

Framework de buenas prácticas para los requisitos de software de aplicaciones Web aplicables al contexto salvadoreño.



ELABORADO POR:
FLORENCE GUADALUPE VILLA-ALTA GARAY
OSCAR ARMANDO CASTILLO RODRÍGUEZ

ANTIGUO CUSCATLÁN, LA LIBERTAD, EL SALVADOR, CENTROAMÉRICA.
ENERO DE 2018

INDICE

Introducción	4
1. ¿Qué Entenderemos por Framework?	5
2. ¿Qué es un Requisito de Software?	5
3. Tipos de Requisito de Software.....	5
3.1 Requisitos Funcionales	5
3.2 Requisitos no Funcionales	6
4. Actividades de Requisitos de Software	6
4.1 Obtención de Requisitos	7
4.2 Análisis de Requisitos.....	7
4.3 Especificación de Requisitos	8
4.4 Validación de Requisitos	8
4.5 Gestión de Requisitos	9
5. Buenas Prácticas	9
5.1 Buenas Prácticas en la Obtención de Requisitos	10
5.2 Buenas Prácticas en el Análisis de Requisitos.....	13
5.3 Buenas Prácticas en la Especificación de Requisitos	17
5.4 Buenas Prácticas en la Validación de Requisitos	19
5.5 Buenas Prácticas en la Gestión de Requisitos.....	21

6. Bibliografía.....	25
----------------------	----

Introducción

El Framework de Buenas Prácticas para los Requisitos de Software, proporciona las principales buenas prácticas para las actividades de requisitos de software (obtención, análisis, especificación, validación y gestión), las cuales se extrajeron de diversas fuentes bibliográficas tales como: frameworks de desarrollo de software, de gestión de proyectos, de definición y mejora de procesos; así como modelos, estándares y métodos o metodologías relacionadas al desarrollo de software con fecha de publicación superior al año 2000. De las cuales por medio de un proceso de análisis y síntesis se obtuvo un listado de las principales prácticas que conformaron el “Framework” que se presenta en este documento. Para profundizar en los resultados referirse a la tesina de la investigación: *“Framework de buenas prácticas para los requisitos de software de aplicaciones Web aplicables al contexto salvadoreño”*, en el repositorio de la Universidad Don Bosco de El Salvador.

Las buenas prácticas del framework están orientadas a proporcionar un mejor entendimiento para la primera fase del proceso de desarrollo de software (especificación del software) y pueden ser puestas en práctica en el desarrollo de software.

Entre las principales fuentes bibliográficas en la que se basa este framework se encuentran: CMMI DEV 1.3, PMBOK V 5.0, SWEBOK V 3.0, HANDBOOK-NASA REV 2.0 y libros de reconocidos autores que sirvieron como fundamentación teórica en materia de requisitos (entre los que se encuentran Software Engineering Quality Practices - Ronald Kirk Kandt y Software Requirements - Wiegers & Beatty). Estas fuentes bibliográficas pueden encontrarse en la sección de Bibliografía del framework.

1. ¿Qué Entenderemos por Framework?

Se entenderá por **framework de buenas prácticas de requisitos de software** a la estructura base que organiza las **buenas prácticas** según las *actividades de requisitos de software*, basado en los actuales frameworks de desarrollo de software, de gestión de proyectos, de definición y mejora de procesos; así como modelos, estándares y métodos o metodologías relacionadas a requisitos de software. Definición basada en Klaus Pohl (2010, pág. 42).

2. ¿Qué es un Requisito de Software?

“Propiedad que un software desarrollado o adoptado debe tener para **resolver un problema** concreto” (Sánchez, Sicilia, & Rodríguez, 2012, pág. 116). En Sommerville and Peter Sawyer, citado por Wiegers (2013, pág. 6) “los requisitos son una **especificación** de lo que debe implementarse. Son descripciones de cómo debe **comportarse el sistema**, o de una propiedad o atributo del sistema. Puede ser una **limitación** en el proceso de desarrollo del sistema”.

3. Tipos de Requisito de Software

“Los dos tipos de requisitos que tradicionalmente se han identificados atendiendo a criterios de funcionalidad son: **requisitos funcionales** y **requisitos no funcionales**” (Sánchez, Sicilia, & Rodríguez, 2012, pág. 121).

3.1 Requisitos Funcionales

“Especifican los **comportamientos** que el producto de software exhibirá bajo **condiciones específicas**, describe lo que los desarrolladores deben implementar para permitir que los usuarios realicen sus tareas (requisitos de usuario), satisfaciendo así los requisitos del negocio” (Wiegers & Beatty, 2013, pág. 9).

3.2 Requisitos no Funcionales

“Son aquellos que especifican **aspectos técnicos** que debe incluir el sistema, y que pueden clasificarse en **restricciones** y **calidades**” (Sánchez, Sicilia, & Rodríguez, 2012, pág. 121).

Dentro de las restricciones se encuentra cualquier limitación a la que se enfrenten los desarrolladores del sistema, y dentro de las calidades se encuentran todas aquellas características que importan al usuario final o cliente, que son relevantes para establecer el **grado de satisfacción** con el sistema final (Sánchez, Sicilia, & Rodríguez, 2012, págs. 121-122).

4. Actividades de Requisitos de Software

Según la guía SWEBOK el área de los requisitos de software, se ocupa de la **obtención**, **análisis**, **especificación**, y **validación** de los requisitos de software así como la **gestión** de los requisitos durante todo el proceso de desarrollo de software, con la denominación *genérica de actividades de requisitos* (Sánchez, Sicilia, & Rodríguez, 2012, págs. 116-117).

A continuación, se muestra una infografía de las actividades de requisitos de software:

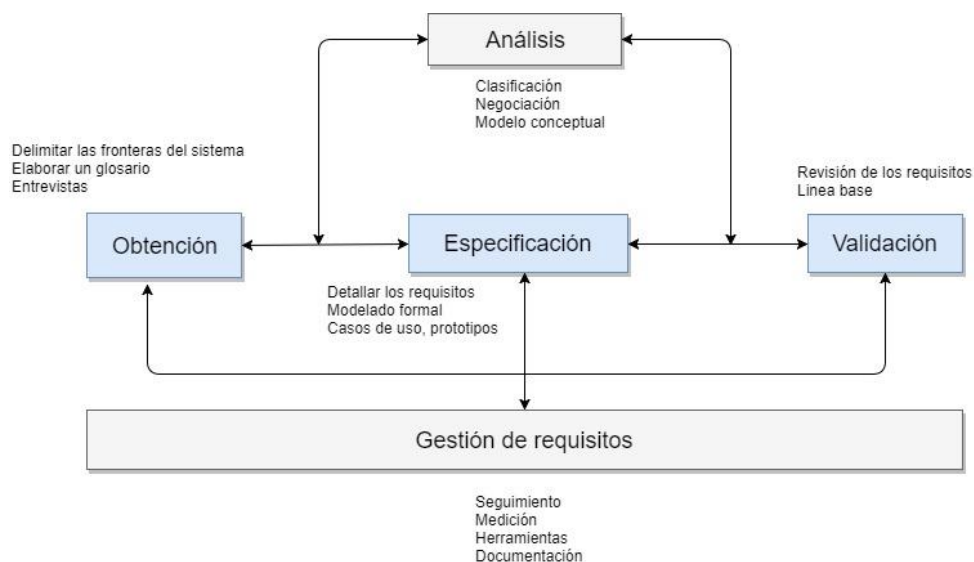


Figura 1. Actividades de requisitos de software. Adaptada de Ingeniería del Software, un enfoque desde la guía del SWEBOK (Sánchez, Sicilia, & Rodríguez, 2012, pág. 127).

4.1 Obtención de Requisitos

La actividad de obtención, es el proceso en el cual **se recolecta la información** necesaria para comenzar a entender el problema que el software debe resolver, e identificar qué es lo que el cliente necesita para poder empezar a definir el rumbo del proyecto a realizar (IEEE, 2014, págs. 5-7). “Consiste en **capturar el propósito y funcionalidades del sistema** desde la perspectiva del usuario” (Sánchez, Sicilia, & Rodríguez, 2012, pág. 117).

Las acciones claves de la obtención de requisitos son (Wiegers & Beatty, 2013, pág. 16):

- Identificar las partes interesadas.
- Comprender las tareas y objetivos de usuario, junto a los objetivos de negocio con los que se alinean estas tareas.
- Aprender sobre el entorno en el que se utilizará el producto.
- Trabajar con los individuos que representan cada clase de usuarios para entender las necesidades de funcionalidad y expectativas de calidad.

4.2 Análisis de Requisitos

“Es el proceso de estudiar las necesidades del usuario para obtener una definición detallada de los requisitos” (Sánchez, Sicilia, & Rodríguez, 2012, pág. 117). Es decir que implica llegar a una comprensión más rica y precisa de cada requisito, representando conjuntos de requisitos de múltiples maneras (Wiegers & Beatty, 2013, pág. 16).

El proceso de analizar los requisitos se utiliza para (IEEE, 2014, pág. 7):

- Detectar y resolver conflictos entre requisitos.
- Descubrir los límites del software y cómo debe interactuar con su entorno organizacional y operativo.
- Elaborar requisitos del sistema para derivar los requisitos de software.

4.3 Especificación de Requisitos

“Es el proceso de **documentar el comportamiento** requerido de un sistema software, a menudo utilizando una **notación de modelado** u otro **lenguaje de especificación**” (Sánchez, Sicilia, & Rodríguez, 2012, pág. 117).

El resultado de la especificación de requisitos es “un conjunto de modelos donde se representan los requisitos desde varias perspectivas (del usuario, estructural, de comportamiento, de flujo de información, entre otros) (Sánchez, Sicilia, & Rodríguez, 2012, págs. 117-118).

Esta actividad de requisitos de software implica la **representación** y **almacenamiento** de los **conocimientos** recopilados de forma persistente y bien organizada (Wiegers & Beatty, 2013, pág. 17).

4.4 Validación de Requisitos

Confirma que se cuenta con la **información correcta** de requisitos que permitirá a los desarrolladores crear una solución que **satisfaga** los objetivos de negocio (Wiegers & Beatty, 2013, pág. 17).

Por medio de la **examinación de los requisitos** se asegura que estos definan el sistema que el cliente y usuarios **desean** (Sánchez, Sicilia, & Rodríguez, 2012, pág. 118). Wiegers & Batty (2013, pág. 17), recomiendan como actividades centrales las siguientes:

- La revisión de los requisitos documentados para corregir cualquier problema antes que el grupo de desarrollo los acepte.
- Desarrollo de pruebas de aceptación.

4.5 Gestión de Requisitos

“La gestión de requisitos incluye todas las **actividades** que mantienen la **integridad, exactitud y la circulación** de los acuerdos de requisitos a lo largo del proyecto” (Wiegers & Beatty, 2013, pág. 458).

Entre las actividades de gestión de requisitos se encuentran (Wiegers & Beatty, 2013, págs. 17-18):

- Definir la **línea base** de los requisitos, una instantánea en el tiempo que representa un conjunto acordado, revisado y aprobado de requisitos funcionales y no funcionales, a menudo para una versión de producto específica o una iteración de desarrollo.
- Evaluar el **impacto** de los cambios de requisitos propuestos e incorporar cambios aprobados en el proyecto de manera **controlada**.
- Mantener los **planes** del proyecto actualizados con los requisitos a medida evolucionan.
- **Negociación** de nuevos compromisos basados en el **impacto estimado** de cambios en los requisitos.
- Definir las **relaciones y dependencias** que existen entre los requisitos.
- **Seguimiento** de los requisitos individuales a sus correspondientes diseños, códigos fuente y pruebas.
- Seguimiento del **estado** de los **requisitos** y cambio de actividad en todo el proyecto.

5. Buenas Prácticas

Una buena practicas es “una actividad o **proceso comprobado** que ha sido utilizado con éxito por **múltiples empresas** y que ha demostrado producir resultados **confiables**” (ISACA, 2017).

A continuación, se describen las buenas prácticas orientadas a cada actividad de requisitos de software:

5.1 Buenas Prácticas en la Obtención de Requisitos

1. Utilizar técnicas para obtención de requisitos

Entre las técnicas para la obtención de requisitos se encuentran las entrevistas, cuestionarios, observación, historia de usuarios, casos de uso, escenarios, prototipos, talleres de facilitación, análisis de documentos, análisis de tormentas de ideas y la examinación de informes de problemas junto a las solicitudes de mejora a los sistemas actuales (PMI, 2013, págs. 114-117).

2. Identificar, verificar e involucrar a las partes interesadas (stakeholders) y administrar los diferentes puntos de vista

Para Wiegers (2013, pág. 48) el identificar los stakeholders evita pasar por alto las necesidades de distintos grupos de un producto de software, permitiendo explorar las tareas que necesitan llevar acabo en el software y el valor que están tratando de lograr. Por su parte el involucrar a los interesados asegura la validación temprana del entendimiento de los requisitos (INTECO, 2008, pág. 18).

SWEBOK (2014, págs. 5-6) determina que el resultado de un software insatisfactorio es debido a que los requisitos se enfocan en un solo grupo de partes interesadas. Por lo que al entregar el software es difícil de usar o subvierte las estructuras culturales o políticas de la organización. Por lo que se necesita identificar, representar y gestionar los “puntos de vista” de los diferentes tipos de partes interesadas.

3. Definir y documentar claramente el alcance y visión del proyecto

SWEBOK (2014, pág. 5) menciona que un elemento crítico de la obtención de requisitos es informar el alcance del proyecto, implicando proporcionar una descripción del software, su propósito y la priorización de entregables.

Esta buena práctica permite tener un mejor entendimiento de los requisitos de software y asegurará que todas las personas involucradas en el proyecto trabajen hacia la misma meta (INTECO, 2008, pág. 18).

En conjunto, la visión y el alcance proporcionan una referencia para evaluar los requisitos propuestos. La visión debe permanecer relativamente estable durante todo el proyecto, pero cada lanzamiento planificado o iteración necesita su propia declaración de alcance (Wiegers & Beatty, 2013, pág. 48).

4. Identificar y evaluar todas las fuentes potenciales de requisitos

Entre las fuentes potenciales que recomienda identificar CMMI (2010, págs. 460-461) y PMBOK (2013, pág. 127) se encuentran: políticas de negocio, procedimientos, estándares, lecciones aprendidas, archivos de proyectos anteriores, requisitos del entorno del negocio, componentes de productos heredados, estatutos reguladores, entre otros.

5. Establecer y mantener acuerdos con los stakeholders (negociación y resolución de conflictos)

Las necesidades y deseos de los stakeholders pueden variar, lo que puede crear conflicto. El objetivo es detectar y exponer los puntos de vista de los actores, resolverse los conflictos (en la medida de lo posible) (Pohl, 2010, pág. 50) .

El proceso de negociación entre las partes del negocio y el equipo del proyecto debe reconocer la legitimidad y la primacía de los insumos empresariales, pero permitir el

descubrimiento de requisitos de software a través de la colaboración y la retroalimentación (Leffingwell, 2011, pág. 107).

6. Comunicar de manera efectiva y utilizar modelos de comunicación

Uno de los principios fundamentales de un buen proceso de obtención de requisitos es el de la comunicación efectiva entre los diversos interesados. Esta comunicación debe continuar durante todo el proceso de “*ciclo de vida de desarrollo de software*” con las diferentes partes interesadas en diferentes momentos. Antes de que comience el desarrollo, los especialistas en requisitos pueden formar el canal para esta comunicación a su vez deben mediar entre el dominio del negocio de los usuarios (y otras partes interesadas) y el mundo técnico del equipo de desarrollo. Para ello un conjunto de modelos en diferentes niveles de abstracción facilita las comunicaciones entre usuarios/partes interesadas y el especialista en requisitos (INTECO, 2008, pág. 5).

La comunicación efectiva es la clave, necesitándose un lenguaje común. Un ejemplo es la “*historia de usuario*” porque proporciona el lenguaje común para crear entendimiento entre el usuario y el equipo técnico (Leffingwell, 2011, pág. 101). Parte importante es comprender que es trabajo del desarrollador hablar el “*idioma del usuario*”, no trabajo del usuario hablar el idioma de los desarrolladores.

5.2 Buenas Prácticas en el Análisis de Requisitos

1. Priorizar los requisitos de software

Para INTECO (2008, pág. 18) esta buena práctica determina aquellos requisitos que se deberían cumplir en la primera versión o producto de software y aquellos que pueden llevarse a cabo en sucesivas versiones.

A su vez hay que considerar que cuanto mayor sea la prioridad, más esencial es el requisito para cumplir los objetivos generales del software. A menudo se clasifican en una escala de punto fijo como: *obligatoria*, *altamente deseable*, *deseable u opcional*, la prioridad tiene que equilibrarse con el costo de desarrollo e implementación (IEEE, 2014, pág. 8).

CMMI (2010, pág. 470) detalla que para conseguir el equilibrio al analizar los requisitos debe considerarse los siguientes factores: coste, calendario, prioridades, componentes reutilizables, entre otros.

2. Analizar los riesgos y viabilidad de los requisitos

Los riesgos comunes en los requisitos incluyen regulaciones e interfaces desconocidas o cambiadas, requisitos faltantes, factibilidad técnica, así como costo e incertidumbre del cronograma. En consecuencia, al identificar los requisitos, los ingenieros deben identificar los requisitos que no tienen estos riesgos, así como identificar los requisitos que podrían tener un impacto significativo. Cuando se ha identificado el riesgo, el personal del proyecto debe realizar estudios de viabilidad y evaluaciones de riesgos (Kirk Kandt, 2006, págs. 119-120).

RUP¹ (2001) recomienda que la identificación de riesgos se realice por medio de escenarios. Una vez identificados los riesgos, el analizar la viabilidad permite comprender los siguientes puntos (Wiegers & Beatty, 2013, pág. 50):

- El encargado de los requisitos debería trabajar con los desarrolladores para evaluar la viabilidad de implementar cada requisito a un costo y rendimiento aceptable en el entorno operativo previsto.
- El analizar la viabilidad permite a los interesados comprender los riesgos asociados con la implementación de cada requisito, incluidos los conflictos y las dependencias con otros requisitos, las dependencias con factores externos y los obstáculos técnicos.
- Los requisitos que son técnicamente inviables o demasiado costosos para implementar pueden quizás simplificarse y aun así contribuir al logro de los objetivos comerciales del proyecto.

3. Construcción de modelos técnicos, simulaciones y prototipos

CMMI (2010, pág. 471) establece que esta buena práctica está orientada para analizar el equilibrio entre las necesidades y las restricciones de las partes interesadas, comúnmente utilizada para reducir el coste del producto y el riesgo del desarrollo del producto.

Wiegers (2013, pág. 50) recomienda que cuando los desarrolladores o usuarios no están seguros acerca de los requisitos, se construya un prototipo de implementación parcial, posible o preliminar para hacer que los conceptos y posibilidades sean más tangibles. Los prototipos permiten a los desarrolladores y usuarios lograr un entendimiento mutuo del problema que se está resolviendo, así como ayudar a validar los requisitos.

¹ RUP: Rational Unified Process.

Entre los modelos que incluye SWEBOK (2014, pág. 8) se encuentran: diagramas de casos de uso, modelos de flujo de datos, modelos de estado, modelos basados en objetivos, interacciones de usuarios, modelos de objetos, modelos de datos, entidad relación, transición de estado, entre otros.

Los factores que influyen en la elección de la notación del modelado son: la naturaleza del problema, la experiencia del ingeniero de software y los requisitos del proceso del cliente (IEEE, 2014, pág. 8).

4. Desarrollar modelado conceptual (contexto)

Un modelo conceptual es una abstracción utilizada por un Ingeniero de Software para comprender un sistema antes de su construcción. La abstracción implica el examen selectivo de ciertos aspectos del mundo real y del sistema deseado, generalmente a un nivel reducido de fidelidad, que aíslan aspectos importantes mientras ignoran otros, aparentemente sin importancia. Por supuesto, lo que es y no es importante en cualquier momento depende de las necesidades de los usuarios de un modelo conceptual (Kirk Kandt, 2006, pág. 116) .

Por su naturaleza, las abstracciones son incompletas e inexactas. Por lo tanto, un buen modelo de un sistema representa los aspectos de un problema crítico para su solución (Kirk Kandt, 2006, pág. 116).

Se utilizan varios esquemas para describir modelos conceptuales, y muchos se basan en el lenguaje de modelado unificado (UML) (Kirk Kandt, 2006, pág. 116).

El proceso de modelado involucra las siguientes actividades (Kirk Kandt, 2006, pág. 116):

- La identificación de los conceptos importantes del mundo real.
- La identificación de las relaciones entre los conceptos.

- La agrupación en unidades de conceptos estrechamente relacionados entre sí. Tales unidades típicamente caracterizan un dominio problemático o un entorno.
- La definición de los atributos de cada concepto, uno para cada tipo de característica única.
- La definición de los comportamientos, o servicios, que cada concepto proporciona a otros conceptos.

Los beneficios de desarrollar un modelo conceptual son varios y se pueden usar para (Kirk Kandt, 2006, pág. 116):

- Verificar las especificaciones del sistema.
- Identificar riesgos de desarrollo.
- Derivar directamente los sistemas de trabajo.
- Formar una base para la reutilización de software.
- Proporcionar una comprensión general del dominio del problema modelado.
- Simular arquitecturas del sistema.

5. Descomposición lógica de los requisitos

SWEBOK (2014, págs. 7-8) expone que esta buena práctica se lleva a cabo por medio de la clasificación, relación y jerarquía de requisitos, algunas series de dimensiones en las que se pueden clasificar los requisitos son: funcional, no funcional, si deriva de uno o más requisitos de alto nivel o de una propiedad emergente, si el requisito está en el producto o en el proceso, prioridad, alcance y volatilidad/estabilidad. Pueden utilizarse otras clasificaciones que se consideren apropiadas y aplicables a la organización.

5.3 Buenas Prácticas en la Especificación de Requisitos

1. Crear artefactos (documentos)

RUP (2001) recomienda documentar los siguientes artefactos: visión y arquitectura del negocio, reglas del negocio, requisitos funcionales y no funcionales, modelar casos de uso y objetos del negocio, atributos de calidad, entre otros.

Wiegers (2013, pág. 52) especifica que entre las reglas de negocio se encuentran las regulaciones gubernamentales, políticas corporativas, estándares y algoritmos computacionales, debiéndose llevar por separado a los requisitos de un proyecto ya que estos tienen una existencia más allá del alcance de un proyecto específico. Algunas reglas dan lugar a requisitos funcionales por lo que se hace necesario definir los enlaces de trazabilidad entre esos requisitos y las reglas correspondientes.

Leffingwell (2011, pág. 40) hace hincapié que la visión del proyecto debe contener los diversos requisitos *no funcionales* tales como: confiabilidad, precisión, rendimiento, calidad, estándares de compatibilidad, entre otros. Los que se consideran que son necesarios para que el sistema cumpla sus objetivos.

2. Documentar los requisitos de forma correcta en un lenguaje natural (descripción no técnica) y al mismo tiempo en lenguaje de requisitos formal utilizando reglas de especificación de requisitos

INTECO (2008, págs. 10-11) recomienda que para escribir los requisitos de forma correcta se debe realizar lo siguiente:

- Eliminar todos los pronombres de la especificación de requisitos, sustituyéndolos por los sujetos.

- Evitar palabras que lleven a la confusión, como “*debería*” ya que da a entender que el requisito es opcional.
- Tener cuidado con adjetivos y adverbios ya que pueden llevar a confusiones.
- Al escribir los requisitos una buena técnica es “*leerlos en alto*”. Y si es posible pedir a alguien que los lea.
- Utilizar una “*convención de nombres*” y definiciones comunes dentro de la organización. De esta forma se aseguran de que todo el proyecto se está usando el mismo vocabulario.

Al documentar los requisitos estos deben ser completos, consistentes y dentro del alcance del proyecto. Un mal uso del lenguaje puede llevar a un mal entendimiento, *horas de trabajo perdido*, una *mala comunicación* entre miembros de equipo y en definitiva una especificación de *poca calidad* (INTECO, 2008, pág. 11).

3. Definir los requisitos teniendo en cuenta la perspectiva del usuario

Para cumplir con esta buena práctica se debe *confirmar* que los involucrados en el negocio tienen el mismo *entendimiento* acerca de los requisitos que la persona que los escribe (INTECO, 2008, pág. 10).

4. Adoptar plantillas de documentos de requisitos

Wiegiers (2013, pág. 51) recomienda adoptar plantillas para documentar la visión, alcance y la especificación de requisitos. Ya que estas proporcionan una estructura consistente para registrar diversos grupos de requisitos e información relacionada. Incluso si no almacena los requisitos en el formulario de documento tradicional, la plantilla le recordará los diversos tipos de información de requisitos para explorar y registrar.

5. La especificación de requisitos no debe ser ambigua

Cada requisito debe ser comprensible por sí mismo. De lo contrario, uno aumenta la probabilidad de que un requisito pueda malinterpretarse porque un lector del requisito puede haber olvidado información especificada en otro requisito. Es decir, los requisitos no deberían tener acoplamiento entre ellos, excepto para representar relaciones de derivación (Kirk Kandt, 2006, pág. 126).

5.4 Buenas Prácticas en la Validación de Requisitos

1. Establecer para cada requisito criterios de validación y aceptación de las partes interesadas

Wiegiers (2013, pág. 53) recomienda pedirles a los usuarios que describan cómo van a determinar si la solución satisface sus necesidades y si es adecuada para su uso. Los criterios de aceptación incluyen un conjunto definido de pruebas de aceptación basadas en los requisitos del usuario, demostrando satisfacción de requisitos no funcionales específicos, seguimiento de defectos y problemas abiertos, infraestructura y capacitación para un despliegue exitoso.

2. Validar la documentación de los requisitos funcionales y técnicos en su contenido, documentación y acuerdos

SWEBOK (2014, pág. 11) expone que los documentos de requisitos pueden estar sujetos a procedimientos de validación y verificación. Los requisitos pueden validarse para garantizar que el Ingeniero de Software haya entendido los requisitos; también es importante verificar que un documento de requisitos cumpla con los estándares de la compañía y que sea comprensible, consistente y completo. En los casos en que las normas o la terminología de la empresa documentada no concuerden con los estándares ampliamente aceptados, se debe acordar una carta entre los dos y adjuntarla al documento.

Los requisitos bien documentados facilitan la detección de cualquier desviación en el alcance acordado para el proyecto (PMI, 2013, pág. 138) .

3. Analizar y validar los requisitos (trazabilidad, supuestos válidos, esenciales y consistentes con el diseño)

CMMI (2010, pág. 466) proporciona las siguientes consideraciones a tener en cuenta para esta buena práctica: la viabilidad, las necesidades de la misión, las restricciones de coste, el tamaño del mercado potencial y la estrategia de la adquisición, dependiendo del contexto del producto. Los atributos de calidad de la arquitectura significativos se identifican en base a la misión y a los factores del negocio. También se establece una definición de la funcionalidad requerida y de los atributos de calidad. Se consideran todos los modos de uso especificados para el producto.

4. Revisar los requisitos y utilizar inspecciones de requisitos formales tanto con usuarios como con proveedores

Para Wiegers (Wiegers & Beatty, 2013) la revisión por pares de los requisitos es particularmente el tipo de revisión rigurosa, siendo una de las prácticas de calidad de software de mayor valor. Recomendamos reunir un pequeño equipo de revisores que representen diferentes perspectivas (como analista, cliente, desarrollador y evaluador) y examine cuidadosamente los requisitos escritos, los modelos de análisis y la información relacionada para detectar defectos. Las revisiones preliminares informales durante el desarrollo de requisitos también son valiosas. Es importante *capacitar a los miembros del equipo* sobre cómo realizar *revisiones efectivas* de los requisitos y adoptar un proceso de revisión para su organización.

5. Utilización de prototipos para validar la interpretación de los requisitos de software (las características clave de las nuevas aplicaciones) y para obtener nuevos requisitos

Los prototipos son comúnmente un medio para validar la interpretación del Ingeniero de Software de los requisitos del software, así como para obtener nuevos requisitos. Al igual que con la obtención, existe una gama de técnicas de prototipos y una serie de puntos en el proceso donde la validación del prototipo puede ser apropiada (IEEE, 2014, pág. 12).

Hay herramientas comerciales disponibles que permiten que un equipo de proyecto simule un sistema propuesto en lugar de o para aumentar las especificaciones de los requisitos escritos. La simulación lleva los prototipos al siguiente nivel, al permitir que el ingeniero de software trabaje con los usuarios para construir rápidamente maquetas ejecutables de un sistema. Los usuarios pueden interactuar con el sistema simulado para validar los requisitos y tomar decisiones de diseño, haciendo que los requisitos cobren vida antes de que se conviertan en el código concreto. La simulación no es un sustituto de la especificación y el análisis de requisitos reflexivos, pero proporciona un suplemento poderoso (Wieggers & Beatty, 2013, pág. 53).

5.5 Buenas Prácticas en la Gestión de Requisitos

1. Gestionar de forma efectiva y eficiente el control de cambios de los requisitos (introducción, cambios y eliminación), utilizando un plan y herramientas de configuración y control de cambios

Para Kirk Kandt (2006, pág. 132) es de gran importancia el controlar como se introducen, cambian y eliminan los requisitos. Ya que el proyecto promedio experimenta un cambio de 25 *por ciento* en los requisitos una vez que se han definido los requisitos para la *primera* versión del sistema, lo que provoca al menos una reducción del cronograma del 25 por ciento. Si la volatilidad de los requisitos no disminuye gradualmente después, el proyecto está fuera de

control y lo más probable es que falle. Además, varios estudios han demostrado que la volatilidad de los requisitos contribuye a la producción ineficiente de software de baja calidad. En consecuencia, los requisitos deben ser gestionados utilizando un proceso de gestión de configuración definido que utilice hojas de control de cambios y herramientas de control de cambios automatizadas que gestionen cada requisito como un elemento de configuración separado. El uso de una herramienta de gestión de configuración permite al personal identificar qué requisitos se han agregado, eliminado o cambiado desde la última línea de base; quien hizo estos cambios; cuando fueron hechos; y la razón para hacerlas.

Wiegiers (2013, pág. 53) recomienda evaluar cada cambio de requisito propuesto para analizar el efecto (impacto) que tendrá en el proyecto. Usar la matriz de trazabilidad de requisitos para identificar los otros requisitos, elementos de diseño, código fuente y pruebas que podría necesitar modificar.

2. Llevar el seguimiento de la vida de un requisito, utilizando la técnica de matriz de trazabilidad de requisitos

Para PMBOK (2013, pág. 117), la matriz de trazabilidad de requisitos es un cuadro que vincula los requisitos del producto desde su origen hasta los entregables que los satisfacen. La implementación de una matriz de trazabilidad de requisitos ayuda a asegurar que cada requisito agrega valor al negocio, al vincularlo con los objetivos del negocio y del proyecto. Proporciona un medio para realizar el seguimiento de los requisitos a lo largo del ciclo de vida del proyecto, lo cual contribuye a asegurar que al final del proyecto se entreguen efectivamente los requisitos aprobados en la documentación de requisitos. Por último, proporciona una estructura para gestionar los cambios relacionados con el alcance del producto.

Wiegiers (2013, pág. 54) expresa que el mantener una matriz de trazabilidad de requisitos es a menudo valioso, y algunas veces necesario, ensamblar un conjunto de enlaces que conecte cada requisito funcional con el diseño y los elementos de código que lo implementan y las pruebas que lo verifican. Tal matriz de trazabilidad de requisitos es útil para confirmar que todos los requisitos se implementan y verifican. También es *útil* durante el *mantenimiento* cuando se debe modificar un requisito. La matriz de trazabilidad de requisitos también puede conectar los requisitos funcionales a los requisitos de nivel superior de los que se derivaron y a otros requisitos relacionados. Llene esta matriz durante el desarrollo, no al final.

3. Establecer la línea base de los requisitos

El utilizar esta buena práctica permite asegurar que cualquier modificación en los requisitos que cambien la línea base se trate como cambios de alcance (INTECO, 2008, pág. 18).

Wiegiers (2013, págs. 53-54) establece que una línea base define un conjunto de requisitos acordados, generalmente para una versión o iteración específica. Una vez que se han establecido los requisitos básicos, los cambios deben realizarse solo a través del proceso de control de cambios del proyecto. Proporcione a cada versión de la especificación de requisitos un identificador único para evitar confusiones entre borradores y líneas de base y entre versiones anteriores y actuales.

4. Supervisar y dar seguimiento al estado de los requisitos de software (especificado, verificado, analizado, entre otros) mediante un proceso definido

El estado de cada requisito debe estar disponible, así como la actividad de cambio para cada uno. Además, se deben recopilar datos resumidos para la cantidad total de cambios que otros han propuesto, aprobado y agregado a los requisitos del software (Kirk Kandt, 2006, págs. 131-132).

Wiegers (2013, pág. 54) recomienda establecer un repositorio con un registro para cada requisito discreto de cualquier tipo que afecte la implementación. Almacene los atributos clave sobre cada requisito, incluido su estado (como propuesto, aprobado, implementado o verificado), para que pueda monitorear el número de requisitos en cada categoría de estado en cualquier momento. El seguimiento del estado de cada requisito a medida que avanza en el desarrollo y las pruebas del sistema proporciona información sobre el estado general del proyecto.

5. Medir el número y la gravedad de los defectos en los requisitos definidos

Cuando las personas ocupadas trabajan en un proyecto complejo, es fácil perder de vista los numerosos problemas que surgen, incluidas las preguntas sobre los requisitos que deben resolverse, las brechas para erradicar y los problemas que surgen de las revisiones de requisitos. Las herramientas de seguimiento de problemas pueden evitar que estos elementos caigan por las grietas. Asignar un único propietario a cada problema. Controle el estado de los problemas de requisitos para determinar el estado general de los requisitos (Wiegers & Beatty, 2013, pág. 54).

El establecer métricas ayudan a identificar la calidad de la actividad de ingeniería de requisitos para que la organización pueda mejorarla (Kirk Kandt, 2006, pág. 132).

6. Formar a los analistas de requisitos

INTECO (2008, pág. 19) expresa que, esta buena práctica asegura que los analistas de requisitos tengan el conocimiento, entre otros aspectos, de cómo escribir y gestionar buenos requisitos.

6. Bibliografía

IEEE. (2014). *SWEBOK-Guide to the Software Engineering Version 3.0* (Segunda ed.). Nueva Jersey: IEEE Computer Society Products and Services.

INTECO. (2008). *Guía Práctiuca de Gestión de requisitos*. Recuperado el 5 de Mayo de 2017

ISACA. (10 de Octubre de 2017). *ISACA trust in, and value from, information system*. Obtenido de <https://www.isaca.org/Pages/Glossary.aspx?tid=2015&char=G>

Kirk Kandt, R. (2006). *Software Engineering Quality Practices* (Primera ed.). New York: Auerbach Publications.

Leffingwell, D. (2011). *Agile Software Requeriments-Lean Requirements Practices For Teams,Programs, and the Enterprise*. Boston: Pearson Education, Inc. Recuperado el 1 de Mayo de 2017

PMI. (2013). *Guía de los fundamentos para dirección de proyectos* (Quinta ed.). Pensilvania, Estados Unidos: Project Management Institute Inc.

Pohl, K. (2010). *Requirements Engineering - Fundamentals, Principles, and Techniques*. Springer-Verlag. Recuperado el 10 de Abril de 2017

Rational Software Corporation. (2 de Octubre de 2001). *Rational Unified Process: Overview*. Obtenido de <http://sce.uhcl.edu/helm/rationalunifiedprocess/>

Sánchez, S. A., Sicilia, M., & Rodríguez, D. (2012). *Ingeniería del software - un enfoque desde la guía SWEBOK* (Primera ed.). Madrid: Alfaomega, Garceta.

SEI. (2010). *CMMI for Development, Version 1.3*. Carnegie Mellon University. Recuperado el 23 de Abril de 2017, de https://resources.sei.cmu.edu/asset_files/TechnicalReport/2010_005_001_15287.pdf

Wiegers, K., & Beatty, J. (2013). *Software Requirements, 3rd Edition*. Redmond: Microsoft Press.