PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

ESCUELA DE POSGRADO

REVISIÓN SISTEMÁTICA DE LA CALIDAD DEL SOFTWARE EN PRÁCTICAS ÁGILES

Tesis para optar el grado de Magíster en Informática con mención en

Ingeniería del Software que presenta

LUIS ALBERTO HUANCA SUNCO

Dirigido por

MAG. LUIS ALBERTO FLORES GARCÍA

Jurados

DR. JOSE ANTONIO POW SANG PORTILLO

MAG. LUIS ALBERTO FLORES GARCÍA

MAG. ENRIQUE ANTONIO MOREY MATOS

SAN MIGUEL, 2015

DEDICATORIA:

A mis queridos padres

Don Rosalio Huanca y Francisca Sunco,

a mis amigos y profesores.

AGRADECIMIENTOS

Mi más profundo y sincero agradecimiento a todas aquellas personas que con su ayuda han colaborado en la realización de la presente tesis de maestría.

A mis profesores y asesor de tesis quienes me mostraron siempre su disposición y dedicaron su valioso tiempo en orientarme.

A mi novia y amigos, por comprender mis momentos de ausencia que he dedicado al extenso trabajo de investigación que ha hecho posible esta tesis, por alentarme siempre a terminarla, por siempre desearme éxitos.

Y finalmente, a cada uno de los integrantes de mi familia que siempre me han ayudado a alcanzar mis metas.

RESUMEN

El desarrollo de software ágil representa un alejamiento importante de los enfoques tradicionales basados en una detallada planificación. Una pregunta fundamental para la mayoría de las organizaciones es saber cuáles prácticas ágiles ayudan, en menor o mayor grado, a mejorar la calidad del producto software.

En este trabajo se muestra el resultado de una revisión sistemática de la literatura que intenta presentar los resultados de estudios empíricos relativos a la evaluación de la calidad en prácticas ágiles publicadas hasta el año 2014. Los estudios encontrados se analizaron siguiendo los requerimientos de calidad definidos en estándares como el ISO/IEC 25010, se catalogaron en cinco grupos: programación en pares, desarrollos guiados por pruebas, extreme programming, scrum y otras prácticas ágiles; finalmente los hallazgos se compararon e interpretaron.

Los resultados de la mayoría de los estudios sugieren que las prácticas ágiles pueden ayudar a mejorar la calidad del producto si son aplicadas correctamente. Los hallazgos significativos de este estudio pueden ser usados como directrices para los interesados en sus propios escenarios.

***Palabras clave**: revisión sistemática de la literatura; desarrollo ágil; calidad; métodos ágiles; prácticas ágiles; ISO/IEC 25010; programación en pares; desarrollos guiados por pruebas; extreme programming; scrum; estudio empírico; experimental.*

ABSTRACT

Agile software development represents a major departure from traditional approaches based on extensive planning. A key question for most organizations is which agile practices helps, to a lesser or greater extent, to improve the software product quality.

This thesis presents the results of a systematic literature review that attempts to present the empirical findings regarding evaluation of quality in agile practices, published up to and including 2014. The studies found were analyzed following quality requirements defined in standards such as ISO/IEC 25010, then categorized into five groups: pair programming, test driven development, extreme programming, scrum and other agile practices; finally the findings were compared and interpreted.

The results of most studies suggest that agile practices can help improve product quality if they are applied correctly. The significant findings of this study could be used as guidelines for those interested in their own scenarios.

Keywords: *systematic literature review; agile development; quality; agile methods; agile practices; ISO/IEC 25010; pair programming; test-driven development; extreme programming; scrum; empirical studies; experimental.*

ÍNDICE GENERAL

AGRADECIMIENTOS ii
RESUMEN iii
ABSTRACT iv
ÍNDICE GENERAL v
ÍNDICE DE FIGURAS vii
ÍNDICE DE TABLAS viii
ACRÓNIMOS Y ABREVIATURAS ix
INTRODUCCIÓN 1
1. Generalidades 2
1.1 Planteamiento del problema 2
1.2 Objetivos 3
1.3 Resultados Esperados 4
1.4 Preguntas de Investigación 4
1.5 Métodos y Procedimientos 4
1.6 Justificación 6
1.7 Alcance y Delimitación 7
2. Marco Conceptual 8
2.1 Metodología de Desarrollo de Software Tradicional 8
2.2 Metodología de Desarrollo de Software Ágil 10
2.2.1. Introducción al modelo ágil y su necesidad 10
2.2.2. La Propuesta Ágil 12
2.2.3. Manifiesto Ágil 13
2.3 Métodos Ágiles 15
2.3.1. Extreme Programming (XP) 15
2.3.2. SCRUM 18
2.3.3. Lean 21
2.3.4. Crystal Clear 22
2.3.5. Feature Driven Development (FDD) 24
2.3.6. Test Driven Development (TDD) 25
2.3.7. Refactorización 26
2.3.8. Programación en pares 27
2.4 Metodologías Ágiles vs Tradicionales 28
2.5 Calidad en el Desarrollo de Software 30
2.5.1. Aseguramiento de la Calidad (*Quality Assurance*) 31
2.5.2. Calidad Interna y Externa 31
2.5.3. Atributos de Calidad 32
2.5.4. Modelos de Calidad 33
2.6 Revisión Sistemática de la Literatura 45

2.6.1. Definición 45
2.6.2. Razones para Adoptar una Revisión Sistemática 45
2.6.3. Características importantes de una Revisión Sistemática 46
2.6.4. El Proceso de la Revisión Sistemática 47
2.6.5. Planificación de la Revisión Sistemática de la Literatura 47
2.6.6. Implementación de la Revisión Sistemática de la Literatura 48
2.6.7. Informe de la Revisión Sistemática de la Literatura 50
2.6.8. Diferencias entre Revisión Sistemática y Revisión Convencional 50
3. Trabajos Relacionados 52
4. Diseño y Realización de la Revisión Sistemática 54
4.1 Diseño de la Revisión Sistemática 54
4.1.1. Protocolo de Revisión 55
4.1.2. Preguntas de Investigación 56
4.1.3. Estrategia de Búsqueda 58
4.1.4. Fuentes de Datos 60
4.1.5. Criterios de búsqueda 60
4.1.6. Fases del proceso de búsqueda 63
4.1.7. Selección de artículos 64
4.1.8. Criterios de Inclusión y Exclusión 64
4.1.9. Evaluación de la calidad 65
4.1.10. Estrategia de Extracción de Datos 66
4.2 Ejecución de la Revisión Sistemática 67
4.2.1. Resultados de la búsqueda y selección de artículos 68
4.2.2. Extracción de Datos 70
4.2.3. Síntesis de los resultados y consideraciones 73
4.3 Resultados 74
4.2.4. Calidad en la Programación en Pares (PP) 78
4.3.1. Calidad en el Desarrollo Guiado por Pruebas (TDD) 79
4.3.2. Calidad en Extreme Programming (XP) 80
4.3.3. Calidad en Scrum 80
4.3.4. Calidad en otras Prácticas Ágiles 81
4.3.5. Limitaciones de esta revisión sistemática 82
5. Conclusiones y Trabajos Futuros 83
5.1 Conclusiones 83
5.2 Trabajos Futuros 85
REFERENCIAS 86
ANEXOS 93
Anexo A: Evaluación de la calidad de los estudios seleccionados. 93
Anexo B: Resumen de los estudios primarios seleccionados. 95

ÍNDICE DE FIGURAS

Figura 2.1: Modelo en cascada (adaptado de [24]) 9
Figura 2.2: Costo del cambio en un desarrollo tradicional (adaptado de [28]) 11
Figura 2.3: Ciclo de vida ágil propuesto (adaptado de [26]) 12
Figura 2.4: Prácticas ágiles en XP y sus relaciones (adaptado de [33]) 18
Figura 2.5: Proceso Scrum (adaptado de [36]) 20
Figura 2.6: Calidad en el Ciclo de vida del software [55] 34
Figura 2.7: Modelo de calidad para calidad interna y externa [55] 34
Figura 2.8: Divisiones familia ISO/IEC 25000 [60] 38
Figura 2.9: Características de calidad del producto ISO/IEC 25010 [60] 40
Figura 2.10: Características de calidad de uso ISO/IEC 25010 [60] 42
Figura 2.11 Procesos del Ciclo de Vida según 12207 [21] 44
Figura 4.1: Procedimiento de búsqueda 55
Figura 4.2: Estrategia de búsqueda 59
Figura 4.3: Cantidad de artículos encontrados por base de datos 69
Figura 4.4: Cantidad de artículos seleccionados por base de datos 69
Figura 4.5: Estudios primarios luego de cada paso de selección 70
Figura 4.6: Cantidad de artículos seleccionados por tipo de estudio 75
Figura 4.7: Cantidad de artículos seleccionados por método ágil utilizado 75
Figura 4.8: Gráfico de tipos de estudios seleccionados por método ágil 76
Figura 4.9: Cantidad de estudios por año de publicación 77

00’

ÍNDICE DE TABLAS

Tabla 2.1: Prácticas de Extreme Programming 17
Tabla 2.2: Siete principios de Lean 22
Tabla 2.3: Propiedades de Crystal 23
Tabla 2.4: Prácticas definidas para FDD 24
Tabla 2.5: Diferencias entre Metodologías Ágiles y Tradicionales [51] 29
Tabla 2.6: Características internas y externas de ISO 9126. (Adaptado de [55]) 36
Tabla 2.7: Diferencias entre revisiones tradicionales y sistemáticas [5] 51
Tabla 4.1: Aplicación del Método PICOC 58
Tabla 4.2: Términos de búsqueda derivados de PICOC 61
Tabla 4.3: Cadenas de búsqueda por base de datos 63
Tabla 4.4: Resultados por bases de datos 68
Tabla 4.5: Listado de los estudios seleccionados 73
Tabla 4.6: Clasificación de tipos de estudios seleccionados por método ágil 76
Tabla 4.7: Estudios separados por categorías ágiles 77

ACRÓNIMOS Y ABREVIATURAS

A lo largo del presente estudio se utilizan los siguientes acrónimos:

ISO:	Organización Internacional de Normalización (International Organization for Standardization).
IEC:	Comisión Electrotécnica Internacional (International Electrotechnical Commission).
XP:	Programación extrema (Extreme programming).
IEEE:	Instituto de Ingenieros Eléctricos y Electrónicos (Institute of Electrical and Electronics Engineers).
RUP:	Proceso Unificado de Rational (Rational Unified Process)
TDD:	Desarrollo guiado por pruebas (Test Driven Development)
FDD:	Desarrollo guiado por características (Feature Driven Development)
TFD:	Desarrollo de Primero escribir las pruebas (Test First Development)
PP:	Programación en Pares (Pair Programming)
SQuaRE:	Requisitos y Evaluación de la Calidad de Software y Sistemas (System and Software Quality Requirements and Evaluation)
ROE:	Rentabilidad Financiera (Return on Equity)
MAG:	Magister.
DR:	Doctor.
Et al.:	(abreviatura latina) y otros o y los demás.

INTRODUCCIÓN

La calidad es una preocupación crítica para lograr el éxito de un producto, sin embargo, como concepto es difícil de definir, entender y medir [19]. El Instituto de Ingenieros Eléctricos y Electrónicos (IEEE) define la calidad como "el grado en que un sistema, componente o proceso cumple con los requerimientos, necesidades y expectativas del cliente/usuario" [20]. Esta definición, como muchas otras, se enfoca en la satisfacción de las necesidades del cliente.

Los métodos ágiles de desarrollo de software son muy populares en la industria. En contraste con los métodos tradicionales, estos sugieren a través de un enfoque iterativo un modo de responder rápidamente a las necesidades cambiantes del cliente, haciendo hincapié en ciclos de lanzamiento cortos y en la integración continua. Con este cambio en el enfoque de desarrollo de software surge la pregunta de si las prácticas ágiles ayudan a mejorar la calidad del producto, y más específicamente cuales de estas lo hacen en mayor o menor grado.

La presente tesis presenta los resultados de una revisión sistemática de la literatura existente sobre estudios empíricos que involucran temas de calidad en desarrollos ágiles, teniendo como objetivo averiguar cuales prácticas aportan calidad al producto software final y en qué grado. El estudio está pensado para ayudar a la comunidad científica que trabaja en el desarrollo ágil, a construir un entendimiento común del estado del arte actual acerca de los enfoques de calidad en las prácticas ágiles.

Está organizado de la siguiente manera: el capítulo 1 presenta las generalidades y el planteamiento del problema a estudiar, el capítulo 2 presenta el marco conceptual necesario para tener una visión general del desarrollo de software ágil y sus principales métodos, principios teóricos de calidad, y los fundamentos de la revisión sistemática como método de investigación; el capítulo 3 detalla los trabajos relacionados en el campo de estudio; el capítulo 4 discute el diseño y ejecución de la revisión sistemática como eje central de esta tesis, y finalmente el capítulo 5 presenta las conclusiones, interpretaciones y trabajos futuros que se desprenden de este estudio.

1. Generalidades

En este capítulo se explican los conceptos necesarios para entender el problema que se desea resolver a través del desarrollo del presente proyecto. Se detalla el contexto general del estudio, así como el resultado esperado, la metodología usada para llegar a este, sus alcances y delimitaciones.

1.1 Planteamiento del problema

Lograr el más alto nivel de calidad del producto final es el objetivo de cualquier organización de desarrollo de software. Estudios sobre las tendencias en ingeniería del software, indican que el mercado actual está caracterizado por la innovación, el rápido desarrollo de aplicaciones y la reducción de la vida de los productos [15]. Ante esto, las organizaciones no buscan un modelo de desarrollo de software que consuma mucho tiempo y esfuerzo, tal como sucedía con los enfoques tradicionales [16].

Las metodologías tradicionales le dan un mayor énfasis a la planificación y control del proyecto, a la especificación precisa de requisitos y al modelado. Son guiados por un modelo de ciclo de vida, tal como el modelo en cascada, para intentar abordar la

mayor cantidad de situaciones de contexto del proyecto en la etapa inicial de planeamiento [17]. Por su naturaleza, su adopción exige un esfuerzo considerable, lo cual es un factor importante principalmente en proyectos pequeños y con requisitos cambiantes.

El enfoque de desarrollo ágil nació para hacer frente a los problemas que se presentaban en las metodologías tradicionales de desarrollo de software, como el desmesurado control, resistencia a cambios y excesiva documentación [1]. Este enfoque ha ganado gran aceptación en el ámbito comercial desde finales de los 90, debido a su gran adaptación a los requisitos volátiles, la colaboración entre desarrolladores y clientes, la temprana entrega del producto, y las mejoras a la calidad del producto final [18]. Por estas razones, cada vez más organizaciones se están moviendo hacia la adopción de metodologías ágiles [16].

Las metodologías ágiles se basan en un conjunto de mejores prácticas pensadas para mejorar el aseguramiento de la calidad y el control. Se puede afirmar que el conjunto de estas mejores prácticas conforman un proceso controlado con calidad incorporada [13]. La calidad del producto se asegura a través de una combinación de estas mejores prácticas, a través de todo el ciclo de vida del software desde la toma de requisitos hasta la entrega final; lo cual muestra una perspectiva diferente a la gestión de calidad tradicional. Con este cambio en el enfoque de desarrollo de software surge la pregunta de en qué medida estas prácticas ágiles ayudan a mejorar la calidad del producto, y más específicamente cuales de estas prácticas lo hacen en mayor o menor grado.

Por tal motivo, se plantea la necesidad de llevar a cabo un estudio que recopile los hallazgos empíricos que involucran temas de calidad en desarrollos ágiles, con el fin de discernir cuales de estas prácticas son más importantes y/o pueden agregar valor al producto final.

1.2 Objetivos

La presente tesis es de naturaleza exploratoria y tiene como objetivo presentar y evaluar estudios empíricos relativos a la calidad en las prácticas ágiles. Los siguientes objetivos requieren ser completados:

- Comprender los conceptos de calidad y metodologías ágiles.
- Resumir el estado del arte de las investigaciones existentes relacionadas a estudios empíricos relativos a la calidad en las prácticas ágiles.
- Identificar las lagunas de investigación existentes en el área de calidad en metodologías ágiles, con el fin de sugerir temas para investigaciones futuras.

1.3 Resultados Esperados

Los resultados esperados del presente estudio son los siguientes:

- Identificar las consideraciones de calidad en desarrollo de software en las metodologías ágiles.
- Identificar las prácticas ágiles más importantes que generan una mejora en la calidad del producto software.

1.4 Preguntas de Investigación

A partir de la problemática del estudio ya definida se proponen las siguientes preguntas de investigación como objetivo a resolver por la revisión sistemática:

- **Pregunta 1**: ¿Cuál es el estado actual del conocimiento acerca de la calidad en las prácticas ágiles?
- **Pregunta 2**: ¿Cuáles son las prácticas más importantes para el logro de calidad en el desarrollo ágil?

1.5 Métodos y Procedimientos

En esta sección se presentan los métodos y procedimientos a seguir para lograr obtener los resultados esperados.

La metodología utilizada en esta investigación es una revisión sistemática de la literatura utilizando las pautas de Kitchenham [1] [2]. Según Kitchenham, una revisión sistemática de la literatura es un medio de identificación, evaluación e interpretación de toda la investigación pertinente disponible acerca de una determinada pregunta de investigación, área temática o fenómeno de interés. Hay muchas razones para llevar a cabo una revisión sistemática de la literatura. Por ejemplo, para resumir las pruebas

existentes en relación con un tratamiento o tecnología, para identificar las lagunas en la investigación actual con el fin de sugerir nuevas líneas de investigación, y proporcionar un marco de trabajo para posicionar nuevas actividades de investigación adecuadas.

Debido al gran número de estudios publicados por diferentes investigadores reportando su experiencia empírica concerniente a la calidad al usar prácticas ágiles, la revisión sistemática se seleccionó como una metodología adecuada para resumir toda esa evidencia existente. Mediante la realización de la revisión sistemática, se pretende recoger las experiencias y estudios reportados por empresas de desarrollo y entornos académicos, que cumplan ciertos criterios de búsqueda. El siguiente paso es categorizar y analizar con el fin de proporcionar una visión general de los resultados de calidad en el uso de las diferentes prácticas ágiles existentes en la actualidad.

El propósito es obtener material de estudio pertinente para responder a las preguntas de investigación definidas en el protocolo de la revisión sistemática. En general, el proceso de revisión sistemática se desarrolló siguiendo los pasos que se detallan a continuación:

- Preparación del protocolo que incluye la definición del proceso.
    - Definición del proceso.
    - Elaboración de las preguntas de investigación.
    - Criterios de inclusión y exclusión.
    - Procedimiento a seguir para el análisis
- Ejecución de la búsqueda preliminar.
    - Definición de la estrategia de búsqueda.
    - Escoger las bases de datos, librerías y otras fuentes de material bibliográfico.
    - Búsqueda.
    - Revisión de los resultados.
    - Síntesis y análisis de los resultados.
    - Depurado y corrección de las cadenas de búsqueda.
- Ejecución de la búsqueda.
    - Selección de bases de datos y cadenas de búsqueda basada en la búsqueda preliminar.
    - Búsqueda.
    - Eliminación de duplicados.

    - Aplicación de los criterios de inclusión y exclusión.
    - Clasificación de los artículos relevantes.
    - Resumen y análisis de los resultados.
- Extracción de Datos.
    - Revisión de los artículos.
    - Clasificación de los artículos.
    - Identificación de estudios primarios.
- Evaluación de la calidad de los estudios.
- Análisis de los resultados.
- Desarrollo de las conclusiones.
- Reporte final y estudios futuros.

El proceso de revisión se inicia con el desarrollo de un protocolo, que es un plan detallado para llevar a cabo la revisión, que incluye las preguntas de investigación, estrategia definida de búsqueda y selección, y criterios de calidad para los estudios primarios. El protocolo se presenta en la sección 4.1.1. En esta revisión sistemática la literatura gris no se tomó en cuenta.

1.6 Justificación

Con la cada vez mayor cantidad de organizaciones que adoptan el uso de metodologías ágiles es importante conocer cuáles de estas prácticas ayudan a mejorar la calidad del producto software.

Según búsquedas bibliográficas preliminares no se encontraron estudios actuales detallados que relacionen y analicen el impacto que tiene el uso de las diferentes prácticas ágiles en la mejora de la calidad del desarrollo del software, y como estas prácticas generan valor en su utilización.

Esto origina la necesidad de realizar una revisión sistemática de la literatura existente con el objetivo de evaluar, de acuerdo a estándares de calidad establecidos, sintetizar y presentar los hallazgos empíricos que involucran temas de calidad en desarrollos ágiles. Este estudio busca establecer cuales prácticas ágiles logran entregar un producto software con mayor calidad, rapidez y mejor aceptación del cliente basándose en estudios empíricos previamente realizados y publicados. A su vez, trata de identificar lagunas de investigación en dicho campo de estudio.

1.7 Alcance y Delimitación

El presente trabajo plantea una revisión sistemática de la literatura existente acerca de la calidad del software en prácticas ágiles. Además presenta un marco teórico referente a los desarrollos ágiles, los conceptos y estándares usados de calidad, y los fundamentos de la revisión sistemática como metodología. Finalmente se proponen conclusiones y trabajos futuros.

Queda fuera del alcance de la aplicación cualquier implementación o validación experimental cuantitativa derivada del presente estudio. El mismo es de naturaleza exploratoria bibliográfica.

2. Marco Conceptual

El objetivo de este capítulo es brindar un marco teórico que permita comprender a cabalidad el presente estudio. Se centrará básicamente en explicar las distintas metodologías ágiles y sus relaciones con las metodologías tradicionales, los conceptos de calidad y estándares usados, así como todo lo referente a la elaboración de la revisión sistemática de la literatura y sus fundamentos. Adicionalmente se añaden conceptos y definiciones que pueden ayudar a entender mejor la presente tesis.

2.1 Metodología de Desarrollo de Software Tradicional

En la década de 1980 y a inicios de la siguiente, había una visión muy difundida de que la forma más adecuada de lograr un mejor software era mediante una cuidadosa planeación del proyecto, aseguramiento de calidad formalizada, el uso de métodos de análisis y el diseño apoyado por herramientas CASE, así como procesos de desarrollo de software rigurosos y controlados. Esta percepción proviene de la comunidad de ingeniería de software, responsable del desarrollo de grandes sistemas de software de larga duración como los sistemas: aeroespaciales y gubernamentales [23].

Para este tipo de desarrollo de sistemas se requiere grandes equipos de desarrolladores, a menudo geográficamente dispersos que laboran por largos periodos de tiempo. Un ejemplo de este tipo de software es el sistema de control de una aeronave moderna, que puede tardar hasta 10 años desde la especificación inicial hasta su implementación [23].

Esta metodología tradicional de desarrollo de software, a menudo representada por el enfoque de desarrollo en cascada o espiral, utiliza un proceso secuencial basado en un detallado planeamiento, que consiste típicamente de una definición de requerimientos, especificación, diseño, construcción, pruebas unitarias, pruebas de integración y pruebas del sistema [24]. Estas metodologías ponen énfasis en la planificación total, desde el principio hasta el final del proyecto, por lo cual son adecuados para problemas bien comprendidos, que no cambian y con salidas bien definidas desde el principio [11].

Algunas de las metodologías tradicionales más conocidas son:

- Modelo en Cascada
- *Rational Unified Process* (RUP)
- Modelo-V
- Modelo Espiral

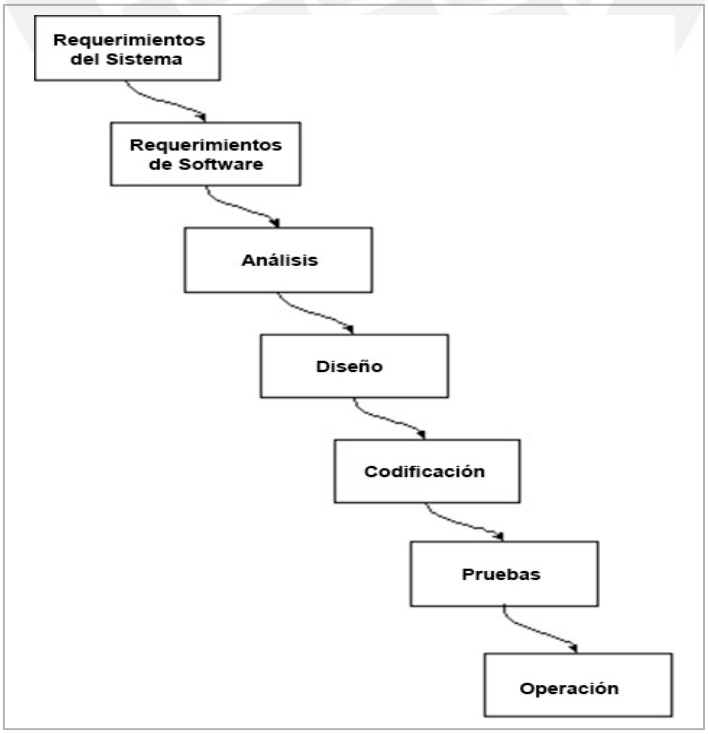

Figura 2.1: Modelo en cascada (adaptado de [24])

La Figura 2.1 muestra el modelo en cascada, el cual es usado como referente común en las prácticas de desarrollo de software tradicional.

Estos enfoques, basados en la planificación previa, incluyen costos operativos significativos en la planeación, el diseño y la documentación del sistema, así como una gran cantidad de personal y consumo de tiempo.

2.2 Metodología de Desarrollo de Software Ágil

Los métodos ágiles de desarrollo de software surgieron a mediados de la década de 1990, como una reacción contra las metodologías tradicionales consideradas excesivamente pesadas y rígidas, por su carácter normativo y fuerte dependencia en la planificación [25]. Las metodologías de desarrollo ágil en cambio son iterativas, se centran en el trabajo en equipo, la colaboración entre el cliente y el desarrollador, la retroalimentación de los clientes en todo el ciclo de vida del proyecto y apoyan la entrega temprana del producto [29].

En la actualidad las metodologías ágiles han adquirido un gran auge dentro de la industria del software, lo cual hace que cada vez más y más organizaciones estén cambiando hacia la adopción de estas metodologías [16].

2.2.1. Introducción al modelo ágil y su necesidad

En las metodologías tradicionales, el desarrollo de software se da a través de procesos de desarrollo un tanto rígidos, con gran consumo de recursos y con un pobre manejo del cambio. Un reciente estudio, realizado sobre la tendencia en ingeniería del software, muestra que el mercado actual está caracterizado por el rápido desarrollo de aplicaciones y la reducción de la vida de los productos [15], lo cual precisa replantear los cimientos sobre los que se sustenta el desarrollo de software tradicional.

El proceso de desarrollo tradicional presentaba diversos problemas, como la excesiva documentación no utilizada, que consume un porcentaje elevado del tiempo de desarrollo de un producto; la lentitud del proceso de desarrollo cuando se trata de sistemas complejos; y la más importante, el cómo afronta el cambio. En este aspecto los métodos tradicionales son presuntuosos, al asumir

que todos los aspectos de un sistema pueden ser definidos antes de empezar su elaboración; asumir que los requerimientos definidos son estables, que la tecnología no cambiará durante un periodo de tiempo o que no habrá sorpresas o desviaciones de su plan original [28].

Estas metodologías tratan de evitar a toda costa que se produzcan cambios en el conjunto inicial de requisitos, puesto que a medida que avanza el proyecto resulta más costoso solucionar los errores detectados o introducir modificaciones [27]. La Figura 2.2 muestra cómo se incrementa el costo de realizar cambios mientras avanzan las etapas en las metodologías tradicionales.

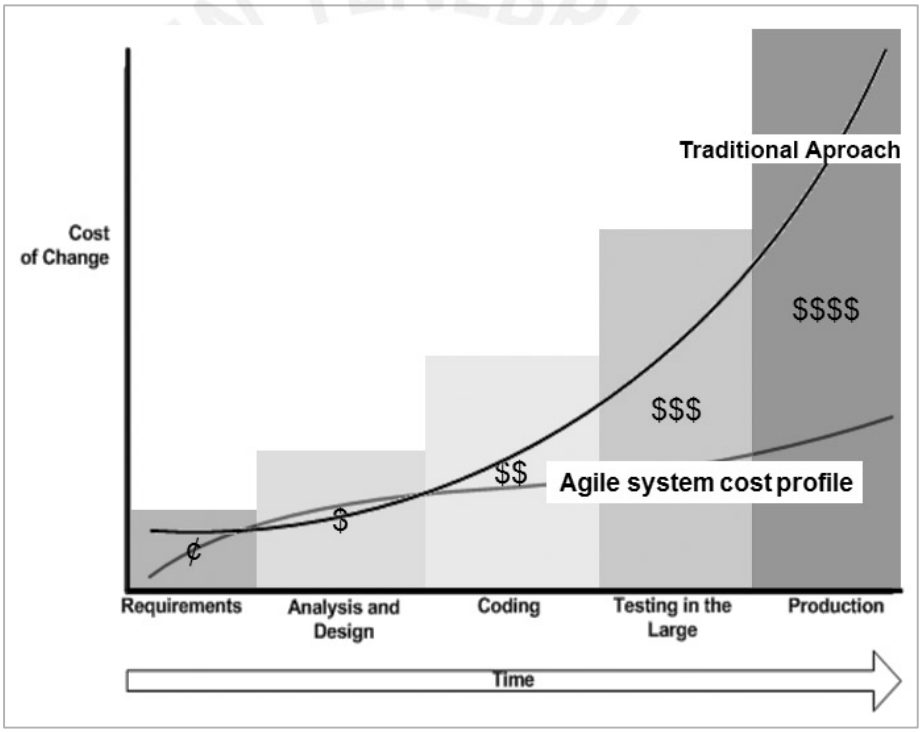

Figura 2.2: Costo del cambio en un desarrollo tradicional (adaptado de [28])

La realidad es que durante la vida de un proyecto de software muchos factores cambian, los requerimientos cambian, las necesidades cambian, las prioridades cambian, la tecnología cambia. Entonces surgen dos preguntas ¿Cómo lidiar con el cambio? y ¿Cómo minimizar el impacto del cambio y los costos? Para intentar solucionar estas preguntas aparece la propuesta ágil.

2.2.2. La Propuesta Ágil

El enfoque de desarrollo ágil nació para hacer frente a los problemas que se presentaban en las metodologías de desarrollo de software tradicionales. A diferencia de estas, las metodologías ágiles hacen frente a la impredecibilidad apoyándose en las personas y su creatividad en lugar de los procesos [30]. Se caracterizan por ciclos iterativos de desarrollos cortos, impulsados por las características del producto, períodos de reflexión e introspección, toma de decisiones colaborativas, incorporación rápida de retroalimentación, y la integración continua de los cambios en el código fuente del sistema en desarrollo [17].

Las metodologías ágiles se basan en el desarrollo iterativo e incremental, donde los requisitos y las soluciones evolucionan a lo largo del proceso de desarrollo. La Figura 2.3 muestra una propuesta de ciclo de vida ágil, donde se puede apreciar su naturaleza iterativa y la participación del equipo y clientes.

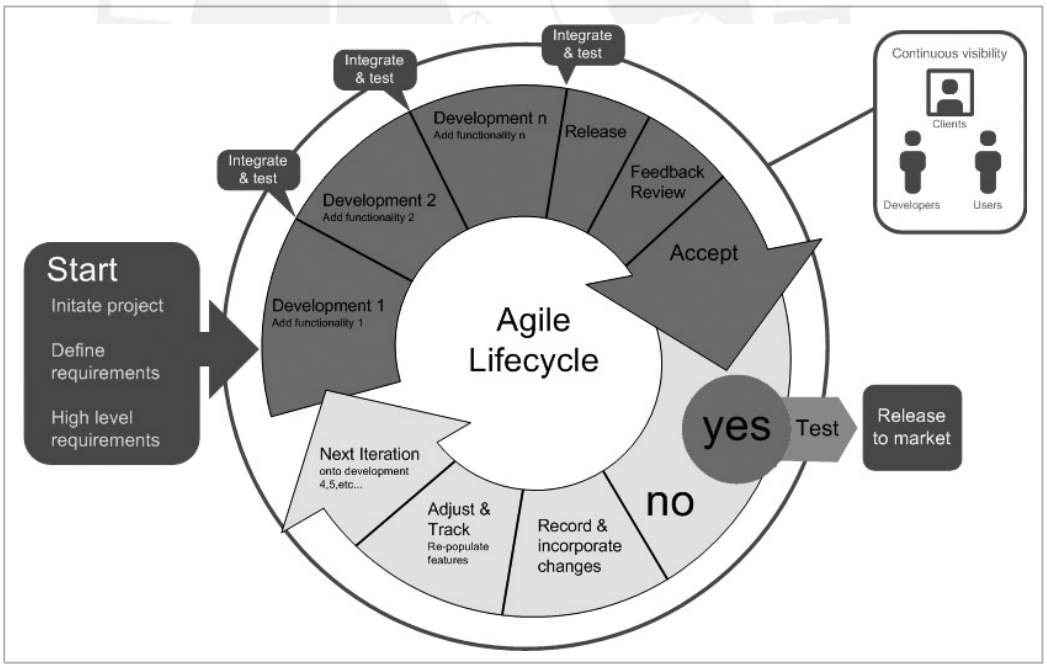

Figura 2.3: Ciclo de vida ágil propuesto (adaptado de [26])

En contraposición a las metodologías convencionales, las metodologías ágiles aparecen como alternativa atractiva para adaptarse al entorno cuando los requisitos son emergentes y cambian rápidamente [17]. De este modo, presentan diversas ventajas tales como:

- Capacidad de respuesta rápida a los cambios a lo largo de todo el desarrollo.
- Entrega continúa y en plazos breves de software funcional, lo cual permite que el cliente verifique si el producto satisface sus necesidades, favoreciendo la disminución de riesgos y dificultades.
- Trabajo conjunto entre el cliente y el equipo de desarrollo, lo cual permite mitigar errores de captura de requerimientos y el exceso de documentación improductiva.
- Importancia de la simplicidad, eliminando el sobre-trabajo innecesario que no aporta valor al negocio.
- Mejora continua de los procesos y el equipo de desarrollo, aumentando la productividad y mejorando la calidad.

El objetivo del desarrollo ágil es la creación de software que funcione, no cumplir con un proceso de desarrollo predefinido. Estas metodologías ágiles se sustentan y fundamentan en el Manifiesto Ágil.

2.2.3. Manifiesto Ágil

El desarrollo ágil o más aun el concepto de agilidad, es una filosofía de desarrollo de software. Tomando como base varias definiciones contemporáneas, Qumer y Henderson-Sellers ofrecieron la siguiente definición de agilidad [125]:

> *"La agilidad es un comportamiento persistente o habilidad, de entidad sensible, que presenta flexibilidad para adaptarse a cambios, esperados o inesperados, rápidamente; persigue la duración más corta en tiempo; usa instrumentos económicos, simples y de calidad en un ambiente dinámico; y utiliza los conocimientos y experiencia previos para aprender tanto del entorno interno como del externo."*

El punto de partida de esta filosofía surge de las ideas definidas en el Manifiesto Ágil, el cual resume los valores y principios fundamentales a considerar para el desarrollo de software de manera ágil.

En 2001, un grupo de diecisiete consultores y profesionales del software de mentalidad independiente se reunieron y firmaron el *Agile Software Development Manifiesto*. El manifiesto propone cuatro valores fundamentales que se complementan en doce principios. Según la Alianza Ágil, estos cuatro valores primordiales son [26]:

> **Individuos e interacciones** sobre procesos y herramientas.
> **Software funcionando** sobre documentación extensiva.
> **Colaboración con el cliente** sobre negociación contractual.
> **Respuesta ante el cambio** sobre seguir un plan.

Para cumplir los valores propuestos se siguen estos doce principios [26]:

1. Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor.
2. Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.
3. Entregamos software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.
4. Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto.
5. Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo.
6. El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.
7. El software funcionando es la medida principal de progreso.
8. Los procesos ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma indefinida.
9. La atención continua a la excelencia técnica y al buen diseño mejora la agilidad.
10. La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.

11. Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados.
12. A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia.

2.3 Métodos Ágiles

Aunque el movimiento ágil está sustentado por valores y principios para el desarrollo de software, estos se reflejan en metodologías que tienen asociadas un conjunto de prácticas, en muchos casos comunes, que buscan la agilidad en el desarrollo. En esta sección se analizan algunas de las metodologías más relevantes.

2.3.1. Extreme Programming (XP)

Extreme Programming o XP es uno de los primeros planteamientos ágiles, propuesto ante el problema de los largos ciclos de desarrollo en las metodologías de desarrollo de software tradicionales. La idea se desarrolló por Kent Beck y Ward Cunningham a finales de la década de los 90's [32].

XP es exitoso porque enfatiza la satisfacción del cliente. En vez de entregar todo lo que el cliente pueda desear en una fecha lejana en el futuro, este proceso está diseñado para entregar el software que los clientes necesitan en el momento en que lo necesitan, es decir un poco a la vez. Además, XP alienta a los desarrolladores a responder a los requerimientos cambiantes de los clientes, aún en fases tardías del ciclo de vida del desarrollo; lo cual es difícil de manejar por el desarrollo de software tradicional [34]. La metodología también enfatiza el trabajo en equipo, tanto gerentes como clientes y desarrolladores son partes del mismo equipo con un fin en común.

Después de varios ensayos y ajustes de sus prácticas, XP fue propuesto como una disciplina para ayudar a las personas a desarrollar software de alta calidad, siguiendo valores y prácticas clave [33]. Los cinco valores en los que se basa XP son los siguientes [34]:

- **Simplicidad**: Hacer lo que es necesario y lo que se ha solicitado, pero no más. Esto maximiza el valor creado para la inversión realizada hasta dicho momento.
- **Comunicación**: Todos son parte del equipo y la comunicación es cara a cara todos los días. Se debe mantener informados a todos los involucrados en el proyecto acerca de todo lo referente al mismo.
- **Retroalimentación**: Tomar cada iteración con compromiso, entregando software funcional. Las entregas se hacen con prontitud y frecuencia, luego se escuchan las observaciones y se hacen los cambios necesarios.
- **Respeto**: Todo el mundo da y siente el respeto que merece como miembro valioso del equipo. Todos aportan valor incluso si es simplemente entusiasmo.
- **Coraje**: Decir la verdad sobre el progreso y las estimaciones. No hay nada a que temer porque nadie trabaja solo. Hay que adaptarse a los cambios cada vez que se producen.

Estos valores se concretan a través de un conjunto de 12 prácticas individuales definidas por Beck [33], dichas prácticas se resumen en la Tabla 2.1:

Prácticas		Descripción
1	**El Juego de la Planificación (*Planning game*)**	La planificación de las características de los próximos lanzamientos se realiza haciendo que los desarrolladores estimen el esfuerzo necesario para la implementación de las historias de los usuarios. En base a esto, los clientes deciden el alcance necesario y fecha de lanzamiento.
2	**Versiones pequeñas**	Lanzar versiones del software constantemente al cliente, a modo de versiones incrementales pequeñas. La nueva versión de los productos se libera por lo menos de modo mensual, o incluso puede ser diariamente.
3	**La Metáfora**	La metáfora es una simple historia que es compartida entre los clientes y los desarrolladores acerca de cómo funciona el sistema.
4	**Diseño simple**	Diseñar la solución funcional más simple para el momento y evolucionarla constantemente para añadirle la flexibilidad necesaria. La complejidad inútil y el código no esencial deben ser eliminados.
5	**Pruebas**	Mediante el uso de desarrollo guiado por pruebas (*TDD: test*

		driven development), los desarrolladores escriben pruebas unitarias antes que el código de producción. Estas pruebas deben pasarse correctamente para que el desarrollo se mantenga funcionando en todo momento.
6	**Refactorización**	Reestructurar el sistema sin cambiar su comportamiento exterior, quitando complejidad duplicada en el código, mejorando la comunicación, simplificando, y añadiendo flexibilidad.
7	**Programación en pares**	Dos programadores escriben todo el código de producción juntos en un solo equipo. Uno escribe el código, y al mismo tiempo, el otro revisa el código para su corrección y comprensibilidad.
8	**Propiedad colectiva**	Todos los desarrolladores son dueños del código fuente. Por lo tanto, pueden cambiar cualquier parte del código en el sistema en cualquier momento.
9	**Integración continua**	Construir e integrar el sistema varias veces al día, cada vez que se haya completado una tarea.
10	**Semana de 40 horas**	Trabajar no más de 40 horas a la semana como una regla. No trabajar nunca horas extras durante dos semanas consecutivas.
11	**Cliente In situ (On-site customer)**	Incluir a un cliente real que puede trabajar con el equipo de desarrollo y que esté disponible a tiempo completo para ayudar a definir el sistema y responder preguntas.
12	**Estándares de codificación**	Los desarrolladores escriben todo el código de acuerdo a normas y estándares.

Tabla 2.1: Prácticas de Extreme Programming

La Figura 2.4 muestra cómo se relacionan las prácticas enumeradas, estas se apoyan entre sí, por lo tanto se debe tener cuidado al modificar cualquiera de ellas, o cuando se decide no incluir una o más de estas prácticas en un proyecto.

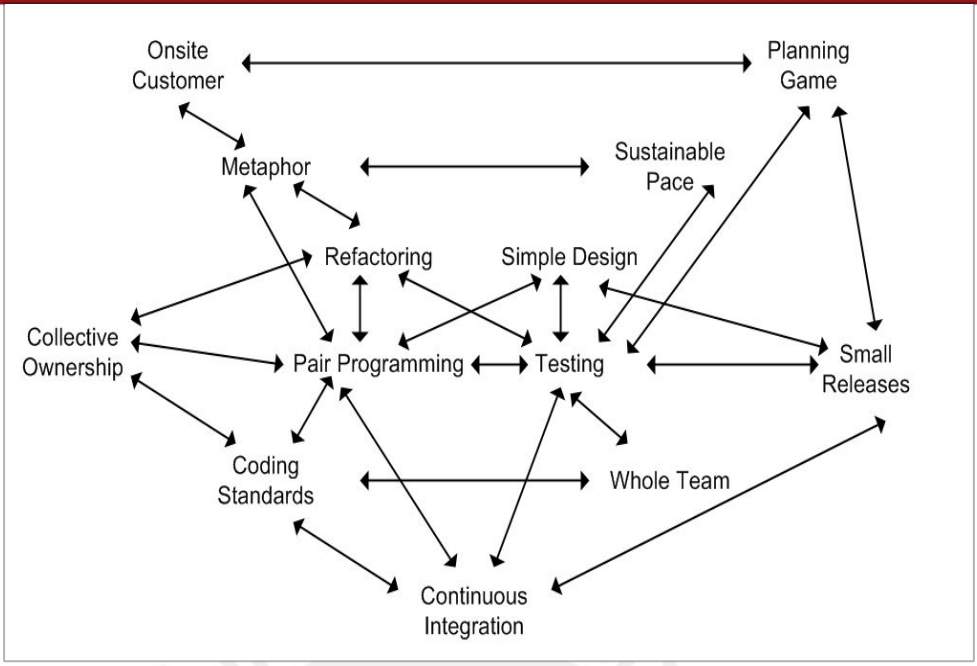

Figura 2.4: Prácticas ágiles en XP y sus relaciones (adaptado de [33])

2.3.2. SCRUM

Scrum es un marco de trabajo sencillo y adaptable utilizado para la gestión de proyectos de desarrollo de software. Este ayuda en la organización de equipos y a tener el trabajo hecho de manera productiva y con mayor calidad [37]. Scrum fue presentado por Jeff Sutherland en *Easel Corporation* en el año 1993 y luego fue formalizado para la industria del software por Ken Schwaber en 1995. El término "Scrum" fue derivado originalmente de la formación Scrum de los jugadores de Rugby. Este término fue identificado por Takeuchi y Nonaka en su trabajo cuando examinaban las mejores prácticas empresariales para la construcción de nuevos productos en el sector automotriz de Japón en 1986. Este fue utilizado como fundamento para la formación de equipos [35] [37].

Aunque Scrum fue originalmente propuesto para la gestión de proyectos de desarrollo de productos, se ha utilizado en gran medida para la gestión de proyectos de software, considerado como un enfoque de desarrollo de software ágil. Mediante el uso de Scrum, el equipo puede entregar software al cliente de modo mucho más rápido, añadiendo energía, concentración y transparencia a la planificación y ejecución del proyecto. Los siguientes factores pueden lograrse mediante la aplicación de Scrum [37]:

- Objetivos individuales alineados con los objetivos corporativos.
- Una cultura impulsada por el desempeño.
- Creación de valor para los accionistas.
- Comunicación estable y consistente rendimiento.

El marco de trabajo de Scrum incluye: tres roles, tres artefactos y cuatro ceremonias. Está diseñado para ofrecer software funcional en Sprints o unos 30 días de iteraciones [36] [37].

Todas las responsabilidades de gestión del proyecto se dividen entre estos tres roles de Scrum [35] [36]:

- **Product Owner**: es el responsable de representar los intereses de todos los interesados en el proyecto (*stakeholders*), mediante la definición de las características del producto documentadas como requisitos. Es responsable de la rentabilidad del producto (ROI), así como de la creación de plan de lanzamiento. La lista de requisitos se llama *Product Backlog*, y es utilizada por el Product Owner para asegurar que las funcionalidades se desarrollen en forma priorizada, de acuerdo con el valor de mercado.

- **Scrum Master**: es un facilitador del proyecto de Scrum. Es responsable del éxito del proyecto, asegurándose de que el equipo siempre este completamente funcional, productivo y que el proceso Scrum se sigua correctamente.

- **El equipo**: es auto organizado y multifuncional, conformado por siete más/menos dos miembros (entre 5 y 9 miembros). Ellos son responsables de desarrollar la funcionalidad de cada iteración y cada proyecto como un todo. Seleccionan la meta del Sprint y especifican los resultados a alcanzar.

Además de los roles, Scrum plantea tres nuevos artefactos, que se utilizan durante todo el proceso Scrum de la siguiente manera [36] [37]:

- **Product backlog**: Es una lista de los requisitos del sistema que están siendo desarrollados por el equipo, mientras que su contenido y

priorización son responsabilidad del *Product Owner*. Existe siempre que el producto existe y crece a medida que el producto crece.

- **Sprint backlog**: Describe el trabajo que un equipo selecciona del *Product backlog* y que se aplicará en el Sprint. Se trata de una imagen en tiempo real del trabajo que el equipo planea realizar durante el sprint.
- **Burndown chart**: Este gráfico se utiliza como una herramienta para ayudar al equipo de desarrollo a completar con éxito un Sprint a tiempo. Muestra las tareas que quedan por hacer en el *Sprint backlog*.

Finalmente, Scrum se conforma de cuatro ceremonias, que son: la planificación del Sprint, revisión del Sprint, reunión diaria Scrum, y la reunión de retrospectiva [36]. La Figura 2.5 muestra estas ceremonias y sus relaciones dentro de un proceso Scrum.

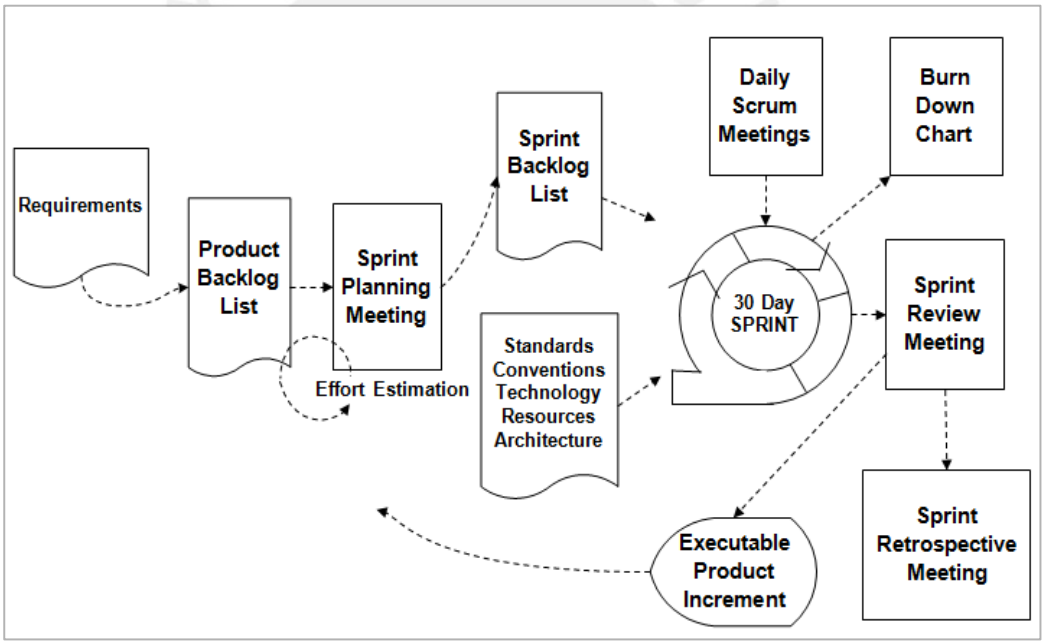

Figura 2.5: Proceso Scrum (adaptado de [36])

Todo el trabajo se realiza en un Sprint, que es una iteración de 30 días. Cada Sprint comienza con la planificación del Sprint. Esta es la actividad donde el *Product owner* trabaja en conjunto con el equipo en decidir las tareas que se van a hacer en el próximo Sprint. Las tareas son priorizadas y seleccionadas del *Product backlog*. Las tareas que se decidieron implementar se ponen en el *Sprint backlog* donde aparecerán a medida que el Sprint evolucione [37].

La segunda ceremonia, que tiene lugar todos los días, es la reunión diaria Scrum. Esta permite que los miembros del equipo se reúnan durante unos 15 minutos para que puedan sincronizar sus trabajos y programar las futuras reuniones si es necesario [36]. En esta reunión, cada miembro responde a tres preguntas sobre el proyecto: ¿Qué han hecho desde la última reunión diaria Scrum?, ¿Qué va a hacer hasta la próxima reunión diaria Scrum? y ¿Qué obstáculos se encontraron durante su trabajo?

La tercera ceremonia es la reunión de revisión del Sprint. Esta reunión dura alrededor de cuatro horas en la cual el equipo presenta al *Product owner,* y otros interesados, las funcionalidades desarrolladas durante el Sprint [36] [37]. Después de esta reunión, el *Scrum Master* lleva a cabo la reunión retrospectiva Sprint [36]. Esta reunión es para que el equipo reflexione y mejore su proceso de desarrollo, para así ser más eficiente en el próximo Sprint.

2.3.3. Lean

Dado el éxito del concepto de Lean en el sistema de producción Toyota, que llevó a la marca a producir automóviles de alta calidad con el más bajo costo y en el menor tiempo posible, los principios de Lean fueron llevados a otras áreas, incluyendo el área del desarrollo de software. En particular, obtuvo significativamente mayor atención tras la publicación del libro *Lean Software Development* por Mary y Tom Poppendieck [38].

Lean está estrechamente relacionado con los enfoques de desarrollo de software ágil. Lean en sí es una forma de pensar acerca de cómo hacer entregas al cliente con mayor rapidez mediante la búsqueda y eliminación de desperdicios (impedimentos para la calidad y la productividad) [41].

Las ideas clave detrás de Lean son poner todos los esfuerzos de desarrollo en las actividades que añaden valor desde el punto de vista de los clientes y analizar e identificar de manera sistemática los desperdicios en el proceso software y retirarlos [38]. Los siete principios de Lean se describen en la Tabla 2.2 [39] [40].

Principios	Descripción
Eliminar desperdicios	Elimine todo aquello en el proceso de desarrollo de software que no genera valor para el cliente, por ejemplo trabajos parcialmente realizados, características adicionales, procesos adicionales, retrasos, defectos, etc.
Embeber a la calidad	La calidad del software debe ser incorporada tan pronto como sea posible y no tardíamente mediante la solución de defectos que las pruebas finales encuentran.
Ampliar el aprendizaje	Los procesos y prácticas en empresas deben apoyar el aprendizaje. Aprender allí comprende obtener una mejor comprensión de las necesidades del cliente, buenas estrategias de pruebas, etc.
Postergar el compromiso	Un compromiso debe retrasarse lo más tarde posible cuando se van a tomar decisiones irreversibles. Por ejemplo, una decisión arquitectónica compleja puede requerir un poco de experimentación y por lo tanto no debe tomarse rápidamente.
Entregar rápido	Crear ciclos con tiempos cortos de desarrollo del producto, reduciendo al mínimo el tiempo desde la recepción de una solicitud de funcionalidad hasta su entrega.
Respetamos a las personas	Poppendieck dio tres principios que se utilizaron en el contexto del caso de Toyota. Liderazgo empresarial; las personas con gerentes que confiaron en ellos y los respetaron son más propensos a convertirse en buenos líderes. Fuerza de trabajo experta; las empresas deben asegurar el conocimiento técnico necesario dentro de sus equipos para el logro de una tarea. Planeamiento y control basado en la responsabilidad; la administración debe confiar en sus equipos y no decirles cómo hacer su trabajo.
Optimizar el todo	Cuando se mejora el proceso de desarrollo de software, el flujo de valor debe ser considerado de un extremo al otro.

Tabla 2.2: Siete principios de Lean

El desarrollo de software Lean, que Poppendieck define como kit de herramientas ágiles, es ligeramente diferente de sus equivalentes en el desarrollo ágil, pero son paralelos [38]. Al aplicar Lean, es común seleccionar un enfoque de software ágil ligero como punto de partida y empezar a aplicar Lean desde allí. Las herramientas Lean incluyen Value Stream Mapping (VSM), Kaizen (mejora continua), Kanban (un sistema de señalización utilizado para señalar la necesidad de un elemento, normalmente usando objetos como fichas, pelotas de colores, o carros vacíos), etc. [41].

2.3.4. Crystal Clear

Crystal es una familia de metodologías definidas por Alistair Cockburn [42]. Existen diferentes metodologías Crystal para diferentes tipos de proyectos. El nombre de "Crystal" se deriva de la clasificación del proyecto a lo largo de dos dimensiones, la criticidad y el tamaño, en una analogía con los cristales geológicos, que también se describen en dos dimensiones: la aspereza del mineral y el color.

Los tipos de proyecto se clasifican según dos factores: el número de personas implicadas en el equipo de desarrollo y el riesgo del proyecto. Crystal dispone de un código de colores, correspondiendo a las metodologías más pesadas los colores más oscuros. Así, los proyectos grandes que implican una mayor coordinación y comunicación se corresponden con los colores más oscuros. Corresponde colores claros a los equipos con ocho o menos personas, el amarillo es para los equipos de 10 a 20 personas, el naranja es para 20 a 50 personas, el rojo es para 50 a 100 personas, y así sucesivamente, a través de marrón, azul y violeta.

La familia de cristal sigue siete propiedades de seguridad [42]. Los tres primeros son el núcleo de Crystal y se requiere que se encuentran en todos los proyectos de entrega frecuente, de mejora reflexiva y estrecha comunicación. Los otros cuatro se pueden agregar para aumentar el margen de seguridad es decir, la seguridad personal, enfoque, fácil acceso a usuarios expertos, y entorno técnico.

La Tabla 2.3 presenta los detalles de cada propiedad según Cockburn [42]:

Propiedades	Descripción
Entregas frecuentes	Entregar el software a los usuarios, y no simplemente la iteración. Entregar código funcional y probado para usuarios reales cada pocos meses se considera como la propiedad más importante de cualquier proyecto, ya sea grande o pequeño.
Mejora reflexiva	Los miembros del equipo son el mejor grupo para decir qué es bueno o malo en su situación. Esta propiedad permite que ellos hagan cambios al tener al equipo reunido, identificar lo que está y no está funcionando después de cada entrega, y discutir maneras de solucionarlo.
Comunicación osmótica	La información fluye en el fondo, donde los miembros del equipo pueden escucharla y recoger información relevante como por ósmosis. Se logra haciendo que el equipo se siente en la misma habitación lo cual hace fácil la comunicación sobre su trabajo.
Seguridad personal	Ser capaz de hablar cuando algo le preocupa al miembro del equipo, sin temor a represalias. Por ejemplo, informar al director del proyecto sobre plazos poco realistas,
Enfoque	Es saber en qué trabajar, mayormente derivándose de los objetivos y prioridades, y luego manejar el tiempo y la mente para trabajar en ello.
Fácil acceso a usuarios expertos	Permite al equipo desplegar y probar las entregas frecuentes, y obtener una rápida retroalimentación sobre la calidad del producto final, las decisiones de diseño, y los requisitos actualizados a la fecha.
Entorno Técnico	Entorno técnico con pruebas automatizadas, gestión de la configuración e integración continua.

Tabla 2.3: Propiedades de Crystal

2.3.5. Feature Driven Development (FDD)

Feature Driven Development (Desarrollo guiado por características) fue creado por Peter Coad y De Luca en 1997 y posteriormente refinada por Palmer entre otros [43][44]. FDD ha sido diseñado y probado para entregar resultados funcionales sólidos, con frecuencia y repetidamente en el desarrollo de software [44]. Se centra en el diseño y construcción de fases en lugar de todo el proceso de desarrollo de software. Está construido en torno a un conjunto básico de "mejores prácticas" de la ingeniería de software. Es mejor aplicar una combinación de todas sus prácticas para obtener su máximo beneficio, pues no existe una práctica única que destaque en todo el proceso. A continuación se describen las prácticas que constituyen FDD [44]:

Prácticas	Descripción
Modelado de objetos de dominio	Construir diagramas de clase representa el principal tipo de objeto, dentro de un dominio del problema, a resolver. Crear el modelo de objetos de dominio proporciona un marco general para agregar funciones, característica por característica, y ayuda a mantener la integridad del sistema conceptual.
Desarrollo por característica	Cualquier función demasiado compleja, incapaz de ser implementada dentro de dos semanas, se divide en características más pequeñas. Esto hará que sea más fácil entregar funciones correctas y ampliar o alterar el sistema.
Propiedad de clases individuales	Asignar una persona o rol que sea responsable de los contenidos de una clase, incluyendo su consistencia, el rendimiento y la integridad conceptual.
Equipos por características	Un equipo por característica suele ser pequeño, de alrededor de tres a seis personas con tareas pequeñas por característica. El equipo de trabajo está formado por propietarios de la Clase que son responsables de modificar las clases para cada función.
Inspecciones	Las inspecciones se realizan para garantizar la buena calidad del diseño y el código, principalmente por personas capacitadas que buscan defectos mediante procesos bien definidos.
Gestión de la configuración	La gestión de configuración se utiliza para manejar el código fuente de todas las características que se han completado hasta la fecha. Mantiene un historial de cambios de clases mientras los equipos las mejoran.
Compilaciones regulares	Construir el sistema completo a intervalos regulares para asegurar que siempre hay un sistema actualizado que pueda demostrarse al cliente.
Visibilidad del progreso y resultados	Mediante el uso de reportes frecuentes, apropiados, precisos y completos del progreso en todos los niveles dentro y fuera del proyecto, los gestores pueden dirigir correctamente el proyecto.

Tabla 2.4: Prácticas definidas para FDD

2.3.6. Test Driven Development (TDD)

Test Driven Development (desarrollo guiado por pruebas) fue introducido como una práctica dentro de *Extreme Programming* (XP) por Kent Beck [32]. Los desarrolladores que utilizan TDD escriben pruebas unitarias automatizadas antes de escribir el código real, motivo por el cual en algunas literaturas también se le conoce como enfoque Test First Development (TFD) [46], aunque para ser exactamente iguales TFD debe incorporar una etapa de refactorización de los cambios aplicados para pasar dichas pruebas.

Las pruebas se escriben en forma de afirmaciones, y el propósito de TDD es definir los requisitos del código que puedan cumplir estas pruebas. Mediante el uso de TDD, los desarrolladores crean los sistemas basándose en ciclos de prueba, desarrollo y refactorización.

Según Beck el uso de TDD se basa en dos simples reglas [45]:

1. **No escribir una línea de código nuevo, a menos que primero tenga una prueba automatizada de errores.**
2. **Eliminar los duplicados.**

Estas dos simples reglas, generan comportamiento individual y grupal complejo. Algunos de las implicaciones técnicas para lograr que funcione son [45]:

- Se debe diseñar de manera orgánica, con el código corriendo para proporcionar retroalimentación entre decisiones.
- Debe escribir sus propias pruebas, ya que no puedes esperar veinte veces al día a que alguien más escriba una prueba.
- Su entorno de desarrollo debe proporcionar una respuesta rápida a los cambios pequeños.
- Su diseño deben estar compuesto de muchos componentes altamente cohesivos y débilmente acoplados, para facilitar las pruebas.

El orden de las tareas de programación, derivados de las reglas seria [46]:

1. Crear la prueba.

2. Ejecutar la prueba: si falló (**ROJO**)
3. Crear código específico para resolver la prueba.
4. Ejecutar de nuevo la prueba: si pasó (**VERDE**)
5. **Refactorizar** el código
6. Ejecutar la prueba de nuevo: si pasó (VERDE)

Rojo / Verde / Refactorizar. El mantra TDD.

Esta práctica constituye, por tanto, la primera línea para garantizar la calidad del producto software, pues ésta garantiza su correcta funcionalidad y entrega rápida al cliente.

2.3.7. Refactorización

La refactorización (*Refactoring*) fue introducida originalmente como una práctica dentro de Extreme Programming, pero explicada a detalle y aplicada por Martin Fowler [47]. Fowler define la refactorización como una técnica disciplinada para la reestructuración de un organismo ya existente de código, alterando su estructura interna sin cambiar su comportamiento externo.

La refactorización es una técnica controlada para mejorar el diseño de un código base existente. Su esencia es la aplicación de una serie de pequeñas transformaciones de preservación de comportamiento, cada transformación (llamada *refactoring*) es "demasiado pequeña como para hacer algo de valor". Sin embargo, el efecto acumulativo de cada una de estas transformaciones es muy significativo. Como cada refactorización es pequeña, es menos probable que salga mal [47] [48].

El sistema se mantiene en pleno funcionamiento después de cada pequeña refactorización, reduciendo las posibilidades de que el sistema pueda quedar seriamente dañado durante la reestructuración, esto permite refactorizar gradualmente un sistema durante un período prolongado de tiempo [47].
Los fundamentos de la refactorización indican que esta se puede emplear en muchos lugares y contextos distintos. Algunas recomendaciones para su implementación [48]:

- Verifique que no haya cambios en la conducta externa, utilizando:
  - Pruebas
  - Herramientas de análisis de código formal
- En la práctica las pruebas bien definidas son esenciales.
- La mejor manera de optimizar el rendimiento es escribir primero un programa bien factorizado, luego optimizarlo.
- El único beneficio de usar objetos es que son más fáciles de cambiar.
- La refactorización le permite mejorar el diseño después de que el código se ha escrito.
- Hacer el diseño primero es aun importante, pero no tan crítico.
- La refactorización es aún un sujeto inmaduro: no se ha escrito mucho y hay muy pocas herramientas.

2.3.8. Programación en pares

Una de las prácticas más utilizadas dentro de XP es la programación en pares. La tasa de éxito de XP es tan impresionante que ha despertado la curiosidad de muchos investigadores de ingeniería de software y consultores. Kent fundador de XP acredita gran parte de este éxito a la utilización de la programación en pares por todos los programadores XP, expertos y novatos por igual [49].

En la programación en pares, dos programadores producen conjuntamente un artefacto (diseño, algoritmo, código). Los dos programadores son como un organismo único, trabajando con un mismo propósito y responsables de todos los aspectos de este artefacto. Uno de los socios, el conductor, controla el lápiz, el ratón o el teclado y escribe el código. El otro socio observa de forma continua y apoya el trabajo del conductor, buscando defectos, pensando en alternativas, buscando recursos, y teniendo en cuenta las implicaciones estratégicas. Los socios cambian deliberadamente papeles periódicamente. Ambos son participantes activos en el proceso en todo momento y comparten por igual la propiedad del producto de trabajo en su totalidad, así se trate del esfuerzo de una mañana o de todo un proyecto [49].

Los beneficios más significativos de la programación en pares son [50]:

- La mayoría de errores se descubren a medida que se escriben en lugar de en las pruebas de control de calidad o en el campo.
- Los defectos de contenido finales son estadísticamente menores.
- Los diseños son mejores y la longitud del código más corta (lluvia de ideas en curso y para retransmisión).
- El equipo resuelve los problemas más rápido.
- Las personas aprenden mucho más, sobre el sistema y sobre el desarrollo de software.
- El proyecto termina con varias personas que entienden cada una de las piezas del sistema.
- Las personas aprenden a trabajar en equipo y a comunicarse más a menudo, creando un mejor flujo de información y de dinámica de equipo.
- Las personas disfrutan más de su trabajo.

2.4 Metodologías Ágiles vs Tradicionales

No existe una metodología universal para hacer frente con éxito a cualquier proyecto de desarrollo de software. Toda metodología debe ser adaptada al contexto del proyecto (recursos técnicos y humanos, tiempo de desarrollo, tipo de sistema, etc.) [23].

Los enfoques de desarrollo tradicionales han existido por un muy largo tiempo. Desde su introducción por Winston W. Royce en 1970, el modelo en cascada ha sido ampliamente utilizado en grandes y pequeños proyectos de software, reportando tener éxito en muchos proyectos. A pesar de su éxito presenta varios inconvenientes, como la linealidad, la rigidez ante los requisitos cambiantes, y los procesos altamente formales independientemente del tamaño del proyecto. Kent Beck tomó en cuenta estos inconvenientes y presentó *Extreme Programing*, la primera metodología ágil propiamente dicha.

Los métodos ágiles se ocupan de requisitos inestables y volátiles mediante el uso de una serie de técnicas, centrándose en la colaboración entre los desarrolladores y los clientes, y centrándose en la entrega temprana del producto.

La Tabla 2.4 muestra un resumen de las diferencias entre las metodologías ágiles y las tradicionales.

	Ágil	Tradicional
Objetivo principal	Obtener valor rápidamente.	Alto aseguramiento.
Enfoque	Adaptativo	Predictivo
Requerimientos	En gran parte emergentes, rápido cambio, desconocidos.	Reconocibles al inicio, en gran parte estables.
Tamaño	Equipos y proyectos pequeños	Equipos y proyectos grandes
Arquitectura	Diseñado para los requerimientos actuales.	Diseñado para los requerimientos actuales y previsibles.
Planeamiento y control	Planes internalizados, control cualitativo	Planes documentados, control cuantitativo
Estilo de gestión	Descentralizado	Autocrático
Perspectiva al cambio	Adaptabilidad al cambio	Sostenibilidad al cambio
Clientes	Clientes dedicados, bien informados, lado a lado, colocados in-situ.	Interacción con clientes solo cuando es necesario, se centra en las disposiciones del contrato.
Desarrolladores	Ágil, bien informados, con ubicación compartida, y de colaboración.	Orientados a la planificación; habilidades adecuadas, acceso a conocimiento externo.
Refactorización	Barato	Costoso
Riesgos	Riesgos desconocidos, mayor impacto.	Riesgos bien entendidos, menor impacto
Documentación	Poca	Demasiada
Énfasis	Orientado a las personas	Orientado al proceso
Retorno de la inversión	Temprano en el proyecto	Al final del proyecto

Tabla 2.5: Diferencias entre Metodologías Ágiles y Tradicionales [51]

Los factores de riesgo más importantes en el desarrollo de un proceso de software son la criticidad del proyecto y responder a los cambios. Los métodos ágiles se utilizan en aplicaciones que se puedan construir de forma rápida y no requieren un extenso

proceso de aseguramiento de calidad. Los sistemas críticos, de alta confiabilidad y seguridad son más adecuados para una metodología pesada tradicional. Si un proyecto es crítico, todos sus requisitos deben estar bien definidos antes del desarrollo del software. La respuesta al cambio se puede resolver utilizando un método ágil, las prácticas ágiles permiten un mejor manejo de los cambios [51]

2.5 Calidad en el Desarrollo de Software

Aunque la calidad del software es fundamental para el éxito de un producto software, como concepto es difícil de definir, describir, comprender y medir [52].

En la literatura la calidad se ha definido de varias formas:

> *"El conjunto de características de un producto o servicio que le confieren su aptitud para satisfacer necesidades expresas o implícitas"*

ISO 8402 [53]

> *"El grado en que un sistema, componente o proceso cumple con los requisitos especificados y las necesidades o expectativas del usuario"*

IEEE [21]

> *"El grado en que el producto software cumple necesidades declaradas e implícitas cuando se usa bajo condiciones especificadas"*

ISO/IEC 25010 [21]

Todas estas definiciones se centran en satisfacer la necesidad del cliente mediante el producto software. Un modo de lograr este fin es mediante la conformidad de los requerimientos especificados. Esto se logra verificar mediante el proceso de aseguramiento de la calidad.

2.5.1. Aseguramiento de la Calidad (*Quality Assurance*)

Según la Norma ISO 9000:2005, el aseguramiento de la calidad es “una parte de la gestión de calidad orientada a proporcionar confianza en que se cumplirán los requisitos de calidad” [54].

El proceso de aseguramiento de calidad es un proceso orientado a proporcionar la adecuada garantía de que los productos software y procesos en el ciclo de vida del proyecto, se ajustan a los requisitos especificados y se adhieren a sus planes establecidos. Para ser imparcial, el aseguramiento de la calidad debe tener libertad de la organización y autoridad sobre las personas directamente responsables de desarrollar el producto software o ejecutar el proceso en el proyecto [57].

La garantía de calidad puede ser interna o externa, la calidad interna tiene como objetivo medir la calidad del software mediante factores medibles durante su desarrollo; la calidad externa pretende medir la calidad del software teniendo en cuenta el comportamiento de este software en un sistema del cual forme parte [22]. La garantía de calidad puede hacer uso de los resultados de otros procesos de apoyo, tales como procesos de verificación, validación, revisiones conjuntas, auditorías, y de resolución de problemas.

El aseguramiento de la calidad en las metodologías ágiles, como XP, está integrado mediante el uso de pruebas unitarias y funcionales, así como a través de la presencia del cliente junto al equipo [32]. Es explícitamente parte de la descripción del rol del cliente asegurarse de que el equipo de desarrollo elabore el software que cumpla con sus requerimientos, mediante la definición y la ejecución de pruebas de aceptación y haciendo revisiones críticas durante el desarrollo.

2.5.2. Calidad Interna y Externa

Los enfoques de calidad interna y externa del producto software se definen claramente desde el estándar ISO/IEC 9126-1 tal como sigue [55]:

- **Calidad Interna**: Se basa en la inspección y evaluación de atributos estáticos, puede utilizarse para medir las propiedades inherentes de un

producto software. Puede ser medida por medio de atributos estáticos de artefactos tales como: i) Especificación de requerimientos, ii) Arquitectura o diseño, iii) Piezas de código fuente, entre otros. La calidad interna se puede medir y controlar en etapas tempranas del ciclo de vida del software.

- **Calidad Externa**: Se basa en la medición y evaluación de atributos dinámicos del producto software ejecutándose en el entorno del sistema en el que está destinado a funcionar: La calidad externa se mide en etapas tardías del ciclo de vida del software, principalmente en etapas de pruebas con el producto software ya en estado operativo.

2.5.3. Atributos de Calidad

Un atributo es una entidad que puede ser verificado o medido en el producto software [62]. Los atributos varían entre los diferentes productos software. Existen diferentes clasificaciones y agrupaciones de atributos de calidad, siendo una de las más representativas la utilizada por ISO/IEC 9126 [55]

ISO define algunos atributos de calidad como:

- Funcionalidad
- Confiabilidad
- Usabilidad
- Mantenibilidad
- Portabilidad
- Eficiencia
- Recuperabilidad
- Interoperabilidad
- Operatividad
- Mutabilidad
- Estabilidad
- Seguridad
- Precisión

La cantidad de atributos definidos en diferentes normas y estándares es muy extensa. En este documento se definirán solamente los atributos utilizados en la presente tesis (ver Tabla 2.6).

2.5.4. Modelos de Calidad

Los modelos de calidad han sido un tema de investigación durante varias décadas. Se han propuesto un gran número de modelos de calidad [56].

Los primeros modelos de calidad del software publicados datan de finales de 1970, cuando Boehm et al. [57], así como McCall, Richards y Walter [58] definen las características de calidad y su descomposición. Los dos enfoques eran similares y utilizan una descomposición jerárquica del concepto de calidad en factores como mantenibilidad o fiabilidad. Diversas variaciones de estos modelos han aparecido en el tiempo, siendo uno de los más populares el modelo FURPS [59] que descompone la calidad en: funcionalidad, facilidad de uso, fiabilidad, rendimiento y compatibilidad. La idea principal de estos modelos es descomponer la calidad a un nivel donde se pueda medir, y así poder evaluar la calidad del producto software.

Este tipo de modelos de calidad se convirtió en la base del estándar ISO/IEC 9126 [55] en el año 1991. La norma define una descomposición estándar en características de calidad y sugiere un pequeño número de métricas para su medición. Sin embargo, estas métricas no cubren todos los aspectos de calidad, por lo cual la norma no operacionaliza completamente la calidad. El sucesor de esta norma, la nueva ISO/IEC 25010 [22], cambia algunas clasificaciones, pero mantiene la descomposición jerárquica general.

A continuación se detallan algunos de estos modelos de calidad incluyendo los utilizados en el presente estudio.

ISO/IEC 9126

ISO 9126 es un estándar para la evaluación de la calidad del producto software. La Figura 2.6 muestra el marco de trabajo del modelo de calidad 9126, en él se muestran las relaciones entre sus diferentes criterios de calidad.

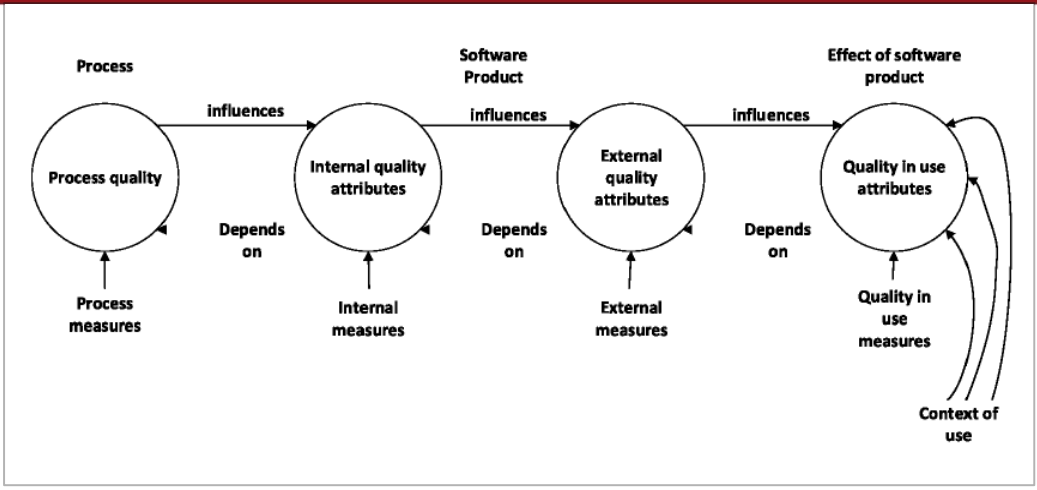

Figura 2.6: Calidad en el Ciclo de vida del software [55]

El modelo de calidad se divide en tres submodelos de calidad del producto software (calidad interna, calidad externa y calidad en uso), diez características de calidad, 24 características secundarias y más de 250 medidas propuestas para cuantificar estas características y subcaracterísticas de calidad.

El modelo de calidad para los atributos externos e internos se muestra en la Figura 2.7. El modelo se divide en seis características o factores de calidad (funcionalidad, confiabilidad, usabilidad, eficiencia, mantenibilidad y portabilidad), que a su vez se subdividen en subcaracterísticas.

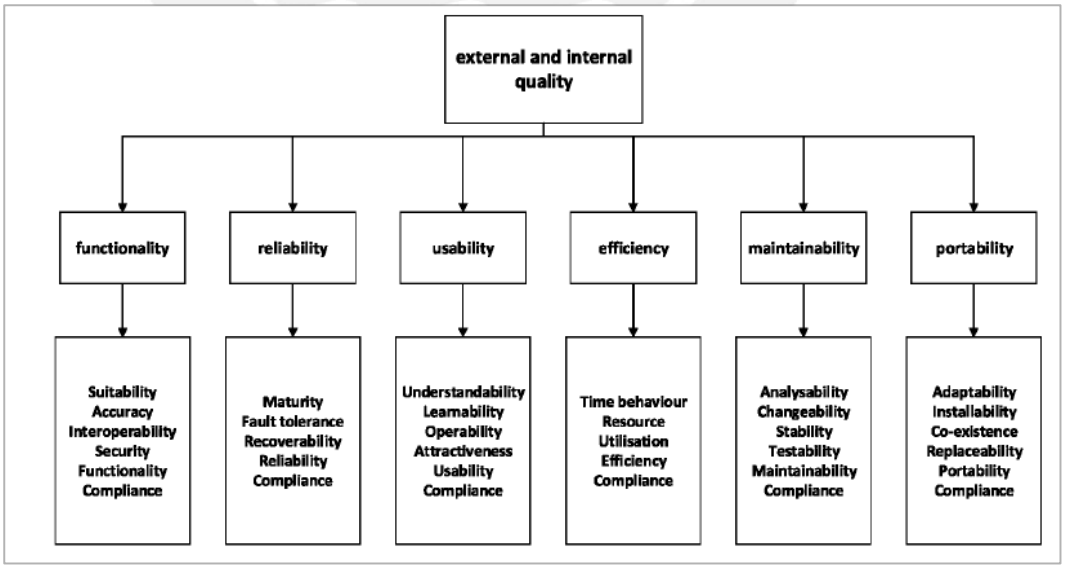

Figura 2.7: Modelo de calidad para calidad interna y externa [55]

ISO/IEC 9126 define los seis factores de calidad del siguiente modo [55]:

- **Funcionalidad**: conjunto de atributos que soporta la existencia de un conjunto de funciones y sus propiedades específicas. Las funciones son tales que satisfacen las necesidades establecidas.
- **Confiabilidad**: conjunto de atributos que soporta la capacidad del software para mantener su nivel de rendimiento bajo condiciones establecidas, por un periodo de tiempo definido.
- **Usabilidad**: conjunto de atributos que soporta el esfuerzo necesario para el uso y la evaluación individual de tal uso mediante un conjunto de usuarios establecidos e implícitos.
- **Eficiencia**: conjunto de atributos que soporta las relaciones entre el nivel de rendimiento del software y el monto de recursos empleados, bajo condiciones establecidas.
- **Mantenibilidad**: conjunto de atributos que soporta el esfuerzo necesario para realizar modificaciones especificadas.
- **Portabilidad**: conjunto de atributos que soporta la habilidad del software para transferirlo de un entorno a otro.

Las características y subcaracterísticas asociados a la calidad interna y externa se detallan y definen en la Tabla 2.6:

Característica	Subcaracterística	Definición
Funcionalidad	**Adecuación**	Capacidad para proporcionar un conjunto de funciones apropiadas para ciertas tareas y objetivos.
	Precisión	Capacidad para proporcionar los resultados o efectos correctos o acordados, con el grado necesario de precisión.
	Interoperabilidad	Capacidad del producto software para interactuar con uno o más sistemas.
	Seguridad	Capacidad para proteger la información y programas de accesos no autorizados, ya sean accidentales o deliberados.
	Conformidad de funcionalidad	Capacidad para adherirse a normas, convenciones, regulaciones y prescripciones similares relacionadas con la funcionalidad.
Confiabilidad	**Madurez**	Capacidad del producto software para evitar fallar como resultado de errores en el software
	Tolerancia a fallos	Capacidad del software para mantener un nivel especificado de prestaciones en caso de fallos o de infringir sus interfaces.
	Recuperabilidad	Capacidad para restablecer un cierto nivel de prestaciones y de recuperación de datos directamente afectados en caso de fallo.
	Conformidad de	Capacidad para adherirse a normas, convenciones o regulaciones relacionadas

	confiabilidad	con la confiabilidad.
Usabilidad	**Comprensibilidad**	Capacidad que permite al usuario entender si el software es adecuado y cómo puede ser usado para tareas o condiciones de uso particulares.
	Facilidad de aprendizaje	Capacidad del producto software que permite al usuario aprender sobre su aplicación.
	Operatividad	Capacidad del producto software que permite al usuario administrarlo y controlarlo.
	Atractivo	Capacidad de ofrecer un entorno atractivo para el usuario.
	Conformidad de usabilidad	Capacidad para adherirse a normas, convenciones, guías de estilo o regulaciones relacionadas con la usabilidad.
Eficiencia	**Comportamiento temporal**	Capacidad para proporcionar tiempos de respuesta, de proceso e índices de respuesta al realizar sus funciones bajo ciertas condiciones.
	Utilización de recursos	Capacidad para usar las cantidades y tipos de recursos adecuados cuando el software lleva a cabo su funcionamiento bajo condiciones determinadas.
	Cumplimiento de eficiencia	Capacidad para adherirse a normas o convenciones relacionadas con la eficiencia.
Mantenibilidad	**Analizabilidad**	Capacidad para poder diagnosticar deficiencias o causas de fallos en el software, o para identificar las partes que han de ser modificadas.
	Mutabilidad	Capacidad del producto software que permite que una determinada modificación o remoción de fallas, sea implementada.
	Estabilidad	Capacidad del producto software para evitar efectos inesperados debidos a modificaciones del software.
	Comprobabilidad	Capacidad del producto software que permite que el software modificado sea validado.
	Cumplimiento de mantenibilidad	Capacidad para adherirse a normas o convenciones relacionadas con la mantenibilidad.
Portabilidad	**Adaptabilidad**	Capacidad de adaptación a diferentes entornos sin la aplicación de otras acciones o medios diferentes a los provistos.
	Instalación	Capacidad del producto software para ser instalado en un cierto entorno.
	Coexistencia	Capacidad para coexistir con otro software independiente, en un entorno común, compartiendo recursos comunes
	Reemplazabilidad	Capacidad del producto software para ser usado en lugar de otro producto software, para el mismo propósito, en el mismo entorno.
	Cumplimiento de portabilidad	Capacidad del producto software para adherirse a normas o convenciones relacionadas con la portabilidad.

Tabla 2.6: Características internas y externas de ISO 9126. (Adaptado de [55])

ISO 9126-1: 1999 define la calidad en el uso de atributos como la visión de calidad que tiene el usuario. Los atributos de calidad en uso se clasifican en cuatro características: eficacia, productividad, seguridad y satisfacción. La calidad en uso determina la capacidad del producto software para lograr estas características específicas.

- La métrica de efectividad evalúa las tareas realizadas por los usuarios que han alcanzado los objetivos con precisión y exhaustividad.
- La métrica de efectividad no evalúa cómo se lograron los objetivos, sólo cuales fueron cumplidos.
- La métrica de productividad evalúa los recursos que los usuarios consumen en relación con la eficacia alcanzada. La métrica de seguridad evalúa el nivel de riesgo para las personas, las empresas, el software, los bienes o el medio ambiente.
- Las métricas de satisfacción evalúan la actitud del usuario hacia el uso del producto.

ISO/IEC 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 [61].

La familia ISO/IEC 25000 es el resultado de la evolución de otras normas anteriores, especialmente de la norma 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 está compuesta de cinco divisiones, las cuales se muestran en la Figura 2.8.

Figura 2.8: Divisiones familia ISO/IEC 25000 [60]

- **ISO/IEC 2500n - División de Gestión de la Calidad:** Las normas que forman este apartado definen todos los modelos, términos y definiciones comunes referenciados por todas las otras normas de la familia 25000. Actualmente esta división se encuentra formada por:
  a) *ISO/IEC 25000 - Guide to SQuaRE*: contiene el modelo de la arquitectura de SQuaRE, la terminología de la familia, un resumen de las partes, los usuarios previstos y las partes asociadas, así como los modelos de referencia.
  b) *ISO/IEC 25001 - Planning and Management*: establece los requisitos y orientaciones para gestionar la evaluación y especificación de los requisitos del producto software.

- **ISO/IEC 2501n - División de Modelo de Calidad:** Las normas de este apartado presentan modelos de calidad detallados incluyendo características para calidad interna, externa y en uso del producto software. Actualmente se encuentra formada por:
  a) *ISO/IEC 25010 - System and software quality models*: describe el modelo de calidad para el producto software y para la calidad en uso. Esta Norma presenta características y subcaracterísticas de calidad frente a las cuales se puede evaluar el producto software.
  b) *ISO/IEC 25012 - Data Quality model*: define un modelo general para la calidad de los datos, aplicable a aquellos datos que se encuentran

almacenados de manera estructurada y forman parte de un Sistema de Información.

- **ISO/IEC 2502n - División de Medición de la Calidad:** Estas normas incluyen un modelo de referencia de la medición de la calidad del producto, definiciones de medidas de calidad (interna, externa y en uso) y guías prácticas para su aplicación. Actualmente se encuentra formada por:
  a) ISO/IEC 25020 - *Measurement reference model and guide.*
  b) ISO/IEC 25021 - *Quality measure elements*.
  c) ISO/IEC 25022 - *Measurement of quality in use.*
  d) ISO/IEC 25023 - *Measurement of system and software product quality.*
  e) ISO/IEC 25024 - *Measurement of data quality.*

- **ISO/IEC 2503n - División de Requisitos de Calidad**: Las normas que forman este apartado ayudan a especificar requisitos de calidad que pueden ser utilizados en el proceso de obtención de requisitos de calidad del producto software a desarrollar, o como entrada del proceso de evaluación. Para ello, este apartado se compone de:
  a) ISO/IEC 25030 - *Quality requirements.*

- **ISO/IEC 2504n - División de Evaluación de la Calidad**: Este apartado incluye normas que proporcionan requisitos, recomendaciones y guías para llevar a cabo el proceso de evaluación del producto software. Esta división se encuentra formada por:
  a) ISO/IEC 25040 - *Evaluation reference model and guide.*
  b) ISO/IEC 25041 - *Evaluation guide for developers, acquirers and independent evaluators.*
  c) ISO/IEC 25042 - *Evaluation modules.*
  d) ISO/IEC 25045 - *Evaluation module for recoverability.*

ISO/IEC 25010

Este modelo de calidad representa la piedra angular en torno a la cual se establece el sistema para la evaluación de la calidad del producto software. En este modelo se determinan las características de calidad que se van a tener en

cuenta a la hora de evaluar las propiedades de un producto software determinado [22].

La calidad del producto software se puede interpretar como el grado en que dicho producto satisface los requisitos de sus usuarios aportando de esta manera un valor. Son precisamente estos requisitos (funcionalidad, rendimiento, seguridad, mantenibilidad, etc.) los que se encuentran representados en el modelo de calidad, el cual categoriza la calidad del producto en características y subcaracterísticas [60].

Este estándar internacional define:

- Un modelo de la calidad del producto software, compuesto de ocho características, que se dividen en subcaracterísticas que se pueden medir interna o externamente.
- Un modelo de calidad en uso del sistema, compuesto por cinco características, que se dividen en subcaracterísticas que se pueden medir cuando un producto se utiliza en un contexto real de uso.

Figura 2.9: Características de calidad del producto ISO/IEC 25010 [60]

La Figura 2.9 detalla el modelo de calidad del producto software, este se compone de ocho características de calidad, las se definen en el estándar tal como sigue:

1) **Adecuación Funcional**: Representa la capacidad del producto software para proporcionar funciones que satisfacen las necesidades declaradas e implícitas, cuando el producto se usa en las condiciones

especificadas. Esta característica se divide a su vez en las siguientes subcaracterísticas: Completitud funcional, Corrección funcional y Pertinencia funcional.

2) **Eficiencia de desempeño**: Esta característica representa el desempeño relativo a la cantidad de recursos utilizados bajo determinadas condiciones. Esta característica se divide a su vez en las siguientes subcaracterísticas: Comportamiento temporal, Utilización de recursos y Capacidad.
3) **Compatibilidad**: Capacidad de dos o más sistemas o componentes para intercambiar información y/o llevar a cabo sus funciones requeridas cuando comparten el mismo entorno de hardware o software. Esta característica se divide a su vez en las siguientes subcaracterísticas: Coexistencia e Interoperabilidad.
4) **Usabilidad**: Capacidad del producto software para ser entendido, aprendido, usado y resultar atractivo para el usuario, cuando se usa bajo determinadas condiciones. Esta característica se divide a su vez en las siguientes subcaracterísticas: Capacidad para reconocer su adecuación, Capacidad de aprendizaje, Capacidad para ser usado, Protección contra errores de usuario, Estética de la interfaz de usuario, y Accesibilidad.
5) **Fiabilidad**: Capacidad de un sistema o componente para desempeñar las funciones especificadas, cuando se usa bajo unas condiciones y periodo de tiempo determinados. Esta característica se divide a su vez en las siguientes subcaracterísticas: Madurez, Disponibilidad, Tolerancia a fallos, y Capacidad de recuperación.
6) **Seguridad**: Capacidad de protección de la información y los datos de manera que personas o sistemas no autorizados no puedan leerlos o modificarlos. Esta característica se divide a su vez en las siguientes subcaracterísticas: Confidencialidad, Integridad, No repudio, Responsabilidad, y Autenticidad.
7) **Mantenibilidad**: Esta característica representa la capacidad del producto software para ser modificado efectiva y eficientemente, debido a necesidades evolutivas, correctivas o perfectivas. Esta característica se divide a su vez en las siguientes subcaracterísticas: Modularidad, Reusabilidad, Analizabilidad, Capacidad para ser modificado, y Capacidad para ser probado.

8) **Portabilidad**: Capacidad del producto o componente de ser transferido de forma efectiva y eficiente de un entorno de hardware, software, operacional o de utilización hacia otro. Esta característica se subdivide a su vez en las siguientes subcaracterísticas: Adaptabilidad, Capacidad para ser instalado, y Capacidad para ser reemplazado.

El modelo de calidad de uso del software está compuesto por cinco características de calidad, las cuales se detallan en la Figura 2.10:

Figura 2.10: Características de calidad de uso ISO/IEC 25010 [60]

Mientras que el modelo de calidad del producto describe las características del producto directamente, el modelo de la calidad en uso analiza las características de las interacciones de los diferentes grupos de interés con el producto. El más prominente de estos grupos de interés es el usuario principal. La calidad en uso, sin embargo, también puede significar la calidad en el mantenimiento, portabilidad del producto, definir el contexto del producto, etc.

1) **Efectividad**: La capacidad del producto software para facilitar a los usuarios alcanzar metas específicas con exactitud y completitud en un contexto específico de uso.
2) **Productividad**: La capacidad del producto software para invertir la cantidad apropiada de recursos en relación a la eficacia alcanzada en un contexto especifico de uso.
3) **Seguridad**: La capacidad del producto software para alcanzar niveles aceptables de riesgo de dañar a las personas, el negocio, el software, la propiedad o el ambiente en un contexto especifico de uso.

4) **Satisfacción**: La capacidad del producto software para satisfacer a los usuarios en un contexto específico de uso.
5) **Contexto de uso**: La capacidad para funcionar en un determinado contexto y su flexibilidad en el mismo.

ISO/IEC 12207: 2008

ISO/IEC 12207 Systems and software engineering - Software life cycle processes [21]: define un marco de referencia común para los procesos del ciclo de vida del software desde la conceptualización de ideas hasta su retirada. Este estándar define 43 procesos que pueden aplicarse durante la adquisición de un producto o servicio de software y durante el suministro, desarrollo, operación, mantenimiento y evolución de productos de software. Cubre además el control y la mejora de estos procesos.

El estándar se divide en dos grandes grupos:

1. Procesos de Contexto del Sistema
   - Procesos de Acuerdo: definen las actividades necesarias para establecer un acuerdo entre dos organizaciones.
   - Procesos Organizacionales Facilitadores de Proyecto: administran la capacidad de la organización para adquirir y suministrar productos o servicios a través de la iniciación, el soporte y el control de proyectos y asegurar los objetivos organizacionales y los acuerdos establecidos.
   - Procesos de Proyecto: definen los procesos relacionados con la planificación, evaluación y control. Los principios relacionados con estos procesos se pueden aplicar en cualquier área de la gestión de una organización.
   - Procesos Técnicos: se utilizan para definir los requisitos de un sistema, para transformar los requisitos en un producto eficaz, para utilizar el producto, para prestar los servicios requeridos, para sostener la prestación de dichos servicios y para disponer del producto cuando se retire del servicio.
2. Procesos Específicos del Software
   - Procesos de Implementación del Software: se utilizan para producir un elemento especifico del sistema (elemento de software) implementado en el software. Estos procesos transforman comportamiento específico, interfaces y limitaciones de ejecución en acciones de implementación

que resultan en un elemento del sistema que satisfaga los requerimientos derivados de los requisitos del sistema.

- Procesos de Soporte del Software: proporcionan un conjunto enfocado específico de actividades para llevar a cabo un proceso de software especializado. Estos procesos ayudan al proceso de implementación de software como parte integral con un propósito distinto.
- Procesos de Reúso de Software: apoyan la capacidad de una organización para reutilizar los elementos de software a través de los límites del proyecto. Estos procesos, por su naturaleza, operan fuera de los límites de cualquier proyecto en particular.

La figura 2.11 muestra cada uno de estos grupos sus subdivisiones y procesos dentro de las mismas.

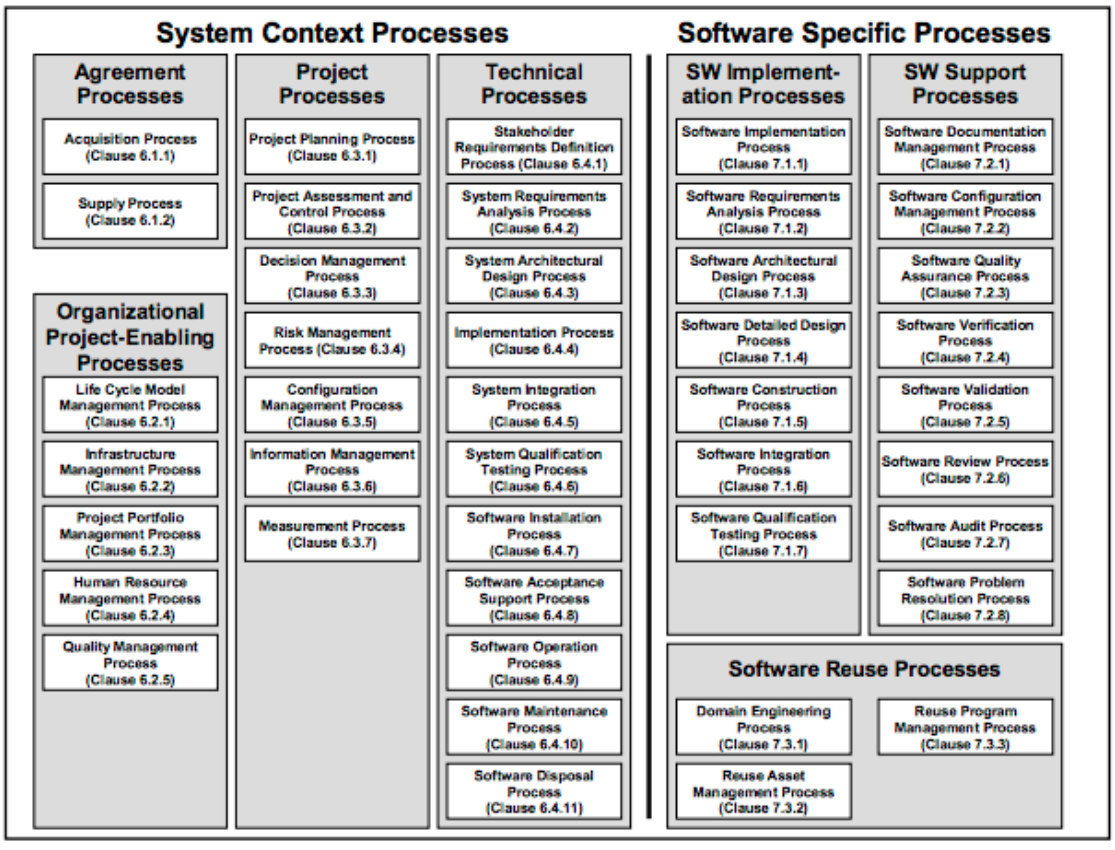

Figura 2.11 Procesos del Ciclo de Vida según 12207 [21]

2.6 Revisión Sistemática de la Literatura

La revisión sistemática de la literatura se convirtió en una metodología de investigación popular desde la década de 1990. En esta década fue ampliamente utilizada en la investigación médica, área de donde proviene originalmente [4].

El número de investigadores de ingeniería de software que realizan revisiones sistemáticas ha aumentado continuamente desde 2004. Muchas publicaciones científicas y revistas de primer nivel tienen secciones especiales para documentos basados en revisiones sistemáticas, adicionalmente un significativo número de conferencias de ingeniería de software buscan presentaciones en esta categoría [4].

2.6.1. Definición

Según Kitchenham, una revisión sistemática de la literatura (comúnmente conocida como revisión sistemática) es una metodología de investigación fiable y rigurosa para identificar, evaluar e interpretar toda la investigación disponible acerca de una pregunta de investigación de un tema en específico. Una revisión sistemática sigue una secuencia bien definida y estricta de pasos metodológicos, de acuerdo con un protocolo desarrollado previamente [1].

Esta sintetiza el trabajo que existe respecto al tema de investigación, obteniendo un panorama resumido de su estado actual, esto permite conocer el avance logrado y lo que aún queda pendiente por investigar y experimentar.

Los estudios fundamentales de la revisión sistemática de la literatura se conocen como estudios primarios, y la revisión sistemática en si se conoce como una forma de estudio secundario.

2.6.2. Razones para Adoptar una Revisión Sistemática

La principal razón para realizar una revisión sistemática es el valor científico que esta tiene en comparación a una revisión de la literatura de cualquier otro tipo. A pesar de la importancia de las revisiones de la literatura (no sistemáticas), incluso cuando estas se llevan a cabo siguiendo sus

correspondientes 'buenas prácticas', estas sufren de falta de rigor científico en la realización de sus diferentes etapas [3].

Hay muchas otras razones para llevar a cabo una revisión sistemática de la literatura tal como lo describe Kitchnham [1]. Las razones más comunes son:

- *"Resumir las pruebas existentes en relación a un tratamiento o tecnología por ejemplo, para resumir la evidencia empírica de los beneficios y limitaciones de un método ágil específico".*
- *"Identificar las lagunas en la investigación actual con el fin de sugerir áreas para una investigación ulterior".*
- *"Proporcionar un marco o base para posicionar adecuadamente nuevas actividades de investigación".*

2.6.3. Características importantes de una Revisión Sistemática

La revisión sistemática difiere de una revisión de la literatura experta convencional; algunas de las características que contribuyen a esta diferenciación se exponen a continuación [1]:

- Uno de los elementos importantes en la revisión sistemática es el desarrollo del protocolo de la revisión. El protocolo de la revisión especifica las preguntas de investigación que se abordan y los métodos que se utilizarán para llevar a cabo una revisión en particular.

- Para llevar a cabo la revisión se utiliza una estrategia de búsqueda definida. El objetivo de la estrategia de búsqueda es identificar el mayor número posible de literaturas relevantes.

- La estrategia de búsqueda y los resultados se documentan para referencias futuras del lector.

- Para evaluar el potencial de estudio primario, la revisión sistemática requiere la especificación de los criterios de inclusión y exclusión para las selecciones de estudio.

- La revisión sistemática específica la información necesaria a ser extraída de los estudios primarios y la evalúa a través de criterios de calidad. Formularios de extracción de datos u otras herramientas de revisión, se utilizan para documentar dicha información extraída.

- La revisión sistemática es considerada como un pre-requisito para el meta-análisis cuantitativo, el cual proporciona estudios de investigación integrados de diversas fuentes sobre el mismo tema.

2.6.4. El Proceso de la Revisión Sistemática

Una revisión sistemática de la literatura consiste de diversas actividades discretas. Las tres fases principales en la revisión sistemática de la literatura son las siguientes [1] [2]:

- Planificación de la revisión
- Implementación de la revisión
- Informe de la revisión

Las etapas antes mencionadas se explican brevemente en las siguientes subsecciones.

2.6.5. Planificación de la Revisión Sistemática de la Literatura

La revisión sistemática de la literatura debe llevarse a cabo de acuerdo a una estrategia de búsqueda predefinida, esta debe permitir evaluar y replicar la búsqueda en su totalidad, por lo cual debe estar planificada al detalle.

La planificación es la fase inicial de la revisión sistemática que comprende un plan de todos los pasos a seguir. El punto de partida es la identificación de la necesidad de hacer la revisión y la formación de las preguntas de investigación que deben ser respondidas por la misma [1]. Las preguntas de investigación se formulan y presentan en el protocolo de la revisión en el capítulo 4.

Después de analizar la necesidad de la revisión en un área en particular, se busca minuciosamente en las bases de datos y fuentes disponibles para ver si existe ya alguna revisión sistemática similar a la que se desea realizar. El

objetivo de la búsqueda es encontrar si ya existe un estudio similar que responde a las preguntas de investigación propuestas.

Luego se desarrolla un protocolo de revisión que especifica los pasos a seguir. Este protocolo predefinido es necesario para reducir los errores de sesgo introducidos por el investigador en cuestión. Los componentes de un protocolo incluyen todos los elementos de la revisión, además de un poco de información de planificación adicional, estos son [1]:

- Antecedentes. La justificación del estudio.
- Las preguntas de investigación que la revisión pretende responder.
- La estrategia que se utilizará para buscar los estudios primarios incluidos los términos de búsqueda y los recursos que se debe buscar. Los recursos incluyen bibliotecas digitales, revistas específicas, y actas de congresos.
- Criterios y procedimientos de selección de los estudios, los cuales se utilizarán para determinar qué estudios serán incluidos o excluidos de la revisión sistemática.
- Listas de verificación y procedimientos de evaluación de la calidad de los estudios. Los investigadores deben desarrollar listas de control de calidad para evaluar los estudios individuales.
- Estrategia de extracción de datos, la cual define cómo se obtendrá la información requerida de cada estudio primario. Si los datos requieren manipulación o suposiciones e inferencias a realizar.
- Síntesis de los datos extraídos, aquí se define la estrategia de síntesis a utilizar, debe aclarar si se pretende un meta-análisis formal y si es así las técnicas que se utilizarán.
- Estrategia de difusión (si es que no está incluido en un documento de puesta en marcha).
- Calendario del proyecto, planeamiento de las actividades de la revisión.

Por último, el protocolo de construcción debe ser revisado por expertos.

2.6.6. Implementación de la Revisión Sistemática de la Literatura

Esta fase se inicia después de la aceptación del protocolo de revisión y se compone de varios pasos. El primer paso es identificar las fuentes de las que

se realiza la revisión. Este proceso de identificación se inicia mediante la búsqueda y consulta de todas las bases de datos disponibles para la literatura primaria.

Los pasos son los siguientes [1] [2]:

i. *Proceso de búsqueda*: La estrategia de búsqueda se define y se sigue para llevar a cabo la revisión. La búsqueda se realiza en las bases de datos electrónicas definidas, así como otras posibles fuentes, como revistas, registros de investigación y listas de referencias obtenidas de los estudios primarios. El sesgo de publicación se debe reducir tanto como sea posible.

ii. *Selección de estudios*: La selección de los estudios se utiliza para identificar y seleccionar los materiales de estudio más apropiados y pertinentes de los documentos ya encontrados. Este proceso de identificación se realiza utilizando los criterios de inclusión y exclusión, definidos en base a las preguntas de investigación.

iii. *Evaluación de la calidad de los estudios*: Se analiza la calidad de los documentos primarios identificados. La calidad se basa en tres factores: validez interna, validez externa y sesgo. Se deberá desarrollar una herramienta de medición de la calidad, la cual es básicamente una lista de factores que necesitan ser evaluados para cada estudio.

iv. *Extracción de datos*: El objetivo de esta etapa es el diseño de formularios de extracción de datos para registrar la información obtenida del estudio primario con precisión. Los datos de los estudios primarios se extraen y se almacenan en los formularios de extracción de datos definidos.

v. *Síntesis de datos*: Consiste en recopilar y resumir los resultados de los estudios primarios incluidos. En resumen, los datos extraídos se sintetizan con el fin de informar los resultados de los estudios primarios examinados. Esta síntesis de los datos extraídos le dará respuesta a la pregunta de investigación propuesta. La respuesta final podría construirse a partir de indicios y pistas de varios trabajos de investigación, la extracción de esta información se puede realizar de diferentes formas y fuentes.

2.6.7. Informe de la Revisión Sistemática de la Literatura

Es importante comunicar los resultados de una revisión sistemática de manera efectiva. Por lo general, las revisiones sistemáticas se reportan al menos en dos formatos [2]:

1. En un informe técnico o en una sección de una tesis doctoral.
2. En un artículo de revista científica o conferencia.

Los artículos de revista científica o conferencia tendrán normalmente una restricción de tamaño. Con el fin de garantizar que los lectores sean capaces de evaluar adecuadamente el rigor y la validez de una revisión sistemática, los artículos deben hacer referencia a un informe técnico o tesis que contenga todos los detalles [2].

2.6.8. Diferencias entre Revisión Sistemática y Revisión Convencional

En la práctica, la distinción entre los artículos de revisión ordinarios y los de una revisión sistemática se puede hacer, en un principio, mediante la comparación de sus estructuras semánticas subyacentes, como lo demuestran los tipos de contenidos en sus respectivos resúmenes, así como en los títulos de las secciones de artículos respectivos [3].

Las revisiones de literatura tradicionales pueden introducir, y lo hacen normalmente, algunos sesgos de investigación en las diferentes etapas del proceso de revisión, que van desde la formulación de la pregunta, a la recopilación de datos, evaluación de datos, el análisis, la interpretación, el resumen y la presentación [3]. La revisión sistemática minimiza estos posibles errores.

La Tabla 2.7 muestra las diferencias entre las revisiones tradicionales y sistemáticas que presentan Dyba et al. y que a su vez son una adaptación de un trabajo de Mulrow y Cook.

Característica	Revisión Tradicional	Revisión Sistemática
Preguntas	A menudo son amplias en alcance.	A menudo se enfocan en preguntas de investigación.
Cómo se identifican las investigaciones	Usualmente no se indica y son potencialmente parcializadas.	Se indican las fuentes (bases de datos de búsqueda) y estrategias de búsquedas.
Selección de investigaciones	Usualmente no se indica y es potencialmente parcializado.	Selección basada en criterios y son uniformemente aplicadas.
Apreciación	Variable.	Valoración crítica y rigurosa.
Síntesis	A menudo son solo resúmenes cualitativos.	Síntesis cuantitativa y/o cualitativa.
Inferencias	A veces basadas en evidencias.	Usualmente basadas en evidencias.

Tabla 2.7: Diferencias entre revisiones tradicionales y sistemáticas [5]

A diferencia de las revisiones tradicionales, las revisiones sistemáticas permiten encontrar la mejor evidencia disponible a través de métodos para identificar, valorar y sintetizar estudios relevantes de manera rigurosa en un tema de investigación en particular. Los métodos que se emplean son previamente definidos y documentados, de tal manera que otros investigadores puedan evaluarlos de manera crítica y los puedan replicar posteriormente [5].

3. Trabajos Relacionados

Existen publicaciones previas que presentan el concepto de las prácticas ágiles como mejores prácticas a considerar para aumentar la calidad del producto software. En la presente revisión sistemática se han incluido y analizado muchos de esos estudios. Sin embargo, no se encontraron estudios actuales que relacionen y analicen el impacto que tiene el uso de las diferentes prácticas ágiles en la mejora de la calidad. Solamente se pudo identificar un único estudio del año 2009 con un tema de estudio análogo pero con diferencias sustanciales.

El estudio de P. Sfetsos y I. Stamelos [12], presenta una revisión sistemática de la literatura sobre la calidad de las prácticas ágiles en estudios empíricos. Los autores tratan de averiguar el estado actual del conocimiento sobre la calidad en las prácticas ágiles y más específicamente cómo estas logran la calidad a modo general. El estudio recopila literatura solo hasta inicios del año 2009, por lo cual comparándolo con la presente tesis se tienen cinco años de investigación científica no contemplados en dicho estudio. Además utiliza el estándar ISO/IEC 9126 para definir un marco para los temas de calidad, este estándar actualmente se encuentra descontinuado.

La presente revisión sistemática, si bien trata un tema similar, está actualizada al presente año 2014, utiliza estándares actuales para su definición de métricas y conceptos, como es el ISO/IEC 25010, y trata de identificar las practicas ágiles que aportan calidad al producto software y en qué grado lo hacen. Además de tener un detalle y profundidad adecuados para un estudio de tesis de maestría.

Un artículo utilizado como guía en la presente tesis es la revisión sistemática sobre estudios empíricos de desarrollo de software ágil, escrito por Dyba y Dingsoyr en el año 2007 [126]. Este artículo presenta un protocolo de revisión sistemática detallado para estudios similares a los usados en la presente tesis. Aunque el tema tratado es distinto, por ser este muy general, se tomó como una guía base para el desarrollo del protocolo de la presente revisión sistemática.

4. Diseño y Realización de la Revisión Sistemática

En este capítulo se describe el diseño y la ejecución de la revisión sistemática de la literatura. Se detalla la planificación y creación del protocolo a seguir, su implementación, ejecución y el análisis final de los estudios encontrados.

4.1 Diseño de la Revisión Sistemática

Esta investigación se ha llevado a cabo siguiendo las guías de procedimiento de Kitchenham para la realización de una revisión sistemática de la literatura [1]. La revisión sistemática tiene por finalidad resumir toda la información existente acerca de un tema, que responda las preguntas de investigación propuestas. Los métodos empleados para facilitar la realización de la revisión sistemática se especifican en un protocolo de revisión [1] [2].

4.1.1. Protocolo de Revisión

Según Kitchenham, el protocolo de la revisión es un plan detallado para la realización de una revisión sistemática de la literatura y proporciona un método para la selección de los estudios primarios [1]. En esta sección se define el protocolo de revisión que se utilizó para llevar a cabo la investigación. El protocolo se desarrolló con base en las directrices para la realización de una revisión sistemática de la literatura [1] y adaptaciones del modelo propuesto en [7].

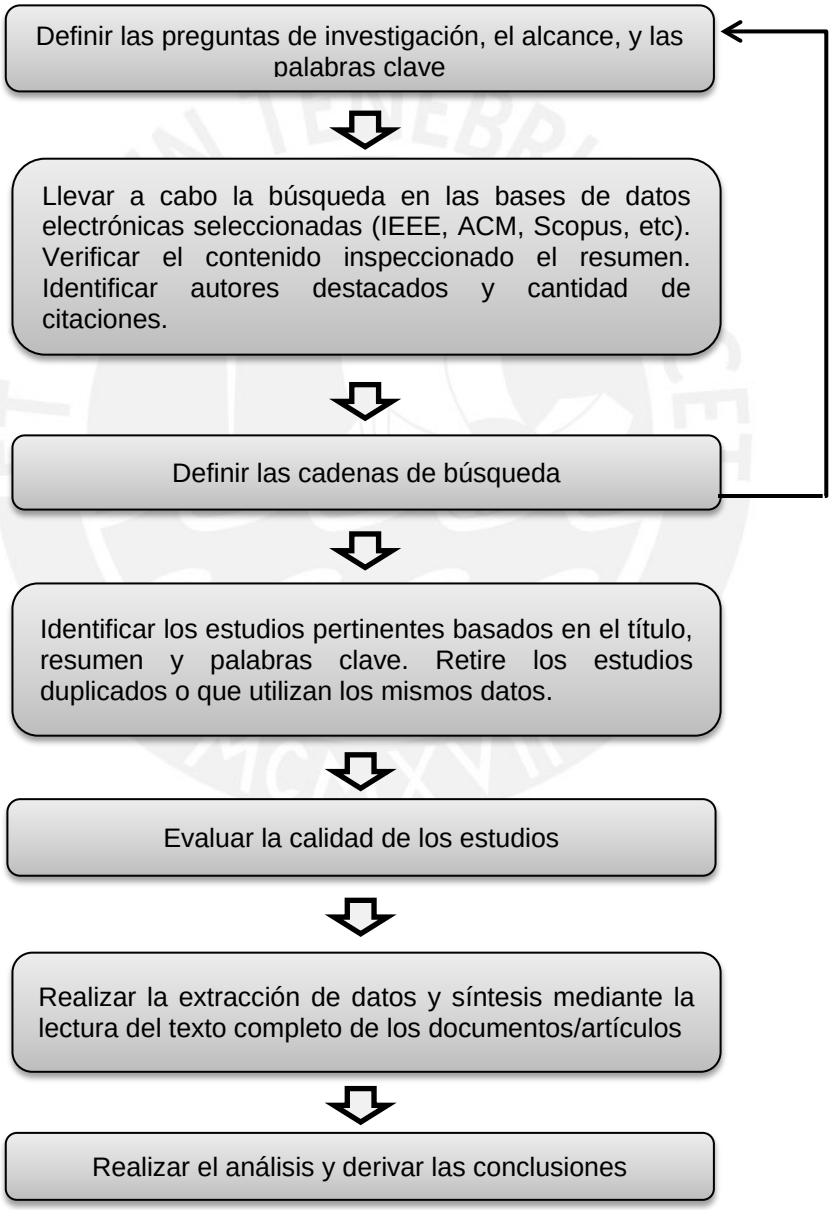

Figura 4.1: Procedimiento de búsqueda

El protocolo de revisión sistemática describe en detalle la metodología a seguir para realizar la búsqueda de los estudios, su selección y posterior análisis. Este procedimiento de búsqueda se muestra en la Figura 4.1.

En primer lugar, se definen las preguntas de investigación y se realiza una búsqueda preliminar en las bases de datos más conocidas, para este estudio se usaron IEEE y ACM. Basado en el resultado de la búsqueda preliminar, se perfeccionan las preguntas de investigación, el alcance, y las palabras clave. Se reformulan las cadenas de búsqueda y la búsqueda se repite en la totalidad de bases de datos escogidas, que incluyen Scopus, ISI Web Science, Science Direct y Springer Link. El objetivo es conseguir documentos iniciales candidatos, tantos como sea posible.

A continuación se utilizan los criterios de inclusión y exclusión para filtrar los documentos más relevantes, estos criterios se basan en las preguntas de investigación. También se identifican autores prominentes en base a la cantidad de citaciones y referencias. Se excluyen las publicaciones duplicadas y las que usan los mismos datos, porque los estudios duplicados podrían sesgar seriamente los resultados. Después, cada estudio se evalúa mediante el uso de un criterio o lista de control de calidad.

La extracción y síntesis de datos se lleva a cabo mediante la lectura del texto completo de todos los trabajos seleccionados que cumplen con los criterios de calidad. Los datos requeridos se extraen y se almacenan inicialmente en un documento de Excel. Se realizan clasificaciones descriptivas con respecto a las conclusiones sobre el contexto de los estudios, las prácticas utilizadas, el entorno, las soluciones recomendadas y conclusiones. La síntesis de datos sirve para derivar las conclusiones a las que llegará el presente estudio.

4.1.2. Preguntas de Investigación

La definición de las preguntas de investigación es la parte más importante para realizar una revisión sistemática, estas permiten dirigir adecuadamente la investigación [1].

Las preguntas de investigación fueron formuladas siguiendo el criterio “PICOC”, cuyas siglas significan *Population* (población), *Intervention* (intervención), *Comparison* (comparación), *Outcomes* (resultados) y *Context* (contexto). Esta perspectiva está detallada en la guía de Petticrew y Roberts para revisiones sistemáticas de ciencias sociales [6]. Cada uno de estos criterios, desde el punto de vista de la ingeniería del software, se pueden definir como sigue [1]:

- *Population*: Es la población que muestra una condición particular en la cual se está interesado. En ingeniería se puede referir a: un rol específico, una categoría, un área de aplicación o un grupo de la industria.
- *Intervention*: Es la metodología de software, herramienta, tecnología, o procedimiento que se ocupa de un tema específico, por ejemplo, las tecnologías para realizar tareas específicas, tales como especificación de requisitos, las pruebas del sistema, o la estimación de costos de software.
- *Comparison*: Es la metodología de la ingeniería de software, herramienta, tecnología, o procedimiento con el que se compara la intervención. El componente de comparación es opcional en el método PICOC.
- *Outcomes*: Son los resultados y deben estar relacionados con factores de importancia para los involucrados, como mejoras de la fiabilidad, reducción de costos, reducción de tiempo de elaboración. Todos los resultados pertinentes deben ser especificados y deben poder ser medibles.
- *Context*: Es el contexto en el cual el estudio toma lugar, los participantes, lo que ayudó u obstaculizó su impacto, es decir aquellos factores que contribuyeron a su éxito o fracaso, así como el proceso de implementación llevado a cabo.

La Tabla 4.1 muestra los resultados que se obtuvieron al aplicar el criterio PICOC al planteamiento de problema a tratar en este estudio.

Criterio	Descripción
***Population* (población)**	Desarrollo de software ágil
***Intervention* (intervención)**	Métodos, técnicas, prácticas ágiles
***Comparison* (comparación)**	No aplica.
***Outcomes* (resultados)**	No definido.
***Context* (contexto)**	Estudios académicos o de la industria de software. Se incluye cualquier tipo de estudio empírico y estudios relativos a temas de calidad.

Tabla 4.1: Aplicación del Método PICOC

El criterio *Comparison* del método no aplica en este estudio, pues el objetivo de este no busca hacer comparaciones entre técnicas, métodos o prácticas ágiles de ningún tipo. De igual manera el criterio *Outcomes* también está vacío pues no se han definido resultados esperados para el presente estudio.

En base al método PICOC, resumido en la tabla anterior, se definen las siguientes preguntas de investigación:

- **Pregunta 1**: ¿Cuál es el estado actual del conocimiento acerca de la calidad en las prácticas ágiles?
- **Pregunta 2**: ¿Cuáles son las prácticas más importantes para el logro de calidad en el desarrollo ágil?

Estas preguntas así definidas permiten conducir la revisión sistemática de la literatura existente del tema en estudio.

4.1.3. Estrategia de Búsqueda

El objetivo de toda revisión sistemática es encontrar la mayor cantidad de estudios primarios posibles relacionados con las preguntas de investigación. Con el fin de cumplir este objetivo, se define una estrategia de búsqueda, la cual se aplicará a las distintas bases de datos electrónicas de estudios científicos.

La estrategia de búsqueda se muestra en la Figura 4.2.

Figura 4.2: Estrategia de búsqueda

La estrategia de búsqueda se define en base a las preguntas de investigación y los resultados obtenidos con el criterio PICOC [6]. Inicialmente se seleccionan las bases de datos donde se realizará la búsqueda y se definen los términos a emplear en la cadena de búsqueda; luego, las fases que tendrá el proceso de búsqueda que incluye el refinamiento de la cadena; y, finalmente, los criterios que se van a tomar para incluir o excluir los artículos encontrados y el almacenamiento de los mismos.

Las siguientes secciones detallan los criterios de búsqueda, así como las bases de datos empleados y los criterios de inclusión y exclusión definidos en la estrategia.

4.1.4. Fuentes de Datos

La estrategia de búsqueda incluyó artículos científicos de bases de datos electrónicas y búsquedas manuales de actas de congresos. Se emplearon las siguientes bases de datos:

- IEEE Xplore (www.ieeexplore.ieee.org)
- Elsevier Science Direct (www.sciencedirect.com)
- ISI Web of Science (apps.webofknowledge.com)
- Scopus (www.scopus.com)
- ACM Digital Library (dl.acm.org)

Para la selección de las bases de datos se examinaron reportes previos sobre revisiones sistemáticas en el campo de la ingeniería de software y se descubrió que la mayor cantidad de resultados útiles se habían encontrado en las bases de datos de IEEE, Science Direct y ACM. Por lo tanto, en un principio se seleccionaron estas bases de datos para las búsquedas. Otra de las razones para la elección de IEEE es que es la mayor asociación innovadora por excelencia en el campo de tecnología. ACM sigue siendo la base de datos más grande del mundo en informática. Finalmente Scopus e ISI se utilizaron debido al renombre que tienen dentro de la comunidad científica.

Para las búsquedas preliminares se planeó usar Google Scholar pero se descartó por la gran cantidad de resultados poco relacionados que devuelve, en su mayoría literatura gris. A su vez, se realizaron búsquedas en las siguientes actas de congresos: *XP*, *Agile Development Conference*, y *XP/Agile Universe*.

4.1.5. Criterios de búsqueda

Los resultados de la búsqueda están muy influenciados por la base de datos y las palabras clave utilizadas en dicha búsqueda. Estas palabras clave derivan directamente de las preguntas de investigación definidas en la sección 4.1.2, tomando en cuenta la información obtenida al aplicar el criterio PICOC (ver Tabla 4.1). Los términos de búsqueda así definidos se describen en la Tabla 4.2.

Los términos utilizados están en el idioma inglés, debido a que la mayoría de las publicaciones científicas aceptan y difunden manuscritos sólo en inglés.

Criterio	Descripción
***Population* (población)**	agile practice / agile software development
***Intervention* (intervención)**	pair programming, planning game, on site customer, test driven development, test first development
***Comparison* (comparación)**	-
***Outcomes* (resultados)**	-
***Context* (contexto)**	empirical, quality

Tabla 4.2: Términos de búsqueda derivados de PICOC

En el criterio *Population* se escogieron términos comúnmente utilizados para referirse a desarrollos de software con metodologías ágiles, y que se consideran sinónimos entre sí.

El criterio *Intervention* se detalló con una lista de prácticas ágiles más significativas escogidas en base a una primera búsqueda preliminar para validar los resultados que mostraban las bases de datos (ver estrategia de búsqueda en 4.1.3). Estas prácticas se explican a detalle en las definiciones de métodos ágiles dentro del marco conceptual, en la sección 2.3.

Finalmente en el criterio *Context* se omitió el detalle de la naturaleza de los estudios académicos o de la industria, pues esto disminuía los resultados encontrados notablemente, omitiendo quizá, algún estudio de interés. El único criterio definido es que sean estudios empíricos relacionados a temas de calidad.

Inicialmente, además de las palabras clave se utilizaron siglas, como PP para *Pair Programming* o TDD para *Test Driven Development*. Sin embargo, la búsqueda inicial incluyendo dichas siglas devolvió un gran número de estudios irrelevantes, finalmente se decidió quitarlas de la cadena de búsqueda.

Las cadenas de búsqueda básicas creadas a partir de los términos *Population*, *Intervention* y *Context* de la Tabla 4.2 se muestran a continuación:

(C1) agile software development **AND** quality

(C2) agile practice **AND** empirical **AND** quality

(C3) pair programming **OR** planning game **OR** on site customer **OR** refactoring **OR** test driven development **OR** test first development **AND** empirical **AND** quality

Estas cadenas básicas se pueden usar para hacer búsquedas independientes en las bases de datos, pues cada una está elaborada mediante combinaciones de los criterios PICOC. Estas cadenas ya se han refinado, luego de una búsqueda preliminar, para devolver la mayor cantidad de estudios relevantes posibles; tal como se definió en la estrategia de búsqueda en la sección 4.1.3.

Las cadenas básicas se pueden combinar utilizando el operador “OR” para hacer una sola búsqueda evitando así los resultados repetidos entre cadenas. Esto quiere decir que los artículos deberían contener términos de cualquiera de las tres cadenas básicas. Finalmente la cadena empleada en las búsquedas es la siguiente:

(C1) **OR** (C2) **OR** (C3)

La cadena de búsqueda varía ligeramente en su sintaxis dependiendo de los requerimientos de la interfaz de búsqueda de la base de datos a consultar. Las cadenas utilizadas para cada una de las bases de datos electrónicas seleccionadas en la sección 4.1.4 se enumeran en la Tabla 4.3.

Base de datos	Cadena de búsqueda	Campos buscados
IEEE Xplore	((agile software development) AND quality) OR ((agile practice) AND empirical AND quality) OR (((((“test driven” OR “test first” AND development) OR refactoring) OR “planning game”) OR “on site customer”) AND empirical AND quality)	Metadata
Elsevier Science Direct	TITLE-ABS-KEY((agile software development AND quality) OR (agile practice AND empirical AND quality) OR (("test driven development" OR "test first development" OR refactoring OR "planning game" OR "on site customer") AND	Abstract, title, keywords

	empirical AND quality))	
ISI Web of Science	(TS=((“agile software development” AND quality) OR (“agile practice” AND empirical AND quality) OR ((“test driven development” OR “test first development” OR refactoring OR “planning game” OR “on site customer”) AND empirical AND quality)))	Topic
Scopus	TITLE-ABS-KEY((“agile software development” AND quality) OR (“agile practice” AND empirical AND quality) OR ((“test driven development” OR “test first development” OR refactoring OR “planning game” OR “on site customer”) AND empirical AND quality))	Abstract, title, keywords
ACM Digital Library	(“Abstract”:”agile software development” AND “Abstract”:”quality”) OR (“Abstract”☹agile practice) AND “Abstract”: “empirical” AND “Abstract”:”quality”) OR ((“Abstract”☹test driven development) OR “Abstract”☹test first development) OR “Abstract”:”refactoring” OR “Abstract”:”planning game” OR “Abstract”:”on site customer”) AND “Abstract”:”empirical” AND “Abstract”:”quality”)	Abstract

Tabla 4.3: Cadenas de búsqueda por base de datos

4.1.6. Fases del proceso de búsqueda

El proceso de búsqueda comprendió dos fases: búsqueda primaria y búsqueda secundaria.

- **Búsqueda primaria**: Comprende la búsqueda de artículos utilizando la cadena de búsqueda definida en la sección anterior, además comprende una etapa de búsqueda preliminar para refinar dicha cadena de búsqueda. Este paso se repitió para cada una de las bases de datos especificadas en la sección 4.1.4. La “literatura gris” fue excluida de las búsquedas por dos razones: es difícil evaluar su calidad y la búsqueda devuelve un gran número de resultados irrelevantes [8].
- **Búsqueda secundaria**: Consiste en la revisión de las referencias y citaciones de los artículos obtenidos de la búsqueda primaria con el fin de

encontrar los artículos relevantes del tema en estudio. Adicionalmente a las bases de datos empleadas para la búsqueda primaria, se empleó Google Scholar solo para determinar citaciones a los artículos seleccionados. La ventaja de Scholar Google es que se obtienen mayor cantidad de citaciones que al emplear Scopus o ISI Web of Science, porque estas dos últimas se centran en indicar las citaciones solo de aquellos artículos que estas mismas bases de datos tienen indexados [5].

4.1.7. Selección de artículos

Para la selección de artículos científicos, se siguieron las pautas de MacDonnell y Shepperd [14]. Para ello, en primer lugar se revisaron los títulos y resúmenes de los artículos encontrados para determinar si deberían ser considerados como estudios primarios. En el caso de no poderse tomar la decisión solo con la información del resumen, se tendría que leer el artículo completo.

4.1.8. Criterios de Inclusión y Exclusión

La realización de una búsqueda en las bases de datos de artículos científicos puede encontrar gran número de estudios, los cuales no necesariamente son relevantes para el tema que se desea investigar. Por este motivo se requiere definir criterios de inclusión y exclusión que definan las pautas a seguir para determinar qué artículos son relevantes o no a esta investigación.

Los estudios candidatos a responder las preguntas de investigación propuestas, se seleccionaron utilizando los siguientes criterios de inclusión y exclusión:

Criterios de inclusión: Para seleccionar los estudios primarios en la presente revisión sistemática, se consideraron los siguientes criterios de inclusión:

- Los estudios deben proporcionar datos empíricos sobre cuestiones de calidad y métricas en prácticas ágiles.
- Los estudios pueden ser académicos y de la industria.
- Los estudios de investigación cuantitativa y cualitativa publicados hasta 2014, sin una fecha de inicio específica.

- Los estudios deben estar escritos en el idioma Inglés.

Criterios de exclusión: Quedarán excluidos los siguientes tipos de estudios:

- Los estudios que no se centran en los problemas de calidad en las prácticas ágiles.
- Los estudios que usan metodologías de desarrollo diferente a la ágil.
- Los estudios que no reportan datos empíricos.
- Los estudios que presentan sólo la opinión del investigador, solo "lecciones aprendidas" y los estudios de solo simulación.

4.1.9. Evaluación de la calidad

Según Kitchenham [1], es necesaria una evaluación de la calidad detallada de los estudios que ya pasaron los criterios de selección de la revisión sistemática de la literatura. Para ello se ha elaborado una lista de verificación de criterios de calidad, tomando como base los criterios más importantes que se espera deba cumplir cada uno de los estudios a analizar y las recomendaciones en [9] [10].

La lista de verificación contiene 10 criterios de calidad, los cuales cubren las tres principales características de calidad que deben tenerse en cuenta al evaluar los estudios en una revisión sistemática: el rigor, la credibilidad y la relevancia.

El Rigor, debe responder a la pregunta: "¿El estudio sigue un enfoque riguroso y adecuado en la aplicación de las diversas metodologías utilizadas?". Se desarrollaron tres criterios para evaluar la justificación, objetivos y contexto del estudio, para lo cual se debe responder a las siguientes preguntas secundarias:

1. ¿El artículo presenta un estudio empírico?
2. ¿Los objetivos de la investigación se han declarado con claridad?
3. ¿Está el contexto del estudio descrito adecuadamente?

Además, se utilizaron otros cuatro criterios para evaluar la validez de la recopilación de datos, los métodos de análisis, y la fiabilidad de los resultados. Estos criterios responden a las siguientes subpreguntas:

4. ¿Es el diseño de la investigación apropiado para abordar los objetivos de la investigación?
5. ¿Es la estrategia de muestreo adecuada para los objetivos de la investigación?
6. ¿Existe un grupo de control con el cual comparar procedimientos?
7. ¿Es el análisis de datos lo suficientemente riguroso?

La Credibilidad, debe responder a la pregunta: "¿Son los resultados del estudio válidos, útiles y bien presentados?". Se desarrollaron otros dos criterios para evaluar si los resultados del estudio son válidos y significativos. Estos criterios responden a las siguientes subpreguntas:

8. ¿La participación de investigador afecta a los resultados ("causa sesgo")?
9. ¿Hay una exposición clara de los resultados?

La Relevancia, debe responder a la pregunta: ¿Son los hallazgos relevantes y útiles para la industria del software y la comunidad de investigación? Un criterio adicional fue desarrollado para evaluar la relevancia de un estudio.

10. ¿El estudio proporciona valor para la investigación o en la práctica?

Cada uno de estos diez criterios se califican de modo booleano '1/0' (Si/No). La suma total de las calificaciones de los criterios se utiliza como medida de confianza para la clasificación de la calidad de cada estudio evaluado, siendo aceptados solo los que tengan la calificación más alta.

4.1.10. Estrategia de Extracción de Datos

En esta etapa se diseña un formulario con el fin de registrar la información obtenida a partir de los estudios primarios [1]. De cada documento primario se extrae la información más importante mediante un formulario de extracción de datos predefinido. Los datos extraídos de cada estudio potencial implican un poco de información general y cierta información específica.

El contenido del formulario de extracción de datos se definió como sigue:

1) Título del Estudio
2) Autor
3) Año de publicación
4) Resumen
5) Tipo de estudio
6) Método ágil usado
7) Entorno del estudio
8) Fecha de la extracción
9) Base de datos encontrada
10) Resultados obtenidos

Estos se almacenan en un inicio en una hoja de Excel, para luego pasarse a algún gestor de referencias bibliográfico que pueda incluir el artículo completo.

4.2 Ejecución de la Revisión Sistemática

La búsqueda se realizó entre los meses de agosto y noviembre del año 2014. Se utilizó la cadena de búsqueda definida en 4.1.5, tomando en cuenta las variaciones de sintaxis para cada una de las bases de datos seleccionadas en la sección 4.1.4.

Al realizar la búsqueda se pudo notar diferencias en las interfaces de las distintas bases de datos. Algunas de las bases de datos ofrecen interfaces fáciles de usar, mientras que otras ofrecen funcionalidades y herramientas adicionales que pueden ser de gran ayuda. Scopus e ISI Web of Knowledge son las que proveen mejores herramientas para la búsqueda y extracción de resultados; IEEE Xplore presenta filtros avanzados que se ejecutan sobre los resultados de modo local, lo cual disminuye el tiempo de respuesta, además de un mejor manejo grupal de estos; finalmente ACM Digital Library mostró una restricción al intentar descargar los resultados en formatos como BibTex o archivos de texto.

Para el ingreso de la cadena de búsqueda, en la base de datos IEEE Xplore, se utilizó la opción *Advanced Search Options* y la pestaña *Command Search*, pues el intentar hacer la búsqueda de cualquier otro modo devolvía pocos resultados. Del mismo modo, en la base de datos Science Direct se utilizó la opción *Advance Search / Expert Search*, seleccionando como filtro *Journal* del área de *Computer Science y Enginering*.

En las demás bases de datos la cadena se usó sin mayor complicación utilizando la opción de búsqueda avanzada y escogiendo solo resultados en inglés.

4.2.1. Resultados de la búsqueda y selección de artículos

De las cinco bases de datos consultadas se recuperaron un total de 780 artículos, siguiendo la búsqueda primaria detallada en la estrategia de búsqueda (ver 4.1.3). Luego como primer paso se procedió a eliminar los resultados repetidos entre bases de datos, hasta este punto no se hizo ningún análisis de contenido. Este proceso devolvió un total de 605 artículos. Los resultados de la búsqueda separados por base de datos se muestran en la Tabla 4.4.

Base de datos	Artículos encontrados	Artículos duplicados	Artículos no duplicados
IEEE Xplore	458	0	458
Elsevier Science Direct	50	11	39
ISI Web of Science	72	36	36
Scopus	54	36	18
ACM Digital Library	146	92	54
Total	**780**	**175**	**605**

Tabla 4.4: Resultados por bases de datos

La Figura 4.3 muestra la cantidad de artículos que fueron encontrados en cada base de datos. Como se puede observar, IEEE Xplore muestra la mayor cantidad de artículos resultantes, esto se debió al modo poco estricto de ingresar la cadena de búsqueda (poco uso de comillas). Esto se hizo de modo intencional pues se identificó que en esta base de datos en particular se obviaban varios artículos interesantes si se utilizaban los términos de la cadena de búsqueda de modo restrictivo para palabras compuestas exactas. El segundo lugar en artículos se obtuvo de ACM Digital Library, en las demás bases de datos los resultados fueron en promedio similares.

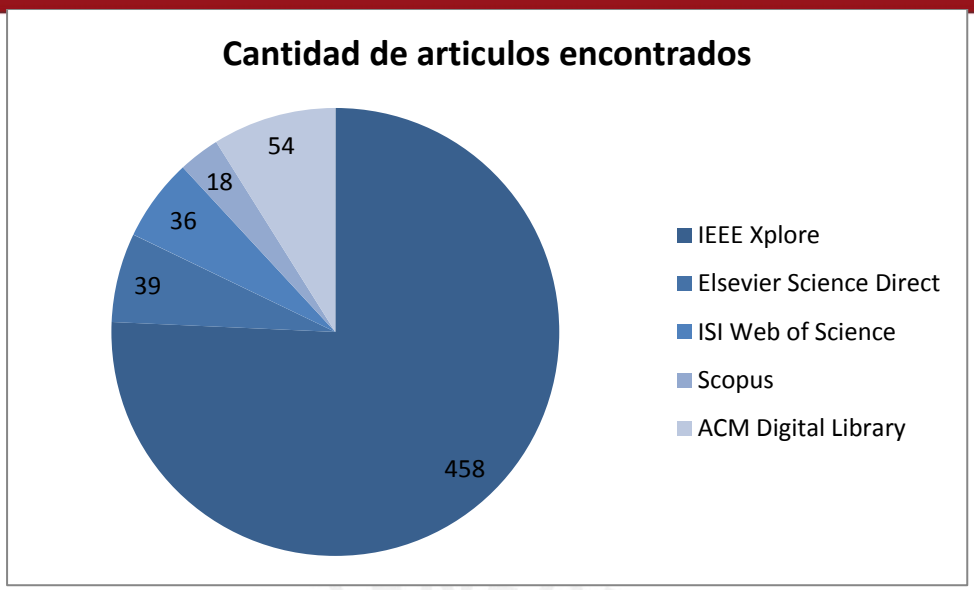

Figura 4.3: Cantidad de artículos encontrados por base de datos

Una vez terminada la búsqueda y eliminados los artículos duplicados se procedió a la selección de artículos relevantes sobre estos **605** resultados. Para ello se analizaron títulos y resúmenes, y se aplicó los criterios de inclusión y exclusión, siguiendo las pautas detalladas en las secciones 4.1.7 y 4.1.8.

Luego de esta etapa la cantidad de artículos que cumplieron todas las observaciones de la revisión sistemática se redujeron a **149**. Estos artículos seleccionados se insertaron en un gestor de bibliografía y se consideraron para la etapa de evaluación de la calidad descrita en la sección 4.1.9. La Figura 4.4 muestra los 149 estudios seleccionados separados por bases de datos.

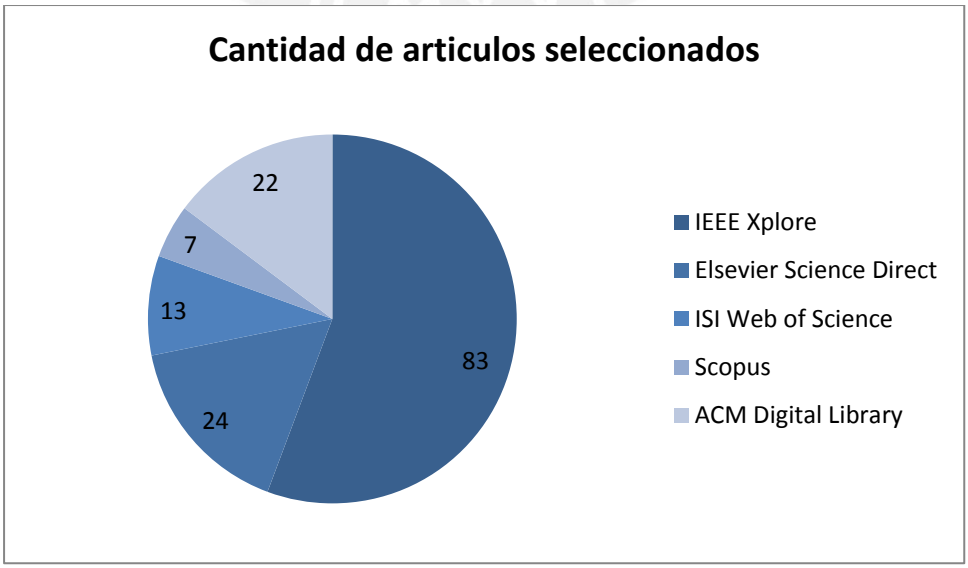

Figura 4.4: Cantidad de artículos seleccionados por base de datos

A los 149 artículos seleccionados se procedió a aplicarles la evaluación de calidad descrita en 4.1.9, donde se considera el rigor, la credibilidad y la relevancia de lo reportado en dichos estudios. Luego de este procedimiento sólo **61** artículos pasaron dicho proceso selectivo, siendo estos los estudios definitivos que se pasaran a analizar en la revisión sistemática. Los resultados detallados de la evaluación de calidad de los estudios aprobados, se muestran en la tabla de calidad del Apéndice A.

La Figura 4.5 resume el proceso seguido y los resultados obtenidos.

Figura 4.5: Estudios primarios luego de cada paso de selección

4.2.2. Extracción de Datos

La extracción de datos se llevó a cabo en los 61 estudios que pasaron el proceso de evaluación de la calidad. Los datos de cada uno de los estudios se registraron primero siguiendo la estrategia de extracción de datos detallada en la sección 4.1.9 en un documento de Excel.

Esta forma ayudó a extraer los detalles primordiales de cada uno de los estudios, para poder realizar la síntesis y mostrar los resultados de la revisión sistemática. Los artículos seleccionados se muestran en la Tabla 4.5.

Cód	Título	Autor	Año	Tipo de Estudio	Ref
E01	The Case for Collaborative Programming	J. T. Nosek	1998	Experimento	[63]
E02	Strengthening the Case for Pair Programming	L. Williams et al.	2000	Experimento	[64]
E03	Integrating Unit Testing Into A Software Development Team's Process	R. Ynchausti	2001	Estudio de caso	[65]
E04	Experimental Evaluation of Pair Programming	J. Nawrocki and A. Wojciechowski	2001	Experimento	[66]
E05	Experiment about Test-First programming	M..M. Müller, and O. Hagner	2002	Experimento	[67]
E06	The Effects of Pair-Programming on Performance in an Introductory Programming Course	C. McDowell et al.	2002	Experimento	[68]
E07	Towards empirical evaluation of TDD in a university environment	M. Pancur et al.	2003	Experimento	[69]
E08	Test-Driven Development as a Defect-Reduction Practice	L. Williams et al.	2003	Estudio de caso	[70]
E09	Assessing test-driven development at IBM	E.M. Maximilien, and L. Williams	2003	Estudio de caso	[71]
E10	Using Test Driven Development in the Classroom	S. H. Edwards	2003	Experimento	[72]
E11	When Does a Pair Outperform Two Individuals?	K.M. Lui and K.C.C. Chan	2003	Experimento	[73]
E12	Pair programming and pair trading: effects on learning and motivation in a CS2 course	T.H. DeClue	2003	Experimento	[74]
E13	Experiences in learning xp practices	B. Tessem	2003	Estudio de caso	[75]
E14	A formal experiment comparing extreme programming with traditional construction	F. Macias et al.	2003	Experimento	[76]
E15	A structured experiment of test-driven development	B. George and L. Williams	2004	Experimento	[77]
E16	Program quality with pair programming in CS1	B. Hanks et al.	2004	Experimento	[78]
E17	Exploring extreme programming in context	L. Layman et al.	2004	Estudio de caso	[79]
E18	Analyses of an agile methodology implementation	S. Ilieva et al.	2004	Estudio de caso	[80]
E19	A cross-program investigation of students' perceptions of agile	G. Melnik and F. Maurer	2005	Cualitativo Encuesta	[81]
E20	On the effectiveness of the test first approach to programming	H. Erdogmus et al.	2005	Experimento	[82]
E21	A Multiple Case Study on the Impact of Pair Programming on Product Quality	H. Hulkko, and P. Abrahamsson	2005	Estudio de caso	[83]
E22	Two controlled experiments concerning the comparison of pair programming to peer review	M.M. Müller	2005	Experimento	[84]
E23	Comparison of student experiences with plan-driven and agile methodologies	C.A.Wellington et al.	2005	Experimento	[85]
E24	A cross-program investigation of student's perceptions of agile methods	G. Melnik, and F. Maurer	2005	Mixto	[86]

E25	Evaluating the efficacy of TDD: industrial case	T. Bhat, and N. Nagappan	2006	Estudio de caso	[87]
E26	Results from introducing component level test automation and TDD	L.O. Damm and L. Lundberg	2006	Estudio de caso	[88]
E27	Pair programming improves student retention, confidence, and quality	C. McDowell, et al.	2006	Estudio de caso	[89]
E28	Critical Personality Traits in Successful Pair Programming	J. Chao and G. Atli	2006	Encuesta Experimento	[90]
E29	Is External Code Quality Correlated with Programming Experience or Feelgood	L. Madeyski	2006	Experimento	[91]
E30	Pair programming productivity: novice–novice vs. expert–expert	K.M. Lui and K.C.C. Chan	2006	Experimento	[92]
E31	Empirical Validation of Test-Driven Pair Programming in Game Development	S. Xu and V. Rajlich	2006	Estudio de caso	[93]
E32	On the Sustained Use of a TDD Practice at IBM	C. Sanchez et al.	2007	Estudio de caso	[94]
E33	An experimental evaluation of the effectiveness and efficiency of the TDD	A. Gupta and P. Jalote	2007	Experimento	[95]
E34	Quality impact of introducing component level test automation and TDD	L.O. Damm and L. Lundberg	2007	Estudio de caso	[96]
E35	Evaluating Pair Programming with Respect to Complexity and Expertise	E. Arisholm et al.	2007	Experimento	[97]
E36	Evaluating performances of pair designing in industry	G. Canfora et al.	2007	Experimento	[98]
E37	Do programmer pairs make different mistakes than solo programmers?	M.M. Müller	2007	Experimento	[99]
E38	Realizing quality improvement through TDD	N. Nagappan et al.	2008	Estudio de caso	[100]
E39	Does TDD Really Improve Software Design Quality?	D. Janzen and H. Saiedian	2008	Experimento Estudio de caso	[101]
E40	Pair programming: what's in it for me?	A. Begel and N. Nagappan	2008	Cualitativo Encuesta	[102]
E41	Pair programming in software development teams	T. Bippet al.	2008	Estudio de caso	[103]
E42	A Case Study on the Impact of Refactoring on Quality and Productivity in an Agile Team	R. Moser et al.	2008	Estudio de caso	[104]
E43	Does TDD Improve the Program Code? Alarming Results from a Comparative Study	M. Siniaalto and P. Abrahamsson	2008	Estudio de caso	[105]
E44	Empirical investigation towards the effectiveness of TFD	L.Huang and M. Holcombe	2009	Experimento	[106]
E45	Implications of integrating TDD into CS1/CS2 curricula	C. Desai, and D. S. Janzen	2009	Experimento	[107]
E46	An Experimental Investigation of Personality Types Impact in Pair Programming	P. Sfetsos et al.	2009	Experimento	[108]
E47	A quantitative approach for evaluating the effectiveness of refactoring in software development process	K. Usha et al.	2009	Estudio de caso	[109]

E48	Distributed agile: project management in a global environment	S. Lee and H.S Yong	2010	Estudio de caso	[110]
E49	Evaluating the Effect of Agile Methods on Software Defect Data and Defect Reporting	K. Korhonen	2010	Estudio de caso	[111]
E50	The impact of test-first programming on branch coverage and mutation score indicator of unit tests	L. Madeyski	2010	Experimento	[112]
E51	A comparative Analysis of the Agile and Traditional Software Development Productivity	W. Carvalho, et al.	2011	Estudio de caso	[113]
E52	Impact of test-driven development on productivity, code and tests	M. Pančur and M. Ciglarič	2011	Experimento	[114]
E53	Scrum+ engineering practices: Experiences of three microsoft teams	L. Williams, et al.	2011	Estudio de caso	[115]
E54	The effectiveness of test-driven development: an industrial case study	T. Dogša and D. Batič	2011	Estudio de caso	[116]
E55	A Case Study Analyzing the Impact of Software Process Adoption on Software Quality	R. Tufail, and A. A. Malik	2012	Estudio de caso	[117]
E56	Development of Auxiliary Functions: Should You Be Agile?	O. A. Lemos, et al.	2012	Experimento	[118]
E57	Quality of Testing in Test Driven Development	A. Cauevic et al.	2012	Experimento	[119]
E58	Refactoring with Unit Testing: A Match Made in Heaven?	F. Vonken and A.Zaidman	2012	Experimento	[120]
E59	A Replicated Experiment on the Effectiveness of Test-first Development	D. Fucci and T. Burak	2013	Experimento	[121]
E60	An experimental evaluation of TDD vs. test-last development with industry professionals	H. Munir, et al.	2014	Experimento	[122]
E61	Systematic analyses and comparison of development performance and product quality of Incremental and Agile Process	A. Tarhan and S. G. Yilmaz	2014	Estudio de caso	[123]

Tabla 4.5: Listado de los estudios seleccionados

4.2.3. Síntesis de los resultados y consideraciones

El estudio siguió el método de síntesis conceptual el cual reúne diferentes entendimientos o conceptos con el propósito de crear un nuevo concepto o conceptos. El enfoque más conocido de este método es el enfoque Meta-etnográfico, el cual combina los resultados de diferentes estudios para crear un entendimiento, del fenómeno en estudio, mayor al de los estudios etnográficos individuales [11].

Para identificar y evaluar los aspectos de calidad en los estudios primarios analizados en esta revisión sistemática, se han considerado dos estándares

industriales reconocidos, el ISO/IEC 12207 [21] y el ISO/IEC 25010 [22]. La norma ISO/IEC 12207 proporciona un marco para los procesos del ciclo de vida del software, tal como se detalló en la sección 2.5.4. De esta norma, solo se utilizaron las definiciones de actividades del área de proceso de desarrollo, pues la mayoría de las prácticas ágiles pueden ser mapeadas directamente a las actividades de esta área de proceso. Por ejemplo, prácticas como *Planning game* y la planificación del *Sprint* incluyen actividades que pueden ser mapeadas directamente a actividades de definición de requisitos, del área de proceso de desarrollo; mientras que *Test driven development*, integración continua y programación en pares se pueden corresponder con actividades de implementación y pruebas de la misma área de proceso de desarrollo.

La norma ISO/IEC 25010 [22], especifica características y subcaracterísticas de calidad, así como métricas asociadas, con la intención de garantizar la calidad del producto software. Las métricas externas son aplicables al software en ejecución, mientras que las métricas internas son las que no dependen de la ejecución del software (medidas estáticas). Las métricas de calidad en el uso sólo están disponibles cuando el producto final se utiliza en condiciones reales. Esta revisión sistemática consideró estos tres tipos de métricas (externas, internas y de calidad en el uso) al evaluar la calidad de los estudios.

Siguiendo este enfoque, los resultados se interpretan y comparan, revelando las diferentes prácticas ágiles utilizadas para medir y asegurar la calidad, en los estudios seleccionados. Finalmente, los resultados fueron interpretados, comparados, agrupados y sintetizados con la finalidad de responder a las preguntas de investigación.

4.3 Resultados

De los **61** estudios empíricos considerados en la revisión sistemática se encontró que, 31 eran experimentos, 25 eran estudios de casos y 5 fueron estudios mixtos (experimento/estudio de caso, experimento/encuesta, estudios de caso/encuesta y encuesta/ estudio cualitativo). La Figura 4.6 muestra esa distribución de resultados clasificados por tipo de estudio.

Los estudios seleccionados cubren una amplia gama de temas de investigación y fueron realizados en diferentes contextos, que van desde proyectos profesionales de empresas de desarrollo de software, hasta experimentos académicos en cursos universitarios. Todos estos detalles y clasificaciones se pueden consultar, para cada estudio seleccionado, en el Anexo B.

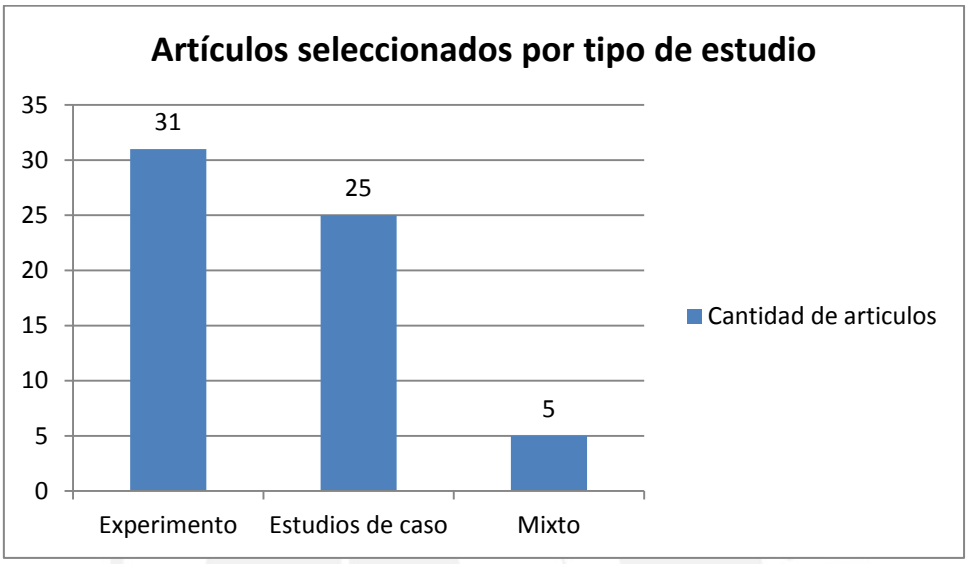

Figura 4.6: Cantidad de artículos seleccionados por tipo de estudio

Los estudios se analizaron y clasificaron en cinco categorías: calidad en desarrollos conducidos por pruebas (*test driven development*), calidad en programación en pares (*TDD y TFP*), calidad usando XP en su totalidad, calidad usando Scrum y calidad en otras prácticas ágiles. Los tipos de estudios clasificados por categorías se resumen en la Figura 4.7.

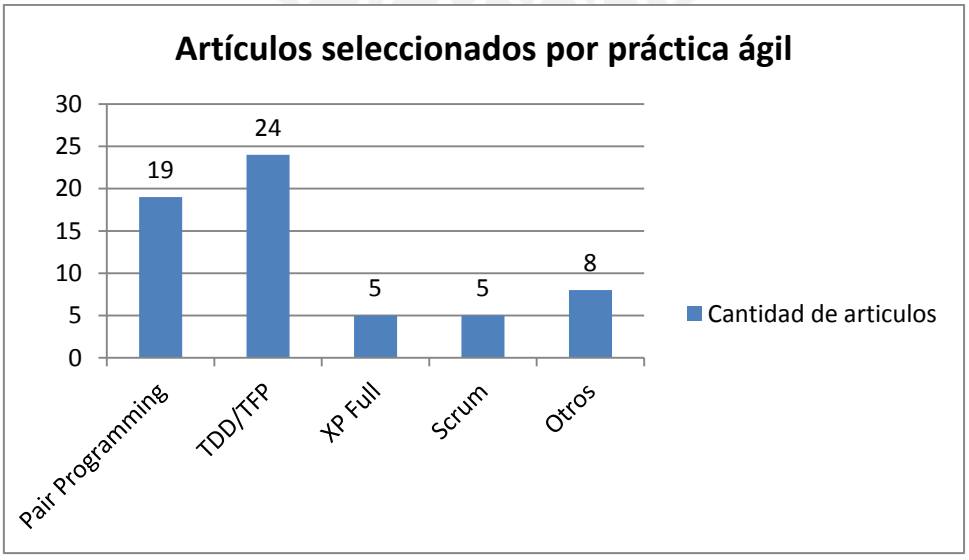

Figura 4.7: Cantidad de artículos seleccionados por método ágil utilizado

Un análisis clasificando a detalle los tipos de estudios por práctica ágil se muestra en la Tabla 4.6 y Figura 4.8.

	Experimento	Estudio de caso	Mixto	*Total*
Pair Programming	14	3	2	19
TDD/TFP	12	10	2	24
XP Full	2	2	1	5
Scrun	0	5	0	5
Otras prácticas	3	5	0	8
Total	31	25	5	**61**

Tabla 4.6: Clasificación de tipos de estudios seleccionados por método ágil

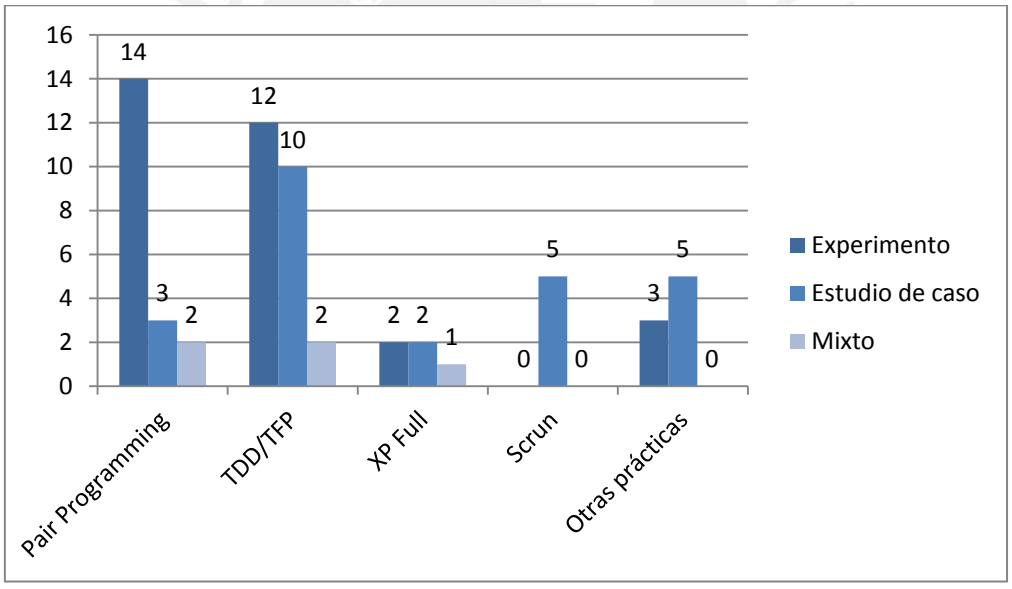

Figura 4.8: Gráfico de tipos de estudios seleccionados por método ágil

La distribución de estudios por año de publicación se muestra en la Figura 4.9. Como se puede observar el estudio más antiguo es del año 1998, siendo incluso previo al manifiesto ágil. El estudio en cuestión es un experimento para evaluar los beneficios de la programación en pares (en este estudio llamado *Collaborative Programming*). Los estudios más recientes llegan a incluir estudios publicados hasta el año 2014.

Figura 4.9: Cantidad de estudios por año de publicación

Para abordar las preguntas de investigación se analizaron los estudios separándolos en las categorías por tipos de metodología ágil utilizada, tal como se especifica en la Figura 4.7, y siguiendo las consideraciones para la síntesis de datos, detallada en 4.2.3.

La Tabla 4.7 muestra cómo se distribuyen los estudios para las distintas categorías ágiles definidas, donde se puede apreciar la predominancia del desarrollo guiado por pruebas y la programación en pares siendo estas las prácticas más importantes.

Categorías	Estudios	Porcentaje
Programación en pares	[63], [64], [66], [68], [73], [74], [78], [83], [84], [89], [90], [91], [92], [97], [98], [99], [102], [103], [108]	31.1%
Desarrollo guiado por pruebas	[65], [67], [69], [70], [71], [72], [77], [82], [86], [87], [88], [94], [95], [96], [100], [101], [105], [106], [107], [112], [116], [119], [121], [122]	39.3%
Extreme programming	[76], [79], [80], [81], [85], [104]	9.8%
Scrum	[110], [113], [115], [117], [123]	8.3%
Otras prácticas ágiles	[75], [93], [109], [111], [114], [118], [120]	11.5%

Tabla 4.7: Estudios separados por categorías ágiles

4.2.4. Calidad en la Programación en Pares (PP)

La programación en pares, ha sido el segundo método más utilizado en los estudios reportados en la presente revisión sistemática (ver Figura 4.7), siendo exhaustivamente investigada y utilizada con éxito en la industria y el mundo académico.

Se encontraron mejoras en la calidad del código y del diseño que van desde un 15% [98] hasta un 65.4% [102], teniendo los resultados positivos más significativos reportados en la mayoría de los estudios: [63], [64], [68], [74], [83], [89], [90], [91], [97], [98], [102], [103], [108].

Para programas complejos, desconocidos y desafiantes, la programación en pares demostró entregar código de mejor calidad que el realizado en solitario [78], [84], [92], [97], [99]. Se reportaron menos defectos en el código en [99], [102], y ninguna diferencia significativa en cuanto a defectos de código en [66]. Se encontró resultados contradictorios para productividad [73], [83] y tiempo gastado [64], [66], [97], [102], [103].

Ciertas combinaciones de parejas de programadores, que tienen en cuenta habilidades, conocimientos y experiencia, pueden aumentar la productividad [92]. Por otra parte, ciertos rasgos de personalidad aseguran código de mejor calidad, tales como la dupla: de “mente abierta” y “responsabilidad” [90], o también en combinaciones de personalidades/temperamentos heterogéneos [108].

Se reportaron muchos otros beneficios como la mejora de la calidad del trabajo en equipo y la comunicación [74], [108], dispersión y mejor comprensión del código [102], [103], mejor información y transferencia de conocimiento [68], [74], mejor y más rápido diseño de algoritmos [73], aumento de la moral [63], y programadores con más confianza [78].

Las desventajas que fueron reportadas incluyen el incremento en esfuerzo [98] y costo [64], especialmente para los productos con mayor calidad [84], una pérdida menor en eficiencia [103], problemas de planificación y conflictos de

personalidad [102]. Problemas de compatibilidad en los pares de programadores se investigaron en algunos de los estudios [90], [92], [108].

4.3.1. Calidad en el Desarrollo Guiado por Pruebas (TDD)

El desarrollo guiado por pruebas fue la práctica más utilizada en los estudios seleccionados en la presente revisión sistemática, estos estudios la consideran la práctica más importante para obtener calidad en los desarrollos ágiles.

La calidad externa, en la mayoría de los experimentos [72], [77], [81], [86], [95], [101], [106] y estudios de casos [65], [70], [71], [87], [88], [94], [96], [100], [116] mostraron una mejora significativa. Siendo medida, en los experimentos por el número de pruebas de aceptación aprobadas, el número total de defectos o por la densidad de defectos (#defectos/KLOC). En los estudios de casos y estudios mixtos, la calidad externa se midió usualmente por el número de defectos encontrados antes de la liberación o defectos notificados por los clientes.

Los defectos se redujeron desde un 5% - 45% en [65], [70], [77], [88], [96], [100], [122] hasta un 50% - 90% en [70], [71], [72], [87]. Los estudios de casos mostraron una mayor mejoría en la calidad externa que los experimentos, esto puede ser debido a los ajustes controlados y las limitaciones de tiempo. No se mostraron diferencias significativas en la calidad externa en [67], [69], [112], [119], [121] y [122].

La calidad interna se midió por diferentes métricas del código fuente, como el tamaño del código, la complejidad ciclomática, el acoplamiento y la cohesión. Mejoras de la calidad interna se reportaron en [94], mientras que no se encontraron diferencias significativas en [101] y [107]. Se reportó una disminución en la complejidad del código y diseño en [94], [105], [107] y [119], especialmente para unidades de diseño pequeñas. Un incremento en la reutilización del código se evidenció en [67], mientras que la cohesión del código no mejoró en [101] y [105].

Se reportó un incremento en la calidad en uso en [112] y [116]. Si bien el código se hace menos complejo, en un estudio la estructura de paquetes se hizo más difícil de cambiar y mantener [105]. Un estudio demostró que el costo

total del desarrollo se redujo, debido a la disminución del costo de fallas evitables [88]. Tres estudios incluyeron el esfuerzo como una variable de la investigación [95], [106] y [122]. Uno de estos estudios mostró que el esfuerzo para las pruebas se incrementó [106], mientras que otro mostró que el esfuerzo total para el desarrollo se redujo [95]. Dos estudios de caso mostraron que el tiempo de desarrollo se incrementó [87], [100].

Los estudios de productividad encontraron resultados contradictorios. Dos experimentos reportaron un aumento en la productividad [95], [106], Dos estudios de caso reportaron una disminución en la productividad [94], [116], mientras que no se mostraron diferencias significativas en [71] y [121].

4.3.2. Calidad en Extreme Programming (XP)

XP como metodología incluye un grupo de prácticas ágiles, que se detallan en 2.3.1; esta sección trata de evaluar el funcionamiento de estas prácticas en conjunto y como aportan al objetivo de conseguir mejorar la calidad.

La mayoría de los estudios seleccionados aplicó prácticas clave de XP en combinación, tales como *plannnig game*, programación en pares, desarrollo dirigido por pruebas y refactorización. Estos estudios ofrecieron una mejora medible de la calidad en [79], [80], [85].

Se encontró que las prácticas de XP en conjunto producen mejores resultados para equipos pequeños en [81], [104]. En un estudio se reportó no encontrar ninguna diferencia, ya sea en calidad interno o externo entre las prácticas de XP y los equipos con metodologías tradicionales [76], tampoco se encontró diferencias en el tamaño del producto.

La productividad se incrementó hasta un 46% en [79], [80], [81] y [104]. Un gran porcentaje de los desarrolladores percibieron una mejora en la calidad del código generado en [81] y se reportó hasta un 13% menos defectos en [80].

4.3.3. Calidad en Scrum

El proceso Scrum reportó mejoras de hasta un 30% en la calidad del producto en un estudio de caso [110]. Se percibió un incremento de la productividad

desde un 16% hasta un 250% en [113], [115] y [123], Las mediciones se hicieron usualmente mediante líneas de código producidas en el ciclo sprint a medir. En un caso de estudio se encontró una marcada mejora de la productividad solo en proyectos medianos y pequeños [113].

Se reportó una disminución en la densidad de defectos de hasta 57% y una disminución del esfuerzo de hasta un 26% en [115], [123]. Se reportó una reducción de errores y defectos de distinta gravedad en [117], [123]. Se incrementó la relación de pruebas pasadas/falladas, hasta casi siete veces en [117].

La relación de esfuerzo de resolución de errores disminuyo un 26%, mientras la eficacia mejoro considerablemente en [123]. Se documentó un incremento en la satisfacción del cliente en [110].

4.3.4. Calidad en otras Prácticas Ágiles

La combinación de refactorización y programación guiada por pruebas ofrecieron un incremento en la calidad con un código más limpio y de mayor cohesión [75], [93].

Planning game reportó una mejora en la estimación del trabajo en [75]. La refactorización mostró un tangible aumento de la calidad y productividad en [93]. Un estudio reportó una mejora en la reutilización y la comprensibilidad del código y el mantenimiento al usar refactorización [109]. El número de defectos se redujo considerablemente en [111], [118] usando combinaciones de TDD y programación en pares.

Mientras que no se encontraron diferencias significativas en cuanto a calidad externa, productividad y pruebas de aceptación al combinar TDD y programación en pares en [114]. El uso de pruebas unitarias durante la refactorización no condujo a resultados consistentes en [120], por lo cual se deduce que no aumentan la calidad en este caso. Se mostró un incremento en el esfuerzo en [118].

4.3.5. Limitaciones de esta revisión sistemática

La validez interna (credibilidad) y validez externa (generalización) de los resultados del estudio se definieron usando las pautas de Lincoln y Guba [124]. A partir de ello se pueden identificar las siguientes limitaciones a esta revisión sistemática: el sesgo en la selección de los estudios, la inexactitud en la extracción de datos y la exclusión de artículos relevantes.

Para minimizar el sesgo en el proceso de selección, se desarrolló un protocolo de investigación (protocolo de la revisión) que define las preguntas de investigación y el proceso de revisión. Sobre la base de este protocolo se desarrolló los términos de búsqueda para la identificación de la literatura relevante. Sin embargo, hay que subrayar que el proceso de búsqueda fue un proceso complicado y no inequívoco, debido a que en los estudios varios términos clave referentes a la calidad no estaban estandarizados de acuerdo con alguna norma de calidad internacional (como el estándar ISO/IEC 25010) o se utilizaban en contextos distintos.

Para minimizar el sesgo de selección se utilizó un proceso de búsqueda y extracción de datos de varias etapas definidas en el protocolo de revisión. El proceso de extracción de datos fue obstaculizado por la forma en que algunos estudios describían su contexto, sus preguntas de investigación, su proceso de muestreo, los métodos y las métricas utilizadas, el proceso de recopilación de datos y sus resultados. Por lo tanto, existe una posibilidad de que algunos de los datos extraídos sean inexactos. La generalización de los resultados puede verse obstaculizado también por las variables no controladas y las métricas utilizadas en algunos de los estudios. Es imposible comparar directamente los resultados de estudios tan variados.

Para reducir el riesgo de excluir artículos relevantes se incluyó una revisión más detallada a aquellos artículos en los cuales se tenían dudas durante la revisión de los resúmenes. Luego parea los estudios candidatos se desarrolló una serie de preguntas para evaluar la calidad de los estudios en si (4.1.9), con esto se logró asegurar que los estudios seleccionados cumplían con el rigor, credibilidad y relevancia necesarios. Sin embargo aún existe la posibilidad de haber pasado por alto algunos estudios relevantes debido a la gran cantidad de resultados devueltos en las búsquedas iniciales.

5. Conclusiones y Trabajos Futuros

En este capítulo se presentan las conclusiones en base al proceso seguido en el transcurso del desarrollo de la presente tesis. Además se presentan sugerencias de trabajos futuros que nacen a partir del estudio realizado, estos indican algunos temas en los cuales se puede seguir investigando y complementar los hallazgos aquí presentados.

5.1 Conclusiones

Este trabajo de tesis presenta los resultados de una revisión sistemática de la literatura que evalúa los enfoques y métricas de calidad en las prácticas ágiles, valiéndose para ello de los estándares definidos en ISO/IEC 25010 e ISO/IEC 12207. Se identificaron 605 estudios, de los cuales 61 resultaron ser estudios empíricos con aceptable rigor, credibilidad y relevancia. Para el análisis en materia de calidad, los estudios se clasificaron en cinco grupos principales: programación en pares, desarrollos guiados por pruebas, extreme programming en su totalidad, scrum y otras prácticas ágiles.

La programación en pares demostró en la mayoría de estudios ser era una práctica ágil exitosa, en términos de calidad. El principal hallazgo de esta práctica es que mejora fuertemente la calidad del código y del diseño. Además, mejora la calidad del trabajo en equipo, mejora la comunicación, la comprensión y la transferencia de conocimientos. En combinación con el desarrollo guiado por pruebas y la refactorización, la programación en pares se convierte en una práctica clave para mejorar la calidad.

Se identificaron una serie de beneficios y limitaciones reportadas por los estudios empíricos usando prácticas ágiles en materia de calidad. Se reportó un amplio rango de mejoras para la programación guiada por pruebas y la refactorización, incluyendo entre otros la mejora de la calidad externa. El desarrollo guiado por pruebas ayudó significativamente a la mejora de la calidad del software en términos de la disminución de la tasa de errores, cuando se emplea en el contexto de la industria. Tal efecto no fue tan claro en los estudios llevados a cabo en entornos académicos, sin embargo ninguno de esos estudios informaron de una disminución de la calidad. Los efectos acerca de la productividad usando el desarrollo orientado a pruebas fueron contradictorios, y sus resultados variaron para diferentes contextos.

El uso de las distintas prácticas de XP en combinación demostró aportar valor al objetivo de conseguir mejorar la calidad, tales como *plannnig game*, programación en pares, desarrollo dirigido por pruebas y refactorización. Estos estudios ofrecieron una mejora medible de la calidad, y resultados más favorables aun al utilizarlas en equipos pequeños. La productividad mostró un considerable incremento reflejándose en una mayor calidad del código generado y una menor cantidad de defectos. Algunos estudios no mostraron mejoras en su análisis de calidad interna y externa pero tampoco mostraron que la calidad disminuya.

El proceso scrum reportó significativas mejoras de la calidad en los estudios de caso y el mayor incremento de productividad en comparación a todas las otras prácticas ágiles. También se detectó una disminución en la densidad de defectos, disminución del esfuerzo, además de un incremento en la satisfacción del cliente. Otras prácticas ágiles como el *planning game*, refactorización y pruebas unitarias son también algunas de las prácticas que contribuyeron activamente a la mejora de calidad.

Se espera que los primeros resultados de este estudio puedan ayudar a los gerentes y a los investigadores en el campo de los métodos ágiles, para comprender mejor la forma de abordar los problemas de calidad en la aplicación de prácticas ágiles.

5.2 Trabajos Futuros

Durante la realización de la presente tesis de maestría se encontraron nuevas áreas para futuras investigaciones, las cuales no han podido ser revisadas, tales como:

Experimentos controlados de comparación: existen pocos estudios en los que se hayan realizado experimentos controlados de comparación entre las prácticas ágiles en ambientes industriales. La mayoría de estudios comparan los beneficios de las prácticas ágiles en comparación a las metodologías tradicionales, pero no cuantifican mediante experimentos empíricos cual práctica ágil ofrece mejores resultados en cuanto a temas de calidad. Esto podría demostrar y resaltar las ventajas e importancia de la aplicación de dichas técnicas.

Elaboración de método de evaluación de atributos de calidad para desarrollos ágiles: en varios estudios analizados para la realización de esta tesis se reporta como un factor importante la falta de consideración del efecto de los atributos de calidad en el planteamiento de un proyecto ágil, siendo esto el causante defectos en el producto software que tiene costos muy elevados para su solución luego de alcanzar cierto avance en el proyecto. Este método deberá poder comprobar la calidad de los distintos atributos de calidad de un modo flexible y actualizable sin quitarle agilidad al proceso, además de poder ser personalizable con base en diferentes tamaños de proyecto y acomodarse a cualquier metodología ágil.

REFERENCIAS

[1] KITCHENHAM, B. “Guidelines for performing Systematic Literature Reviews in Software Engineering”. *Software Engineering Group, Department of Computer Science*, Keele University and Empirical Software Engineering National ICT Australia Ltd, EBSE Technical Report, EBSE-2007-01, 2007.

[2] KITCHENHAM, B. “Procedures for performing systematic reviews”. *Software Engineering Group*, Department of Computer Science, Keele University and Empirical Software Engineering National ICT Australia Ltd, Keele University Technical Report TR/SE-0401, 2004.

[3] BIOLCHINI, J. et al. “Systematic review in software engineering”. *System Engineering and Computer Science Department COPPE/UFRJ,* Technical Report RT – ES, v. 679, n. 05, 2005.

[4] BABAR, M A; ZHANG, H. “Systematic literature reviews in software engineering: Preliminary results from interviews with researchers”. En *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement. IEEE Computer Society*, 2009.

[5] POW SANG PORTILLO, J. A. “*Técnicas para la Estimación y Planificación de Proyectos de Software con Ciclos de Vida Incremental y Paradigma Orientado a Objetos*”. Tesis Doctoral. Informática. Madrid: Universidad Politécnica de Madrid, Facultad de Informática, 2012.

[6] PETTICREW, M; ROBERTS, H. “Systematic reviews in the social sciences: A practical guide*”.* John Wiley & Sons Publishing, 2008.

[7] DYBA, T., KITCHENHAM, B. A., JORGENSEN, M. “Evidence-based software engineering for practitioners. Software”, IEEE, 2005.

[8] Engström, E., Skoglund, M., Runeson, P. "Empirical evaluations of regression test selection techniques: a systematic review." *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, ACM, 2008.

[9] LEEDY, P. D; ORMROD, J. E. “Practical research: Planning and design”, Pearson Merril Prentice Hall, vol. 8, 2005.

[10] SPENCER, L. et al. “Quality in qualitative evaluation: A framework for assessing research evidence”. London: Government Chief Social Researcher’s Office, 2003.

[11] NOBLIT, G. W.; HARE, R. D., “Meta-ethnography: Synthesizing qualitative studies”. Sage Publications, London, 1988.

[12] SFETSOS, P; STAMELOS, I. G. “Empirical studies on quality in agile practices: A systematic literature review”. En *Quality of Information and Communications Technology (QUATIC), 2010 Seventh International Conference on the*. IEEE, 2010.

[13] SFETSOS, P; STAMELOS, I. G. “Agile software development quality assurance”, Igi Global, 2007.

[14] MACDONELL, S., SHEPPERD, S. “Comparing Local and Global Software Effort Estimation Models - Reflections on a Systematic Review”. *Proceedings ESEM 2007*, IEEE Computer Society, EEUU, 2007.

[15] BOEHM, B. "A view of 20th and 21st century software engineering." Proceedings of the 28th international conference on Software engineering. ACM, 2006.

[16] BENEFIELD, G. "Rolling out agile in a large enterprise." *Hawaii International Conference on System Sciences*, Proceedings of the 41st Annual. IEEE, 2008.

[17] NERUR, S. et al, "Challenges of migrating to agile methodologies." *Communications of the ACM 48*, no. 5, 2005.

[18] HUO, M. et al, "Software quality and agile methods." *In Computer Software and Applications Conference*, COMPSAC 2004. Proceedings of the 28th Annual International, IEEE, 2004.

[19] KITCHENHAM, B and PFLEEGER S. L, "Software Quality: The Elusive Target", IEEE Software, Vol. 13, No. 1, 1996.

[20] IEEE, "IEEE 1074:1997 - Standard for Software Life Cycle Processes", 1998.

[21] ISO/IEC, "ISO/IEC 12207: 2008 - Information Technology - Software Life Cycle Processes", 2008.

[22] ISO/IEC, "ISO/IEC 25010:2011 - Systems and Software Quality Requirements and Evaluation (SQuaRE) - System and Software Quality Models", 2011

[23] SOMMERVILLE, I. "Software Engineering. International computer science series", Addison-Wesley, 9th ed, 2011.

[24] ROYCE W. "Managing the Development of Large Software Systems: Concepts and Techniques" *Proceedings of IEEE WESCON 26*, 1970.

[25] RODRIGUEZ G. P. "Estudio de la aplicación de metodologías ágiles para la evolución de productos software." PhD diss., Universidad Politécnica de Madrid, Facultad de Informática, 2008.

[26] ALLIANCE, Agile. "Agile Manifesto" Online at http://www.agilemanifesto.org.

[27] BOEHM, B. W. "Software engineering economics", Prentice Hall, 1st ed, 1981.

[28] GELBWAKS, M. "Why do we need agile". Borland special conference, 2006.

[29] KOSKELA, J. "Software Configuration Management in Agile Methods." VTT Publications, ESPOO, 2003.

[30] COCKBURN, A. and HIGHSMITH, J. "Agile software development: The people factor", IEEE Computer, 2001.

[31] HIGHSMITH, J. "Agile Project Management: Principles and Tools", Cutter Consortium, 2004.

[32] BECK, K. "Planning Extreme Programming". *Addison-Wesley Professional*, 2001.

[33] BECK, K. "Extreme Programming Explained: Embrace Change", *Addison-Wesley*, Reading, MA, 1999

[34] AGILE, Process. "Extreme Programming" Online at http://www.extremeprogramming.org

[35] SCHWABER, K, and BEEDLE, M. "Agile Software Development with Scrum", Prentice Hall, 2001.

[36] SCHWABER, K. "Agile project management with Scrum". Vol. 7. Redmond: Microsoft press, 2004.

[37] SUTHERLAND, J. and SCHWABER, K. “The Scrum Papers: Nuts, Bolts, and Origins of an Agile Process”, Patient Keeper Inc, 2007.

[38] POPPENDIECK, M. and POPPENDIECK, T. “Lean Software Development: An Agile Toolkit”. *Addison-Wesley Professional*, 2003.

[39] POPPENDIECK, M. “Lean Software Development,” in *ICSE COMPANION '07: Companion to the Proceedings of the 29th International Conference on Software Engineering*, 2007.

[40] PETERSEN, K., “Is Lean Agile and Agile Lean? A Comparison Between Two Software Development Paradigms,” *Modern Software Engineering Concepts and Practices: Advanced Approache,* IGI Global, 2010.

[41] HIBBS, C. et al. “Art of Lean Software Development”. *O"Reilly Media, Inc.*, 2009.

[42] COCKBURN, A., “Crystal Clear: A Human-Powered Software Development Methodology for Small Teams”. *Addison-Wesley*, 2001.

[43] COAD, P. et al. "Feature-driven development." Java Modeling in Color with UML, *Prentice Hall*, 1999.

[44] PALMER, S. R. and FELSING, J. M. “A Practical Guide to Feature-Driven Development”, *Prentice Hall*, 2002.

[45] BECK, K. “Test-Driven Development: by example”. *Addison-Wesley Professional*, 2003.

[46] KOSKELA, L. “Test Driven: Practical TDD and Acceptance TDD for Java Developers”, *Manning Publications Co.*, 2007.

[47] FOWLER, M. “Refactoring: Improving the Design of Existing Code”. *Addison-Wesley*, 1997.

[48] FOWLER, M. et al. "Refactoring: Improving the Design of Existing Programs." 1999.

[49] WILLIAMS, L, et al. "Strengthening the Case for Pair Programming." *IEEE software*, 2000.

[50] COCKBURN, A., and WILLIAMS, L. “The Costs and Benefits of Pair Programming”. *Extreme programming examined*, 2000.

[51] AWAD, M. A. "A Comparison Between Agile and Traditional Software Development Methodologies." *University of Western Australia*, 2005.

[52] KITCHENHAM, B. and WALKER, J. "A Quantitative Approach to Monitoring Software Development", *Software Engineering Journal*, 1989.

[53] ISO, “ISO 8402: Quality Vocabulary,” in International Organization for Standardization, Geneva, 1986.

[54] ISO, “ISO 9000: 2005 Sistemas de Gestión de la Calidad: Fundamentos y Vocabulario” en International Organization for Standardization, 2005.

[55] ISO/IEC, "ISO/IEC 9126-1 Software Engineering: Product Quality. part 1: Quality Model" 2001.

[56] KLAS, M. et al. ”CQML scheme: A classification scheme for comprehensive quality model landscapes”. In Proceedings of the 35th Euromicro Conference on Software Engineering and Advanced Applications, 2009.

[57] BOEHM, B.W. et al. “Characteristics of Software Quality”, North-Holland, Amsterdam, 1978.

[58] MCCALL, J.A. et al. ”Factors in Software Quality”. National Technical Information Service, Springfield, 1977.

[59] GRADY, R.B. and CASWELL, D.L. “Software Metrics: Establishing a Company-Wide Program”, Prentice Hall, Englewood Cliffs, 1987.

[60] ISO, 25000. "Portal ISO 25000" Online at http://iso25000.com.

[61] ISO, “ISO 25000: 2011 - Systems and Software Quality Requirements and Evaluation (SQuaRE)”, International Organization for Standardization, 2011.

[62] BUCUR, I. "On Quality and Measures in Software Engineering." *Journal of Applied Quantitative Methods*, 2006.

[63] NOSEK J.T. “The Case for Collaborative Programming,” *Comm. ACM*, Vol. 41, 1998.

[64] WILLIAMS L. et al. "Strengthening the Case for Pair Programming," *IEEE Software*, vol. 17, 2000.

[65] YNCHAUSTI R. A. “Integrating Unit Testing Into A Software Development Team's Process”, *proceedings of the 2nd International Conference on Extreme Programming and Flexible Processes in Software Engineering*, Italy, 2001.

[66] NAWROCKI J. and WOJCIECHOWSKI A. “Experimental Evaluation of Pair Programming”, *proceeding of European Software Control and Metrics*, 2001.

[67] MÜLLER M. and HANGER O. “Experiment about Test-First programming”, *IEEE Proceedings on Software 149*, 2002.

[68] MCDOWELL C. et al. "The Effects of Pair-Programming on Performance in an Introductory Programming Course," *proceedings of the 33rd SIGCSE technical symposium on Computer science education*, 2002.

[69] PANCUR M. et al. ”Towards Empirical Evaluation of Test-Driven Development in a University Environment”, *proceedings of the International Conference on Computer as a Tool*, 2003.

[70] WILLIAMS L. et al. ”Test-Driven Development as a Defect-Reduction Practice”, *proceedings of the 14th International Symposium on Software Reliability Engineering*, USA, 2003.

[71] MAXIMILIEN E.M. and WILLIAMS L. “Assessing Test-Driven Development at IBM”, *proceedings of the 25th International Conference on Software Engineering*, IEEE Computer Society, 2003.

[72] EDWARDS S. H. “Using Test-Driven Development in the Classroom: Providing Students with Automatic, Concrete Feedback on Performance”, *International Conference on Education and Information Systems: Technologies and Applications*, USA, 2003.

[73] LUI K. M. and CHAN K. C. C. “When Does a Pair Outperform Two Individuals?”, *LNCS*, volume 2675/2003, 2003.

[74] DECLUE T. H. “Pair programming and pair trading: effects on learning and motivation in a CS2 course”, *Journal of Computing Sciences in Colleges*, 2003.

[75] TESSEM B. “Experiences in learning xp practices: a qualitative study”, *in: XP 2003*, vol. 2675, Berlin, 2003.

[76] MACIAS F. et al. ”A formal experiment comparing extreme programming with traditional software construction”, *proceedings of the Fourth Mexican International Conference on Computer Science*, 2003.

[77] GEORGE B. and WILLIAMS L. A. “A structured experiment of test-driven development”, *Information and Software Technology*, 2004.

[78] HANKS B. et al. ”Program quality with pair programming in CS1”, *proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education*, 2004.

[79] LAYMAN L. et al. “Exploring extreme programming in context: an industrial case study”, *Agile Development Conference*, 2004.

[80] ILIEVA S. et al. “Analyses of an agile methodology implementation”, *proceedings 30th Euromicro Conference*, IEEE Computer Society Press, 2004.

[81] MELNIK G. and MAURER F. “A cross-program investigation of students' perceptions of agile methods”, *proceedings of the 27th International Conference on Software Engineering*, 2005.

[82] ERDOGMUS H. et al. “On the effectiveness of the test-first approach to programming", *IEEE Transactions on Software Engineering 31*, 2005.

[83] HULKKO H. and ABRAHAMSSON P. “A Multiple Case Study on the Impact of Pair Programming on Product Quality”, *proceedings of the 27th International Conference on Software Engineering*, New York, USA, 2005.

[84] MÜLLER M.M. "Two controlled experiments concerning the comparison of pair programming to peer review." *Journal of Systems and Software*, 2005.

[85] WELLINGTON C. A. et al. “Comparison of student experiences with plan-driven and agile methodologies”, *proceedings of the 35th ASEE/IEEE Frontiers in Education Conference*, 2005.

[86] MELNIK G. and MAURER F. “A cross-program investigation of student’s perceptions of agile methods”, *International Conference on Software Engineering*, USA, 2005.

[87] BHAT T. and NAGAPPAN N. “Evaluating the efficacy of test-driven development: industrial case studies”, *proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*, 2006.

[88] DAMM L.O. and LUNDBERG L. “Results from introducing component level test automation and Test-Driven Development”, *Journal of Systems and Software*, 2006.

[89] MCDOWELL C. et al. “Pair programming improves student retention, confidence, and program quality." *Common. ACM 49*, 2006.

[90] CHAO J. and ATLI G. “Critical Personality Traits in Successful Pair Programming”, *AGILE’06*, IEEE Computer Society, 2006.

[91] MADEYSKI L. “Is External Code Quality Correlated with Programming Experience or Feelgood Factor?”, *XP 2006*, 2006.

[92] LUI K. M. and CHAN K. C. C. “Pair programming productivity: novice–novice vs. expert–expert”. *Human-Computer Studies 64*, 2006.

[93] XU S. and RAJLICH V. “Empirical Validation of Test-Driven Pair Programming in Game Development”, *proceedings of the 5th IEEE/ACIS International Conference on Computer and Information Science*, 2006.

[94] SANCHEZ C. et al. “On the Sustained Use of a Test-Driven Development Practice at IBM”, *proceedings of the AGILE 2007*, 2007.

[95] GUPTA A. and JALOTE P. “An experimental evaluation of the effectiveness and efficiency of the test driven development”, *proceedings of the First International Symposium on Empirical Software Engineering and Measurement*, USA, 2007.

[96] DAMM L.O. and LUNDBERG L. “Quality impact of introducing component level test automation and test-driven development”, *Software Process Improvement*, Vol. 4764, Springer, 2007.

[97] ARISHOLM E. et al. "Evaluating Pair Programming with Respect to System Complexity and Programmer Expertise", *IEEE Transactions in Software Engineering*, 2007.

[98] CANFORA G. et al. ”Evaluating performances of pair designing in industry”, *Journal of Systems and Software*, vol. 80, 2007.

[99] MÜLLER M. M. "Do programmer pairs make different mistakes than solo programmers?" *Journal of Systems and Software*, 2007.

[100] NAGAPPAN N. et al. “Realizing quality improvement through test driven development: results and experiences of four industrial teams”, *Empirical Software Engineering*, 2008.

[101] JANZEN D. and SAIEDIAN H. “Does Test-Driven Development Really Improve Software Design Quality?”, *EEE Software*, 2008.

[102] BEGEL A. and NAGAPPAN N. “Pair programming: what's in it for me?”, *proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, 2008.

[103] BIPP T. et al. "Pair programming in software development teams - An empirical study of its benefits", *Information and Software Technology 50*, 2008.

[104] MOSER R. et al. “A Case Study on the Impact of Refactoring on Quality and Productivity in an Agile Team”, *LNCS*, vol. 5082, 2008.

[105] SINIAALTO, M., and ABRAHAMSSON, P. “Does test-driven development improve the program code? Alarming results from a comparative case study”, *In Balancing Agility and Formalism in Software Engineering*, Springer Berlin, 2008.

[106] HUANG L. and HOLCOMBE M. “Empirical investigation towards the effectiveness of Test First programming”. *Information and Software Technology*, 2009.

[107] DESAI C. and JANZEN D. S. “Implications of integrating test-driven development into CS1/CS2 curricula”, *proceedings of the 40th ACM technical symposium on Computer science education,* USA, 2009.

[108] SFETSOS P. et al. “An Experimental Investigation of Personality Types Impact on Pair Effectiveness in Pair Programming”. *Empirical Software Engineering*, vol. 14, 2009.

[109] USHA, K. et al. "A quantitative approach for evaluating the effectiveness of refactoring in software development process", *proceeding of International Conference on Methods and Models in Computer Science*, IEEE, 2009.

[110] LEE S. and YONG H. S. “Distributed agile: project management in a global environment”, *Empirical Software Engineering*, vol. 15, 2010.

[111] KORHONEN, K. "Evaluating the Effect of Agile Methods on Software Defect Data and Defect Reporting Practices - A Case Study" *Quality of Information and Communications Technology (QUATIC)*, 2010.

[112] MADEYSKI, L. "The impact of test-first programming on branch coverage and mutation score indicator of unit tests: An experiment." *Information and Software Technology 52*, 2010.

[113] CARVALHO, W. C. D. S. et al. "A Comparative Analysis of the Agile and Traditional Software Development Processes Productivity", *proceedings of the 30th International Conference of the Chilean Computer Science Society*, 2011.

[114] PANČUR, M., and CIGLARIČ, M. "Impact of test-driven development on productivity, code and tests: A controlled experiment", *Information and Software Technology*, 2011.

[115] WILLIAMS, L. et al. "Scrum + engineering practices: Experiences of three Microsoft teams". *In Empirical Software Engineering and Measurement (ESEM)*, 2011.

[116] DOGŠA, T., and BATIČ, D. "The effectiveness of test-driven development: an industrial case study", *Software Quality Journal*, 2011.

[117] TUFAIL, R., and MALIK, A. A. "A Case Study Analyzing the Impact of Software Process Adoption on Software Quality", *proceedings of the 10th International Conference on Frontiers of Information Technology*, 2012.

[118] LEMOS, O. A. L. "Development of auxiliary functions: should you be agile? an empirical assessment of pair programming and test-first programming", *proceedings of the 2012 International Conference on Software Engineering*, 2012.

[119] CAUEVIC, A. et al "Quality of testing in test driven development", *In Quality of Information and Communications Technology (QUATIC)*, 2012.

[120] Vonken, F. and Zaidman, A "Refactoring with Unit Testing: A Match Made in Heaven?", *In Reverse Engineering (WCRE)*, 2012.

[121] FUCCI, D. and TURHAN, B. "A Replicated Experiment on the Effectiveness of Test-first Development", *In Empirical Software Engineering and Measurement*, 2013.

[122] MUNIR, H. et al. "An experimental evaluation of test driven development vs. test-last development with industry professionals", *proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, ACM, 2014.

[123] TARHAN, A. and YILMAZ, S. G. "Systematic analyses and comparison of development performance and product quality of Incremental Process and Agile Process", *Information and Software Technology*, 2014.

[124] LINCOLN, Y. S. "Naturalistic Inquiry", Sage, Thousand Oaks, 1985.

[125] QUMER, A., HENDERSON-SELLERS, B. "An Evaluation of the Degree of Agility in Six Agile Methods and its Applicability for Method Engineering", *in Information and Software Technology*, 2007.

[126] DYBÅ, T. and DINGSØYR, T. "Empirical studies of agile software development: A systematic review", *Information and software technology*, 2008.