APROXIMACIÓN METODOLÓGICA PARA LA CREACIÓN DE INFORMES DE FALLOS Y MEJORES PRÁCTICAS PARA EL PROCESO DE MANTENIMIENTO DE SOFTWARE

VÍCTOR HUGO MERCADO

UNIVERSIDAD DE MEDELLÍN
FACULTAD DE INGENIERÍA
MAESTRÍA EN INGENIERÍA DE SOFTWARE
MEDELLÍN
2019

APROXIMACIÓN METODOLÓGICA PARA LA CREACIÓN DE INFORMES DE FALLOS Y MEJORES PRÁCTICAS PARA EL PROCESO DE MANTENIMIENTO DE SOFTWARE

VÍCTOR HUGO MERCADO

Trabajo de grado para optar al título de
Magister en Ingeniería de Software

Director
Bell Manrique Losada, Ph.D.
Codirector
Juan Ricardo Cogollo, Ph.D.(c)

UNIVERSIDAD DE MEDELLÍN
FACULTAD DE INGENIERÍA
MAESTRÍA EN INGENIERÍA DE SOFTWARE
MEDELLÍN
2019

Dedicado a mis padres. Gracias por educarme con bases sólidas que han hecho de mi la persona que soy, porque su amor y apoyo siempre están presente sin importar los kilómetros que nos separan físicamente; a mis hermanos y a esas personas que de una u otra forma aportaron su grano de arena para que esto sea una realidad.

AGRADECIMIENTOS

Agradezco de manera muy especial a mis asesores y docentes que aportaron su conocimiento y nutrieron mi ser, que me guiaron en el proceso de investigación que concluye con este trabajo, el cual me deja un gran aprendizaje.

Quiero agradecer a la Universidad de Medellín, por permitirme ser parte de su familia, por prestar sus instalaciones para que este proceso pudiera ser una realidad; a Satrack, porque su apoyo inicial fue fundamental y a la Universidad de Antioquia por abrirme sus puertas y creer en mí.

A mis amigos, compañeros de estudio y de trabajo, que me ayudaron y siempre estuvieron prestos para lo que necesitara, muchas gracias por su buena vibra y buena energía.

CONTENIDO

pág.

LISTA DE TABLAS

LISTA DE FIGURAS

pág.

LISTA DE ANEXOS

RESUMEN

El mantenimiento es una fase del ciclo de vida del software donde se mejora el software que se encuentra en funcionamiento, ya sea para adaptarse a cambios o para corregir fallos que se detectan durante la operación, este trabajo se enfoca en el mantenimiento correctivo, teniendo en cuenta que de este depende la operatividad del software en el tiempo. Sin embargo, hacer modificaciones al software en producción no es una tarea fácil y menos si se realizan sin usar buenas prácticas, esto genera retrasos en los tiempos de solución de los fallos e inyección de fallos nuevos al momento de realizar correcciones.

El objetivo de esta investigación es proponer un acercamiento metodológico para abordar el proceso de mantenimiento de software que incluye buenas prácticas desde que se reporta un fallo hasta que se resuelve. Este acercamiento también propone unos artefactos que ayudarán a mejorar el proceso de atención y corrección de fallos.

Palabras clave: Mantenimiento, buenas prácticas, reporte de fallos, bugs, metodologías.

INTRODUCCIÓN

El siguiente trabajo está organizado en diferentes capítulos, orientados al desarrollo de los objetivos propuestos para esta investigación, que buscan proponer un acercamiento metodológico con buenas prácticas para la creación de informes de fallos y para el proceso general de mantenimiento de software.

El mantenimiento es una fase del ciclo de vida del software, que inicia con la liberación del software al usuario final. En adelante, como mantenimiento se consideran los nuevos cambios ya sea por fallos o por nuevas necesidades del negocio. Este trabajo se enfoca en el mantenimiento correctivo, que se realiza cuando el sistema presenta fallas que afectan la operación.

El objetivo de esta investigación es proponer un acercamiento metodológico para el proceso de mantenimiento de software, en términos de 4 fases: Reporte del Fallo, Pre análisis, Análisis e Implementación, Adicionalmente se proponen algunas actividades y prácticas que son trasversales al proceso.

Esta propuesta fue implementada en un caso de estudio en una compañía en la ciudad de Medellín -Colombia, donde se pudieron tener evaluaciones de la misma y determinar oportunidades de mejora. Finalmente se concluye que la propuesta realizada es pertinente y ayuda al proceso de mantenimiento. Se espera que, en trabajos futuros, teniendo esta propuesta desarrollada, se realice un análisis de cómo medir las diferentes prácticas propuestas para poder determinar indicadores de mejora.

PARTE I

PRÓLOGO

"Lo único constante es el cambio" – Heráclito

CAPÍTULO 1 PLANTEAMIENTO DEL PROBLEMA

En el siguiente capítulo se describen los diferentes apartes que dan origen al planteamiento del problema de investigación para esta propuesta, por lo cual se brinda una contextualización del tema en general para llegar al problema particular.

1.1. JUSTIFICACIÓN

El objetivo de este trabajo es contribuir a la mejora del proceso que realizan los equipos de mantenimiento en compañías que desarrollan software, específicamente mejorar los tiempos de atención, análisis, corrección y calidad de implementación cuando se hacen mantenimientos correctivos.

Para una compañía puede resultar más costoso hacer mantenimiento al software que desarrollarlo [1], ello se justifica en la alta rotación de personal, la escasa documentación del software y la baja calidad de los informes de fallos[2]. En el contexto de compañías donde existen equipos de mantenimiento, se necesita disponer de prácticas que garanticen que un informe de fallo contiene la información que el desarrollador necesita para proceder con la solución de manera efectiva. En este trabajo se propone una aproximación metodológica que incluye la selección de buenas prácticas, de forma que sirva de base futura para posible implementación de algoritmos que permitan clasificación automática de informes de fallos y su respectivo análisis para intentar detectar fallos en el código fuente.

La notificación / corrección de fallos es una parte importante del proceso de desarrollo de software, que normalmente implica la coordinación de múltiples individuos [3], esto agrega relevancia a la necesidad de la disponer de una metodología que guie el proceso de mantenimiento correctivo, para orientar por una ruta optima a los roles que se involucran en las tareas de atención y solución de fallos. Adicionalmente, los desarrolladores no suelen registrar los enlaces entre los informes de fallos en un sistema de seguimiento de problemas y los correspondientes cambios de fijación en un repositorio de versiones [4]. Por lo anterior,

esta propuesta considera las mejores prácticas para la corrección de fallos, mitigando impactos que afectan el seguimiento y generan pérdida de conocimiento a la hora de atender nuevos casos.

En este sentido, el diseñar una aproximación metodológica como esta, permitiría contar con: Un esquema de mejores prácticas a implementar en todo el proceso de mantenimiento correctivo desde el reporte del fallo hasta la corrección de este.

1.2. PROBLEMA DE INVESTIGACIÓN

La ingeniería de Software se define como "la aplicación de un sistema sistemático, disciplinado, con enfoque cuantificable para el desarrollo, operación y mantenimiento de software" [5], esta disciplina nació a raíz de la crisis del software que se presentó a finales de la década de los 60 y principios de la década de los 70, con ello nace el concepto de ciclo de vida del software, y sus fases Análisis de requisitos, Diseño, Construcción, Pruebas y Mantenimiento.

Una de las fases fundamentales del ciclo de vida del software, es el mantenimiento, del cual depende que el software sea operativo y utilizado la mayor cantidad de tiempo posible [6]. En esta fase se definen diferentes categorías de mantenimiento tales como el correctivo, adaptativo, perfectivo y preventivo [7]. Esta investigación se enfoca en el mantenimiento correctivo, el cual se define como toda modificación reactiva (o reparaciones) de un producto de software luego de su entrega, para corregir los problemas descubiertos [8]. En esta categoría se incluye el mantenimiento de emergencia, que es una modificación no programada que se realiza para mantener temporalmente un producto de software operativo en espera del mantenimiento correctivo [8].

Muchas técnicas se desarrollaron a lo largo de los años para detectar automáticamente fallos en el software. A menudo, estas técnicas se basan en métodos formales y sofisticados

análisis de programas. Si bien estas técnicas son valiosas, pueden ser difíciles de aplicar, y no siempre son eficaces para encontrar fallos reales [9], ya que cada software tiene sus propias particularidades.

Es importante tener en cuenta que los informes de fallos son el principal medio por el cual los usuarios de un sistema pueden comunicar un problema a los desarrolladores, y su contenido es importante no sólo para apoyarlos en el mantenimiento del sistema, sino también como base de herramientas automatizadas para ayudar en las desafiantes tareas de encontrar y corregir fallos, esto sería posible llenando repositorios de bugs de minería para la información, a través del desarrollo de herramientas para apoyar actividades como la localización de errores, y esto podría ayudar hasta en la construcción de la próxima generación de sistemas de rastreo de errores [10].

Lo anterior dado que los desarrolladores de software se basan en información textual esencial de informes de fallos (como comportamiento observado, comportamiento esperado y pasos para reproducir) para clasificar y solucionar fallos de software [11], pero realmente se encuentra que existe un claro desajuste entre la información que los desarrolladores desearían ver en un informe de fallo y la información que realmente aparece [10]. Este tipo de novedades aumentan el nivel de complejidad al momento de analizar, detectar y corregir fallos en los sistemas de información.

Actualmente es muy complejo obtener informes de fallos de alta calidad que permitan a los equipos de desarrollo generar de forma automática una clasificación de ellos [12]. Para lo anterior hay que tener en cuenta el problema de comunicación que se da entre las dos partes involucradas en este proceso, quien reporta el fallo y quien lo corrige. Para quien lo reporta algunas veces es complejo ser muy específico debido a su bajo conocimiento técnico, si este es un usuario final, y quien recibe el informe el desarrollador se le dificulta entender que está reportando el usuario. En este sentido, los intereses de información al momento de generar el informe de fallos son diferentes, lo cual genera que el desarrollador quien usa estos informes como un insumo fundamental, a la hora de dar solución a la incidencia le sea mucho más difícil analizar el fallo y determinar su ubicación y solución.

En este sentido esta propuesta busca formular una aproximación metodológica que contenga una serie de mejores prácticas desde el momento de generar un reporte de fallos hasta la corrección, facilitando el análisis de un estado inicial y permitiendo que a futuro en otras investigaciones esta propuesta pueda ser una base para que se puedan ejecutar herramientas de detección automática de los fallos que se reportan, en caso de contar con estas. Hasta el momento de la fijación o corrección de fallos, buscando mitigar los impactos negativos a la hora de atender y solucionar el incidente reportado.

1.3. HIPÓTESIS

Aplicando una aproximación metodológica para el proceso de mantenimiento correctivo, se facilitarían el reporte de fallos, la comprensión del reporte por parte de los desarrolladores, y la implementación de la solución.

1.3.1. Pregunta de investigación

¿Puede lo anterior reducir la cantidad de fallos que se inyectan durante la corrección, el número de reprocesos y el tiempo de solución en un proceso de mantenimiento de software?

1.4. OBJETIVOS DE LA INVESTIGACIÓN

El objetivo general es proponer una aproximación metodológica que incluya un conjunto de mejores prácticas en el proceso de mantenimiento correctivo, para aplicar en equipos de desarrollo y soporte de software.

Teniendo en cuenta el objetivo general se proponen los siguientes objetivos específicos, con el fin de lograr lo planteado y buscar responder la pregunta formulada en la hipótesis:

a. Explorar las técnicas e instrumentos propuestos en investigaciones anteriores para el reporte, análisis y fijación de fallos e identificar las mejores prácticas implementadas en estos.
b. Definir las etapas y los componentes que debería tener una metodología para el proceso de mantenimiento correctivo, a partir del análisis de las técnicas exploradas
c. Especificar un conjunto de mejores prácticas para todo el proceso de mantenimiento correctivo de software.
d. Proponer un acercamiento metodológico, que incluya las mejores prácticas para el proceso de mantenimiento correctivo de software basado en los objetivos anteriores
e. Validar la metodología propuesta aplicándola en un equipo de mantenimiento de una compañía particular, para identificar las mejoras en facilidad del proceso de reporte de fallos, el número de fallos detectados y la inyección de fallos en la corrección.

1.5. ESTRUCTURA DE LA TESIS

Este trabajo está organizado en 7 capítulos agrupados en 6 partes, como se ilustra en la **Figura 1**.

A continuación, se describen brevemente los capítulos de este trabajo con su respectivo contenido:

Error! Reference source not found. **- INTRODUCCIÓN**
CAPÍTULO 1. Introducción: En este capítulo se busca introducir al lector en el tema a abordar en el resto del documento, describiendo un poco el mantenimiento de software y los problemas que actualmente enfrenta esta parte del ciclo de vida del software, se muestran los objetivos de la investigación y una hipótesis que busca dar solución al problema que se plantea.

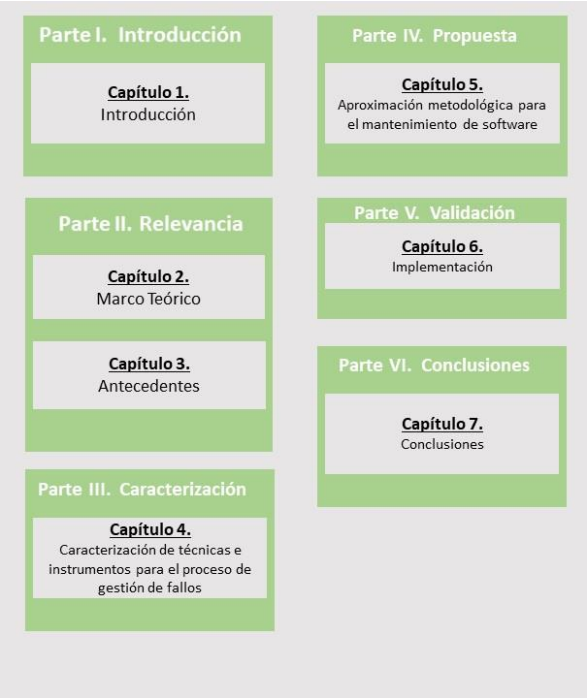


Figura 1. Estructura de la tesis

Error! Reference source not found. **– RELEVANCA**

Error! Reference source not found.. Marco teórico: describe diferentes conceptos y modelos de calidad que apoyan la investigación realizada, con el objetivo de dar mayor claridad al lector en el documento en general.

Error! Reference source not found.. Antecedentes: la revisión de la literatura es el marco central de este capítulo, donde se describe el estado actual del mantenimiento de software y la generación de reportes de fallos, se realiza un análisis de las propuestas formuladas por diferentes autores que buscan dar solución al problema en general es decir todo el clico de vida del fallo o algún problema específico, por ejemplo, algunos autores se enfocan en el reporte de fallos otros en la implementación de la solución, etc.

Error! Reference source not found.**II - CARACTERIZACIÓN**

Error! Reference source not found.. Caracterización de técnicas e instrumentos para el proceso de gestión e fallos: se describe todo el proceso realizado para la selección, evaluación y priorización de las diferentes prácticas y artefactos que harán parte de la

propuesta de solución, en este capítulo también se realiza una revisión a la literatura, pero más enfocada en esa detección de prácticas e instrumentos.

PARTE IV. PROPUESTA

Error! Reference source not found.. Propuesta de la aproximación metodológica, en este capítulo se realiza la presentación de la propuesta solución del problema, donde se muestran las diferentes fases del proceso de mantenimiento, las prácticas e instrumentos que se recomienda implementar para lograr mejorar dicho proceso.

Error! Reference source not found. **- VALIDACIÓN**

Error! Reference source not found.. Implementación, en este capítulo se describe el caso de estudio en el cual se realiza el proceso de implementación y se realiza la evaluación de la propuesta documentada en el capítulo anterior.

Error! Reference source not found.7. Análisis de los resultados, en este se muestran los resultados, obtenidos luego de la evaluación y la aplicación de un caso de estudio, donde se verifica si la propuesta es pertinente o no.

Error! Reference source not found. **- CONCLUSIONES**

Error! Reference source not found.8. Finalmente se plantean conclusiones a partir de los resultados y se plantean alternativas de trabajo futuro.

PARTE II

RELEVANCIA

"Puede decirse que el grito de la historia nace con nosotros y que es uno de nuestros dones más importantes. En cierto sentido somos históricos todos los hombres." -- Thomas Carlyle

CAPÍTULO 2 MARCO TEÓRICO

En este capítulo se describen conceptos manejados en el trabajo investigativo, los cuales buscan dar mayor claridad al lenguaje utilizado en los diferentes apartes que conforman el documento.

2.1. INGENIERIA DE SOFTWARE

La ingeniería de software es una disciplina de la ingeniería que se interesa por todos los aspectos de la producción de software [13]. Este término fue sugerido en conferencias organizadas por la OTAN en 1968 y 1969, Se propuso que la adopción de un enfoque de ingeniería para el desarrollo de software reduciría los costos de desarrollo de software y conduciría a un software más confiable [14]. Esta disciplina busca fortalecer todo el ciclo de vida del software agregando nuevas técnicas y prácticas en todas sus fases.

2.2. CICLO DE VIDA DEL SOFTWARE

Se denomina ciclo de vida del software, a las fases presentes desde su concepción hasta la desinstalación del mismo [15]. Diferentes organizaciones que se encargan de estándares internacionales generan normas sobre el ciclo de vida, que incluyen fases o etapas que lo componen, algunas de estas organizaciones son la IEEE (Instituto de Ingenieros Eléctricos y Electrónicos) y la ISO/IEC (**International Standards Organization/International Electrochemical Commission**), quienes proponen definiciones del ciclo de vida muy cercanas, y se divide en fases, con el objetivo de tener trazabilidad y control, desde el momento que se inicia el proceso de creación de una solución de software hasta que dicho software se desinstala y deja de usarse.

2.3. MANTENIMIENTO DE SOFTWARE

El proceso de mantenimiento se centra en el cambio que se deriva de los fallos que se detectan, las mejoras que se solicitan, y las modificaciones por temas legales o reestructuración de reglas del negocio. Se le considera como una nueva aplicación del ciclo de vida, pero a un software existente en una iteración de desarrollo [15]. El mantenimiento cobra importancia cuando el software es de uso masivo y tiene una larga vida, ya que esto ayuda a que la solución se mantenga funcional y actualizada de acuerdo a las necesidades de los usuarios. La norma ISO 14764 [16] indica cómo gestionar el proceso de mantenimiento de software, divide el proceso de mantenimiento en 6 actividades; implementación del proceso, análisis de problemas y modificaciones, implementación de la modificación, revisión/aceptación de mantenimiento, migración y jubilación, indicando que para activar el proceso de mantenimiento del software debe existir un requisito que lo solicite.

2.4. MANTENIMIENTO CORRECTIVO.

El mantenimiento correctivo se define como aquel que se encarga de corregir fallos en los sistemas, en otras palabras una corrección reactiva y se realiza luego de la entrega de software, según [7].

2.4.1. CICLO DE VIDA DEL MANTENIMIENTO CORRECTIVO DE SOFTWARE

A continuación, se describen las fases que conforman el ciclo de vida del mantenimiento correctivo de software[17] ver **Figura 2** para un mayor entendimiento del proceso de manera visual, basado en el ciclo de vida de un fallos:

Fase 1 – Reporte de la falla: En esta fase es donde el analista, cliente o QA, detecta que se está presentando una novedad o anomalía en el sistema lo que produce una falla, y

decide generar el reporte de esta novedad en un tiquete o por medio de la herramienta que esté disponible para el reporte de las fallas.

Fase 2 – Pre Análisis: En la fase de pre análisis, el analista de desarrollo responsable de recibir el reporte de fallas, hace un breve análisis del caso para determinar si la información que contiene es suficiente para darle atención, en caso que no se tenga información suficiente el reporte será devuelto a quien lo genero con el propósito que este diligencie la información faltante, entonces el cliente o analista que reporta la falla recibe el reporte de vuelta, él debe revisar qué información se le solicita y diligenciarla para volver a enviar el reporte correctamente diligenciado.

En caso de que la información del reporte sea suficiente el analista determinara si atiende el caso o se lo asigna a otro desarrollador y determina un tiempo estimado de solución mediante una estimación donde se le indican tallas o puntos al caso para determinar su grado de complejidad. En este pre-análisis se puede determinar que el reporte no es una falla si no una mala interpretación de la solución, en estos casos el tiquete se le da solución aclarando la duda que se tenga respecto a la usabilidad del producto, o puede ser una solicitud de un nuevo requerimiento entonces se solicita manejarlo por los canales correspondientes.

Fase 3 – Análisis y corrección: En esta fase el analista que tiene el caso asignado realiza la verificación de la falla, analiza y detecta porque se genera y realiza los correctivos correspondientes y las pruebas necesarias para dar solución a la falla reportada, en un ambiente controlado no productivo.

Fase 4 – Implementación de la solución: en esta fase el analista desarrollador hace el paso a producción con los cambios que dan solución a la falla reportada, y se encarga de actualizar y documentar en los diferentes repositorios que se tengan disponibles (en el caso que se tengan).

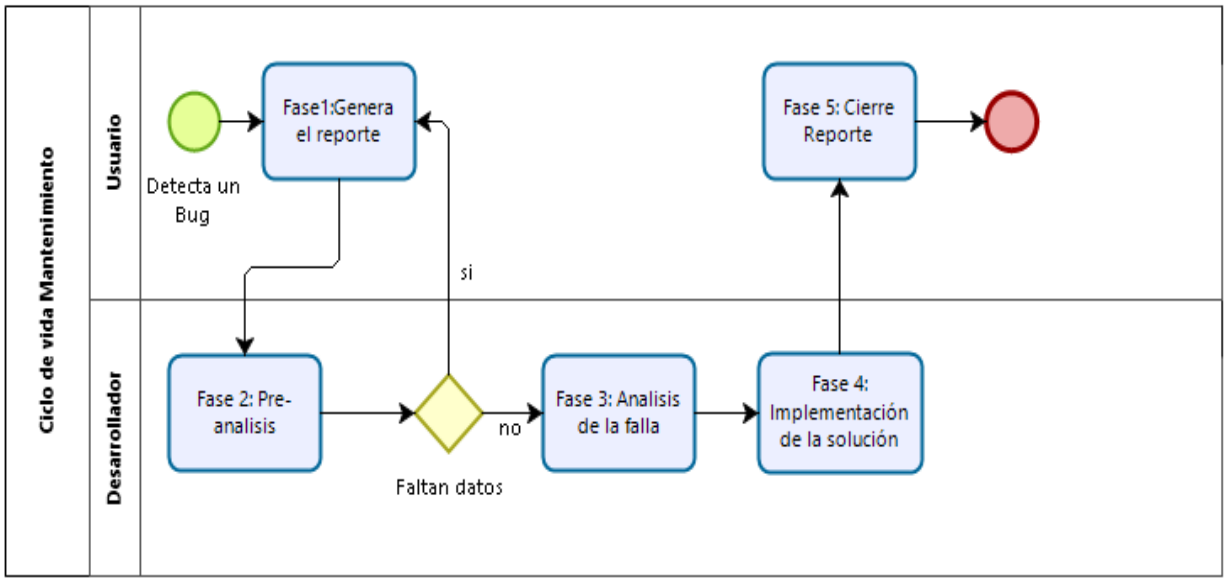


Figura 2. Ciclo de vida del mantenimiento de software

2.5. FALLOS (*Bugs*).

Los fallos son los que bloquean el uso del sistema o de algunos módulos en específico, o no permiten la ejecución de funcionalidades específicas, los cuales son detectados por los usuarios en el momento de usar el sistema y este no les permite ejecutar funcionalidades que debería realizar, o por diferentes sistemas que se tengan para controlar y monitorear el buen funcionamiento de los diferentes sistemas de información [18].

2.6. INFORMES DE FALLOS

Un informe de fallo es un tipo de reporte que se usa para informar al equipo de soporte acerca de problemas en el software [19]. Los informes se pueden generar en el ciclo de desarrollo del software, en este caso la detección del fallo y generación del informe la realiza el equipo de prueba o el usuario final y generalmente se hace para los procesos de calidad

que se tengan para cada proyecto, nuestro objeto de estudio será principalmente en los reportes que se generan cuando el software ya está en producción. Los informes de fallos tienen un ciclo de vida y en este se dan diferentes estados teniendo como estado inicial 'Nuevo' y como estado final 'Resuelto'[20]. Es importante tener en cuenta que del estado Resuelto se puede pasar a otros estados los cuales dependen de la forma en que éste se haya resuelto [21].

2.7. CMMI DEV – PARA DESARROLLO

El modelo CMMI-DEV (CMMI-Para desarrollo) proporciona una orientación para aplicar las buenas prácticas CMMI en una organización de desarrollo. Las buenas prácticas del modelo se centran en las actividades para desarrollar productos y servicios de calidad con el fin de cumplir las necesidades de clientes y usuarios finales [22], este modelo propone mejores prácticas en todo el proceso de desarrollo de software. Sin embargo, no se tiene un área de proceso que se dedique de manera exclusiva al mantenimiento de software, pero a lo largo del modelo es posible encontrar prácticas que se relacionan y se abordar en este trabajo:

- Contratar personas para realizar el mantenimiento y soporte.
- Formar a personas para realizar el mantenimiento y soporte.
- Contratar el mantenimiento y soporte.
- Crear usuarios expertos en las herramientas seleccionadas" [22]

2.8. CMMI PARA SERVICIOS

CMMI-SVC (CMMI para servicios), surge con el fin de cubrir la necesidad de un mercado creciente del sector industrial que ofrece servicios a sus clientes, el cual está contribuyendo significativamente al desarrollo y crecimiento global.

Todas las prácticas de CMMI-SVC se centran en actividades del proveedor de servicios. Siete áreas de proceso se centran en prácticas específicas de servicios, abordando los procesos para la gestión de capacidad y disponibilidad, continuidad del servicio, prestación de servicios, resolución y prevención de incidencias, transición del servicio, desarrollo del sistema de servicio, y gestión estratégica de servicios [23].

Este estándar se tiene en cuenta, ya que el proceso de mantenimiento de software actúa como un servicio en el que tenemos clientes o usuarios que reportan fallos.

2.9. NORMA ISOIEC 25000

ISO/IEC 25000, conocida como SQuaRE (*System and Software Quality Requirements and Evaluation*), es una familia de normas que tiene por objetivo la creación de un marco de trabajo común para evaluar la calidad del producto software.

La familia ISO/IEC 25000 es el resultado de la evolución de otras normas anteriores, especialmente de las normas ISO/IEC 9126, que describe las particularidades de un modelo de calidad del producto software, e ISO/IEC 14598, que abordaba el proceso de evaluación de productos software. Esta familia de normas ISO/IEC 25000 se encuentra compuesta por cinco divisiones [24].

La norma ISO/IEC 25000, al igual que CMMI proporciona Guías en este caso para mejorar la calidad del software, sin embargo estas normas no se enfocan en la calidad del producto final y cómo evaluar si este satisface o no los requerimientos de los usuarios y los estándares esperados, por tanto sigue siendo tema de estudio como hacer seguimiento a esta parte del proceso, es decir luego que el producto ha sido desarrollado y puesto en producción [25]. Esta norma será tenida en cuenta dado que es una guía para obtener software de calidad y el ciclo de mantenimiento de software es internamente un pequeño ciclo de desarrollo de software, con la diferencia que el resultado final no será nuevo software si no software modificado, en el caso de mantenimiento correctivo.

2.10. ISO/IEC 12207

Esta norma internacional establece un marco común para los procesos del ciclo de vida del software, Con una terminología bien definida, que puede ser referenciada por la industria del software. Se aplica a la adquisición de productos y servicios de sistemas y software, para el suministro, desarrollo, operación, mantenimiento y eliminación de productos de software, ya sea que se realice internamente o externamente a una organización [18].

2.11. DEFINICIÓN DE METODOLOGÍA

El término metodología se define como el grupo de mecanismos o procedimientos racionales, empleados para el logro de un objetivo, o serie de objetivos que dirige una investigación científica[26]. Este término se encuentra vinculado directamente con la ciencia, sin embargo, la metodología puede presentarse en otras áreas como la educativa, en donde se encuentra la metodología didáctica o la jurídica en el derecho.

El término metodología se usa en diferentes contextos, tanto científicos como industriales, en esta investigación el contexto que se le da al término se relaciona directamente a la industria del desarrollo de software, en el cual se hace referencia al conjunto de técnicas, procedimientos y soportes documentales que se emplean en el diseño de sistemas de información. Su objetivo principal es exponer una serie de técnicas clásicas y modernas de modelado de sistemas que permitan desarrollar un software de calidad, que incluyen heurísticas de construcción y criterios de comparación de modelos de sistemas [23].

2.12. PRÁCTICAS

De acuerdo a la Real academia de la Lengua, Práctica se define como usar o ejercer actividades continuamente [27].

En el ámbito de ingeniería de software prácticas son todas las actividades que realizan los diferentes equipos que trabajan en torno al desarrollo de software y su ciclo de vida, según [28], estos subdividen las prácticas en diferentes niveles con el fin de proponer mejoras a estas ya que en la ingeniería de software lo que se busca es el mejoramiento continuo del proceso de desarrollo.

2.13. BUENAS PRÁCTICAS

Según Capers Jones [29], las buenas prácticas son las que se pueden clasificar de acuerdo a las líneas de función de cada software, y que se pueda medir sus mejoras en el ciclo de vida del software e identificar sus actores involucrados, ya que indica que existen muchos marcos de trabajo, pero pocos demuestran con datos estadísticos su efectividad, por tanto se considera como buena práctica aquella que ayuda a que aumente la calidad del software, la productividad de los desarrolladores y la satisfacción del cliente. Las prácticas deben permitir generar mediciones, para así indicar que una práctica se califica como mejor sobre otra.

CAPÍTULO 3 ANTECEDENTES

3.1. REVISIÓN DE LITERATURA

En esta investigación se desarrolló un proceso sistemático de revisión del estado del arte para determinar el estado actual del problema que se está analizando, se adaptó la metodología propuesta por Kitchenham [30], quien divide el proceso de revisión de literatura en 3 etapas: planificación de la revisión, desarrollo de la revisión y publicación de los resultados teniendo en cuenta estas etapas el proceso se desarrolló de la siguiente manera:

3.1.1. PLANIFICACIÓN DE LA REVISIÓN

En esta etapa se busca tener claridad sobre el problema a investigar y los objeticos de dicha investigación, para luego de esto iniciar un plan de investigación [31] para la búsqueda de la información se seleccionaron las siguientes fuentes de información:

- Bases de datos: SCOPUS, ScienceDirect, IEEE
- Bibliotecas digitales: ACM, Access engineering

Para esto se utilizaron cadenas de búsqueda como **"***Complexity in software bug fixes"* y "*Debugging Tools"," software lifecycle"," software maintenance and best practices"," bug life cycle"*, las cuales están orientadas a términos y conceptos que nos ayuden a dar respuesta a la pregunta de investigación.

3.1.2. DESARROLLO DE LA REVISIÓN

Luego de tener definidas las fuentes y las cadenas de búsqueda se procedió a iniciar la fase 2 de la metodología guía. fue necesario definir los criterios de inclusión y exclusión debido a que los resultados obtenidos sin criterios, eran muy amplios y genéricos ver **Figura 3**, los criterios de inclusión definidos son: i) que sean de la rama del conocimiento Ingeniería de software, ii) publicaciones mayores al año 2000, iii) orientado al mantenimiento de software,

estos criterios se definieron con el objetivo de obtener resultados que sean más cercanos y alineados al problema de investigación, también se busca que sean publicaciones recientes ya que el problema o caso de estudio está en constante evolución, lo cual requiere la validación y actualización constante de estudios sobre este problema, ver **Figura 4**.

Fuente de Busqueda	# resultados
SCOPUS	83.422
ScienceDirect	2.954
IEEE	1.730
ACM	3.452
Access engineering	2.343
Total	93.901

Figura 3. Resultados de búsqueda inicial

Fuente de Busqueda	# resultados
SCOPUS	2.328
ScienceDirect	973
IEEE	728
ACM	469
Access engineering	749
Total	5.247

Figura 4. Resultados de búsqueda aplicando criterios.

Con los últimos resultados se realizó una selección final de 37 artículos y 4 libros, para esta selección se tuvieron en cuenta los siguientes criterios de inclusión: i) El problema o caso de estudio debe ser abordado, ii) que proponga soluciones con mejores prácticas, iii) que le propuesta implique una validación, estos nos permitieron revisar los diferentes resultados y determinar cuáles se ajustaban mejor a la investigación que se pretende realizar.

Luego de hacer esta selección y con el objeto de un mejor entendimiento del estado del arte del problema a estudiar se subdividen los trabajos priorizados en 4 categorías:

1. Informes de fallos. Estos son la base por medio de la cual los equipos de mantenimiento de software trabajan, ya que su labor es dar por resueltos todos los informes que son notificados.

2. Estimación de los informes de fallos. Se conoce con este nombre a la actividad donde se hace una clasificación inicial de los reportes de fallos con el fin de conocer cuáles son válidos, cuáles no y por qué, de esta manera se optimiza el tiempo de respuesta y solución a los usuarios, ya que muchos no son válidos porque no tienen suficiente información o porque están repetidos, o porque no son fallos como tal sino nuevos requerimientos, entre otras actividades que se desarrollan en esta etapa.
3. Detección de los fallos. En esta categoría se agrupan los artículos que evidencian el estado actual en el que se encuentra este problema según los autores seleccionados, validando qué se ha realizado hasta hoy para que la detección de fallos tanto manualmente como de forma automática sin depender de un desarrollador; y por último la categoría.
4. Solución o fijación de fallos. En esta se busca conocer estado actual del problema en esta fase del proceso de mantenimiento correctivo, buscando detectar las mejores prácticas implementadas para la solución de fallos, con el fin de poder hacerle un seguimiento y documentarlo correctamente.

3.1.3. PUBLICACIÓN DE LOS RESULTADOS

INFORMES DE FALLOS

La calidad en los informes de fallos es un factor determinante en el momento que un ingeniero de software toma uno para solucionarlo [32], dado que si éste no es claro las probabilidades de dar una solución se verán reducidas y será necesario esperar por más información, lo que implica perder tiempo tratando de entender el fallo, O. Chaparro [33] resalta la importancia de tres aspectos fundamentales a la hora de hacer un reporte de fallos: *Comportamiento observado, Comportamiento esperado y Pasos para reproducir*. Ante la ausencia de dichos aspectos la probabilidad de no resolver el fallo informado es alta, por tanto, propone una versión de reporte de fallos que cierre la brecha entre el informador y el ingeniero de desarrollo, que mejore los tiempos de solución y la efectividad del equipo de mantenimiento. A pesar de que el trabajo enuncia una propuesta puntual de una versión de informes de fallos, pero no la presenta, describe como analizaron 2912 reportes de

diferentes proyectos y herramientas como por ejemplo (Jira, GitHub), con lo cual concluyeron que existe una relación directa entre los fallos no resueltos y los fallos con deficiencias en la información, lo cual afecta el análisis y solución por parte de los responsables de soporte.

CLASIFICACIÓN DE INFORMES DE FALLOS

La clasificación de informes de fallos requiere que un analista verifique la descripción de la incidencia que se reporta [34], luego identifica si se puede reproducir o solucionar, si tuvo reporte//solución previa, o si actualmente hay un reporte del mismo fallo. Según Tessone y Schweitzer [35], el Triaje y procesamiento de informes de fallos es una tarea importante en la ingeniería de software, que puede afectar de manera crucial la calidad del producto, la reputación del proyecto, la motivación del usuario y, por tanto, el éxito a largo plazo de un proyecto.

Existe una propuesta que permite mejorar notablemente la predicción del tiempo que requiere un fallo reportado para que se solucione, esta fue desarrollada por Thung [2]. Y en ella se evidencia la necesidad de hacer un cambio a la medición del costo en la corrección de los fallos, ya que generalmente se realiza desde el momento en que se genera el informe del fallo hasta la corrección de este, no siendo esta la mejorar práctica según este autor.

Generalmente los equipos que se dedican al mantenimiento de software no se concentran al 100% en la corrección de fallos, por esto el autor Thung, propuso un "enfoque basado en la clasificación que predice el esfuerzo en términos de líneas aglomeradas". Este consiste en asignar un esfuerzo de fijación de fallos a categorías alta y baja, "a partir de la extracción de las características textuales de las palabras que aparecen en los campos de resumen y descripción de los informes de fallos" [36]. Lo propuesto en esta investigación es una aproximación interesante al momento de hacer una clasificación inicial de informe de fallos, para tener un Triaje donde se indique un estado inicial, además de información del esfuerzo que se requiere para la solución. Esto permitirá tener métricas al final para validar si se está

cumpliendo o si se debe mejorar con los tiempos que el equipo de trabajo tenga como promesa de servicio con sus usuarios.

Para la realización de una estimación en la propuesta de Tessone y Schweitzer [35], se presenta un método para identificar reportes de fallos válidos, lo cual indica si es un fallo de software real, si están duplicados y si contienen suficiente información para ser procesados de inmediato. Esto se basa en 9 métricas, pero al final en la discusión los investigadores indican que este método, aunque no es cerrado y puede aplicar en otros proyectos, requiere de ciertos cambios los cuales no están previstos en la investigación que ellos realizaron. Por consiguiente, este método podría ser válido en el presente proyecto de investigación puesto que genera valor indicando si los informes de fallos son válidos y si estos se deben trabajar o no con prioridad.

Teniendo claro que la comunidad de reporteros de informes de fallos con los que se realizara la validación es reducida pues es un entorno corporativo con un manejo de 5000 reportes al año aproximadamente, y el autor realizó su investigación con un entorno de software libre con un mayor número de reportes de fallos.

Hay un factor que cobra importancia cuando se habla de grandes proyectos y este es la asignación del informe de fallos a un desarrollador ya que, si se hace de forma incorrecta, es decir que se asigna a un desarrollador que desconoce sobre el proyecto al que pertenece el fallo o se asignó a un desarrollador que está cargado con muchos informes por solucionar, el informe podría tardar más tiempo del esperado en ser resuelto o ser asignado al desarrollador correcto. Jeong, Kim, y Zimmermann [37] con el objetivo de descubrir redes de desarrolladores, estructuras de equipo y ayudar a asignar mejor a los desarrolladores los informes de fallos, realizaron implementaciones que generan un valor agregado en la asignación efectiva del informe al desarrollador en grandes equipos, ya que agiliza la atención de incidencias pues la asignación se hace al desarrollador adecuado. Para el caso de estudio que se pretende abordar sería de gran ayuda ya que los equipos, aunque no son grandes, esto representa una buena práctica, ya que ayudaría a disminuir los tiempos de atención perdidos por malas asignaciones; contar con la matriz de expertos y de

escalamiento en cada proyecto, adicionalmente es importante tener el conocimiento de las diferentes técnicas que existen y pueden ayudar a mejorar la clasificación de fallos y la asignación de estos a los desarrolladores.

DETECCIÓN DE FALLOS

Dentro del ciclo de vida del bug, luego de lograr entender el fallo, reproducirlo y asignarlo, sigue la detección del origen de este [38], en el mantenimiento correctivo se realizan tareas que dan solución temporal a los fallos y existen las soluciones de raíz, estas soluciones son más demoradas ya que requieren un mayor análisis y detectar el origen real del problema [7], si el reporte del fallo contiene información que referencia módulos del sistema y temas técnicos, esto ayuda a que la detección se realice de manera más eficiente y rápida, pero generalmente los reportes son realizados por usuarios finales, que no cuentan con mayores conocimientos técnicos sobre el sistema [29], por esto es importante al momento de reportar fallos adjuntar imágenes o videos que incluyan pantalla donde se genera el fallo, para ayudar a que se logre una detección más rápida del mismo, cuando no se cuenta con esta información se pueden ampliar los tiempos de detección y es frecuente que se demore más el bug, ya que muchas veces detectar el fallo es más complejo que la solución a implementar.

Jana *et al.* [39] proponen un algoritmo que permite la detección de fallos de forma automática, el cual tiene en cuenta 3 problemas que los autores identifican: la exploración de ruta de fallo, la falta de un oráculo de fallo y la localización de fallos. Ellos abordaron estos tres problemas y diseñaron, implementaron y evaluaron EPEX (Explorador de ruta de fallo), un algoritmo novedoso que puede detectar fallos en código C secuencial. Para proponer este algoritmo realizaron un estudio de cómo los programadores comunican los fallos controlados. Esta propuesta tiene en cuenta que, si bien no existen restricciones sobre los tipos de datos/valores que pueden utilizarse para comunicar fallos, los programadores C, en general, crean un protocolo de fallo informal y específico del programa y lo siguen en todas las funciones falibles para comunicar fallos.

Esta propuesta está lejos de dar una solución satisfactoria al problema que representa la detección automática de fallos, dada su complejidad. Los autores se centran en garantizar las rutas de los fallos esperados en el sistema, pero generalmente los fallos que se presentan en el sistema no son controlados.

La propuesta de Kim *et al*. [40], es un algoritmo que se basa en el histórico de fallos que se corrigieron antes, esto da una mayor certeza a la búsqueda de fallos automáticos, ya que aprende del histórico y toma patrones propios de cada aplicación. Esta se basa en patrones genéricos y se adapta al comportamiento particular de cada aplicación, esto lo hace más competitiva e interesante frente a propuestas anteriores.

CORRECIÓN DE FALLOS

Cuando se realiza corrección de fallos, se puede inyectar un nuevo fallo al sistema, el cual no se detecta hasta que un nuevo informe de fallo se reporta. Asaduzzaman *et al.* [41] proponen una metodología para detectar fallos que se inyectan durante la corrección de otros, este es un punto importante a tener en cuenta en la presente investigación ya que esto se presenta con frecuencia en los entornos laborales. Este trabajo tiene un gran componente manual lo que lo hace poco eficiente pues requiere tiempo de análisis, adicional a la implementación del algoritmo Diff que les permite detectar las diferencias del código fuente antes del cambio y luego de que se fija la corrección. Este algoritmo podría ser de gran utilidad siempre y cuando se encuentre la manera de hacer la clasificación y tamizaje de forma automática. En este trabajo también se detectó que los días más propensos a cambios que inyectan fallos son los viernes, lo que podría ayudar a tener este factor en cuenta en el caso de estudio a desarrollar.

Según Echeverría *et al.* [42], la fijación de fallos en las líneas de producto de software industriales, es menos eficiente al realizar la corrección de fallos en la capa de configuración, en la capa de realización y en la propagación. Para llegar a estas conclusiones fue necesario realizar un caso de estudio, el cual se basó en dar respuesta a varias preguntas, pero no propone soluciones o mejores prácticas a la hora de solucionar fallos, solo se reconoce que

para la corrección de estos el nivel de dificultad está asociado al número de productos que se debe replicar o afectar a la hora de hacer un cambio o corrección.

A la hora de hacer la corrección de los fallos y con el fin de hacer seguimiento de éstos, se espera que los desarrolladores registren cómo, cuándo, dónde y quién corrige los fallos. Bajo esta premisa los autores Romo *et al*. [43] realizan un análisis entre los repositorios de controles de cambios y el bug tracking, con el fin de detectar diferencias entre ambas herramientas, respecto a un fallo corregido y de esta manera poder disminuir la brecha entre ambos artefactos. Esta investigación es importante para compañías donde se manejan registros y control de los cambios, del software, en bases de datos diferentes, ya que de esta forma se puede validar qué tan confiable es la información en cualquiera de estas bases de datos, y puede ayudar a validar respecto al problema que se encontró y revisar qué buenas prácticas implementar para evitar diferencias entre los repositorios de los fallos y versionamiento del código fuente ya que el ideal es mantener ambos actualizados y sincronizados.

PARTE III

CARACTERIZACIÓN

"Comienza a manifestarse la madurez cuando sentimos que nuestra preocupación es mayor por los demás que por nosotros mismos." -- Albert Einstein

CAPÍTULO 4 CARACTERIZACIÓN DE TÉCNICAS E INSTRUMENTOS PARA EL PROCESO DE GESTIÓN DE FALLOS

En este capítulo se desarrolla la especificación de un conjunto de mejores prácticas para el proceso general de mantenimiento correctivo de software. Para tal fin, se muestra cómo se ejecutó un proceso de selección, evaluación y priorización de las prácticas, que se tendrán en cuenta como taxonomía base en el momento de definir la aproximación metodológica.

4.1. IDENTIFICACIÓN DE PRÁCTICAS PARA EL REPORTE, ANÁLISIS Y FIJACIÓN DE FALLOS

4.1.1. REVISÓN Y SELECCIÓN DE LAS PRÁCTICAS

A continuación, se reportan las prácticas de mantenimiento de software, identificadas en la revisión de literatura, esta identificación y selección se realizó teniendo en cuenta la descripción que los autores hacían referente a si las practicas, si eran positivas o negativas frente al proceso de mantenimiento correctivo de software y cuales podrían generar mejores resultados frente a otras, algunas se nombraron de manera textual en sus investigaciones y otras fueron deducidas teniendo en cuenta el contexto y el problema abordado.

Luego se presenta su clasificación de acuerdo a las siguientes fases del ciclo de vida del mantenimiento de software: Fase 1 – Reporte, Fase 2 – Pre-análisis, Fase 3 – Análisis y corrección, y Fase 4 – Implementación. Adicionalmente, se definen algunas prácticas genéricas que son comunes a todas las fases.

- **Fase 1 – Reporte de la falla**

Según Chaparro [33] el reporte de fallos es la materia prima con la que el desarrollador analiza el caso reportado, pero existe una brecha entre lo que reporta el usuario final y lo que espera encontrar el desarrollador, por lo tanto, propone que un informe de fallo debe

describir los siguientes componentes: *Comportamiento observado, Comportamiento esperado* y*, Pasos para reproducir el error*. Si el informe contiene estos tres elementos se facilita encontrar las líneas de código fuente donde se genera el fallo, y se puede trabajar el incidente de manera más eficiente.

Jie y Xiaoyin [21], realizan un análisis de los reportes de fallos y proponen las siguientes prácticas con el fin de aumentar su calidad y permitir una mejor clasificación:

- Evitar errores en la descripción de los pasos para reproducir el error: según su investigación errores en esta parte son los más dañinos a la hora de un desarrollador tomar el fallo para atenderlo.
- Completar la información del reporte de fallo: ya que si la información no es suficiente la calidad en la atención del fallo disminuye.
- Agregar un resumen al informe de fallos.
- Crear títulos Estructurados (nombre del proyecto-módulo del proyecto, etc.), que sean más propensos a una automatización.

Davies y Roper [10] proponen algunas características que consideran importantes para que un reporte de fallo pueda trasmitir de forma correcta lo que está informando el usuario, el desarrollador lo pueda entender y se logre dar solución de manera más efectiva.

- Pasos para reproducir: Un conjunto claro de instrucciones que el desarrollador puede utilizar para reproducir el error en su propia máquina.
- Trazas de pila: Traza de pila producida por la aplicación, más a menudo cuando informa un fallo en la aplicación.
- Casos de prueba: uno o más casos de prueba que el desarrollador puede utilizar para determinar cuándo han solucionado el fallo.
- Comportamiento observado: lo que el usuario vio ocurrir en la aplicación como resultado del fallo.
- Capturas de pantalla: Una captura de pantalla de la aplicación mientras se está produciendo el fallo.
- Comportamiento esperado: Lo que el usuario espera que suceda, por lo general contrasta con el comportamiento observado.

- Ejemplos de código: Un ejemplo de algún código que puede causar el fallo.
- Resumen: Un resumen corto (generalmente una oración).
- Versión: cuál es la versión de la aplicación que el usuario estaba utilizando en el momento del fallo.
- Informes de fallo: un informe de fallo producido por la aplicación (para el caso de las aplicaciones que generan este tipo de informes).

- **Fase 2 – Pre Análisis**

En esta fase es importante realizar una clasificación del fallo, de esta forma se podrá realizar una estimación de qué tanto esfuerzo se requiere para solucionarlo. Como buenas prácticas, Thung [36] propone clasificar los fallos basado en la cantidad de líneas de código a intervenir.

El tiempo que transcurre desde que se genera un caso con un fallo de sistema hasta que éste es resuelto, puede ser alto, muchas veces debido al tiempo que el caso demora en ser asigado al desarrollador correcto, por esto Jeong [37] propone las siguientes prácticas, para mitigar esta situación:

- Contar con una base de datos de inventario del sistema y sus diferentes módulos.
- Contar con una base de datos de los ingenieros encargados de la solución de los fallos.
- Relacionar los diferentes módulos con los ingenieros expertos para que se pueda hacer una asignación de los fallos a los expertos de cada módulo.
- Reasignar un caso, simpre y cuando se busque solucionar de raiz [3].
- Reasignar y equilibrar la carga de los que solucionan los reportes.

Según Tessone [35], se deben tener las siguientes prácticas para ayudar a una posible clasificación de los fallos de forma automatizada:

- Definir e identificar a los usuarios que reportan los fallos tanto como los que lo solucionan
- Calificar la calidad del reporte tanto usuario como Analista: para que esto pueda ir dando una evaluación a quien reporta el fallo y a quien lo soluciona, ya que

muchas veces los analistas les dan prioridad a casos de reporteros más confiables.

- Reasignar a quien lo reporta, si el reporte tiene mala calidad (falta información).
- Definir tiempos máximos de espera para el cambio de este estado.

- **Fase 3 – Análisis y corrección**

En esta fase se proponen prácticas tanto en el momento de analizar la falla del software como en el momento de implementar cambios que la corrijan, por tanto, también se deben realizar pruebas. Según Ruiz y Polo [29] estas prácticas son:

- Realizar diferentes pruebas en busca de fallos para minimizar el riesgo de afectar el software de forma negativa.
- Realizar análisis estáticos e inspecciones para elevar los niveles de eficiencia de eliminación de defectos por encima del 95 por ciento.

En la fase de corrección es importante tener en cuenta que al modificar el código se pueden agregar fallas en la seguridad del sistema que se está modificando, y éste quedaría vulnerable a ataques o errores de memoria, aun cuando se esté solucionando el fallo reportado, por tanto Jana y Kang [39] proponen:

- Usar herramientas para la detección de errores que afectan la seguridad.

Dado que muchas veces toma más tiempo la detección y ubicación del fallo que el tiempo que toma la corrección del mismo [11], se sugiere como buena práctica:

- Realizar un pre-análisis del fallo al hacer estimaciones para determinar si la detección del mismo es más compleja que la posible solución.

En el momento que se realizan las implementaciones de las correcciones es posible caer en errores, los cuales pueden pasar desapercibidos, por esto Hovemeyer y Pugh [9] hablan de diversas fuentes existentes hoy en día de patrones de errores, como la siguiente, la cual se incluye como buena práctica.

- Contar con una base de datos de patrones de errores, debidamente caracterizados y documentados para que la persona que desarrolla la solución y

quien la prueba, pueda validar en ésta de acuerdo a lo que están trabajando y detectar los patrones de errores que podrían estar incidiendo.

De acuerdo al estudio realizado por Böhme [44], las siguientes estrategias para el diagnóstico de los fallos deben ser tenidas en cuenta al momento de realizarlos:

- Hacer razonamiento hacia adelante. Los programadores siguen cada paso computacional, basado en el reporte del fallo.
- Hacer razonamiento hacia atrás. Los programadores parten de lo inesperado.
- Analizar y comprender el código. Los programadores leen el código para entender.
- Realizar pruebas de manipulación de entrada. Los programadores construyen un caso de prueba similar.
- Realizar análisis fuera de línea. Los programadores analizan una traza de error o un volcado de núcleo (por ejemplo, a través de *valgrind, strace*).
- Seguir la intuición. El desarrollador utiliza su experiencia de un parche anterior.

- **Fase 5 – Implementación**

Los cambios son inevitables en el desarrollo de software y cada cambio realizado en el código fuente de un sistema o software representa el riesgo de que pueda introducir un fallo cuando se corrige uno, por tanto, se proponen buenas prácticas que ayudan a detectar cuando éste se presenta y algunas para mitigar este tipo de incidentes. Basado en el caso de estudio realizado en software libre Android por Asaduzzaman [41], se proponen las siguientes prácticas:

- Documentar el código con una estructura estándar, la cual incluya el ID del reporte del fallo y las palabras claves asociadas al reporte.
- Restringir los cambios efectuados los viernes y sábados, ya que son los más propensos a inyectar fallos.
- Actualizar los estados de los reportes de fallos ya que estos permiten hacer un filtrado más eficiente a la hora de revisar incidentes reincidentes o fallos ocasionados por un mantenimiento anterior.

Estas prácticas también son mencionadas por Pan y Whitehead [40], ya que son la base fundamental para hacer memorias de los fallos y permiten que en un futuro se puedan implementar herramientas de automatización.

Una vez que las aplicaciones de software están instaladas y en uso, tres cosas pueden suceder: se encuentran errores que deben ser corregidos, nuevas características son agregadas en respuesta a las necesidades de negocios y cambios en las leyes y regulaciones; los proveedores de software pueden sacar nuevas versiones de paquetes de software o agregar nuevas características por una tarifa. Esta parte de la ingeniería de software no está bien cubierta por la literatura [29].

Cuando se hacen correcciones a productos de software industrial, el nivel de complejidad se aumenta, ya que generalmente éstos están divididos en muchos módulos y un cambio en un módulo puede afectar a uno o más módulos, aun cuando estos parezcan no tener relación [42], por esto se sugiere como buena práctica:

- Realizar un análisis del impacto a causar en todos los módulos existentes, antes de hacer implementación del mantenimiento correctivo, en caso de no tener documentación de todos los módulos, hacer el cambio de manera controlada, y dejar este cambio en seguimiento al menos 48 horas para garantizar que la corrección agregada no genere nuevos fallos.
- Realizar los cambios expertos que conozcan no solo un módulo del sistema a afectar, si no que los conozca todos para evitar afectaciones colaterales.
- Generar un listado de los diferentes módulos donde sea necesario implementar cambios.

A la hora de hacer mantenimiento correctivo se cuenta con dos fuentes de información que generalmente se tienen de forma independiente, una de ellas es el sistema que se utiliza para reportar el fallo y otra es el sistema donde se lleva el control de cambios de la solución y el mismo código fuente. Romo y otros [43] proponen como buenas prácticas:

- Mantener sincronizados los dos repositorios, el de reporte del fallo y el de control de cambios del producto y código fuente.

- Estandarizar la forma de documentar el código fuente a la hora de hacer cambios correctivos, en esta documentación debe estar relacionado el ID del caso que reportó la falla.
- Tener procesos que se encarguen de hacer la sincronización de forma automática, así se cierra la brecha que normalmente se tiene entre estas dos herramientas.

- **Genéricas.**

En esta categoría se incluyen prácticas que se deben tener en cuenta en todas las fases anteriormente descritas.

Ruiz y Polo [6] proponen como prácticas para mejorar el proceso de mantenimiento de software la definición de una ontología para el mantenimiento de software, donde se especifican aspectos tanto estáticos como dinámicos y propone tener documentados todos los artefactos del producto por cada versión que se tenga del mismo, esto con el fin de dar gestión del conocimiento al mismo.

En el marco de esta propuesta resulta interesante resaltar las siguientes prácticas:

- Descomponer las actividades realizadas en las diferentes fases del mantenimiento correctivo.
- Definir restricciones temporales entre las actividades (por ejemplo, el orden en el que deben realizarse las actividades).
- Controlar las actividades y la ejecución de los procesos durante el proceso de mantenimiento.

Según Jones [29], el análisis y propuesta de buenas prácticas para el proceso de mantenimiento de software es complejo dado que el mantenimiento y el trabajo de mejora requiere una evaluación no solo de los cambios en sí mismos, sino también un análisis detallado y completo de la estructura y el código de la aplicación heredada que se está modificando. Por tanto, este autor propone una serie de buenas prácticas, de las cuales se toman las siguientes:

- Asignar especialistas en mantenimiento en lugar de desarrolladores.
- Implementar procedimientos formales de gestión de cambios.

- Implementar herramientas formales de gestión de cambios.
- Usar bibliotecas de pruebas de regresión formales.
- Realizar estudios automatizados de análisis de complejidad de aplicaciones heredadas.
- Realizar búsqueda y eliminación de todos los módulos propensos a errores en aplicaciones heredadas.
- Identificar los códigos muertos en las aplicaciones heredadas y eliminarlos.
- Realizar refactorización a las aplicaciones antes de mejoras importantes.
- Utilizar el diseño formal y las inspecciones de código fuente en las principales actualizaciones.
- Realizar un seguimiento de todos los defectos informados por el cliente.
- Realizar seguimiento al tiempo de respuesta desde el envío hasta la reparación de defectos.
- Realizar seguimiento al tiempo de respuesta desde la presentación hasta la finalización de la petición del cambio.
- Hacer un seguimiento a la disponibilidad del software para los clientes.

Cuando se está en la fase de pruebas, luego que se ha desarrollado la solución, es muy importante tener buenos esquemas de pruebas que permitan detectar posibles fallos del sistema. Se considera la propuesta de Hughes y Ab [45], quienes proponen una solución automática para la detección de fallos que busca implementar buenas prácticas que ayuden a tener estándares que faciliten estas implementaciones en un futuro. Se propone como buenas prácticas:

- Marcar los fallos detectados ya sea de forma manual o automática una vez sean encontrados, para que al volver a hacer revisiones no se vuelva a revisar la misma falla.
- Crear vínculos entre los reportes de fallos y los repositorios de control de cambios de los diferentes sistemas de información.

Lo anterior es de suma importancia debido a que gracias a esto se podría saber si un cambio se realiza debido a un reporte o si hay varios cambios que se realizan. Debido a esto Nguyen

[4], proponen una solución automática que rescata la relación entre los cambios y los reportes de fallos, para lo cual se deben cumplir las siguientes prácticas:

- Implementar y mantener actualizado un repositorio para el reporte de fallos y control de cambios.
- Intervenir en el repositorio de reporte de fallos, el que genera el reporte y el desarrollador.
- Actualizar de manera paralela el repositorio de reporte como el de control de cambios del sistema que está afectando.
- Tener en cuenta que el que reporta está describiendo un fallo y el desarrollador que soluciona está describiendo de una forma técnica, por esto es posible que el texto no sea parecido.
- Evitar hacer relaciones entre la descripción del fallo y la descripción de la solución, ya que son perspectivas diferentes la del usuario y el desarrollador.

Actualmente son pocas las investigaciones que abarcan todo el proceso de mantenimiento de software, debido a esto la selección de las prácticas y los componentes (que al usarlos generan buenas prácticas), se realiza en la revisión de diferentes autores y con diferentes enfoques, logrando el objetivo de cubrir todas las fases del ciclo de vida del mantenimiento desde el reporte del fallo hasta la solución e implementación. En el siguiente aparte de este capítulo se presentan tabuladas las prácticas identificadas.

4.1.2. ORGANIZACIÓN DE LAS PRÁCTICAS

Luego de haber realizado la selección de las prácticas e indicar su origen, según se indica en el apartado anterior, en la **Tabla 1** presenta su tabulación y organización.

***Tabla 1**. Listado de prácticas seleccionadas*

		Prácticas
Reporte de la falla	**1**	Describir los componentes: CO, CE, y PPRE.
	2	Verificar la descripción de los PPRE
	3	Adjuntar imágenes en el reporte
	4	Agregar un resumen al informe de fallos.

		Prácticas
	5	Generar títulos estructurados
	6	Definir e identificar los usuarios.
	7	Calificar la calidad del reporte (desarrollador)
	8	Calificar la calidad de la solución (usuario)
Pre Análisis	**9**	Clasificar fallos de acuerdo a las líneas a modificar
	10	Contar con inventario del sistema y sus módulos.
	11	Contar con una base de datos de los ingenieros encargados de la solución de los fallos.
	12	Relacionar módulos con sus expertos.
	13	Reasignar un caso para dar solución de raíz al fallo.
	14	Reasignar y equilibrar la carga a quienes solucionan.
	15	Reasignar a quien lo reportó, si éste tiene mala calidad
	16	Definir tiempos máximos de espera, al devolverlo.
Análisis y corrección	**17**	Realizar diferentes tipos de pruebas.
	18	Realizar análisis estáticos e inspecciones
	19	Reconocer que modificar código puede agregar falla.
	20	Usar herramientas para la detección de fallos de seguridad
	21	Realizar un pre-análisis del fallo al hacer estimaciones para determinar si la detección del mismo es más compleja que la posible solución
	22	Contar con una BD de patrones de errores.
	23	Usar estrategias para un análisis efectivo.
implementación solución	**24**	Documentar el código con una estructura estándar.
	25	Restringir los cambios efectuados los viernes y sábados, ya que son los más propensos a inyectar fallos.
	26	Actualizar los estados de los reportes de fallos.
	27	Hacer un análisis del impacto a causar.
	28	Hacer el cambio de manera controlada.
	29	Realizar los cambios expertos que conozcan muchos módulos del sistema.
	30	Listar los módulos donde sea necesario implementar cambios.
Genéricas	**31**	Descomponer las diferentes actividades, del proceso de mantenimiento.
	32	Definir restricciones temporales entre las actividades.
	33	Controlar las actividades y la ejecución de las tareas.
	34	Usar especialistas en mantenimiento en lugar de desarrolladores.
	35	Usar procedimientos formales de gestión de cambios.
	36	Usar herramientas formales de gestión de cambios.
	37	Usar bibliotecas de pruebas de regresión formales.
	38	Realizar estudios automatizados de análisis de complejidad de aplicaciones heredadas.
	39	Buscar y eliminar todos los módulos propensos a errores en aplicaciones heredadas.
	40	Identificar todos los códigos muertos en las aplicaciones heredadas.
	41	Renovar o re factorizar aplicaciones antes de mejoras importantes.
	42	Utilizar el diseño formal y las inspecciones de código en las principales actualizaciones.
	43	Realizar un seguimiento de todos los defectos informados por el cliente.
	44	Seguir el tiempo de respuesta desde el envío hasta la reparación de defectos.
	45	Seguir el tiempo de respuesta desde la presentación hasta la finalización de la petición del cambio.
	46	Hacer seguimiento de la disponibilidad del software para los clientes.
	47	Mantener sincronizados los repositorios asociados al proceso.
	48	Estandarizar la forma de documentar el código fuente.
	49	Tener procesos que se encarguen de hacer la sincronización de forma automática.
	50	Marcar los fallos una vez sean encontrados.
	51	Contar con un repositorio para el reporte de fallos.
	52	Contar con un repositorio para el control de cambios

		Prácticas
	53	Intervenir en el repositorio tanto el que genera el reporte como el desarrollador.
	54	Actualizar todos los repositorios el desarrollador.
	55	Tener en cuenta la diferencia en la descripción dependiendo el rol de los participantes del proceso.
	56	Evitar hacer relaciones entre la descripción del fallo y la descripción de la solución, ya que son perspectivas diferentes la del usuario y el desarrollador.

4.2. PRIORIZACION Y EVALUACION DE LAS PRÁCTICAS SELECCIONADAS

4.2.1. DEFINICIÓN Y SELECCIÓN DE LOS CRITERIOS DE EVALUACION

Los criterios de evaluación están íntimamente relacionados con el concepto de mejores prácticas [46], los criterios son las características que se espera tenga una buena práctica. Es importante tener en cuenta que el contexto en el que se desarrolla una buena práctica se define por el conjunto de factores que limitan o dirigen la actividad de los distintos actores que comparten ese contexto. De ahí que sea difícil valorar una buena práctica sin definir el contexto en el que se produce [47]. En el caso particular de este trabajo, el contexto es el proceso de mantenimiento de software, por lo que finalmente se definen criterios que debe tener una buena práctica y asociadas al contexto donde se busca implementarlas. Los criterios que se proponen a continuación fueron obtenidos de Sarco [48] y la Comunidad de Prácticas en APS Nodo Chile [49] y son los siguientes:

1. Reproducible: Ayuda a que el reporte se pueda reproducir en todo momento.
2. Claro: el reporte no es ambiguo se puede entender cuál es el fallo que se está reportando.
3. Identificable: El reporte cuenta con un ID único para poder hacer seguimiento.
4. Comunicable: permite la comunicación entre el reportero y el desarrollador ya que se cuenta con datos de ambos para que puedan interactuar si así lo requiere el reporte del fallo.
5. Reincidencias detectables: permite detectar reincidencias en los fallos [48].
6. Resultado Valioso: representa o agrega un resultado valioso para el usuario

7. Sencillez: Es sencilla y simple.
8. Consecuente con la situación: emerge como respuesta a una situación que es necesario modificar o mejorar.
9. Pertinente: es pertinente y adecuada al contexto local en donde se implementa.
10. Sostenible: es sostenible en el tiempo, puede mantenerse y producir efectos duraderos.
11. Replicable: fomenta la replicación de la experiencia en una situación distinta, pero con condiciones similares.
12. Innovadora: la innovación no sólo implica una nueva acción, sino que puede ser un modo diferente y creativo de realizar prácticas tradicionales o de reorganizarlas.
13. Evaluable: considera elementos de evaluación de resultados, retroalimentación de las acciones y reorganización de ellas a partir de lo aprendido [49].

Este listado se seleccionó teniendo en cuenta el objetivo de esta investigación, en los cuales se expresan características que hacen parte de una buena práctica y de un buen reporte de fallos.

4.2.2. APLICACIÓN DE UN MÉTODO PARA LA PRIORIZACIÓN Y SELECCIÓN

El proceso de evaluación y priorización se realizó apoyado en el método Delphi, el cual está fundamentado en el principio de inteligencia colectiva y busca lograr un consenso de opiniones expresadas de forma individual por un grupo de expertos en un tema a evaluar [50]. Este método se desarrolló en 3 fases según lo propone Delphi las fases son: Fase preliminar: Consiste en la definición del contexto, objetivos, elementos a evaluar y la selección de los expertos; Fase exploratoria: Consiste en la elaboración y aplicación de cuestionarios según sucesivas vueltas, de forma que con las respuestas más comunes de la primera se formula la siguiente vuelta. Fase final: consiste en el análisis estadístico y presentación de la información.

- **Fase preliminar:**

  Para el caso de esta investigación luego de tener los criterios para evaluar, se procede a realizar la elección de los expertos a participar en el proceso de evaluación.

Se seleccionaron 8 personas con amplios conocimientos en el tema de investigación y algunos con experiencia en el ámbito laboral específicamente en las áreas de mantenimiento de software.

El panel se llevó a cabo en la sede de la Universidad de Medellín y algunos estuvieron vía Teleconferencia, los asistentes fueron citados con anterioridad y se les compartió un documento de contextualización del proyecto y un listado de las prácticas a evaluar, con el objetivo que llegaran a la sesión con más claridad del proceso a realizar. Cada práctica se cruzó con los criterios de valuación creando una matriz como se muestra en la **Figura 5**. Para su evaluación, el experto debía indicar un rango de 1 a 5, donde 1 es *no aplica* y 5 es *aplica*.

			1	2	3	4	5	6	7	8	9	10	11	12	13
Etapas o fases		**Prácticas**	**Reproducible**	**Claro**	**Identificable**	**Comunicable**	**Reincidencias**	**Resultado valioso**	**Sencillez**	**Consecuente**	**Pertinente**	**Sostenible**	**Replicable**	**Innovadora**	**Evaluable**
Reporte de la falla	**1**	Describir los componentes: CO, CE, y PPRE.													
	2	Verificar la descripción de los PPRE													
	3	Adjuntar imágenes en el reporte													
	4	Agregar un resumen al informe de fallos.													
	5	Generar títulos Estructurados													
	6	Definir e identificar los usuarios.													
	7	Calificar la calidad del reporte (desarrollador)													
	8	Calificar la calidad de la solución (usuario)													

Figura 5. Muestra del formulario Evaluación

- **Fase exploratoria**

  donde los expertos hacen una evaluación inicial con un *feedback* respecto a las prácticas propuestas y criterios, y se proponen cambios a lo planteado por el moderador.

Los expertos realizaron una evaluación inicial para todas las prácticas. Luego, se obtuvieron comentarios y observaciones de todos los expertos, de los cuales se resaltan las que listamos a continuación:

- Los criterios del 1 al 5, deben ser aplicados solo en la fase 1 del proceso de mantenimiento.
- Algunas prácticas se pueden unificar con otras como por ejemplo (la 17 y 18), en razón a que son similares y apuntan hacia un mismo objetivo.

Con estas observaciones se realizaron los cambios sugeridos, quedando los criterios del 1 al 5 válidos sólo para las prácticas que tienen que ver con el reporte de fallos y al unificar prácticas similares, se redujeron a 47 prácticas, de 56 que se tenían planteadas inicialmente.

En un segundo momento de esta fase, los expertos vuelven a encontrar prácticas que se pueden unificar o eliminar porque son redundantes o similares, esto se determina verificando cual es el objetivo de las prácticas y que busca mejorar cada una, quien la debe implementar y en qué fase, se procede a ejecutar estos cambios sugeridos y las prácticas que finalmente se evalúan son 44.

- **Fase 3**

  Se realizó una segunda sesión donde se analizaron las 44 prácticas resultantes ver Tabla 4 de la anterior sesión, y los expertos hacen las evaluaciones con la escala definida. De esta manera se obtienen las evaluaciones que serán analizadas en el siguiente aparte de este documento.

4.2.3. RESULTADOS

A continuación, mostraremos los resultados obtenidos luego de realizar las sesiones de evaluación a las practicas que harán parte del a propuesta, los datos obtenidos se tabularon y se promediaron los valores en una sola matriz, donde se obtuvo el promedió del resultado general de cada practica ver **Figura 5**, se calcula un promedio de cada practica según cada criterio evaluado ver **Figura 6**, donde se muestra un ejemplo del resultado obtenido de una

práctica en la evaluación de cada criterio. El objetivo de promediar estos datos era saber de 0 a 5 que tan aceptada era una práctica o no, según los expertos, por esto se calcularon valores promedio, en la siguiente sección se realizara un análisis de estos resultados.

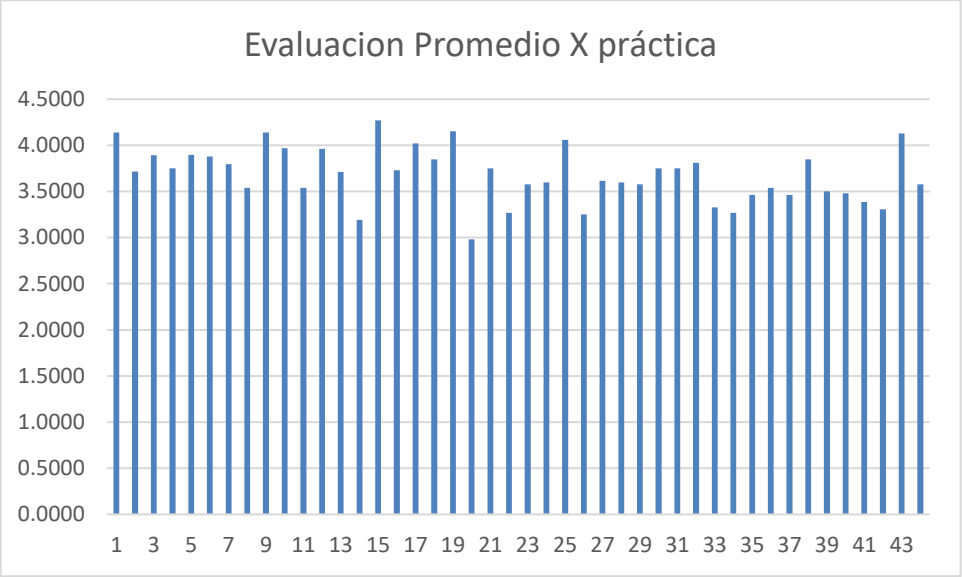


Figura 6. Resultados Evaluación Prácticas

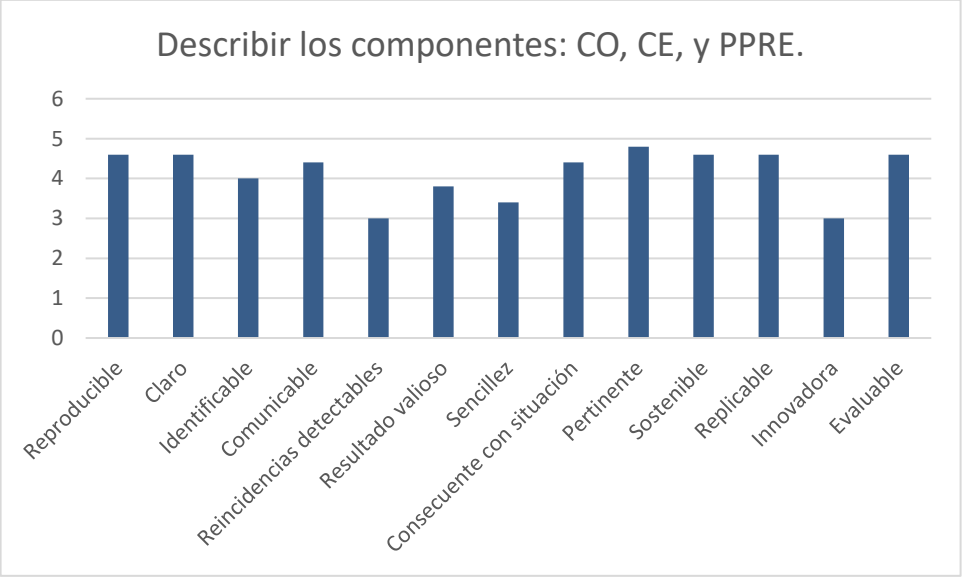


Figura 7. Muestra resultado evaluación de una práctica puntual

4.2.4. ANÁLISIS DE RESULTADOS

Teniendo los resultados de las evaluaciones realizadas por los expertos, se procede con su análisis, encontrando que en general las prácticas tienen una buena aceptación, así: 13 tienen una evaluación promedio (valoración 3) lo que equivale a una aceptación del 60%, mientras 31 prácticas tienen una evaluación promedio (valoración 4) asociándose con una aceptación del 80%. Esto indica que las prácticas seleccionadas son pertinentes para el objetivo de la investigación y están alineadas al mejoramiento del proceso de mantenimiento de software.

Al realizar un análisis detallado de las evaluaciones se encuentra que las prácticas que menor y mayor aceptación obtuvieron son:

- La práctica con menor aceptación es "Evitar implementaciones los días viernes y sábados", esta obtuvo una calificación promedio de 2,98 siendo un valor muy cercano a 3, por esto no fue descalificada. El criterio que más bajó esta evaluación fue la *innovación* con promedio en 2, pero tiene en contra los criterios: *consecuentes con la situación*, *Pertinente* y *sostenible*, valorados en 4. Esto indica que, aunque no es innovadora es válida para ayudar a mejorar el proceso.
- En el extremo opuesto la práctica mejor valorada es "Documentar el código con una estructura estándar.", con una evaluación promedio de 4,1. El criterio que más la castiga es *Evaluable*, por lo que se puede deducir entonces que la práctica es pertinente, pero se debe tener en cuenta de qué manera se podría evaluar en la mejora el proceso.

En la **Figura 8** se presenta una muestra de las evaluaciones realizadas por los exportes y que fueron posteriormente analizadas. Para detallar todas las evaluaciones realizadas, ver el anexo 1.

			1	2	3	4	5	6	7	8	9	10	11	12	13
Etapas o fases		Prácticas	Reproducible	Claro	Identificable	Comunicable	Reincidencias detectables	Resultado valioso	Sencillez	Consecuentes sitúacción	Pertinente	Sostenible	Replicable	Innovadora	Evaluable
Reporte de la falla	1	Describir los componentes: CO, CE, y PPRE.	4	4	5	5	4	4	4	4	4	4	5	4	4
	2	Verificar la descripción de los PPRE	5	4	4	4	3	4	5	5	3	3	5	1	4
	3	Adjuntar imágenes en el reporte	4	5	5	4	4	4	5	5	5	5	5	2	4
	4	Agregar un resumen al informe de fallos.	5	4	4	4	4	5	4	4	4	4	4	2	5
	5	Generar títulos Estructurados	3	3	3	3	3	4	4	2	2	4	4	1	3
	6	Definir e identificar los usuarios.	5	4	4	5	4	4	4	4	4	4	4	2	4
	7	Calificar la calidad del reporte (desarrollador)	4	4	4	4	4	4	4	4	4	4	4	2	4
	8	Calificar la calidad de la solución (usuario)	5	5	5	5	4	5	4	4	4	4	5	3	4
Pre Análisis	9	Clasificar Fallos de acuerdo a las líneas a modificar	5	5	5	5	5	5	4	5	4	5	4	2	4
	10	Contar con inventario del sistema y sus módulos.	4	4	4	4	4	4	4	4	4	4	4	3	4
	11	Contar con una base de datos de los ingenieros encargados de la solución de los fallos.	3	3	3	3	4	4	3	4	4	4	4	2	3
	12	Relacionar módulos con sus expertos.	4	4	4	4	4	5	4	4	4	4	4	4	4
	13	Reasignar un caso para dar solución de raíz al fallo.	4	4	4	5	4	5	4	4	4	4	4	3	3
	14	Reasignar y equilibrar la carga a quienes	1	3	3	3	3	3	3	3	3	3	3	2	2
	15	Reasignar a quien lo reportó, si éste tiene mala calic	4	4	4	4	3	4	3	4	4	5	4	2	4

Figura 8. Muestra de la evaluación por expertos

PARTE IV

PROPUESTA

"No basta saber, se debe también aplicar. No es suficiente querer, se debe también hacer." -- Goethe

CAPÍTULO 5 APROXIMACIÓN METODOLÓGICA

Las metodologías de desarrollo de software contienen un conjunto de pasos, técnicas, herramientas y artefactos que permiten a los equipos de desarrollo generar productos de software de calidad [15]. Esta investigación se enfoca en el mantenimiento de software como una fase del ciclo de vida de este Jones [29] y Cataldi et al [15] se refieren a esta fase como un mini ciclo de vida del software, pero este tiene sus propias particularidades, como por ejemplo todo los cambios al implementarse se hacen sobre software en ambiente productivo, por eso la relevancia en proponer un acercamiento metodológico dedicado solo a esta fase.

Para representar el acercamiento metodológico propuesto, se implementan diagramas y elementos del estándar para modelado de procesos de desarrollo de software llamado SPEM (*Software Process Engineering Metamodel*) [51], la selección de este modelo se realiza debido a que este permite representar diferentes procesos de desarrollo de software y sus componentes, proporcionando una sintaxis y estructura para cada aspecto del proceso, tales como roles, fases, actividades, productos de trabajo, guías, herramientas, etc. [52]. Esto se ajusta al objetivo que se tiene con la propuesta a desarrollar, en la **Figura 9**, se pueden observar los elementos usados en este acercamiento metodológico.

- **Rol del proceso**

Define las responsabilidades sobre productos de trabajo específicos y define los roles que ejecutan y asisten en actividades específicas, los roles identificados en esta propuesta se determinan teniendo en cuenta el proceso de mantenimiento correctivo y los diferentes actores que hacen parte de este, a continuación, se describen los roles:

- Usuario o Cliente: son las personas que interactúan con el sistema y detectan los fallos, estos tienen la responsabilidad de generar un reporte del fallo tan pronto es detectado, documentando el mismo para que pueda ser atendido y solucionado.
- Analista nivel 1: se encarga de recibir los reportes de fallos y realizar un pre-análisis; en caso de que sea necesario, debe contactar al usuario para obtener más información

acerca del reporte, tan pronto se tenga toda la información necesaria éste soluciona el caso o lo escala a un analista nivel 2, de acuerdo al nivel de complejidad del reporte.

- Analista nivel 2: Es el encargado de recibir los reportes, los soluciona y hace pruebas, debe solicitar y gestionar los pasos a producción con los cambios que se requieran para solucionar el fallo.

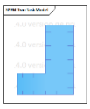
Elemento	Estereotipo
Rol	
Fase	
Actividad	
Definición de herramienta	
Producto de Trabajo	
Guía	

Figura 9. Estereotipos Gráficos De Los Elementos SPEM

- **Fases**

Las fases son el conjunto de actividades o momentos en los cuales se desarrollan actividades que tienen como objetivo un producto de trabajo. El acercamiento metodológico consta de cuatro fases: Reporte de fallos, Pre-análisis, Análisis e Implementación. En las fases se pueden manejar estados del reporte de fallo, los cuales no son restrictivos, lo cual quiere decir que se pueden ejecutar de forma secuencial o se puede navegar de una a otra fase, lo importante es poder lograr el objetivo dar solución al fallo.

- **Actividades**

  Las actividades incluyen las tareas, operaciones y acciones que son desempeñadas por un rol o las que este puede asistir. Una actividad se puede componer de elementos atómicos llamados tareas y éstos a su vez en pasos.

- **Definición de herramientas**

  Las herramientas son esos artefactos que ayudan a facilitar la gestión y el desarrollo del proceso, para el caso de estudio se propone el uso de herramientas para el registro de los fallos, herramientas para el versionamiento del software, bases de datos con inventarios de los sistemas de información.

- **Productos de trabajo**

  Es cualquier elemento generado, usado, o modificado por un proceso. Esto puede ser una pieza de información, un documento, un modelo, código fuente y demás. En cada fase se obtiene un producto de trabajo, por ejemplo, en la de reporte de fallo el producto es un reporte de fallo, debidamente documentado y claro.

- **Guía**

  La guía puede ser asociada con los elementos de SPEM, para proporcionar información más detallada a los actores acerca de los elementos asociados, es decir indica instrucciones o recomendaciones a seguir, en nuestra propuesta contamos con un conjunto de buenas prácticas a implementar.

5.1. REPRESENTACIÓN GRÁFICA DE LA PROPUESTA

En la **Figura 10** se muestra gráficamente la propuesta en notación SPEM; se representan las diferentes fases, roles y actividades y el flujo que se da entre ellos. En los siguientes apartes se describen detalladamente los diferentes componentes y fases de la propuesta.

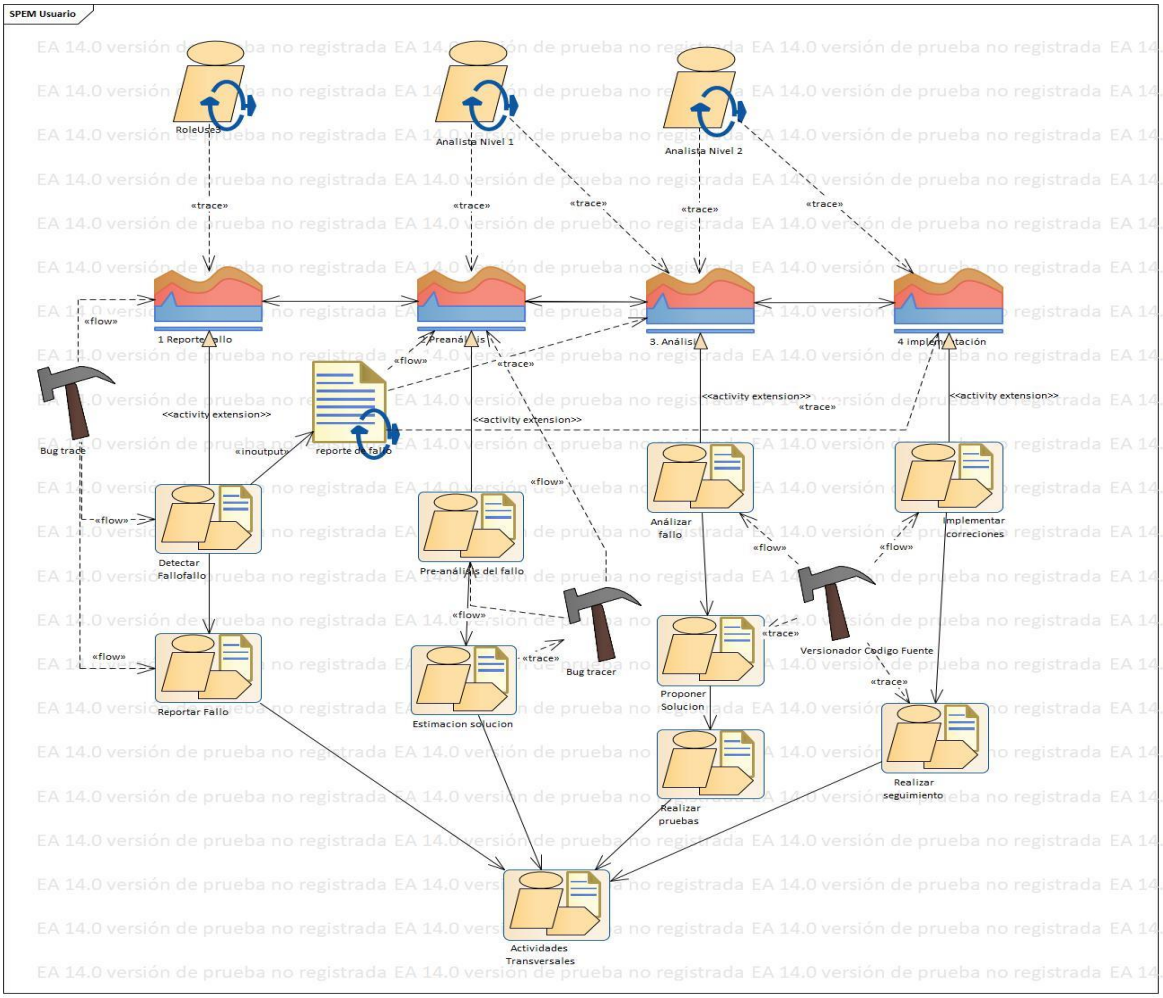


Figura 10. Aproximación metodológica

5.2. FASES Y COMPONENTES

Las fases en que divide esta propuesta se basan en el ciclo de vida de un bug propuesto por M. D. y otros [38] estas son: reporte de fallos; el cual inicia cuando se detecta y reporta un fallo, luego sigue la pase de pre análisis; en esta un analista verifica el reporte y hace una estimación inicial del tiempo de solución y en el caso que se requiera lo reasignara a otro analista, luego sigue la fase de análisis; en esta se hace la detección del origen del fallo, su corrección y las pruebas para garantizar que este quede corregido y no afecte otras

funcionalidades del sistema en general, luego sigue la implementación; donde se hace las correcciones en el ambiente productivo, es allí donde se corrige y solucionado el fallo.

Adicionalmente esta propuesta tiene en cuenta algunas metas claves de procesos definidos del modelo CMMI DEV en su área de proceso de soporte, tales como Definir, integrar y ejecutar consistentemente las actividades de ingeniería de software para producir el software, mantener la consistencia entre los productos de trabajo de software, planear las actividades de prevención de defectos, buscar e identificar las causas comunes de los defectos, Priorizar y eliminar las causas comunes de los defectos.

CMMI DEV se subdivide en cinco áreas: Análisis causal y resolución (CAR), gestión de configuración (CM), análisis de decisiones y resolución (DAR), medición y análisis (Ma) y aseguramiento de la calidad del proceso y del producto (PPQA) [22], muy alineados a estos procesos se encuentran los que propone la ISO/IEC 12207 para el soporte de software el cual se subdivide en ocho subprocesos: gestión de la documentación del Software, gestión de la configuración del software, aseguramiento de la calidad del software, verificación del software, validación del software, revisión del Software, auditoría de software, problemas de software [14], en el detalle de la propuesta las buenas prácticas están alineadas a estos procesos, debido que el objetivo de la propuesta es ayudar al proceso a lograr una madurez, en la cual se puedan tener mejores métricas y mejor control de las diferentes actividades que se requieren para que el proceso se pueda llevar a cabo correctamente.

5.2.1. FASE 1 REPORTE DE FALLOS

Esta fase inicia cuando un usuario encuentra fallos mientras usa el sistema en un ambiente productivo. A continuacion se listan prácticas seleccionadas en el numeral anterior y que buscan mejorar esta fase del proceso, adicionalmente se listan algunos artefactos que se deben tener en esta fase.

- **Herramienta**:

Aplicación para la creación y gestión del Reporte de fallo, se recomienda el uso de una herramienta para crear registro de los fallos en la cual se pueda documentar según las practicas propuestas.

Actividad 1: Detectar fallo

Descripción de la actividad

El usuario detecta el fallo sea porque este se generó al interactuar con el sistema o porque está haciendo pruebas del funcionamiento del sistema, en el momento que detecta la falla debe verificar que realmente sea una falla lo que se está presentando, es decir que lo pueda reproducir.

Entradas:

- Navegación en el sistema
- Datos para realizar transacciones o consultas en el sistema.

Salidas:

- Fallo en el sistema.

Tareas:

- Realizar pruebas con diferentes datos.
- Verificar desde diferentes terminales.

Actividad 2: Generar reporte del fallo

Descripción de la actividad

En esta actividad el usuario debe registrar el fallo en la herramienta que sea seleccionada para desarrollar dicha gestión, la correcta documentación del reporte por parte del usuario ayudará a que la solución sea más efectiva.

Es importante que los usuarios tengan claro el funcionamiento de la solución, de no ser así, podrían llegar reportes que realmente no son fallas sino mal entendimiento del sistema.

Entradas:

- Descripción del fallo

Salidas:

- Reporte de fallo registrado y debidamente documentado.

Tareas:

- Registrar fallo
- Hacer seguimiento al fallo

Buenas Prácticas:

Tabla 2 *Prácticas fase 1 reporte de fallos*

1	Describir los componentes: CO, CE, y PPRE.
2	Verificar la descripción de los PPRE
3	Adjuntar imágenes en el reporte
4	Agregar un resumen al informe de fallos.
5	Generar títulos Estructurados
6	Definir e identificar los usuarios.
7	Calificar el reporte (desarrollador y Usuario); esta práctica ayudara a conocer tanto la calidad del reporte como la calidad de la solución del mismo.

5.2.2. FASE 2 PREANÁLISIS

Luego de que el fallo es reportado este llega al área de soporte, por lo tanto, se recomienda tener un analista disponible para recibir todos los reportes de fallos.

Herramienta:

- Aplicación para la creación y gestión del Reporte de fallo.
- Base de datos o Inventarios de las soluciones y sus diferentes módulos.

Actividad 1: Ejecución del Pre-análisis

El analista debe hacer un pre-análisis del reporte, determinando si está debidamente diligenciado, si realmente es un fallo o no, si se puede reproducir y siempre se genera con las mismas variables o si el fallo se presenta con diferentes variables.

Entradas:

- Reporte de fallo.
- Bases de datos con historiales de los reportes.

Salidas:

- Reporte de fallo actualizado.

Tareas:

- Realizar pre-análisis del caso.
- Detectar reincidencias.
- Agregar comentarios y documentación al reporte.
- Reasignar el caso.

Actividad 2: Estimación de la posible solución

Luego de tener claro el comportamiento del fallo, el analista debe hacer una estimación de la posible solución y una reasignación en el caso que sea necesario, todo esto con el objetivo de tener un panorama general del fallo e identificar si es posible iniciar el proceso de análisis qué dará solución a este. En el momento que el analista decide analizar el caso o reasignarlo finaliza esta fase.

Entradas:

- Reporte de fallo.

Salidas:

- Reporte de fallo actualizado.
- Reporte de fallo reasignado.

Tareas:

- Estimar el tiempo de solución del fallo
- Actualizar estados del reporte.
- Agregar comentarios y documentación al reporte.
- Reasignar el caso.

Buenas Prácticas:

Tabla 3 *Prácticas fase 2 pre análisis*

8	Clasificar Fallos de acuerdo a las líneas a modificar
9	Contar con inventario del sistema y sus módulos.
10	Contar con una base de datos de los ingenieros encargados de la solución de los fallos.
11	Reasignar un caso para dar solución de raíz al fallo.
12	Reasignar un caso para dar solución de raíz al fallo o equilibrar la carga de los analistas.
13	Definir tiempos máximos de espera, al devolverlo.

Es importante que el analista y el usuario tengan una buena comunicación, de esta manera los reportes tendrán mayor dinámica y podrán ser resueltos efectivamente, eliminando tiempos muertos a la espera de más datos o tratando de entender el reporte. En el momento del analista hacer una estimación, es importante que se asesore de los analistas expertos en el módulo que presenta la falla, de esta manera la estimación será más cercana a la realidad.

5.2.3. FASE 3 ANÁLISIS

Esta fase inicia cuando el reporte es asignado al analista que lo estudiará y solucionará.

- **Herramienta**:
  - Aplicación para la creación y gestión del reporte de fallo.
  - Base de datos o Inventarios de las soluciones y sus diferentes módulos
  - Base de datos con patrones de fallos.
  - Herramienta de control de cambios en el código fuente.
- **Actividad 1: Revisión y análisis fallo**

El analista procede a realizar la revisión del caso que permita encontrar la falla y los factores que la generan. Para esto debe validar que la falla se presente siempre o si es intermitente, si le pasa a un solo usuario o a todos.

Entradas:

- Reporte de fallo.
- Rama del versionamiento de la solución.

Salidas:

- Reporte de fallo actualizado.
- Propuesta de solución del fallo.
- Rama del código actualizada con la mejora.

- **Actividad 2: Proponer solución**

luego de encontrar la falla el analista debe proponer la solución, esta implicara realizar cambios, que pueden ser desde registros en la base de datos o código fuente de las soluciones, cambios en Procedimientos almacenado en la base de datos, etc.

Entradas:

- Reporte de fallo.
- Rama del versionamiento de la solución.

Salidas:

- Reporte de fallo actualizado
- Documento para la realización de pruebas
- Rama del código actualizada con la mejora.

Actividad 3: Realizar pruebas

Se deben realizar diferentes pruebas, con el objetivo de garantizar que los cambios si dan solución al fallo y que al agregar las correcciones no se generen fallos en otras partes del sistema.

Entradas:

- Reporte de fallo.
- Propuesta de solución.
- Rama del versionamiento de la solución.

Salidas:

- Reporte de fallo actualizado.
- Documento certificación pruebas.

Buenas Prácticas:

Tabla 4 *Prácticas fase 3 Análisis*

14	Realizar diferentes tipos de pruebas (estáticos e inspecciones entre otros).
15	Tener estimaciones del impacto que puede agregar falla.
16	Usar herramientas para la detección de fallos de seguridad
17	Realizar un pre análisis del fallo al hacer estimaciones para determinar si la detección del mismo es más compleja que la posible solución
18	Contar con una BD de patrones de errores.
19	Usar estrategias para un análisis efectivo.

El analista debe realizar pruebas, en ambientes controlados, se debe garantizar que estos ambientes contengan las mismas características del ambiente productivo, algunas veces esto se dificulta porque no se cuenta con estos ambientes, se debe tener en cuenta que no es buena práctica hacer pruebas en ambientes productivos.

5.2.4. FASE 4 IMPLEMENTACIÓN

Luego de tener la incidencia corregida, se procede a realizar el cambio en producción.

- **Herramienta**:
  - Aplicación para la creación y gestión del reporte de fallo.
  - Base de datos o inventarios de las soluciones y sus diferentes módulos
  - Base de datos con patrones de fallos.
  - Herramienta de control de cambios en el código fuente.

- **Actividad 1: Paso a producción de las correcciones**

El analista solicita al área encargada de realizar pasos a producción, ya que el desarrollador que propone la solución no debería hacer la implementación, de esta forma los cambios se realizan de manera controlada; al analista que propone el cambio es responsable de este, por tanto, debe estar acompañando a quien hace la implementación, para realizar pruebas y el seguimiento adecuado.

Entradas:

- Reporte de fallo actualizado, con cambios realizados y propuesto.
- Documentación del cambio propuesto y código de respaldo para el cambio.

Salidas:

- Reporte de fallo actualizado y resuelto
- Documentación en los diferentes repositorios.

- **Actividad 2: Realizar seguimiento**

Luego de implementado el cambio en el ambiente productivo el analista debe hacer seguimiento y pruebas para garantizar que todo quede corregido.

Entradas:

- Reporte de fallo actualizado, con cambios realizados y propuesto.

- Documentación del cambio propuesto y código de respaldo para el cambio.

Salidas:

- Reporte de fallo actualizado y resuelto
- Documentación en los diferentes repositorios.
- Pruebas exitosas y solución del fallo.

Buenas Prácticas:

Tabla 5 *Prácticas Fase 4 Implementación*

20	Documentar el código con una estructura estándar.
21	Restringir los cambios efectuados los viernes y sábados, ya que son los más propensos a inyectar fallos.
22	Actualizar los estados de los reportes de fallos.
23	Hacer un análisis del impacto a causar.
24	Hacer el cambio de manera controlada.
25	Deben realizar los cambios expertos que conozcan muchos módulos del sistema.
26	Listar los módulos donde sea necesario implementar cambios.

Es responsabilidad del desarrollador actualizar las herramientas de control de cambio y de gestión de los fallos, para que estas estén alineadas.

RECOMENDACIONES GENERALES

- **Herramienta**:
  - Aplicación para gestionar Reportes de fallos
  - Aplicación para versionar el software y tener un control de cambios.

- **Actividad 1: Actividades trasversales al proceso**

  Los diferentes participantes del proceso deben estar actualizando las diferentes herramientas (reportes de fallos, control de cambios), y notificar a los demás miembros el estado de los reportes que tengan a cargo.

Se deben planificar las diferentes actividades a realizar teniendo en cuanta los estados de los reportes pendientes por atender.

Entradas:

- Los reportes de fallo son entradas trasversales.

Salidas:

- Reportes de fallos resueltos.
- Documentación en herramientas.

Buenas Prácticas:

Las siguientes prácticas son transversales a todo el proceso de mantenimiento, por tanto, se deben tener en cuenta en todo el proceso, los cuales buscan lograr un nivel de madurez en los procesos, que se reflejaran en mejorar la calidad de los productos generados en dicho proceso, en este caso en la calidad de la atención y corrección de fallos.

Tabla 6 *Prácticas genéricas*

27	Descomponer las diferentes actividades, del proceso de mantenimiento.
28	Control de las actividades y de la ejecución de las tareas.
29	Use especialistas en mantenimiento en lugar de desarrolladores.
30	Usar procedimientos y herramientas formales de gestión de cambios.
31	Usar bibliotecas de prueba de regresión formales.
32	Realizar estudios automatizados de análisis de complejidad de aplicaciones heredadas.
33	Buscar y eliminar todos los módulos propensos a errores en aplicaciones heredadas.
34	Identificar todos los códigos muertos en las aplicaciones heredadas.
35	Renovar o re factorizar aplicaciones antes de mejoras importantes.
36	Realizar un seguimiento de todos los defectos informados por el cliente.
37	Seguir el tiempo de respuesta desde el envío hasta la reparación de defectos.
38	Seguir el tiempo de respuesta desde la presentación hasta la finalización de la petición del cambio.
29	Hacer seguimiento de la disponibilidad del software para los clientes.
40	Mantener sincronizados los repositorios asociados al proceso.

41	Estandarizar la forma de documentar el código fuente.
42	Tener procesos que se encarguen de hacer esta sincronización de forma automática.
43	Deben ser marcados los fallos una vez sean encontrados.
44	Contar con un repositorio para el reporte de fallos.
45	Contar con un repositorio para el control de cambios
46	Deben intervenir En el repositorio tanto el que genera el reporte como el desarrollador.
47	El desarrollador debe actualizar todos los repositorios.
48	Tener en cuenta la diferencia en la descripción dependiendo el rol de los participantes del proceso.

5.3. ALCANCE DE LA PROPUESTA

Esta propuesta no es un acercamiento metodológico que se pueda definir como tradicional o ágil, ya que, su objetivo es mejorar el proceso de mantenimiento correctivo, no todo el ciclo de vida del software. Adicionalmente, se busca potenciar al equipo responsable mediante la optimización de los tiempos de respuesta y atención de los incidentes que se reportan como fallos.

La mayoría de los trabajos encontrados se enfocan en una pequeña parte del proceso de mantenimiento de software, en este no se había trabajado de manera holística, por lo tanto, se toman partes de diferentes autores para armar todo el proceso completo, pero esto se convierte en una limitante a la hora de buscar referentes con los cuales comparar la propuesta realizada.

Tener en cuenta todos los escenarios posibles en un software es una actividad compleja, debido a que estos son infinitos las formas de interacción con diferentes variables en un sistema, por lo cual se debe tener presente que siempre será necesario aplicar mantenimiento correctivo.

PARTE V

VALIDACIÓN

"La vida no es sino una continua sucesión de oportunidades para sobrevivir." –
Gabriel García Márquez

CAPÍTULO 6 IMPLEMENTACIÓN

6.1. CASO DE ESTUDIO

6.1.1. CARACTERIZACIÓN DE LA COMPAÑÍA

La compañía en la cual se realizó la implementación y prueba del acercamiento metodológico es Satrack. Ofrece servicios tecnológicos de monitoreo de vehículos por medio del GPS y servicios especializados asociados al monitoreo y opera en diferentes ciudades del país, en varios países de Latino América y en Estados Unidos.

Satrack es una empresa que ofrece servicios tecnológicos, el personal y la operación está distribuido en las gerencias de ventas, operaciones, mercadeo, de las personas y gerencia de investigación y desarrollo, estas ultima cuenta con profesionales en ingeniería de sistemas e ingenieros electrónicos y mecánicos en cargados de desarrollar e integrar hardware a las soluciones que la compañía ofrece a sus clientes, también de proveer las soluciones de software para los clientes internos, y brindar soporte y atención a los requerimientos de los clientes.

La compañía recibe más de 50 reportes de fallo al día por parte de sus clientes, los cuales son escalados por medio del Área de Servicio al Cliente, los analistas de SAC reciben las notificaciones y las escalan al Área de Ingeniería, en el caso que sea necesario.

El Área de Ingeniería está dividida en equipos de desarrollo, tales como desarrollos internos, desarrollo clientes, Soporte y contención, Plataforma, TI e infraestructura, Entre otros, los cuales tienen a cargo productos en fase de desarrollo. El equipo de contención o soporte el cual está compuesto por analistas encargados de mantener la estabilidad de la plataforma, uno de ellos es un analista de nivel 1 el cual se encarga de recibir los reportes, en caso de que los pueda solucionar, los soluciona y si no, lo escala a un nivel 2, el cual puede ser un

analista del mismo equipo de contención o un miembro del equipo de desarrollo, los cuales aún están en proceso de estabilización.

6.2. PROCESO DE IMPLEMENTACIÓN

El primer paso de la implementación de la propuesta fue realizar una socialización del acercamiento metodológico propuesto, al equipo de soporte en Satrack. Ellos realizan una evaluación de la propuesta para determinar si tiene aspectos por mejorar o si cumple con las necesidades que actualmente presenta el equipo de soporte.

Como la propuesta pretende mejorar todo el proceso del ciclo de vida del mantenimiento de software, se necesita realizar una implementación por fases en el orden la propuesta realizada en el capítulo 5 para no afectar el proceso como tal y que los datos que se obtengan sean coherentes con la realidad, para que pueda luego ser evaluada la implementación.

El equipo de trabajo del caso de estudio actualmente está implementando el marco de trabajo Scrumban, el cual está alineado a metodologías de desarrollo de software agiles y es una mezcla de Scrum y Kamban, esta toma las mejores prácticas de estos referentes y las implementa en equipos de trabajo donde se atienden requerimientos de manera dinámica y cambiante y se atacan problemas de errores de software, como lo son de mantenimiento de software [53], en Satrack se tomó la práctica de aplicar iteraciones semanales, por lo cual se realizó la implementación de la propuesta durante 2 semanas.

6.2.1. PRIMER CICLO DE IMPLEMENTACIÓN

A continuación se describe el proceso de implementacion de las diferntes fases de la propuesta, como evaluación piloto se toman casos especificos que incluían clientes considerados TOP (clientes mas importantes) por la compañía. En cada fase de la propuesta implementada se describen las acciones principales realizadas, y los sucesos relevantes en

ellas, que corresponden a hallazgos importantes en el proceso y/o a información significativa a resaltar por parte de los actores participantes.

Fase 1 reporte de fallos

Para llevar a cabo esta implementación, se realizaron las siguientes acciones principales:

- En representación de los usuarios, invitación a dos Analistas de Servicio al cliente, dado que éstos son un proveedor principal de reportes de fallos.
- Proceso de sensibilización de la propuesta.
- Entrega de la descripción de la Fase 1 con las prácticas a implementar.
- Suministro de recomendaciones para aplicación de las prácticas, resaltando la importancia de documentar correctamente los reportes y de aplicar fielmente las prácticas sugeridas.

Sucesos relevantes: Los analistas de SAC inidicaron que hicieron un ejercicio serio de aplicación de las prácticas a la hora de crear los reportes de fallos. A pesar de esto, indicaron algunas dificultades a la hora de su aplicación, prácticas como "Generar títulos Estructurados" no dependen de ellos, si no de una decisión conjunta entre todos o de directrices de arquitectura que deben convertirse en estándar a seguir en los equipos.

Fase 2 Pre análisis

En la implementacion de esta fase se realizaron las siguientes acciones principales:

- Apropiación y liderazgo de esta fase por parte del analista.
- Solicitud de apoyo a toda la gerencia por parte del analista, para levantar el inventario de las soluciones.
- Levantamiento e identificación de soluciones potenciales.
- Intervención del scrum master, como lider del proceso velando por el cumplimento de la propuesta

Sucesos relevantes: El analista identificó que para poder reasignar casos no existía una matriz, dado que es una tarea individual que generalmente se basa en el conocimiento

particular del analista, pero no es un proceso documentado. Se sugiere empezar a documentar este tipo de informacion ya que ayuda a una gestión efectiva.
Por otro lado el Scrum master le hace enfasis al analista de la importancia de solicitar calidad en los reportes. En caso de no estar documentados solicitar la información o realizar una reasignación, para crear el hábito de documentar correctamente.

Fase 3 Análisis

En la implementacion de esta fase se realizaron las siguientes acciones principales:

- Implementación de prácticas propuestas, cuando llegaron reportes al estado de análisis
- Uso de herramientas de apoyo para la trazabilidad de los reportes y el versionamiento del codigo fuente.
- Realización de pruebas
- Detección de la falla

Sucesos relevantes: El analista manifiesta mucha dificultad a la hora de hacer pruebas, ya que no cuentan con ambientes que faciliten esta tarea. Adicionalmente, en algunos casos no tenía contexto de toda la solución, lo que hace mas complejo hacer pruebas pues hay muchas limtaciones. En cuanto a la detección de la falla manifiestan que se facilitó al tener casos mas documentados, ya que no solo estaba documentado por el usuario, sino que el analsista que realizó el pre-analisis también documentó el reporte con mas información.

Fase 4 Implementación

En la implementacion de esta fase se realizaron las siguientes acciones principales:

- Al momento de realizar la implementacion, comunicación a las personas de soporte TI de la realización del piloto, dado que son los que realizan los pasos a produccion.
- Identificación de prácticas propuestas que ya hacen parte del proceso de la compañía, por parte del analista.
- Actualización de herramientas de versionamiento al realizar la implementación de las correciones de fallos.

Sucesos relevantes: Algunas de las prácticas propuestas que generaron complejidad e inconformidad en su implementación, son por ejemplo "*Documentar el código con una estructura estándar*", es complejo generar estándares ya que esto no depende tanto del área de soporte si no de arquitectura, quienes estan trabajando en su adopción. Adiconalmente, manifiestan que documentar dos herramientas se vuelve muy tedioso ya que requiere mayor cantidad de tiempo.

6.2.2. SEGUNDO CICLO DE IMPLEMENTACIÓN

En esta fase se realiza una evaluación del proceso con los miembros del equipo en una sesión de retrospectiva, que se realiza normalmente al finalizar la iteración, en donde brindan sus observaciones respecto a la propuesta y diligencian una encuesta para obtener datos de la evaluación y generar datos concluyentes.

Retrospectiva 1:

La retrospectiva es una reunión que se realiza al finalizar cada iteración, con el objetivo del mejoramiento continuo. El equipo, conformado por 4 personas, aprovechó este espacio para hacer una evaluación de la propuesta. Adicionalmente, invitaron a los 2 analistas de servicio al cliente que estaban apoyando la implementación de la Fase 1. Todos los participantes de la reunión brindaron sus apreciaciones respecto a la propuesta, entre las cuales se pueden resaltar:

- Se requiere, no sólo involucrar al equipo para la aplicación de la aproximación metodológica, si no direccionarla desde Arquitectura o Gerencia, ya que se proponen herramientas o prácticas que no dependen del equipo de trabajo.
- Hay prácticas que deben ser más detalladas o mejor explicadas.
- Hay prácticas innovadoras, como el generar una calificación del reporte de parte y parte. Es una práctica que suena muy bien y podría ser un nuevo indicador para los equipos.
- Se deben desarrollar actividades antes de empezar a implementar la propuesta, que permita identificar aspectos previos o pre-requisitos para una implementación.
- Es muy interesante que se trabaje en mejorar el proceso de mantenimiento.

- La propuesta no tiene en cuenta los casos tipo *fire* (casos que son para atender tan pronto se generen).

Luego de tener estas apreciaciones de los participantes, la Scrum master orienta la discusión hacia cómo pueden, como equipo, ayudar a mejorar la propuesta para que pueda aplicarse exitosamente. Ante esto, los participantes indicaron dos aspectos importantes: (i) la mejor forma de mejorarla es usarla y hacer las modificaciones que se requieran en el camino, pues la propuesta es una buena base para iniciar; y (ii) Realizar un proceso juicioso y serio en la implementación, buscando medir de alguna forma la aplicación de las prácticas y el efecto en la mejora del proceso.

Retrospectiva 2:

Luego de haber pasado 2 semanas con la implementación de la propuesta, se realiza nuevamente una reunión de retrospectiva, la cual se centra nuevamente en el tema de la implementación. En esta ocasión se le propone una evaluación a la metodología, con los criterios de evaluación mostrados en apartado 6.3, a lo cual los 6 participantes de la reunión acceden. Inicialmente tuvieron diferencias en el resultado de la evaluación, pues de 9 criterios (6/3,7/2, 5/4) luego de realizar un consenso, dio como resultado final (6/3), lo cual se podrá observar en la Tabla 12 que se describe en el análisis de resultados.

6.3. CRITERIOS DE VALIDACIÓN

Los criterios de evaluación se obtienen del trabajo propuesto por Monie [54], quien plantea una herramienta para la evaluación y comparación de metodologías. En este caso se realizó únicamente la evaluación y se hizo la adaptación de algunas preguntas, ya que la propuesta de referencia es orientada a proyectos de minería de datos. De los 4 aspectos que propone el autor, se tendrán en cuenta los dos que más se ajustan al objetivo de esta investigación. Los criterios son los siguientes, y para cada uno se proponen una serie de características:

Criterio 1: Nivel de detalle en la descripción de las actividades de cada fase:

- ¿Se definen actividades específicas para cada fase del proceso?
- ¿Se explicitan los pasos a seguir para llevar a cabo cada actividad?
- ¿Se definen las entradas de cada actividad?
- ¿Se definen las salidas o entregables de cada actividad?
- ¿Se provee una guía de buenas prácticas para cada una de las actividades específicas?

Criterio 2 Escenarios de aplicación:

- ¿Se especifican actividades para la definición y el análisis del problema u oportunidad con el cual colaborará?
- ¿Se consideran puntos de partida alternativos donde el usuario no es claro con el reporte del problema?
- ¿La metodología es independiente del dominio de aplicación?
- ¿La metodología es aplicable a proyectos de diferente tamaño?

Estas preguntas se aplicaron en una encuesta, que fue diligenciada por los 6 miembros del equipo de mantenimiento, indicando si se cumple o no con cada característica por criterio, como se presenta en la Tabla 11. De esta manera se realizó la evaluación obteniendo un puntaje de (6/3) dependiendo de si aplican o no en la aproximación metodológica propuesta en este trabajo.

***Tabla 7**. Muestra de formato de evaluación del acercamiento metodológico*

Aspecto	**Preguntas**	**Si**	**No**
Nivel de detalle en la descripción de las actividades de cada fase	¿Se definen actividades específicas para cada fase del proceso?		
	¿Se explicitan los pasos a seguir para llevar a cabo cada actividad?		
	¿Se definen las entradas de cada actividad?		
	¿Se definen las salidas o entregables de cada actividad?		
	¿Se provee una guía de buenas prácticas para cada una de las actividades específicas?		
Escenarios de aplicación	¿Se especifican actividades para la definición y el análisis del problema u oportunidad con el cual colaborará?		
	¿Se consideran puntos de partida alternativos donde el usuario no es claro con el reporte del problema?		
	¿La metodología es independiente del dominio de aplicación?		
	¿La metodología es aplicable a proyectos de diferente tamaño?		

6.4. RESULTADOS

Los resultados de la implementación se obtienen y revisan en las retrospectivas realizadas y descritas en la fase de implementación el luego de finalizar los esprints donde se ejecutó el piloto, mostrando una aceptación de la propuesta con las sugerencias mostradas en el numeral 6.2.2, el objetivo de la implementación era medir que tan aceptable y pertinente es la propuesta realizada, por eso se aplicaron unos criterios de evaluación, que generaron resultados individuales de los participantes, y en la última retrospectiva se obtiene una evaluación que se concertó entre todos los miembros del equipo, y es la que se muestra en la tabla 12 en el apartado de análisis de los resultados, donde se puede notar que de 9 criterios evaluados la propuesta cumple con 6, adicionalmente el equipo concuerda con indicar que la propuesta es acertada y pertinente ya que les ayudara a lograr un proceso más maduro y organizado.

6.5. ANÁLISIS DE RESULTADOS

Luego de realizar la implementación y ejecutar la reunión con los miembros del equipo, se obtienen diferentes posturas respecto a la propuesta, algunos coinciden que ésta es pertinente y propone buenas prácticas y otros indican que las prácticas son complejas de implementar y de definir si se aplican o no. Se realizó un proceso de discusión y se logró obtener un consenso en el cual todos acordaron dar la evaluación que se puede observar en la Tabla 12. En el Anexo 2 se incluyen los resultados de todas las evaluaciones realizadas.

***Tabla 8**. Resultado de Evaluación Conjunta del equipo*

Aspecto	Preguntas	Si	No
Nivel de detalle en la descripción de las actividades de cada fase	¿Se definen actividades específicas para cada fase del proceso?	x	
	¿Se explicitan los pasos a seguir para llevar a cabo cada actividad?		x
	¿Se definen las entradas de cada actividad?	x	
	¿Se definen las salidas o entregables de cada actividad?	x	

	¿Se provee una guía de buenas prácticas para cada una de las actividades específicas?	x	
Escenarios de aplicación	¿Se especifican actividades para la definición y el análisis del problema u oportunidad con el cual colaborará?	x	
	¿Se consideran puntos de partida alternativos donde el usuario no es claro con el reporte del problema?		x
	¿La metodología es independiente del dominio de aplicación?		x
	¿La metodología es aplicable a proyectos de diferente tamaño?	x	
		6	3

En este resultado se puede observar que la propuesta cumple con 6/9 características de los aspectos evaluados, lo cual corresponde al 67%. Se puede considerar un resultado favorable, pero con algunos aspectos por mejorar, esto sin dejar de lado el hecho que la propuesta es un acercamiento metodológico, y hace parte de una fase del ciclo de vida del software que no abarca todo el proceso de desarrollo del producto. Se reconoció que de manera inicial los miembros del equipo visualizarán el acercamiento metodológico propuesto como una metodología que abarca todo el ciclo de vida del software, lo cual se clarificó luego del proceso de evaluación.

En el proceso de evaluación realizado en las dos retrospectivas, fue posible identificar varios factores susceptibles de mejora:

- Elementos no evidenciados en la aproximación metodológica, como lo es que algunas prácticas requieren del apoyo de otras áreas, no solo del equipo de mantenimiento.
- Ejercicio previo a la implementación en la que se identifique si un equipo de mantenimiento cumple con los pre-requisitos necesarios para aplicar la propuesta.
- Gestión o aseguramiento, previo a la aplicación de la propuesta, no solo de los miembros del equipo de mantenimiento, sino el de toda la compañía, en especial de las áreas que interactúan con la de soporte, como lo son TI, Equipos de desarrollo y Arquitectura empresarial.
- La propuesta inicialmente no incluía SPEM, lo cual pudo generar un choque inicial en el acercamiento a la propuesta y algunas confusiones, no se tenia claro que practicas implementar en que fases y en qué actividades específicas se debían realizar en cada parte del proceso.

Estos factores sirvieron para mejorar la propuesta realizada inicialmente, dando como resultado la que se describe finalmente en el capítulo 5.

PARTE VI

CONCLUSIONES

"La vida no es la que uno vivió, sino la que uno recuerda y cómo la recuerda para contarla." -- Gabriel García Márquez

CAPÍTULO 7 CONCLUSIONES Y TRABAJO FUTURO

7.1. CONCLUSIONES

Luego de haber realizado la investigación y generar una aproximación metodológica para el proceso de mantenimiento de software, se puede concluir que:

- El proceso de mantenimiento de software, es una de las fases más importantes y costosas en el ciclo de vida del software, ya que de éste depende la operatividad del software y el mantenerse funcional a lo largo del tiempo y los cambios que surgen en el medio.
- El proceso de la selección de las mejores prácticas, fue arduo debido a que requirió una revisión de la literatura desde diferentes partes del proceso, sin embargo, a medida que el proceso de mantenimiento vaya mejorando con las practicas propuestas pueden ir surgiendo nuevas practicas que ayuden a mejorar dicho proceso, por lo tanto, se puede decir que al aplicar la propuesta para que esta sea valida en el tiempo se deberán ir haciendo ajustes.
- La propuesta planteada es una primera aproximación metodológica aplicable y válida ya que aporta al mejoramiento del proceso de mantenimiento correctivo de software. Además, es pertinente ya que aporta buenas prácticas y recomienda el uso de artefactos que ayudan a mejorar el proceso de manera significativa. Se detectan mejoras a realizar en cuanto a la evaluación de la implementación, es decir si se puede realmente comprobar una mejora en el proceso de mantenimiento.
- el mantenimiento correctivo siempre estará presente, dado que por más escenarios que se abarquen, siempre saldrán nuevos cuando el software está en producción y es utilizado por los usuarios.
- En el equipo de trabajo donde se implementó la propuesta se muestra interés en la misma debido a que les ayuda a tener mas claridad del proceso, pero se

encuentra que muchas de las practicas que se proponen no dependen del equipo como tal para su implementación ya que son mas temas de arquitectura o políticas de compañía.

- La comunicación es un factor importante a la hora de implementar metodología con mejores prácticas.

7.2. TRABAJOS FUTUROS

En los trabajos futuros se espera profundizar más en si la propuesta realmente ayuda a optimizar el proceso de mantenimiento, dado que por el alcance de la investigación no se logró colectar datos que permitieran comparar si el equipo es más eficiente al implementar la propuesta (comparando un estado anterior y un estado posterior a la implementación).

Adicionalmente, se espera en investigaciones futuras crear las métricas específicas para cada práctica y de esta manera definir y mostrar con resultados estadísticos por qué se reconocen como buenas prácticas. La mayoría de los autores se refieren al tema de buenas prácticas, pero no existen validaciones o evaluaciones de ellas con datos estadísticos que demuestren cómo las prácticas propuestas mejoran el proceso.

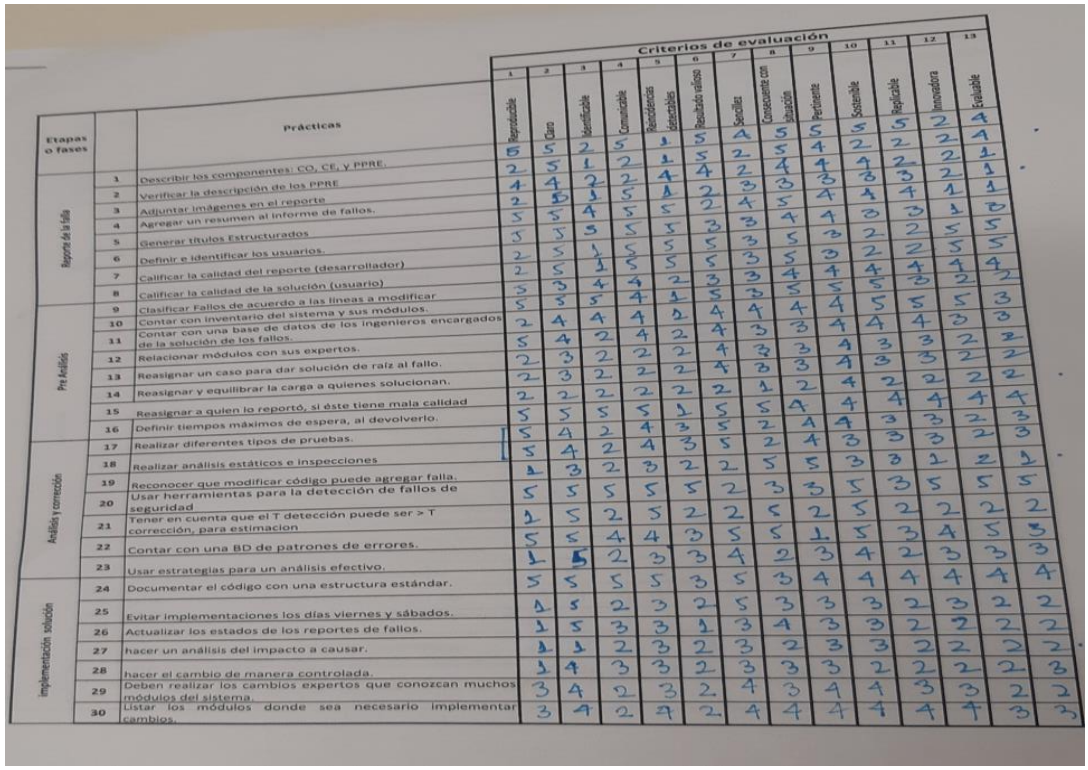
ANEXOS

ANEXO 1 EVALUACIONES DE LAS PRACTICAS

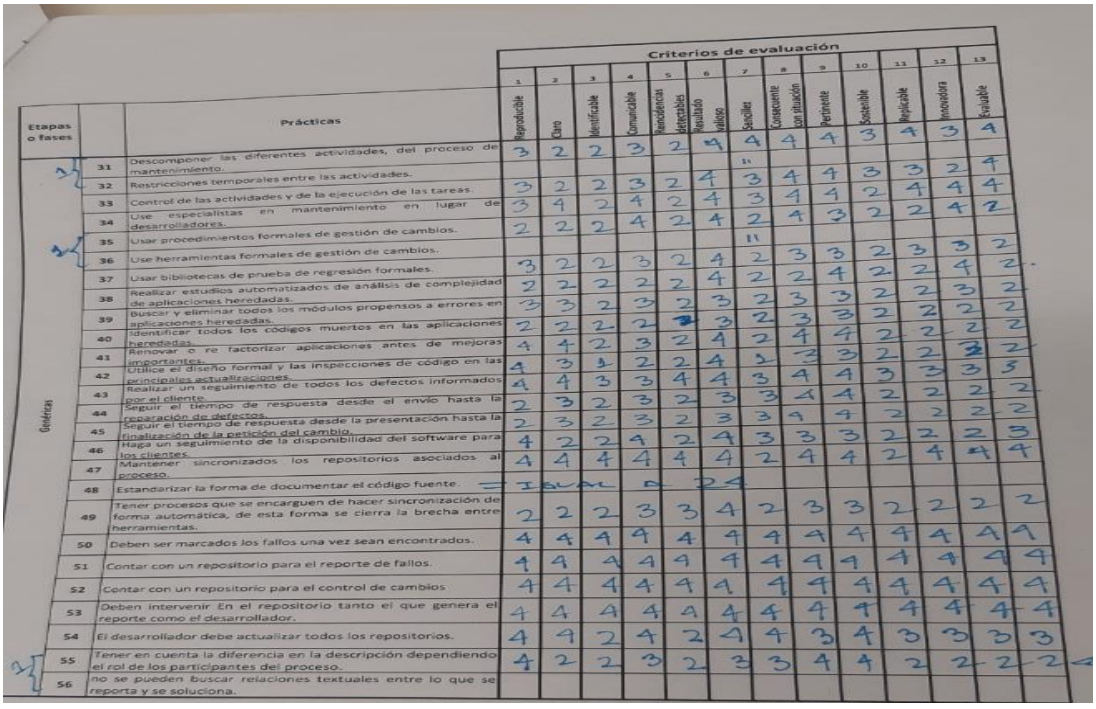
			Criterios de evaluación												
			1	2	3	4	5	6	7	8	9	10	11	12	13
Etapas o fases		Prácticas	Reproducible	Claro	Identificable	Comunicable	Reincidencias detectables	Resultado valioso	Sencillez	Consecuente con situación	Pertinente	Sostenible	Replicable	Innovadora	Evaluable
Reporte de la falla	1	Describir los componentes: CO, CE, y PPRE.	5	4	3	5	2	1	1	4	5	5	5	3	5
	2	Verificar la descripción de los PPRE	5	4	3	5	2	1	1	4	5	5	5	3	5
	3	Adjuntar imágenes en el reporte	1	5	5	5	1	3	5	4	5	2	2	3	4
	4	Agregar un resumen al informe de fallos.	4	5	3	5	1	3	5	3	5	3	3	2	4
	5	Generar títulos Estructurados	5	5	5	5	2	2	4	5	5	4	5	3	5
	6	Definir e identificar los usuarios.	3	3	5	4	4	2	5	5	5	5	5	2	4
	7	Calificar la calidad del reporte (desarrollador)	2	5	5	4	4	3	3	4	5	4	4	4	5
	8	Calificar la calidad de la solución (usuario)	1	5	3	4	3	5	3	4	5	3	3	5	5
Pre Análisis	9	Clasificar Fallos de acuerdo a las lineas a modificar	1	2	3	1	1	5	1	5	5	4	4	4	5
	10	Contar con inventario del sistema y sus módulos.	3	3	5	5	4	5	4	4	5	3	4	1	3
	11	Contar con una base de datos de los ingenieros encargados de la solución de los fallos.	4	3	5	5	5	5	1	5	5	2	4	3	2
	12	Relacionar módulos con sus expertos.	5	3	5	5	5	5	1	5	5	1	4	3	2
	13	Reasignar un caso para dar solución de raíz al fallo.	3	3	2	2	1	5	4	5	5	2	2	4	1
	14	Reasignar y equilibrar la carga a quienes solucionan.	2	2	1	1	1	5	1	5	5	1	1	5	1
	15	Reasignar a quien lo reportó, si éste tiene mala calidad	2	2	1	1	1	5	1	5	5	1	1	5	1
	16	Definir tiempos máximos de espera, al devolverlo.	1	1	1	1	5	5	1	5	4	4	2	5	2
Análisis y corrección	17	Realizar diferentes tipos de pruebas.	4	3	2	1	1	5	2	4	5	3	3	2	3
	18	Realizar análisis estáticos e inspecciones	4	3	2	1	1	5	2	4	5	3	3	2	3
	19	Reconocer que modificar código puede agregar falla.	1	1	1	1	5	5	3	5	5	2	2	2	4
	20	Usar herramientas para la detección de fallos de seguridad	4	3	5	4	4	5	1	4	5	3	4	2	5
	21	Tener en cuenta que el T detección puede ser > T corrección, para estimacion	1	4	4	2	3	5	2	4	5	3	2	2	4
	22	Contar con una BD de patrones de errores.	5	5	5	5	5	5	1	5	5	2	3	1	4
	23	Usar estrategias para un análisis efectivo.	5	5	5	3	4	4	2	4	3	2	4	2	1
Implementación solución	24	Documentar el código con una estructura estándar.	5	5	5	5	3	5	2	3	4	4	4	1	1
	25	Evitar implementaciones los días viernes y sábados.	1	1	1	1	1	2	1	3	4	5	1	1	1
	26	Actualizar los estados de los reportes de fallos.	3	4	5	3	5	4	2	4	5	5	3	1	1
	27	hacer un análisis del impacto a causar.	1	1	1	1	5	5	3	5	5	2	2	2	4
	28	hacer el cambio de manera controlada.	4	4	2	3	1	5	2	5	5	2	3	1	3
	29	Deben realizar los cambios expertos que conozcan muchos módulos del sistema.	2	2	4	1	5	4	1	3	5	1	2	1	4
	30	Listar los módulos donde sea necesario implementar cambios	5	5	5	5	2	2	4	5	5	4	5	3	5

tapas / fases		Prácticas	Criterios de evaluación												
			1	2	3	4	5	6	7	8	9	10	11	12	13
			Reproducible	Claro	Identificable	Comunicable	Reincidencias detectables	Resultado valioso	Sencillez	Consecuente con situación	Pertinente	Sostenible	Replicable	Innovadora	Evaluable
Genéricas	31	Descomponer las diferentes actividades, del proceso de mantenimiento.	4	3	1	1	2	4	5	5	5	3	4	1	2
	32	Restricciones temporales entre las actividades.	4	2	1	1	2	5	3	5	5	2	4	1	2
	33	Control de las actividades y de la ejecución de las tareas.	5	2	1	1	2	5	2	5	5	1	5	1	2
	34	Use especialistas en mantenimiento en lugar de desarrolladores.	2	2	4	1	5	4	1	3	5	1	2	1	4
	35	Usar procedimientos formales de gestión de cambios.	5	5	5	5	4	4	2	4	4	1	3	1	4
	36	Use herramientas formales de gestión de cambios.	5	5	5	5	5	4	1	3	4	1	4	2	4
	37	Usar bibliotecas de prueba de regresión formales.	5	4	5	4	5	5	1	4	3	1	4	5	5
	38	Realizar estudios automatizados de análisis de complejidad de aplicaciones heredadas.	4	3	3	2	1	5	1	1	2	1	4	5	2
	39	Buscar y eliminar todos los módulos propensos a errores en aplicaciones heredadas.	2	2	2	1	1	3	1	1	1	5	2	1	4
	40	Identificar todos los códigos muertos en las aplicaciones heredadas.	2	2	2	2	1	5	1	3	4	4	1	5	4
	41	Renovar o re factorizar aplicaciones antes de mejoras importantes.	2	2	1	1	1	5	2	4	5	3	1	4	3
	42	Utilice el diseño formal y las inspecciones de código en las principales actualizaciones.	4	3	2	1	1	5	2	4	5	3	3	2	3
	43	Realizar un seguimiento de todos los defectos informados por el cliente.	5	5	5	5	4	4	1	3	3	1	4	1	2
	44	Seguir el tiempo de respuesta desde el envío hasta la reparación de defectos.	1	2	1	1	1	5	5	4	5	4	3	2	5
	45	Seguir el tiempo de respuesta desde la presentación hasta la finalización de la petición del cambio.	1	2	1	1	1	5	5	4	5	4	3	2	5
	46	Haga un seguimiento de la disponibilidad del software para los clientes.	1	2	1	2	1	5	1	3	4	2	4	1	5
	47	Mantener sincronizados los repositorios asociados al proceso.	5	3	3	4	4	3	1	4	5	1	3	4	2
	48	Estandarizar la forma de documentar el código fuente.	5	5	5	5	3	5	2	4	4	4	4	1	2
	49	Tener procesos que se encarguen de hacer sincronización de forma automática, de esta forma se cierra la brecha entre herramientas.	4	3	3	5	4	5	1	2	3	1	1	5	3
	50	Deben ser marcados los fallos una vez sean encontrados.	4	4	4	5	5	5	5	3	5	3	3	1	2
	51	Contar con un repositorio para el reporte de fallos.	5	5	5	5	5	5	1	5	5	2	3	1	4
	52	Contar con un repositorio para el control de cambios	5	4	5	5	4	5	1	3	2	2	4	1	1
	53	Deben intervenir En el repositorio tanto el que genera el reporte como el desarrollador.	5	5	5	5	5	5	1	2	2	1	4	1	1
	54	El desarrollador debe actualizar todos los repositorios.	5	5	5	5	5	5	1	2	2	1	3	1	1
	55	Tener en cuenta la diferencia en la descripción dependiendo el rol de los participantes del proceso.	1	5	3	4	2	3	4	3	5	1	1	2	2
	56	no se pueden buscar relaciones textuales entre lo que se reporta y se soluciona.	2	4	3	5	3	3	2	5	4	3	3	1	1

- La 24 y la 48 suenan similares
- se debe buscar redaccion estandar: por ejemplo iniciar con verbos en infinitivo.
- la 49 es confusa: Sincronizar qué? Que herramientas?



Etapas o fases		Prácticas	Criterios de evaluación												
			1 Reproducible	2 Claro	3 Identificable	4 Comunicable	5 Reincidencias detectables	6 Resultado valioso	7 Sencillez	8 Consecuente con situación	9 Pertinente	10 Sostenible	11 Replicable	12 Innovadora	13 Evaluable
Reporte de la falla	1	Describir los componentes: CO, CE, y PPRE.	5	5	2	5	1	5	4	5	5	5	5	2	4
	2	Verificar la descripción de los PPRE	2	5	1	2	1	5	2	5	4	2	2	2	4
	3	Adjuntar imágenes en el reporte	4	4	2	2	4	4	2	4	4	4	2	2	1
	4	Agregar un resumen al informe de fallos.	2	5	1	5	1	2	3	3	3	3	3	2	1
	5	Generar títulos Estructurados	5	5	4	5	5	2	4	5	4	4	4	1	1
	6	Definir e identificar los usuarios.	5	5	5	5	5	3	3	4	4	3	3	1	2
	7	Calificar la calidad del reporte (desarrollador)	2	5	1	5	5	5	3	5	3	2	2	5	5
	8	Calificar la calidad de la solución (usuario)	2	5	1	5	5	5	3	5	3	2	2	5	5
Pre Análisis	9	Clasificar Fallos de acuerdo a las líneas a modificar	5	3	4	4	2	3	3	4	4	4	4	4	4
	10	Contar con inventario del sistema y sus módulos.	5	5	5	4	1	5	3	5	5	5	5	3	2
	11	Contar con una base de datos de los ingenieros encargados de la solución de los fallos.	2	4	4	4	2	4	4	4	4	5	5	5	3
	12	Relacionar módulos con sus expertos.	5	4	2	4	2	4	3	3	4	4	4	3	3
	13	Reasignar un caso para dar solución de raíz al fallo.	2	3	2	2	2	4	3	3	4	3	3	2	2
	14	Reasignar y equilibrar la carga a quienes solucionan.	2	3	2	2	2	4	3	3	4	3	3	2	2
	15	Reasignar a quien lo reportó, si éste tiene mala calidad	2	2	2	2	2	2	1	2	4	2	2	2	2
	16	Definir tiempos máximos de espera, al devolverlo.	5	5	5	5	1	5	5	4	4	4	4	4	4
Análisis y corrección	17	Realizar diferentes tipos de pruebas.	5	4	2	4	3	5	2	4	4	3	3	3	3
	18	Realizar análisis estáticos e inspecciones	5	4	2	4	3	5	2	4	3	3	3	2	3
	19	Reconocer que modificar código puede agregar falla.	1	3	2	3	2	2	5	5	3	3	1	2	1
	20	Usar herramientas para la detección de fallos de seguridad	5	5	5	5	5	2	3	3	5	3	3	5	5
	21	Tener en cuenta que el T detección puede ser > T corrección, para estimacion	1	5	2	5	2	2	5	2	5	2	2	2	2
	22	Contar con una BD de patrones de errores.	5	5	4	4	3	5	5	1	5	3	4	5	3
	23	Usar estrategias para un análisis efectivo.	1	5	2	3	3	4	2	3	4	2	3	3	3
Implementación solución	24	Documentar el código con una estructura estándar.	5	5	5	5	3	5	3	4	4	4	4	4	4
	25	Evitar implementaciones los días viernes y sábados.	1	5	2	3	2	5	3	3	3	2	3	2	2
	26	Actualizar los estados de los reportes de fallos.	1	5	3	3	1	3	4	3	3	2	2	2	2
	27	hacer un análisis del impacto a causar.	1	1	2	3	2	3	2	3	3	2	2	2	2
	28	hacer el cambio de manera controlada.	1	4	3	3	2	3	3	3	2	2	2	2	3
	29	Deben realizar los cambios expertos que conozcan muchos módulos del sistema.	3	4	2	3	2	4	3	4	4	3	3	2	2
	30	Listar los módulos donde sea necesario implementar cambios.	3	4	2	3	2	4	4	4	4	4	4	3	3



Etapas o fases		Prácticas	Criterios de evaluación												
			1 Reproducible	2 Claro	3 Identificable	4 Comunicable	5 Reincidencias detectables	6 Resultado valioso	7 Sencillez	8 Consecuente con situación	9 Pertinente	10 Sostenible	11 Replicable	12 Innovadora	13 Evaluable
Genéricas	31	Descomponer las diferentes actividades, del proceso de mantenimiento.	3	2	2	3	2	4	4	4	4	3	4	3	4
	32	Restricciones temporales entre las actividades.	3	2	2	3	2	4	3	4	4	3	3	2	4
	33	Control de las actividades y de la ejecución de las tareas.	3	4	2	4	2	4	3	4	4	2	4	4	4
	34	Use especialistas en mantenimiento en lugar de desarrolladores.	2	2	2	4	2	4	2	4	3	2	2	4	2
	35	Usar procedimientos formales de gestión de cambios.													
	36	Use herramientas formales de gestión de cambios.													
	37	Usar bibliotecas de prueba de regresión formales.	3	2	2	3	2	4	2	3	3	2	3	3	2
	38	Realizar estudios automatizados de análisis de complejidad de aplicaciones heredadas.	2	2	2	2	2	4	2	2	4	2	2	4	2
	39	buscar y eliminar todos los módulos propensos a errores en aplicaciones heredadas.	3	3	2	3	2	3	2	3	3	2	2	3	2
	40	Identificar todos los códigos muertos en las aplicaciones heredadas.	2	2	2	2	3	3	2	3	3	2	2	2	2
	41	Renovar o re factorizar aplicaciones antes de mejoras importantes.	4	4	2	3	2	4	2	4	4	2	2	2	2
	42	Utilice el diseño formal y las inspecciones de código en las principales actualizaciones.	4	3	1	2	2	4	1	3	3	2	2	3	2
	43	Realizar un seguimiento de todos los defectos informados por el cliente.	4	4	3	3	4	4	3	4	4	3	3	3	3
	44	Seguir el tiempo de respuesta desde el envío hasta la reparación de defectos.	2	3	2	3	2	3	3	4	4	2	2	2	2
	45	Seguir el tiempo de respuesta desde la presentación hasta la finalización de la petición del cambio.	2	3	2	3	2	3	3	4	4	2	2	2	2
	46	Haga un seguimiento de la disponibilidad del software para los clientes.	4	2	2	4	2	4	3	3	3	2	2	2	3
	47	Mantener sincronizados los repositorios asociados al proceso.	4	4	4	4	4	4	2	4	4	2	4	4	4
	48	Estandarizar la forma de documentar el código fuente.	I	G	U	A	L	D	A	D					
	49	Tener procesos que se encarguen de hacer sincronización de forma automática, de esta forma se cierra la brecha entre herramientas.	2	2	2	3	3	4	2	3	3	2	2	2	2
	50	Deben ser marcados los fallos una vez sean encontrados.	4	4	4	4	4	4	4	4	4	4	4	4	4
	51	Contar con un repositorio para el reporte de fallos.	4	4	4	4	4	4	4	4	4	4	4	4	4
	52	Contar con un repositorio para el control de cambios	4	4	4	4	4	4	4	4	4	4	4	4	4
	53	Deben intervenir En el repositorio tanto el que genera el reporte como el desarrollador.	4	4	4	4	4	4	4	4	4	4	4	4	4
	54	El desarrollador debe actualizar todos los repositorios.	4	4	2	4	2	4	4	3	4	3	3	3	3
	55	Tener en cuenta la diferencia en la descripción dependiendo el rol de los participantes del proceso.	4	2	2	3	2	3	3	4	4	2	2	2	2
	56	no se pueden buscar relaciones textuales entre lo que se reporta y se soluciona.													

Evaluación 3

Evaluación 4

Evaluación 5

Evaluación 6

ANEXO 2 EVALUACIONES PROPUESTA

Aspecto	Preguntas	Si	No
Nivel de detalle en la descripción de las actividades de cada fase	¿Se definen actividades específicas para cada fase del proceso?	x	
	¿Se explicitan los pasos a seguir para llevar a cabo cada actividad?		x
	¿Se definen las entradas de cada actividad?	x	
	¿Se definen las salidas o entregables de cada actividad?	x	
	¿Se provee una guía de buenas prácticas para cada una de las actividades específicas?	x	
Escenarios de aplicación	¿Se especifican actividades para la definición y el análisis del problema u oportunidad con el cual colaborará?	x	
	¿Se consideran puntos de partida alternativos donde el usuario no es claro con el reporte del problema?		x
	¿La metodología es independiente del dominio de aplicación?		x
	¿La metodología es aplicable a proyectos de diferente tamaño?	x	
	Juan	6	3

Aspecto	Preguntas	Si	No
Nivel de detalle en la descripción de las actividades de cada fase	¿Se definen actividades específicas para cada fase del proceso?	x	
	¿Se explicitan los pasos a seguir para llevar a cabo cada actividad?	x	
	¿Se definen las entradas de cada actividad?	x	
	¿Se definen las salidas o entregables de cada actividad?		x
	¿Se provee una guía de buenas prácticas para cada una de las actividades específicas?	x	
Escenarios de aplicación	¿Se especifican actividades para la definición y el análisis del problema u oportunidad con el cual colaborará?	x	
	¿Se consideran puntos de partida alternativos donde el usuario no es claro con el reporte del problema?		x
	¿La metodología es independiente del dominio de aplicación?		x
	¿La metodología es aplicable a proyectos de diferente tamaño?	x	
	Daniel	6	3

Aspecto	Preguntas	Si	No
Nivel de detalle en la descripción de las actividades de cada fase	¿Se definen actividades específicas para cada fase del proceso?	x	
	¿Se explicitan los pasos a seguir para llevar a cabo cada actividad?		x
	¿Se definen las entradas de cada actividad?	x	
	¿Se definen las salidas o entregables de cada actividad?	x	
	¿Se provee una guía de buenas prácticas para cada una de las actividades específicas?	x	
Escenarios de aplicación	¿Se especifican actividades para la definición y el análisis del problema u oportunidad con el cual colaborará?	x	
	¿Se consideran puntos de partida alternativos donde el usuario no es claro con el reporte del problema?	x	
	¿La metodología es independiente del dominio de aplicación?		x
	¿La metodología es aplicable a proyectos de diferente tamaño?	x	
	Erica SAC	7	2

Aspecto	Preguntas	Si	No
Nivel de detalle en la descripción de las actividades de cada fase	¿Se definen actividades específicas para cada fase del proceso?	x	
	¿Se explicitan los pasos a seguir para llevar a cabo cada actividad?		x
	¿Se definen las entradas de cada actividad?	x	
	¿Se definen las salidas o entregables de cada actividad?	x	
	¿Se provee una guía de buenas prácticas para cada una de las actividades específicas?	x	
Escenarios de aplicación	¿Se especifican actividades para la definición y el análisis del problema u oportunidad con el cual colaborará?	x	
	¿Se consideran puntos de partida alternativos donde el usuario no es claro con el reporte del problema?		x
	¿La metodología es independiente del dominio de aplicación?		x
	¿La metodología es aplicable a proyectos de diferente tamaño?		x
	Estiven SAC	5	4

Aspecto	Preguntas	Si	No
Nivel de detalle en la descripción de las actividades de cada fase	¿Se definen actividades específicas para cada fase del proceso?	x	
	¿Se explicitan los pasos a seguir para llevar a cabo cada actividad?		x
	¿Se definen las entradas de cada actividad?	x	
	¿Se definen las salidas o entregables de cada actividad?	x	
	¿Se provee una guía de buenas prácticas para cada una de las actividades específicas?	x	
Escenarios de aplicación	¿Se especifican actividades para la definición y el análisis del problema u oportunidad con el cual colaborará?		x
	¿Se consideran puntos de partida alternativos donde el usuario no es claro con el reporte del problema?		x
	¿La metodología es independiente del dominio de aplicación?	x	
	¿La metodología es aplicable a proyectos de diferente tamaño?		x
	Alex Analista Nivel 1	5	4

CAPÍTULO 8 REFERENCIAS

[1] J. Erazo, M. Andrés, F. Gómez, and F. J. Pino, "Generando productos software mantenibles desde el proceso de desarrollo : El modelo de referencia MANTuS Creating maintainable software products from the development process : The reference model MANTuS," *Ingeniare. Rev. Chil. Ing.*, vol. 24, pp. 420–434, 2016.

[2] F. Thung, "Automatic prediction of bug fixing effort measured by code churn size," *Proc. 5th Int. Work. Softw. Min. - SoftwareMining 2016*, pp. 18–23, 2016.

[3] P. J. Guo, "' Not My Bug !' and Other Reasons for Software Bug Report Reassignments," 2011.

[4] A. T. Nguyen, "Multi-layered Approach for Recovering Links between Bug Reports and Fixes," pp. 1–11.

[5] P. Bourque and R. E. Fairley, *Guide to the Software Engineering - Body of Knowledge.* 2014.

[6] F. Ruiz and M. Polo, "Mantenimiento del Software," *Grup. Alarcos, Dep. Inform{á}tica la Univ. Castilla-La Mancha*, p. 109, 2007.

[7] I. Sommerville, *INGENIERÍA DE SOFTWARE*, 7th ed. MAdrid, 2005.

[8] P. Bourque and R. E. Fairley, *Guide to the Software Engineering - Body of Knowledge.* 2014.

[9] D. Hovemeyer and W. Pugh, "Finding Bugs is Easy."

[10] S. Davies and M. Roper, "What ' s in a Bug Report ?"

[11] R. Bryce, "BUG WARS : A COMPETITIVE EXERCISE TO FIND BUGS IN," pp. 43–50.

[12] T. Zimmermann and A. Artıs, "Impact of switching bug trackers: a case study on a medium-sized open source project," *Hal.Inria.Fr (Hal-01951176V2F)*, 2019.

[13] Ian Sommerville, *Ingeniería de software 9*, 9th ed. .

[14] BRITISH STANDARD Technical Committee ISO/IEC, "Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) —System and software quality Models BS ISO/IEC 25010:2011," 2013.

[15] A. H. Galvis Panqueva, "Ingenieria de Software Educativo," *Informática Ind.*, p. 687, 1992.

[16] ISO/IEC, *INTERNATIONAL STANDARD ISO / IEC Software Engineering — Software Life Cycle Processes — Maintenance 14764*, vol. 2006. 2006.

[17] M. D'Ambros, M. Lanza, and M. Pinzger, "A bug's life visualizing a bug database," *Viss. 2007 - Proc. 4th IEEE Int. Work. Vis. Softw. Underst. Anal.*, pp. 113–120, 2007.

[18] ANSI, *International Standard ISO / IEC FDIS 9126-1*, vol. 2011. 2011.

[19] Y. Wang *et al.*, "Bug Localization via Supervised Topic Modeling," *Proc. - IEEE Int. Conf. Data Mining, ICDM*, vol. 2018-Novem, pp. 607–616, 2018.

[20] N. Chapin, "Software maintenance life cycle." pp. 6–13, 2003.

[21] Z. Jie, W. Xiaoyin, H. A. O. Dan, X. I. E. Bing, Z. Lu, and M. E. I. Hong, "A survey on bug-report analysis," vol. 58, no. February, pp. 1–24, 2015.

[22] SEI, "CMMI® para Desarrollo. Guía para la integración de procesos y la mejora de productos. Tercera edición., Versión 1.3," p. 555, 2010.

[23] C. Servicios and C. Svc, "CMMI ® para Servicios, Versión 1.3," 2013.

[24] "NORMAS ISO 25000." [Online]. Available: https://iso25000.com/index.php/normas-iso-25000. [Accessed: 11-Apr-2019].

[25] M. Rodríguez and M. Piattini, "Experiencias en la Industria del Software: Certificación del Producto con ISO/IEC 25000," p. 14, 2015.

[26] "Metodología - Qué es y Definición 2019." [Online]. Available: https://conceptodefinicion.de/metodologia/. [Accessed: 11-Apr-2019].

[27] Real Academia Española © Todos los derechos reservados, "practicar | Definición de practicar - Diccionario de la lengua española - Edición del Tricentenario." [Online]. Available: https://dle.rae.es/?id=TtAtLcR. [Accessed: 08-Feb-2019].

[28] OMG, "Kernel and Language for Software Engineering Methods (Essence)," *2007 4th IEEE Int. Work. Vis. Softw. Underst. Anal.*, no. Versión 1.2, p. 300, 2018.

[29] Capers Jones, "Software Engineering Best Practices," in *The Data Science Handbook*, 2017, pp. 217–227.

[30] B. Kitchenham, "Procedures for Performing Systematic Reviews."

[31] B. M. Losada, "A formalization for mapping discourses from business-based technical documents into controlled language texts for requirements elicitation," 2014.

[32] T. Zimmermann, R. Premraj, N. Bettenburg, S. Just, A. Schröter, and C. Weiss, "What makes a good bug report?," *IEEE Trans. Softw. Eng.*, vol. 36, no. 5, pp. 618–643, 2010.

[33] O. Chaparro, "Improving bug reporting, duplicate detection, and localization," *Proc. -*

2017 IEEE/ACM 39th Int. Conf. Softw. Eng. Companion, ICSE-C 2017, pp. 421–424, 2017.

[34] J. Kanwal and O. Maqbool, "Bug prioritization to facilitate bug report triage," *J. Comput. Sci. Technol.*, vol. 27, no. 2, pp. 397–412, 2012.

[35] C. J. Tessone, C. J. Tessone, and F. Schweitzer, "Categorizing Bugs with Social Networks : A Case Study on Four Open Source Software Communities Categorizing Bugs with Social Networks : A Case," no. August 2017, 2013.

[36] F. Thung, "Automatic prediction of bug fixing effort measured by code churn size," *Proc. 5th Int. Work. Softw. Min. - SoftwareMining 2016*, pp. 18–23, 2016.

[37] G. Jeong, S. Kim, and T. Zimmermann, "Improving Bug Triage with Bug Tossing Graphs," 2009.

[38] I. Service, "A bug ' s life," p. 2013, 2009.

[39] S. Jana, Y. Kang, S. Roth, and B. Ray, "Automatically Detecting Error Handling Bugs using Error Specifications."

[40] S. Kim, K. Pan, and E. J. Whitehead, "Memories of Bug Fixes," 2006.

[41] M. Asaduzzaman, M. C. Bullock, C. K. Roy, and K. A. Schneider, "Bug Introducing Changes : A Study with Android."

[42] J. Echeverría, F. Pérez, J. I. Panach, and C. Cetina, "Evaluating Bug-Fixing in Software Product Lines : an Industrial Case Study."

[43] B.A. Romo. A. Capiluppi. T. Hall, "Filling the Gaps of Development Logs and Bug Issue Data," *Proc. 10th Int. Symp. Open Collab.*, no. Section 2, pp. 2–5, 2014.

[44] M. Böhme, E. O. Soremekun, and A. Zeller, "Where Is the Bug and How Is It Fixed ? An Experiment with Practitioners ∗," pp. 117–128, 2017.

[45] J. Hughes and Q. Ab, "Find More Bugs with QuickCheck !," 2016.

[46] S. Macías *et al.*, "Disponible en: http://www.redalyc.org/articulo.oa?id=56726350013," 2013.

[47] T. G. Ramírez and M. R. López, "El Valor Añadido De Las Buenas Prácticas Con Tic En Los Centros Educativos," *Educ. Knowl. Soc.*, vol. 11, no. 1, pp. 262–282, 2010.

[48] JOSE PABLO SARCO, "¿Como escribir un buen reporte de Fallas/Bugs? – Testing en Español." [Online]. Available: https://josepablosarco.wordpress.com/2010/12/09/¿como-escribir-un-buen-reporte-

de-fallasbugs/. [Accessed: 09-May-2019].

[49] "¿Qué es una Buena Práctica? | Comunidad de Prácticas en APS." [Online]. Available: http://buenaspracticasaps.cl/que-es-una-buena-practica/. [Accessed: 28-Feb-2019].

[50] M. Carre, "Metodología," vol. 38, no. 1, pp. 185–194, 2009.

[51] V. H. Menéndez Domínguez and M. E. Castellanos Bolaños, "SPEM: Software Process Engineering Metamodel," *Rev. Latinoam. Ing. Softw.*, vol. 3, no. 2, p. 92, 2015.

[52] J. Montalvo-garcia, J. B. Quintero, and B. Manrique-losada, "CRISP-DM / SMEs Metodología de Analítica de Datos para PYME sin ánimo de lucro - ESAL Prefacio," pp. 1–29, 2019.

[53] M. Yilmaz and R. V. O'Connor, "Gamifikacijski pristup s integriranom Scrumban metodologijom u poboljšanju razvoja softvera: Turska analiza slučaja," *Teh. Vjesn.*, vol. 23, no. 1, pp. 237–245, 2016.

[54] J. M. Moine, "Metodologías para el descubrimiento de conocimiento en bases de datos : un estudio comparativo," p. 111, 2013.

Cambios realizados en el documento final del trabajo de grado de maestría del estudiante Victor Hugo Mercado Ramos:

APROXIMACIÓN METODOLÓGICA PARA LA CREACIÓN DE INFORMES DE FALLOS Y MEJORES PRÁCTICAS PARA EL PROCESO DE MANTENIMIENTO DE SOFTWARE

Cambio solicitado	**Cambio realizado**	**Ubicación en el documento inicial**	**Ubicación en el documento final**
Se sugieren cambios de redacción.	se aceptan cambios sugeridos.	Página 12	Página 12
Agregar Referencia.	se agrega la referencia solicitada.	Página 15	Página 15
Para decir que es actual, la referencia es de más de 10 años. Es necesario actualizar y referencia bien (falta el año)	Se realízala actualización de la referencia solicitada.	Página 16	Página 16
¿Falta algo? La frase parece cortada	se realizan cambios en la redacción, para ar mayor claridad a párrafo confuso.	Página 20	Página 20
Esta referencia parece ser clave para el planteamiento de la tesis, sin embargo, no está completa y no se encuentra en la red. Es necesario completarla.	Se procede con la corrección solicitada. Se agregan detalles a la referencia.	Página 23	Página 23
Posiblemente es una figura que no aporta nada al trabajo, ni siquiera está citada.	Se agrega texto que hace referencia a la figura 2.	Página 25.	Página 23
Es necesario referenciar	Se agrega referencia solicitada.	Página 28	Página 28
Mejorar redacción	Se hacen cambios en la redacción, según lo sugerido.	Página 30	Página 30
Qué tipo de criterios. ¿De inclusión, de exclusión?	Se define que los criterios eran de inclusión.	Página 30	Página 30

Cambio solicitado	Cambio realizado	Ubicación en el documento inicial	Ubicación en el documento final
Sería conveniente que se argumente mejor la definición de criterios. El objetivo de los criterios es mejorar la calidad de estudios seleccionados, no disminuir la muestra. La disminución de la muestra, al contrario de ser positivo, es un asunto negativo para el establecimiento del estado del arte de un trabajo de investigación.	Se hacen cambios en la redacción aplicando las sugerencias.	Página 30	Página 30
Cuáles fueron los criterios para esto?	Se describen los criterios según se solicita.	Página 31	Página 31
Por ejemplo, estas prácticas, si fueron identificadas, ¿por qué no están en el listado de buenas prácticas?	Si están son las practicas 34,35,36. No se aplican cambios seda respuesta a la pregunta.	Página 45	
Cómo se hizo esta selección? ¿Qué criterios se tuvieron en cuenta? ¿Qué relación tienen estos criterios para seleccionarlas y	Se agregan descripción del proceso de identificación y selección de las practicas.	Página 48	Página 39 y 48

Cambio solicitado	Cambio realizado	Ubicación en el documento inicial	Ubicación en el documento final
listarlas, con el origen?			
Debe ser homogéneo el término, o buenas o mejores.	Se realiza el cambio solicitado, se cambia termino.	Página 49	Página 49
De qué forma se definieron estos criterios. Realmente se definieron como iniciativa propia o bien, se tomaron de algún referente? Sería conveniente citarlo, si es el último caso.	Se describe de donde se obtuvieron los criterios y se realiza la citación previa al listado de los criterios.	Página 49	Página 50
Se recomienda nombrar las fases para identificarlas y darle una idea al lector de qué se trata cada una de ellas. También es bueno hacer claridad en las fases. ¿Fueron definidas por los autores, son parte del método Delphi definido por un autor diferente?	Se agrega descripción según lo solicitado.	Página 51	Página 51
En esta tabla no existe la relación, se supone que esa relación la hacen en la sesión programa,	Se realizan ajustes para una mejor interpretación de la idea.	Página 51	Página 51

Cambio solicitado	Cambio realizado	Ubicación en el documento inicial	Ubicación en el documento final
¿los expertos? ¿Entonces, lo que se muestra en la Tabla no es la relación sino el instrumento base para hacer la sesión con expertos?			
¿Para quién? Esto puede ser subjetivo. Es necesario escribir claramente cómo se definió que una práctica era redundante, similar y que por lo tanto se podría eliminar	Se agregan cambios en la redacción teniendo en cuenta la sugerencia de la evaluadora.	Página 53	Página 53
Es indispensable mostrar los resultados antes de presentar el análisis. Es conveniente presentar un apartado de resultados y uno de discusión o análisis	Se crea un aparte de resultados.	Página 53	Página 53 y 54
¿Dónde se puede ver esto? ¿De qué forma se tabularon los resultados?	Se agregan graficas que ayudan a visualizar esto.	Página 53	Página 54
¿Esto sería una muestra de una evaluación? ¿No de las evaluaciones analizadas? Insisto	Se agrega redacción para aclarar la duda planteada por la evaluadora.	Página 54	Página 55

Cambio solicitado	Cambio realizado	Ubicación en el documento inicial	Ubicación en el documento final
en que es necesario mostrar la forma, técnica o método de análisis y tabulación de resultados. Adicional, hacer claridad sobre las tablas y figuras que se muestran en el desarrollo del trabajo			
Mejorar la calidad de la tabla. Aunque se sugiere citarla como una figura, ya que no es una tabla.		Página 54	Página 56
¿Entonces si son las realizadas?	Si, son las evaluaciones realizadas.	Página 54	Página 56
Sugiero un apartado o capítulo que sea limitaciones o alcance de la propuesta. Esto iría en dicho apartado.	Se crea un apartado en el capítulo de la propuesta para definir el alcance de esta.	Página 56	Página 72
Debe tener una justificación mucho más profunda sobre el uso de esta notación para representar el modelo.	Se hacen cambios de redacción para ajustarse a las sugerencias.	Página 56	Página 58
Es necesario que se maneje un formato único para todo el documento: fuentes y	Se hacen ajustes en el todo el documento.	Página 57	Diferentes paginas del documento.

Cambio solicitado	Cambio realizado	Ubicación en el documento inicial	Ubicación en el documento final
tamaños de letra, en el texto, títulos, subtítulos, figuras y tablas.			
¿Cómo se identificaron estos roles? Qué pasa si la empresa o quien utilice la propuesta (metodología)	Se describe como se identificaron los roles.	Página 57	Página 59
¿Este ciclo de vida es propuesto por alguien o por los autores? Si es propuesto por los autores, ¿cómo se llegó a esa propuesta? ¿De lo contrario, quien lo propone?	Se describe que es propuesto por unos autores y se agrega la referencia.	Página 60	Página 62
Cuáles?	Se definen las metas claves, que usa CMMI	Página 60	Página 63
Dónde?	Se atiende sugerencia.	Página 60	Página 63
Cómo se logró esta alineación?	Se atiende sugerencia.	Página 60	Página 63
Al ser una compañía de diferente naturaleza a TI, sería conveniente caracterizarla mejor y describir mejor sus procesos y el área donde se hizo la validación	Se agrega una descripción de la compañía.	Página 71	Página 74

Cambio solicitado	Cambio realizado	Ubicación en el documento inicial	Ubicación en el documento final
Cuantos?	Se define la cantidad de reportes.	Página 71	Página 74
Quienes?	Se describen los stakeholders	Página 71	Página 74
Cuáles?	Se describen los equipos de ingeniería.	Página 71	Página 74
De la propuesta de esta tesis o se refiere a procesos de la compañía.	De la propuesta se agrega texto.	Página 71	Página 75
se necesita realizar una implementación por fases ¿Por qué?	Se agrega descripción que da claridad a la pregunta	Página 72	Página 75
Dónde?	Se explica que en el capítulo anterior se realiza la propuesta	Página 72	Página 75
Se sugiere explicar.	Se aplica sugerencia.	Página 72	Página 75
No es claro lo que se quiere decir	Se realizan cambios solicitados.	Página 72	Página 75
Qué tiene que ver esta fase con la 6.2.1? ¿Qué tiene que ver con las fases de la Figura 3?	Se mejoro el titulo de la 6.2.1, ya que se realizan y describe las dos implementaciones en las diferentes fases de la propuesta.	Página 72	Página 75
No se representaron en el modelo? En la figura? Qué tiene que ver esta acciones con lo definido en la descripción de la aproximación?	En la presentación se hará claridad que estas actividades no se describen en la propuesta, pero fueron necesarias para dará conocer el contexto de lo que se pretendía realizar.	Página 72	Página 75
Igual que en la evaluación con expertos, es	Se crea un aparte de resultados.	Página 78	Página 81

Cambio solicitado	Cambio realizado	Ubicación en el documento inicial	Ubicación en el documento final
necesario tener un apartado de resultados y uno de análisis o discusión de los mismos. No es suficiente con mostrar un formulario de ejemplo con los resultados. Se debe mejorar.			
Las conclusiones son pobres. Se deben mejorar. Hay varios análisis de resultados de evaluaciones y validaciones que se hicieron que deberían ser reportados en las conclusiones. Deben mejoradas considerablemente.	Se mejoran las conclusiones	Página 81	Página 85
Haría parte, mas que de las conclusiones, de las limitaciones de la propuesta, el alcance o amenazas de la investigación.	Se hace el cambio y se lleva a las limitaciones	Página 81	Página 72
Idem	Se hace el cambio y se lleva a las limitaciones	Página 81	Página 72
Párrafo muy extenso	Se divide el párrafo	Página 12	Página 12
Párrafo extenso. Divida en varios	Se divide el párrafo	Página 16	Página 16

Cambio solicitado	Cambio realizado	Ubicación en el documento inicial	Ubicación en el documento final
párrafos de forma apropiada.			
¿Es la pregunta de la investigación? Si la respuesta es si, escriba el respectivo título aparte.	Se coloca aparte la pregunta de investigación.	Página 17	Página 17
Párrafo muy extenso.	Se divide el párrafo	Página 31 y 32	Página 32 y 33
De acuerdo al formato, debe eliminar al autor	Se deja igual como esta porque en la revisión de la literatura se mencionan los autores.	Página 32	Página 34
algunas confusiones ¿cuáles?	Se describen las confusiones.	Página 79	Página 84