datos usados por el sistema.** Cuanto más alto sea el volumen de datos (número de archivos, tamaño de base de datos, etcétera), más probable será que haya inconsistencias de datos que reduzcan la calidad del sistema.

Se plantea en este estudio la evidencia de que las técnicas de desarrollo de software que son utilizadas, afectan directamente a la etapa de mantenimiento aportando a disminuir o aumentar el atributo de mantenibilidad del sistema [1, 4].

Enfocados en la técnica de desarrollo Orientado a Objetos se definen métricas más específicas. Basadas en este paradigma de programación, se puede cuantificar cuán mantenible o no es un sistema. Algunas de estas métricas se definen como se indica a continuación [1].

1. **Número de hijos** (*number of children*, NOC). Ésta es una medida del número de subclases inmediatas en una clase. Mide la amplitud de una jerarquía de clase, mientras que DIT mide su profundidad. Un valor alto de NOC puede indicar mayor reutilización.
2. **Respuesta por clase** (*response for a class*, RFC). RFC es una medida del número de métodos que potencialmente podrían ejecutarse en respuesta a un mensaje recibido por un objeto de dicha clase. RFC se relaciona con la complejidad. Cuanto más alto sea el valor para RFC, más compleja será una clase y, por ende, es más probable que incluya errores.
3. **Falta de cohesión en métodos** (*lack of cohesion in methods*, LCOM). LCOM se calcula al considerar pares de métodos en una clase. LCOM es la diferencia entre el número de pares

de método sin compartir atributos y el número de pares de método con atributos compartidos.

Otras métricas utilizadas en el desarrollo de software OO son las presentadas por Acosta [51]. A continuación, se mencionan.

Métricas MOOD (*Metrics for Object oriented Design*)

1. MHF (*Method Hiding Factor*) - Proporción de métodos ocultos: Es la proporción de la suma de los métodos privados en todas las clases respecto al número total de métodos definidos en el sistema. Se propone como una medida de encapsulamiento y cantidad relativa de información oculta.
2. AHF (*Attribute Hiding Factor*) - Proporción de atributos ocultos: De manera similar, representa la proporción de la suma de las invisibilidades de los atributos en todas las clases respecto al número total de atributos definidos en el sistema. Se propone como una medida de encapsulamiento.
3. MIF (*Method Inheritance Factor*) - Proporción de métodos heredados: Representa la proporción de la suma de todos los métodos heredados en todas las clases respecto al número total de métodos (localmente definidos más los heredados) en todas las clases. Esta métrica da indicios del nivel de rehuso, pero a la vez puede señalar una disminución en la comprensibilidad.
4. AIF (*Attribute Inheritance Factor*) - Proporción de atributos heredados: Señala la proporción del número de atributos heredados respecto al total de atributos. Tiene idéntica interpretación que MIF.

Métricas a nivel de acoplamiento

1. CBO (*Coupling Between Objects*) - Acoplamiento entre Objetos: Para una clase determinada indica el número de clases a las cuales está ligada. Existe dependencia entre dos clases cuando una de ellas usa métodos o variables de la otra. Para el cálculo de esta métrica no se consideran las clases relacionadas por herencia.

Métricas a Nivel de Clases

1. DIT (*Depth of Inheritance Tree*) - Profundidad en árbol de herencia: Mide la distancia entre un nodo y la raíz en una jerarquía de herencia. En el nivel cero de la jerarquía se encuentra

la clase raíz. Esta métrica permite predecir la complejidad de una clase y el potencial de rehúso. Altos niveles de herencia indican objetos complejos y bajos niveles implican código escrito de manera funcional.

2. **WMC (Weighted Methods per Class)** - Métodos ponderados por clase: Mide la complejidad de una clase. El número de métodos y su complejidad es un indicador razonable de la cantidad de esfuerzo necesaria para implementar y comprobar una clase. Además, cuanto mayor sea el número de métodos, más complejo será el árbol de herencia.

Según la Norma 25010, las características de calidad de un producto de software se clasifican como se muestran en la Figura 3. La mantenibilidad está incluida entre las ocho características principales [52].

Según Ruiz [53], existen unos pocos factores que afectan directamente la mantenibilidad de un producto. Los tres más significativos son:

- **Proceso de desarrollo:** la mantenibilidad debe formar parte integral del proceso de desarrollo del software y ser uno de sus principios rectores.
- **Documentación:** En múltiples ocasiones, ni la documentación ni las especificaciones de diseño están disponibles y, por tanto, los tiempos y costos de comprensión, corrección y/o modificación se incrementan.
- **Comprensión de Programas:** La causa básica de los altos costos del mantenimiento es la

presencia de obstáculos a la comprensión del funcionamiento del producto.

Como resultado se ha obtenido la Figura 4 que resume el conjunto de métricas internas básicas que se encuentran relacionadas con las sub-características de capacidad de ser analizado (analizabilidad), capacidad de ser modificado (cambiabilidad), estabilidad y capacidad de ser probado, las cuales definen a la característica de mantenibilidad de acuerdo con la norma ISO/IEC 9126 [50].

ESPECIFICACIÓN DEL REQUISITO DE MANTENIBILIDAD

La Ingeniería de Requisitos es un proceso que abarca un conjunto de actividades que se realizan comúnmente al inicio del ciclo de desarrollo de software. La elicitación de los requisitos, que tiene por objetivo descubrir las necesidades de los usuarios y capturar los requisitos del sistema, es la etapa más importante de este proceso y su correcta aplicación repercute en gran medida en el éxito del producto final [54].

Dada la importancia de la correcta aplicación del proceso de Ingeniería de requerimientos, diversos modelos y propuestas metodológicas entregan recomendaciones y plantillas para la especificación de requisitos de manera consistente. En dichas plantillas, se incorpora la especificación de los requisitos no funcionales, en donde corresponde especificar, entre otros atributos, el requisito

Fuente: Rodríguez y Fernández [52].

Figura 3. Modelo de calidad del producto software según la ISO/IEC 25010.

Fuente: Irrazábal y Garzás [50].

Métricas de calidad	Analizabilidad	Cambiabilidad	Estabilidad	Capacidad de ser probado
Complejidad ciclomática (CC)	X	X		X
Cantidad de las instrucciones (LOC)	X			
Promedio del tamaño de las instrucciones (PLOC)	X	X		X
Frecuencia de los comentarios (PCOM)	X			
Peso de los métodos por clase (WMC)	X			
Número de clases base (NCB)	X			
Número de saltos incondicionales (NSI)		X	X	X
Número de niveles anidados (NNA)		X		X
Acoplamiento entre objetos (CBO)	X	X	X	X
Ausencia de cohesión (LCOM)	X	X	X	X
Profundidad del árbol de herencia (DIT)	X	X	X	X
Componentes llamados directamente (CCD)			X	
Número de hijos (NOC)	X	X	X	X
Número de salidas de estructuras condicionales (NSE)				X
Respuestas de una clase (RFC)				X
Número de mensajes enviados en un método (NOM)	X	X	X	X
Cobertura de las pruebas unitarias (COB)			X	X
Número de errores de las pruebas unitarias (UTE)			X	X
Violaciones en el código fuente (VCF)	X			
Violaciones de estilo (VST)	X			
Distancia a la secuencia principal (D)		X	X	
Número de dependencias cíclicas (CYC)	X	X		
Acoplamiento aferente y eferente (CAF, CEF)		X	X	
Código duplicado (CDU)	X	X		

Figura 4. Métricas internas relacionadas con las subcaracterísticas de la mantenibilidad de acuerdo a la norma ISO/IEC 9126.

de mantenibilidad. A continuación, a modo de ejemplo, se mencionan algunas de estas plantillas propuestas.

- La norma IEEE 830 [55], provee una plantilla para la especificación de requisitos, en donde se considera incorporar los requisitos no funcionales, los cuales deben ser escritos de manera cuantitativa o en términos de requisitos funcionales. Además, en relación a la evolución del sistema, indica que se debe describir los principios fundamentales en los cuales se basa el sistema, cualquier cambio anticipado debido a la evolución del hardware, las necesidades cambiantes del usuario, etc. Esta sección es útil para los diseñadores de sistemas, ya que pueden ayudarlos a evitar decisiones de diseño que limitarían posibles cambios futuros en el sistema.
- La Norma ISO 29110 [56], en su paquete de despliegue Análisis de requerimientos de software, incluye como activos del proceso tres plantillas alternativas para la especificación de requisitos. Una de ellas es una adaptación de la norma IEEE 830 donde se mantiene una sección para la especificación de los requisitos no funcionales. La segunda alternativa es la plantilla Construx (www.construx.com) cuya sección 3.5.4 solicita explícitamente la especificación del atributo de mantenibilidad. La tercera alternativa corresponde a la plantilla VOLERE, donde también se incluye una sección específica y explícita para la especificación del atributo de mantenibilidad.

Como se puede observar, existe una clara conciencia de la importancia de especificar el requisito de

mantenibilidad. Sin embargo, no basta con saber que se debe hacer. Es aún más relevante saber cómo hacerlo. Es por esto, que a partir de los desafíos recolectados desde la literatura y las métricas expuestas anteriormente, a continuación se plantean una serie de consideraciones y recomendaciones a tener en cuenta a la hora de especificar el requisito de mantenibilidad:

1. La mantenibilidad es una propiedad emergente de los sistemas, es decir, es un requisito que no puede ser direccionado mediante una única definición o especificación, sino que depende de cómo interactúan distintos componentes.
2. Es muy importante que los atributos de calidad - entre ellos la mantenibilidad - sean especificados de manera objetiva y verificable. Esto significa que debe existir una manera factible de comprobar que efectivamente se cumplió con la especificación.
3. Se recomienda que la mantenibilidad sea especificada mediante aspectos asociados al proceso que se debe seguir durante la implementación de un nuevo producto de software. Esto es para asegurar que mientras se está creando un nuevo producto, se incorporen todos los elementos necesarios para que el producto sea posteriormente mantenible.
4. De forma más concreta, se recomienda explicitar el uso de una arquitectura de software basada en capas. También el uso de un estándar de codificación en donde se incluyan todos aquellos elementos y prácticas que se utilizan en las métricas vistas. Por ejemplo, aquellos elementos planteados en la Figura 4.
5. Se recomienda la creación y uso de una *checklist* para la etapa de validación de requisitos y revisión del documento de especificación.
6. Se recomienda que, durante el tiempo de vida útil del software, se registre formalmente la información de los cambios realizados, de tal manera de contar con estadísticas que permitan a futuro determinar la calidad del producto, en relación a su atributo de mantenibilidad.

CONCLUSIONES

Como se indicó al inicio del documento, este artículo presenta la investigación realizada para responder tres preguntas asociadas a la mantenibilidad del software. Además, en base a una revisión de la literatura se identificaron los desafíos pendientes

y las métricas existentes para la mantenibilidad del software, con el objetivo de explicitar las consideraciones y recomendaciones necesarias a la hora de especificar el requisito de mantenibilidad.

Como se ha visto durante las secciones previas, existen métricas que permiten cuantificar las características de calidad de un sistema de software. El atributo de mantenibilidad está clasificado con un atributo no funcional, cuya medición debe ser asociada a la medición de algunos atributos funcionales.

Como se indicó previamente, la investigación reportada busca responder tres preguntas de investigación, las cuales a continuación son desarrolladas.

¿Cómo se puede cuantificar de manera objetiva la mantenibilidad de un producto de software?

La evaluación de la calidad de un producto de software debe ser comprobada de manera cuantitativa es por esto que tanto los requisitos funcionales como no funcionales deben tener métricas asociadas que indique el resultado en términos valóricos para determinar una apreciación objetiva del cumplimiento del requisito.

Abordando las métricas internas relacionadas al código fuente y al paradigma de programación utilizado, como en el desarrollo orientado a objetos, se puede cuantificar la mantenibilidad realizando mediciones específicas al código fuente del sistema a través de las métricas del producto, la codificación debe ser realizada con altos estándares y técnicas que ayuden a futuro tanto a la comprensión del sistema como a la modificación del mismo. Para esto, las organizaciones desarrolladoras de software deben incorporar entre sus herramientas de trabajo, algunas que gestionen las características del código, que permitan realizar el cálculo de las métricas definidas y determinar cuán mantenible es el producto en un instante cualquiera.

Si la documentación existente es de calidad (si se encuentra actualizada) aporta directamente a las medidas de analizabilidad y comprensibilidad del producto de software. Es por esto que es de suma relevancia, que cada vez que se realiza el mantenimiento, también se debe mantener la documentación del producto.

El proceso de mantenimiento debe ser documentado, en particular con datos estadísticos asociados a

las métricas de proceso relacionadas al tiempo y cantidad de intervenciones realizadas al producto de software durante su vida útil, en este punto es importante definir el registro de los datos como parte obligatoria del proceso de mantenimiento, para que estas estadísticas se transformen en medidas objetivas de determinación del atributo de mantenibilidad.

¿Qué atributos o componentes se pueden utilizar para evaluar qué tan mantenible es un producto de software?

El análisis de la mantenibilidad está relacionado tanto al proceso de desarrollo y mantenimiento como al producto de software.

En relación al proceso, se deben utilizar las estadísticas de vida del sistema (sistemas legados), solicitudes de cambios, solicitudes de correcciones, tiempos de realización de análisis de los cambios como de la implementación de los mismos.

En relación al producto, el análisis de la estructura del código fuente, permite determinar de acuerdo a la técnica de programación utilizada, por ejemplo, la programación orientada a objeto, qué tan mantenible es el sistema, a través del análisis de las estadísticas de acoplamiento entre objetos, herencia, profundidad en el árbol de herencia y métodos heredados por mencionar algunos.

Si bien es cierto, según lo indicado, es posible evaluar qué tan mantenible es un producto, la complejidad radica en que muy pocas organizaciones cuentan con los datos históricos que se requieren.

Como se mencionó previamente, la documentación asociada al software, su calidad y vigencia de la misma generan la base desde donde se puede iniciar la evaluación del producto.

¿Qué evidencias se pueden presentar para asegurar que un producto sí es mantenible?

Se ha logrado identificar un conjunto de elementos que podrían ser utilizados:

- Estadísticas de las solicitudes de cambio, ya sea para mantenimiento correctivo, perfectivo o adaptativo. Antes y después de la realización de alguna intervención de mantenimiento en el sistema.
- Valores de las métricas relacionadas al análisis del código fuente, métricas de acoplamiento, herencia etc.

- El tiempo de vida de un sistema legado, cantidad de módulos (tamaño del sistema), estructura del código fuente son algunas de las estadísticas que permiten evidenciar si un sistema es mantenible o no.

Al igual que en la pregunta previa, la complejidad aquí radica en que las organizaciones no acostumbran mantener el registro estadístico de los cambios realizados durante el tiempo de vida de un producto.

Finalmente, la principal conclusión a la que se ha llegado luego de esta investigación, es que, dada la criticidad del atributo de mantenibilidad, durante la etapa de Ingeniería de Requisitos, a la hora de especificar los requisitos, es necesario utilizar algún mecanismo que incorpore la definición de algunas prácticas que deben ser incorporadas de manera obligatoria en todo sistema, que asegure que el sistema será efectivamente mantenible a través del tiempo.

Las intervenciones sobre el producto de software deben ser controladas y evaluadas de acuerdo a las métricas de mantenibilidad mencionadas. Cada vez que se realiza un cambio se debe verificar si este introdujo o no errores como efectos colaterales a las modificaciones.

Junto a lo anterior, se debe contar con una guía que recomiende a las organizaciones la recolección de datos históricos y su gestión estadística para poder identificar si el producto sigue siendo mantenible a través del tiempo. La incorporación de más acciones al proceso de mantenimiento, que muchas veces se realiza de manera urgente, repercutirá positivamente en una mejor toma de decisiones a la hora de determinar la continuidad del software o proceder a su retiro o migración. Dicha decisión será fundamentada y respaldada cuantitativamente, guiándose por las mejores prácticas necesarias para mantener la calidad del software hasta el final de su vida útil.

REFERENCIAS

- [1] I. Sommerville. "Software Engineering". 9th Ed. Addison-Wesley. 2011.
- [2] P. Alfonzo y S. Mariño, "Los estándares internacionales y su importancia para la industria del software". Técnica Adm. Vol. 1 Nº 2. 2013.

- [3] M. Piattini, M. Polo, F. Ruiz e I Fernández. “Mantenimiento del Software: Conceptos, métodos, herramientas y outsourcing”. Ra-ma. 1998.
- [4] M. Piattini y F. García. “Calidad en el desarrollo y mantenimiento del software”. Ra-ma. 2003.
- [5] P. Bourque and R. Fairly. “Guide to the software engineering body of knowledge version 3.0”. IEEE Computer society. 2014.
- [6] D. Albuquerque, B. Cafeo, A. Garcia, S. Barbosa, S. Abrahão y A. Ribeiro. “Quantifying usability of domain-specific languages: An empirical study on software maintenance”. *J. Syst. Softw.* Vol. 101, pp. 245-259. March, 2015.
- [7] J. Misra, S. Sengupta, D. Rawat, M. Savagaonkar and S. Podder. “Data-Driven Application Maintenance: Experience from the Trenches”. In *IEEE/ACM 4th International Workshop on Software Engineering Research and Industrial Practice (SER&IP)*, pp. 48-54. 2017.
- [8] S. Mani. “Improving enterprise software maintenance efficiency through mining software repositories in an industry context”. In *Companion Proceedings of the 36th International Conference on Software Engineering*, pp. 706-709. 2014.
- [9] P. Li and E. Wohlstadter. “Dynamic round-trip GUI maintenance”. In *ACM/IEEE 30th International Conference on Software Engineering*, pp. 851-854. 2008.
- [10] E. Juergens. “Software maintenance research that is empirically valid and useful in practice”. *It - Inf. Technol.* Vol. 58, Issue 3, pp. 145-149. 2016.
- [11] L. Seinturier and M. Van den Brand. “Preface to the special section on software evolution, adaptability, and maintenance”. *Sci. Comput. Program.*, 2013.
- [12] G. ArunKumar and R. Dillibabu. “Design and Application of New Quality Improvement Model: Kano Lean Six Sigma for Software Maintenance Project”. *Arab. J. Sci. Eng.* Vol. 41, Issue 3, pp. 997-1014. 2016.
- [13] C. Fitzgerald. “You Don’t Know Jack About”. *Commun. ACM.* Vol. 52, Issue 11, pp. 54-58. 2009.
- [14] M. Di Penta and J.I. Maletic. “Guest editorial: special section on software maintenance and evolution”. *Empir. Softw. Eng.* Vol. 20, Issue 2, pp. 410-412. 2015.
- [15] C.-J. Xiong, M. Xie and S.-H. Ng. “Optimal software maintenance policy considering unavailable time”. *J. Softw. Maint. Evol. Res. Pract.* Vol. 2011, Issue 23, pp. 21-33. 2010.
- [16] R. Capilla, J.C. Dueñas and R. Ferenc. “A retrospective view of software maintenance and reengineering research”. *J. Softw. Evol. Process.* Vol. 2013, Issue 25, pp. 569-574. 2013.
- [17] H.P. Breivold, I. Crnkovic and M. Larsson. “A systematic review of software architecture evolution research”. *Inf. Softw. Technol.* Vol. 54, Issue 1, pp. 16-40. January, 2012.
- [18] T. Arbuckle. “Studying software evolution using artefacts’ shared information content”. *Sci. Comput. Program.* Vol. 76, Issue 12, pp. 1078-1097. December, 2011.
- [19] J. Buckley. “Requirements-Based Visualization Tools for Software Maintenance and Evolution”. *IEEE Comput*, pp. 106-108. 2009.
- [20] N. Veerman. “Automated mass maintenance of a software portfolio”. *Sci. Comput. Program.* Vol. 62, Issue 3, pp. 287-317. October, 2006.
- [21] V. Cortellessa, R. Mirandola and P. Potena. “Managing the evolution of a software architecture at minimal cost under performance and reliability constraints”. *Sci. Comput. Program.* Vol. 98, pp. 439-463. February, 2015.
- [22] H. Xie, J. Yang, C.K. Chang and L. Liu. “A statistical analysis approach to predict user’s changing requirements for software service evolution”. *J. Syst. Softw.* Vol. 132, pp. 147-164. October. 2017.
- [23] J. Choudhari and U. Suman. “Story Points Based Effort Estimation Model for Software Maintenance”. *Procedia Technol.* Vol. 4, pp. 761-765. January, 2012.
- [24] J. Choudhari and U. Suman. “Extended iterative maintenance life cycle using eXtreme programming”. *ACM SIGSOFT Softw. Eng. Notes.* Vol. 39, Issue 1, pp. 1-12. 2014.
- [25] A. Hira and B. Boehm. “Using Software Non-Functional Assessment Process to Complement Function Points for Software Maintenance”. In *Proceedings of the 10th*

- ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, pp. 1-6. 2016.
- [26] R.D. Banker, S.M. Datar, C.F. Kemerer and D. Zweig. "Software complexity and maintenance costs". *Commun. ACM*. Vol. 36, Issue 11, pp. 81-94. 1993.
- [27] D. Hocking and J. Celko. "Software maintenance: A budgeting dilemma". In *Proceedings of the fifteenth SIGCSE technical symposium on Computer science education*, pp. 125-129. 1984.
- [28] J. Choudhari and U. Suman. "Phase wise effort estimation for software maintenance". In *Proceedings of the CUBE International Information Technology Conference*, pp. 397-402. 2012.
- [29] A. De Lucia, E. Pompella and S. Stefanucci. "Effort estimation for corrective software maintenance". In *Proceedings of the 14th international conference on Software engineering and knowledge engineering*, pp. 409-416. 2004.
- [30] V. Nguyen, B. Boehm and P. Danphitsanuphan. "A controlled experiment in assessing and estimating software maintenance tasks". *Inf. Softw. Technol.* Vol. 53, Issue 6, pp. 682-691. June, 2011.
- [31] K. Wei, K. Crowston, N.L. Li and R. Heckman. "Understanding group maintenance behavior in Free/Libre Open-Source Software projects: The case of Fire and Gaim". *Inf. Manag.* Vol. 51, Issue 3, pp. 297-309. April, 2014.
- [32] T. Mens. "Introduction to the special issue on software maintenance and evolution research". *Empir. Softw. Eng.* Vol. 20, Issue 5, pp. 1193-1197. 2015.
- [33] Y. Dittrich. "Software engineering beyond the project - Sustaining software ecosystems". *Inf. Softw. Technol.* Vol. 56, Issue 11, pp. 1436-1456. November, 2014.
- [34] K. Jaber, B. Sharif and C. Liu. "A Study on the Effect of Traceability Links in Software Maintenance". *IEEE Access*. Vol. 1, pp. 726-741. 2013.
- [35] S. Sarkar, R. Sindhgatta and K. Pooloth. "A collaborative platform for application knowledge management in software maintenance projects". In *ACM Compute*, p. 1. 2008.
- [36] A. Vizcaíno, M. Piattini and F. García. "Desarrollo Global de Software". Ra-ma. 2014.
- [37] V. Bauer and A. Vetro'. "Comparing reuse practices in two large software-producing companies". *J. Syst. Softw.* Vol. 117, pp. 545-582. July, 2016.
- [38] A. Verma, G.S. Kamboj and I. Kaur. "Genetic optimized software component reuse model for reducing development and maintenance efforts". *Int. J. Adv. Appl. Sci.* Vol. 4, Issue 10, pp. 144-149. 2017.
- [39] M. Shahin, M. Ali Babar and L. Zhu. "Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices". *IEEE Access*. Vol. 5, pp. 3909-3943. 2017.
- [40] T.H. Ng, Y.T. Yu, S.C. Cheung and W.K. Chan. "Human and program factors affecting the maintenance of programs with deployed design patterns". *Inf. Softw. Technol.* Vol. 54, Issue 1, pp. 99-118. January, 2012.
- [41] D. Ibarra Guzmán, U. C. Islas, C.P. Corona y B.E. Pedro-Za Méndez. "Metodología ágil scrumban en el proceso de desarrollo y mantenimiento de software de la norma moprosoft". *Res. Comput. Sci.* Vol. 79, pp. 97-107. 2014.
- [42] Y. Yamato, S. Katsuragi and N. Miura. "Software Maintenance Evaluation of Agile Software Development Method Based on OpenStack". *IEICE Trans. Inf. Syst.* Vol. E98-D, Issue 7, pp. 1377-1380. 2015.
- [43] D. Federico Sara, S. Noelia, A. Mariela y M. Diana. "Construcción de una Metodología para la Priorización y Selección de Nuevos Requerimientos a Implementar en Software en Etapa de Mantenimiento". In *XVII Workshop de Investigadores en Ciencias de la Computación*. 2015.
- [44] P. Bhatt, W.K. G. Shroff and A.K. Misra. "Influencing Factors in Outsourced Software Maintenance ACM SIGSOFT Software Engineering Notes". *Softw. Eng. Notes*. Vol. 31, Issue 3, pp. 1-6. 2006.
- [45] A. Vizcaino, J.P. Soto, F. García, F. Ruiz y M. Piattini. "Aplicando gestión del conocimiento en el proceso de mantenimiento del software". *Rev. Iberoam. Intel. Artif.* Vol. 10, Issue 31, pp. 91-98. 2006.
- [46] Z. Chentouf. "A Cognitive System to Teach Software Maintenance Project Staffing".

- Tem Journal-Technology Educ. Manag. Informatics. Vol. 6, Issue 4, pp. 699-706. 2017.
- [47] F. Ruiz, M. Polo y M. Piattini. "Utilización de Investigación-Acción en la Definición de un Entorno para la Gestión del Proceso de Mantenimiento del Software".
- [48] D. Carrizo y A. Alfaro. "Método de aseguramiento de la calidad en una metodología de desarrollo de software: un enfoque práctico Quality assurance method in a software development methodology: a practice approach". *Ingeniare. Revista Chilena de Ingeniería*. Vol. 26 N° 1, pp. 114-129. 2018.
- [49] M.G. Estayno, G.N. Dapozo, L.R. Cuenca Pletsch y C.L. Greiner. "Modelos y métricas para evaluar calidad de software". In XI Workshop de Investigadores en Ciencias de la Computación, pp. 382-388. 2009.
- [50] E. Irrazábal y J. Garzás. "Análisis de métricas básicas y herramientas de código libre para medir la mantenibilidad". *REICIS. Rev. Española Innovación, Calid. e Ing. del Softw.* Vol. 6 N° 3, pp. 56-65. 2010.
- [51] J. Acosta, G. Dapozo, C. Greiner y M. Estayno. "Evaluación de mantenibilidad de un gestor de contenidos open source utilizando métricas de orientación a objetos". In X Jornadas Argentinas de Software Libre, pp. 15-29. 2013.
- [52] M. Rodríguez y C.M. Fernández. "Certificación de la Mantenibilidad del Producto Software: Un Caso Práctico". *Rev. Latinoam. Ing. Softw.* Vol. 3 N° 3, pp. 127-134. 2015.
- [53] J.M. Ruiz, C.D. Pacifico y M.M. Pérez. "Clasificación y Evaluación de Métricas de Mantenibilidad Aplicables a Productos de Software Libre". In XIX Workshop de Investigadores en Ciencias de la Computación, pp. 460-464. 2017.
- [54] Í. Donoso Barraza y V. Vega Zepeda. "Factores sociales y humanos que afectan el proceso de educación de requerimientos: Una revisión sistemática". *RISTI - Rev. Iber. Sist. e Tecnol. Inf.*, N° 24, pp. 69-83. 2017.
- [55] IEEE. "IEEE Recommended Practice for Software Requirements Specifications". Vol. 1998. 1998.
- [56] "Software engineering - Lifecycle profiles for Very Small Entities (VSEs) - Management and engineering guide: Generic profile group: Entry profile". 2012.