Prácticas para apoyar la captura y especificación de requisitos de mantenibilidad del producto software

Monografía para optar al título de Ingeniero de Sistemas

Natalia Orozco Yasnó

Brayan Stiven García Navia

Director: Mg. Daniel Eduardo Paz Perafán

Codirector: PhD. Francisco José Pino Correa

Codirector: PhD (c). Sandra Lorena Buitrón Ruiz

Universidad del Cauca

Facultad de Ingeniería Electrónica y Telecomunicaciones

Departamento de Sistemas

Línea de Investigación de Ingeniería del software: producto y proceso

Popayán, noviembre 24 de 2021

Tabla de contenido

Índice de figuras

Índice de tablas

Capítulo 1 - Introducción

1.1.Planteamiento del problema y justificación

Una propiedad de diversos sistemas software es que requieren continuamente cambiar y evolucionar para adaptarse a nuevas necesidades o corregir fallos, por lo tanto, las organizaciones de desarrollo de software se enfrentan al reto de hacer que el software sea altamente mantenible. En este sentido, la mantenibilidad es una característica de calidad esencial para las organizaciones, ya que las tareas de mantenimiento consumen una gran proporción del esfuerzo total gastado en el ciclo de vida del software[1][1][2].

Esta característica, según la norma ISO 25010 [3], representa la capacidad del producto software para ser modificado efectiva y eficientemente, debido a necesidades evolutivas[2], correctivas[3] o perfectivas[4]. Esta característica se subdivide a su vez en las siguientes sub características: modularidad, grado de un sistema que permite que un cambio en un componente tenga un impacto mínimo en los demás; reusabilidad, grado en el que un activo puede ser utilizado en más de un sistema, o en la construcción de otros activos; analizabilidad, grado en que es posible evaluar el impacto sobre un producto o sistema de un cambio previsto; capacidad para ser modificado, grado en que un producto o sistema pueden ser modificado sin introducir defectos o degradar la calidad del producto; y capacidad de prueba, grado con el que se pueden establecer criterios de prueba para un sistema, producto o componente. Por lo tanto, el desafío de la mantenibilidad no se limita a realizar cambios en el código fuente, sino que requiere una mejor comprensión de los elementos que afectan la mantenibilidad desde las diferentes sub características que la constituyen.

Se estima, según [4][5], que el 66% de los costos del ciclo de vida del software son invertidos en el mantenimiento del producto. Por otra parte, el 61% del tiempo que dedican los programadores al desarrollo es invertido en la etapa de mantenimiento, y sólo el 39% es

[1] Ciclo de vida del software: está compuesto por las etapas de: análisis de requisitos, diseño de la arquitectura, construcción de software, pruebas, despliegue y mantenimiento [37].
[2] Necesidad evolutiva: se presenta cuando hay que realizar cambios en el sistema por nuevos requisitos [25]

[3] Necesidad correctiva: se presenta cuando hay que corregir los defectos y errores del sistema, o reemplazo o actualización de los elementos del sistema [25]

[4] Necesidad perfectiva: se presenta cuando hay que prevenir la posibilidad de obsolescencia de los componentes y las tecnologías utilizadas, o porque existen nuevas normas [25]

empleado en nuevos desarrollos [6]. Por lo tanto, uno de los desafíos que tienen las empresas desarrolladoras es crear productos software altamente mantenibles [7][8]. Para lograrlo, las empresas ejecutan diversas buenas prácticas de programación durante el ciclo de vida del software, algunas de ellas son: especificar en el requisito qué es necesario implementar y no cómo se llevará a cabo, crear módulos con la menor interrelación posible, tener como máximo 3 niveles de profundidad en la herencia, evitar la duplicación de código y documentar los casos de prueba que han sido realizados [4][5][6].

En este orden de ideas, la mantenibilidad no suele ser considerada importante durante la etapa de análisis [1][2][9][10], dentro de la cual se realiza el descubrimiento de requisitos (captura), escritura de los requisitos (especificación), resolución de conflictos (negociación), orden de implementación de requisitos (priorización), y se garantiza que las necesidades son especificadas adecuadamente (validación). Especialmente, los requisitos de mantenibilidad (RM) son descuidados por las empresas de desarrollo, entendiendo un RM como una restricción sobre el sistema software, la cual busca aumentar la eficiencia y eficacia en las actividades[5] de modificación del sistema. Este descuido se debe a que se enfocan en las acciones que debe realizar el software sin considerar cuales son las restricciones que permiten controlar y aumentar la capacidad del producto software para ser modificado[11].

Las empresas de desarrollo al abordar la mantenibilidad dentro de la etapa de análisis presentan las siguientes problemáticas: generalmente no logran capturar fácilmente RMs vinculando a diferentes tipos de stakeholders tales como usuarios finales del negocio, usuarios de dirección de la organización y equipo de desarrollo de software [12]; los RMs se representan de manera vaga, ambigua e incompleta [13][14]; y no tienen una comprensión clara de cómo lograr el cumplimiento de los RMs durante el desarrollo del sistema software. Estas dificultades generan varias consecuencias tales como: el equipo de desarrollo confunde o malinterpreta los RMs [13][14], las aplicaciones software son construidas de tal manera que es difícil realizar cambios y las tareas de mantenimiento consumen muchos recursos[6] y se realizan de manera tediosa [15].

Desde la perspectiva de la literatura, se ha establecido que pocos trabajos han investigado la captura y especificación de RMs, no existe una definición común del concepto de RM y además los RMs identificados en propuestas como [1] [13] y [12] suelen ser abstractos, subjetivos o

[5] Actividades de mantenimiento: Las actividades de mantenimiento pueden estar asociadas a correcciones, evoluciones o adaptaciones.
[6] Recurso: activo necesario para realizar una determina tarea de un producto. Un recurso puede ser humano, técnico, financiero o físico. Consultado de la RAE.

planteados desde la etapa de desarrollo. Por otra parte, no se identificó una propuesta que permita describir RMs considerando las diferentes sub características de mantenibilidad.

De acuerdo a lo expresado anteriormente, fueron creadas 12 prácticas las cuales apoyan la captura y especificación de RMs del producto software. Las prácticas están pensadas para ser utilizadas desde la etapa de análisis del desarrollo de un nuevo software. En el contexto de este proyecto una práctica es una directriz que orienta la interacción entre los usuarios finales del negocio, analistas, arquitectos y equipo de desarrollo con la finalidad de capturar uno o varios RMs y especificarlos mediante una notación. Cada una de las prácticas está constituida por 10 campos, entre los principales se tiene: una directriz para capturar los RMs; consideraciones que describen un conjunto de restricciones o aspectos a tener en cuenta en los diferentes elementos que constituyen a los RMs generados; forma en la cual se aumentará la capacidad de mantenimiento dentro del sistema a partir de los RMs; y la notación que deberá ser seguida para la especificación de los RMs.

Las prácticas propuestas permitirán a las empresas desarrolladoras establecer de manera clara y fácil RMs, entender como plasmar esos RMs a través de una notación y determinar como el RM va a repercutir en la mantenibilidad del producto software.

Para el desarrollo de las prácticas se generaron varios aportes en el conocimiento los cuales consisten en: (i) la caracterización de los elementos involucrados en la mantenibilidad del producto software; (ii) el establecimiento de cuales elementos podrían ser abordados durante la etapa de análisis; (iii) una estrategia de investigación planteada para la creación de las prácticas la cual podrá ser reutilizada en la creación de prácticas para capturar otros requisitos de calidad; (iv) 12 prácticas para capturar RMs las cuales plantean un conjunto de directrices que orientan la interacción entre los usuarios finales del negocio, analistas y equipo de desarrollo con la finalidad de capturar los RM y una notación que permite describir los RM capturados.

1.2.Pregunta de investigación

¿Cómo orientar la captura y especificación de requisitos de mantenibilidad del producto software con el fin de disminuir el esfuerzo y costo de mantenimiento realizado por las empresas de desarrollo?

1.3.Objetivos

Objetivo general

- Diseñar un conjunto de prácticas que apoyen la captura y especificación de requisitos de mantenibilidad del producto software.

Objetivos específicos

- Caracterizar, desde la literatura, los elementos involucrados en la mantenibilidad del producto software.
- Diseñar a nivel conceptual y metodológico un conjunto de prácticas que apoyen la captura y especificación de los requisitos de mantenibilidad del producto software, a partir de los elementos anteriormente caracterizados.
- Evaluar la idoneidad de las prácticas propuestas mediante el desarrollo de una aplicación utilizando la técnica de estudio de caso.

1.4.Metodología

Para el desarrollo de este trabajo se utilizó la metodología de investigación acción-multiciclo con bifurcaciones [16], el cual está conformado por un conjunto de ciclos, en los cuales se realiza de manera general las siguientes actividades:

- Diagnóstico: identificar el tema de investigación, analizar la literatura relevante, y planificar y diseñar el proyecto de investigación.
- Acción: definir los pasos de acción e implementación.
- Reflexión: monitorizar la investigación, evaluar en términos de preguntas de investigación y mejorar el plan y el diseño.

Los ciclos que conformaron la metodología a seguir corresponden al conceptual, metodológico, de evaluación y documentación. La descripción completa de las actividades se encuentra al inicio de los capítulos 3, 4 y 5. A continuación, se describen las principales actividades que conforma cada ciclo.

Ciclo de investigación para la caracterización de los elementos involucrados en la mantenibilidad del producto software.

a) Definición de un protocolo que tenía como fin buscar en la literatura de propuestas de investigación que aborden prácticas, atributos y métricas de mantenibilidad.

b) Análisis de las propuestas encontradas con el fin de identificar los elementos involucrados en la mantenibilidad.
c) Establecimiento de cuales elementos de mantenibilidad se encontraron a partir de modelos de calidad, artículos sobre prácticas de mantenibilidad, artículos sobre métricas de mantenibilidad y artículos sobre atributos de mantenibilidad.
d) Correlación entre elementos identificados y las sub características de la mantenibilidad según la ISO 25010.
e) Descripción de los elementos de mantenibilidad identificados y conclusiones acerca de la caracterización realizada.

Ciclo metodológico para el desarrollo de las prácticas

a) Creación de un modelo en BPMN que definió los paso para crear las prácticas y sus campos.
b) Selección de uno o varios de los elementos de mantenibilidad, con el fin de establecer uno o varios RMs que permitieran su logro.
c) Planteamiento de uno o varios RMs que permitieran el logro de los elementos seleccionados y además que fueran requisitos de producto software.
d) Planteamiento de una directriz que permitiera capturar uno o varios de los RMs establecidos en la actividad anterior.
e) Definición de una notación para escribir los RMs capturados mediante la directriz propuesta.
f) Análisis de la directriz planteada para capturar los RMs y de la notación propuesta, con el fin de determinar qué elementos de mantenibilidad podrán ser potenciados por la práctica propuesta.
g) Establecimiento de cómo los RMs capturados mediante la práctica beneficiarán la mantenibilidad del producto software a desarrollar.
h) Definición de los campos que conforman la práctica para capturar los RMs.
i) Justificación de los elementos de mantenibilidad que no pudieron ser abordados desde la IR.

Ciclo de evaluación para evaluar la idoneidad de las prácticas propuestas.

a) Diseño del estudio de caso estableciendo objetivo, objeto, aspecto evaluado, preguntas de investigación, contexto y unidades de análisis, criterio de selección, instrumentos de

evaluación, indicadores y métricas, sujetos de investigación y planeación del proceso de intervención. El estudio de caso aplicó la metodología de Runeson [17].

b) Desarrollo del estudio de caso siguiendo la planificación establecida, posteriormente se recolectaron los datos y se recogió la evidencia.
c) Análisis de los resultados obtenidos en el estudio de caso y generación de las conclusiones que permitieron determinar la idoneidad de las practicas propuestas.

Ciclo de documentación.

a) Paralelamente a los anteriores ciclos se realizó la elaboración de la monografía del trabajo de grado la cual describe la información recopilada a través del estado actual del conocimiento, las prácticas propuestas, la validación de las prácticas propuestas, y las actividades realizadas en la metodología para alcanzar los objetivos del proyecto.
b) Divulgación y sustentación de los resultados de investigación.

1.5.Estructura de la monografía

La presente monografía organiza su contenido de la siguiente manera:

- Capítulo 2: en esta sección se presentan las definiciones y fundamentos teóricos que guiaron la investigación, la cual tiene como fin garantizar un conocimiento uniforme y completo.
- Capítulo 3: en esta sección se muestra la caracterización de un conjunto de elementos involucrados en la mantenibilidad del producto software. Estos elementos fueron establecidos a partir de modelos de calidad y propuestas de investigación que abordan atributos, métricas y prácticas de mantenibilidad.
- Capítulo 4: en esta sección se describen las actividades realizadas para la construcción de las prácticas que orientan la captura y especificación de RMs, posteriormente se describen cada una de las prácticas creadas junto con los diferentes campos que las constituyen.
- Capítulo 5: en esta sección se presenta la aplicación y evaluación de las prácticas desarrolladas a través de un estudio de caso, incluyendo el contexto de la investigación, resultados y análisis.
- Capítulo 6: en esta sección se presentan las conclusiones, lecciones aprendidas y trabajo futuro.

Capítulo 2 – Contexto teórico y estado del arte

2.1.Contexto teórico

Esta sección presenta algunas definiciones y fundamentos teóricos que guiaron la investigación, la cual tiene como fin garantizar un conocimiento uniforme.

2.1.1. Requisito

Los requisitos para un sistema son las descripciones de lo que debe hacer el sistema, los servicios que brinda y las limitaciones en su operación. En otras palabras, los requisitos reflejan las necesidades de los clientes de un sistema que debe cumplir un determinado propósito [18]. Existen requisitos de producto y requisitos de proceso, los cuales, según [19], [20], son:

Requisito de producto, es una necesidad o una restricción directa sobre el software que se va a desarrollar, o es una limitación del comportamiento que debe tener el software a desarrollar. Un ejemplo de este tipo de requisitos son los siguientes:

- Yo como administrador necesito ver las estadísticas de los planes de mejora para tener conocimiento del progreso de cada plan en ejecución.
- Yo como administrador necesito que el sistema cierre sesión cuando haya pasado 5 minutos de inactividad en la aplicación para conservar la seguridad del aplicativo.
- La aplicación debe ser compatible con todas las versiones de Windows, desde Windows 95.
- Toda funcionalidad del sistema y transacción de negocio debe responder al usuario en menos de 5 segundos.
- El nuevo sistema y sus procedimientos de mantenimiento de datos deben cumplir con las leyes y reglamentos de protección de datos médicos.
- El nuevo sistema se acogerá a las reglas de las licencias generales públicas (GNU), es decir será gratuito, código abierto en el que cualquiera podrá cambiar el software, sin patentes y sin garantías.

Requisito de proceso, es una restricción sobre el proceso de desarrollo de software, la cual es derivada de políticas y procedimientos impuestos directamente por la empresa de desarrollo, clientes o un tercero, como un regulador de seguridad. Entre los requisitos de este tipo se encuentran, la metodología de desarrollo (RUP, SCRUM); estándares de procesos [19];

técnicas rigurosas para capturar requisitos, validar o verificar requisitos con el fin de reducir las fallas que pueden conducir a una confiabilidad inadecuada; entornos de desarrollo; ambientes tecnológicos de desarrollo y pruebas. Un ejemplo de este tipo de requisitos son los siguientes:

- El procedimiento de desarrollo de software a usar debe estar definido explícitamente (en manuales de procedimientos) y debe cumplir con los estándares ISO 9000.
- La metodología de desarrollo de software será RUP
- El sistema debe ser desarrollado utilizando las herramientas Netbeans, StarUML, MOCKFLOW, Edraw.
- Las pruebas de software se ejecutarán utilizando Selenium y Ruby como herramienta y lenguaje Scripting para automatización de software testing.
- Cada dos semanas deberán producirse reportes gerenciales en los cuales se muestre el esfuerzo invertido en cada uno de los componentes del nuevo sistema[18], [20]

2.1.2. Tipos de requisitos

Los requisitos se componen por requisitos funcionales y no funcionales.

Requisitos funcionales: son los servicios que el sistema debe proveer, y de cómo se debe comportar o no el sistema en situaciones específicas[18] Algunos ejemplos de requisitos funcionales son los siguientes:

- Yo como administrador necesito que el sistema me permita gestionar los usuarios de la aplicación.
- Yo como gerente necesito que la aplicación me deje iniciar sesión cuando coloques mis respectivas credenciales.

Requisitos no funcionales (RNF): en [21] se plantea que en la comunidad de Ingeniería de Requisitos no hay consenso sobre “qué son” los RNF y tampoco en cuanto a cómo se capturan, especifican o validan. La literatura utiliza una variedad de términos en las definiciones de RNF y con significados poco precisos, que dan lugar a la ambigüedad sobre su alcance o lo que representan. El soporte conceptual del actual proyecto de investigación lo conforman las definiciones de Cysneiros [22], que considera a los RNF como requisitos de calidad y como restricciones.

Los RNF como requisitos de calidad, representan restricciones o las cualidades que el sistema debe tener tales como: precisión, usabilidad, seguridad, rendimiento, confiabilidad, performance entre otras. Al considerar a los RNF como requerimientos de calidad, se contemplarán las seis características del estándar de calidad internacional ISO/IEC 25010 [23], ya que incluye los atributos mencionados en las definiciones de Cysneiros en [22].

Los RNF representan restricciones globales sobre el sistema, un requisito funcional, el proceso de desarrollo o sobre el proceso de despliegue [22]. La lista de RNF debe capturarse y especificarse antes de iniciar el documento de requisitos. Algunos ejemplos de RNF son los siguientes:

- Requisito de usabilidad. El personal médico podrá utilizar todas las funciones del sistema después de cuatro horas de formación. Después de esta formación, el número medio de errores cometidos por usuarios experimentados no superará los dos por hora de uso del sistema.
- Requisito de eficiencia de desempeño. Toda funcionalidad del sistema y transacción de negocio debe responder al usuario en menos de 5 segundos.
- Requisito de disponibilidad. El MHC-PMS estará disponible para todas las clínicas durante el horario laboral normal (de lunes a viernes, de 08.30 a 17.30). El tiempo de inactividad dentro del horario normal de trabajo no excederá de cinco segundos en un día cualquiera.
- Requisito de seguridad. Todas las comunicaciones externas entre servidores de datos, aplicación y cliente del sistema deben estar encriptadas utilizando el algoritmo RSA.

2.1.3. Ingeniería de requisitos

En [24] plantea que, la ingeniería de requisitos se ocupa de descubrir, obtener, desarrollar, analizar y determinar métodos de verificación, validación, comunicación, documentación y gestión de requisitos. El resultado de la ingeniería de requisitos es una jerarquía de requisitos que:

- Permite un entendimiento acordado entre las partes interesadas (por ejemplo, adquirentes, usuarios, clientes, operadores, proveedores).
- Está validada contra las necesidades del mundo real.
- Se puede implementar.
- Proporciona una base para verificar diseños y aceptar soluciones.

Dentro de esta, es clave clarificar varios conceptos que permitirán que ese proceso tenga una adecuada implementación. La IR está conformada por las siguientes actividades según [18], [24]:

- Entender el contexto, consiste en identificar los stakeholders que tienen un rol o interés en el sistema deseado y seleccionar las fuentes de los requisitos, por otro lado, se hace una estimación de si las necesidades identificadas del usuario pueden ser satisfechas con las tecnologías actuales de software y hardware. El estudio considera si el sistema propuesto será rentable desde un punto de vista comercial y si puede desarrollarse dentro de las limitaciones presupuestarias existentes. Un estudio de viabilidad debería ser relativamente barato y rápido. El resultado debe informar la decisión de si seguir adelante o no con un análisis más detallado
- Captura, consiste en capturar los requisitos funcionales y no funcionales del sistema que los stakeholders consideren necesarios, adicionalmente, en esta actividad se analizan los requisitos con el propósito de identificar requisitos faltantes, incompletos, conflictivos, no realizables, ambiguos, inconsistentes, entre otros; para ello existen unas técnicas claras y precisas como las entrevistas y los escenarios.
- Negociación, consiste en establecer con los stakeholders soluciones a las problemáticas identificadas en los requisitos, además, que la resolución de los problemas no ha corrompido ni comprometido sus intenciones.
- Especificación de requisitos, consiste en representar los requisitos mediante notaciones como casos de uso, historias de usuario, prototipos, notaciones matemáticas. Estas notaciones permiten que estos requisitos queden claros, trazables, verificables, no ambiguos, entre otras; además, traduce la información recopilada durante la actividad de análisis en un documento que define un conjunto de requisitos. Se pueden incluir dos tipos de requisitos en este documento. Los requisitos del usuario son declaraciones abstractas de los requisitos del sistema para el cliente y el usuario final del sistema; los requisitos del sistema son una descripción más detallada de la funcionalidad que se proporcionará.
- Verificación, consiste en garantizar que los requisitos cumplan con las características de calidad como completos, no ambiguos, verificables y se expresan con un nivel de detalle apropiado.

- Validación, consiste en presentar los requisitos especificados y analizados a los stakeholders con el fin de garantizar que las necesidades y expectativas se hayan captado y expresado adecuadamente.
- Priorización, consiste en priorizar los diferentes requisitos que se van a desarrollar según la visión del sistema, la urgencia, las limitaciones de tiempo, complejidad o preferencias que tengan los stakeholders.

2.1.4. Mantenibilidad

Según la norma ISO 25010 [23], representa la capacidad del producto software para ser modificado efectiva y eficientemente, debido a necesidades evolutivas, correctivas o perfectivas. Entendiendo una necesidad evolutiva como aquella que se presenta cuando hay que realizar cambios en el sistema por nuevos requisitos [25] entendiendo una necesidad correctiva como aquella que se presenta cuando hay que corregir los defectos y errores del sistema, o reemplazo o actualización de los elementos del sistema [25]; y entendiendo una necesidad perfectiva como aquella que se presenta cuando hay que prevenir la posibilidad de obsolescencia de los componentes y las tecnologías utilizadas, o porque existen nuevas normas [25]. Esta característica se subdivide a su vez en las siguientes sub características:

- Modularidad, grado de un sistema que permite que un cambio en un componente tenga un impacto mínimo en los demás.
- Reusabilidad, grado en el que un activo puede ser utilizado en más de un sistema, o en la construcción de otros activos.
- Analizabilidad, grado en que es posible evaluar el impacto sobre un producto o sistema de un cambio previsto.
- Capacidad para ser modificado, grado en que un producto o sistema pueden ser modificado sin introducir defectos o degradar la calidad del producto.
- Capacidad de prueba, grado con el que se pueden establecer criterios de prueba para un sistema, producto o componente.

Por lo tanto, el desafío de la mantenibilidad no se limita a realizar cambios en el código fuente, sino que requiere una mejor comprensión de los elementos que afectan la mantenibilidad desde las diferentes sub características que la constituyen.

2.1.5. Producto software

Un software de programación es el conjunto de utilidades y herramientas utilizadas para el desarrollo, programación o creación de programas o aplicaciones informáticas por parte de los programadores. Dichas utilidades y herramientas pueden hacer uso de diversos lenguajes de programación y metodologías de desarrollo a través de, como mínimo, un editor de texto y un compilador.

El producto software está constituido por los siguientes elementos:

- **Sentencia,** las sentencias son los elementos básicos en los que se divide el código en un lenguaje de programación. Al fin y al cabo, un programa no es más que un conjunto de sentencias que se ejecutan para realizar una cierta tarea. Una sentencia puede tener varias líneas de código.
- **Líneas de código,** cuando un programador se enfrenta a un problema, lo analiza, y al final acaba generando líneas de código en su programa. Estas líneas de código son las sentencias del lenguaje que esté utilizando, las cuales harán que el programa funcione. Su más mínima expresión es una única línea de código.

  Componente, un componente es una parte de un programa que va desde una sección o rutina del mismo, una clase, hasta un conjunto de archivos asociados a una función.

  Módulo, un módulo es una parte de un programa. Un módulo está constituido por un conjunto de funciones relacionadas y altamente cohesivas entre sí. Las funciones realizadas entre módulos son únicas e independientes.

2.2.Estado del arte

Fue realizada una revisión sistemática de la literatura con el fin de identificar propuestas que aborden características de mantenibilidad desde etapas de análisis o diseño. En las siguientes secciones se presenta el diseño de la revisión sistemática, las propuestas encontradas y una conclusión respecto a cómo las propuestas encontradas apoyan la captura de RMs desde la ingeniería de requisitos.

2.2.1. Diseño de la revisión sistemática

La revisión consideró aspectos del protocolo propuesto por Kitchenham [26] por medio del cual se desarrolló la planificación y ejecución de la revisión. En la planificación se establecieron los siguientes elementos: pregunta de investigación, criterios de inclusión y exclusión y cadenas de búsqueda. En la ejecución se definieron los siguientes pasos: (i) aplicar

las cadenas de búsqueda a diversas bases de datos; (ii) leer el título, resumen y conclusiones, aplicando los criterios de inclusión y exclusión, los cual genero un conjunto de publicaciones potenciales; (iii) retirar los estudios duplicados; (iv) realizar una lectura completa a las publicaciones potenciales y aplicar los criterios de inclusión y exclusión, lo cual generó un conjunto de publicaciones seleccionadas; (v) extraer datos y realizar la síntesis.

Protocolo de búsqueda

Durante la actividad de planeación de la revisión se definió el protocolo, el cual especifica el método que se utilizará en la revisión sistemática a fin de reducir la parcialidad de los investigadores. El protocolo de revisión desarrollado fue realizado por un autor y revisado por los autores restantes para evitar un sesgo en la revisión.

Pregunta de Investigación

Es la pregunta que dirige los estudios primarios en el proceso de búsqueda. La pregunta de investigación formulada es:

- ¿Cuáles propuestas abordan la mantenibilidad desde las etapas de análisis y diseño?
- ¿Cuáles son los alcances y características de estas propuestas?

Criterios de inclusión y exclusión

Los estudios de investigación incluidos en esta revisión deben cumplir con los criterios establecidos para asegurar que se encontraron todos los estudios relevantes. Todos los criterios de inclusión deben ser satisfechos por un estudio para agregarlo a la revisión. Si un estudio cumple algún criterio de exclusión, no se agrega a la revisión. A continuación, son presentados los criterios de inclusión y exclusión para esta revisión sistemática.

Criterios de inclusión

- Propuestas de investigación que aborden características que se debe tener en cuenta en la mantenibilidad del software en la etapa de análisis o diseño de software OR propuestas de investigación que aborden como identificar o especificar requisitos de mantenibilidad software.
- Estudios publicados en Workshop, conferencias, revistas o reportes técnicos.
- Estudios publicados entre el año 1990 y 2021.

Criterios de exclusion

- Estudios que no aborden la mantenibilidad de software.
- Artículos publicados en idiomas diferentes al inglés o español.

Fuentes de búsqueda

Las fuentes electrónicas utilizadas en el proceso de búsqueda fueron las siguientes:

- Scopus
- Science Direct
- IEEE
- ACM
- Google scholar

Se seleccionaron de acuerdo con los siguientes requisitos:

- Permiten utilizar expresiones lógicas o un mecanismo similar.
- Permiten búsquedas de longitud completa, o búsquedas basadas sólo en campos específicos de los estudios.
- Están disponibles en la institución del investigador.
- Cubren el área de investigación de interés para esta revisión: Ciencias de la Computación.

Cadenas de búsqueda

Para crear la cadena de búsqueda, se identificaron los términos principales de la pregunta de investigación definida. Los términos de búsqueda sofisticados fueron formados mediante la incorporación de términos y sinónimos alternativos usando la expresión booleana 'OR' y combinando los términos de búsqueda principales con 'AND'. Los siguientes términos generales de búsqueda se utilizaron para la identificación de estudios primarios:

- (“attributes” OR “characteristics” OR “qualities” OR “properties” OR “features” OR “conditions”) AND (“maintainability”)
- (“particularities” OR “peculiarities”) AND (“of maintainability”)
- ("software maintainability" AND "attribute")

Estrategia de Búsqueda

El proceso de selección para incluir los estudios relevantes en la revisión se llevó a cabo siguiendo los pasos mostrados en la figura 1, los cuales son descritos a continuación.

1. Aplicar las cadenas de búsqueda a las bases de datos. Esta búsqueda devuelve un conjunto de publicaciones que satisfacen la cadena de búsqueda en el título, resumen o palabras clave.

2. Leer el título, resumen y conclusiones. Se revisaron los títulos, resúmenes y conclusiones de las publicaciones devueltas mediante la cadena de búsqueda. Posteriormente se aplicaron primero los criterios de inclusión y posteriormente los criterios de exclusión para determinar las publicaciones a incluir y excluir respectivamente. El conjunto de publicaciones que cumplieron los criterios de inclusión se denominó "publicaciones potenciales".

3. Retirar los estudios duplicados. Paralelo al paso 2, las publicaciones devueltas mediante la cadena de búsqueda fueron analizadas por título y autores con el objetivo de encontrar publicaciones duplicadas. Si hay duda si dos o más publicaciones están duplicadas se revisaron los resúmenes.

4. Realizar una lectura completa de las "publicaciones potenciales". Se leyó exhaustivamente todo el contenido de las publicaciones potenciales. Los criterios de exclusión e inclusión se aplicaron nuevamente, de forma que sólo fueron seleccionadas publicaciones que respondan a la pregunta de investigación. El resultado fue un conjunto de publicaciones denominadas "publicaciones seleccionadas"

5. Extraer datos y realizar síntesis. El proceso de extracción y síntesis de datos se llevó a cabo mediante la lectura de todos los artículos que se incluyeron en la revisión.

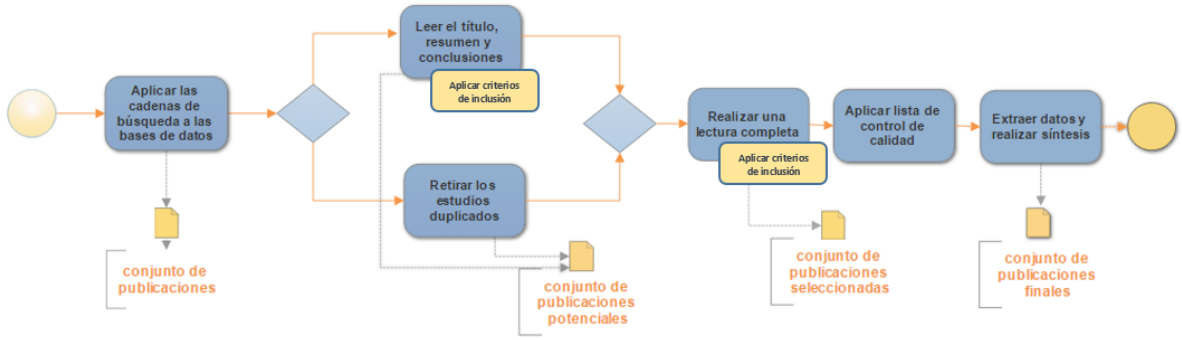


Figura 1. Estrategia de búsqueda. Autor fuente propia.

2.2.2. Trabajos relacionados

A partir del diseño de la estrategia de búsqueda se determinaron una serie de dimensiones para clasificar cada propuesta seleccionada, las cuales fueron: tipo de contribución (herramienta, enfoque, propuesta metodológica o estudio), forma de validación (estudio de caso, encuesta, experimento, ilustración o experiencias) y objetivo de la propuesta los cuales son: abordar RM (ARM), proponer atributos de mantenibilidad (atributos), plantear aspectos de diseño que aumentan la mantenibilidad (diseño) o establecer prácticas que favorecen la mantenibilidad (prácticas). Las propuestas que comprenden la actual revisión de la literatura y sus respectivas dimensiones se encuentran en la tabla 1. A continuación se exponen cada una de las propuestas encontradas clasificadas de acuerdo con su objetivo:

Tabla 1. Trabajos relacionados clasificados de acuerdo con su dimensión

No	Ref	Objetivo	Forma de validación	Tipo de contribución
1	[27]	Diseño	Estudio de caso	Análisis de modularidad y atributos de mantenibilidad
2	[28]	Atributos	Estudio de caso	Método
3	[29]	ARM	Estudio de caso	Notación
4	[30]	Atributos	No la validan	Clasificación de atributos de mantenibilidad
5	[31]	ARM	No la validan	Modelo
6	[32]	Diseño	Estudio de caso	Framework
7	[33]	Prácticas	Ilustración	Modelo
8	[34]	Diseño	Estudio de caso	Análisis de patrones de diseño y mantenibilidad
9	[35]	Prácticas	Experimento	Práctica

10	[36]	Prácticas	Ilustración	Framework
11	[37]	Prácticas	Estudio de caso	Modelo

Propuestas que abordan RM

En [29] se presenta un enfoque para representar RM con el fin de resolver algunos problemas que han identificado en la literatura. La representación propuesta utiliza un perfil UML el cual propone un conjunto de mecanismos de extensión tales como: estereotipos, valores etiquetados y restricciones. La propuesta describe dos términos base para representar los RM, tales como: objetivo de mantenibilidad y operaciones para implementar un objetivo de mantenibilidad. Teniendo en cuenta que una operación puede llevar a cabo varios objetivos de mantenibilidad. Por otra parte, se definen relaciones de herencia entre objetivos y operaciones y también se definen interdependencias entre operaciones que afectan los objetivos de mantenibilidad.

En [31] se propone un conjunto de conceptos generados a partir de las normas ECSS (European Cooperation on Space Standardization), ISO 9126 e IEEE 830, los cuales permiten describir varios tipos de requisitos de mantenibilidad a nivel de sistema, software y hardware. Posteriormente, establecen un conjunto de requisitos de mantenibilidad a nivel de proyecto. Algunos requisitos son: tiempo para localizar fallos, definir operaciones para detectar, corregir y controlar fallos. Esta propuesta tiene una limitante porque los requisitos de mantenibilidad son presentados de una manera muy abstracta y son planteados para orientar cambios en el producto software.

Propuestas que plantean atributos de mantenibilidad

En [28] se plantea que los atributos de mantenibilidad deben ser clasificados según su relación con el tiempo y las necesidades de soporte. Los atributos relacionados con el tiempo se dividen entre proceso y operación, donde algunos de los atributos del proceso son: modularización, establece si la unidad a reemplazar en el mantenimiento es un módulo del sistema; estandarización, permite comprobar la compatibilidad de un elemento a agregar en el sistema con los estándares del mercado; simplicidad, indica el número de actividades utilizadas para realizar un cambio; facilidad de diagnóstico, permite comprobar la facilidad para detectar y diagnosticar un fallo durante la tarea de mantenimiento; e identificación, está asociada a la facilidad con la que se identifican los elementos que deben mantenerse y los casos de prueba.

En [30] el resultado principal 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 propuesto por RUP (Rational Unified Process) el cual plantea las siguientes actividades, entendimiento de negocio, requisitos, análisis y diseño, implementación, pruebas y despliegue. El objetivo de los atributos es ofrecer a las empresas desarrolladoras de software un conjunto de aspectos que se deben considerar durante el proceso de desarrollo de software con el fin de incrementar la mantenibilidad del producto software.

Propuestas que abordan aspectos de diseño que aumentan la mantenibilidad

En [38] se analiza cómo las características[7] que utilizan diferentes componentes afectan la capacidad de cambio y estabilidad del producto software. El análisis se realiza desde la etapa de requisitos, para ello utilizan 3 conceptos dispersión, enredos y cortes transversales. La dispersión ocurre cuando un elemento de un componente fuente está relacionado con múltiples elementos de un componente objetivo. Los enredos ocurren cuando un elemento del componente objetivo está relacionado o tiene múltiples dependencias trazables con elementos del componente fuente y por último los cortes transversales ocurren cuando un elemento de un componente fuente presenta dispersión sobre los elementos objetivo y, en al menos uno los elementos del componente objetivo presentan enredos. A partir de un estudio de caso, presentado en [39], que involucró 3 líneas de producto software, el trabajo de investigación concluyó que cuanto mayor es el grado de dispersión, enredos y cortes transversales el impacto del cambio sobre una funcionalidad es mayor, y se afecta negativamente la estabilidad del sistema.

En [32] se presenta un framework de mantenibilidad y las técnicas que promueven la mantenibilidad en tres niveles de abstracción los cuales son: sistema, arquitectura y componente. La mantenibilidad en la dimensión del sistema se trata desde un punto de vista del negocio, por ejemplo: estimar el esfuerzo requerido para adaptar el producto software a otros contextos, estimar cuán frecuente se requiere modificar el software y anticipar los costos de mantenimiento basado en la longevidad estimada del sistema, entre otras. En la dimensión de arquitectura se enfatiza en la capacidad de cambio, portabilidad y extensibilidad del sistema, se precisa que una arquitectura mantenible debe ser fácilmente modificada tanto en los cambios

[7] Característica: aspecto distintivo visible para el usuario, atributo de calidad o propiedad de un producto software.

previsibles como en los que no. En la dimensión de componente se enfatiza en la interoperabilidad entre componentes de diferentes distribuidores, los métodos de testing que se utilizan y la estabilidad de los componentes a usar.

En [34] a partir de un conjunto de estudios de caso los autores buscan identificar cuál es la relación entre los patrones de diseño y la mantenibilidad del software, algunos de los principales resultados fueron los siguientes: en cuatro de cinco estudios de caso, se aumentó el número de instancias de un patrón de diseño, es decir se hizo mayor uso del patrón y en todos ellos, cada característica de calidad ISO / IEC 9126 (incluida la mantenibilidad) se aumentó en comparación con la revisión anterior donde se aplicaban menos instancias del patrón. Estos resultados apoyan la hipótesis de que el uso de los patrones de diseño tiene un impacto positivo rastreable en la mantenibilidad.

En [36] se plantea un framework que analiza las relaciones entre las preocupaciones y la mantenibilidad para sistemas software orientados a aspectos. Entendiendo una preocupación como cualquier consideración que puede afectar la implementación de un programa tales como: una funcionalidad, un atributo de calidad, un componente. El framework establece tres elementos, un metamodelo para representar las relaciones entre los módulos y las preocupaciones, una terminología para poder analizar como las preocupaciones impactan en la mantenibilidad del software, e identifica un conjunto de métricas que permiten junto con la terminología evaluar la mantenibilidad del sistema.

Propuestas que establecen prácticas que favorecen la mantenibilidad

En [33] se plantea un conjunto de actividades y mejores prácticas que deben seguir los stakeholders con el fin de mejorar la mantenibilidad del producto software. Las actividades y prácticas están clasificadas en las etapas de planificación, arquitectura, codificación, testing y mantenibilidad en el despliegue. En la fase de análisis algunas de las actividades que plantean son las siguientes: documentar los requisitos de mantenibilidad, evaluar el impacto de la mantenibilidad de los posibles componentes a desarrollar, determinar un perfil en las operaciones que mayor mantenibilidad podrían llegar a recibir. Una limitante de este estudio es que no establecen como realizar estas prácticas.

En [35] utilizando lógica difusa, se presenta un enfoque para identificar problemas de comprensibilidad en artefactos tales como: documentación, requisitos, mensajes de commit, entre otros. La identificación de los problemas se realiza desde tres perspectivas léxica,

semántica y sintáctica. Desde la perspectiva léxica compara si los artefactos contienen palabras o frases que se encuentran en resúmenes de problemas relacionados con la comprensibilidad. Desde la perspectiva semántica se comparan las expresiones lingüísticas que componen los artefactos con las expresiones lingüísticas que aparecen en los resúmenes de problemas relacionados con la comprensibilidad. Desde la perspectiva de sintaxis se comparan oraciones que componen los artefactos con oraciones específicas que aparecen con frecuencia en los resúmenes de problemas relacionados con la comprensibilidad.

En [37] se proponen un conjunto de prácticas que permiten aumentar la mantenibilidad del software. Las prácticas abordan las etapas de análisis de requisitos de software, diseño arquitectural de software, diseño detallado de software, construcción de software, pruebas de evaluación de software y operación del software. Desde la perspectiva del análisis de requisitos plantean las siguientes prácticas: incluir solamente los requisitos indispensables para el funcionamiento del sistema; ser conciso en su declaración sin añadir datos innecesarios; y especificar el qué y no el cómo.

2.2.3. Conclusión

A partir del mapeo de la literatura se ha identificado que no existe una definición común del concepto de RM, se carece de un análisis sistémico de los atributos de mantenibilidad que deben ser abordados desde la IR. Algunos trabajos como [40][29][30] plantean que desde la IR se debe abordar aspectos asociados a la modularidad, tiempos para diagnosticar y realizar cambios que permitan corregir fallos en el sistema, estándares para documentar software y componentes que pueden tener un alto grado de cambio cuando el software está en operación.

Por otra parte, las propuestas identificadas en la literatura carecen de una guía clara, completa y sistemática para capturar RM mediante un conjunto de prácticas que orientan la interacción entre los usuarios finales del negocio, analistas y equipo de desarrollo. Desde la perspectiva de la especificación de requisitos de mantenibilidad únicamente se identificó 1 articulo [29], pero son necesarias nuevas notaciones y artefactos para representar las características propias de la mantenibilidad.

La validación de las propuestas es otro aspecto crucial de su análisis, [40][28][29][32][34] y [37] realizan estudios de caso, pero no muestran suficiente información relacionada a la planificación, ejecución y/o resultados y únicamente [38] fue ejecutada en un entorno industrial, lo cual no permite observar si las propuestas generan un impacto positivo en el

producto final. Respecto a las propuestas restantes [30][31] no son validadas y [33][36] utilizan la ilustración, lo cual evidencia poca confianza en la utilidad de las propuestas.

2.2.4. Aportes

El valor generado con la presente investigación consiste en:

- La caracterización de un conjunto de elementos involucrados en la mantenibilidad del producto software. Los elementos fueron establecidos a partir de modelos de calidad, y propuestas de investigación que abordan atributos, métricas y prácticas de mantenibilidad. Esta caracterización se considera un aporte debido a las siguientes razones: (i) los elementos encontrados se clasificaron, entre elementos y sub elementos lo cual permite una relación y asociación entre estos, (ii) se homogeneizó la definición de cada uno de los elementos, (iii) se determinó para cada uno de ellos qué sub características de mantenibilidad de la ISO 25010 potencia y (iv) la caracterización sirve como punto de partida para otro tipo de propuestas que abarquen la mantenibilidad desde otros enfoques.
- La definición de un conjunto de prácticas que orientan la captura y especificación de RM del producto software. Entiendo una práctica como una directriz que orienta la interacción entre los usuarios finales del negocio, analistas y equipo de desarrollo con la finalidad de capturar uno o varios RMs y especificarlos mediante una notación. Las prácticas fueron construidas tomando como partida los elementos de mantenibilidad involucrados.
- La definición de un proceso que orienta cómo construir los diferentes elementos que conforman las prácticas. Se considera que el proceso es un valor generado debido a que puede ser reutilizado para crear otras prácticas para capturar otro tipo de requisitos no funcionales.

Las prácticas propuestas pretenden ayudar a los analistas a identificar y especificar más fácilmente RM de un producto software, los cuales posteriormente sean satisfechos mediante un conjunto de actividades de desarrollo de software. En este sentido, la presente investigación busca a partir de los RM aumentar la capacidad de mantenimiento del software, disminuir los costos de la mantenibilidad y mejorar la disponibilidad del producto software.

Capítulo 3 – Elementos de mantenibilidad

En este capítulo se muestra la caracterización de un conjunto de elementos involucrados en la mantenibilidad del producto software. Los elementos fueron establecidos a partir de modelos de calidad, y propuestas de investigación que abordan atributos, métricas y prácticas de mantenibilidad. Aunado a esto, en el contexto de este proyecto de investigación se endiente por elemento de mantenibilidad todas aquellas propiedades que repercuten de manera positiva o negativa sobre la mantenibilidad del producto software.

En las siguientes secciones se muestra la estrategia de investigación usada para la identificación de los elementos, se presenta la descripción de los elementos identificados, los sub elementos que pueden llegar a constituirlos, las referencias bibliográficas a partir de las cuales se identificaron, la relación de cada uno con las sub características de la mantenibilidad que propone la ISO 25010 y finalmente se presenta un conjunto de conclusiones obtenidas a partir de la caracterización.

3.1.Estrategia para encontrar los elementos de mantenibilidad

La estrategia para identificar y describir los elementos de mantenibilidad se constituyó de las siguientes actividades:

Definición del protocolo de búsqueda: inicialmente fue definido un protocolo de búsqueda con el objetivo de establecer los proyectos de investigación a partir de los cuales identificar los elementos de mantenibilidad. A continuación, se definen los ítems que constituyen el protocolo:

- Pregunta de investigación. ¿Qué elementos involucrados en la mantenibilidad del producto software existen en la literatura?
- Criterios de inclusión. (i) Propuestas de investigación sobre modelos, atributos, métricas y prácticas que abordan elementos de la mantenibilidad del software. (ii) Estudios publicados en Workshop, conferencias, revistas o reportes técnicos. (iii) Estudios publicados entre el año 2010 y 2021.
- Criterios de exclusión. (i) Estudios que no aborden la mantenibilidad de software. (ii) Artículos publicados en idiomas diferentes al inglés o español.
- Fuentes de búsqueda. (i) Scopus. (ii) ScienceDirect.

- Cadenas de búsqueda. Las cadenas de búsqueda fueron formadas mediante la incorporación de términos y sinónimos alternativos usando la expresión booleana 'OR' y combinando los términos de búsqueda principales con 'AND'. Para identificar artículos que abordan prácticas de mantenibilidad se utilizó la siguiente cadena de búsqueda, ((“software”) AND (“practice”) AND (“maintainability”)). Para la identificación de métricas que abordan la mantenibilidad se utilizó la siguiente cadena de búsqueda, ((“software”) AND (“metric” OR “measure”) AND (“maintainability”)). Para la identificación de atributos que abordan la mantenibilidad se utilizó la siguiente cadena de búsqueda, (("software attribute") OR ("software property") OR ("software feature") OR ("software characteristic") AND ("maintainability")). Para la identificación de modelos de calidad que abordan la mantenibilidad se analizaron trabajos como [41], [42] a partir de los cuales se seleccionaron los modelos de McCall[43], Dromey[44], QMOOD[45], ISO 9126[46] y CQ-M[47].

Identificación de propuestas que cumplen los criterios de inclusión: para identificar las propuestas que cumplen los criterios de inclusión se siguieron los siguientes pasos.

1. Fueron aplicadas las cadenas de búsqueda a las bases de datos.
2. Se revisaron los títulos, resúmenes y conclusiones de las publicaciones devueltas mediante la cadena de búsqueda. Posteriormente, se aplicaron primero los criterios de inclusión y posteriormente los criterios de exclusión para determinar las publicaciones a incluir y excluir respectivamente. El conjunto de publicaciones que cumplieron los criterios de inclusión se denominó "publicaciones potenciales".
3. Paralelo al paso 2, las publicaciones devueltas mediante la cadena de búsqueda fueron analizadas por título y autores con el objetivo de encontrar publicaciones duplicadas. Si hay duda si dos o más publicaciones están duplicadas se revisaron los resúmenes.
4. Se leyó exhaustivamente todo el contenido de las publicaciones potenciales. Los criterios de exclusión e inclusión se aplicaron nuevamente, de forma que sólo fueron seleccionadas publicaciones que respondan a la pregunta de investigación. El resultado fue un conjunto de publicaciones denominadas "publicaciones seleccionadas".

Identificación de los elementos de mantenibilidad: en primera instancia se realizó una lectura de los modelos de calidad seleccionados (McCall, Dromey QMOOD, ISO 9126 y CQ-M). Como resultado se identificaron 27 elementos de mantenibilidad los cuales estaban

definidos de manera explícita en los modelos de calidad. En segunda instancia se realizó una lectura completa de todas las publicaciones seleccionadas asociadas a atributos de mantenibilidad, después se leyeron las publicaciones seleccionadas asociadas a métricas de mantenibilidad y por último se leyeron publicaciones seleccionadas asociadas a prácticas para aumentar la mantenibilidad de un producto software. De cada publicación se identificaron los elementos de mantenibilidad y si era posible se obtuvo la definición de cada elemento.

Clasificación de los elementos: como resultado de la actividad anterior fueron obtenidos 27 elementos de mantenibilidad identificados a partir de modelos de calidad, 23 elementos de mantenibilidad a partir de atributos de mantenibilidad, 52 elementos de mantenibilidad a partir de métricas de mantenibilidad, y 47 elementos de mantenibilidad a partir de prácticas de mantenibilidad.

Unión de cada uno de los elementos obtenidos: en esta actividad se unieron los elementos de mantenibilidad anteriormente obtenidos, en este proceso fueron encontrados los siguientes escenarios:

1. Los elementos presentaban el mismo nombre y la misma descripción. En este caso como había unanimidad en su nombre y descripción se dejaba el elemento con el nombre y la descripción.
2. Los elementos presentaban el mismo nombre, pero distinta definición. En este caso se aplicaron los siguientes criterios:
    a. Si las definiciones se referían a elementos distintos se dejaban como tal
    b. Si las definiciones eran complementarias, es decir significaban lo mismo, pero daban detalles que se complementaban entre sí, se hizo una definición propia dando los detalles de cada definición.
3. Los elementos presentaban distintos nombres, pero misma definición o una definición semánticamente igual. En este caso se dejó el nombre del elemento más significativo y la definición más explicativa.

Como resultado de esta actividad obtuvimos 95 elementos sin repetir.

Correlación entre elementos identificados y las sub características de la mantenibilidad según la ISO 25010: se determinó cuales sub características de la mantenibilidad planteadas por la ISO 25010 son potenciadas por los elementos de mantenibilidad anteriormente unidos. Para lograrlo fue realizado un análisis semántico de los elementos de mantenibilidad y de las

sub características de la mantenibilidad planteadas por la ISO 25010 (Modularidad, Reusabilidad, Analizabilidad, Capacidad de ser modificado, Capacidad de ser probado) y además consideramos trabajos como [30], [37].

3.2.Elementos identificados y su relación con la ISO 25010

En la tabla 2 se muestran los elementos de mantenibilidad, su repercusión en las sub características y las referencias discriminadas

Tabla 2. Elementos de mantenibilidad, su repercusión en las sub características y las referencias discriminadas

No	Nombre elemento	Subelemento	Referencias de los modelos	Referencias de artículos que proponen atributos	Referencias de artículos que proponen métricas	Referencias de artículos que proponen prácticas	ISO 25010				
							Modularidad	Reusabilidad	Analizabilidad	Capacidad de ser modificado	Capacidad de ser probado
1	Complejidad (Sencillez)		[43], [45]	[48]–[50]	[51], [52]	[30], [53]			+	+	+
2		Complejidad ciclomática de mcCabe	----	----	[54]–[61]	[62]					
3	Herencia		[45]	[48], [49]	[51], [52], [60], [63]–[65]	[30]		+	+	+	+
4		Profundidad del árbol de herencia	----	----	[56], [66]–[69]	[62]					
5		Cantidad de hijos	----	----	[56], [57], [60], [66]–[68]	----					

6		Rompe trandiciones	----	----	[61]	----					
7		Herencia rechazada	----	----	[61], [70]	----					
8		Clase base conoce a la clase derivada	----	----	[61]	----					
9	Cohesión		[44], [45]	[48]–[50]	[51], [52], [56], [65]–[68], [71]–[74]	[30], [53], [62]	+	+	+	+	+
10	Acoplamiento (Débilmente acoplado)		[44], [45]	[49], [50]	[52], [56], [57], [59], [60], [63], [65], [67], [68], [71], [73], [74]	[30], [53], [62]	+	+	+	+	+
11		Acoplamiento aferente	----	[48]	[51], [68]	----					
12		Acoplamiento eferente	----	[48]	[51], [68]	----					
13		Acoplamiento por referencia	----	[75]	[76]	----					
14		Acoplamiento de interacción	----	----	[76]	----					

15		Acoplamiento de herencia	----	----	[76]	----					
16		Cantidad de mensajes enviados por una entidad en tiempo de ejecución	----	----	[76]	----					
17		Acoplamiento semántico	----	----	[76]	----					
18		Acoplamiento lógico	----	----	[76]	----					
19		Cantidad de mensajes enviados a una entidad en tiempo de ejecución	----	----	[76]	----					
20		Cantidad de métodos invocados en tiempo de ejecución	----	----	[76]	----					
21		Cantidad de clases usadas	----	----	[62][76]	----					

		en tiempo de ejecución									
22		Cantidad de accesos a una entidad en tiempo de ejecución	----	----	[76]	----					
23		Cantidad de accesos por una clase u objeto en tiempo de ejecución	----	----	[76]	----					
24		Ponderación cognitiva de acoplamiento por referencia	----	[75]	----	----					
25	Abstracción		[44], [45]	[49]	[60], [72], [74]	[62]	+	+	+	+	+
26		Abstracción funcional	----	[48]	[51]	----					
27	Encapsulamiento		[44], [45]	[48], [49]	[51], [63], [72]	[30]			+	+	+
28	Líneas de código		----	----	[51], [52], [54]–[58],	[62]				+	+

					[60], [66], [68], [71], [74], [77]						
29		Líneas de código en blanco.	----	[48], [78]	[51], [72]	----					
30	Documentación y comentarios	Cantidad de comentarios	----	[48], [79]	[51], [55], [72]	----			+	+	+
31		Comentarios vs líneas de código	----	----	----	[30]					
32		Líneas de código vs comentarios	----	----	[55]	----					
33		Comentarios iniciales en cada método o función describiendo cada segmento funcional	----	----	----	[80]					

34		Documentación	[44]	----	[74], [81], [82]	[30], [53], [62]					
35	Constitución de una clase	Mensajes	----	[49]	----	----	+		+	+	
36		Cantidad de métodos	----	----	[54], [56], [66], [68], [69], [77]	[62]					
37		Respuesta para una clase	----	----	[54], [66], [67]	----					
38		Envidia de características	----	----	[70]	----					
39		Clase dios	----	----	[61], [70]	----					
40		Método dios	----	----	[70]	----					
41		Clase mal posicionada	----	----	[70]	----					
42		Número de puntos de salida en un método	----	----	[54]	----					
43		Clase perezosa	----	----	[61]	----					

44		Anti-Singleton	----	----	[61]	----					
45	Tamaño		----	----	----	[30]				+	+
46		Tamaño de método	----	----	----	[53]					
47		Tamaño del diseño	----	[49]	----	----					
48		Volumen	----	----	----	[62]					
49		Cantidad de sentencias del código	----	----	----	[62]					
50		Cantidad de atributos y métodos	----	----	----	[62]					
51		Número de módulos	----	----	[55], [56]	----					
53	Polimorfismo		[45]	[49]	----	[30]		+			
54	Flexibilidad		----	[49], [50]	----	----				+	
55	Comprensibilidad		----	[49], [50]	----	----			+	+	+

56		Facilidad de lectura	----	[79]	----	[30]					
57		Auto-descriptivo	[43]–[45]	----	----	----					
58		Facilidad de entendimiento	----	----	----	[30]					
59		Adherirse a estándares	----	----	----	[30], [53]					
60		Nombres significativos de las funciones	----	----	----	[80]					
61	Propensión a defectos		----	[78]	----	----	+			+	
62	Propensión al cambio		----	[78]	----	----	+			+	
63	Consistencia		[43], [45]	----	----	[30]			+	+	+
64	Cumplimiento de mantenibilidad		[83]	----	----	----			+	+	+
65	Completo		[44]	----	----	----					

66	Consistente		[44]	----	----	----			+	+	+
67	Resuelto		[44]	----	----	----			+	+	
68	No redundante		[44]	----	----	----			+	+	+
69	Ajustable		[44]	----	----	----			+	+	
70	Utilizado		[44]	----	----	----			+	+	
71	Genérico		[45]	----	----	----	+	+			
72	Especificado		[44]	----	----	----		+	+	+	+
73	Cirugía con escopeta		----	----	[70]	----	+		+	+	+
74	Interfaces	Uso de interfaces en vez de la implementación	----	----	[70]	----	+	+	+	+	
75		Segregación de interfaces	----	----	[70]	----					
76		Cuchillo de ejército suizo	----	----	[61]	----					

77	Recursos del proyecto	Habilidad del desarrollador	----	[79]	----	----			+	+	+
81		Experiencia del personal IT	----	----	[74], [82]	----					
82	Duplicación de código		----	----	[61], [70]	[30], [53]			+	+	+
83	Código spaghetti		----	----	[61]	----	+		+	+	+
84	Generalidad especulativa		----	----	[61]	----			+	+	+
85	Dependencias	Dependencia inestable	----	----	[61]	----	+	+		+	
86		Dependencia como HUB	----	----	[61]	----					
87		Dependencia cíclica	----	----	[61]	----					
88	Capacidad de expansión		----	----	----	[30]				+	
89	Trazabilidad		----	----	----	[30]			+	+	+
90	Reuso de algoritmos		----	----	----	[84]		+		+	

91	Tecnologías para desarrollar el software	Uso de librerías de terceros	----	----	----	[84]					
92		Uso de frameworks de desarrollo	----	----	----	[84]					
93		Técnicas o paradigmas de programación	----	----	----	[84]					
94		Lenguaje de programación	----	----	----	[84]					
95		Modelos de arquitectura y diseño	----	----	----	[84]					

3.3.Descripción de los elementos identificados.

A partir de la búsqueda en la literatura se encontraron los siguientes elementos con su definición y subelementos si es el caso.

1. **Complejidad (Sencillez):** se evidencia este elemento cuando el código fuente es comprensible, evitando prácticas que aumenten la complejidad [30], [43], [45], [48]–[53].
    a. **Complejidad ciclomática de mcCabe:** se evidencia este elemento en la cantidad de segmentos de código sin ramificaciones en un mismo método. Las estructuras de código que representan la ramificación se definen como: 'for', 'while', 'do', 'if', 'case' (opcional), 'catch' (opcional) y el operador ternario (opcional). [55]–[62], [85]
2. **Herencia:** se evidencia este elemento cuando es posible implementar jerarquías de clases de objetos a partir de clases preexistentes con propósito de reutilización de código y creación de programas extensos [30], [45], [48], [49], [51], [52], [61], [64]–[66].
    a. **Profundidad del árbol de herencia:** se evidencia este elemento en la cantidad de jerarquías de herencia que existen [56], [62], [66]–[69].
    b. **Cantidad de hijos:** se evidencia este elemento por la cantidad de subclases directas de una clase en particular [56], [57], [60], [66]–[68].
    c. **Rompe tradiciones:** se evidencia este elemento cuando una clase heredada proporciona un gran conjunto de nuevos servicios que no están relacionados con los proporcionados por la clase base.[61]
    d. **Herencia rechazada:** se evidencia este elemento en la arquitectura de un sistema cuando una clase no necesita o no usa los métodos o variables que hereda del padre [61], [70].
    e. **Clase base conoce a la clase derivada:** se evidencia este elemento cuando las clases base tienen conocimiento acerca de sus clases derivadas.[61]
3. **Cohesión:** se evidencia este elemento cuando un componente tiene todos sus elementos estrechamente ligados entre sí y todos contribuyen a lograr un único objetivo o función [30], [44], [45], [48]–[53], [56], [62], [65]–[68], [72]–[74], [86]
4. **Acoplamiento (Débilmente acoplado):** se evidencia este elemento en el grado de interdependencias que existe entre módulos de software. [30], [44], [45], [49], [50], [52], [53], [56], [57], [59], [60], [62], [63], [65], [67], [68], [71], [73], [74]

a. **Acoplamiento aferente:** se evidencia este elemento en la especificación del número de clases que hacen uso de una clase específica [48], [51], [68].
b. **Acoplamiento eferente:** se evidencia este elemento en la especificación de número de otras clases utilizadas por una clase específica [48], [51], [68].
c. **Acoplamiento por referencia:** se evidencia este elemento en el número de operaciones de un componente que hace referencia a un atributo de otro componente, ya sea por una variable, una instanciación de la clase, parámetros de métodos, entre otras [75], [76].
d. **Acoplamiento de interacción:** se evidencia este elemento en las invocaciones entre diferentes métodos y sus variables compartidas. Aplica tanto para métodos como para clases.[76]
e. **Acoplamiento de herencia:** se evidencia este elemento cuando una clase es una subclase de otra. [76]
f. **Cantidad de mensajes enviados por una entidad en tiempo de ejecución:** se evidencia este elemento en la cantidad de mensaje enviados desde una entidad en tiempo de ejecución, ya sea una clase o un objeto. [76]
g. **Acoplamiento semántico:** se evidencia este elemento en donde las clases están relacionadas conceptualmente. Este elemento usa la información de los comentarios, identificadores, nombres de variables, para identificar relaciones conceptuales entre entidades. [76]
h. **Acoplamiento lógico:** se evidencia este elemento en las dependencias secuenciales entre versiones del producto software, esto es si un módulo A se cambia en una versión, un módulo B se cambia en la siguiente versión. [76]
i. **Cantidad de mensajes enviados a una entidad en tiempo de ejecución:** se evidencia este elemento en la cantidad de mensajes recibidos por una entidad en tiempo de ejecución, ya sea una clase o un objeto. [76]
j. **Cantidad de métodos invocados en tiempo de ejecución:** se evidencia este elemento en la cantidad de métodos invocados por una clase o un objeto en tiempo de ejecución. [76]
k. **Cantidad de clases usadas en tiempo de ejecución:** se evidencia este elemento en la cantidad de clases usadas por un objeto en tiempo de ejecución. [62], [76]
l. **Cantidad de accesos a una entidad en tiempo de ejecución:** se evidencia este elemento en la cantidad de accesos a una clase u objeto en tiempo de ejecución. [76]

m. **Cantidad de accesos por una clase u objeto en tiempo de ejecución:** se evidencia este elemento en la cantidad de accesos realizados por una entidad a otra en tiempo de ejecución, ya sea clase u objeto. [76]

n. **Ponderación cognitiva de acoplamiento por referencia:** se evidencia este elemento en el número de operaciones de un componente que hace referencia a un atributo de otro componente teniendo en cuenta el tiempo que tarda una persona en considerar que un atributo sea estático, por herencia o dinámico [75].

5. **Abstracción:** se evidencia este elemento en una medida del aspecto de generalización-especificación del diseño. Las clases de un diseño que tienen uno o más descendientes exhiben esta propiedad de abstracción [49]. Un objeto / módulo es suficientemente abstracto si no existe un concepto de nivel superior obvio y útil que abarque el componente [44], [45], [60], [62], [72], [74].
    a. **Abstracción funcional:** se evidencia este elemento en la proporción del número de métodos heredados por una clase al número total de métodos disponibles para acceder a los métodos miembros de la clase [48], [51].
6. **Encapsulamiento:** se evidencia este elemento cuando se agrupan datos y métodos en una única entidad con identidad propia, en donde ciertos datos y métodos están ocultos al exterior y solo mediante los métodos públicos es posible acceder a ellos [30], [44], [45], [48], [49], [51], [63], [72].
7. **Líneas de código:** se evidencia este elemento en la cantidad de líneas de código de una clase, módulo o sistema [51], [52], [55]–[58], [60], [62], [66], [68], [71], [74], [77], [85]
    a. **Líneas de código en blanco:** se evidencia este elemento en el número de las líneas de código en blanco [48], [51], [72], [78].
8. **Documentación y comentarios:**
    a. **Cantidad de comentarios:** se evidencia este elemento como el número total de comentarios del código fuente [48], [51], [55], [76], [79].
    b. **Comentarios vs líneas de código:** se evidencia este elemento en la relación existente entre el número total de líneas de código y el número de líneas de comentarios [30]
    c. **Líneas de código vs comentarios:** se evidencia este elemento en la cantidad de líneas de código que hay por línea de comentario. Puede dar una idea de que tan documentada en cuestión de comentarios está una clase o módulo con respecto a su tamaño. [55]

d. **Comentarios iniciales en cada método o función describiendo cada segmento funcional** [80]
e. **Documentación:** se evidencia este elemento en un componente cuando su propósito, estrategia, intención y propiedades se definen de forma explícita y precisa dentro del componente [30], [44], [53], [62], [74], [81], [82].

9. **Constitución de una clase:**
   a. **Mensajes:** se evidencia este elemento en el número de métodos públicos que están disponibles como servicios para otras clases. Esta es una medida de los servicios que brinda una clase [49]
   b. **Cantidad de métodos:** se evidencia este elemento por la cantidad de métodos locales de una clase en particular [56], [62], [66], [68], [69], [77], [85]
   c. **Respuesta para una clase:** se evidencia este elemento en la cantidad de métodos locales que son llamados dentro de una clase [66], [67], [69]
   d. **Envidia de características:** se evidencia este elemento en la arquitectura de un sistema cuando un método es más usado por otra clase que por la clase en la que se encuentra [70].
   e. **Clase dios:** se evidencia este elemento en la arquitectura de un sistema cuando una clase toma demasiadas responsabilidades relativas a la clase en la que se encuentra [61], [70].
   f. **Método dios:** se evidencia este elemento en la arquitectura de un sistema cuando un método realiza demasiadas funcionalidades comparado con otros métodos de la misma clase [70].
   g. **Clase mal posicionada:** se evidencia este elemento en la arquitectura de un sistema cuando una clase necesita clases de otros paquetes más que las de su propio paquete [70].
   h. **Número de puntos de salida en un método:** se evidencia este elemento en la cantidad de sentencias "return" en un método [85]
   i. **Clase perezosa:** se evidencia este elemento cuando una clase no hace lo suficiente para pagar por ella misma. Es decir, tiene poca o nula funcionalidad e impacto en el sistema. [61]
   j. **Anti-Singleton:** se evidencia este elemento cuando una clase proporciona variables de clase mutables que exhiben las propiedades de variables globales. [61]
10. **Tamaño:** se evidencia este elemento en la cantidad de código del sistema[30].

a. **Tamaño de método:** se evidencia este elemento en el tamaño de un método porque puede afectar negativamente a la analizabilidad del código y por ende aumentar costos en el mantenimiento, se sugiere un tamaño no mayor a 30 líneas de código. [53]
b. **Tamaño del diseño:** se evidencia este elemento en la cantidad de número de clases utilizadas en un diseño [49].
c. **Volumen**: se evidencia este elemento en la cantidad de operaciones y operando usados en el código. [62]
d. **Cantidad de sentencias del código:** se evidencia este elemento en la cantidad de sentencias en el código. La diferencia entre sentencias y líneas de código es que una sentencia puede ser compuesta por más de una línea de código, por lo tanto, representa otra unidad de medida.[62]
e. **Cantidad de atributos y métodos:** se evidencia este elemento en la cantidad de atributos y métodos en el código fuente.[62]
f. **Número de módulos:** se evidencia este elemento en la cantidad de módulos que hay en un sistema. Si la cantidad de módulos es muy elevada comparada con el tamaño del sistema, puede haber un problema de abstracción. Mientras una cantidad moderada puede ayudar a la reusabilidad del código.[55], [56]

11. **Polimorfismo**: se evidencia este elemento en una estructura de herencia, al sobrescribir métodos de clase base en clases derivadas logrando así múltiples comportamientos de objetos cuando se invoca a un mismo método en distintos contextos [30], [45]. Puesto que es una medida de los servicios que se determinan dinámicamente en tiempo de ejecución [49]
12. **Flexibilidad**: se evidencia este elemento al permitir la incorporación de cambios en un diseño. La capacidad de un diseño de adaptarse.[49], [50]
13. **Comprensibilidad:** se evidencia este elemento al comprender y aprender fácilmente el diseño. Se relaciona con la complejidad de la estructura de diseño [49], [50]
    a. **Facilidad de lectura**: se evidencia este elemento en el grado en que un lector puede entender fácil y rápidamente el código fuente. [30], [79]
    b. **Auto-descriptivo:** se evidencia este atributo en un componente si su propósito, estrategia, intención o propiedades son claramente evidentes a partir de la elección de los nombres de los módulos, funciones, métodos o variables y son significativos y congruentes con el contexto de la aplicación. [43]–[45]

c. **Facilidad de entendimiento:** se evidencia este elemento al intentar estimar la correlación entre el estilo de la escritura de código fuente y la documentación correspondiente.[30]
d. **Adherirse a estándares:** se evidencia este elemento al adherirse a estándares en el nombrado de variables, paquetes u otros componentes para mejorar la analizabilidad y capacidad para ser modificado.[30], [53]
e. **Nombres significativos de las funciones**: [80]

14. **Propensión a defectos:** se evidencia este elemento en la identificación de clases o módulos que posiblemente tendrán más defectos en una versión futura del producto software [78]
15. **Propensión al cambio:** se evidencia este elemento en la identificación de clases o módulos que posiblemente tendrás más cambios en una versión futura del producto software. [78]
16. **Consistencia:** se evidencia este elemento cuando en el diseño e implementación del producto software se utilizan prácticas, técnicas y notaciones uniformes. [30], [43], [45]
17. **Cumplimiento de mantenibilidad:** se evidencia este elemento en la capacidad del producto software para adherirse a estándares o convenciones relacionados con la mantenibilidad. [23]
18. **Completo:** en un componente se evidencia este elemento cuando su definición e implementación permiten que se cumplan todas sus funciones previstas de tal manera que no se afecte la confiabilidad o funcionalidad. [44]
19. **Consistente:** en un componente se evidencia este elemento cuando su uso mantiene sus propiedades o funcionalidad, y si todos sus componentes contribuyen y refuerzan su intención o efecto general. [44]
20. **Resuelto:** se evidencia este elemento en el producto software cuando se utilizan estructuras de control adecuadas a la estructura de datos que contienen la información a tratar. Un defecto o falta de evidencia de este elemento es el uso de un único bucle para procesar un array multi-dimensional. [44]
21. **No redundante:** se evidencia este elemento en un componente que tiene todos los componentes necesarios y solo los componentes necesarios para definir e implementarlo. Componentes más allá de lo necesario y suficiente para especificar el proceso, cálculo, estructura de datos o interfaz de usuario violan la propiedad de efectividad para el componente en particular. [44]

22. **Ajustable:** en un componente se evidencia este elemento cuando se utilizan constantes asociadas a valores que nunca cambian dentro del código, y además se utiliza el número mínimo de variables de propósito único necesarias para respaldar el cálculo que realiza. [44]
23. **Utilizado:** un componente es utilizado si se ha definido y luego se utiliza dentro de su alcance. [44]
24. **Genérico:** se evidencia este elemento en un componente si sus cálculos no están atados a un cierto tipo de dato, sino que se pueden abstraer para varios tipos. Ej: Un componente que permite ordenar números y nombres. [45]
25. **Especificado:** se evidencia este elemento en un componente si su funcionalidad está descrita por precondiciones y post condiciones. [44]
26. **Cirugía con escopeta:** se evidencia este elemento en la arquitectura de un sistema cuando al hacer un cambio en una clase no introduce defectos a las clases que se comunican con ella. [70]
27. **Interfaces**
    a. **Uso de interfaces en vez de la implementación:** se evidencia este elemento cuando se utilizan interfaces para separar la implementación de los servicios que ofrece un componente y de esta forma disminuir el acoplamiento [70]
    b. **Segregación de interfaces:** se evidencia este elemento cuando una clase implementa una interfaz con todos los métodos que necesita [70]
    c. **Cuchillo de ejército suizo:** se evidencia este elemento cuando se evita diseñar interfaces con numerosos métodos que intentan anticipar toda posible necesidad.

       En este sentido se debe diseñar las interfaces con métodos que sean cohesivos entre sí. [61]
28. **Recursos del proyecto:**
    a. **Habilidad del desarrollador:** se evidencia este elemento en las habilidades de comprensión del desarrollador, es decir, la capacidad del mismo para comprender el fragmento de texto. [79]
    b. **Experiencia del personal IT:** se evidencia este elemento en la experiencia del personal IT en el área que actualmente está trabajando, ya sea gerenciando, desarrollando en un lenguaje de programación en específico, con una metodología en concreto,etc. [74], [82]

29. **Duplicación de código:** se evidencia este elemento cuando se evita duplicar el código en una clase o método. [30], [53], [61], [70]
30. **Código spaghetti:** se evidencia este elemento cuando se evita hacer uso de una estructura de software hecha a la rapidez, para un propósito en específico que la hace difícil de extender y optimizar. [61]
31. **Generalidad especulativa:** se evidencia este elemento cuando se evita manejar casos especiales en el código que no se requieren, pero se especula que se requieran algún día [61]
32. **Dependencias:**
    a. **Dependencia inestable:** se evidencia este elemento cuando un sistema o componente depende de otros subsistemas o componentes menos estables que él mismo. Esto puede causar un efecto dominó de cambios en el sistema. Se detecta ampliamente en paquetes. [61]
    b. **Dependencia como HUB:** se evidencia este elemento cuando una abstracción tiene dependencias (salientes y entrantes) de un gran número de otras abstracciones. Detectado en clases y paquetes. [61]
    c. **Dependencia cíclica:** se evidencia este atributo en un subsistema (componente) que está involucrado en una cadena de relaciones que rompen la naturaleza a cíclica deseable de la estructura de dependencia de un subsistema. Los subsistemas involucrados en un ciclo de dependencia son difíciles de liberar, mantener o reutilizar de forma aislada. Detectado en clases y paquetes. [61]
33. **Capacidad de expansión:** se evidencia este elemento cuando un cambio físico a la información, funciones computacionales, o almacenamiento de datos puede ser realizado fácilmente [30]
34. **Trazabilidad:** se evidencia este elemento en la habilidad para trazar la representación de diseño o los componentes actuales del programa hacia los requerimientos. [30]
35. **Reuso de algoritmos:** se evidencia este elemento en la complejidad en un producto software porque puede verse simplificada por un amplio reusó de algoritmos cada vez que se pueda para evitar duplicación de código, mejorar la capacidad de ser modificado del mismo. [84]
36. **Tecnologías para desarrollar el software:**
    a. **Uso de librerías de terceros:** se evidencia este elemento en la simplificación del uso de librerías de terceros ya que no es necesario reinventar la rueda cuando ya

hay una solución probada y funcional en una librería de un tercero ya sea open-source o propietaria. [84]

b. **Uso de frameworks de desarrollo:** se evidencia este elemento en la simplificación del uso de frameworks de desarrollo que facilitan e implementan arquitecturas como MVC e instan a los desarrolladores a seguir el modelo que el frameworks ha implementado. [84]
c. **Técnicas o paradigmas de programación:** se evidencia este elemento en la técnica de programación que se usa, ya sea programación orientada a aspectos, programación orientada a objetos, desarrollo de dirigido por modelos, basado en componentes, entre otras. [84]
d. **Lenguaje de programación:** se evidencia este elemento en el lenguaje de programación que se usa, sus facilidades, APIs que ellas tengan, también el paradigma que manejen y que este se ajuste al proyecto a realizar. [84]
e. **Modelos de arquitectura y diseño:** se evidencia este elemento por la presencia de modelos de arquitectura y diseño, ya que ayudan a la reusabilidad de los componentes y a tener mayor control sobre los cambios. [84]

3.4.Conclusiones.

- Ningún artículo de investigación analizado planteaba de manera explícita elementos que deberían ser considerados desde la IR. En este orden de ideas, es necesario establecer un conjunto de criterios que permitan determinar que un elemento de mantenibilidad debe ser abordado desde IR.
- A partir del análisis de los trabajos de investigación se identificaron y caracterizaron 97 elementos y subelementos que constituyen las diferentes sub características de la mantenibilidad, planteadas por la ISO 25010.
- Durante el análisis de los trabajos de investigación se identificó que los elementos de mantenibilidad comenzaban a repetirse, en especial los elementos obtenidos en métricas de mantenibilidad. La cohesión, el acoplamiento y la herencia son los elementos más mencionados en los artículos investigados.
- La mayoría de revisiones o mapeos de la literatura presentaban un conjunto de elementos de mantenibilidad más no planteaban una definición para esos elementos. Al hacer el cruce entre los diferentes artículos se logró encontrar las definiciones para todos los elementos.

- En algunos casos fue complejo encontrar los elementos de mantenibilidad debido a que tuvimos que diferenciar entre buenas prácticas de programación y propiedades que repercuten de manera positiva o negativa sobre la mantenibilidad del producto software.

Capítulo 4 – Prácticas que orientan la captura y especificación de RMs

En este capítulo se presentan un conjunto de prácticas que orientan la captura y especificación de RM del producto software. Entiendo una práctica como una directriz que orienta la interacción entre los usuarios finales del negocio, analistas, arquitectos y equipo de desarrollo con la finalidad de capturar uno o varios RMs y especificarlos mediante una notación. Las prácticas fueron construidas tomando como partida los elementos identificados en el capítulo anterior.

4.1.Estrategia de investigación para crear las prácticas

El desarrollo de las practicas fue orientado por un conjunto de actividades, las cuales se encuentran modeladas mediante un diagrama de flujo utilizando la notación BPMN en la figura 2. El desarrollo de las actividades propuestas y los resultados generados en cada actividad fueron evaluados con los directores y codirectores con el fin de aumentar el grado de confiabilidad de las prácticas obtenidas. A continuación, se describe cada una de las actividades:

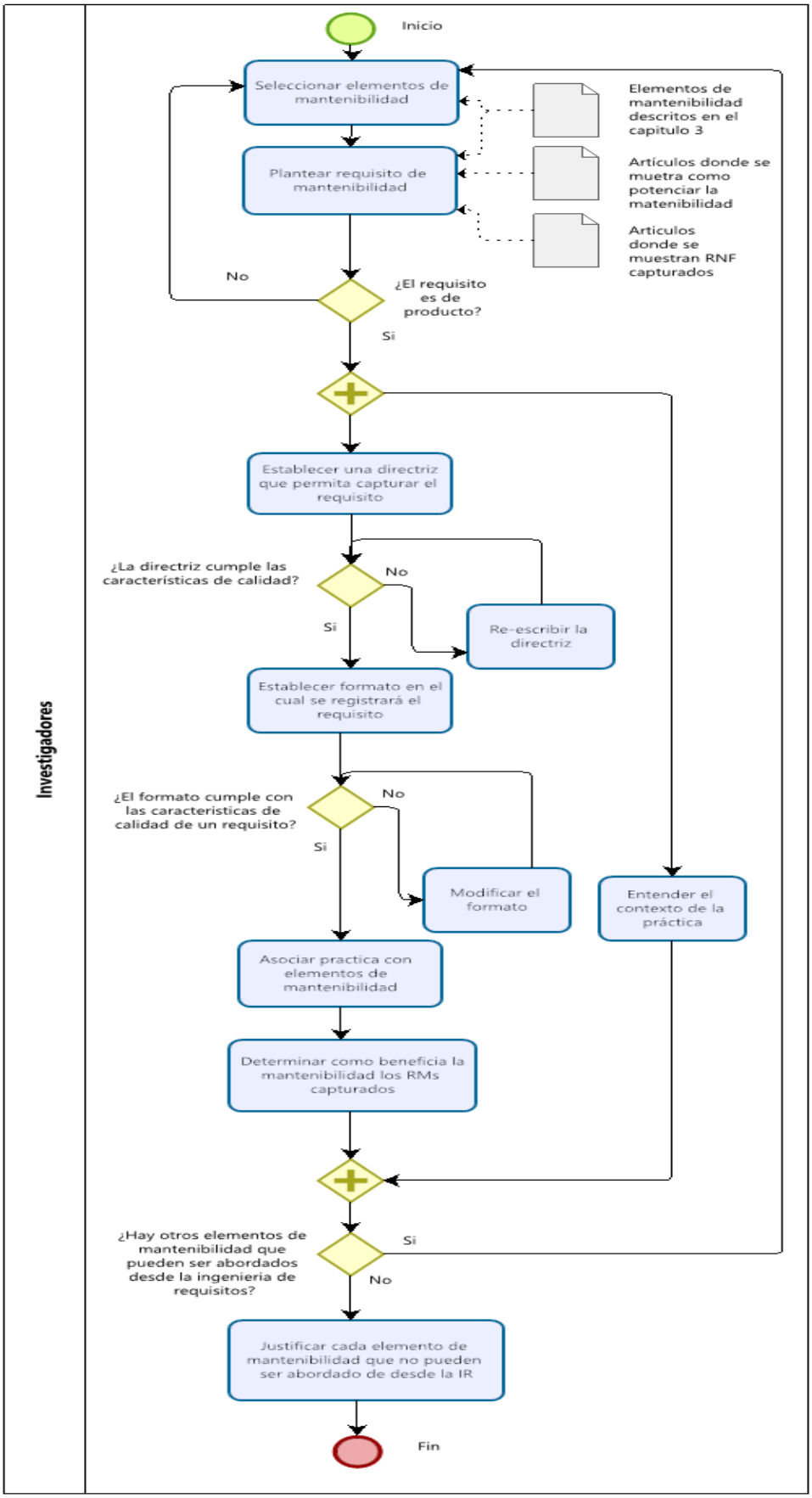


Figura 2. Diagrama de flujo con notación BMPN que permitió crear las prácticas. Autor fuente propia.

Seleccionar elementos de mantenibilidad: en esta actividad fueron seleccionados uno o varios de los elementos de mantenibilidad definidos en el capítulo 3, con el fin de establecer uno o varios requisitos de mantenibilidad que permitieran su logro. El reto de esta actividad fue determinar cuáles elementos podrían ser abordados directamente desde la ingeniería de requisitos, cuales indirectamente y cuales no podían ser abordados.

Plantear requisito de mantenibilidad: en esta actividad fueron planteados uno o varios requisitos de mantenibilidad que permitiera el logro de los elementos seleccionados y además que fueran un requisito de producto software.

El primer reto de esta actividad fue distinguir entre requisitos de mantenibilidad del producto software y requisitos de mantenibilidad del proceso. Entendiendo el primero como una necesidad o una restricción directa sobre el software que se va a desarrollar y el segundo como una restricción sobre las actividades que conducen el desarrollo del software.

El segundo reto de esta actividad fue distinguir entre requisitos de mantenibilidad y buenas prácticas de programación. Entendiendo un RM como una restricción sobre el producto software, la cual busca aumentar la eficiencia y eficacia en las actividades de modificación del producto y el segundo como un conjunto de enfoques empíricamente probados para el desarrollo de software que atacan la raíz de problemas en la construcción del software.

Los requisitos planteados por los investigadores fueron presentados en primera instancia al director y en segunda instancia a los codirectores, con el fin de establecer si el requisito planteado era un requisito de mantenibilidad del producto software y no era un requisito de mantenibilidad del proceso o una buena práctica de programación.

Establecer una directriz que permita capturar el requisito: en esta actividad fue planteada una directriz que permitiera capturar los requisitos de mantenibilidad establecidos en la actividad anterior. Al momento de crear la directriz se buscó que fuera:

- **No ambigua:** está expresada de tal manera que se pueda interpretar de una sola forma y sea fácil de entender.
- **Factible:** indica que mediante la directriz es posible durante la etapa de análisis generar un RM del producto software.
- **Directa:** indica que mediante la directriz es posible generar uno o más RMs que pueden ser escritos llenando los campos que propone la notación.

Entender el contexto de la práctica: en esta actividad se buscó entender con un alto grado de profundidad los diferentes conceptos involucrados en: la directriz, las consideraciones, los elementos que conforman la notación y cómo se potencia la mantenibilidad en los RMs generados. Un ejemplo es la actividad que se realizó para entender qué es un log, los niveles de logs, la validación y verificación de un componente, cómo implementar el polimorfismo, cómo documentar un código, entre otros.

Establecer formato en el cual se registrará el requisito: en esta actividad fue definida una notación, la cual servirá como modelo para escribir los RMs capturados mediante la directriz propuesta anteriormente. La notación presenta unos espacios los cuales deben ser diligenciados por el analista. Durante la definición de la notación buscamos que el RM escrito mediante ella, cumpla con el siguiente conjunto de características formuladas por la norma ISO/IEC 29148 [87]:

- **No ambiguo.** Esta característica indica que el requisito está expresado de tal manera que se pueda interpretar de una sola forma y sea fácil de entender.
- **Completo.** Esta característica indica que el requisito describe suficientemente la capacidad y necesidad que los stakeholders buscan.
- **Singular.** Esta característica indica que la declaración del requisito incluye solo un requisito sin uso de conjunciones.
- **Factible.** Esta característica indica que el requisito es técnicamente alcanzable, no requiere grandes avances tecnológicos y se ajusta a las limitaciones del sistema (por ejemplo, costo, cronograma, técnico, legal, regulatorio) con un riesgo aceptable.
- **Trazable.** Esta característica indica que el RM es trazable con respecto a determinados requisitos funcionales y no funcionales.
- **Verificable.** Esta característica indica que la declaración del requisito permite demostrar que el sistema satisface el requisito especificado.

Asociar práctica con elementos de mantenibilidad: en esta actividad se realizó un análisis de la directriz planteada para capturar los RMs y de la notación propuesta, con el fin de determinar qué elementos de mantenibilidad planteados en la primera actividad podrán ser potenciados por la práctica propuesta.

Asociar las sub características de mantenibilidad potenciadas mediante los RMs capturado: en esta actividad se realizó un análisis de cuáles son las sub características de mantenibilidad que potencian los elementos de mantenibilidad asociados (ver tabla 2).

Considerando el análisis anterior, fueron establecidas las sub características de mantenibilidad que serán potenciadas por los RMs capturados.

Determinar cómo beneficia la mantenibilidad los RMs capturados: a partir de la definición de los elementos seleccionados en la actividad anterior, se estableció cómo los RMs capturados mediante la práctica beneficiarán la mantenibilidad del producto software a desarrollar.

Justificar elementos de mantenibilidad que no pudieron ser abordados desde la IR: en esta actividad se justificaron los elementos de mantenibilidad que no pudieron ser abordados desde la IR, debido a que ellos podían ser analizados desde las etapas de diseño, implementación, pruebas o despliegue.

4.2.Descripción de los roles que pueden hacer uso de las prácticas

En esta sección, a partir de trabajos como [88][89][90], se definen los diferentes roles que están más presentes en un equipo de desarrollo de software, los cuales pueden utilizar las prácticas de mantenibilidad planteadas.

Expertos en la materia: el experto en la materia es la persona o personas de las que se capturan los requisitos. Estas son las personas que saben lo que debe hacer el software y cómo funcionan los procesos de negocio. Los expertos en la materia rara vez provienen de tecnologías de información (TI), excepto cuando la solución está diseñada para brindar soporte a TI. Este rol recibirá con mayor frecuencia el beneficio del sistema.

Analistas funcionales: los analistas funcionales tienen la tarea de obtener requisitos claros, concisos y no conflictivos de los expertos en la materia, que pueden o no comprender cómo puede ser utilizada la tecnología para transformar procesos de negocio de una manera positiva.

Arquitecto de soluciones: el arquitecto de soluciones es responsable de transformar los requisitos capturados por el analista funcional en una arquitectura, la cual será usada por el resto de equipo para crear la solución. El arquitecto de soluciones es el responsable de hacer coincidir las tecnologías con el problema que se está resolviendo.

Líder de desarrollo: el rol del líder de desarrollo se enfoca en proporcionar más detalles a la arquitectura. Esto sería incluir la creación de especificaciones detalladas del programa. Se encarga de unir la arquitectura con los desarrolladores. El líder de desarrollo también se encarga de revisar el código fuente con el objetivo de garantizar su calidad y es la primera línea de

apoyo para los desarrolladores que necesitan ayuda en la comprensión de un concepto o la resolución de un problema.

Asegurador de calidad: la función del asegurador de calidad es encontrar errores antes de que lleguen a los clientes finales. Para la cual utiliza una variedad de técnicas que van desde ingresar datos y jugar con el sistema hasta scripts de prueba formalizados y automatizados. El equipo de aseguramiento de calidad es responsable de garantizar la calidad de la solución y que se ajuste a los requisitos recopilados por el analista funcional. A veces, el equipo de aseguramiento de calidad es conocido por el nombre de tester.

Despliegue: el rol de despliegue se encarga de lograr que la solución sea utilizada por los clientes finales. Crea un programa que construye el entorno operativo que necesita la solución, incluida la configuración de servidores de prueba y desarrollo, la realización de cambios de configuración, la identificación de software conflictivo y la manipulación de cualquier componente del entorno que la solución pueda necesitar para llegar a los clientes finales. El procedimiento de despliegue puede realizarse de manera manual o automatizada.

Gerente de desarrollo: examina, evalúa y valora posibles cambios en el mercado. Determina constantemente formas de mejorar el conjunto de habilidades del equipo de desarrollo, prioriza múltiples necesidades de los proyectos de desarrollo con el fin de mejorar la eficiencia del equipo.

4.3.Presentación de las prácticas

Las siguientes prácticas fueron creadas a partir del análisis del capítulo 3, y además se utilizó como punto de partida las siguientes referencias de la literatura [91],[92], [93]. Las prácticas orientarán al analista en cómo capturar RM del producto software. Los RM generados posteriormente pueden ser alamcenados en un documento de especificación de requisitos.

4.3.1. Descripción de los campos que constituyen las prácticas propuestas

Todas las prácticas propuestas están constituidas por un conjunto de campos que guiarán la captura y especificación de los RM. En la figura 3 se muestran los 10 campos que constituyen una práctica y al lado de cada campo su identificador. Los campos del 1 al 7 definen lineamientos para la captura de los RM, mientras que los campos 8, 9 y 10 son los únicos que deben ser diligenciados por el analista con el objetivo de especificar el RM. A continuación, se describen los campos que conforman una práctica:

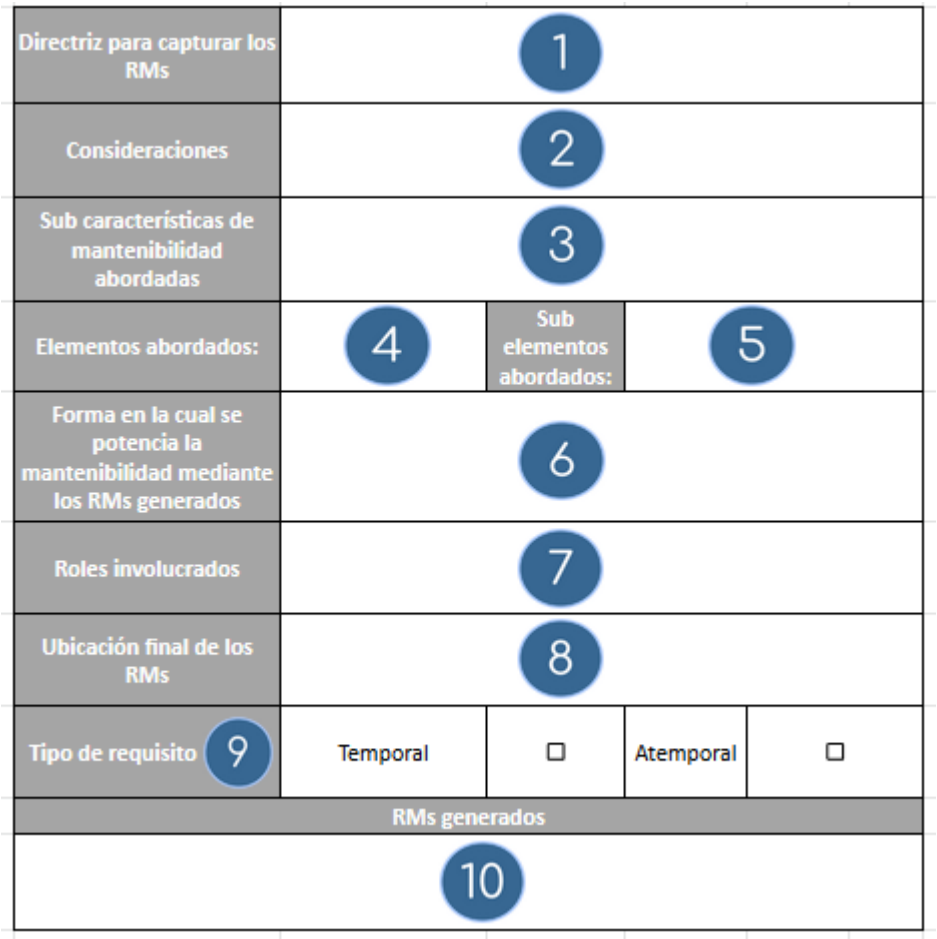


Figura 3. Campos que constituyen una práctica. Autor fuente propia.

Directriz para capturar los RMs (1): este campo establece una instrucción que debe ser seguida por el analista y otros roles de la empresa desarrolladora con el propósito de obtener RMs para el producto software a desarrollar.

Consideraciones (2): en este campo se describe un conjunto de restricciones o aspectos a tener en cuenta en los diferentes elementos que constituyen a los RMs generados mediante la práctica.

Sub características de mantenibilidad abordadas (3): este campo nombra las sub características de mantenibilidad de la ISO 25010 que serán potenciadas en su logro mediante los RMs generados. Las sub características identificadas fueron establecidas a partir de qué sub características potencian los elementos abordados.

Elementos abordados (4): este campo nombra los elementos de mantenibilidad que serán potenciados en su logro mediante los RMs generados.

Sub elementos abordados (5): este campo nombra los sub elementos de mantenibilidad que serán potenciados en su logro mediante los RMs generados. Se debe considerar que en algunos casos no todos los sub elementos que constituyen un elemento van a ser potenciados.

Forma en la cual se potencia la mantenibilidad mediante los RMs generados (6): considerando que los RMs generados se implementan exitosamente, este campo describe la forma en la cual se aumentará la capacidad de mantenimiento dentro del sistema a partir de los RMs. La descripción fue realizada considerando los elementos y subelementos asociados en los campos 3 y 4.

Roles involucrados (7): este campo plantea los roles con los que puede interactuar el analista funcional, con el propósito de obtener los RMs. La interacción estará guiada por la directriz planteada en la práctica.

Ubicación final de los RMs (8): este campo plantea el nombre del artefacto donde se almacenarán los RMs capturados y especificados por la práctica. La ubicación puede ser la ruta al documento de especificación de requisitos donde se almacenan los RF y RNFs, la herramienta software para la gestión de proyectos (por ejemplo, JIRA, trello, clickup).

Tipo de requisito (9): este campo permite establecer mediante un check si los RMs generados son temporales o atemporales. Un RM es *temporal* si será establecido únicamente para un determinado producto software a desarrollar, por otra parte, un RM es *atemporal* si será establecido en varios productos software a desarrollar.

RMs generados (10): este campo contendrá la especificación del RM generado a partir de los lineamientos plateados en los campos anteriores. Este campo planteará una notación que deberá ser seguida en la especificación de los RMs.

4.3.2. Presentación de las prácticas propuestas

A partir de los elementos propuestos y sub elementos caracterizados en el capítulo 3, y de acuerdo al proceso seguido en la sección 4.1, fueron elaboradas 12 prácticas, las cuales buscan apoyar y orientar a los analistas en la captura y especificación de RMs del producto software a desarrollar. Los RMs generados posteriormente pueden ser almacenados en un documento de especificación de requisitos. Los RMs generados van a ser desarrollados en las etapas de diseño, implementación, despliegue y pruebas. En el anexo 8, se presentan ejemplos de captura y especificación de RMs utilizando cada una de las prácticas propuestas

En la tabla 3 se muestra una práctica que busca establecer los estándares o documentos para escribir el código fuente.

Tabla 3. Práctica para establecer lineamientos en la codificación del software

1				
Directriz para capturar los RMs	Establezca los estándares o documentos que provean lineamientos para escribir el código fuente durante la fase de desarrollo.			
Consideraciones	El estándar o documento a seleccionar mínimo debe plantear lineamientos para: 1. La notación del código fuente. Por ejemplo, Pascal case, Camel case, entre otros. 2. El nombrado de variables globales y locales, considerando prefijos, sufijos, patrones. 3. El nombrado de archivos, paquetes, clases, módulos, procedimientos, funciones, entre otros.			
Sub características de mantenibilidad abordadas	Analizabilidad, capacidad de ser modificado, capacidad de ser probado			
Elementos abordados:	Complejidad, cumplimiento de mantenibilidad, consistencia, comprensibilidad, código spaghetti	**Sub elementos abordados:**	Facilidad de lectura, auto-descriptivo, facilidad de entendimiento, adherirse a estándares, nombres significativos de las funciones.	
Forma en la cual se potencia la mantenibilidad mediante los RMs generados	El RM generado a través de esta práctica busca que el código fuente sea más comprensible, auto-descriptivo y fácil de entender debido a que se adhiere a unos lineamientos particulares definidos en un documento o estándar.			
Roles involucrados	Líder de desarrollo, arquitecto de soluciones			
Ubicación final de los RMs				
Tipo de requisito	**Temporal**	☐	**Atemporal**	☐
RMs generados				
El código fuente del sistema a desarrollar debe seguir los lineamientos planteados en el documento o estándar con nombre _____, con versión _____, que se encuentra en el siguiente enlace o ubicación _____.				

En la tabla 4 se muestra una práctica que busca establecer los estándares o documentos para documentar el código fuente.

Tabla 4. Práctica para establecer lineamientos para la documentación del código fuente.

2				
Directriz para capturar los RMs	Establezca los estándares o documentos que se van a utilizar en la etapa de desarrollo del producto software para proveer los lineamientos de la documentación del código fuente.			
Consideraciones	El estándar o documento a seleccionar mínimo debe plantear lineamientos acerca de: 1. La escritura de comentarios, esto incluye sangría, número de líneas máximo por comentario, separadores, etc. 2. Lineamientos para documentar el código fuente.			
Sub características de mantenibilidad abordadas	Reusabilidad, analizabilidad, capacidad de ser modificado, capacidad de ser probado			
Elementos abordados:	Documentación y comentarios, comprensibilidad, cumplimiento de mantenibilidad, especificado	**Sub elementos abordados:**	Comentarios iniciales en cada método o función describiendo cada segmento funcional, cantidad de comentarios, documentación.	
Forma en la cual se potencia la mantenibilidad mediante los RMs generados	El RM generado a través de esta práctica busca homogeneizar la manera en que se documenta el código fuente y aumentar la facilidad de con la cual los desarrolladores entienden el propósito del código fuente.			
Roles involucrados	Líder de desarrollo			
Ubicación final de los RMs				
Tipo de requisito	**Temporal**	☐	**Atemporal**	☐
RMs generados				
La documentación del código fuente del sistema a desarrollar debe seguir los lineamientos planteados en el documento o estándar con nombre _____, con versión _____, que se encuentra en el siguiente enlace o ubicación _____.				

En la tabla 5 se muestra una práctica que busca desde una etapa temprana establecer un puntaje de mantenibilidad para módulos más propensos a cambios, más propensos a errores de codificación en la fase de pruebas y módulos que potencialmente se le añadirán más funcionalidades.

Tabla 5. Práctica para establecer puntaje de mantenibilidad.

3			
Directriz para capturar los RMs	Establezca los módulos que cumplan con las siguientes características: 1. Módulos más propensos a cambios teniendo en cuenta que están ligados a elementos tales como integraciones con otros sistemas, normas, leyes, tecnologías, los cuales por su naturaleza cambian constantemente. 2. Módulos más propensos a errores de codificación en la fase de pruebas. 3. Módulos a los cuales potencialmente se les podrá agregar múltiples funcionalidades. A cada módulo identificado se les asignará un puntaje de mantenibilidad que deberá ser cumplido durante la etapa de desarrollo, con el fin de mejorar la mantenibilidad del producto software.		
Consideraciones	1. El puntaje de mantenibilidad a ser establecido para un módulo está basado en el modelo de calidad SQALE. Este modelo establece que la mantenibilidad del software está directamente relacionada con el costo de corregir la suma de malas prácticas que ocasionan deudas técnicas. Malas prácticas pueden ser no colocar un default en un switch, tener un método con más de 7 parámetros, un alto acoplamiento entre componentes, entre otras. 2. Entre más malas prácticas se deban corregir menor va a ser la capacidad de mantenimiento del software. 3. Los módulos deben identificarse a partir de los requisitos funcionales. 4. El puntaje de mantenibilidad establece 5 rangos A, B, C, D y E. El rango A va del puntaje 0 al 0.05, el B va desde 0.06 al 0.1, el C va desde 0.11 al 0.20, el D va desde 0.21 al 0.5 y el E va desde el 0.51 al 1. El puntaje de mantenibilidad A quiere decir que el esfuerzo para mantener el código es menor o igual al 5% del tiempo invertido en el desarrollo de la aplicación, el puntaje de mantenibilidad B es del 6% a 10% , el puntaje C es del 11% a 20%, el puntaje D es del 21% al 50% y el puntaje E mayor al 50%. Por ejemplo, si un módulo tomó 100h/hombre desarrollarlo y su puntaje de mantenibilidad es A, quiere decir que el costo o esfuerzo para mantener el código es menor o igual a 5h/hombre. 5. Si el puntaje de mantenibilidad es A quiere decir que el software desarrollado viola pocas buenas prácticas de programación, la deuda técnica es baja y por ende la capacidad de mantenimiento es alta, mientras que un puntaje de mantenibilidad E quiere decir que el software desarrollado viola muchas buenas prácticas, la deuda técnica es alta y por ende la capacidad de mantenimiento es baja. 6. Para calcular el puntaje de mantenibilidad proponemos utilizar la herramienta SonarQube la cual se encuentra en la siguiente guía oficial https://docs.sonarqube.org/latest/.		
Sub características de mantenibilidad abordadas	Modularidad, reusabilidad, analizabilidad, capacidad de ser modificado, capacidad de ser probado		
Elementos abordados:	Complejidad, Herencia, Cohesión, Acoplamiento, Abstracción, Encapsulamiento, Líneas de código, Documentación y comentarios, constitución de una clase, tamaño,	**Sub elementos abordados:**	Complejidad ciclomática de mcCabe, profunidad del árbol de herencia, cantidad de hijos, rompe tradiciones, herencia rechazada, clase conoce a la clase derivada, Acoplamiento aferente,

	polimorfismo, comprensibilidad, consistencia, cumplimiento de mantenibilidad, consistente, no redundante, ajustable, utilizado, interfaces, duplicación de código, código spaghetti, reuso de algoritmos, tecnologías para desarrollar el software.		acoplamiento eferente, acoplamiento de herencia, acoplamiento semántico ,abstracción funcional, líneas de código en blanco, mensajes, cantidad comentarios, respuesta para una clase, cantidad de métodos, envidia de características, clase dios, método dios, clase mal posicionada, número de puntos de salida de un método, clase perezosa, anti-singleton, tamaño de método, facilidad de lectura, adherirse a estándares, segregación de interfaces, cuchillo de ejercito suizo

Forma en la cual se potencia la mantenibilidad mediante los RMs generados	Los RMs generados a través de esta práctica buscan desde una etapa temprana establecer un puntaje de mantenibilidad que debe ser cumplido por un determinado módulo, con el propósito de mejorar la mantenibilidad del producto software a desarrollar.			
Roles involucrados	Arquitecto de soluciones, expertos en la materia, líder de desarrollo, asegurador de calidad, gerente de desarrollo			
Ubicación final de los RMs				
Tipo de requisito	**Temporal**	☐	**Atemporal**	☐

RMs generados
1.El módulo con nombre _____ que se compone de los siguientes requisitos funcionales identificados con los ID: ____, _____, _____, _____, _____ debe tener un puntaje de mantenibilidad que esté entre A y B. El puntaje de mantenibilidad ha sido planteado porque el módulo cumple con las siguientes características: _____, _____.

En la tabla 6 se muestra una práctica que busca establecer el STACK tecnológico para el producto a desarrollar con el propósito de disminuir problemas de compatibilidad y de dependencias entre diferentes módulos de software.

Tabla 6. Práctica para establecer el STACK tecnológico.

4				
Directriz para capturar los RMs	Establezca el STACK tecnológico para el producto software a desarrollar.			
Consideraciones	1. Para definir el STACK se debe especificar los siguientes elementos que apliquen: a. Framework de desarrollo b. Lenguaje de programación c. Librerías adicionales de terceros d. Servidor web e. Sistema operativo del servidor web f. Gestor de base de datos g. Proveedor de servicios en la nube 2. Si se contempla separación entre back y front definir los elementos que apliquen tanto para back como para front. 3. Se puede apoyar en STACKs de la industria como LAMP, WAMP, entre otros. Pero se debe especificar los elementos mencionados en la primera consideración.			
Sub características de mantenibilidad abordadas	Modularidad, reusabilidad, capacidad de ser modificado			
Elementos abordados:	Tecnologías para desarrollar el software, Dependencias	**Sub elementos abordados:**	Técnica o paradigmas de programación, uso de librerías de terceros, lenguaje de programación, uso de frameworks de desarrollo, dependencia inestable	
Forma en la cual se potencia la mantenibilidad mediante los RMs generados	Los RMs generados por esta práctica buscan mejorar la elección de los elementos del STACK, en busca de disminuir problemas de compatibilidad entre ellos, y eliminar substancialmente dependencias hacia componentes inestables que afectan en gran medida la mantenibilidad del producto software.			
Roles involucrados	Arquitecto de soluciones, despliegue, expertos en la materia			
Ubicación final de los RMs				
Tipo de requisito	**Temporal**	☐	**Atemporal**	☐
RMs generados				
A. Producto software con back y front en el mismo proyecto. 1. La aplicación debe ser desarrollada en el framework con nombre _____, versión _____. 2. La aplicación debe ser desarrollada con el lenguaje de programación con nombre _____ , versión _____. 3. La aplicación debe incluir la librería con nombre _____, versión _____. 4. La aplicación debe trabajar con el servidor web con nombre _____, versión _____. 5. El servidor web donde se desplegará la aplicación debe tener el sistema operativo con nombre _____, versión _____. 6. La aplicación a desarrollar debe trabajar con el sistema gestor de base de datos con nombre _____, versión _____. 7. La aplicación a desarrollar debe trabajar con el proveedor de servicios en la nube con nombre _____. B. Producto software con separación entre back y front 1. El front-end debe ser desarrollado en el framework con nombre _____, versión _____.				

2. El front-end debe ser desarrollado con el lenguaje de programación con nombre _____, versión _____.
3. El front-end debe incluir la librería con nombre _____, versión _____.
4. El front-end debe trabajar con el servidor web con nombre _____, versión _____.
5. El servidor web donde se desplegará el front-end debe tener el sistema operativo con nombre _____, versión _____.
6. El back-end debe ser desarrollado en el framework con nombre _____, versión _____.
7. El back-end debe ser desarrollado con el lenguaje de programación con nombre _____ , versión _____.
8. El back-end debe incluir la librería con nombre _____, versión _____.
9. El back-end debe trabajar con el servidor web con nombre _____, versión _____.
10. El servidor web donde se desplegará el back-end debe tener el sistema operativo con nombre _____, versión _____.
11. La aplicación a desarrollar debe trabajar con el sistema gestor de base de datos con nombre _____, versión _____.
12. La aplicación a desarrollar debe trabajar con el proveedor de servicios en la nube con nombre _____.

En la tabla 7 se muestra una práctica que busca establecer un registro de eventos para el producto software a desarrollar, así como establecer los niveles que este usará, la codificación de los eventos, y el formato de la información desplegada por el registro de eventos.

Tabla 7. Práctica para establecer el registro de eventos y sus lineamientos.

5				
Directriz para capturar los RMs	1. Determine si es necesario un registro de eventos (logs) para la aplicación. 2. Si es necesario el registro de eventos, establezca: a. La descripción de los niveles de eventos para el sistema, y qué eventos están asociados a cada nivel. b. Los lineamientos para la codificación de los eventos del sistema. c. El formato de la información del registro de eventos.			
Consideraciones	1. Un evento es una acción u ocurrencia que puede ser identificada por el sistema y que tiene importancia para el mismo. Un evento puede ser un error producido en el sistema, una advertencia, una funcionalidad ejecutada, entre otros. Algunos ejemplos de eventos son: inicio de sesión, problemas en la disponibilidad de la base de datos, errores al ejecutar un pago. 2. Un log o registro de eventos es la documentación producida automáticamente de eventos relevantes para un sistema en particular, la cual va acompañada de un sello de tiempo o time stamp. 3. Un evento está asociado a un nivel, estos niveles varían dependiendo de las librerías que se utilizan para este propósito. Los niveles más comunes son: DEBUG, TRACE, INFO, WARN, FATAL y ERROR. 4. Un evento puede ser clasificado mediante un código. El código puede estar constituida prefijos, sufijos, rangos, patrones entre otros. Ejemplos de codificación de eventos pueden ser: ADV-900 para advertencias, ERRLOGIN-1022 para errores de login, de 0 a 349 errores de pagos, entre otros. 5. El formato de la información del registro de eventos se refiere a cómo debe ser descrito un evento y su nivel de especificidad en el código fuente. En el formato se define si debe ir el nombre de la funcionalidad, el nombre del módulo, el stack trace, una combinación de estas u otros elementos. Un ejemplo de un formato es: Nombre de la funcionalidad, nombre del módulo, excepción generada, stack trace.			
Sub características de mantenibilidad abordadas	Modularidad, capacidad de ser modificado			
Elementos abordados:	Propensión a defectos, propensión al cambio, completo	**Sub elementos abordados:**	No aplica	
Forma en la cual se potencia la mantenibilidad mediante los RMs generados	Los RMs generado a través de esta práctica busca estandarizar la manera en que se identifica un evento, la manera en que se clasifica mediante un código, la información que constituye a un evento y los niveles que el registro de eventos tendrá. Esto con la finalidad de que el registro de eventos sea efectivo y uniforme para así ayudar en la identificación, el diagnóstico y el tiempo de la resolución de cualquier tipo de evento.			
Roles involucrados	Líder de desarrollo, arquitecto de soluciones			
Ubicación final de los RMs				
Tipo de requisito	**Temporal**	☐	**Atemporal**	☐
RMs generados				

1. Se deben almacenar los eventos de la aplicación en un registro histórico de eventos.
2. La codificación de los eventos debe seguir los lineamientos planteados en el documento con nombre _____, con versión _____, que se encuentra en el siguiente enlace o ubicación _____
3. El formato de la información de los eventos debe estar detallado en el documento con nombre _____, con versión _____, que se encuentra en el siguiente enlace o ubicación _____.
4. La descripción de los niveles y cuales eventos están asociados a cada nivel debe estar registrada en el documento con nombre _____, con versión _____, que se encuentra en el siguiente enlace o ubicación _____.

En la tabla 8 se muestra una práctica para establecer los lineamientos y escenarios durante el envío de los logs locales de una aplicación de escritorio a un servidor.

Tabla 8. Práctica para establecer envíos de logs locales de app de escritorio a un servidor.

6				
Directriz para capturar los RMs	1. Si es el producto software a desarrollar es una aplicación de escritorio, determine si es necesario enviar la información del registro local de eventos (logs) a un servidor. 2. Si es necesario, establezca los escenarios en que estos logs locales van a ser enviados al servidor.			
Consideraciones	1. Normalmente los logs de las aplicaciones de escritorio son almacenados localmente, por lo tanto, cuando hay un evento como un error, una advertencia, o un mal comportamiento de la aplicación, el equipo de desarrollo no tendrá la información de los eventos debido a que estarán en el computador del usuario donde ocurrieron. La solución a esto, es enviar los logs a un servidor, para que el equipo de desarrollo pueda ayudar en la identificación, el diagnóstico y el tiempo de la resolución de cualquier tipo de evento. 2. Es posible enviar toda la información almacenada de los diferentes niveles de los eventos, enviar información correspondiente a un determinado evento, a un nivel, generada durante determinadas franjas de tiempo o demás lineamientos que el equipo pueda establecer. 3. Puede apoyarse en la práctica número 5 para entender y establecer niveles, formato y codificación de los logs.			
Sub características de mantenibilidad abordadas	Modularidad, capacidad de ser modificado			
Elementos abordados:	Propensión a defectos, propensión al cambio	**Sub elementos abordados:**	No aplica	
Forma en la cual se potencia la mantenibilidad mediante los RMs generados	Los RMs generado a partir de esta práctica busca mediante un registro de eventos ayudar en la identificación, el diagnóstico y el tiempo de la resolución de cualquier tipo de evento.			
Roles involucrados	Líder de desarrollo, arquitecto de soluciones, despliegue			
Ubicación final de los RMs				
Tipo de requisito	**Temporal**	☐	**Atemporal**	☐
RMs generados				
1. La aplicación de escritorio debe enviar la información del registro de eventos local a un servidor. 2. La frecuencia de envío de los eventos debe estar determinada por los lineamientos planteados en el documento con nombre _____, versión _____, que se encuentra en el siguiente enlace o ubicación _____.				

En la tabla 9 se muestra una práctica que busca tempranamente la independencia de la presentación de la lógica de negocio, así como también la independencia de la lógica del negocio con el medio de persistencia.

Tabla 9. Práctica para establecer independencia de lógica de negocio, medio de persistencia y presentación.

7				
Directriz para capturar los RMs	Determine con el cliente final si durante el ciclo de vida del producto software a desarrollar: a. La presentación del producto software va a cambiar, o se van a añadir más tipos de presentación al producto. b. El medio de persistencia del producto software va a cambiar o se van a añadir más medios de persistencia al producto.			
Consideraciones	1. La presentación de un producto software es la parte con la que los usuarios interactúan. Corresponde a todo lo que el usuario ve cuando usa el producto, desde fuentes y colores hasta menús desplegables y controles deslizantes. Los diferentes tipos de presentación de un producto pueden ser: la terminal de un sistema operativo, apps para relojes, celulares y televisores inteligentes, apps para consolas de video juegos, páginas web, entre otras. 2. La lógica de negocio de un producto software es la encargada de ejecutar todas las diferentes reglas de negocio con el propósito de satisfacer una acción realizada por el cliente de la aplicación. 3. El medio de persistencia es donde la aplicación almacena de manera estructurada sus datos. Los medios de persistencia pueden ser: archivos de texto, archivos binarios, bases de datos relacionales, bases de datos no relacionales, entre otras.			
Sub características de mantenibilidad abordadas	Modularidad, reusabilidad, analizabilidad, capacidad de ser modificado, capacidad de ser probado			
Elementos abordados:	Acoplamiento, cohesión, líneas de código, tamaño, flexibilidad, duplicación de código, capacidad de expansión, interfaces, polimorfismo	**Sub elementos abordados:**	Tamaño del diseño, cantidad de sentencias de código, número de módulos, uso de interfaces en vez de la implementación, segregación de interfaces	
Forma en la cual se potencia la mantenibilidad mediante los RMs generados	Los RMs generados a partir de esta práctica buscan aumentar la facilidad de expansión del producto software, debido a que es posible agregar varios tipos de presentación conservando la misma lógica de negocio y además es posible cambiar el tipo de medio de persistencia sin cambiar la lógica de negocio implementada. Por otra parte se favorece el bajo acoplamiento entre componentes más propensos a cambios, principalmente el medio de persistencia y la presentación del producto.			
Roles involucrados	Líder de desarrollo, Arquitecto de soluciones, expertos en la materia			
Ubicación final de los RMs				
Tipo de requisito	**Temporal**	☐	**Atemporal**	☐
RMs generados				
1. La lógica de negocio del producto debe ser independiente de la presentación con la cual interactúa el usuario, de tal manera que cuando se agreguen nuevos tipos de presentación, o se modifique la tecnología de la presentación, sea posible utilizar la lógica de negocio del producto sin estar obligado a modificarla. 2. La lógica de negocio del producto software debe ser independiente del medio de persistencia, de tal manera que cuando se cambie el medio de persistencia o la tecnología del medio de persistencia, la lógica de negocio del producto no cambie.				

En la tabla 10 se muestra una práctica que busca establecer mensajes significativos al usuario cuando ocurre un problema en el sistema, y la identificación del uso de mensajes únicos con sus respectivos códigos de error.

Tabla 10. Práctica para establecer mensajes significativos.

8				
Directriz para capturar los RMs	Determine: a. Si cuando ocurre un error causado por problemas del sistema al usuario, se le debe mostrar el código del error y un mensaje significativo con el propósito de que el usuario comprenda el contexto del error. b. Si cada uno de los códigos de error debe tener un único mensaje de error significativo asociado.			
Consideraciones	1. Para realizar esta práctica se debe previamente haber realizado la práctica número 5 o haber establecido una codificación interna de eventos, especialmente para los eventos de error.			
Sub características de mantenibilidad abordadas	Capacidad de ser modificado, modularidad			
Elementos abordados:	Propensión a defectos, propensión al cambio, completo	**Sub elementos abordados:**	No aplica	
Forma en la cual se potencia la mantenibilidad mediante los RMs generados	Los RMs generados a partir de esta práctica buscan brindarle al usuario de la aplicación el contexto del error del sistema generado e información que pueda compartir con el equipo de soporte, con el propósito de que el equipo de soporte pueda dar una respuesta rápida en lugar de buscar cuál de las muchas razones pudo haber causado el problema.			
Roles involucrados	Líder de desarrollo, expertos en la materia			
Ubicación final de los RMs				
Tipo de requisito	**Temporal**	☐	**Atemporal**	☐
RMs generados				
1. Cuando ocurra un error causado por problemas del sistema, al usuario final se le debe mostrar el código del error y un mensaje significativo con el propósito de que el usuario comprenda el contexto del error. 2. Cada uno de los códigos de error causados por problemas del sistema debe tener un único mensaje de error asociado.				

En la tabla 11 se muestra una práctica que busca establecer componentes desarrollados previamente por la empresa con el propósito de reutilizarlos en el producto software a desarrollar.

Tabla 11. Práctica para establecer componentes desarrollados previamente por la empresa que se van a reutilizar.

9				
Directriz para capturar los RMs	Establezca el nombre de los componentes desarrollados previamente por la empresa que se van a reutilizar en este producto software			
Consideraciones	1. El componente a reutilizar debe haber sido testeado y validado previamente con el propósito de garantizar que sus requisitos funcionales y no funcionales se cumplan. Entendiendo un componente testeado como aquel que ha pasado por el departamento de pruebas y ha cumplido sus requisitos asociados, por otra parte, un componente validado es aquel que ha pasado por el departamento de producto y/u operaciones y ha validado que cumple la funcionalidad tal y como la pidió el cliente. 2. Se debe tomar en cuenta los parámetros de entradas y salidas del componente a reutilizar para determinar si se ajustan a la necesidad del producto software a desarrollar. 3. Inspeccionar el código fuente implica identificar y revisar donde se encuentra implementada una determinada funcionalidad con el propósito de corregir un error, agregar nuevas funcionalidades o adaptar funcionalidades.			
Sub características de mantenibilidad abordadas	Modularidad, reusabilidad, analizabilidad, capacidad de ser modificado y capacidad de ser probado.			
Elementos abordados:	Reuso de algoritmos, duplicación de código, no redundante, acoplamiento, dependencias	**Sub elementos abordados:**	Acoplamiento lógico, dependencia cíclica	
Forma en la cual se potencia la mantenibilidad mediante los RMs generados	El RM generado a través de esta práctica busca identificar y reutilizar componentes desarrollados en proyectos previos que han sido testeados y validados, con el fin de disminuir la duplicación de código en el producto software a desarrollar por la empresa, y reducir la cantidad de código que se debe inspeccionar.			
Roles involucrados	Líder de desarrollo, arquitecto de soluciones			
Ubicación final de los RMs				
Tipo de requisito	**Temporal**	☐	**Atemporal**	☐
RMs generados				
1. El componente con nombre _____ debe ser reutilizado en el producto software para cumplir con el siguiente propósito: _____. Está implementado con las siguientes tecnologías y versiones: _____, _____ . El componente y la documentación que indica su funcionamiento, está alojado en el repositorio: _____.				

En la tabla 12 se muestra una práctica que busca establecer los componentes que deben ser desarrollados de tal manera que puedan ser reutilizados en futuros proyectos.

Tabla 12. Práctica para establecer componentes que deben ser desarrollados de tal manera que puedan ser reutilizados en futuros proyectos.

10				
Directriz para capturar los RMs	Establezca los componentes que deben ser desarrollados de tal manera que puedan ser reutilizados en futuros proyectos.			
Consideraciones	Para el componente a desarrollar que va a ser reutilizado en futuros proyectos se debe considerar los siguientes aspectos: 1. Se debe plantear una captura de requisitos funcionales y no funcionales para el componente en particular. 2. Se debe indicar la forma en la cual se reutilizará el componente, ya sea biblioteca, librería, framework, una clase, entre otras. 3. Se debe indicar la forma tecnológica en la cual se podrá utilizar, instalar o desplegar el componente, por ejemplo, en java se podrá utilizar archivos .jar, .war, en python archivos .py 4. Se debe indicar los servicios que ofrece junto con sus entradas, descripción de la salida que se genera y el nombre de dichos servicios 5. Se debe crear una documentación adecuada que facilite su búsqueda en repositorios de componentes, evaluación, adaptación a nuevos entornos, integración con otros componentes y acceso a información de soporte. 6. La última parte del RM que plantea la descripción de los servicios que ofrece el componente se obtienen con los requisitos funcionales y no funcionales indicados en los requisitos que se hayan levantado hasta el momento. 7. Inspeccionar el código fuente implica identificar y revisar donde se encuentra implementada una determinada funcionalidad con el propósito de corregir un error, agregar nuevas funcionalidades o adaptar funcionalidades.			
Sub características de mantenibilidad abordadas	Modularidad, reusabilidad, analizabilidad, capacidad de ser modificado y capacidad de ser probado.			
Elementos abordados:	Reuso de algoritmos, duplicación de código, cohesión, no redundante	**Sub elementos abordados:**	No aplica	
Forma en la cual se potencia la mantenibilidad mediante los RMs generados	El RM generado a través de esta práctica busca desde una etapa temprana como es el análisis, identificar componentes que deben ser reutilizados en futuros proyectos de software, los cuales al final del desarrollo serán testeados, validados y expondrán unas interfaces que permitirán su reutilización, de esta forma se busca disminuir el código duplicado, reducir la cantidad de código que se debe inspeccionar y facilitar la reutilización de código fuente en futuros proyectos de software.			
Roles involucrados	Arquitecto de soluciones y líder de desarrollo			
Ubicación final de los RMs				
Tipo de requisito	**Temporal**	☐	**Atemporal**	☐
RMs generados				

1. El componente con nombre _____ debe ser implementado de tal manera que pueda ser reutilizado en futuros proyectos. El componente está constituido por los siguientes requisitos funcionales identificados con los ID: _____, _____.
2. El componente con nombre _____ debe estar implementado con las siguientes tecnologías y versiones: _____, _____
3. La forma en la cual se debe reutilizar el componente con nombre _____ es: _____.
4. La forma tecnológica en la cual se debe utilizar, instalar o desplegar el componente con nombre _____es: _____
5. El componente con nombre _____ y su documentación que indica su funcionamiento debe estar alojado en el repositorio: _____.
6. El componente con nombre _____ debe ofrecer los siguientes servicios:
Nombre del servicio: _____.
Entradas al servicio: _____, _____.
Descripción de la salida que genera el servicio: _____.

En la tabla 13 se muestra una práctica que busca establecer componentes que exponen funcionalidades reutilizables dentro del producto software a desarrollar.

Tabla 13. Práctica para establecer componentes que exponen funcionalidades reutilizables dentro del producto software a desarrollar.

11				
Directriz para capturar los RMs	Establezca el componente que va a ser reutilizado dentro del producto software a desarrollar			
Consideraciones	1. Se debe determinar aquellos requisitos funcionales que compartan características en común y que puedan ser encapsuladas en un componente, las cuales pueden ser reutilizables por otros componentes dentro del producto software. 2. Se debe indicar el nombre del servicio, sus entradas y la descripción de la salida. Estos datos son necesarios porque permiten a la empresa una fácil búsqueda para futuros proyectos de la empresa. 3. Inspeccionar el código fuente implica identificar y revisar donde se encuentra implementada una determinada funcionalidad con el propósito de corregir un error, agregar nuevas funcionalidades o adaptar funcionalidades. 4. Para el componente planteado puede establecer un índice de mantenibilidad siguiendo los lineamientos planteados en la práctica número 3.			
Sub características de mantenibilidad abordadas	Modularidad, reusabilidad, analizabilidad, capacidad para ser modificado, capacidad para ser probado			
Elementos abordados:	Encapsulamiento, flexibilidad, reuso de algoritmos, interfaces, duplicación de código, acoplamiento, cohesión, no redundante	**Sub elementos abordados:**	No aplica	
Forma en la cual se potencia la mantenibilidad mediante los RMs generados	El RM generado a través de esta práctica busca identificar componentes que exponen funcionalidades reutilizables dentro del producto software a desarrollar. Estos componentes permitirán disminuir el código duplicado, el acoplamiento, la cantidad de código que se debe inspeccionar, aumentar la cohesión y facilitar la reutilización de código fuente del producto de software.			
Roles involucrados	Arquitecto de soluciones y líder de desarrollo			
Ubicación final de los RMs				
Tipo de requisito	**Temporal**	☐	**Atemporal**	☐
RMs generados				
1. El componente con nombre _____ debe ser implementado de tal manera que debe poder ser reutilizado dentro del producto software. El componente está constituido por los siguientes requisitos funcionales identificados con los ID: _____, _____. 2. El componente con nombre _____ debe estar implementado con las siguientes tecnologías y versiones: _____, _____. 3. El componente con nombre _____ debe ofrecer los siguientes servicios: Nombre del servicio: _____. Entradas al servicio: _____, _____ Descripción de la salida que genera el servicio: _____.				

En la tabla 14 se muestra una práctica que busca que el producto software a implementar, sea desarrollado de tal manera que al cambiar el sistema externo con el cual inicialmente interopera, por otro sistema externo que ofrezca el mismo servicio, no haya necesidad de afectar la lógica del negocio del producto.

Tabla 14. Práctica para independencia de la lógica de negocio con diferentes sistemas que ofrezcan el mismo servicio.

12				
Directriz para capturar los RMs	Determine: a. Si el producto software a desarrollar se va a comunicar con un sistema externo. b. Si a corto o mediano plazo, la comunicación se va a realizar con un sistema externo diferente, pero que ofrezca el mismo servicio.			
Consideraciones	1. Es posible que a corto o mediano plazo la comunicación se realice con otro sistema externo que cumpla el mismo objetivo por diferentes razones tales como: inestabilidad técnica del sistema externo, caducidad o cambios en contratos para la integración con el sistema externo, entre otras. 2. La lógica de negocio de un producto software es la encargada de ejecutar todas las diferentes reglas de negocio con el propósito de satisfacer una acción realizada por el cliente de la aplicación. 3. Una interface para la comunicación entre sistemas es un mecanismo que a través de rutinas, protocolos y definiciones permite la comunicación con sistemas externos, logrando así que sistemas heterogéneos intercambien información.			
Sub características de mantenibilidad abordadas	Modularidad, reusabilidad, capacidad de ser modificado			
Elementos abordados:	Abstracción, Propensión al cambio, interfaces, polimorfismo	**Sub elementos abordados:**	Uso de interfaces en vez de la implementación, Abstracción funcional	
Forma en la cual se potencia la mantenibilidad mediante los RMs generados	El RM generado a partir de esta práctica busca que el producto software a implementar, sea desarrollado de tal manera que al cambiar el sistema externo con el cual inicialmente interopera, por otro sistema externo que ofrezca el mismo servicio, no haya necesidad de afectar la lógica del negocio del producto, favoreciendo así la reusabilidad y la capacidad de ser modificado.			
Roles involucrados	Expertos en la materia, Arquitecto de soluciones			
Ubicación final de los RMs				
Tipo de requisito	**Temporal**	☐	**Atemporal**	☐
RMs generados				
1. La interface para la comunicación con el sistema externo con nombre _____, debe ser implementada de tal manera que al cambiar el sistema externo por otro que ofrezca el mismo servicio no se requiera cambiar la lógica del negocio del producto.				

4.3.3. Elementos de mantenibilidad potenciados en las prácticas propuestas

Con las prácticas propuestas se logró potenciar un total de 31 elementos de mantenibilidad (ver sección 3.2), los cuales equivalen a un 86% del total de elementos de mantenibilidad, y 44 sub elementos de mantenibilidad, los cuales equivalen a un 72% del total de sub elementos de mantenibilidad caracterizados en el capítulo 3.

En este sentido, el 14% de los elementos mantenibilidad y el 28% de sub elementos de mantenibilidad no se pudieron potenciar mediante RMs desde la etapa de análisis por dos razones principales. La primera es que algunos de estos elementos sólo podrían ser potenciados con RMs de proceso y no de producto como por ejemplo el elemento de recursos del proyecto y sus sub elementos asociados. La segunda razón se debe a que varios de estos elementos y sub elementos se pueden logran en las etapas de diseño e implementación, por ejemplo, cantidad de clases usadas en tiempo de ejecución y cantidad de métodos invocados en tiempo de ejecución deben ser abordadas desde la etapa de desarrollo.

4.3.4. Como los RM generados por las prácticas potencian las sub características de mantenibilidad de la ISO 25010

En la figura 4, se muestra un gráfico radial en el que se puede evidenciar cuantas veces cada una de las sub características de mantenibilidad es potenciada por las prácticas propuestas. En este sentido cada sub característica de mantenibilidad puede ser potenciada 12 veces, ya que 12 es el total de prácticas que se presentaron. La sub característica que más se cubre con las prácticas es la capacidad de ser modificado con un total de 12, lo cual quiere decir que el 100% de las prácticas potencian individualmente a la capacidad de ser modificado. La segunda sub característica que más se cubre es la modularidad con un total de 10, lo cual quiere decir que el 83% de las prácticas potencian individualmente a la modularidad. La siguiente sub característica más cubierta es la reusabilidad con un total de 8, lo cual quiere decir que el 66% de las prácticas potencias individualmente a la reusabilidad. Y las últimas sub características analizabilidad y capacidad para ser probado llevan la misma frecuencia de 7, lo cual quiere decir que el 58% de las prácticas las potencian individualmente.

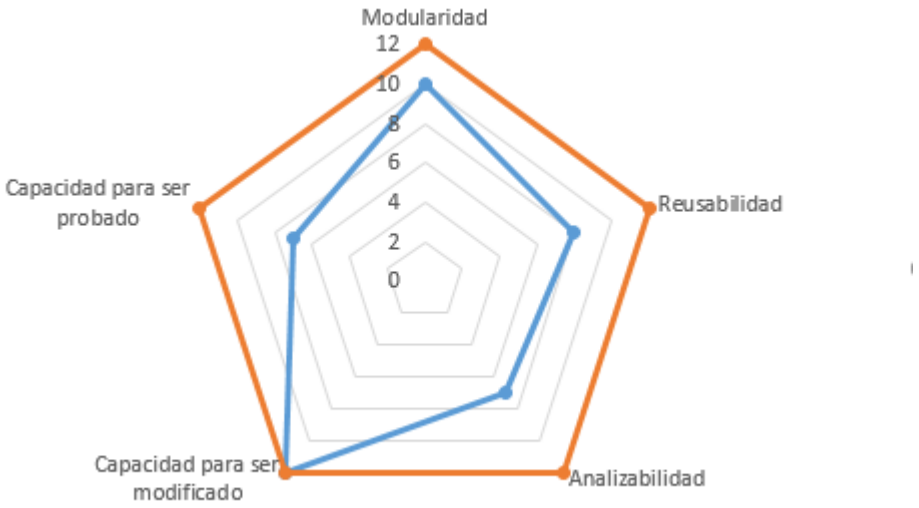


Figura 4. Cómo las prácticas potencian a las sub características de mantenibilidad. Autor fuente propia.

4.4.Conclusiones encontradas

- Se identificó que el RM planteado en la práctica 12, puede ser también considerando un requisito de interoperabilidad, y los RMs planteados en la práctica 8 pueden ser considerados como requisitos de usabilidad. En este sentido es posible que un requisito no funcional esté asociado a 2 características de calidad.
- Se identificó que las prácticas 1, 2, 3, 4, 7, 9 y 12 generan requisitos de mantenibilidad que plantean restricciones sobre la estructura interna del producto software a desarrollar, las prácticas 6 y 8 generan RM que plantean restricciones sobre el comportamiento del producto software a desarrollar, y las prácticas 5, 10 y 11 generan RMs que plantean restricciones sobre la estructura interna y el comportamiento del producto software a desarrollar.
- Dependiendo de las empresas y equipos de desarrollo, algunos RMs generados a partir de las prácticas propuestas pueden ser aplicados en uno o varios proyectos de software de las empresas o equipos de desarrollo.

Capítulo 5 – Evaluación de la propuesta

En este capítulo se presenta la aplicación y evaluación de las prácticas desarrolladas a partir de un estudio de caso, incluyendo el contexto de la investigación, resultados y análisis. Para la evaluación de las prácticas fue seleccionada la metodología de estudio de caso debido a que según Yin [94] es un método que estudia un fenómeno en su contexto real, buscando mantener la integridad y las características significativas de los eventos.

Los estudios de caso han sido desarrollados siguiendo el protocolo de Runenson y Host descrito en [17] y es de tipo simple holístico según el enfoque presentado en [94], debido a que la guía fue aplicada a una aplicación web la cual corresponde a la unidad de análisis.

A continuación, se describe el protocolo del estudio de caso en términos de: objetivo, objeto, aspectos a evaluar, contexto de las organizaciones participantes, criterio de selección de las organizaciones, preguntas de investigación, indicadores, instrumentos de evaluación, sujetos de investigación, procedimiento de campo, recolección de datos, análisis, resultados de la intervención, validez y limitaciones.

5.1.Diseño del estudio de caso

Objetivo: evaluar la idoneidad de las prácticas en términos de su utilidad, conocimiento adquirido por los stakeholders y correctitud.

Objeto: el objeto del estudio de caso son las prácticas para la captura y especificación de RM durante el desarrollo de un nuevo producto software.

Aspecto evaluado: el estudio de caso pretende evaluar la idoneidad de las prácticas propuestas en términos de: (i) utilidad, hace referencia a si las prácticas permiten capturar y especificar RMs desde la etapa de análisis para que de esta manera se pueda abordar tempranamente la mantenibilidad del producto software a desarrollar.(ii) conocimiento adquirido por los stakeholders, hace referencia a si los stakeholders (clientes, analistas y desarrolladores) lograron adquirir nuevo conocimiento asociado a conceptos básicos de mantenibilidad, elementos de mantenibilidad que conforman a un producto software, importancia de abordar la mantenibilidad de manera temprana en el desarrollo de un nuevo producto software y cómo capturar y especificar RMs desde la etapa de análisis. (iii) correctitud, hace referencia a si los nombres de los campos y su descripción son suficientes y

están correctamente planteados con el propósito de capturar y especificar RMs desde la etapa de análisis.

Preguntas de investigación: las preguntas de investigación principales y adicionales que apoyan este mecanismo de evaluación preliminar se describen en la tabla 15.

Tabla 15. Preguntas de investigación del estudio de caso.

Tipo	**Preguntas**
Principal	¿Las prácticas son idóneas para capturar y especificar requisitos de mantenibilidad desde la etapa de análisis de un producto software?
Secundarias (PS1)	¿Los campos que constituyen las prácticas son útiles para la captura y especificación de RM?
Secundarias (PS2)	¿Las prácticas propuestas permiten que los stakeholders adquieran conocimiento acerca de los RM?
Secundarias (PS3)	¿Los RMs especificados siguen el orden y la naturaleza de la información solicitada por la notación propuesta por las prácticas?

Contexto y unidades de análisis: los RMs se aplicaron desde la etapa de análisis correspondiente al desarrollo de un producto software que soporta las actividades de la Oficina de Control Interno (OCI) de la Universidad del Cauca. Especialmente el producto software debía permitir la gestión de los usuarios que pueden hacer uso de la aplicación, la gestión de los planes de mejora de la Universidad del Cauca, la auditoria en los planes de mejora en ejecución y la generación de estadísticas de los planes de mejora finalizados y en ejecución.

La OCI busca en un futuro poder hacer sus diferentes actividades de forma sistematizada, ya que por el momento la gestión de planes de mejora se realiza de forma manual mediante el uso de hojas de cálculo. Los requisitos funcionales de la aplicación se encuentran en el anexo 1.
En este sentido, el estudio de caso se aprovechó de la actual necesidad y fueron definidos 24 RMs, de los cuales únicamente 10 fueron implementados por cuestiones de tiempo, disponibilidad y conocimiento del desarrollador. Los RMs lograron aumentar la capacidad de mantenimiento del producto software.

Criterio de selección: los stakeholders buscaban que la aplicación a desarrollar tuviera una alta mantenibilidad. Aunado a esto, el equipo de desarrollo está conformado por estudiantes de entre 8vo y 10mo semestre los cuales no tienen mucha experiencia en el desarrollo de software y la mantenibilidad no es un punto de vital importancia en sus desarrollos. En este sentido con las prácticas se busca que el equipo de desarrollo aprenda a capturar, y especificar RMs desde

la etapa de análisis, también a entender la importancia de abordar la mantenibilidad tempranamente y sus conceptos más importantes

Instrumentos de evaluación: dentro de esta investigación los instrumentos empleados aportan datos que posteriormente ayudarán a contribuir en la mantenibilidad del sistema. Para esta investigación, los instrumentos seleccionados son:

- **Observación de campo:** este instrumento permitió a los investigadores observar el comportamiento de los analistas durante el proceso de la captura y especificación de los RMs generados a partir de las prácticas.
- **Encuesta:** este instrumento permitió obtener información cuantitativa y cualitativa acerca de los RMs generados y de las prácticas especificadas para capturarlos y especificarlos.
- **RMs especificados a partir de la notación propuesta por las prácticas:** este instrumento corresponde a los diferentes RMs especificados que siguen la notación planteada en el campo número 10 de las prácticas (ver sección 4.3.1).
- **Componente desarrollado:** este instrumento corresponde a un componente que fue desarrollado el cual satisface algunos RM.

Indicadores y métricas : en la tabla 16 se muestran las preguntas de investigación junto con el conjunto de indicadores y métricas para dar respuesta a cada una de ellas.

Tabla 16. Conjunto de indicadores y métricas.

Preguntas de investigación	**Indicadores**	**Tipo de indicador**	**Mediciones**	**Instrumentos**
¿Los campos que constituyen las prácticas son útiles para la captura y especificación de RM?	Tiempo de captura	Cuantitativo	Tiempo utilizado para capturar los RM que la práctica genera	Observación de campo
	Tiempo de especificación	Cuantitativo	Tiempo utilizado para especificar los RM que la práctica genera	Observación de campo
	Total de RMs capturados y especificados	Cuantitativo	Cantidad de RMs que se capturaron y especificaron	RMs especificados a partir de la notación propuesta por las prácticas
	Facilidad de entendimiento	Cualitativo	Facilidad de entendimiento de cada uno de los campos	Observación de campo Encuesta

			de las prácticas para capturar y especificar RMs	
	Total de prácticas que generaron RMs a considerar en el desarrollo	Cuantitativo	Cantidad de prácticas que generaron RMs a considerar en el desarrollo del producto software	RMs especificados a partir de la notación propuesta por las prácticas
	Total prácticas que no generaron RMs a considerar en el desarrollo	Cuantitativo	Cantidad de prácticas que no generaron RMs a considerar en el desarrollo del producto software	RMs especificados a partir de la notación propuesta por las prácticas
	Cumplimiento de RMs	Cualitativo	Grado en el cual los componentes desarrollados cumplen con los RM	Componente desarrollado
¿Las prácticas propuestas permiten que los stakeholders adquieran conocimiento acerca de los RM?	Conceptos no entendidos	Cualitativo	Conceptos propuestos en las prácticas que no fueron entendidos por los stakeholders	Observación de campo Encuesta
¿Los RMs especificados siguen el orden y la naturaleza de la información solicitada por la notación propuesta por las prácticas?	Total de RMs que no siguen el orden y naturaleza de la información	Cuantitativo	Cantidad de RMs que no siguen el orden y la naturaleza de la información solicitada por la notación propuesta.	RMs especificados a partir de la notación propuesta por las prácticas
	Cantidad de RMs que siguen el orden y la naturaleza de la información	Cuantitativo	Cantidad de RMs que siguen el orden y la naturaleza de la información solicitada por la notación propuesta.	RMs especificados a partir de la notación propuesta por las prácticas
	Elementos nuevos a las notaciones de los RMs	Cualitativo	Elementos nuevos agregados a las notaciones para especificar los RMs.	Observación de campo Encuesta
	Campos nuevos en las prácticas	Cualitativo	Campos nuevos agregados a las prácticas que ayuden a la captura y especificación de RMs.	Observación de campo Encuesta

Sujetos de investigación: en cuanto a los sujetos de investigación, el equipo de investigación estuvo compuesto por:

- Un analista, encargado de capturar los RMs generados de acuerdo al producto software que se quiere construir.

- Tres clientes, los cuales son las personas que van a utilizar el producto software a desarrollar.
- Dos investigadores, encargados de observar al analista durante la captura y especificación de los RMs generados a partir del producto software a desarrollar.
- Un desarrollador, encargado de implementar un componente software teniendo en cuenta los RMs que genero el analista.

Planeación del proceso de intervención:

- **Contextualización de la propuesta:** fue presentado el propósito de la investigación a los clientes, analista y equipo de desarrollo involucrado en el producto software asociada a las actividades de la OCI (ver sección 5.1).
- **Capacitación detallada de las prácticas propuestas:** al inicio de la capacitación se presentaron los conceptos principales involucrados en la mantenibilidad del software, los elementos de mantenibilidad identificados en la literatura y la importancia de capturar RMs de manera temprana. Además, se explicaron cada uno de los campos de las prácticas propuestas para poder capturar y especificar RMs exitosamente.
- **Aplicación de las prácticas para el producto:** los clientes**,** analista y desarrollador utilizaron las prácticas propuestas para capturar y especificar RMs los cuales se considerarían en el desarrollo del producto software.
- **Implementación de una parte del producto software:** el desarrollador implementó una serie de componentes los cuales cumplían con un conjunto de RMs capturados, con el propósito de establecer si era factible satisfacer los RMs identificados de manera temprana.
- **Entrega de los componentes:** el analista y desarrollador entregaron los componentes desarrollados a los investigadores junto con las experiencias y lecciones aprendidas. Posteriormente se tuvo una reunión con los clientes donde se mostró y entregó los componentes desarrollados en el estudio de caso.
- **Cierre del proceso:** los clientes, analista y desarrollador diligenciaron una encuesta para dar por finalizado el proceso.

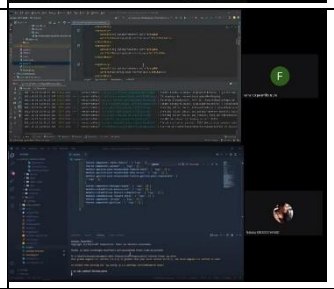
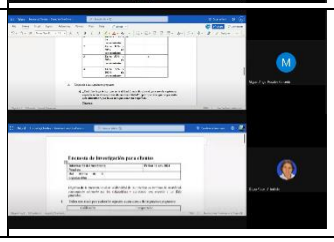
5.2.Intervención y recolección de datos a través del estudio de caso.

Lugar en el cual se desarrolló el estudio de caso

El estudio de caso fue ejecutado un entorno virtual mediante la plataforma de Google Meet. En la tabla 17 se muestra la ejecución del proceso de intervención del estudio de caso.

Tabla 17. Ejecución del plan de intervención.

Actividad	**Fecha**	**Registro fotográfico**
Contextualización de la propuesta	15 de octubre del 2021	
Capacitación detallada de las prácticas propuestas:	15 de octubre del 2021	

Aplicación de las prácticas para el producto	19, 21 octubre del 2021	
Implementación de una parte del producto software	21 octubre al 2 de noviembre 2021	
Entrega de los componentes	11 noviembre 2021	 
Cierre del proceso	11 noviembre 2021	

Participantes de la intervención

En el estudio de caso los participantes fueron Miguel Rosales, cliente de la OCI, en el cargo de técnico administrativo, con experiencia de 5 años; Deysi Potosí, cliente de la OCI, en el cargo de profesional universitaria, con experiencia de 20 años; Pablo Mage, ingeniero en electrónica y telecomunicaciones, en el cargo de docente de la universidad del Cauca, con experiencia de 21 años; María Fernanda Jaramillo, en el cargo de analista y desarrolladora, con experiencia de 6 meses. Todos los participantes conocían el funcionamiento de la gestión de control interno de la Universidad del Cauca, pero ninguno sabía qué eran los requisitos de mantenibilidad ni mucho menos como capturarlos de acuerdo a las necesidades del proyecto.

Medidas obtenidas a partir de la intervención

En el anexo 2 se encuentran las encuestas diligenciadas por los clientes y el analista. En la tabla 18 se muestran las métricas obtenidas a través del estudio de caso:

Tabla 18. Métricas obtenidas del estudio de caso

Métricas obtenidas a través del uso de las prácticas	
Métricas	**Estudio de caso**
Tiempo promedio para capturar los RMs que genera cada práctica.	5 min
Tiempo promedio para especificar los RMs que genera cada práctica.	7 ½ min
Total de RMs capturados y especificados	24
Cantidad de prácticas que generaron RMs a considerar en el desarrollo del producto software	8
Total, prácticas que no generaron RMs a considerar en el desarrollo	4
Grado en el cual los componentes desarrollados cumplen con los RM	Los componentes desarrollados cumplieron con el 100% de los RMs asignados a para ellos.
Cantidad de RMs que no siguen el orden y la naturaleza de la información solicitada por la notación propuesta.	1
Cantidad de RMs que siguen el orden y la naturaleza de la información solicitada por la notación propuesta.	24
Elementos nuevos agregados a las notaciones para especificar los RMs.	1 (Implementación de mejora 1, ver sección 5.6.1)

5.3. Requisitos de mantenibilidad capturados

El analista junto con los clientes y equipo de desarrollo capturaron 24 RMs teniendo en cuenta toda la aplicación. A continuación, se muestran los RMs capturados y especificados junto con la práctica que los generó:

Práctica 1: Práctica para establecer lineamientos en la codificación del software

RMs generados:

1. El código fuente del sistema a desarrollar debe seguir los lineamientos planteados en el documento o estándar con nombre Estándar_codificación_app, con versión 1.0, que se encuentra en el siguiente enlace o ubicación Estándar_codificación_app.
2. **Nota:** ver anexo 3 donde se encuentran los lineamientos para escribir el código fuente.

Práctica 2: Práctica para establecer lineamientos para la documentación del código fuente.

RMs generados:

3. La documentación del código fuente del sistema a desarrollar debe seguir los lineamientos planteados en el documento o estándar con

nombre Estándar_documentacion_app, con versión 1.0, que se encuentra en el siguiente enlace o ubicación Estándar_documentacion_app.

4. **Nota:** ver anexo 4 donde se encuentran los lineamientos para la documentación del código fuente.

Práctica 3: Práctica para establecer puntaje de mantenibilidad.

RMs generados:

5. El módulo con nombre gestión de planes de mejora que se compone de los siguientes requisitos funcionales identificados con los ID: HU-09, HU-11, HU-13, HU-15, HU-23, HU-28 debe tener un puntaje de mantenibilidad que esté entre A y B. El puntaje de mantenibilidad ha sido planteado porque el módulo cumple con las siguientes características: más propenso a cambios y potencialmente se le agregará múltiples funcionalidades.

Práctica 4: Práctica para establecer el STACK tecnológico.

RMs generados:

4. El front-end debe ser desarrollado en el framework con nombre Angular, versión 11.2.5
5. El front-end debe ser desarrollado con el lenguaje de programación con nombre Typescript, versión 4.0.7
6. El front-end debe incluir la librería con nombre NX, versión 11.5.1
7. El servidor web donde se desplegará el front-end debe tener el sistema operativo con nombre Debian versión 10.
8. El back-end debe ser desarrollado en el framework con nombre SpringBoot, versión 2.3.7.
9. El back-end debe ser desarrollado con el lenguaje de programación con nombre Java, versión 1.8.
10. El back-end debe incluir las librerías con nombres spring-security-jwt versión 1.0.9, spring-boot-starter-mail, mysql-connector-java, spring-security-oauth2 versión 2.3.3.
11. El back-end debe trabajar con el servidor web con nombre Apache Tomcat, versión 9.0.2.
12. El servidor web donde se desplegará el back-end debe tener el sistema operativo con nombre Debian, versión 10.

13. La aplicación a desarrollar debe trabajar con el sistema gestor de base de datos con nombre MySQL, versión 8.0.23.

Nota: estos RMs buscan disminuir problemas de compatibilidad entre los elementos del STACK tecnológico, y eliminar substancialmente dependencias hacia componentes inestables que afectan en gran medida la mantenibilidad del producto software.

Práctica 5: Práctica para establecer el registro de eventos y sus lineamientos

RMs generados:

14. Se deben almacenar los eventos de la aplicación en un registro histórico de eventos.
15. La codificación de los eventos debe seguir los lineamientos planteados en el documento con nombre Lineamientos_log_app, con versión 1.0, que se encuentra en el siguiente enlace o ubicación Lineamientos_log_app .
16. El formato de la información de los eventos debe estar detallado en el documento con nombre Lineamientos_log_app, con versión 1.0, que se encuentra en el siguiente enlace o ubicación Lineamientos_log_app .
17. La descripción de los niveles y cuales eventos están asociados a cada nivel debe estar registrada en el documento con nombre Lineamientos_log_app, con versión 1.0, que se encuentra en el siguiente enlace o ubicación Lineamientos_log_app .

Nota: ver anexo 5 donde se encuentran los lineamientos para el registro de eventos.

Práctica 7: práctica para establecer independencia de lógica de negocio, medio de persistencia y presentación.

RMs generados:

18. La lógica de negocio del producto debe ser independiente de la presentación con la cual interactúa el usuario, de tal manera que cuando se agreguen nuevos tipos de presentación, o se modifique la tecnología de la presentación, sea posible utilizar la lógica de negocio del producto sin estar obligado a modificarla.
19. La lógica de negocio del producto software debe ser independiente del medio de persistencia, de tal manera que cuando se cambie el medio de persistencia o la tecnología del medio de persistencia, la lógica de negocio del producto no cambie.

Práctica 8: Práctica para establecer mensajes significativos

RMs generados:

20. Cuando ocurra un error causado por problemas del sistema, al usuario final se le debe mostrar el código del error y un mensaje significativo con el propósito de que el usuario comprenda el contexto del error.
21. Cada uno de los códigos de error causados por problemas del sistema debe tener un único mensaje de error asociado.

Práctica 11: Práctica para establecer componentes que exponen funcionalidades reutilizables dentro del producto software a desarrollar.

RMs generados:

22. El componente con nombre gráficos debe ser implementado de tal manera que pueda ser reutilizado dentro del producto software. El componente está constituido por los siguientes requisitos funcionales identificados con los ID: HU-ME-1003, HU-ME-1004.
23. El componente con nombre gráficos debe estar implementado con las siguientes tecnologías y versiones: Angular, Versión 11.5.1.
24. El componente con nombre gráficos debe ofrecer los siguientes servicios:

    Nombre del servicio: pastel-gráfico.

    Entradas al servicio: conjunto de datos que permitirán dibujar el gráfico de tipo pastel, título que se le colocará al gráfico tipo pastel.

    Descripción de la salida que genera el servicio: gráfico tipo pastel con los datos a dibujar junto con el título del pastel gráfico.

    Nombre del servicio: porcentaje-gráfico.

    Entradas al servicio: conjunto de datos que permitirán dibujar el gráfico de tipo porcentaje, título que se le colocará al gráfico tipo porcentaje.

    Descripción de la salida que genera el servicio: gráfico de tipo porcentaje, junto con el título del gráfico de tipo porcentaje.

    Nombre del servicio: circulo-gráfico.

    Entradas al servicio: dato que se colocará dentro del gráfico de tipo círculo, título que se le colocará al gráfico tipo círculo.

    Descripción de la salida que genera el servicio: gráfico del círculo con el dato a dibujar junto con su título.

Durante la fase de análisis 4 prácticas no permitieron capturar RMs. A continuación, se justifica por qué la práctica no permitió capturar el RM:

Práctica 6: Práctica para establecer envíos de logs locales de app de escritorio a un servidor.
Justificación: en la aplicación no se tiene previsto hacer una versión de escritorio por lo tanto esta práctica no genera RMs.

Práctica 9: Práctica para establecer componentes desarrollados previamente por la empresa que se van a reutilizar.
Justificación: la OCI no tiene proyectos previos por lo tanto no se puede aplicar esta práctica.

Práctica 10: Práctica para establecer componentes que deben ser desarrollados de tal manera que puedan ser reutilizados en futuros proyectos.
Justificación: tanto el analista como el equipo de desarrollo no establecieron componentes a desarrollar para ser reutilizados en futuros proyectos.

Práctica 12: Práctica para independencia de la lógica de negocio con diferentes sistemas que ofrezcan el mismo servicio.
Justificación: la aplicación a desarrollar no debe interactuar con otras aplicaciones.

5.4.Cumplimiento durante la etapa de desarrollo de los requisitos de mantenibilidad

Para determinar que RMs podrían ser implementados en una parte del producto software teniendo en cuenta el tiempo, disposición y conocimiento del desarrollador, se llegó a la conclusión que los RMs 1, 2, 3, 4, 5, 6, 8, 9, 10, 11, 13, 18, 19, 22, 23 y 24 serían implementados en su contexto original exceptuando los RMs 1 y 2 puesto que se implementaron solo en el componente con nombre *gráficos*. Este componente tiene como propósito ser reutilizado en la aplicación para mostrar las estadísticas de acuerdo a los planes de mejora creados, evaluados y terminados para cualquier usuario con acceso a la página principal de la OCI. Los gráficos a los cuales da acceso este componente son gráficos de pastel, de porcentajes y de símbolos numéricos.

Requisito 1 y 2 de mantenibilidad

Para el cumplimiento de los RMs número 1 y 2, el analista con el apoyo de los investigadores creó dos documentos, uno para para establecer lineamientos de codificación (ver anexo 3) y el otro para establecer lineamientos para documentar el código (ver anexo 4).

Para el RM número 1 de codificación el desarrollador implementó el código fuente teniendo en cuenta los siguientes ítems del documento:

- Nombrado de ficheros.
- Bootstrapping.
- Selector de componentes.
- Selector de directivas.
- Clases.
- Constantes.
- Interfaces.
- Propiedades y métodos.

De la misma manera, ocurrió con el RM número 2, donde el desarrollador documentó el código teniendo en cuenta los siguientes ítems del documento:

- Reglas para documentar atributos
- Reglas para documentar métodos
- Reglas para documentar componentes

El componente *graficos* fue desarrollado sólo con la tecnología del front-end, por consecuencia sólo la sección del front-end de los 2 documentos aplicaría para cumplir a cabalidad con estos 2 RMs. En la figura 5 se muestran los archivos que constituyen al componente *gráficos*. El nombrado de los archivos sigue los lineamientos planteados en el RM 1. En la figura 6 se muestra la clase *PorcentajeGraficoComponent* que evidencia el cumplimiento de los lineamientos para codificar, planteados por el RM número 1 y para documentar el código, planteados por el RM número 2. El código fuente completo del componente se encuentra en el anexo 6.

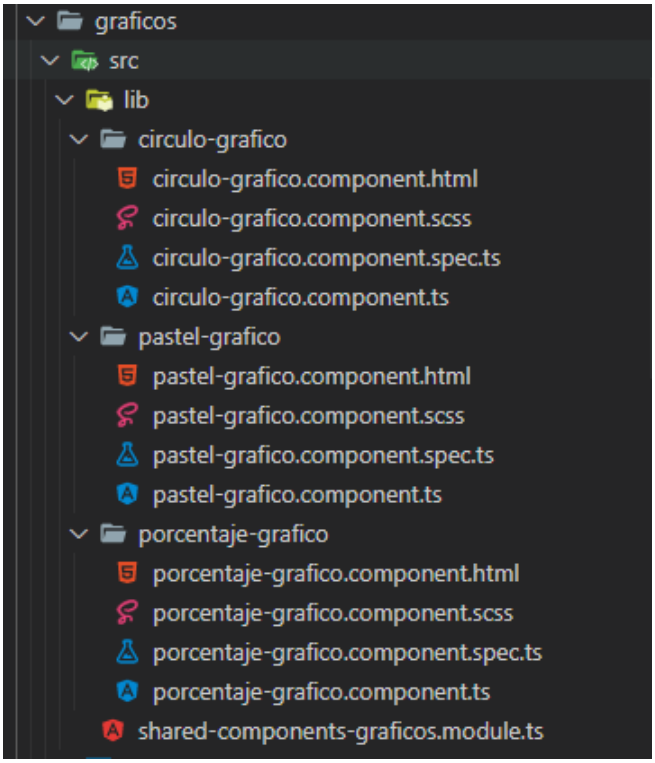


Figura 5. Árbol de archivos del componente gráficos. Autor fuente propia.

```
export class PorcentajeGraficoComponent implements OnInit {

  /**
   * Número que determina el porcentaje a renderizar
   * Se determina desde el componente desde donde
   * será usado
   * @type {number}
   * @memberof PorcentajeGraficoComponent
   */
  @Input() porcentaje: number;
  /**
   * String que determina el titulo de la mat-card
   * Se determina desde el componente desde donde
   * será usado
   * @type {string}
   * @memberof PorcentajeGraficoComponent
   */
  @Input() titulo: string;
  /**
   * Número que guarda el cálculo desde donde iniciará el trazo
   * del borde del círculo svg
   * @type {number}
   * @memberof PorcentajeGraficoComponent
   */
  dashoffset: number;

  /**
   * Retorna el valor desde donde inicia
   * el trazo del círculo (stroke-dashoffset)
   * que representará el porcentaje
   * @param {number} stroke longitud del lienzo
   * @return {*}  {number} valor del stroke-dashoffset
   * @memberof PorcentajeGraficoComponent
   */
  calcularDesplazamiento(stroke: number): number{
    return stroke - (stroke * this.porcentaje) / 100;
  }
```

Figura 6. Documentación y codificación del componente porcentaje-gráfico. Autor fuente propia.

Estos RMs permitieron que el código fuente sea más comprensible, auto-descriptivo, fácil de entender. También permitió homogeneizar la manera en que se documenta el código fuente y aumentar la facilidad con la cual los desarrolladores entienden su propósito.

Requisito 3 de mantenibilidad

Para cumplir con el RM número 3, el desarrollador, junto con los investigadores, utilizó SonarQube para verificar si el código fuente del back-end y del front-end del módulo denominado gestión de planes de mejora cumplía con el puntaje de mantenibilidad expuesto en el RM.

Como se observa en las figuras 7 y 8 los componentes que constituyen el módulo tienen un puntaje de mantenibilidad de A, con lo cual se cumple con el RM. Esto quiere decir que el esfuerzo para mantener el código del módulo es menor o igual al 5% del tiempo invertido en su desarrollo.

El cumplimiento del RM fue logrado gracias a satisfacer los demás RMs generados y algunas buenas prácticas tales como: nombres significativos de métodos, clases, variables; separación de módulos con única responsabilidad; cada función no debe tener más de 17 líneas de código; entre otros.

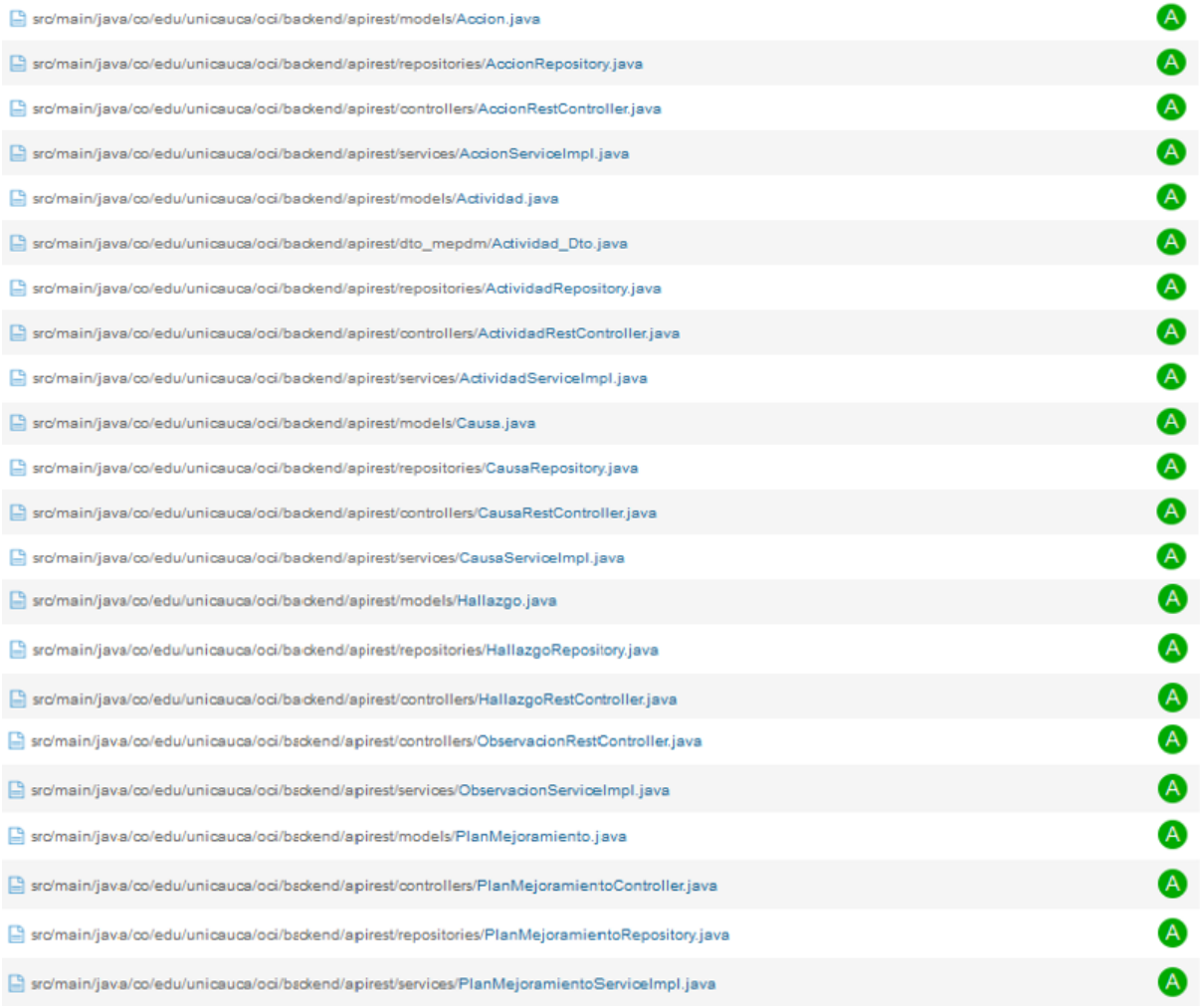


Figura 7. Puntaje de mantenibilidad arrojado por SonarQube en los componentes: plan de mejoramiento, hallazgos, causas, acciones, actividades y observaciones en el back-end. Autor fuente propia.

Figura 8. Puntaje de mantenibilidad arrojado por SonarQube en los componentes: plan de mejoramiento, hallazgos, causas, acciones y actividades en el front-end. Autor fuente propia.

Este RM permitió desde una etapa temprana establecer una meta cualitativa clara y no ambigua sobre la mantenibilidad del producto a desarrollar. Al cumplir con el RM se evidenció que el módulo quedó más independiente de otros módulos, se potenciaron las buenas prácticas para cumplir con el puntaje planteado desde la etapa de análisis.

Requisito 4, 5, 6, 8, 9, 10, 11 y 13 de mantenibilidad

Estos RMs buscan establecer el STACK tecnológico para el producto a desarrollar con el propósito de disminuir problemas de compatibilidad y de dependencias entre diferentes módulos de software.

En las figuras 9, 10 y 11 se puede observar el cumplimiento de los RMs 4, 5 y 6 para la tecnología front-end

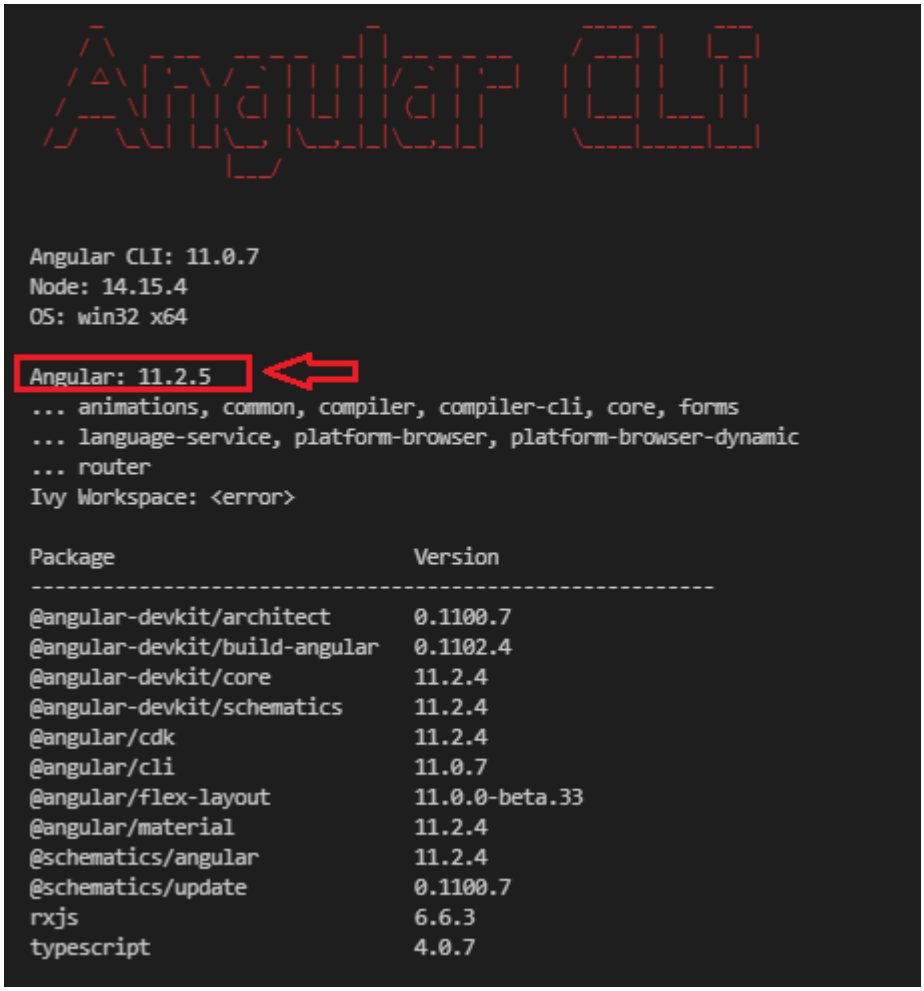


Figura 9. Framework Angular con versión 11.2.5 para el front-end. Autor fuente propia.

Figura 10. Lenguaje de programación Typescript con versión 4.0.7 para el front-end. Autor fuente propia.

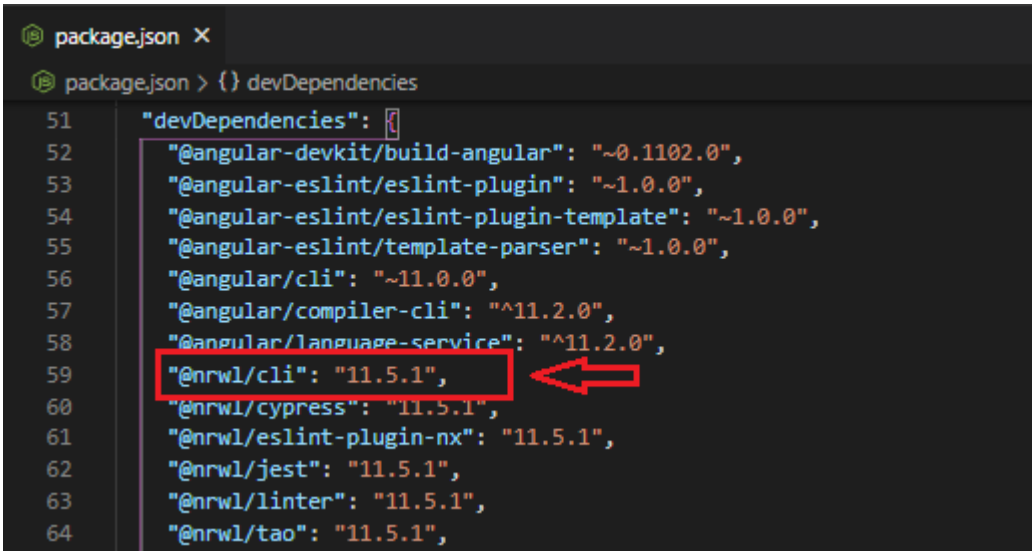


Figura 11. Librería nx o también conocida como @nrwl/cli con versión 11.5.1 para el frond-end. Autor fuente propia.

En las figuras 12, 13, 14, 15, 16 y 17 se muestra el cumplimiento de los RMs 8, 9, 10 y 11 para la parte del back-end.

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.3.7.RELEASE</version>
    <relativePath/> <!-- lookup parent from repository -->
</parent>
```

Figura 12. Framework spring-boot con versión 2.3.7 utilizada en el back-end. Autor fuente propia.

```
<properties>
    <java.version>1.8</java.version>
</properties>
```

Figura 13. Lenguaje de programación java con versión 1.8 para el back-end. Autor fuente propia.

```
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-jwt</artifactId>
    <version>1.0.9.RELEASE</version>
</dependency>
```

Figura 14. Librería spring-security-jwt que se utilizó en el back-end. Autor fuente propia.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-mail</artifactId>
</dependency>
```

Figura 15. Librería spring-boot-starter-mail que se utilizó en el back-end. Autor fuente propia.

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <scope>runtime</scope>
</dependency>
```

Figura 16. Librería mysql-connector-java que se utilizó en el back-end. Autor fuente propia.

```
<dependency>
    <groupId>org.springframework.security.oauth</groupId>
    <artifactId>spring-security-oauth2</artifactId>
    <version>2.3.3.RELEASE</version>
</dependency>
```

Figura 17. Librería spring-security-oauth2 que se utilizó en el back-end. Autor fuente propia.

En la figura 18 se evidencia el cumplimiento del RM 13 que establece el sistema gestor de base de datos a utilizar junto con su versión.

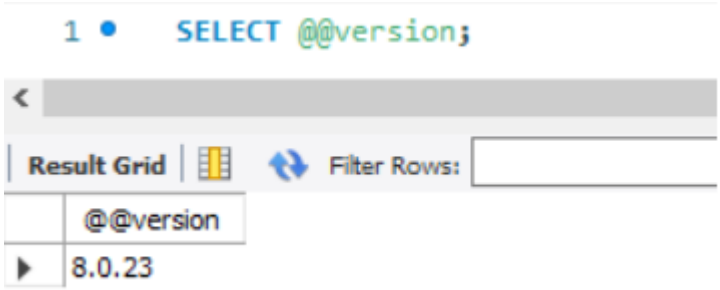


Figura 18. Gestor de base de datos MySQL en la versión 8.0.23. Autor fuente propia.

El cumplimiento de estos RMs permitió que durante el desarrollo de los componentes no se encontrara ninguna incompatibilidad entre dependencias y se logró eliminar substancialmente dependencias hacia componentes inestables que afectan en gran medida la mantenibilidad del producto software.

Requisito 18 de mantenibilidad.

Para el cumplimiento de este RM, se utilizó el modelo cliente-servidor, el modelo REST, una API con las tecnologías HTTP, servicios RESTFul y el patrón inyección de las independencias, que permitieron hacer la separación entre la presentación del producto y la lógica de negocio del producto. La presentación del producto se refiere a todo lo que el usuario ve cuando usa el producto, desde fuentes, colores hasta menús desplegables y controles deslizantes. La lógica de negocio del producto es donde se ejecutan todas las diferentes reglas de negocio con el propósito de satisfacer una acción realizada por el usuario.

En las figuras 19 y 20 se evidencia el uso de la API en el componente *causas*. La figura 19 muestra la petición al servidor *getAccionesPorIdCausa/{idCausa}* desde el front-end, y la figura 20 muestra la lógica de negocio correspondiente a esa petición, la cual se encuentra en el back-end.

```
public getAccionPorIdCausa(idCausa: number): Observable<any>{
  return this.http.get<any>(this.urlEndPoint+`getAccionesPorIdCausa/${idCausa}`);
}
```

Figura 19. Llamada a API getAccionesPorIdCausa desde el front-end. Autor fuente propia.

```
@GetMapping("/getAccionesPorIdCausa/{idCausa}")
public ResponseEntity<?> getAccionesPorIdCausa(@PathVariable("idCausa") Integer idCausa){
    List<Accion> acciones = new ArrayList<>();
    Map<String, Object> response = new HashMap<>();
    try {
        acciones.addAll(this.accionService.getAccionesPorIdCausa(idCausa));
    } catch (DataAccessException e) {
        response.put("mensaje", "Error al realizar la consulta en la base de datos");
        response.put("error", e.getMessage()+" "+ e.getMostSpecificCause().getMessage());
        return new ResponseEntity<Map<String, Object>>(response, HttpStatus.INTERNAL_SERVER_ERROR);
    }

    if(acciones.isEmpty()) {
        response.put("mensaje", "No existen acciones asociadas a la causa "+idCausa);
        ResponseEntity<Map<String, Object>> objRespuesta = new ResponseEntity<Map<String,Object>>(response,HttpStatus.NOT_FOUND);
        return objRespuesta;
    }
    else {
        return new ResponseEntity<List<Accion>>(acciones, HttpStatus.OK);
    }
}
```

Figura 20. Lógica de la API getAccionesPorIdCausa en el back-end. Autor fuente propia.

De esta manera, si se decide hacer una versión móvil para la aplicación, se conserva la misma lógica de negocio y la nueva presentación móvil sólo tendría que consumir estos servicios ya implementados. El cumplimiento de este RM aumenta la facilidad de expansión del producto software, debido a que es posible agregar varios tipos de presentación conservando la misma lógica de negocio.

Requisito 19 de mantenibilidad

Para el cumplimiento de este RM, se utilizó el modelo cliente-servidor, el patrón de diseño repository y la tecnología Hibernate que funciona como una implementación de Java Persistence API (JPA). Las anteriores tecnologías permitieron que la persistencia de la información del sistema sea independiente de la base de datos, utilizando una notación en común.

En la figura 21 se evidencia el uso de la dependencia de Hibernate en el archivo de dependencias del back-end. En la figura 22 se muestra las notaciones de JPA utilizadas en la clase *Evaluacion*, estas notaciones no dependen de la base de datos a utilizar. Y por último en la figura 23 se muestra una porción del archivo application.properties donde están las variables utilizadas para el gestor de base de datos MySQL. Si el equipo de desarrollo quiere hacer un cambio de MySQL a Postgres lo único que tendrían que modificar en el archivo application.properties son las variables url, driver, plataforma y credenciales. De esta manera la lógica de negocio queda independiente del medio de persistencia que se utilice.

```xml
<dependency>
    <groupId>org.hibernate.validator</groupId>
    <artifactId>hibernate-validator</artifactId>
    <version>6.1.6.Final</version>
</dependency>
<dependency>
    <groupId>org.hibernate.validator</groupId>
    <artifactId>hibernate-validator-annotation-processor</artifactId>
    <version>6.0.2.Final</version>
</dependency>
```

Figura 21. Dependencia de Hibernate en el pom.xml. Autor fuente propia.

```java
@Entity
@Table(name = "evaluaciones")
public class Evaluacion {

    @Id
    @Column(name = "id_evaluacion")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @Column(name = "observacion")
    private String observacion;

    @Column(name = "fecha_evaluacion")
    private Date fecha_evaluacion;

    @Column(name = "estado")
    private String estado;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "ID_EVIDENCIA")
    @JsonIgnoreProperties({"hibernateLazyInitializer", "handler"})
    private Evidencia objEvidencia;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "ID_PERSONA")
    @JsonIgnoreProperties({"hibernateLazyInitializer", "handler"})
    private Person objPersona;
```

Figura 22. Notaciones JPA en la clase Evaluación. Autor fuente propia.

application.properties

src > main > resources > application.properties

Fernanda J, a month ago | 2 authors (You and others)

```
1  #DATA SOURCE (MySQL 8.0)
2  spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
3  spring.datasource.url= jdbc:mysql://db-proyecto-oci.ctiqnkvpv1sw.us-east-1.rds.amazonaws.com:3306/db_proyecto_oci
4  spring.datasource.username=root
5  spring.datasource.password=
6
7  #JPA para STS4
8  #spring.jpa.generate-ddl=false
9  #spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL55Dialect
10 spring.jpa.database-platform=org.hibernate.dialect.MySQL55Dialect
```

Figura 23. variables para MySQL en el archivo application.properties. Autor fuente propia.

El cumplimiento de este RM favorece la flexibilidad, la capacidad de expansión, también ayuda a tener un bajo acoplamiento entre componentes, lo cual hace que se potencien las 5 sub características de mantenibilidad de la ISO 25010.

Requisito 22, 23 y 24 de mantenibilidad

En las figuras 24 y 25 se muestra una parte de la implementación del componente *gráficos*, la cual expone una interfaz que permite que sea reutilizable dentro del producto software. El componente cumple con el RM 22 debido a que satisface los requisitos funcionales HU-ME-1003, HU-ME-1004. El componente cumple con el RM número 23 debido a que fue desarrollado con la tecnología de angular en la versión 11.5.1. El componente cumple con el RM número 24 debido a que ofrece los servicios de circulo-grafico, pastel-grafico y porcentaje-grafico con sus diferentes entradas y salidas expuestas en el RM, las cuales se pueden llamar desde cualquier parte de la aplicación haciendo uso de sus selectores como se evidencia en la figura 26 sin introducir duplicidad de código.

```
@Component({
  selector: 'unicauca-porcentaje-grafico',
  templateUrl: './porcentaje-grafico.component.html',
  styleUrls: ['./porcentaje-grafico.component.scss']
})
export class PorcentajeGraficoComponent implements OnInit {

  /**
   * Número que determina el porcentaje a renderizar
   * Se determina desde el componente desde donde
   * será usado
   * @type {number}
   * @memberof PorcentajeGraficoComponent
   */
  @Input() porcentaje: number;
  /**
   * String que determina el titulo de la mat-card
   * Se determina desde el componente desde donde
   * será usado
   * @type {string}
   * @memberof PorcentajeGraficoComponent
   */
  @Input() titulo: string;
  /**
   * Número que guarda el cálculo desde donde iniciará el trazo
   * del borde del circulo svg
   * @type {number}
   * @memberof PorcentajeGraficoComponent
   */
  dashoffset: number;

  /**
   * Retorna el valor desde donde inicia
   * el trazo del círculo (stroke-dashoffset)
   * que representará el porcentaje
   * @param {number} stroke longitud del lienzo
   * @return {*}  {number} valor del stroke-dashoffset
   * @memberof PorcentajeGraficoComponent
   */
  calcularDesplazamiento(stroke: number): number{
    return stroke - (stroke * this.porcentaje) / 100;
  }
```

Figura 24. Archivo .ts del componente porcentaje-grafico. Autor fuente propia.

```
<mat-card>
  <div class="box">
    <span class="text">{{ titulo }}</span>
    <div class="percent">
      <svg>
        <circle cx="55" cy="55" r="55"></circle>
        <circle
          cx="55"
          cy="55"
          r="55"
          [ngStyle] = "{'stroke-dashoffset': dashoffset}"
        ></circle>
      </svg>
      <div class="number">
        <h2>{{ porcentaje }}<span>%</span></h2>
      </div>
    </div>
  </div>
</mat-card>
```

Figura 25. Archivo html del componente porcentaje-grafico. Autor fuente propia.

plan-mejoramiento.component.html

libs > modules > estadisticas > feature-estadisticas > src > lib > plan-mejoramiento >

```
13          <!--Detalles del plan de mejoramiento-->
14          <div class="container-cards">
15            <unicauca-porcentaje-grafico
16              [porcentaje]="avance"
17              [titulo]="titleAvance"
18            >
19            </unicauca-porcentaje-grafico>
20            <unicauca-circulo-grafico
21              [texto]="noAcciones"
22              [titulo]="titleAcciones">
23            </unicauca-circulo-grafico>
24            <unicauca-porcentaje-grafico
25              [porcentaje]="cumplimiento"
26              [titulo]="titleCumplimiento"
27            >
28            </unicauca-porcentaje-grafico>
29          </div>
30
```

Figura 26. Un componente que hace uso de la reusabilidad llamando el componente porcentaje-grafico a través de su selector. Autor fuente propia.

Como se evidencia en la figura 26, el componente del plan-mejoramiento utilizó el selector del componente del porcentaje-grafico para que en esa parte de la vista se visualizara los dos porcentajes gráficos, uno para los avances y otro para mostrar los porcentajes de cumplimiento. Como se puede observar, si no se hubiera hecho el porcentaje-grafico de forma reusable, en el caso del componente plan de mejoramiento, se hubieran duplicado los archivos .ts y los archivos html de las figuras 24 y 25.

5.5.Conocimiento obtenido en los participantes del estudio de caso.

En la tabla 19 se muestran las respuestas que permiten conocer el nivel de conocimiento ganado por los participantes durante el estudio de caso

Tabla 19. Respuestas acerca del nivel de conocimiento ganado por los participantes durante los estudios de caso

Nivel de conocimiento	Nivel de conocimiento antes del proceso de intervención	Nivel de conocimiento después del proceso de intervención
Desarrollador	Entre 30% y 50% de conocimiento.	Entre 70% y 100% de conocimiento.
Cliente 1	Entre 0 – 30% de conocimiento	Entre 30% y 50% de conocimiento.
Cliente 2	Entre 0 – 30% de conocimiento	Entre 50% y 70% de conocimiento.
Cliente 3	Entre 0 – 30% de conocimiento	Entre 70% y 100% de conocimiento.

En la tabla 20 se muestran las preguntas y respuestas de tipo si/no realizadas al analista puesto que fue la persona encargada de capturar y especificar los RMs a través de las prácticas propuestas.

Tabla 20. Respuestas de tipo si/no realizadas al analista

Aspectos de respuestas tipo Si/No	Número de encuestados que marcaron Si	Número de encuestados que marcaron No
¿Considera que los campos definidos en las prácticas son suficientes?	1	0
¿Añadiría elementos a algunas notaciones de RMs propuestas?	0	1

En la tabla 21 se muestran los conceptos que no fueron entendidos por los stakeholders durante el desarrollo del estudio de caso.

Tabla 21. Conceptos que no fueron entendidos por los stakeholders durante el estudio de caso

Rol del evaluado	Conceptos no entendidos
Desarrollador	Temporal, atemporal y puntaje de mantenibilidad.
Cliente 1	Temporal y atemporal.
Cliente 2	Temporal y atemporal.
Cliente 3	Temporal y atemporal.

5.6.Resultados del estudio de caso

5.6.1. Aspectos por mejorar identificados en las prácticas

En el estudio de caso se aplicó una primera versión de las prácticas propuestas, lo cual generó un conjunto de aspectos por mejorar. Los aspectos por mejorar 1 y 3 fueron aplicados en la actual versión de las prácticas que se muestran en el capítulo 4.

Mejora 1

En la figura 27 se muestra la notación que se utilizó para la captura y especificación de los RMs que genera la práctica 3

RMs generados
A. Para módulos propensos a cambios 1. El módulo con nombre _____ que se compone de los siguientes requisitos funcionales identificados con los ID: ____, _____, _____, _____, _____ debe tener un puntaje de mantenibilidad que esté entre A y B. El puntaje de mantenibilidad ha sido planteado porque el módulo es más propenso a cambios. actividades B. Para módulos más propensos a errores en la fase de pruebas. 1. El módulo con nombre _____ que se compone de los siguientes requisitos funcionales identificados con los ID: ____, _____, _____, _____, _____ debe tener un puntaje de mantenibilidad que esté entre A y B. El puntaje de mantenibilidad ha sido planteado porque el módulo es más propenso a errores de codificación en la fase de pruebas. C. Para módulos potencialmente con más funcionalidades. 1. El módulo con nombre _____ que se compone de los siguientes requisitos funcionales identificados con los ID: ____, _____, _____, _____, _____ debe tener un puntaje de mantenibilidad que esté entre A y B. El puntaje de mantenibilidad ha sido planteado porque el módulo potencialmente se le agregará múltiples funcionalidades.

Figura 27. Notación que se utilizó para la práctica 3. Autor fuente propia.

Cuando se realizó el proceso para la captura y especificación de los RMs, los RMs asociados a la práctica 3 fueron especificados de la siguiente manera:

1. El módulo con nombre gestión de planes de mejora que se compone de los siguientes requisitos funcionales identificados con los ID: HU-09, HU-11, HU-23, HU-13, HU-15, HU-28 debe tener un puntaje de mantenibilidad que esté entre A y B. El puntaje de mantenibilidad ha sido planteado porque el módulo es más propenso a cambios.
2. El módulo con nombre gestión de planes de mejora que se compone de los siguientes requisitos funcionales identificados con los ID: HU-09, HU-11, HU-23, HU-13, HU-15, HU-28 debe tener un puntaje de mantenibilidad que esté entre A y B. El puntaje de mantenibilidad ha sido planteado porque el módulo potencialmente se le agregará múltiples funcionalidades

Como se puede evidenciar, estos 2 RMs son repetidos ya que se trata del mismo módulo con la misma restricción de puntaje de mantenibilidad, pero con diferente razón o característica por

la cual se estableció. El analista expresó la limitación de la notación anteriormente propuesta y se cambió por la que tenemos actualmente documentada en el capítulo 4 en la sección de las prácticas, que es la que se muestra en la figura 28:

RMs generados
1.El módulo con nombre _____ que se compone de los siguientes requisitos funcionales identificados con los ID: _____, _____, _____, _____, _____ debe tener un puntaje de mantenibilidad que esté entre A y B. El puntaje de mantenibilidad ha sido planteado porque el módulo cumple con las siguientes características: _____, _____.

Figura 28. Notación actualizada para la práctica 3. Autor fuente propia.

Esta notación es más flexible, ya que permite incorporar las 3 características que son: más propenso a cambios, más propenso a errores en la fase de pruebas y que potencialmente se le agregarán más funcionalidades. De esta manera, con la nueva notación, se hizo el cambio de los 2 requisitos por el siguiente:

El módulo con nombre gestión de planes de mejora que se compone de los siguientes requisitos funcionales identificados con los ID: HU-09, HU-11, HU-23, HU-13, HU-15, HU-28 debe tener un puntaje de mantenibilidad que esté entre A y B. El puntaje de mantenibilidad ha sido planteado porque el módulo cumple con las siguientes características: más propenso a cambios y potencialmente se le agregará múltiples funcionalidades.

Mejora 2

Por otra parte, también se identificó que sería útil incluir lineamientos para la interacción de los analistas, clientes y equipo de desarrollo cuando se están capturando los RMs.

Mejora 3

Se identificó que cuando los RMs capturados y especificados se vayan a plasmar en el documento que tiene todos los requisitos del proyecto es necesario agregar la información del campo consideraciones debido a que este define una serie de conceptos necesarios para entender los RMs.

5.6.2. Utilidad de las prácticas para la captura y especificación de RMs

Las prácticas propuestas permitieron capturar y especificar 24 RMs sobre toda la aplicación de la OCI. Cada RM se buscó que fuera no ambiguo, completo, singular, factible, trazable y verificable. La cantidad y calidad final de los RMs especificados fue conseguida gracias a los

campos directriz, consideraciones, forma en la que potencia la mantenibilidad, roles y notación.

En este sentido, las prácticas propuestas ayudan al analista y al equipo en general del producto software a entender la necesidad de capturar y especificar RMs desde una etapa temprana como es la de análisis. Especialmente durante el proceso de captura y especificación de requisitos el analista y equipo opinó que el campo *forma en la que potencia la mantenibilidad los RMs generados,* es muy importante ya que indica la contribución de la práctica hacia la mantenibilidad de producto.

Por otro lado, el analista y equipo de desarrollo estableció que los RMs son igual o más importantes que los requisitos funcionales ya que van a beneficiar la capacidad de mantenimiento del producto software.

En cuanto al tiempo utilizado para capturar y especificar RMs, las prácticas propuestas lograron un tiempo promedio de 5 minutos y 7 ½ minutos para la captura y especificación de los RMs respectivamente. Estos tiempos son relativamente bajos gracias a que las prácticas proveen directrices, consideraciones y una notación que guían al analista en la captura y especificación.

Por otra parte, se identificó que las prácticas fueron fáciles de entender y que proveen una guía clara para poder capturar y especificar RMs desde la etapa de análisis. A excepción del concepto de atemporal y temporal el cual se tuvo que recordar durante la captura y especificación de los RMs, además de la aclaración de la definición e interpretación del rango del puntaje de mantenibilidad en el caso de la práctica 3.

En el mismo orden de ideas, la práctica número 3 fue la que más dificultad causó a los clientes y al analista, principalmente por la interacción que se debía generar al considerar componentes que serían más propensos a cambios, a errores y a cuáles componentes se podrían agregar más funcionalidades.

5.6.3. Conocimiento adquirido por los stakeholders

Durante la encuesta realizada los stakeholders manifestaron que antes del proceso de intervención sabían muy poco acerca de mantenibilidad y su importancia calificando su conocimiento en el intervalo de 0 a 30% en el caso de los clientes y de 30 a 50% en el caso del analista. Después de la intervención, los stakeholders adquirieron nuevo conocimiento acerca de cómo abordar la mantenibilidad desde la etapa de análisis, entendieron los conceptos básicos que rodean la mantenibilidad, la importancia de abordar la mantenibilidad tempranamente y

cómo esto repercutiría positivamente al costo final y a la calidad del producto desde su creación.

Profundizando en los datos recogidos mediante el instrumento de la encuesta, se evidenció que los stakeholders entre más conocimiento previo tenían en tecnología y experiencia trabajando con equipos de desarrollo, más conocimiento adquirieron sobre mantenibilidad al cierre del proceso. En el caso del cliente 1, no tenía mucha experiencia trabajando con equipos de desarrollo, y define su conocimiento adquirido entre el intervalo de 30 a 50%. En el caso del cliente 2, tenía más experiencia trabajando con equipos de desarrollo y más conocimiento en tecnología, este define su conocimiento adquirido entre el intervalo de 50 a 75%. En el caso del cliente 3, tiene un amplio conocimiento en tecnología y experiencia trabajando con equipos de desarrollo y define su conocimiento adquirido entre el intervalo de 75 al 100% y por último el analista que tiene amplio conocimiento en tecnología, pero poca experiencia trabajando con equipos de desarrollo a nivel industrial define también su conocimiento adquirido entre el 75 al 100%.

Teniendo en cuenta lo anterior, se puede concluir que no importa el conocimiento previo de los stakeholders las prácticas benefician a que se adquiera conocimiento acerca de la mantenibilidad, su importancia y cómo capturar y especificar RMs desde la etapa de análisis.

5.6.4. Correctitud de los campos de las prácticas

Los stakeholders calificaron los campos de las prácticas como fáciles, suficientes, correctos y que guían de manera clara la obtención y especificación de los RMs, de tal manera que no era necesario agregar campos nuevos a las prácticas. Por otro lado, los stakeholders sugirieron las siguientes mejoras, (i) cambiar la notación para la práctica 3, la cual en ocasiones podía generar RMs repetidos, (ii) incluir lineamientos para la interacción de los analistas, clientes y equipo de desarrollo cuando se están capturando los RMs, y (iii) agregar una consideración la cual planea que cuando los RMs capturados y especificados se vayan a plasmar en el documento que tiene todos los requisitos del proyecto, es necesario agregar la información del campo consideraciones debido a que este define una serie de conceptos necesarios para entender los RMs.

Sólo 2 RM especificados no correspondían a la naturaleza de la información solicitada en la notación. Dichos RMs fueron el 11 y el 24. En el RM 11 se confundió el servidor web para el back-end con el nombre del dominio y en el RM 24 los nombres de las entradas de los servicios

y las salidas estaban escritos de una manera muy técnica. En dicha sesión los investigadores identificaron el error y se procedió a reescribir los RMs junto con los stakeholders.

Por otra parte, el campo de roles involucrados permitió al equipo de desarrollo tener una sesión eficiente ya que el analista tuvo la capacidad de separar las prácticas donde se necesitaba la interacción con los clientes y las prácticas donde no era necesaria, de esta manera se hizo un uso eficaz del talento humano para la sesión y no se ocupó más del tiempo necesario de los clientes.

Teniendo en cuenta este y todos los puntos anteriores, podemos llegar a la conclusión que las prácticas propuestas son idóneas para capturar y especificar RMs desde la etapa de análisis.

En la tabla 22 se muestra de manera sintetizada los valores obtenidos para cada una de las métricas utilizadas para evaluar la idoneidad de la propuesta, desde las perspectivas de utilidad, conocimiento adquirido y correctitud.

Tabla 22. Métricas utilizadas para evaluar la idoneidad de las prácticas.

Utilidad	
Nombre de métrica	**Valor**
Tiempo promedio para capturar los RMs que genera cada práctica.	5 min
Tiempo promedio para especificar los RMs que genera cada práctica.	7 ½ min
Cantidad de prácticas que generaron RMs a considerar en el desarrollo del producto software	8
Total, prácticas que no generaron RMs a considerar en el desarrollo	4
Grado en el cual los componentes desarrollados cumplen con los RM	Los componentes desarrollados cumplieron con el 100% de los RMs asignados a para ellos.
Facilidad de entendimiento de los campos para los stakeholders	4 (escala del 1 al 5, donde 1 es difícil y 5 muy fáciles de entender)
Conocimiento	
Nombre de métrica	**Valor**
Nivel de conocimiento del analista antes del proceso de intervención	De 30% a 50%
Nivel de conocimiento del analista después del proceso de intervención	De 70% a 100%
Nivel de conocimiento promedio de los clientes antes del proceso de intervención	De 0% a 30%
Nivel de conocimiento promedio de los clientes después del proceso de intervención	De 50% a 70%

Cantidad de conceptos que no entendieron los stakeholders	3 (temporal, atemporal, puntaje de mantenibilidad)
Correctitud	
Nombre de métrica	**Valor**
Cantidad de RMs que siguen el orden y la naturaleza de la información solicitada por la notación propuesta.	24
Cantidad de RMs que no siguen el orden y la naturaleza de la información solicitada por la notación propuesta.	1
Elementos nuevos agregados a las notaciones para especificar los RMs.	1
Campos nuevos agregados a las prácticas.	0

5.6.5. Discusión

A partir de la experiencia que generó el estudio de caso ejecutado en la aplicación web que soportará las actividades de la OCI, se pueden resaltar las siguientes lecciones aprendidas por parte de la organización y los investigadores:

- La experiencia en industria del equipo de desarrollo es importante a la hora de definir los documentos para los RMs 1 y 2 puesto que, se plantean lineamientos para la estandarización de la codificación y la documentación de toda la aplicación o un segmento de esta.
- Debido a que algunas prácticas manejan conceptos técnicos y además la mantenibilidad no es muy conocida por analistas y clientes, es necesario antes de iniciar la captura y especificación de RMs hacer una capacitación previa sobre los conceptos más relevantes sobre mantenibilidad, sus sub características, la problemática abordada, definición y propósitos de cada campo que constituye las prácticas propuestas y un ejemplo donde se aplica al menos una práctica.
- El analista debe establecer el orden de las prácticas en la captura y especificación para tener un mejor control a la hora de hacer las preguntas que generan RMs.
- Debido a que algunas prácticas involucran los conceptos de modularidad y componentes es necesario que previo a la captura se hayan identificado los requisitos funcionales y los módulos que constituirán la aplicación.
- Para capturar los RMs se debe plantear una sesión de mínimo 1 hora para la capacitación, mínimo 1 hora para la captura y un tiempo prudente para la especificación.

- Así como se construyeron prácticas para capturar RMs también es posible plantear prácticas para capturar otros tipos de requisitos de calidad.

5.7. Limitaciones de la evaluación y su gestión

En la aplicación de las prácticas se identificaron 3 limitaciones: (i) el desconocimiento de la definición de mantenibilidad y sus sub características, el concepto de RM y la importancia de obtenerlos desde una etapa temprana del ciclo de vida del software por parte del analista, desarrollador y los clientes, (ii) el poco tiempo para la implementación de los RMs generados, y (iii) los pocos desarrolladores participantes. Con el propósito de mitigar el impacto de cada limitación se implementaron las siguientes estrategias:

- Antes de la captura y especificación, fue realizada a los participantes una capacitación de las siguientes temáticas: (i) los conceptos más relevantes sobre mantenibilidad, sus sub características y la problemática abordada, (ii) definición y propósitos de cada campo que constituye las prácticas propuestas y (iii) un ejemplo donde se aplicó una práctica.
- Se acordó con los participantes realizar varias sesiones para optimizar los resultados. Estas sesiones fueron la contextualización de la propuesta, presentar las prácticas, captura y especificación de RMs, definición de componentes a desarrollar (paralelo), desarrollo de los RMs generados y acordados en anteriores sesiones y el cierre del proceso junto con una encuesta.
- Para maximizar los resultados y lecciones aprendidas, los investigadores siempre estuvieron al tanto de cualquier inquietud o error que podían presentar tanto los clientes como el analista y desarrollador en cualquiera de las sesiones mencionadas en el anterior ítem.

5.8. Plan de validez

Para tratar las amenazas a la validez de los estudios de caso se han considerado diferentes aspectos, los cuales se describen a continuación:

- **Diseño de los estudios de caso.** El diseño ha sido basado en el protocolo planteado en [17] para realizar estudios de casos y sus elementos han sido confrontados con la lista de chequeo para estudios de caso en ingeniería de software propuesto por [95], dando como resultado que el diseño cumple con los elementos propuestos en un alto porcentaje.

- **Validez de constructo.** Para las preguntas establecidas en el estudio de caso se han definido medidas objetivas, verificables y trazables que podrán permitir la obtención de información fiable acerca de los criterios de completitud, correctitud e idoneidad de las prácticas. Se han utilizado múltiples fuentes de evidencia: registro de archivos, entrevistas y observación participativa, obtenidas desde diferentes roles participantes en el proceso de captura y especificación de los RMs. Además, se ha mantenido una cadena de evidencia permitiendo la trazabilidad entre preguntas de investigación, elementos de la planificación de los estudios de caso, datos almacenados, evidencias y análisis.
- **Validez interna.** Apoyándonos en las prácticas propuestas aplicadas a toda la aplicación de la OCI se pudo capturar y especificar RMs que aumentaron la capacidad de mantenimiento del nuevo producto software. Toda la información obtenida del proceso se logró plasmar en los campos 8, 9 y 10 de cada práctica propuesta.
- **Validez externa**. Para maximizar los resultados, los investigadores observaron como el analista, desarrollador y los clientes realizaban el proceso de captura y especificación de RMs. Aunque fue necesario durante ciertos momentos resolver dudas e inquietudes sobre algunos conceptos o elementos de las prácticas, por ejemplo, cómo llenar un campo en la plantilla o cuál era la respuesta más apropiada.

Capítulo 6 - Conclusiones, productos generados, lecciones aprendidas y trabajo futuro

En esta investigación han sido desarrolladas mediante el método de investigación-acción 12 prácticas para la captura y especificación de RMs. Las prácticas se componen de los siguientes campos: (i) una directriz para capturar los RMs; (ii) consideraciones que describen un conjunto de restricciones o aspectos a tener en cuenta en los diferentes elementos que constituyen a los RMs generados mediante la práctica; (iii) sub características de mantenibilidad abordadas (iv) elementos de mantenibilidad que serán potenciados en su logro mediante los RMs generados; (v) sub elementos de mantenibilidad que serán potenciados en su logro mediante los RMs generados; (vi) forma en la cual se aumentará la capacidad de mantenimiento dentro del sistema a partir de los RMs; (vii) los roles con los que puede interactuar el analista funcional, con el propósito de obtener los RMs; (viii) la ruta al documento de especificación de requisitos donde se almacenarán los RMs; (ix) Un check que permite establecer si los RMs generados son temporales o atemporales y (x) la notación que deberá ser seguida para la especificación de los RMs generados por la práctica.

6.1. Conclusiones

A continuación, se presentan las conclusiones a las que se llegó al finalizar este trabajo investigativo:

- A partir del mapeo de la literatura se identificó que no existe una definición común del concepto de RM, se carece de un análisis sistémico de los elementos de mantenibilidad que deben ser abordados desde la ingeniería de requisitos, y no se identificó una guía clara, completa y sistemática para capturar RMs a partir de la interacción entre los usuarios finales del negocio, analistas y equipo de desarrollo.
- A partir de un mapeo sistemático de la literatura fueron obtenidos 36 elementos y 61 sub elementos de mantenibilidad, los cuales fueron correlacionados con las sub características de mantenibilidad de la ISO 25010. Durante la obtención de los elementos se identificó que la mayoría de ellos se repetían, no estaban definidos y no estaban asociados a una su característica de mantenibilidad.
- La complejidad temporal no se tomó en cuenta en el proyecto de investigación debido a que en la revisión sistemática del capítulo 3 no se halló esta complejidad, ni su definición teniendo en cuenta los criterios de inclusión y exclusión utilizados, sin

embargo, se debe tener en cuenta el riesgo que presenta para la mantenibilidad de un producto software.

- Con el propósito de capturar y especificar RMs del producto software fueron planteadas 12 prácticas constituidas por 10 campos. Cada uno de los campos buscaba facilitar el proceso de captura de los RMs, permitir entender la importancia de los RMs y su repercusión en la mantenibilidad del producto software, y finalmente buscar que la especificación de los RMs fuera no ambigua, completa, singular, factible, trazable y verificable.
- Cada práctica ayuda al analista a adquirir conocimiento acerca de la importancia de cada RM generado, que sub características de mantenibilidad se pueden potenciar al implementarlo y los elementos abordados.
- En la ejecución del estudio de caso no se encontró ningún conflicto entre las prácticas diseñadas, sin embargo, puede existir la posibilidad de conflictos en determinadas situaciones. En este caso se debería escoger la práctica que potencia los elementos y sub características más buscadas por la empresa para el producto a desarrollar.
- A partir del estudio de caso se ha determinado que el diseño de las prácticas permite que la curva de aprendizaje sea exponencial, pero se necesita una capacitación previa de los conceptos más relevantes acerca de la mantenibilidad debido a que la mantenibilidad no es muy conocida por analistas y clientes.
- A partir de la ejecución del estudio de caso, se estableció que las prácticas propuestas son idóneas para capturar y especificar RMs desde la etapa de análisis debido a que el analista, desarrollador y clientes plantearon que las prácticas: (i) proponen lineamientos para capturar de manera clara RMs, (ii) ayudan a entender la necesidad de capturar y especificar RMs desde una etapa temprana como es la de análisis, (iii) permiten que el analista rápidamente pueda entender los conceptos fundamentales para capturar los RMs y especificarlos y (iv) plantean un conjunto de campos suficientes, correctos y guían de manera clara la obtención y especificación de los RMs.
- A partir del mapeo de la literatura se identificó que no existe una definición común del concepto de RM.
- Debido a que algunas prácticas manejan conceptos técnicos y además la mantenibilidad no es muy conocida por analistas y clientes, es necesario antes de iniciar la captura y especificación, hacer una capacitación previa sobre los conceptos más relevantes acerca

de la mantenibilidad, sus sub características, definición y propósitos de cada campo que constituye las prácticas propuestas y un ejemplo donde se aplica al menos una práctica.

- A partir del estudio de caso se determinó que otro aporte de las practicas es ayudar al analista a adquirir conocimiento acerca de la importancia de cada RM generado, que sub-características de mantenibilidad se pueden potenciar al implementarlo y los elementos abordados.

6.2. Lecciones aprendidas

La realización de este trabajo de investigación permitió adquirir las siguientes experiencias:

- Con el fin de generar propuestas que satisfagan necesidades encontradas en la literatura es necesario realizar mapeos o revisiones de la literatura respaldados con bases de datos científicas como: Scopus, Science Direct IEEE y ACM.
- Dentro de los procesos de investigación es necesario que el investigador sea ordenado frente a la recolección y documentación de las ideas que puedan ir surgiendo durante el proceso, de manera que no se pierdan las reflexiones y puedan ser utilizadas en la estructuración de la solución.
- El uso de herramientas complementarias tales como: el diccionario de la lengua española (RAE) y el diccionario de sinónimos y antónimos son importantes para el proceso de investigación en el momento de la redacción del documento de trabajo de grado ya que permiten utilizar las palabras adecuadas en el contexto de la investigación.
- Actividades de supervisión y guía periódica para el avance de la investigación, realizadas por el director del trabajo de grado, permiten un desarrollo continuo de los productos de trabajo y generan un aprendizaje adecuado de los conocimientos involucrados en la investigación.
- La investigación basada únicamente en referentes de la literatura omite diversos aspectos prácticos observables únicamente en entornos reales, por lo tanto, la investigación se debe complementar mediante experimentos, focus group y estudios de caso mientras sea posible.
- Para el éxito del estudio de caso es necesario que los involucrados tengan disponibilidad de tiempo y actitud para adquirir el conocimiento generado en la investigación, realizar las actividades y generar observaciones con una actitud crítica y constructiva.
- Debido a la pandemia la interacción con los directores y codirectores fue realizada de manera virtual, lo cual implicó utilizar diferentes herramientas informáticas para lograr

desarrollar los diferentes componentes de las prácticas y objetivos del proyecto de investigación. En este sentido es posible realizar un proyecto de manera virtual si se tienen las tecnologías disponibles y la disciplina.

6.3 Trabajos futuros

En este trabajo se han analizado algunos puntos que pueden ser tenidos en cuenta para trabajos futuros:

- Crear una herramienta software que permita la captura y especificación de RMs a través de las prácticas propuesta.
- Aplicar y analizar el comportamiento de las prácticas en otros estudios de caso en empresas más grandes.
- Desarrollar prácticas para otros requisitos de calidad tales como fiabilidad, seguridad, usabilidad, teniendo en cuenta los 10 campos descritos.
- Creación de un artículo científico para compartir los resultados obtenidos en la investigación.
- Incluir en las prácticas propuestas lineamientos para la interacción de los analistas, clientes y equipo de desarrollo cuando se están capturando los RMs.

Referencias bibliográficas

[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, no. 2, pp. 212–238, 2012.

[2] X. Luo, Z. Ge, F. Guan, and Y. Yang, "A method for the maintainability assessment at design stage based on maintainability attributes," in *2017 IEEE International Conference on Prognostics and Health Management (ICPHM)*, 2017, pp. 187–192.

[3] ISO/IEC, "ISO-IEC 25010: 2011 Systems and Software Engineering-Systems and Software Quality Requirements and Evaluation (SQuaRE)-System and Software Quality Models," 2011, doi: https://doi.org/10.3403/30215101u.

[4] S. W. L. Yip and T. Lam, "A software maintenance survey," in *Proceedings of 1st Asia-Pacific Software Engineering Conference*, 1994, pp. 70–79.

[5] C. E. H. Chua, S. Purao, and V. C. Storey, "Developing maintainable software: The Readable approach," *Decision Support Systems*, vol. 42, no. 1, pp. 469–491, 2006.

[6] F. J. Pino, F. Ruiz, F. Garcia, and M. Piattini, "A software maintenance methodology for small organizations: Agile_MANTEMA," *Journal of Software: Evolution and Process*, vol. 24, no. 8, pp. 851–876, 2012.

[7] B. Kumar, "A survey of key factors affecting software maintainability," in *2012 international conference on computing sciences*, 2012, pp. 261–266.

[8] B. Seref and O. Tanriover, "Software code maintainability: A literature review," *International Journal of Software Engineering & Applications (IJSEA)*, vol. 7, pp. 69–87, 2016.

[9] J. D. Erazo, A. S. Florez, and F. J. Pino, "Análisis y clasificación de atributos de mantenibilidad del software: una revisión comparativa desde el estado del arte," *Entre Ciencia e Ingeniería*, vol. 10, no. 19, pp. 40–49, 2016.

[10] R. Saini, S. K. Dubey, and A. Rana, "Analytical study of maintainability models for quality evaluation," *Indian Journal of Computer Science and Engineering*, vol. 2, no. 3, pp. 449–454, 2011.

[11] R. Malhotra and K. Lata, "A systematic literature review on empirical studies towards prediction of software maintainability".

[12] A. Abran, K. T. Al-Sarayreh, and J. J. Cuadrado-Gallego, "Measurement Model of Software Requirements Derived from System Maintainability Requirements.," in *SEKE*, 2010, pp. 153–158.

[13] L. Liu, X. Zhu, and Y. Wang, "Software maintainability requirements modeling based on UML Profile," in *Proceedings of the IEEE 2012 Prognostics and System Health Management Conference (PHM-2012 Beijing)*, 2012, pp. 1–4.

[14] M. Mari, "The impact of maintainability on component-based software systems," in *null*, 2003, p. 25.

[15] S. Velmourougan, P. Dhavachelvan, R. Baskaran, and B. Ravikumar, "Software development life cycle model to improve maintainability of software applications," in *2014 Fourth International Conference on Advances in Computing and Communications*, 2014, pp. 270–273.

[16] F. J. Pino, M. Piattini, and G. Horta Travassos, “Managing and developing distributed research projects in software engineering by means of action-research,” *Revista Facultad de Ingeniería Universidad de Antioquia*, no. 68, pp. 61–74, 2013.

[17] P. Runeson and M. Höst, “Guidelines for conducting and reporting case study research in software engineering,” *Empirical software engineering*, vol. 14, no. 2, pp. 131–164, 2009.

[18] I. Sommerville, “Software engineering 9th,” 2011.

[19] R. Singh, “International Standard ISO/IEC 12207 software life cycle processes,” *Software Process Improvement and Practice*, vol. 2, no. 1, pp. 35–50, 1996.

[20] P. Bourque and R. Dupuis, “Guide to the Software Engineering Body of Knowledge Version 3.0 SWEBOK. IEEE, 2014.”

[21] S. del Valle Rojo and A. Oliveros, “Elicitación y especificación de requerimientos no funcionales para aplicaciones web,” 2014.

[22] L. M. Cysneiros and E. Yu, “Non-functional requirements elicitation,” in *Perspectives on software requirements*, Springer, 2004, pp. 115–138.

[23] ISO and IEC, “ISO/IEC-25010:2011 ISO/IEC. . Systems and software engineering-Systems and software Quality Requirements and Evaluation (SQuaRE) - System and softwre quality models.,” 2011.

[24] ISO/IEC 2011 © IEEE 2011, “ISO/IEC/IEEE 29148:2011 Systems and software engineering — Life cycle processes — Requirements engineering ,” 2011.

[25] F. J. Pino, F. Ruiz, F. Garcia, and M. Piattini, “A software maintenance methodology for small organizations: Agile_MANTEMA,” *Journal of Software: Evolution and Process*, vol. 24, no. 8, pp. 851–876, 2012.

[26] B. Kitchenham and S. Charters, “Guidelines for performing systematic literature reviews in software engineering,” 2007.

[27] 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, no. 2, pp. 212–238, 2012.

[28] X. Luo, Z. Ge, F. Guan, and Y. Yang, “A method for the maintainability assessment at design stage based on maintainability attributes,” in *2017 IEEE International Conference on Prognostics and Health Management (ICPHM)*, 2017, pp. 187–192.

[29] L. Liu, X. Zhu, and Y. Wang, “Software maintainability requirements modeling based on UML Profile,” in *Proceedings of the IEEE 2012 Prognostics and System Health Management Conference (PHM-2012 Beijing)*, 2012, pp. 1–4.

[30] J. D. Erazo, A. S. Florez, and F. J. Pino, “Análisis y clasificación de atributos de mantenibilidad del software: una revisión comparativa desde el estado del arte,” *Entre Ciencia e Ingeniería*, vol. 10, no. 19, pp. 40–49, 2016.

[31] A. Abran, K. T. Al-Sarayreh, and J. J. Cuadrado-Gallego, “Measurement Model of Software Requirements Derived from System Maintainability Requirements.,” in *SEKE*, 2010, pp. 153–158.

[32] M. Mari, “The impact of maintainability on component-based software systems,” in *null*, 2003, p. 25.

[33] S. Velmourougan, P. Dhavachelvan, R. Baskaran, and B. Ravikumar, “Software development life cycle model to improve maintainability of software applications,” in *2014 Fourth International Conference on Advances in Computing and Communications*, 2014, pp. 270–273.

[34] S. Rochimah, P. G. Nuswantara, and R. J. Akbar, “Analyzing the Effect of Design Patterns on Software Maintainability: A Case Study,” in *2018 Electrical Power, Electronics, Communications, Controls and Informatics Seminar (EECCIS)*, 2018, pp. 326–331.

[35] C. Chen, M. Shoga, B. Li, and B. Boehm, “Assessing software understandability in systems by leveraging fuzzy method and linguistic analysis,” *Procedia Computer Science*, vol. 153, pp. 17–26, 2019.

[36] E. Figueiredo, C. Sant’Anna, A. Garcia, T. T. Bartolomei, W. Cazzola, and A. Marchetto, “On the maintainability of aspect-oriented software: A concern-oriented measurement framework,” in *2008 12th European Conference on Software Maintenance and Reengineering*, 2008, pp. 183–192.

[37] J. Erazo Martínez, A. Florez Gómez, and F. J. Pino, “Generando productos software mantenibles desde el proceso de desarrollo: El modelo de referencia MANTuS,” *Ingeniare. Revista chilena de ingeniería*, vol. 24, no. 3, pp. 420–434, 2016.

[38] 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, no. 2, pp. 212–238, 2012.

[39] J. M. Conejero *et al.*, “Early evaluation of technical debt impact on maintainability,” *Journal of Systems and Software*, vol. 142, pp. 92–114, 2018.

[40] 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, no. 2, pp. 212–238, 2012.

[41] R. E. Al-Qutaish, “Quality models in software engineering literature: an analytical and comparative study,” *Journal of American Science*, vol. 6, no. 3, pp. 166–175, 2010.

[42] S. SPROGE and R. CEVERE, “Quality models in software product development life cycle,” 2012.

[43] J. A. McCall, P. K. Richards, and G. F. Walters, *Factors in software quality: concepts and definitions of software quality*. Rome air development center, Air force Systems command, 1977.

[44] R. G. Dromey, “A model for software product quality,” *IEEE Transactions on software engineering*, vol. 21, no. 2, pp. 146–162, 1995.

[45] J. Bansiya and C. G. Davis, “A hierarchical model for object-oriented design quality assessment,” *IEEE Transactions on software engineering*, vol. 28, no. 1, pp. 4–17, 2002.

[46] ISO/IEC, “ISO/IEC FDIS 9126-1:2000(E) Information technology — Software product quality,” 2000.

[47] S. D. Kim and J. H. Park, “C-QM: A Practical Quality Model for Evaluating COTS Components.,” in *Applied Informatics*, 2003, pp. 991–996.

[48] K. Gupta and A. Chug, “Evaluation of instance-based feature subset selection algorithm for maintainability prediction,” in *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, 2017, pp. 1482–1487.

[49] U. Dayanandan and K. Vivekanandan, “An empirical evaluation model for software architecture maintainability for object oriented design,” in *Proceedings of the International Conference on Informatics and Analytics*, 2016, pp. 1–4.

[50] J. de A. G. SARAIVA, “Classifying metrics for assessing object-oriented software maintainability: a family of metrics’ catalogs,” 2014.

[51] Y. Suresh, J. Pati, and S. K. Rath, “Effectiveness of software metrics for object-oriented system,” *Procedia technology*, vol. 6, pp. 420–427, 2012.

[52] M. Kretsou, E.-M. Arvanitou, A. Ampatzoglou, I. Deligiannis, and V. C. Gerogiannis, “Change impact analysis: A systematic mapping study,” *Journal of Systems and Software*, p. 110892, 2020.

[53] I. Malavolta, R. Verdecchia, B. Filipovic, M. Bruntink, and P. Lago, “How maintainability issues of android apps evolve,” in *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2018, pp. 334–344.

[54] A. V. Phan, P. N. Chau, M. Le Nguyen, and L. T. Bui, “Automatically classifying source code using tree-based approaches,” *Data & Knowledge Engineering*, vol. 114, pp. 12–25, 2018.

[55] J. L. Font, P. Iñigo, M. Domínguez, J. L. Sevillano, and C. Amaya, “Analysis of source code metrics from ns-2 and ns-3 network simulators,” *Simulation Modelling Practice and Theory*, vol. 19, no. 5, pp. 1330–1346, 2011.

[56] E. M. Arvanitou, A. Ampatzoglou, A. Chatzigeorgiou, M. Galster, and P. Avgeriou, “A mapping study on design-time quality attributes and metrics,” *Journal of Systems and Software*, vol. 127, pp. 52–77, 2017.

[57] I. Chowdhury and M. Zulkernine, “Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities,” *Journal of Systems Architecture*, vol. 57, no. 3, pp. 294–313, 2011.

[58] D. Rodríguez, R. Ruiz, J. C. Riquelme, and J. S. Aguilar–Ruiz, “Searching for rules to detect defective modules: A subgroup discovery approach,” *Information Sciences*, vol. 191, pp. 14–30, 2012.

[59] D. Durisic, M. Nilsson, M. Staron, and J. Hansson, “Measuring the impact of changes to the complexity and coupling properties of automotive software systems,” *Journal of Systems and Software*, vol. 86, no. 5, pp. 1275–1293, 2013.

[60] D. Feitosa, A. Ampatzoglou, A. Gkortzis, S. Bibi, and A. Chatzigeorgiou, “CODE reuse in practice: Benefiting or harming technical debt,” *Journal of Systems and Software*, vol. 167, p. 110618, 2020.

[61] F. A. Fontana, V. Lenarduzzi, R. Roveda, and D. Taibi, “Are architectural smells independent from code smells? An empirical study,” *Journal of Systems and Software*, vol. 154, pp. 139–156, 2019.

[62] A.-J. Molnar and S. Motogna, “Longitudinal Evaluation of Open-Source Software Maintainability,” *arXiv preprint arXiv:2003.00447*, 2020.

[63] K. A. M. Ferreira, M. A. S. Bigonha, R. S. Bigonha, L. F. O. Mendes, and H. C. Almeida, “Identifying thresholds for object-oriented software metrics,” *Journal of Systems and Software*, vol. 85, no. 2, pp. 244–257, 2012.

[64] H. Mumtaz, P. Singh, and K. Blincoe, “A systematic mapping study on architectural smells detection,” *Journal of Systems and Software*, p. 110885, 2020.

[65] E. Fernandes *et al.*, “Refactoring effect on internal quality attributes: What haven’t they told you yet?,” *Information and Software Technology*, vol. 126, p. 106347, 2020.

[66] S. Gupta and A. Chug, “Assessing Cross-Project Technique for Software Maintainability Prediction,” *Procedia Computer Science*, vol. 167, pp. 656–665, 2020.

[67] C. Y. Chong and S. P. Lee, “Analyzing maintainability and reliability of object-oriented software using weighted complex network,” *Journal of Systems and Software*, vol. 110, pp. 28–53, 2015.

[68] P. He, B. Li, X. Liu, J. Chen, and Y. Ma, “An empirical study on software defect prediction with a simplified metric set,” *Information and Software Technology*, vol. 59, pp. 170–190, 2015.

[69] F. A. Fontana and M. Zanoni, “Code smell severity classification using machine learning techniques,” *Knowledge-Based Systems*, vol. 128, pp. 43–58, 2017.

[70] A. Yamashita and S. Counsell, “Code smells as system-level indicators of maintainability: An empirical study,” *Journal of Systems and Software*, vol. 86, no. 10, pp. 2639–2653, 2013.

[71] M. R. Jose, “A theoretical framework for the maintainability model of aspect oriented systems,” *Procedia Computer Science*, vol. 62, pp. 505–512, 2015.

[72] A. S. Nuñez-Varela, H. G. Pérez-Gonzalez, F. E. Martínez-Perez, and C. Soubervielle-Montalvo, “Source code metrics: A systematic mapping study,” *Journal of Systems and Software*, vol. 128, pp. 164–197, 2017.

[73] T. Mariani, T. E. Colanzi, and S. R. Vergilio, “Preserving architectural styles in the search based design of software product line architectures,” *Journal of Systems and Software*, vol. 115, pp. 157–173, 2016.

[74] D. Albuquerque, B. Cafeo, A. Garcia, S. Barbosa, S. Abrahao, and A. Ribeiro, “Quantifying usability of domain-specific languages: An empirical study on software maintenance,” *Journal of Systems and Software*, vol. 101, pp. 245–259, 2015.

[75] G. A. S. Sheela and A. Aloysius, “Design and analysis of aspect oriented metric CWCoAR using cognitive approach,” in *2017 World Congress on Computing and Communication Technologies (WCCCT)*, 2017, pp. 195–197.

[76] E. Fregnan, T. Baum, F. Palomba, and A. Bacchelli, “A survey on software coupling relations and tools,” *Information and Software Technology*, vol. 107, pp. 159–178, 2019.

[77] G. Szőke, G. Antal, C. Nagy, R. Ferenc, and T. Gyimóthy, “Empirical study on refactoring large-scale industrial systems and its effects on maintainability,” *Journal of Systems and Software*, vol. 129, pp. 107–126, 2017.

[78] R. Malhotra and M. Khanna, “Threats to validity in search-based predictive modelling for software engineering,” *IET Software*, vol. 12, no. 4, pp. 293–305, 2018.

[79] T. Siddiqui and A. Ahmad, “Mining Software Repositories for Software Metrics (MSR-SM): Conceptual Framework”.

[80] N. Yoshida, M. Kinoshita, and H. Iida, “A cohesion metric approach to dividing source code into functional segments to improve maintainability,” in *2012 16th European Conference on Software Maintenance and Reengineering*, 2012, pp. 365–370.

[81] R. Setiawan, Z. E. Rasjid, and A. Effendi, “Design metric indicator to improve quality software development (Study case: Student desk portal),” *Procedia Computer Science*, vol. 135, pp. 616–623, 2018.

[82] N. Gorla and S.-C. Lin, “Determinants of software quality: A survey of information systems project managers,” *Information and Software Technology*, vol. 52, no. 6, pp. 602–610, 2010.

[83] M. Jackson, “ISO/IEC. . Systems and software engineering-Systems and software Quality Requirements and Evaluation (SQuaRE) - System and softwre quality models.,” *Switzerland ISOIEC Towards a System of System Methodologies Journal of Operational Research Society*, vol. 35 SRC-, pp. 473–486, 1984.

[84] E.-M. Arvanitou, A. Ampatzoglou, A. Chatzigeorgiou, and J. C. Carver, “Software engineering practices for scientific software development: A systematic mapping study,” *Journal of Systems and Software*, p. 110848, 2020.

[85] A. V. Phan, P. N. Chau, M. Le Nguyen, and L. T. Bui, “Automatically classifying source code using tree-based approaches,” *Data & Knowledge Engineering*, vol. 114, pp. 12–25, 2018.

[86] M. R. Jose, “A theoretical framework for the maintainability model of aspect oriented systems,” *Procedia Computer Science*, vol. 62, pp. 505–512, 2015.

[87] ISO/IEC/, IEEE, and 29148, “Systems and software engineering — Life cycle processes — Requirements engineeringSystems and software engineering — Life cycle processes — Requirements engineering,” 2011.

[88] IBM, “Breaking Down Software Development Roles,” 2006.

[89] S. Kiwi, “Software Development Team Structure: Important Roles & Responsibilities.” https://steelkiwi.com/blog/software-development-team-structure/

[90] P. Borges, P. Monteiro, and R. J. Machado, “Mapping RUP roles to small software development teams,” in *International Conference on Software Quality*, 2012, pp. 59–70.

[91] M. Margie Semilof, “software stack,” 2020. https://searchapparchitecture.techtarget.com/definition/software-stack

[92] TUTORIALS POINT- simply easy learning, “log4j - Logging Levels,” 2021. https://www.tutorialspoint.com/log4j/log4j_logging_levels.htm

[93] IBM, “Log levels,” 2021. https://www.ibm.com/docs/en/was/8.5.5?topic=logger-log-levels

[94] K. Yin Robert, “Case study research and applications: design and methods.” Los Angeles, CA: Sage Publications, 2017.

[95] M. Host and P. Runeson, “Checklists for software engineering case study research,” in *First international symposium on empirical software engineering and measurement (ESEM 2007)*, 2007, pp. 479–481.