UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES

MÁSTER UNIVERSITARIO EN DIRECCIÓN Y GESTIÓN DE PROYECTOS

TRABAJO FIN DE MÁSTER

ESTIMACIÓN DE ESFUERZO EN PROYECTOS DE DESARROLLO DE SOFTWARE CON METODOLOGÍAS ÁGILES

AUTOR:
ERWIN R. MÉNDEZ

DIRECTORES:
MARTA FERNÁNDEZ DIEGO
FERNANDO GONZÁLEZ LADRÓN DE GUEVARA

VALENCIA, ESPAÑA
JULIO, 2018

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

Resumen

La estimación de esfuerzo es una actividad fundamental en la industria de desarrollo de software. Ya sea mediante el empleo de modelos simples basados en el juicio u opinión de uno o varios miembros del equipo de desarrollo (expertos en el área) o modelos más complejos que utilizan datos históricos, algoritmos y/o herramientas sistematizadas, la finalidad de este proceso de estimación es poder contar con un valor representativo del esfuerzo que requiere llevar a cabo el proyecto, sirviendo de apoyo a las tareas de planificación y gestión del mismo. A su vez, constituye el principal soporte para la toma de decisiones respecto a factores como coste, recursos, duración y entregables del proyecto.

En el presente trabajo de investigación se estudia el concepto de estimación de esfuerzo dentro del contexto de proyectos de desarrollo de software con metodologías ágiles. Para ello, primero se presentan las características particulares del movimiento ágil desde una perspectiva general, y luego se destacan los aspectos principales de las metodologías ágiles más conocidas en la actualidad. Asimismo, se introduce el concepto de estimación de esfuerzo, abarcando desde los aspectos más generales hasta su aplicación específica en proyectos de desarrollo ágil.

Para analizar el estado del arte del uso de modelos de estimación de esfuerzo en este contexto, se ha desarrollado una revisión sistemática de literatura, en la que se han utilizado cinco motores de búsqueda o librerías digitales (ACM Digital Library, IEEE Xplore, Science Direct, Scopus, Web of Science) para la identificación de artículos de revistas científicas o conferencias publicados entre Diciembre 2013 y Diciembre 2017, respecto a modelos de estimación de esfuerzo utilizados en proyectos de desarrollo ágil. Un total de 46 estudios primarios han sido obtenidos después de la depuración de los resultados de la búsqueda inicial, los cuales han sido sometidos a una serie de preguntas de investigación para la extracción de datos relevantes para este estudio y el posterior análisis comparativo de los mismos. Esta revisión de literatura está basada en un estudio realizado en 2014, por lo que se ha intentado mantener, en la medida de lo posible, una coherencia con el proceso de análisis utilizado en dicho estudio. Esto permite obtener resultados comparables y complementarios a los anteriores, revelando una nueva tendencia en el uso de métodos algorítmicos y técnicas de Machine Learning para la estimación de esfuerzo en proyectos desarrollados con metodologías ágiles.

Además, se ha realizado un análisis exploratorio del repositorio ISBSG en su versión R12-2012. Esta evaluación, que sirve como valor añadido al estudio, se enfoca en aquellos proyectos desarrollados con metodologías ágiles y en la identificación de las

variables relacionadas con el esfuerzo. Ambos resultados (la revisión de literatura y el análisis exploratorio) son comparados y combinados para obtener una visión conjunta del progreso de la estimación de esfuerzo en entornos de desarrollo ágil.

Palabras clave: *Estimación de esfuerzo, metodologías ágiles, proyectos de desarrollo de software, revisión sistemática de literatura, ISBSG*

Abstract

Effort estimation is one of the fundamental practices in the software development industry. Whether it is through simple, expert judgment based models or more complex models that use historical data, algorithms and computational tools, the purpose of this estimation process is to obtain a representative value of the required effort to carry out a software project, supporting the planning and management activities thereof. Also, this value is the main input for decision making regarding cost, resources, duration, and deliverables of the project.

In the present work, the concept of effort estimation is studied in the context of agile software development projects. First, the characteristics of the agile movement are presented from a general perspective and then the main agile methodologies are highlighted. Likewise, effort estimation is first overviewed in a general context, to later develop the specifics of its application in agile software development.

To analyze the state of the art of effort estimation models in this context, a systematic literature review is developed, using five search engines or digital libraries (ACM Digital Library, IEEE Xplore, Science Direct, Scopus, Web of Science) to identify journal articles and conference papers published between December 2013 and December 2017, regarding effort estimation models in agile software development. 46 primary studies resulted after the search results depuration, which have been analyzed through a series of research questions for the extraction of relevant data and the comparative analysis thereof. This literature review is based on a previous study in 2014, so a coherence with the process of analysis use in that study has been kept as far as possible. This allows obtaining comparable and complementary results in relation to the previous study, revealing also a new trend on using algorithm methods and Machine Learning techniques for effort estimation in Agile projects.

Also, an exploratory analysis of ISBSG repository has been carried out for the 2012 release. This evaluation, as added value to the study, is focused on projects developed with agile methodologies and the identification of relevant variables for effort estimation. Both, the SRL and the exploratory analysis, are compared and combined to get a general overview of the progress of effort estimation in Agile.

Keywords: *Effort estimation, agile methodologies, software development projects, systematic literature review, ISBSG*

Índice general

Índice de tablas

Índice de figuras

Capítulo 1

Introducción

Determinar el esfuerzo requerido para llevar a cabo una tarea es parte de todo proceso de desarrollo de software. Ya sea mediante métodos formales o el uso de valoraciones subjetivas basadas en la experiencia de los miembros del equipo del proyecto, la estimación de esfuerzo juega un papel sumamente importante en este proceso. Esto se debe principalmente a que la habilidad de entregar el proyecto de software a tiempo, dentro del presupuesto y con el conjunto de funcionalidades especificadas es uno de los aspectos más críticos de los proyectos de software, y todo ello va estrechamente ligado a estimaciones previstas durante las etapas de planificación del proyecto (Laird y Brennan, 2006).
Dado el creciente uso de las metodologías ágiles para el desarrollo de software (*Agile Software Development* o ASD), se manifiesta un interés cada vez mayor en desarrollar modelos que permitan calcular valores estimados de tamaño, coste y esfuerzo requerido para llevar a cabo proyectos de software dentro de esta categoría. Estas metodologías, basadas en procesos de desarrollo iterativo e incremental, desafían los métodos convencionales de estimación de esfuerzo, principalmente debido al entorno dinámico, flexible y adaptativo que las caracteriza. Por tanto, no resulta práctico intentar aplicar directamente los métodos de estimación utilizados habitualmente en las metodologías tradicionales de desarrollo, sino que se requiere de procesos específicamente ideados para funcionar en el contexto del desarrollo ágil.

Una serie de métodos de estimación de esfuerzo han sido desarrollados para dar solución a esta necesidad. Usman, Mendes, Weidt, y Britto (2014) resumen los resultados de los estudios publicados desde el 2001 hasta el 2013, en relación a modelos de estimación de esfuerzo en ASD. Este estudio, presentado como una revisión de literatura sistemática, permite comprender el estado del arte de estos modelos desde cuatro perspectivas interrelacionadas: la técnica de estimación desarrollada, los predictores de esfuerzo empleados, el contexto de los proyectos analizados para la implementación práctica del modelo y la metodología ágil utilizada. Partiendo de los resultados encontrados en dicho estudio, se ha decidido llevar a cabo una nueva revisión de literatura, considerando únicamente los estudios publicados a partir de Diciembre 2013 hasta finales de 2017.

Este nuevo estudio tiene como objetivo principal dar a conocer el progreso en el desarrollo de los modelos de estimación en los últimos 4 años, así como también realizar un análisis comparativo en relación a los resultados del estudio anterior. Además, se incorpora una revisión de las características de los proyectos de software contenidos en el repositorio repositorio ISBSG, con miras a un posterior trabajo que permita desarrollar un modelo de estimación de esfuerzo basado en las variables disponibles en este repositorio de datos.

Las secciones presentadas a continuación permiten abordar el objeto de investigación descrito anteriormente, yendo desde los aspectos conceptuales hasta los resultados del trabajo empírico con el repositorio. Para ello, primero se presentan los fundamentos y conceptos principales relacionados con las metodologías de desarrollo ágil en el Capítulo 2. En el Capítulo 3 se plantea la estimación de esfuerzo desde un punto de vista general y, además, se describen los métodos de estimación más utilizados en entornos de ASD en la actualidad. Como herramienta de investigación para obtener una noción concreta del estado de los modelos de estimación de esfuerzo en ASD, se ha llevado a cabo una revisión de literatura sistemática, cuya metodología de elaboración y resultados obtenidos se pueden ver en el Capítulo 4. Asimismo, se desarrolla un análisis general de los proyectos en el repositorio ISBSG en el Capítulo 5, en relación a aspectos como la metodología de desarrollo implementada, la calidad de los datos, las métricas de tamaño utilizadas y los valores de esfuerzo presentados. Por último, el Capítulo 6 presenta las conclusiones del estudio, las limitaciones relacionadas al mismo y las futuras líneas de trabajo.

Capítulo 2

Metodologías ágiles para el desarrollo de software

Desarrollo de Software Ágil (ASD, *Agile Software Development*), o simplemente "Agile", es el término por el que se conocen un conjunto de prácticas de desarrollo de software que han logrado un creciente interés entre investigadores y profesionales de la industria de la ingeniería software durante los últimos años. Según Dingsoyr, Dybå, y Moe (2010), ASD es el paradigma más importante que ha sacudido el mundo del desarrollo de software durante la última década, cuyas ideas principales han comenzado a influenciar otras áreas de conocimiento, particularmente la gestión de proyectos. A continuación se describe qué es el desarrollo de software ágil, cuáles son las metodologías ágiles más conocidas y en qué consiste la gestión de proyectos en entornos de ASD.

2.1. Desarrollo de software ágil

2.1.1. Fundamentos y aspectos generales

El Project Management Institute (PMI) y "the Agile Alliance" (2017) consideran *Agile* como un paradigma definido por valores, guiado por principios y manifestado a través de diferentes prácticas, los cuales han de ser seleccionados acorde a las necesidades de los proyectos de desarrollo de software. Dichos valores y principios se presentan en el "Manifiesto ágil"[1]. Los métodos ágiles intentan ofrecer una respuesta a los negocios que ávidamente requieren de procesos de desarrollo menos rígidos y más rápidos (Abrahamsson, Conboy, y Wang, 2009). Highsmith y Cockburn (2001), por otro lado, destacan que lo novedoso de estos métodos no es el uso de un nuevo conjunto de prácticas de desarrollo de sistemas, sino el reconocimiento de las personas como los principales impulsores del éxito de los proyectos, junto con un enfoque en la efectividad y la maniobrabilidad. De hecho, el término mismo fue acuñado mucho

[1] agilemanifesto.org

después del surgimiento, desarrollo e implementación práctica de las que luego pasaron a denominarse metodologías ágiles.

Para Williams y Cockburn (2003), uno de los puntos comunes entre las metodologías ASD es el reconocimiento de que el desarrollo de software es un proceso empírico. Esto es, a diferencia de otras ingenierías, el proceso de desarrollo de software debe procurar ser flexible y no estrictamente predictivo, dado que el mismo estará sujeto a una serie de cambios, en gran parte impredecibles al inicio del desarrollo del producto. Por lo tanto, el desarrollo ágil requiere de pequeños ciclos de inspección-adaptación, con una constante retroalimentación, más que intentar contar con un proceso definido con alto nivel de exactitud y detalle desde las etapas iniciales del proyecto de desarrollo de software.

Erickson, Lyytinen, y Siau (2005), por otro lado, ven el concepto de agilidad como la eliminación de la pesada carga comúnmente asociada con las metodologías de desarrollo de software tradicionales, en la medida de lo posible, para promover la respuesta rápida a los cambios del entorno, cambios en los requerimientos de usuarios, aceleración de las "fechas tope" (*deadlines*), entre otros factores. Estos autores añaden que los métodos tradicionales a menudo resultan rígidos e inertes, por lo que no pueden responder a los cambios lo suficientemente rápido y, por lo tanto, no son viables en muchos de los casos reales en proyectos de desarrollo de software.

En un intento de articular una definición de *agilidad* en desarrollo de sistemas de información, Conboy (2009) asume una taxonomía que identifica un método (o componente del mismo) como ágil si:

- Posee la capacidad de crear cambio, aceptar el cambio, y aprender del cambio;
- Contribuye al valor percibido por parte del cliente (económico, calidad, simplicidad); y
- Está continuamente disponible, requiriendo el mínimo de tiempo y coste para preparar los componentes para su uso.

Como se observa, no existe una definición única y lo suficientemente clara sobre lo que constituye una práctica, método, modelo de desarrollo o metodología ágil (Abrahamsson et al., 2009; Abrahamsson, Salo, Ronkainen, y Warsta, 2017). Es decir, cada investigador adapta la definición de dicho concepto (normalmente asociándolo a prácticas específicas), teniendo en cuenta los valores y principios generales que identifican el movimiento ágil. Gran parte de este desacuerdo se debe a que, como suele suceder en el campo de la ingeniería de software, la práctica ha guiado la investigación con la creación, promoción y difusión de tales métodos gracias fundamentalmente a los esfuerzos de especialistas y consultores en el área (Conboy, 2009).

En general, se podrían identificar como ASD aquellas metodologías que permiten desarrollar software de manera incremental e iterativa, no precisando de documentación formal y extensiva, basadas en la interacción entre las partes interesadas y orientadas a la consecución de resultados efectivos para el cliente en el menor tiempo posible. En contraste, los denominados métodos "tradicionales" son predictivos, esto es, intentan anticipar el conjunto completo de requerimientos y reducir costes eliminando posibles

cambios (Highsmith y Cockburn, 2001). Los partidarios de métodos tradicionales son comúnmente vistos como defensores de la planificación extensiva, los procesos codificados y la reutilización rigurosa (Dybå y Dingsøyr, 2008). El desarrollo de software ágil, por otro lado, no se propone tener una visión holística respecto a los distintos requerimientos del proyecto y las relaciones entre los mismos desde el comienzo, sino que se crean paquetes de funcionalidades que han de ser priorizadas, detalladas y abordadas de acuerdo con cada una de las iteraciones que han de componer el proyecto.

2.1.2. Antecedentes

Como se ha comentado anteriormente, el término "ágil" fue adoptado y popularizado por la comunidad de desarrolladores de software tras la publicación del manifiesto ágil en 2001. Sin embargo, gran parte de los enfoques y técnicas ágiles conocidos y utilizados hasta el día de hoy tienen su origen años antes de la difusión del manifiesto (PMI, 2017). De hecho, es en 1998 que se utiliza por primera vez el término *Agile* para referirse a procesos de desarrollo software (Dybå y Dingsøyr, 2008).

Los enfoques ágiles de desarrollo de software pueden verse como un subconjunto de los modelos de desarrollo **iterativo e incremental**, que se remontan hasta mediados del 1950, o incluso antes (Larman y Basili, 2003). Cockburn (2008) define **desarrollo iterativo** como una estrategia de programación del retrabajo en la que se dedica tiempo para revisar y mejorar las partes del sistema, contrario a la planificación orientada a tenerlo todo correctamente definido desde el principio. Asimismo, presenta el **desarrollo incremental** como la subdivisión del trabajo en pequeñas porciones y la programación de las mismas para ser desarrolladas e integradas a medida que son completadas. Cockburn, además, argumenta que ambos modelos son necesarios para aprovechar las oportunidades de mejorar el proceso de desarrollo y la calidad final del producto desarrollado.

En 1985, Tom Gilb (1985) introduce el **modelo evolutivo** como respuesta ante el modelo en cascada (*Waterfall*) utilizado convencionalmente hasta dicha fecha, y que describe como no realista y peligroso para los objetivos primarios de cualquier proyecto de software. Este modelo evolutivo estaba basado en una serie de principios enfocados principalmente en los siguientes aspectos: entrega de "algo" (funcionalidad) a un usuario final real, medición del valor añadido a los usuarios, y ajuste del diseño y los objetivos basado en las realidades observadas. Para ello, el autor define un conjunto de características críticas, entre las cuales destacan entrega temprana y frecuente, ciclos completos de análisis-diseño-construcción-prueba en cada paso (iteración), orientación al usuario y enfoque a resultados, no a procesos.

En su descripción histórica sobre los modelos iterativo e incremental, Larman y Basili (2003) mencionan la práctica del **prototipado** evolutivo, comúnmente utilizado en la década de 1980 para la creación de sistemas de inteligencia artificial. Más adelante, en 1990, Tripp y Bichelmeyer (1990) presentan el *rapid prototyping* como una alternativa

para resolver los problemas de eficiencia asociados a los modelos tradicionales y mejorar la efectividad de los mismos. Entre las ventajas de este nuevo método se encontraba la posibilidad de que los usuarios podían probar el sistema (antes de tener la versión final del producto de software), identificar problemas y ayudar en la selección de una interfaz apropiada.

En 1986, Barry Boehm presenta el **modelo en espiral** (*spiral model*) como nuevo marco de referencia para guiar el proceso de desarrollo de software. Dicho modelo consistía en la progresión de una misma secuencia de pasos (identificados por cada una de las fases del proceso de desarrollo), para cada una de las porciones del producto final y cada uno de los niveles de elaboración. En particular, el modelo en espiral aporta un mayor enfoque a la gestión del riesgo, en contraste con el modelo en cascada (caracterizados por la documentación extensiva) o el evolutivo (orientado a la codificación) (Boehm, 1986).

Otra de las prácticas importantes en el desarrollo y conceptualización de lo que luego pasaría a conocerse como ASD es la **refactorización**. Este proceso consiste en realizar cambios en el código de forma que no se altere el comportamiento externo del software pero que mejore el funcionamiento interno del mismo (Fowler, 1997). Más adelante, la refactorización aparecería entre las prácticas principales de metodologías como *eXtreme Programmming* (Beck, 1999).

La Figura 2.1 resume los principales acontecimientos previo a la declaración del manifiesto ágil.

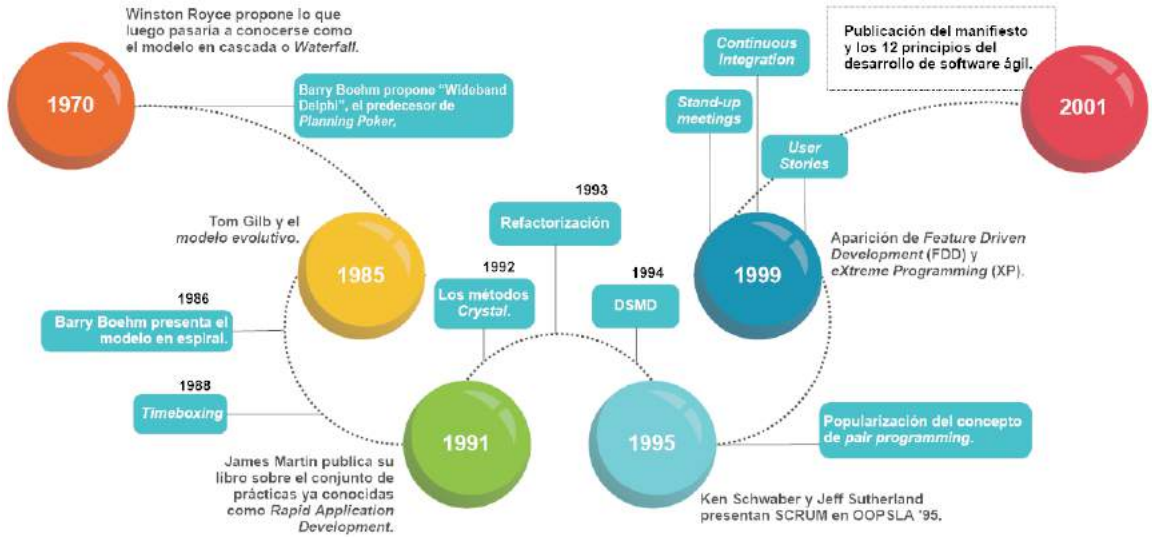

Figura 2.1: Antecedentes del manifiesto ágil. Fuente: Elaboración propia.

2.1.3. El manifiesto ágil

El manifiesto ágil y los 12 principios del software ágil[2] fueron publicados en 2001, cuando un grupo de 17 profesionales de la industria de desarrollo de software decidió reunirse y formular una alternativa a los procesos de desarrollo tradicionales aplicados durante los años anteriores, principalmente orientados a la documentación y dirigidos por la planificación. El objetivo principal era encontrar una base común que integrara lo que ya se encontraban implementando en sus respectivas organizaciones (Hazzan y Dubinsky, 2014). Tal y como resaltan estos autores, la mera formulación de un manifiesto implica que, aunque haya principios e ideas comunes, los mismos pueden ser aplicados de forma diferente por medio de métodos específicos, tales como *Scrum, Extreme Programming, Lean, DSDM (Dynamic Systems Development Method), Adaptive Software Development, Cristal, FDD (Feature Driven Development)*, entre otros.

El manifiesto ágil consta de cuatro puntos claves, los cuales se resumen en la Figura 2.2.

Figura 2.2: Manifiesto ágil

- **Individuos e interacciones sobre procesos y herramientas**: da mayor énfasis a las relaciones entre los individuos involucrados en los procesos de desarrollo de software, más que a los procesos mismos o las herramientas utilizadas. Este principio permite dar prioridad a las personas, la comunicación y la interacción, teniendo especialmente en cuenta la influencia que ejerce la toma de decisiones sobre cada uno de estos factores.
- **Software funcionando sobre documentación extensiva**: procura como objetivo fundamental del equipo de desarrollo la entrega de soluciones de software, de forma constante y en el menor intervalo de tiempo posible. La documentación

[2] agilemanifesto.org/iso/es/principles.html

se reduce a un nivel apropiado, evitando consumir tiempo que puede dedicarse al desarrollo, prueba o puesta en marcha del software.
- **Colaboración con el cliente sobre la negociación contractual**: da preferencia a la cooperación y el trabajo en equipo sobre la estipulación de contratos estrictos y extensivos.
- **Respuesta ante el cambio sobre seguir un plan**: busca la adaptación a los cambios introducidos durante las fases del ciclo de desarrollo de software, de forma flexible y sin comprometer la calidad del producto final; para ello, el equipo de proyecto debe procurar contar con las herramientas necesarias para realizar ajustes en el momento en que sean requeridos.

2.2. Metodologías ágiles

Dentro del marco conceptual de ASD se pueden agrupar una gran variedad de metodologías y prácticas que comparten los principios comunes del desarrollo de software ágil. Sin embargo, es evidente que el uso del término "metodología" puede resultar controvertido en este contexto. Por ejemplo, Highsmith (2002) prefiere utilizar el concepto "ecosistemas" para capturar mejor la orientación hacia personas y sus interacciones dentro de un contexto organizacional dinámico y en constante cambio.

La Figura 2.3 muestra un esquema con las principales prácticas relacionadas con ASD, subdivididas en categorías o "tribus" (*tribes*) que representan cada una de las áreas de interés dentro del movimiento ágil. Estas tribus hacen referencia al conjunto de disciplinas principales de la comunidad ágil que participan en las conferencias anuales organizadas por *Agile Alliance* (Bossavit, 2012).

Figura 2.3: Mapa de metodologías ágiles. Fuente: agilealliance.org.

Por otro lado, Webb (2016) propone una perspectiva ágil aún más completa y compleja, que contiene un número mayor de marcos de trabajo dentro de ASD. El objetivo de esta propuesta[3] es presentar ASD como un conjunto de prácticas altamente interconectadas que transforman ideas en valor a través de una serie de zonas: *Initiate, Discover, Deliver, Scaling, Leadership, Release*. Regularmente se inicia por un enfoque en la entrega de soluciones mediante el uso de métodos ágiles (*Deliver, Release*) y luego, a medida que la organización va alcanzando cierto nivel de madurez en la implementación de estas metodologías, surge la necesidad de definir la forma en que se realiza el trabajo y se diseñan las soluciones (*Initiate, Discover*), y escalar tales principios a lo largo de toda la organización y el equipo de gestión (*Scaling, Leadership*). Sin embargo, esta vista panorámica no pretende ser exhaustiva, sino dar a entender que *ser ágil no es tan simple como seguir una única metodología*.

En el presente trabajo de investigación se han seleccionado las principales metodologías ágiles las que se muestran en la Tabla 2.1. Para la descripción se sigue la estructura planteada por Qumer y Henderson-Sellers (2008).

[3] blog.deloitte.com.au/navigating-the-agile-landscape/

Tabla 2.1: Metodologías de desarrollo ágil. Adaptado de (Qumer y Henderson-Sellers, 2008).

Metodologías	Referencias principales	Tamaño del proyecto	Tamaño del equipo	Modelo de desarrollo	Distribución física	Cultura de negocio
Adaptive Software Development	(Highsmith, 2000)	Grandes y complejos	No especificado	Desarrollo iterativo, rápido y distribuido	Equipos coubicados y distribuidos	No especificada
Crystal	(Cockburn, 2002)	Pequeños y medianos	Un equipo, máximo de 6 personas (Crystal Clear); múltiples equipos: máximo de 40 personas (Crystal Orange), entre 40 y 80 personas (Crystal Red)	Desarrollo iterativo y rápido	Equipos coubicados (sin soporte para desarrollo distribuido)	No especificada
Dynamic Systems Development Method (DSDM)	(Stapleton, 1997)	Pequeños y grandes (aplicaciones de negocio)	Mínimo 2 y máximo 6 personas (múltiples equipos)	Desarrollo iterativo, rápido y cooperativo	No especificada	Colaborativa y cooperativa

Tabla 2.1: Metodologías de desarrollo ágil. Adaptado de (Qumer y Henderson-Sellers, 2008). [cont.]

Metodologías	Referencias principales	Tamaño del proyecto	Tamaño del equipo	Modelo de desarrollo	Distribución física	Cultura de negocio
Extreme Programming (XP)	(Beck, 2000)	Pequeños y medianos	Máximo 10 personas	Desarrollo iterativo y rápido	Equipos coubicados y distribuidos (iteraciones limitadas)	Colaborativa y cooperativa
Feature Driven Development (FDD)	(Palmer & Felsing, 2002)	Pequeños, medianos y grandes (proyectos y aplicaciones de negocio)	Sin límites (escalable desde pequeños hasta grandes equipos)	Diseño y construcción iterativos	No especificada	No especificada
Lean Software Development	(M. B. Poppendieck & Poppendieck, 2003)	Grandes y complejos	Hasta 50 personas	Desarrollo iterativo y rápido	Equipos coubicados y distribuidos	Basada en los principios de "Lean Thinking"
Scrum	(Schwaber, 1995; Schwaber & Beedle, 2002)	Pequeños y medianos (escalable para grandes proyectos)	Máximo 10 personas (múltiples equipos)	Desarrollo iterativo y rápido	No especificada	No especificada

Teniendo en cuenta las representaciones anteriores y otras clasificaciones presentadas en (Abrahamsson et al., 2017; Dybå y Dingsøyr, 2008; Highsmith, 2002; PMI, 2017), a continuación se describen cada una de estas metodologías.

2.2.1. Adaptive Software Development

Se enfoca principalmente en el desarrollo de sistemas grandes y complejos. Este método enfatiza el desarrollo iterativo e incremental, con prototipado constante. Según Highsmith (2002), esta metodología provee un trasfondo filosófico a los métodos ágiles, mostrando como las organizaciones de desarrollo de software pueden responder ante los cambios mediante la gestión de los mismos (adaptación continua), en lugar de tratar de evitarlos.

Un ciclo de vida de un proyecto bajo esta metodología suele contener las siguientes características básicas:

a) *Centrado en la misión*. Las actividades en cada ciclo de desarrollo deben estar justificadas por los objetivos generales del proyecto.
b) *Basado en componentes*. El desarrollo no debería estar orientado a la mera ejecución de tareas, sino enfocado a la construcción de software que funcione.
c) *Iterativo*. Teniendo en cuenta la inestabilidad e incertidumbre de los procesos de desarrollo de software, el esfuerzo de desarrollo debería estar enfocado en *rehacer*, en lugar de intentar *hacerlo bien desde el principio*.
d) *Time-boxed* o con restricción de tiempo. La ambigüedad en proyectos complejos puede ser resuelta mediante el establecimiento de límites de tiempo de forma regular.
e) *Impulsado por la gestión del riesgo*. El desarrollo de componentes de alto riesgo debería iniciarse tan pronto como sea posible.
f) *Tolerante al cambio*. Los cambios son frecuentes en el desarrollo de software; por tanto, es más importante aprender a adaptarse a estos que tratar de controlarlos.

Tabla 2.2: Aspectos clave y deficiencias de *Adaptive Software Development*.

Aspectos clave (+)	Principales deficiencias (-)
Colaboración; Desarrollo iterativo; Organizaciones vistas como sistemas adaptativos.	Comúnmente considerado como modelo conceptual de cultura organizacional más que como una práctica de desarrollo de software.

2.2.2. Crystal

Crystal incluye un número de metodologías que pueden ser seleccionadas de acuerdo a las características individuales de cada proyecto. Estas se dividen por color, indicando la

robustez de la metodología (mientras más oscuro el color, más robusta la metodología). Las diferentes metodologías Crystal son *Clear, Yellow, Orange, Red, Blue*, cada uno de los cuales corresponde al tamaño y nivel de criticidad del proyecto diferentes. Alistair Cockburn[4] es el autor de estos métodos.

Las únicas metodologías de esta familia que han sido desarrolladas y utilizadas hasta el momento son *Crystal Clear* y *Crystal Orange* (Abrahamsson et al., 2017). La primera se centra en la comunicación de equipos pequeños que desarrollan software de bajo nivel de criticidad, mientras que la segunda esta diseñada para proyectos de tamaño mediano, con un total de hasta 40 miembros y duración de hasta dos años.

Tabla 2.3: Aspectos clave y deficiencias de las metodologías Crystal.

Aspectos clave (+)	Principales deficiencias (-)
Habilidad de escoger la metodología más adecuada (cada una con la misma base de valores y principios fundamentales); Variedad de técnicas, roles, herramientas y estándares.	Gran parte de la documentación existente se enfoca únicamente en el uso e implementación de Crystal Clear, por lo que se carece de información respecto al resto de metodologías. Solo cubren proyectos de bajo nivel de criticidad e implementados por equipos localizados en el mismo lugar (no distribuidos).

2.2.3. Dynamic Systems Development Method (DSDM)

Dynamic Systems Development Method (DSDM) es una extensión de las prácticas conocidas como *Rapid Application Development* (RAD). Es considerado el primer método de desarrollo de software ágil (Abrahamsson et al., 2017). La idea fundamental de DSDM consiste en la preferencia a fijar los recursos y el tiempo requerido para el desarrollo de una funcionalidad en primera instancia, y luego ajustar el porcentaje que ha de ser desarrollado de dicha funcionalidad. DSDM divide los proyectos en tres fases (pre-proyecto, ciclo de vida del proyecto y pos-proyecto) y está basado en nueve principios fundamentales: participación del usuario, empoderamiento del equipo de proyecto, entrega frecuente, enfoque en las necesidades actuales del negocio, desarrollo iterativo e incremental, reversión de cambios, alcance fijado a alto nivel antes del inicio del proyecto, prueba de desarrollo a lo largo del ciclo de vida del proyecto, y comunicación efectiva y eficiente.

Este método ha sido ampliamente implementado en el Reino Unido desde mediados de 1990, convirtiéndose en una alternativa viable frente a RAD. Asimismo, ha sido aplicada en proyectos pequeños y grandes, con la precondición de que requiere subdividir el

[4]alistair.cockburn.us

trabajo en componentes que puedan ser desarrollados por equipos pequeños en el caso de proyectos de gran tamaño.

Tabla 2.4: Aspectos clave y deficiencias de DSDM.

Aspectos clave (+)	Principales deficiencias (-)
Aplicación de controles a RAD; Utilización de *timeboxing*, prototipado y diferentes roles de usuario.	Solo lo miembros del consorcio tienen acceso a recursos sobre la aplicación real del método.

2.2.4. Extreme Programming (XP)

Extreme Programming (XP)[5] fue desarrollado por Kent Beck, Ward Cunningham y Ron Jeffries, a finales de la década de 1990. Tal y como se denota de su nombre, se centra principalmente en las prácticas de desarrollo de software. Además, XP hace énfasis en los valores de comunidad, simplicidad, retroalimentación y coraje como parte de sus características principales.

XP considera 12 prácticas como principales, las cuales son:

- *Planning game* (priorización de las historias de usuario y acuerdos sobre el contenido de la primera iteración),
- ciclos cortos,
- metáforas (definiciones de las funcionalidades del sistema),
- diseño simple,
- pruebas (*Test Driven Development*),
- refactorización,
- *Pair programming* (dos personas trabajando sobre el mismo código),
- propiedad colectiva o *collective ownership* (cualquier miembro del equipo puede trabajar en cualquier parte del código en cualquier momento),
- integración continua o *continuous integration*,
- 40 horas a la semana (sin semanas que incluyan horas extras de forma consecutiva),
- clientes "on-site" (presentes y disponibles a tiempo completo para el equipo), y
- aplicación de estándares de desarrollo.

La revisión XP2 ha añadido nuevas prácticas como primarias, entre las cuales destacan: espacio de trabajo informativo, trabajo energizado, historias de usuario y diseño incremental (Dybå y Dingsøyr, 2008). Esta nueva versión también incluye un conjunto de 11 prácticas corolarias, que requieren del dominio y conocimiento de las primarias.

XP, además, ha contribuido en gran manera a la redefinición del "coste del cambio" y al esfuerzo por la excelencia técnica mediante la refactorización y el enfoque de

[5] www.extremeprogramming.org

desarrollo conocido como *Test Driven Development*.

Tabla 2.5: Aspectos clave y deficiencias de XP.

Aspectos clave (+)	Principales deficiencias (-)
Desarrollo orientado al cliente; Implementación de equipos pequeños; Búsqueda de excelencia técnica; Promoción del uso de estándares de codificación.	Se ofrece menos atención a la supervisión y gestión general, y más al desempeño individual. Cualquier resistencia frente a los principios y prácticas generales de XP, tanto por parte de los miembros del equipo como de los clientes, puede provocar que falle todo el proceso.

2.2.5. Feature Driven Development (FDD)

Feature Driven Development (FDD) es un enfoque ágil y adaptativo, con un énfasis en el modelo de objeto inicial, la división del trabajo en funcionalidades o *features* y el diseño iterativo de las mismas. Fue desarrollado por Jeff De Luca y Peter Coad, siendo primero reportado en (Coad, Luca, y Lefebvre, 1999) y publicándose posteriormente la guía práctica en (Palmer y Felsing, 2002). FDD consiste de cinco procesos secuenciales (desarrollo del modelo general, construcción de la lista de funcionalidades, planificación por funcionalidad, diseño por funcionalidad y construcción por funcionalidad) y tres categorías de roles (roles clave, roles de apoyo y roles adicionales).

En la práctica, los procesos de FDD son similares a los de XP y otras metodologías ágiles. Sin embargo, algunos aportes distintivos son las soluciones que ofrece este modelo para respuesta al cambio y la orientación hacia el modelado inicial robusto, no tan enfocado en la refactorización. Asimismo, el valor global de una funcionalidad y su programación en el tiempo es determinado en las fases iniciales del proyecto, considerándolo como un aspecto primariamente técnico (a diferencia de Scrum, en el que el cliente juega un papel sumamente importante en la priorización de las funcionalidades del sistema).

Tabla 2.6: Aspectos clave y deficiencias de FDD.

Aspectos clave (+)	Principales deficiencias (-)
Proceso de desarrollo simple (cinco pasos); Desarrollo orientado a objetos; Iteraciones muy cortas (desde horas hasta dos semanas); Diseño e implementación por funcionalidad.	Se enfoca únicamente en el diseño e implementación, por lo que necesita de otros enfoques complementarios.

2.2.6. Lean Software Development

Lean Software Development es una adaptación de los principios aplicados en la industria de manufactura y producción, específicamente el sistema de producción de la compañía Toyota desde 1950. En (PMI, 2017) se considera que las metodologías ágiles y el método Kanban[6] son descendientes de *Lean Thinking*, compartiendo características como entrega de valor, respeto a las personas, disminución del desperdicio, transparencia, adaptación al cambio y mejora continua. Es por ello que algunos consideran Lean Software Development como una disciplina relacionada a ASD, no un subconjunto de la misma (Anderson, 2012).

Entre las contribuciones de *Lean Software* al movimiento ASD se encuentran el enfoque estratégico, el vínculo con los sistemas de producción, el concepto de riesgo empresarial y la extensión de los objetivos. Por otro lado, en (Razzak, 2016, pp. 61–62) se destaca la existencia de ciertos retos a la hora de intentar implementar *Lean* en la industria del desarrollo de sistemas. Por ejemplo, el concepto de valor no es sencillo de definir en la ingeniería de software porque no está limitado a un único esfuerzo acotado en el tiempo. Además, el factor humano es dominante en el desarrollo de sistemas (un proceso creativo, basado en el conocimiento y la experiencia), mientras que en manufactura y producción, las personas son mayormente requeridas para operar maquinarias automatizadas.

Tabla 2.7: Aspectos clave y deficiencias de *Lean Software Development*.

Aspectos clave (+)	Principales deficiencias (-)
Enfoque de siete disciplinas principales: eliminación del desperdicio, amplificación del aprendizaje, toma de decisión lo más tarde posible, entrega lo más rápido posible, empoderamiento del equipo de trabajo, integridad y vista del panorama completo.	El éxito del proyecto se vuelve altamente dependiente de la cohesión y disciplina de los miembros del equipo y sus compromisos individuales.

2.2.7. Scrum

Según *Scrum Alliance*[7], Scrum es un conjunto de principios y prácticas enfocados en la entrega de productos de software en ciclos cortos, permitiendo rápida retroalimentación, mejora continua y adaptación rápida al cambio. El término se deriva de la estrategia en el juego de "rugby" donde se intenta colocar el balón devuelta al campo de juego mediante trabajo en equipo (Schwaber y Beedle, 2002).

[6]Este método, inspirado en *Lean Manufacturing*, es menos prescriptivo y menos disruptivo que algunos de los métodos ágiles (PMI, 2017).

[7]www.scrumalliance.org

Este enfoque fue desarrollado por Ken Schwaber y Jeff Sutherland, con colaboraciones posteriores de Mike Beedle. La idea principal es que el desarrollo de sistemas envuelve una serie de variables técnicas y ambientales (requerimientos, tiempo, recursos y tecnología) que muy probablemente han de cambiar a lo largo del proceso, lo cual lo hace impredecible y complejo, requiriendo gran flexibilidad para ser capaz de responder a los cambios. Es por ello que esta metodología esta más orientada a la gestión del proceso de desarrollo de sistemas.

En Scrum, el software es desarrollado por equipos auto-organizables en iteraciones (comúnmente conocidas como *sprints*), iniciando con la planificación y terminando con una revisión retrospectiva. Cada *sprint* cuenta con un conjunto de funcionalidades que han de ser implementadas. Estas, a su vez, se encuentran registradas en el *Product Backlog*, que define todo el trabajo requerido para completar el proyecto (por ejemplo, funcionalidades, corrección de errores y defectos del sistema, peticiones de cambio y actualizaciones tecnológicas).

Entre los principales roles utilizados en Scrum se encuentran el *Product Owner*, el *Scrum Master* y el *Scrum Team*. El *Product Owner* decide qué elementos del *backlog* han de ser desarrollados en el próximo *sprint* por parte de los miembros del *Scrum Team*, quienes participan en la estimación, coordinación, ejecución y seguimiento del trabajo diario durante el proceso de desarrollo. Uno de los miembros del equipo, el *Scrum Master*, se encarga de garantizar que el proyecto es llevado a cabo acorde a los valores, prácticas y reglas definidos por Scrum, y que el mismo progresa según lo planificado. Este último rol, además, interactúa tanto con el equipo de desarrollo como con el cliente, procurando resolver cualquier impedimento que obstaculice el avance del trabajo de forma efectiva.

Tabla 2.8: Aspectos clave y deficiencias de Scrum.

Aspectos clave (+)	Principales deficiencias (-)
Equipos independientes, pequeños, multifuncionales y auto-organizables; Seguimiento diario (*stand-up meetings*); Enfoque a gestión del proyecto.	Las fases de integración y prueba no aparecen detalladas; Se recomienda su implementación con equipos de más de 10 personas.

2.3. Gestión de proyectos en ASD

Las metodologías ágiles de desarrollo de software recomiendan el uso de equipos auto-organizables y multifuncionales, gestión menos restrictiva y participación extensiva de los usuarios. En algunos casos, el rol de *Project Manager* (PM) no es reconocible o el equipo de desarrollo es el que asume muchas de las responsabilidades tradicionales del PM (Taylor, 2016). De manera que estas metodologías difieren considerablemente en relación a la forma en que cada una realiza la gestión de

proyectos. Por ejemplo, Abrahamsson, Warsta, Siponen, y Ronkainen (2003) destacan los siguientes paralelismos al respecto:

- Scrum está explícitamente ideado para la gestión de proyectos de desarrollo de software ágil, mientras que XP requiere de guías suplementarias para realizar las tareas de gestión.
- En *Adaptive Software Development* se promueve una cultura de desarrollo en la que la gestión tiene que ceder para dar respuesta a los cambios en el proyecto; en contraste, FDD asume que el PM tiene la decisión final sobre el alcance, duración y recursos del proyecto.
- DSDM se enfoca en facilitar el trabajo del equipo de desarrollo mediante el seguimiento diario del progreso del proyecto, mientras que Crystal intenta abordar el tema de la gestión por mediante la propuesta de una serie de metodologías que han de adecuarse al propósito y cualidades particulares del proyecto.

VersionOne, una empresa proveedora de soluciones "ágiles" para organizaciones interesadas en implementar este tipo de metodologías, publica anualmente un informe sobre el estado del uso de metodologías ágiles en la industria[8], incluyendo organizaciones de servicios tecnológicos, financieros, gubernamentales, farmacéuticos y de salud, manufactura, entre otras. En 2018, este informe presentó entre los principales beneficios de la adopción de *Agile*: a) habilidad de gestionar el cambio de prioridades, b) visibilidad del proyecto, c) alineamiento del negocio con las Tecnologías de la Información, d) entrega más rápida de soluciones de software, y e) mayor productividad del equipo del proyecto.

De un total de 1,492 encuestados, el 70 % indicó utilizar Scrum como metodología de desarrollo ágil, ya sea individualmente (56 %) o en combinación con otras metodologías como XP (6 %) o Kanban (8 %). Por otro lado, un 14 % señaló utilizar una combinación de múltiples metodologías. En relación a las técnicas ágiles, las más destacadas son: *stand-up meetings*, planificación por iteración/*sprint*, reuniones retrospectivas, reuniones de revisión de la iteración o *sprint* y uso de iteraciones cortas (VersionOne, 2018). Otro aspecto que resalta este reporte es que la satisfacción del cliente ocupa la primera posición con un 57 %, entre las métricas para medir el éxito de los proyectos ágiles, seguida de la entrega a tiempo (55 %) y el valor del negocio (53 %).

En cuanto a la relación entre ASD y las áreas de conocimiento del *Project Management Body of Knowledge* (PMBoK), Dolan (2007) ha identificado como cada uno de los principios y mejores prácticas del desarrollo ágil que se vinculan directamente con dichas áreas del PMBoK. Esto se puede observar en la Tabla 2.9, en la que se ha añadido la gestión de las partes interesadas (*stakeholders*), que se incluye como nueva área de conocimiento a partir de la quinta edición del PMBoK.

[8] stateofagile.versionone.com

Tabla 2.9: Mapeo entre ASD y las áreas de conocimiento del PMBoK.

Áreas de conocimiento del PMBoK	**Principios de desarrollo de software ágil**	**Mejores prácticas en ASD**
Gestión de la integración	Integración continua	• Tratar una interfaz de usuario como un contrato entre el equipo de desarrollo y el sistema. • Construir y probar las interfaces de usuario en las primeras etapas del proyecto, gestionando a tiempo los puntos de riesgo del proyecto. • Construir emuladores para probar el funcionamiento del sistema como está destinado a ser utilizado.
Gestión del alcance	Aceptación del cambio	• Utilizar iteraciones rápidas para desarrollar software que funcione y del que se pueda recibir retroalimentación. • Planificar iteraciones cortas, reduciendo la incertidumbre y descomponiendo el trabajo en pequeños entregables. • Programar demostraciones del funcionamiento del software a los clientes al final de cada iteración.
Gestión del tiempo	Estimación y gestión del tiempo	• Utilizar el *timeboxing* en la programación de la duración de las actividades del proyecto. • Estimar el trabajo a realizar utilizando puntos de historia (*story points*), así como también evaluar y mejorar los resultados de las estimaciones realizadas en cada iteración. • Considerar el trabajo completado en iteraciones pasadas como medida máxima de trabajo estimado para nuevas iteraciones.
Gestión de los costos	Entrega continua	• Definir una lista de criterios y condiciones que el software ha de cumplir para ser considerado como *Done* (o Listo). • Elaborar escenarios de prueba *end-to-end*, en los que las partes interesadas puedan determinar en cualquier momento cuando una funcionalidad ha sido implementada correctamente o dentro de los criterios de aceptación.

Tabla 2.9: Mapeo entre ASD y las áreas de conocimiento del PMBoK. [cont.]

Áreas de conocimiento del PMBoK	**Principios de desarrollo de software ágil**	**Mejores prácticas en ASD**
Gestión de la calidad	TDD y automatización de las pruebas	• Utilizar TDD como enfoque de pruebas de desarrollo. • Automatizar los casos de prueba, en especial los de más alta prioridad. • Implementar la práctica de integración continua (*continuous integration*), garantizando que los cambios más recientes no hayan desestabilizado el funcionamiento del proyecto desarrollado y que los mismos cumplen con los criterios de calidad requeridos.
Gestión de los recursos humanos	Liderazgo y equipos auto-gestionados	• Contar con equipos que sean responsable de sí mismos. • Asegurar que cada iteración cuenta con reuniones retrospectivas de las que se pueda obtener resultados significativos y accionables. • Reconocer las fortalezas del equipo y desarrollar un ambiente donde la retroalimentación crítica y honesta es valorada.
Gestión de las comunicaciones	Comunicación efectiva y equipos motivados	• Hacer uso de reuniones cortas para identificar el progreso del trabajo y los puntos de bloqueo que obstaculicen el avance del proyecto. • Demostrar el progreso del proyecto constantemente a las partes interesadas. • Contar con sesiones de planificación que permitan realizar la priorización de las funcionalidades próximas a ser abordadas, en las que participen cada una de los roles involucrados en el proyecto.
Gestión de los riesgos	Mitigación de los riesgos	• Trabajar sobre los elementos de mayor riesgo en las etapas tempranas del proyecto. • Identificar y mitigar los riesgos antes de que se conviertan en problemas reales para el proyecto, mediante prácticas como *continuous integration* y las reuniones retrospectivas.

Tabla 2.9: Mapeo entre ASD y las áreas de conocimiento del PMBoK. [cont.]

Áreas de conocimiento del PMBoK	**Principios de desarrollo de software ágil**	**Mejores prácticas en ASD**
Gestión de las adquisiciones	Gestión ágil de los contratos	• Garantizar que el uso de las buenas prácticas de desarrollo ágil se encuentre estipulado en el contrato y en la definición de los entregables. • Enfocarse en la buena comunicación, la integración y los modelos de prueba para mantener al equipo alineado con los entregables establecidos.
Gestión de las partes interesadas	Trabajo colaborativo	• Considerar al usuario final como el principal *stakeholder* del proyecto. • Garantizar la participación activa de cada uno de los interesados del proyecto. • Promover una cultura de honestidad y confianza.

A primera vista, tal y como destaca Dolan (2007), podría parecer que las prácticas de gestión de proyectos definidas en el PMBoK se ajustan únicamente a modelos secuenciales como *Waterfall*. No obstante, en el mapeo anterior se puede observar como cada una de las áreas de conocimiento del PMBoK pueden ser identificadas y abordadas por medio de una serie de mejores prácticas de desarrollo ágil. Algunas de estas áreas han de ser percibidas como primarias y recibirán mayor énfasis en el movimiento ágil (integración, recursos humanos y comunicaciones), mientras que otras han de requerir mayor flexibilidad para su gestión (alcance, riesgos y contrataciones). Con la nueva guía del PMBOK 6ª edición han aparecido cambios importantes para incorporar las metodologías ágiles en los fundamentos para la dirección de proyectos establecidos por el PMI. Esta edición contiene numerosas referencias a prácticas adaptativas e iterativas, incluyendo las metodologías ágiles. De hecho, cada área de conocimiento posee una sección dedicada a la aplicación de las metodologías ágiles en su contexto. Además existe un nuevo apéndice dedicado a la gestión de proyectos ágiles y otras prácticas iterativas[9].

Por otro lado, IPMA ha incluido una certificación orientada al liderazgo de proyectos en entornos ágiles (*IPMA Agile Leadership*[10]). Esta certificación se enfoca en el establecimiento y reconocimiento de las competencias individuales del profesional, sirviendo como adicional a otras certificaciones como Scrum y SAFe[11].

[9] www.pmi.org/pmbok-guide-standards/foundational/pmbok
[10] blog.ipma.world/ipma-agile-leadership
[11] www.scaledagile.com/certification/about-safe-certification

PRINCE2, por su parte, cuenta con una guía práctica que permite combinar la flexibilidad de las metodologías ágiles con la gobernabilidad de PRINCE2 (AXELOS, 2015). Esta guía intenta capacitar a los profesionales en entornos de ASD para completar el proyecto a tiempo, mantener la calidad, adaptarse al cambio, mantener la estabilidad del equipo del proyecto y gestionar adecuadamente las expectativas de las partes interesadas.

Capítulo 3

Estimación de esfuerzo en proyectos de desarrollo de software

El desarrollo de un sistema software es una tarea compleja y conlleva una serie de procesos destinados al cumplimiento de los objetivos del proyecto de software y las expectativas de los usuarios finales. Estos procesos suelen cambiar dependiendo del nivel organizativo de la empresa, la complejidad del producto, las plataformas de desarrollo, los factores humanos, etc. Además, con vistas a ser competitivas, las organizaciones requieren que el departamento de sistemas de información tenga un desempeño flexible y de calidad, en un entorno de recursos limitados.

Por tanto, la planificación se convierte en el elemento esencial para procurar la monitorización y control de tales procesos (Sommerville, 2011). Poder disponer de estimaciones precisas que permitan determinar el coste del producto, el tamaño relativo del proyecto de desarrollo y el esfuerzo necesario para desarrollarlo es de suma importancia para la realización de licitaciones, negociaciones contractuales, planificación, monitorización y control (S. K. Sehra, Brar, Kaur, y Sehra, 2017), así como también para contribuir el éxito de estos proyectos de desarrollo (Trendowicz y Jeffery, 2014). A continuación se presentan los conceptos relacionados a la estimación de esfuerzo, tanto en un contexto general como su aplicación detro de las metodologías de desarrollo ágil.

3.1. Estimación de esfuerzo en un contexto general

El objetivo principal de la estimación de los costes y esfuerzo de los proyectos software es determinar la carga de trabajo y sus costes de acuerdo con el ciclo de vida del sistema. Tales estimaciones deberían ser lo suficientemente fiables como para permitir

el establecimiento de objetivos claros y alcanzables (Top, Ozkan, Nabi, y Demirors, 2011). El componente fundamental del coste de desarrollo de software es atribuible al esfuerzo humano y la mayoría de los métodos de estimación de coste se enfocan en este aspecto y proporcionan estimaciones en términos de personas/mes (Borade y Khalkar, 2013).

Los gestores de proyecto requieren modelos predictivos formales que faciliten estimaciones, no solo en las fases de inicio, sino también mientras se lleva a cabo el proyecto; errar en la estimación del coste puede ocasionar que los sistemas sean más caros, excediendo el presupuesto al haber dedicado más recursos, con funciones no suficientemente desarrolladas, de menor calidad y no entregados en tiempo y plazo. Asimismo, puede dar lugar a una pérdida de contratos o incurrir en penalizaciones por parte de los clientes. En fin, la estimación de esfuerzo es una tarea compleja que precisa del conocimiento de una serie de factores que afectan a las actividades, individuales y colectivas, que conforman dichos proyectos (Trendowicz, Münch, y Jeffery, 2011).

Existen varios elementos que influyen de forma negativa en el proceso de estimación de software. Uno de ellos es la ausencia de información histórica que incluya tanto el conocimiento de los expertos del área en cuestión como su experiencia en proyectos anteriores (Borade y Khalkar, 2013). Estos datos históricos, aunque generalmente incompletos, inciertos y con ruido (datos no estructurados o difíciles de interpretar) (Reddy, Sudha, Sree, y Ramesh, 2010), ayudan a la toma de decisiones sobre los requerimientos necesarios para la construcción de modelos de estimación fiables (S. K. Sehra et al., 2017). Otro problema importante, que probablemente es la causa de todos los demás problemas, reside en la naturaleza misma de estos proyectos, donde: (a) el talento humano ejerce un papel fundamental, (b) las tareas realizadas son esencialmente de carácter no repetitivo, y (c) los elementos relacionados al sistema, la organización y el entorno del proyecto son altamente dinámicos.

En la actualidad, el software puede ser desarrollado siguiendo diversos enfoques, tales como los presentados en la Figura 3.1. Factores como el tamaño del equipo, el nivel organizativo, la gestión de las comunicaciones, la experiencia de los miembros del equipo, el tipo de producto de software a desarrollar, entre otros, estarán directamente relacionados al contexto de desarrollo del software. Por ello, la forma como el software es estimado, dentro del contexto de un proyecto, puede variar dependiendo del enfoque de desarrollo adoptado (Britto, Mendes, y Börstler, 2015).

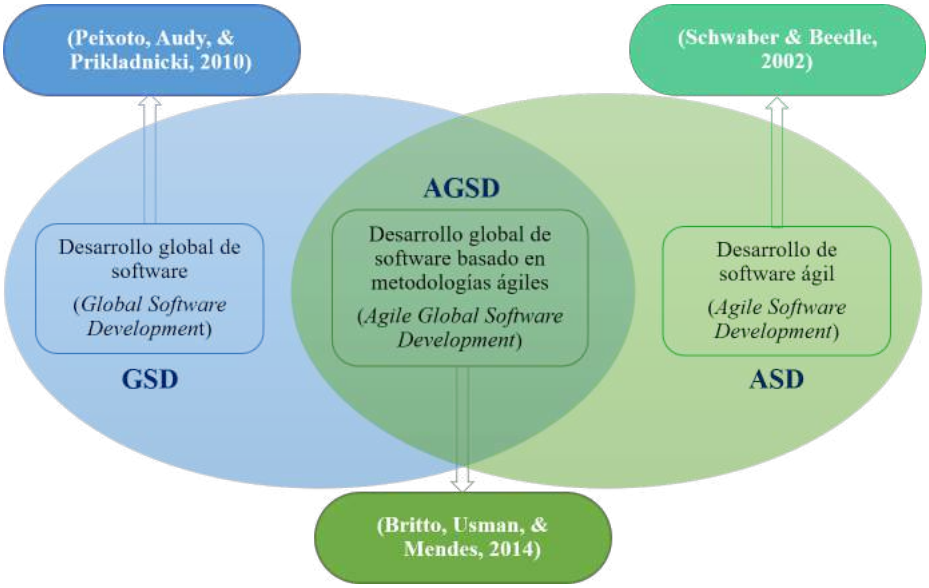

Figura 3.1: Contextos de desarrollo de software. Fuente: Elaboración propia.

Teniendo en cuenta una visión general, se puede constatar que existen distintas clasificaciones de los métodos de estimación de esfuerzo, cada una de ellas con el objetivo de representar categorías y subcategorías de métodos, estableciendo las particularidades de cada una. Los apartados a continuación muestran algunas de estas clasificaciones, así como también el uso de los predictores de esfuerzo y las métricas de precisión.

3.1.1. Métodos de estimación

A la hora de considerar un método de estimación, además del nivel de precisión, se debe considerar su viabilidad dentro de un marco organizativo con una serie de recursos disponibles (datos históricos disponibles, juicio de expertos, etc.) e incluso que dicho método tenga en cuenta el logro de objetivos organizativos (aportación de valor, entrega en plazo, etc.), lo que en definitiva viene a suponer que el proyecto tenga éxito (Trendowicz et al., 2011).

Numerosos métodos de estimación han sido propuestos por los investigadores desde el inicio de la estimación de software como área de investigación (Magne Jørgensen, 2014). Estos métodos abarcan desde los que están basados en la opinión intuitiva de expertos hasta aquellos compuestos por algoritmos complejos (Borade y Khalkar, 2013). Pueden utilizarse de forma aislada o combinados (híbridos) para aumentar su eficacia y precisión. De hecho, la inclusión de elementos subjetivos como la opinión de expertos en los modelos analíticos permite la reducción de la cantidad de variables que serían incluidas en el modelo, así como el uso de otro tipo de factores de carácter cualitativo.

Los primeros enfoques para estimar el esfuerzo en el desarrollo de software aparecieron a finales de la década de los 60 (Dejaeger, Verbeke, Martens, y Baesens, 2012) y se basaban fundamentalmente en la valoración de expertos (Nelson, 1967). En estos casos, un "experto" (persona con el conocimiento y experiencia suficientes en el área en cuestión y en el uso del tipo de aplicación de software a desarrollar) proponía una estimación del esfuerzo necesario para el proyecto de desarrollo de software en base a su experiencia. Posteriormente, aparecieron los modelos formales tales como COCOMO, SLIM y análisis de los puntos de función (*Function Points Analysis*). Regularmente, estos modelos formales requieren del uso intensivo de datos, para lo cual se valen de uno o varios repositorios de los que se puedan extraer muestras relevantes para el estudio.

3.1.1.1. Clasificación de los métodos de estimación

Existen varios esquemas de clasificación de métodos de estimación de esfuerzo de proyectos de desarrollo de software. La Figura 3.2 presenta una adaptación gráfica de la clasificación sugerida por S. K. Sehra et al. (2017) y Wen, Li, Lin, Hu, y Huang (2012).

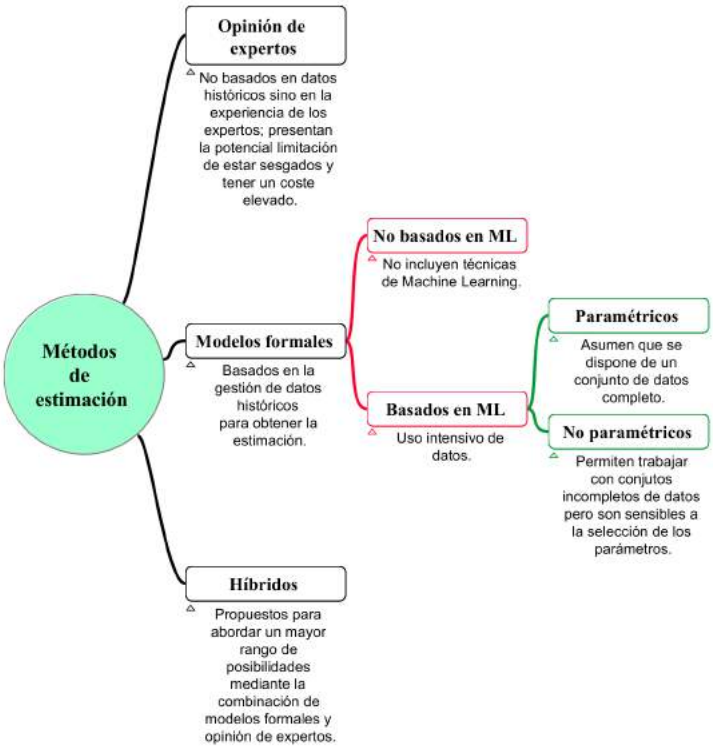

Figura 3.2: Métodos de estimación de esfuerzo. Fuente: Elaboración propia.

En esta misma perspectiva, la Tabla 3.1 desarrolla e ilustra la Figura 3.2, presentando algunos ejemplos de métodos de estimación para cada uno de los subgrupos referidos.

Tabla 3.1: Clasificación de métodos de estimación de esfuerzo (ejemplos).

Tipo de método	Referencias
Opinión de expertos:	
Expert Judgement	M. Shepperd y Cartwright (2001); Magne Jørgensen (2004a); Moløkken-Østvold, Haugen, y Benestad (2008); Magne Jørgensen (2010); Magne Jørgensen (2011); Magne Jørgensen y Løhre (2012)
Work Breakdown Structure (WBS)	Z. Li y Keung (2010)
Modelos formales no basados en ML:	
COCOMO	Boehm (1981)
COCOMO II	Boehm, Madachy, y Steece (2000)
SLIM	Putnam (1978)
Function Points Analysis (FPA)	Albrecht y Gaffney (1983)
Modelos formales basados en ML (Paramétricos):	
Case-Based Reasoning (CBR) o Analogy-Based Estimation (ABE)	Auer, Trendowicz, Graser, Haunschmid, y Biffl (2006); Chiu y Huang (2007); Kocaguneli, Menzies, y Keung (2011); Azzeh (2012); Kocaguneli, Menzies, Bener, y Keung (2012); Bardsiri, Abang Jawawi, y Khatibi (2014); Chu (2015)
Regression Analysis	Mittas y Angelis (2008)
Support Vector Regression (SVR)	Oliveira (2006)
Artificial Neural Networks (ANN)	Barcelos Tronto, Silva, y Santâ??Anna (2007); Hamza, Kamel, y Shams (2013)
Decision Trees (DT)	Nassif, Azzeh, Capretz, y Ho (2013)
Bayesian Networks (BN)	Dragicevic, Celar, y Turic (2017)
Modelos formales basados en ML (No paramétricos):	
Genetic Algorithms (GA)	Ferrucci, Gravino, Oliveto, y Sarro (2010)
Fuzzy Logic (FL)	Muzaffar y Ahmed (2010)
Windowing approach	Chris Lokan y Mendes (2009); Amasaki (2011)

Híbridos:	
Ensemble models	Papatheocharous y Andreou (2012)
Software Required Specification based estimation (SRS)	Sharma, Vardhan, y Kushwaha (2014)
Use case models	Issa, Odeh, y Coward (2006)

Trendowicz y Jeffery (2014) presentan un modelo de clasificación similar, aunque más estructurado y detallado. La Figura 3.3 muestra la estructura propuesta por estos autores.

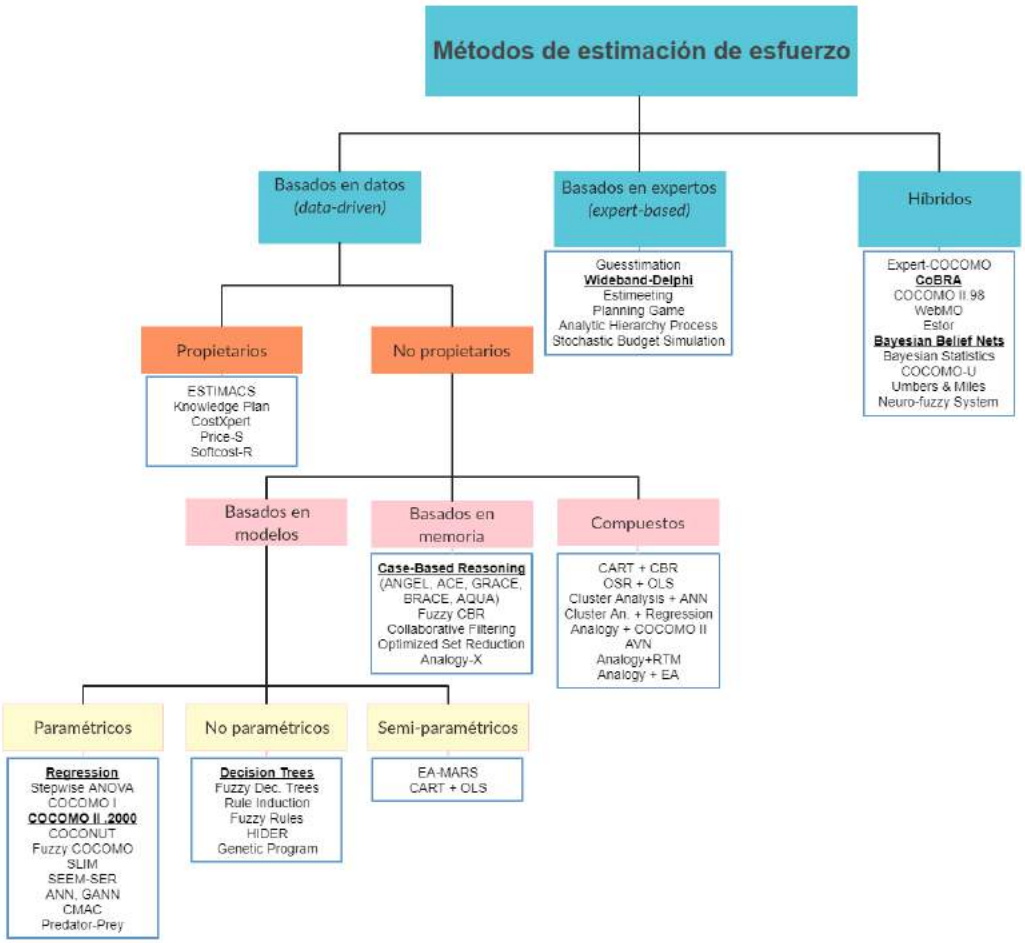

Figura 3.3: Clasificación de los métodos de estimación de esfuerzo según Trendowicz y Jeffery (2014).

Segun la figura anterior, los modelos de estimación de esfuerzo pueden clasificarse en tres categorías principales:

- **Basados en datos**: este grupo incluye aquellos métodos que predicen el esfuerzo basados únicamente en el análisis cuantitativo de datos históricos. Es decir, los

datos y características del proyecto actual son comparados con otros proyectos ya completados con el objetivo de encontrar relaciones que permitan proyectar una estimación del esfuerzo requerido. Estos métodos tienen entre sus ventajas la participación reducida de los "expertos", alta flexibilidad de los modelos de estimación, gran cantidad de soporte documental, disponibilidad en cualquier etapa del desarrollo de software y coste de aplicación relativamente bajo (que dependerá principalmente de la disponibilidad de los datos y las herramientas para manejarlos). Por otro lado, el uso de repositorio de datos históricos añade el riesgo de inconsistencia en los datos, cierto nivel de complejidad en el manejo de los mismos y, en algunos casos, una reusabilidad limitada de los resultados (Trendowicz y Jeffery, 2014, pp. 166–169).

- **Basados en expertos**: constituyen el subconjunto de métodos más utilizado, tanto por académicos como por los profesionales en la industria (Usman, Börstler, y Petersen, 2017). Comúnmente, 70-80 % de las estimaciones realizadas en la industria son hechas por uno o más expertos sin el uso de modelos de estimación formales, sino basados en la experiencia y el entendimiento del contexto organizacional del proyecto (Trendowicz y Jeffery, 2014, p. 169). No obstante, la participación activa y el tiempo requerido por parte de los miembros del equipo de proyecto (específicamente los directores y jefes de equipo) puede resultar en un incremento en los costes del proyecto. Además, el grado de incertidumbre aumenta debido a que las estimaciones están basadas apreciaciones subjetivas.
- **Híbridos**: estos métodos, tal y como el nombre sugiere, utilizan una combinación de los dos anteriores. Dicha combinación puede ser vista desde dos perspectivas: (a) la integración de diferentes paradigmas de estimación para conformar un modelo híbrido, o (b) el uso de múltiples enfoques aplicados independientemente sobre el mismo proyecto para obtener resultados que puedan ser combinados en una estimación final del esfuerzo (Trendowicz y Jeffery, 2014, pp. 143–144).

Esta clasificación ha sido realizada en base al tipo de datos que utilizan como entrada y el principio de estimación empleado. Sin embargo, la mayoría de estos métodos no han sido aplicados más allá del ámbito académico y de investigación; solo unos cuantos (que se encuentran resaltados en la Figura 3.3) han sido aceptados por la industria y un número inferior son los realmente aplicados en la práctica.

3.1.2. Predictores de esfuerzo

Según Usman et al. (2017), el tamaño y los factores de coste (*cost drivers*) constituyen los predictores de esfuerzo, siendo el tamaño el más importante. Los apartados 3.1.2.1 y 3.1.2.2 desarrollan estos dos elementos.

3.1.2.1. Métricas de tamaño

En proyectos de software, el esfuerzo requerido para el desarrollo de una aplicación (análisis, diseño, codificación, prueba y documentación) dependerá del tamaño del software a ser desarrollado (J. Singh, 2017). Trendowicz y Jeffery (2014) lo refieren como el *volumen* del proyecto de software, comúnmente medido como el tamaño de los entregables del proyecto. Las principales métricas utilizadas para estos fines son las líneas de código fuente, los puntos de función, los puntos de historia y los puntos de caso de uso.

Líneas de código fuente (*Source Lines of Code*)

El método de medición mediante SLOC consiste en la estimación de las líneas de código fuente requeridas para el desarrollo de una aplicación de software, lo que se traduciría en una estimación del tamaño del producto de software que ha de ser desarrollado. SLOC se encuentra entre los primeros métodos de medición de tamaño, con la característica de que el conteo puede realizarse con mayor facilidad que en otras métricas, incluso de forma automática (Galorath y Evans, 2006).

Este método no es el más recomendado en la actualidad. Esto se debe, principalmente, a la dificultad de determinar la cantidad de líneas de código al inicio de un proyecto, sobre todo cuando se van a utilizar diversas herramientas y lenguajes de programación (Bundschuh y Dekkers, 2008). De manera que, las estimaciones mediante SLOC han de ser más precisas hacia el final del proyecto, cuando las líneas de código pueden ser correctamente contadas.

Puntos de función (*Function Points*)

Functional Size Measurement (FSM) se basa en la evaluación de los requerimientos funcionales de los usuarios para determinar los "puntos de función" y con ello estimar el tamaño y esfuerzo requeridos para un proyecto de software determinado (Bundschuh y Dekkers, 2008). *Function Point Analysis* (FPA) fue el primer método utilizado para adoptar este concepto, desarrollado por Allan J. Albrecht en la década de 1970 (ISO/IEC, 2012). FSM, a diferencia de SLOC, es independiente del lenguaje de programación utilizado y es más entendible por parte de los clientes, dado que se basa en la funcionalidad que ha de ser entregada a los usuarios y no en la manera que dicha funcionalidad ha de ser desarrollada (Bundschuh y Dekkers, 2008, p. 211).

Existen cinco métodos FSM estandarizados por la ISO/IEC[1], los cuales son: **IFPUG** (ISO/IEC 20926), **COSMIC** (ISO/IEC 19761), **NESMA** (ISO/IEC 24570), **FiSMA** (ISO/IEC 29881) y **Mark II** (ISO/IEC 20968). Entre estos, IFPUG (International Function Point Users Group)[2] es la asociación de medición de software más grande en

[1] www.iso.org/standard/60176.html
[2] www.ifpug.org

el mundo y el método, que lleva el mismo nombre, es el más utilizado en la selección de proyectos de repositorios como ISBSG (González-Ladrón-de-Guevara, Fernández-Diego, y Lokan, 2016). Además de este, COSMIC y NESMA (implementado sobre todo en los Países Bajos) son comúnmente utilizados (Huijgens, Bruntink, Deursen, Storm, y Vogelezang, 2016).

Puntos de historia (*Story Points*)

Un punto de historia es una medida que expresa de forma relativa el tamaño de una historia de usuario (*user story*) o funcionalidad (Coelho y Basu, 2012). Este valor depende de factores como la complejidad de desarrollo, el esfuerzo requerido, el riesgo inherente al desarrollo, etc. Al ser un valor de tamaño relativo, una US a la que se asigna un SP igual a 2 ha de ser el doble de una US cuyo SP es igual a 1 (en complejidad, esfuerzo, riesgo, tiempo, etc.). Los puntos de historia son la medida de tamaño más utilizada en ASD (Usman y Britto, 2016).

Frente a las estimaciones expresadas en horas, que suelen ser más restrictivas, imprecisas y confusas para el equipo de desarrollo, los puntos de historia permiten obtener valores más precisos, reducen el tiempo de planificación, predicen con mayor exactitud las fechas de entrega y ayudan a los equipos a mejorar su rendimiento (Sutherland, 2013). Además, los puntos de historia son una métrica más relevante para la medición del trabajo realizado (Cobb, 2015, p. 106), dado que están relacionados directamente con las funcionalidades a ser desarrolladas y no solo con el tiempo de desarrollo requerido. Las series Fibonacci (1,2,3,5,8,13,21...) y *T-shirt sizes* (S, M, L, XL...) son métricas utilizadas para la estimación de los puntos de historia (p. 107).

Puntos de caso de uso (*Use Case Points*)

El método basado en puntos de caso de uso (UCP) fue desarrollado por Gustav Karner en 1993. Este método, similar a Mark II, fue ideado para aquellos desarrolladores que hacen uso de las herramientas del lenguaje de modelado UML (*Unified Modeling Language*) y del proceso de desarrollo RUP (*Rational Unified Process*) (Chemuturi, 2009). Los casos de uso describen las tareas elementales que un sistema ha de llevar a cabo y las relaciones entre estas tareas y el entorno exterior (Galorath y Evans, 2006, p. 263).

Según (Saroha y Sahu, 2015), el cálculo de UCP se realiza teniendo en cuenta los siguientes pasos:

1) Cálculo del peso no ajustado del actor (*Unadjusted Actor Weight*, UAW), basado en los tipos de interacción del usuario con el sistema (simple, medio, complejo).
2) Cálculo del peso del caso de uso (*Unadjusted Use Case Weight*, UUCW), basado en el número de interacciones del caso de uso con el sistema (simple, medio, complejo).

3) Cálculo de los puntos de caso de uso no ajustados (*Unadjusted Use Case Point*, UUCP), obtenido mediante la adición de los dos pesos anteriores.
4) Cálculo de los puntos de caso de uso ajustados (*Adjusted Use Case Point*, AUCP), obtenido mediante la multiplicación de UUCP por el factor de complejidad técnica (*Technical Complexity Factor*, TCF) y el factor de complejidad del entorno (*Environmental Complexity Factor*, ECF).

Los puntos de caso de uso han demostrado ser una métrica significante y consistente del tamaño del proyecto. Sin embargo, Galorath y Evans (2006) considera que una de las razones por la que los puntos de casos de uso no son ampliamente utilizados para la estimación del tamaño del software es la falta de medidas ampliamente aceptadas para este método.

3.1.2.2. Factores de coste

Los factores de coste permiten determinar el esfuerzo requerido para completar un proyecto de software mediante el uso de multiplicadores relacionados con el entorno, equipo de trabajo, herramientas de desarrollo, etc. (Alsmadi y Nuser, 2013). El modelo COCOMO (*Constructive Cost Model*) (Boehm, 1981; Boehm, Madachy, y Steece, 2000), por ejemplo, a través de parámetros relacionados con el coste del proyecto, proporciona un valor estimado del mismo. En dicho modelo se define un conjunto de factores que pueden ser categorizados en los siguientes grupos (Tsui, Karam, y Bernal, 2016):

- Atributos del producto
  - Seguridad requerida
  - Tamaño de la base de datos
  - Complejidad del producto
- Atributos del computador
  - Restricciones del tiempo de ejecución
  - Restricciones de memoria principal
  - Complejidad del computador
  - Tiempo de respuesta del computador
- Atributos de las personas
  - Capacidad de los analistas
  - Capacidad de los programadores
  - Experiencia en el uso de aplicaciones
  - Experiencia en el uso de máquinas virtuales
  - Experiencia con el lenguaje de programación
- Atributos del proyecto
  - Uso de prácticas modernas
  - Uso de herramientas de software
  - Tiempo de desarrollo requerido

3.1.3. Métricas de precisión

Al comparar modelos de estimación o evaluar el desempeño de los mismos se hace necesario el uso de ciertas técnicas estadísticas que permitan validar los resultados obtenidos. Las principales métricas de precisión pueden observarse en la Tabla 3.2.

Tabla 3.2: Métricas de precisión en estimación de software. Fuente: Malathi y Sridhar (2012).

Métrica de precisión	**Descripción**	**Fórmulas**
Mean Magnitude Relative Error (MMRE)	Permite obtener la magnitud de error relativo de la estimación realizada.	$MRE_i = \frac{\lvert EE_i - ER_i \rvert}{ER_i}$ $MMRE = \frac{1}{N}\sum_{i=1}^{N} MRE_i$
Prediction (x) o PRED(x)	Determina el porcentaje de elementos estimados cuyo valor relativo de error es menor que x.	$PRED(x) = \frac{k}{N}$
Mean Absolute Relative Error (MARE)	Medida de la precisión en valores porcentuales.	$MARE = \frac{1}{N}\sum_{i=1}^{N} MRE * 100$
Balance Relative Error (BRE)	Determina el nivel de parcialidad y desviación estándar de los valores estimados.	$BRE = \frac{\lvert EE - ER \rvert}{min(EE, ER)}$
Median Magnitude of Relative Error (MdMRE)	Indica la mediana de los valores de MRE.	$MdMRE = median(MRE)$
Root Mean Square Error (RMSE)	Calcula la variación entre los valores predichos por un modelo o estimador y los valores reales.	$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(EE - ER)^2}$

Donde,

- EE = esfuerzo estimado
- ER = esfuerzo real
- i = elemento estimado (tarea, iteración, proyecto, etc.)
- N = cantidad total de elementos estimados
- k = número de elementos estimados para los cuales se cumple que $MRE >= x$.

MMRE y PRED constituyen las dos métricas de precisión más utilizadas en ingeniería de software (M. Jørgensen, 2007; Kitchenham, Pickard, MacDonell, y Shepperd, 2001; Usman et al., 2014), las cuales fueron introducidas por S. D. Conte, Dunsmore, y Shen (1985). Algunas alternativas frente a MMRE que se proponen son *Mean Balanced Relative Error* (MBRE), *Weighted Mean of Quartiles of relative errors (WMQ)* y *Mean Variation from Estimate* (MVFE) (M. Jørgensen, 2007). Además, este autor señala que el nivel de precisión aceptable para un modelo específico dependerá, entre otros factores, de la exactitud de medios alternativos para proveer valores estimados de esfuerzo en un contexto en particular.

3.2. Estimación de esfuerzo en ASD

En entornos de desarrollo ágil, la estimación del esfuerzo desempeña un papel importante durante la planificación de las iteraciones y del proyecto en general (Usman et al., 2017). En una revisión sistemática de literatura (*Systematic Literature Review*, SLR) realizada por Usman et al. (2014), junto con el posterior estudio realizado mediante una encuesta a profesionales de la industria de desarrollo de software (Usman, Mendes, y Börstler, 2015), se identifican una serie de características respecto a la estimación de esfuerzo en ASD, las cuales se resumen en los siguientes puntos:

- La estimación de esfuerzo en ASD es una actividad principalmente llevada a cabo durante las etapas de planificación del *release* o de cada iteración; en algunos casos, también se realizan estimaciones de esfuerzo durante las reuniones diarias o inclusive durante la fase de oferta.
- Las fases de implementación y prueba son las principales actividades para las cuales se realiza estimación de esfuerzo.
- Scrum y XP son las principales metodologías de desarrollo ágil utilizadas en la industria.
- Las técnicas de estimación más utilizadas son las basadas en juicios subjetivo de expertos, tales como el *Planning Poker* y el método UCP.
- MMRE es la métrica mayormente utilizada en la literatura relacionada con estimación de esfuerzo en ASD. Sin embargo, en la práctica solo suele realizarse una simple comparación entre los valores estimados y los reales.
- Los factores de tamaño y coste constituyen los predictores de esfuerzo. Para el cálculo del tamaño, las métricas de FSM, los puntos de historia y los puntos de casos de uso son utilizadas con mayor frecuencia. En cuanto a los factores de

coste, aquellos relacionados con el equipo (habilidades y experiencia) son los más significativos, en concordancia con los valores que promueve ASD, donde se valora más a los individuos y equipos que a los procesos y herramientas.

Los entornos de desarrollo ágil han permitido el regreso a la estimación de esfuerzo basada en la valoración de expertos (Moløkken-Østvold, Haugen, y Benestad, 2008). Sin embargo, en ausencia de datos históricos y expertos, ni la estimación por analogía ni el *Planning Poker* son de gran utilidad. Por esto es preciso apoyarse en técnicas de estimación matemáticas que proporcionen estimaciones más precisas de la fecha de entrega del proyecto, su coste, esfuerzo y duración (Popli y Chauhan, 2014). En la actualidad, las áreas de investigación que están demandando una mayor atención por parte de los investigadores son las relativas a las métricas de tamaño, las técnicas de *Machine Learning*, los métodos basados en juicios de expertos, *ensemble models*, y las técnicas de validación de los modelos de estimación (Idri, Abnane, y Abran, 2015).

3.2.1. *Planning Poker*

Planning Poker es un método de estimación de esfuerzo basado en el consenso de un grupo de expertos (Gandomani, Wei, y Binhamid, 2014). Fue primero introducido por Grenning (2002) y popularizado más adelante por Cohn (2005), considerándolo como un método compatible con el enfoque hacia las personas que es característico de las metodologías ágiles.

El proceso de estimación de esfuerzo suele ocurrir como se describe en la Figura 3.4. Las cartas utilizadas suelen contener valores correspondientes a la serie Fibonacci, los cuales han de representar el número de *story points*, días ideales o cualquier unidad de estimación que el equipo esté utilizando (Gandomani et al., 2014).

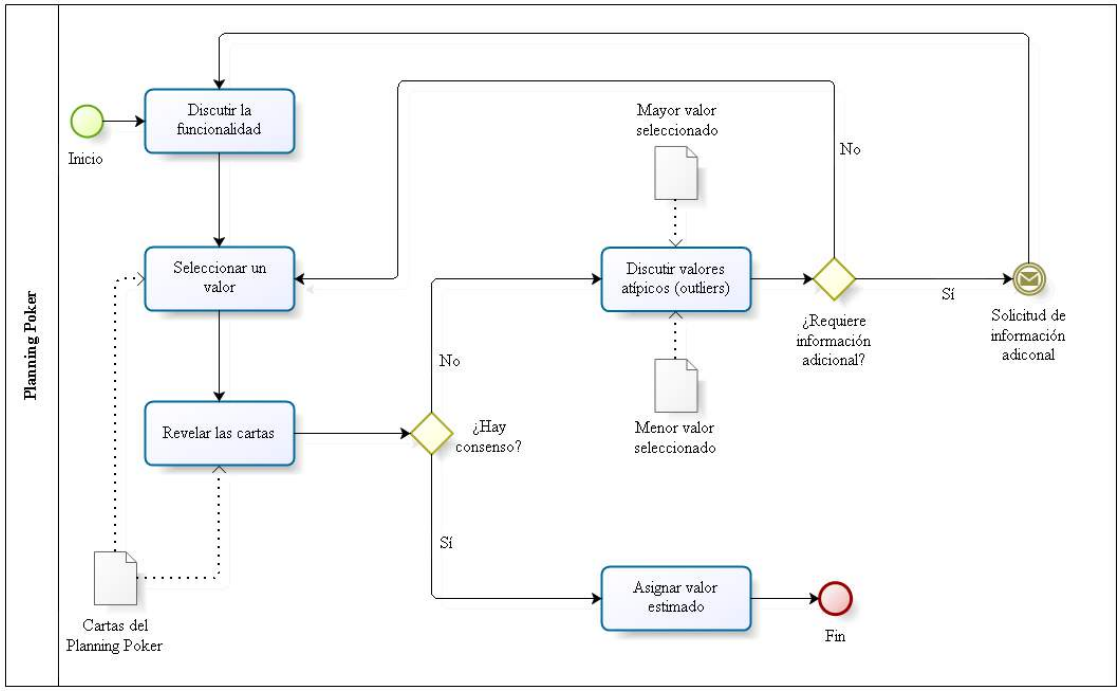

Figura 3.4: Proceso de estimación mediante Planning Poker. Fuente: Elaboración propia.

El proceso comienza con la discusión de la funcionalidad, realizando preguntas al representante por parte del cliente (*product owner*) según sea requerido. Luego de esto, cada estimador o experto selecciona una carta de forma privada, la cual representará su estimación para dicha funcionalidad. Todas las cartas son reveladas al mismo tiempo. Si hay un consenso, entonces el valor seleccionado se asigna como estimado de la funcionalidad en cuestión. En caso contrario, se procede a discutir los valores aislados o *outliers*; esto es, los estimadores que hayan seleccionado el menor y el mayor valor han de exponer y justificar sus razones y opiniones al respecto.

El proceso es repetido hasta llegar a un consenso o decidir que no se cuenta con la información necesaria para realizar la estimación de una funcionalidad determinada, en cuyo caso se ha de solicitar la información adicional a las partes correspondientes para volver a realizar la estimación.

Entre las razones por las que se recomienda utilizar *Planning Poker* para sesiones de estimación de *story points*, Cohn (2005) destaca:

- Combinación de las opiniones de múltiples expertos, que forman parte del equipo multifuncional del proyecto, para realizar la estimación.
- Posibilidad de dar clara justificación de los valores estimados propuestos, mejorando la precisión y procurando que no haya falta de información.
- Empleo de discusiones de grupo como base para realizar las estimaciones conduce a mejores resultados al utilizar el promedio de los valores estimados individuales

y eliminar los valores extremos.

3.2.2. *Wideband Delphi*

Wideband Delphi, al igual que *Planning Poker*, se basa en la estimación por consenso. Este método es ampliamente utilizado cuando la estimación se basa en la estructura de desglose de trabajo (*Work Breakdown Structure*, WBS), así como también para la estimación indirecta de la calidad, dado que el esfuerzo para desarrollar software comprende los costes de calidad, creación y estimación (Gandomani et al., 2014). Asimismo, se da mayor énfasis a la comunicación e interacción entre los participantes.

Según Stellman y Greene (2005), para utilizar este método se debe seguir el proceso Delphi, que incluye dos reuniones dirigidas por un moderador y compuesta por un equipo de 3 a 7 personas. En la primera reunión, el equipo de estimación crea la WBS y discute la suposiciones respecto a cada elemento descrito en esta estructura. Una vez finalizada la reunión, cada miembro del equipo de estimación ha de asignar un valor estimado del esfuerzo para cada actividad. En la segunda reunión se revisan los estimados de forma grupal y se llega a un consenso. Al finalizar este proceso, el *project manager* sumariza los resultados y los revisa con el equipo, quedando listos para ser utilizados como base para la planificación del proyecto de software.

Las principales ventajas de Wideband Delphi son la simplicidad del proceso, la documentación y evaluación de todas las asunciones, la colaboración del equipo de desarrollo y la fiabilidad de la estimación basada en consenso frente a la estimación individual. Como desventajas, por otro lado, están la necesidad de cooperación directa de la alta gestión, las posibles dificultades a la hora de intentar llegar a un acuerdo grupal y la dependencia en la experiencia del equipo de estimación (Gandomani et al., 2014).

3.2.3. Estimación por analogía

En la estimación de esfuerzo por analogía se utiliza información sobre proyectos anteriores, de características similares, para predecir el esfuerzo de un nuevo proyecto (J. Li, Ruhe, Al-Emran, y Richter, 2007). Esta similitud puede incluir el tipo de aplicación, el lenguaje de programación utilizado, la metodología de desarrollo empleada y las funcionalidades (Leung, 2002, p. 159). En la clasificación propuesta por Trendowicz y Jeffery (2014) (ver Figura 3.3), este método puede encontrarse en la categoría de modelos no propietarios basados en memoria (por ejemplo, *Case-Based Reasoning* o CBR, y *Analogy-X*) y los compuestos (por ejemplo, *Analogy + COCOMO II*).

Según M. Shepperd y Schofield (1997), las principales actividades en la estimación por analogía son: (a) identificación del problema como un nuevo caso, (b) búsqueda de casos similares en un repositorio, (c) reutilización del conocimiento derivado de casos

anteriores y (d) propuesta de una solución para el caso nuevo. Dicha solución ha de ser revisada a la luz de casos reales y los resultados han de ser añadidos al repositorio de casos completos. La Tabla 3.3 resume las principales ventajas y desventajas de estos modelos de estimación.

Tabla 3.3: Ventajas y desventajas de los modelos basados en analogía. Fuentes: Leung (2002) y J. Li et al. (2007).

Ventajas (+)	Desventajas (-)
▪ Son más propensos a ser aceptados por los usuarios, frente a resultados obtenidos mediante el uso de modelos formales, dado que la estimación por analogía se asemeja más a la forma de razonamiento humano. ▪ Pueden lidiar con dominios poco conocidos y de múltiples variables debido a que la solución estará basada en lo que ha ocurrido realmente. ▪ Evitan los problemas asociados a la elicitación y recopilación del conocimiento de expertos. ▪ Pueden ser aplicados en las fases iniciales del proyecto. ▪ Pueden manejar dimensiones de similitud con diferentes escalas.	▪ Solo puede ser utilizado cuando existe un histórico de datos de otros proyectos. ▪ La calidad de los estimados dependerá principalmente de la calidad del histórico de datos. ▪ La precisión del método dependerá de la posibilidad de encontrar analogías (casos similares). ▪ En caso de no encontrar algunas de las dimensiones de similitud (valores perdidos), se necesitará determinar una serie de reglas de adaptación apropiadas.

3.2.4. Método de puntos de casos de uso (*Use Case Point*)

Introducido por Karner (1993), el método UCP utiliza diagramas de caso de uso para realizar la estimación. Los casos de uso son escenarios que describen como un conjunto de actores (humanos, dispositivos, sistemas de software) puede alcanzar un objetivo mediante la interacción con el sistema que está siendo descrito (Ochodek, 2016). El cálculo de los puntos de caso de uso se realiza como se ha descrito anteriormente en la sección 3.1.2.1.

En cuanto al uso de este método en entornos ágiles, Kassab (2015) destaca que las técnicas basadas en escenarios para expresar requerimientos funcionales son también utilizadas por los equipos de desarrollo ágil. Jacobson, Spence, y Kerr (2016), además,

propone *Use-Case 2.0* como una alternativa ágil inspirada por las historias de usuario y las metodologías Scrum y Kanban. Estos autores consideran los siguientes principios básicos para la implementación exitosa de los casos de uso: (1) mantener la simplicidad mediante el uso de historias; (2) entender el panorama general del sistema, cómo este ha de ser usado, quienes inician la interacción, y qué otras partes están involucradas; (3) enfocarse en el valor que el sistema ha de proveer a los usuarios finales y a otras partes interesadas; (4) construir el sistema en partes pequeñas, cada una con valor añadido para el usuario; (5) entregar el sistema de forma incremental, proveyendo una versión que pueda ser sometida a prueba por parte de los usuarios; y (6) adaptar según las necesidades del equipo.

Otros ejemplos de adaptaciones del método UCP para el desarrollo de software ágil son descritos por Khatri, Malhotra, y Johri (2016) y Hamouda (2014).

3.2.5. Métodos basados en regresión

El análisis de regresión es una herramienta estadística para la investigación de las relaciones entre dos o más variables; en su forma más simple, muestra la relación entre una variable independiente (X) y una variable dependiente (Y) (Anandhi y Chezian, 2014). La ventaja de los modelos de regresión está en que cuentan con una base matemática sólida, así como también la posibilidad de identificar la línea o curva que mejor se ajusta al conjunto de datos analizados (Finnie, Wittig, y Desharnais, 1997). Estos métodos suelen ser simples y ampliamente aceptados y utilizados en otros modelos como COCOMO II y SLIM (Saroha y Sahu, 2015).

Los modelos de regresión son utilizados regularmente para la validación de otros métodos de estimación de esfuerzo. Por ejemplo, Magne Jørgensen (2004) hace uso de la regresión lineal para modelar la precisión y parcialidad de la estimación realizada. El uso de la regresión lineal se debió principalmente a la simplicidad de estos modelos y la existencia de herramientas de análisis para el apoyo en la interpretación de las relaciones encontradas. Anandhi y Chezian (2014), por otro lado, presentan la evaluación de la estimación de esfuerzo utilizando el repositorio COCOMO, mediante el uso de algoritmos como M5 y regresión lineal.

Los métodos basados en regresión pueden ser clasificados como estándar o robusto (Boehm, Abts, y Chulani, 2000, p. 31). La regresión estándar se refiere al clásico enfoque general de regresión lineal mediante el uso de mínimos cuadrados, mientras que la regresión robusta sería una mejora de la anterior, con el objetivo de dar solución al problema de los valores aislados que suelen ser observados en conjuntos de datos relacionados a proyectos de desarrollo de software (pp. 31-33).

3.2.6. Métodos basados en *Machine Learning*

Los métodos basados en *Machine Learning* (ML) resultan de gran utilidad para la construcción de modelos predictivos para la determinación de valores de esfuerzo en el campo del desarrollo de software. Uno de los aspectos más importantes que permiten el uso de técnicas de ML para la estimación de esfuerzo en desarrollo de software es el hecho de que investigaciones previas ya han identificado un conjunto significativo de atributos relevantes para la estimación de esfuerzo y, además, se cuenta con históricos de datos que han sido definidos en relación a estos atributos (Srinivasan y Fisher, 1995). Según Monika y Sangwan (2017), entre las principales técnicas de ML utilizadas para estimación de esfuerzo se pueden mencionar las siguientes:

- ***Artificial Neural Network*** (ANN). Las redes neuronales, que intentan emular el funcionamiento del cerebro humano, son ampliamente utilizadas en distintos campos de estudio. Algunas aplicaciones de ANN en estimación de esfuerzo se pueden observar en Papatheocharous y Andreou (2007); Barcelos Tronto, Simoes da Silva, y Sant'Anna (2008); y Rijwani y Jain (2016).
- ***Fuzzy Logic***. Esta técnica es utilizada para resolver problemas que se encuentran descritos mediante cuantificadores lingüísticos o que son muy complejos para ser comprendidos cuantitativamente (Alostad, Abdullah, y Aali, 2017). Esta basada en la teoría conjuntos difusos introducida por Lotfy Zadeh en 1965 (Hamdy, 2012). Algunos de los estudios en los que se presenta la implementación de la lógica difusa para la estimación de esfuerzo pueden encontrarse en Azzeh, Neagu, y Cowling (2008) y Tang, Zhang, Wang, Wang, y Liu (2015).
- ***Genetic Algorithms*** (GA). Son algoritmos de búsqueda adaptativos basados en la teoría de Darwin sobre selección natural; esto es, se considera una población de "individuos" con todas las posibles soluciones para el problema en cuestión, de las cuales se han de seleccionar las que mejor se ajusten a los criterios definidos (B. K. Singh y Misra, 2012). En (Benala, Dehuri, Satapathy, y Madhurakshara, 2012) se muestran algunos ejemplos de esta técnica como modelo de estimación.
- ***Support Vector Machine*** (SVM). Este método permite reconocer patrones para resolver problemas en análisis de clasificación y regresión (Aljahdali, Sheta, y Debnath, 2015). En (Oliveira, 2006), por ejemplo, es utilizado en comparación con el método de regresión lineal para la estimación de esfuerzo utilizando el repositorio de proyectos de software de la NASA.
- ***Bayesian Network***. Es útil para representar las relaciones y dependencias entre variables (Monika y Sangwan, 2017, p. 96). Según Dragicevic, Celar, y Turic (2017), algunos problemas típicos de los modelos de estimación de esfuerzo, costo y calidad pueden ser solucionados mediante el uso de redes bayesianas, debido a que estas son flexibles y simples de construir (basadas puramente en juicio de expertos, datos empíricos o la combinación de ambos), reflejan relaciones causales, incorporan explícitamente la incertidumbre como una distribución probabilística de cada variable, permiten representar gráficamente el modelo de estimación y pueden ser implementadas con repositorios que contengan datos perdidos.

Capítulo 4

Revisión sistemática de literatura sobre métodos de estimación de esfuerzo en desarrollo de software ágil

Una Revisión Sistemática de Literatura (SLR = *Systematic Literature Review*), según Kahn (2003), es un trabajo de investigación que identifica un conjunto de estudios relevantes y sumariza sus resultados utilizando una metodología científica. Consiste de un proceso bastante riguroso, que provee un medio transparente y sistemático para la recolección, síntesis y evaluación de los resultados de la investigación acerca de un tópico o cuestión particular (Sweet y Moynihan, 2007). Este concepto fue originalmente concebido en el ámbito de la Medicina, como medio para la investigación basada en evidencias. El propósito de las mismas, por tanto, no es solo reunir toda la evidencia existente en relación a un conjunto de preguntas de investigación, sino también servir de apoyo para el desarrollo de guías prácticas basadas en evidencia empírica, en contraste con revisiones de expertos basadas en una selección de literatura *ad hoc* (Kitchenham et al., 2009).

En este capítulo se ha desarrollado una SLR sobre métodos de estimación de esfuerzo en entornos de desarrollo ágil. Esta pretende complementar los resultados presentados en una SLR anterior (Usman et al., 2014), extendiendo el marco temporal y comparando los resultados obtenidos para cada caso. Para ello, en primer lugar se explica la metodología utilizada y el proceso seguido en el desarrollo de la SLR. En segundo lugar se presentan los resultados obtenidos en la selección de los estudios primarios, que a su vez conformarán el subconjunto de estudios que serán analizados en la tercera sección de este apartado. Por último se resumen las principales conclusiones de la investigación realizada.

4.1. Antecedentes

Tal y como se ha mencionado anteriormente, el presente estudio está basado en la revisión presentada por Usman et al. (2014). Esta revisión fue realizada con la intención de agregar todos los estudios relacionados con la estimación de esfuerzo en entornos de ASD. Para ello, los autores analizan un total de 25 estudios primarios, de los cuales se obtienen los resultados resumidos en la Tabla 4.1.

Tabla 4.1: Resultados de Usman et al. (2014).

Aspecto	Principales conclusiones
Técnicas de estimación de esfuerzo y tamaño en ASD	De entre las técnicas de estimación de esfuerzo que han sido investigadas, el juicio de expertos, Plannng Poker y UCP se muestran como las más utilizadas. En relación a las métricas de precisión, MRE y MMRE son frecuentemente utilizadas, mientras que solo dos estudios hacen uso de BRE. El 26 % de los estudios no indica haber aplicado ninguna métrica para el cálculo de la precisión de la estimación; en la mayoría de casos que sí se indica la métrica de precisión, se supera el umbral del 25 %.
Predictores de esfuerzo utilizados	Story Points y Use Case Points son las métricas de tamaño más utilizadas. No obstante, no hay consenso en relación a los factores de coste que deberían ser utilzados en entornos de desarrollo ágil.
Características de los datasets utilizados	Los repositorios de datos de dominio industrial son utilizados en la mayoría de estudios (76 %). Además, el 72 % de los repositorios considerados son de carácter interno a la compañía, no conteniendo datos de proyectos de más de una organización.
Métodos ágiles, actividades y niveles de planificación considerados	XP y Scrum son las metodologías ágiles más utilizadas, además de ser las únicas identificadas en los estudios primarios analizados por los autores. Solo las fases de implementación y pruebas han sido consideradas como actividades de desarrollo por separado. El 36 % de los estudios ha tenido en cuenta todas las actividades del proceso de desarrollo para la estimación del esfuerzo. Por otro lado, se indica que la estimación de esfuerzo es llevada a cabo principalmente durante la planificación del proyecto o la iteración, a pesar de que el 48 % de los estudios no especifica el nivel de planificación en el cual la estimación de esfuerzo fue desarrollada.

4.2. Metodología

Como guía principal para la elaboración de esta SLR, se han tomado en consideración los lineamientos propuestos en (Kitchenham, Budgen, y Brereton, 2015). El estudio anterior estuvo basado en (Kitchenham y Charters, 2007). El proceso seguido y los resultados de la revisión de los estudios primarios se analizan a continuación.

4.2.1. Preguntas de investigación

Las preguntas de investigación (RQ = *Research Questions*) en torno a las cuales se desarrolla este estudio son las siguientes:

- RQ1: ¿Qué métodos han sido utilizados para la estimación de esfuerzo o tamaño en ASD?
  - RQ1a: ¿Qué métricas han sido utilizadas para determinar la precisión de los métodos de estimación?
  - RQ1b: ¿Cuál es el nivel de precisión de estos métodos?
- RQ2: ¿Qué predictores de esfuerzo han sido utilizados en estudios sobre estimación de esfuerzo en ASD?
- RQ3: ¿Cuáles son las características del dataset utilizado en estudios sobre estimación de esfuerzo/tamaño en ASD?
- RQ4: ¿Qué metodologías ágiles han sido utilizadas en estudios sobre estimación de esfuerzo/tamaño?
  - RQ4a: ¿Qué actividades de desarrollo han sido objeto de estimación?
  - RQ4b: ¿A qué niveles de planificación se realiza la estimación?

4.2.2. Proceso de búsqueda

En el estudio previo se consultaron ocho base de datos o motores de búsqueda de referencias bibliográficas, los cuales son *ACM Digital Library, IEEE Xplore, Science Direct, SCOPUS, WOS, EI Compendex, INSPEC y Springer Links*. Sin embargo, para el presente trabajo de investigación sólo se han considerado las primeras cinco de las librerías digitales antes mencionadas, esto debido a que no se pudo obtener acceso para consultar en la base de datos de registros bibliográficos del resto de las librerías. En todo caso, los motores consultados se encuentran entre las principales fuentes de referencias para búsqueda de artículos de revistas y conferencias a nivel internacional, según se menciona en (Kitchenham et al., 2015, pp. 62, 310, 311).

Se utilizaron las mismas palabras claves incluidas en el *Search Query* (SQ) o cadena de búsqueda original, aunque sí fue necesario ajustarlo según los parámetros requeridos en cada motor o librería digital. La Tabla 4.2 muestra el SQ utilizado para cada motor de búsqueda.

Tabla 4.2: Adaptación de la cadena de búsqueda.

Motor de búsqueda	Cadena de búsqueda
ACM Digital Library - IEEE Xplore	(Agile OR "extreme programming" OR Scrum OR "feature driven development" OR "dynamic systems development method" OR "crystal software development" OR "crystal methodology" OR "adaptive software development" OR "lean software development") AND (estimat* OR predict* OR forecast* OR calculat* OR assessment OR measur* OR sizing) AND (effort OR resource OR cost OR size OR metric OR "user story" OR velocity) AND (software)
SCIENCE DIRECT	TITLE-ABSTR-KEY ((Agile OR "extreme programming" OR Scrum OR "feature driven development" OR "dynamic systems development method" OR "crystal software development" OR "crystal methodology" OR "adaptive software development" OR "lean software development")) and TITLE-ABSTR-KEY ((estimat* OR predict* OR forecast* OR calculat* OR assessment OR measur* OR sizing) AND (effort OR resource OR cost OR size OR metric OR "user story" OR velocity) AND (software))
SCOPUS	TITLE-ABS-KEY((agile OR "extreme programming" OR Scrum OR "feature driven development" OR "dynamic systems development method" OR "crystal software development" OR "crystal methodology" OR "adaptive software development" OR "lean software development") AND (estimat* OR predict* OR forecast* OR calculat* OR assessment OR measur* OR sizing) AND (effort OR resource OR cost OR size OR metric OR "user story" OR velocity) AND (software))
Web of Science (WOS)	TS=((Agile OR "extreme programming" OR Scrum OR "feature driven development" OR "dynamic systems development method" OR "crystal software development" OR "crystal methodology" OR "adaptive software development" OR "lean software development") AND (estimat* OR predict* OR forecast* OR calculat* OR assessment OR measur* OR sizing) AND (effort OR resource OR cost OR size OR metric OR "user story" OR velocity) AND (software))

Estas modificaciones sirven principalmente para limitar la búsqueda únicamente en los metadatos (título, resumen o palabras clave) de los estudios indexados en estos motores de búsqueda, no considerando el texto completo, cuando este estuviera disponible. Esto

ha de permitir que los estudios resultantes en cada búsqueda sean lo más relevante posible al tema en cuestión y, por ende, se reduce la muestra de estudios que han de ser analizados en las etapas posteriores. Usman et al. (2014) no especifican esta parte del proceso de búsqueda.
Por otro lado, el marco temporal del presente estudio va desde Diciembre 2013[1] hasta Diciembre 2017, incluyendo ambos. Luego de ejecutada cada búsqueda, los resultados eran almacenados en el gestor bibliográfico Zotero[2]. La eliminación de los duplicados se realizó de la siguiente forma: se organizaron las referencias en orden alfabético por motor de búsqueda y título. Se eliminaban los repetidos a partir de la segunda ocurrencia. El resultado final de este proceso se puede observar en la Tabla 4.3.

Tabla 4.3: Resultados de búsqueda en librerías digitales

Motor de búsqueda	Resultados con duplicados	Resultados sin duplicados
ACM Digital Library	170	170
IEEE Xplore	202	171
Science Direct	38	19
SCOPUS	333	182
Web of Science	325	96
Total	**1068**	**638**

Para analizar el desempeño de las palabras claves utilizadas, la Tabla 4.4 relaciona cada término de búsqueda con la cantidad de artículos asociados y el porcentaje respecto de la muestra obtenida, después de la eliminación de los duplicados. Es importante destacar que los resultados se basan en la información que se importa automáticamente desde las librerías a Zotero. Por tanto, es posible que algunos campos considerados en el proceso de búsqueda no se encuentren en dicho gestor bibliográfico, lo cual puede explicar el hecho de que el término de búsqueda "software" no muestre un 100 %.

En esta tabla, además, se observa que algunos términos no muestran ninguna coincidencia entre los resultados, como es el caso de "Crystal Methodology", "Crystal Software Development" y "Dynamic Systems Development Method". Aparte de "Software" (92.95 %) y "Agile" (91.07 %) que conforman las palabras claves principales de la búsqueda, otros términos con el mayor número de publicaciones relacionadas son "Measur*" (de *Measurement*), "Size" y "Cost", con un porcentaje de 36.68 %, 33.23 % y 29.47 %, respectivamente.

[1] Se consideró el mes de diciembre dado que en Usman et al. (2014), el proceso de búsqueda fue realizado durante la primera semana de dicho mes, por lo que algunas referencias no fueron consideradas.

[2] zotero.org

Tabla 4.4: Desempeño de los términos de búsqueda

Términos de búsqueda	Cantidad de artículos	Porcentaje
ASD:		
Agile	581	91.07 %
Scrum	119	18.65 %
Extreme Programming	22	3.45 %
Lean Software Development	11	1.72 %
Adaptive Software Development	2	0.31 %
Feature Driven Development	2	0.31 %
Crystal Methodology	-	-
Crystal Software Development	-	-
Dynamic Systems Development Method	-	-
Estimación:		
Measur*	234	36.68 %
Estimat*	166	26.02 %
Predict*	120	18.81 %
Calculat*	112	17.55 %
Calculat	33	5.17 %
Forecast*	19	2.98 %
Esfuerzo/Tamaño:		
Size	212	33.23 %
Cost	188	29.47 %
Effort	168	26.33 %
Resource	165	25.86 %
Metric	139	21.79 %
Velocity	23	3.61 %
Sizing	10	1.57 %
User Story	10	1.57 %
Dominio/Área:		
Software	593	92.95 %

4.2.3. Criterios de inclusión y exclusión

Se han de incluir aquellas publicaciones que cumplan con cada una las siguientes características: (a) relacionadas con técnicas o métodos de estimación de esfuerzo, (b) basadas en alguna de las metodologías de Agile Software Development, (c) escritas en inglés, (d) publicadas en conferencias o revistas científicas, y (e) con evidencia empírica de trabajo realizado.

En caso de no cumplir con alguna de estas características, dicho estudio será descartado/excluido. Asimismo, se habría de excluir aquellos estudios que estuvieran

relacionados únicamente con la fase de mantenimiento de software o enfocados únicamente a la medición del rendimiento (por ejemplo, velocidad del equipo de desarrollo). Estos criterios de inclusión/exclusión serán considerados en las fases 1 y 2 del proceso de selección de estudios primarios que se describe en el apartado 4.3.

4.2.4. Criterios de calidad

Usman et al. (2014) definen un *checklist* de calidad para este propósito, siguiendo el modelo propuesto por Kitchenham y Charters (2007). Todos estos criterios son valorados en base a la escala predefinida (Y = 1, N = 0, P = 0.5), donde:

- *Y* indica que el artículo **cumple** con el criterio de calidad en cuestión.
- *N* indica que el artículo **no cumple** con el criterio de calidad en cuestión.
- *P* indica que el artículo cumple **parcialmente** con el criterio de calidad en cuestión.

La puntuación máxima es 13, que se corresponde con el total de criterios de calidad. Serán excluidos aquellos estudios cuya valoración final sea inferior a **3.25** (utilizando la misma regla del primer cuartil empleada en el estudio anterior). Los criterios considerados para la evaluación de calidad de los artículos pueden observarse en la Tabla 4.6 del apartado 4.3.3.

4.3. Proceso de selección de estudios primarios

En la Figura 4.1 puede observarse la distribución de los estudios encontrados por año y tipo de artículo. Durante el período en consideración se mantiene cierto balance y leve tendencia creciente respecto al total de artículos publicados cada año e indexados en los motores de búsqueda aquí especificados. La menor cantidad corresponde al 2015 (138) y la mayor al 2017 (172). En relación al 2013, sólo se han tomado en cuenta los estudios publicados en el mes de diciembre, por ello solamente se han obtenido 4 estudios para dicho año.

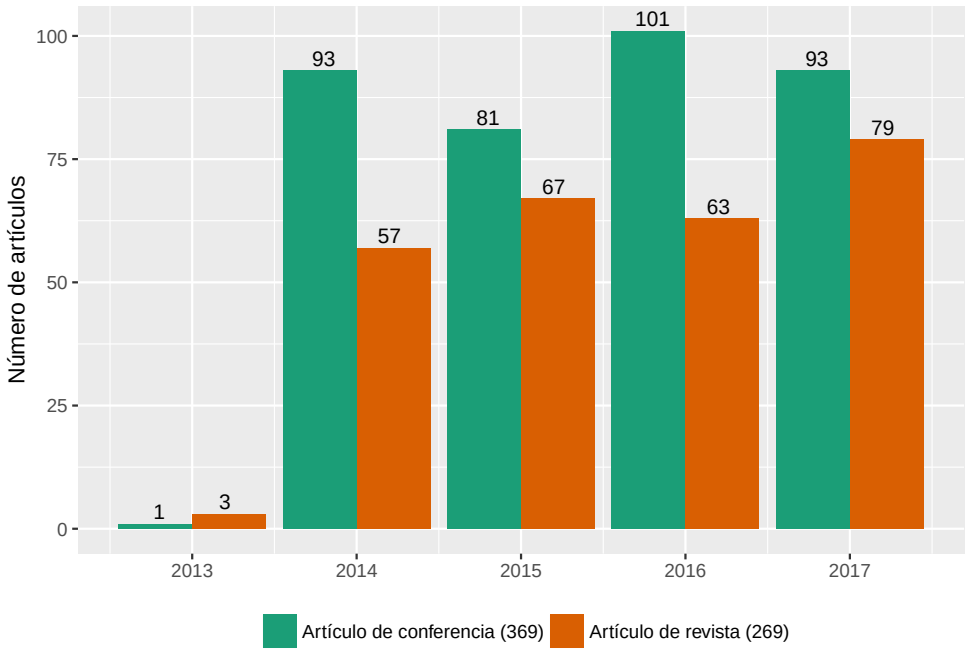

Figura 4.1: Número de artículos por año y tipo

Este conjunto de 638 estudios será sometido a las fases que se describen a continuación, por parte de un equipo compuesto por tres investigadores. El objetivo de este ejercicio es obtener un subconjunto depurado de estudios primarios y poder realizar el análisis de los mismos según las preguntas de investigación consideradas.

4.3.1. Fase 1: Filtrado por título y resumen

En esta etapa se han analizado las referencias obtenidas desde los motores de búsqueda, una vez eliminados los resultados duplicados. Los criterios de inclusión/exclusión fueron aplicados únicamente sobre el título y el resumen (*abstract*) de los artículos. Además, se han considerado las palabras clave como el tercer elemento de soporte para la toma de decisión respecto a si un artículo habría de ser seleccionado o no.

Un total de 89 estudios fueron seleccionados en esta etapa, lo que representa el 13.95 % de los 638 artículos considerados en esta primera etapa de eliminación.

4.3.2. Fase 2: Filtrado por texto completo

Esta fase consiste en la revisión del contenido de los artículos resultantes de la Fase 1, confirmando que ciertamente se cumplen los criterios de inclusión/exclusión definidos. Para aquellos artículos que no fueron encontrados en formato digital en la web se procedió a la solicitud directa a los autores vía correo electrónico y/o al uso del servicio

de prestamos inter-bibliotecarios de la universidad. La Tabla 4.5 resume los resultados de la revisión realizada en esta fase.

Tabla 4.5: Artículos eliminados en la Fase 2.

Motivo	Número de artículos
Total Fase 1	**89**
No están basados en estimación de esfuerzo o tamaño	13
No están basados en metodologías ASD	5
No están escritos en inglés	-
No han sido publicados en una conferencia o revista científica	-
No presentan evidencia empírica del estudio realizado	28
No se dispone del texto completo	1
Subtotal de estudios eliminados	**47**
Total restante	**42**

Un total de 42 estudios han sido seleccionados, lo que representa el 47.19 % de los artículos que habían superado la primera fase de filtrado. Entre estos, la mayor parte de los estudios eliminados (28 artículos) se debe a la falta de evidencia empírica o demostración práctica de algún modelo de estimación de esfuerzo, cinco de los cuales corresponden a revisiones sistemáticas de literatura (Bilgaiyan, Mishra, y Das, 2016; Bilgaiyan, Sagnika, Mishra, y Das, 2017; Britto, Usman, y Mendes, 2014; Schweighofer, Kline, Pavlič, y Heričko, 2016; Usman et al., 2014). Por otro lado, solo un artículo no pudo ser recuperado en formato digital para su revisión, por lo que ha sido igualmente descartado.

4.3.3. Fase 3: Revisión de criterios de calidad

La Tabla 4.6 sumariza la valoración asignada por criterio de calidad para el conjunto de estudios que fueron analizados en esta fase del proceso de selección.

Tabla 4.6: Lista de criterios de revisión de la calidad de los artículos.

		Valoración de los artículos		
#	Criterios de calidad	Y	N	P
1	Objetivos claramente especificados	38	0	4
2	Estudio diseñado para alcanzar los objetivos	34	1	7
3	Las técnicas de estimación utilizadas están claramente descritas y su selección justificada	33	2	7
4	Las variables consideradas por el estudio son medidas apropiadamente	32	0	10
5	Métodos de recopilación de datos descritos adecuadamente	36	1	5

6	Datos recolectados descritos adecuadamente	33	2	7
7	El propósito del análisis de los datos es claro	32	2	8
8	Las técnicas estadísticas utilizadas para analizar los datos se encuentran descritas y su uso justificado	30	7	5
9	Resultados negativos presentados (si los hay)	21	12	9
10	Se incluye la discusión sobre cualquier problema de validez/fiabilidad de los resultados	25	14	3
11	Todas las preguntas de investigación son respondidas adecuadamente	22	3	17
12	Claridad de conexión entre los datos, la interpretación y las conclusiones	31	0	11
13	Los resultados están basados en múltiples proyectos	23	17	2
	Promedio	**30**	**4.69**	**7.31**

De la tabla anterior se destaca que la gran mayoría de artículos han obtenido una valoración de *SI* cumple (Y) para todos los aspectos de calidad considerados, con un promedio de 30 del total de 42 estudios resultantes de la etapa anterior.

El contraste más significativo se observa en los criterios 10 y 13, donde se muestra mayor número de artículos que *NO* cumplen (N) con los criterios de calidad, con valores de 14 y 17 respectivamente. En algunos casos se observa que los autores no mencionan los problemas de validez relacionados a los métodos presentados (criterio 10). Esto resulta importante a la hora de analizar la fiabilidad de los resultados, así como también ayuda a interpretar en qué medida tales resultados pueden verse sesgados por el punto de vista subjetivo del investigador (Runeson y Höst, 2009). En cuanto al uso de uno o varios proyectos en la implementación y prueba de los modelos propuestos (criterio 13), esto se ha de analizar más adelante en el apartado 4.4.2.3 como parte de la respuesta a la RQ3. Por último, el criterio 11 muestra la mayor cantidad de estudios con valoración parcial (P), indicando que 17 artículos no responden adecuadamente a todas las preguntas de investigación formuladas en este estudio.

4.3.4. Fase 4: *Snowballing*

Esta etapa consiste en la utilización de las referencias bibliográficas de los artículos para identificar estudios adicionales relacionados con el objeto de búsqueda (Wohlin, 2014). En el presente trabajo de investigación, luego de la revisión de calidad, se procedió a realizar el *snowballing* de los estudios resultantes. Este proceso se realiza de forma recursiva, esto es, que las referencias identificadas son sometidas nuevamente a las fases 1, 2 y 3 del proceso de selección y, a su vez, se realiza nuevamente el *snowballing* de los estudios restantes hasta que ninguna referencia nueva fuese identificada.

Un total de 4 estudios fueron recuperados luego de la realización de este proceso, lo que resulta finalmente en 46 estudios primarios.

4.4. Análisis de los estudios primarios

La Figura 4.2 resume el proceso seguido para la selección de los estudios primarios y los resultados obtenidos luego de cada fase, según se describe en el apartado anterior.

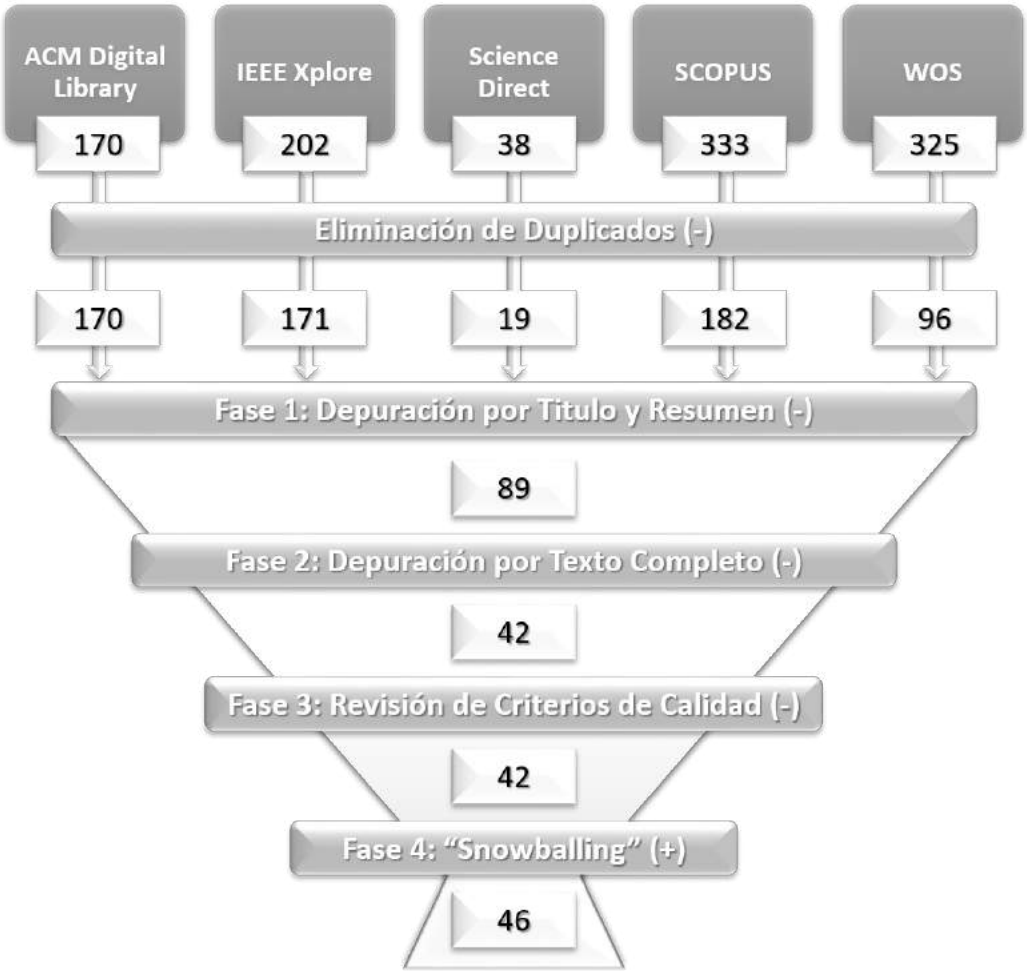

Figura 4.2: Proceso de selección de estudios primarios.

A continuación se presenta el análisis de los estudios primarios en dos secciones. Primero se observan las características bibliométricas de este subconjunto de estudios, con un enfoque en el tipo de artículo, la distribución geográfica de las publicaciones y las relaciones entre los autores. En segundo lugar se abordan las preguntas de investigación planteadas en el apartado 4.2.1.

4.4.1. Análisis bibliométrico

La Figura 4.3 muestra la distribución de los estudios primarios según el tipo de artículo (de revista o conferencia) y año de publicación. Del total de 46 estudios, 30 pertenecen a publicaciones en conferencias y 16 a publicaciones en revistas científicas.

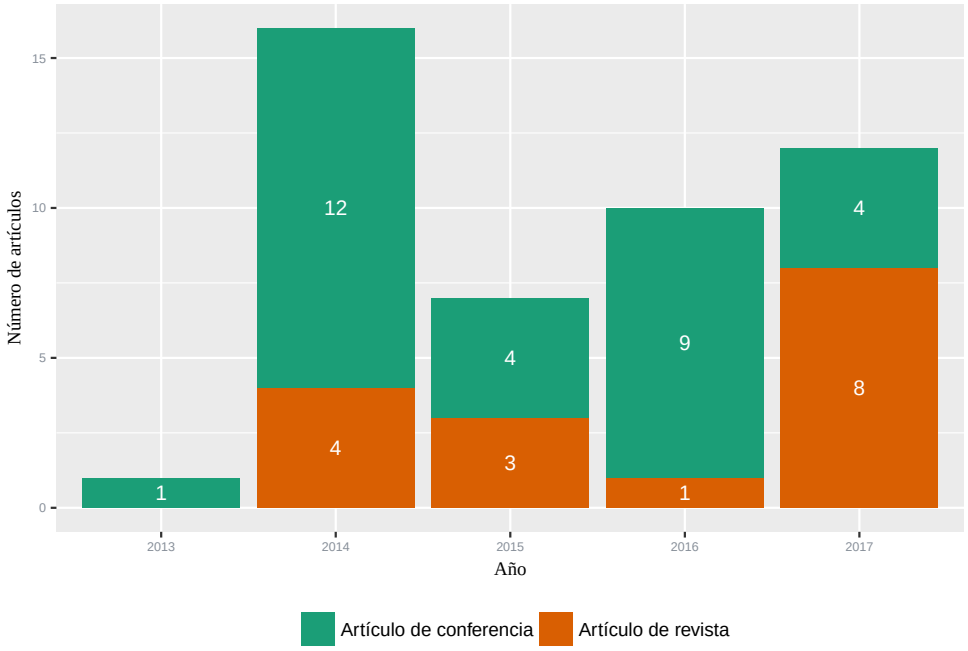

Figura 4.3: Número de estudios primarios por año y tipo

Las conferencias y revistas correspondientes a los estudios primarios se muestran en la Tabla 4.7. Un total de 25 conferencias y 15 revistas diferentes han sido identificadas en estos estudios.

Tabla 4.7: Conferencias o revistas científicas relacionadas a los estudios primarios

Conferencia/Revista	Estudios primarios
Conferencias:	
International Workshop on Software Measurement (IWSM) - MENSURA	S14 (2016), S20 (2017), S39 (2014)
EUROMICRO - Conference on Software Engineering and Advanced Applications (SEAA)	S17 (2014), S28 (2014), S43 (2016)
International Conference on Computing for Sustainable Global Development (INDIACom)	S7 (2014), S33 (2015)

International Workshop on Emerging Trends in Software Metrics	S6 (2014)
Annual International Software Testing Conference	S8 (2013)
Asia-Pacific Software Engineering Conference (APSEC)	S9 (2016)
International Workshop on Empirical Software Engineering in Practice (IWESEP)	S10 (2017)
Workshop on Information Technology and Systems	S11 (2014)
International Conference on Software Engineering Advances (ICSEA)	S16 (2014)
International Conference on Optimization, Reliabilty, and Information Technology (ICROIT)	S18 (2014)
Computer Society Signature Conference on Computers, Software and Applications (COMPSAC)	S19 (2016)
International Conference on Contemporary Computing (IC3)	S22 (2016)
International Conference on Predictive Models and Data Analytics in Software Engineering	S25 (2016)
International Conference on Information Systems and Computer Networks (ISCON)	S27 (2014)
Agile Processes in Software Engineering and Extreme Programming	S29 (2015)
International Conference on Advances in Management Engineering and Information Technology	S32 (2015)
International Conference on Evaluation and Assessment in Software Engineering	S35 (2017)
International Conference on Industrial Engineering and Operations Management (IEOM)	S36 (2015)
Symposium on Information and Communication Technology	S37 (2016)
International Conference on Information and Computer Technologies (ICICT)	S38 (2014)
International Conference on Information & Communication Technology and Systems (ICTS)	S41 (2017)
International Conference on Software Engineering and Knowledge Engineering (SEKE)	S42 (2014)
International Conference on Software and Systems Processes	S44 (2016)
International Conference on Reliability, Infocom Technologies and Optimization (ICRITO)	S45 (2016)
Agile Conference	S46 (2014)
Revistas Científicas:	
International Journal of Advanced Computer Science and Applications	S3 (2017), S30 (2017)
International Journal of Software Engineering and its Applications	S1 (2014)
IEEE Transactions on Software Engineering	S2 (2016)
International Journal of Applied Engineering Research	S4 (2015)
The International Journal of Engineering and Science (IJES)	S5 (2014)
International Journal of Process Management and Benchmarking	S12 (2017)
International Journal of Engineering and Technology	S13 (2014)

Journal of Systems and Software	S15 (2017)
Journal of Software: Evolution and Process	S21 (2017)
Innovations in Systems and Software Engineering	S23 (2017)
Procedia Computer Science	S24 (2015)
Information and Software Technology	S26 (2015)
Frontiers in Artificial Intelligence and Applications	S31 (2014)
IEEE Access	S34 (2017)
Journal of Information Science and Engineering	S40 (2017)

Debido a que este conjunto de estudios se encuentra disperso en un número significativo de conferencias y revistas científicas, solamente destacan aquellas que cuenta con más de un artículo. Por ejemplo, IWSM-MENSURA (3), EUROMICRO - SEAA (3) e INDIACom (2) son las conferencias más destacadas dentro de este conjunto, de la misma forma que *International Journal of Advanced Computer Science and Applications* (2) entre las revistas científicas.

La Figura 4.4 muestra una concentración de las publicaciones en relación a modelos de estimación de esfuerzo durante el periodo 2014-2017, distribuida principalmente entre un conjunto de países asiáticos y europeos. Para este caso se ha utilizado como lugar de referencia la ubicación del centro de investigación o universidad al que pertenecen los autores principales. De un total de 21 países, resalta la presencia de la India con 15 artículos publicados. Cabe destacar que 9 de estos comparten los mismos autores, lo cual se puede observar en la Figura 4.5 al ver los artículos relacionados a Chauhan, N. y Popli, R. (6 artículos), y Rath, S. y Satapathy, S. (3 artículos). Otros países que presentan más de una publicación dentro de este conjunto son Alemania, Italia, Pakistán, Estados Unidos, Malasia y Turquía.

País	Estudios
India	S4, S7, S8, S12, S13, S18, S22, S23, S24, S27, S33, S36, S38, S42, S45
Alemania	S21, S28, S43, S44
Italia	S16, S25, S29, S35
Pakistán	S30, S34, S40
Estados Unidos	S11, S19
Malasia	S9, S37
Turquía	S17, S20
Australia	S2
Canadá	S39
Croacia	S15
Egipto	S46
Eslovenia	S32
España	S26
Indonesia	S41
Irán	S1
Kuwait	S3
Nigeria	S5
Países Bajos	S6
Polonia	S14
República Checa	S31
Tailandia	S10

Figura 4.4: Distribución geográfica de los estudios primarios. Fuente: Elaboración propia.

Otro aspecto importante es la relación entre los diferentes grupos de autores y los artículos publicados. La Figura 4.5 muestra esta distribución, donde solo se consideran aquellos autores con más de una publicación dentro de este periodo. Chauhan, N. y Popli, R., tal y como se comentó previamente, comparten 6 artículos (S7, S8, S18, S27, S33, S38), que corresponde a la mayor cantidad publicada por autor en este período. Además, se observa que cuatro de estos fueron escritos en el 2014. Seguido de estos autores se encuentran Lenarduzzi, V. y Taibi, D., con tres artículos en común (S16, S29, S35), dos de los cuales también incluyen a Lunesu, I. (S29, S35). De igual forma, Rath, S. y Satapathy, S. son autores de tres de los estudios primarios considerados (S23, S24, S42), dos de los cuales también incluyen a Panda, A. (S24, S42).

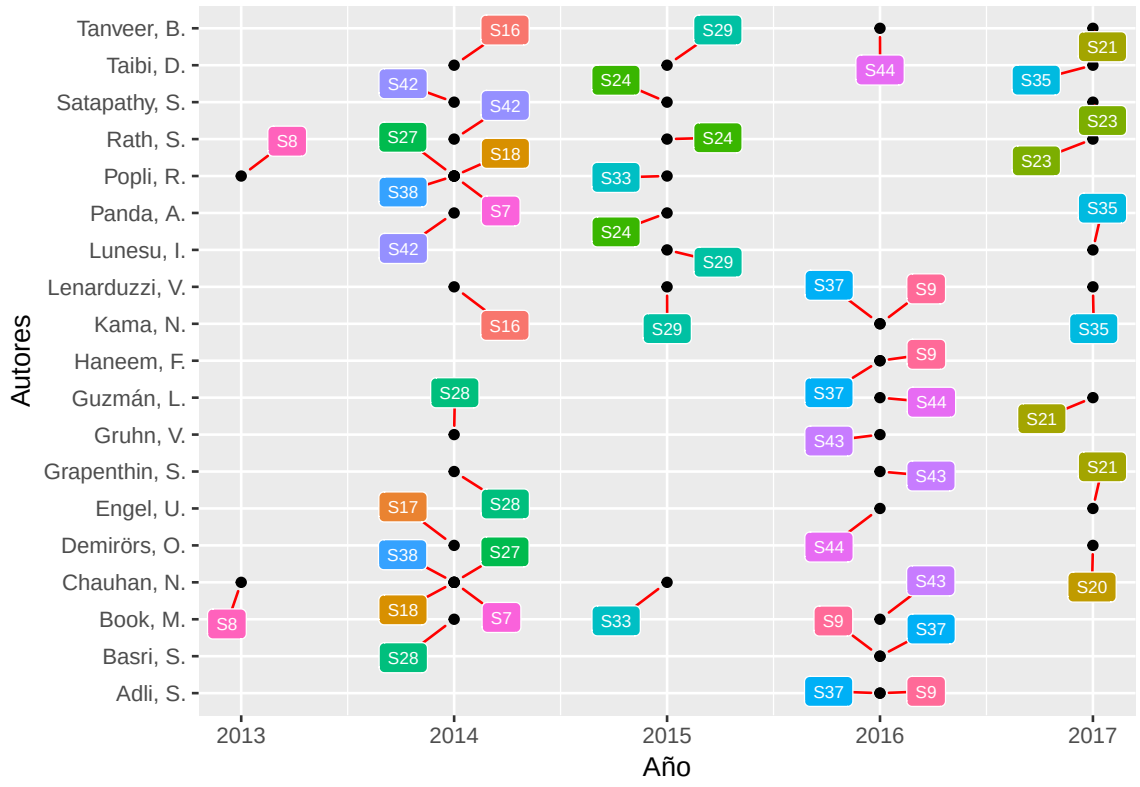

Figura 4.5: Relación entre autores

4.4.2. Respuestas a las preguntas de investigación

En esta sección se incluyen los resultados derivados de la extracción de los datos de los 46 estudios primarios. Estos datos, pues, han de ser analizados para dar respuesta a las preguntas de investigación (RQ) planteadas en este estudio.

4.4.2.1. RQ1: ¿Qué métodos han sido utilizados para la estimación del esfuerzo o tamaño en ASD?

De los 46 estudios considerados, solo uno no especifica la técnica o método utilizado para la estimación del esfuerzo o tamaño del software (S28). La Tabla 4.8 muestra los diferentes métodos utilizados en los estudios seleccionados, tanto para la estimación del esfuerzo como del tamaño del proyecto de software.

Tabla 4.8: Métodos de estimación utilizados

Método de estimación	Estudios	Cantidad	%
Basados en expertos:			
Planning Poker	S1, S17, S19, S21, S25, S26, S29, S30, S32, S34, S43 (Interaction Room Annotations), S44 (Estimation Game)	12	26.09 %
Expert Judgement	S1, S6, S10, S21, S37	5	10.87 %
Wideband Delphi	S1, S34	2	4.35 %
Basados en datos:			
Algorithmic Methods	S7, S8, S18, S27, S33 (Estimated Story Points), S12 (Estimation Process at Initial Stage), S22, S40 (Risk Aware and Quality Enriched Estimation Model)	8	17.39 %
Functional Size Measurement (FSM)	S4, S14, S17, S20 (COSMIC), S6 (IFPUG), S12 (FPA), S16, S29 (Simplified Function Points)	8	17.39 %
Machine Learning	S11 (Ensemble-based Method), S14 (HKO method), S19 (Naive Bayes, J48, Random Forest, Logistic Model Tree), S23 (Decision Tree, Stochastic Gradient Boosting, Random Forest), S25 (SVM, KNN, Decision Tree, Naive Bayes), S42 (SVR)	6	13.04 %

Neural Networks	S2 (Long-Deep Recurrent Neural Network (LD-RNN)), S24 (General Regression Neural Network (GRNN), Probabilistic Neural Network (PNN), Group Method of Data Handling (GMDH), Polynomial Neural Network and Cascade-Correlation Neural Network), S31 (Feed-forward), S41, S46 (Multilayer Perceptron)	5	10.87 %
Regression	S20 (Single and Multiple Regression), S39	2	4.35 %
Swarm Intelligence	S5 (Particle Swarm Optimization), S13 (Harmony Search Algorithm)	2	4.35 %
Bayesian Network	S15	1	2.17 %
COCOMO II	S37	1	2.17 %
Fuzzy Logic	S3	1	2.17 %
Monte Carlo	S31	1	2.17 %
PCA Model	S36	1	2.17 %
Statistical Combination	S32	1	2.17 %
Híbridos:			
Use Case Point (UCP)	S14 (Modification), S45, S46 (UCP + Quick Function Point)	3	6.52 %
Change Effort Prediction	S9, S37	2	4.35 %
Experience Factory	S35	1	2.17 %
Ontology Model	S34	1	2.17 %
Prioritization of Stories	S38	1	2.17 %
No mencionado	S28	1	2.17 %

Utilizando la clasificación de S. K. Sehra et al. (2017) y Wen et al. (2012), en combinación con la representación gráfica de Trendowicz y Jeffery (2014) (ver apartado 3.1.1.1, se han identificado y categorizado los métodos de estimación empleados en los estudios primarios. Dentro de aquellos basados en expertos (*expert-based*) se encuentran *Planning Poker* (12), juicio de expertos o *expert judgement* (5), y *Wideband Delphi* (2). *Planning Poker* es el más utilizado, confirmando que los proyectos basados en el uso de metodologías de desarrollo ágil suelen utilizar técnicas que facilitan la colaboración

y consenso para la estimación del esfuerzo requerido y/o del tamaño del software a desarrollar. Se observa, además, que *Planning Poker* suele utilizarse en combinación con otros métodos como *Expert Judgement* (S1, S21), *Wideband Delphi* (S1, S34), *Machine Learning* (S19, S25), COSMIC (S17) y FPA (S29), principalmente para realizar comparaciones en relación a la precisión de los mismos.

Entre los métodos de estimación basados en el cálculo del tamaño funcional del software, se han identificado 8 estudios que presentan el uso de métricas como COSMIC, IFPUG y SiFP. Estas son evaluadas más adelante al presentar las métricas de tamaño utilizadas (RQ2), pero también aparecen aquí representadas dado que, tal y como destaca Usman et al. (2014), la estimación del esfuerzo se deriva habitualmente del tamaño del relativo del proyecto.

Por igual, una serie de métodos algorítmicos son presentados en 8 estudios, según se observa en la tabla anterior. Cabe destacar que este subconjunto de estudios comparte particularmente algunas características en común, entre estas:

a) Han sido propuestos, en su mayoría, por los mismos autores (S7, S8, S18, S27, S33) o hacen referencia al trabajo de estos (S12, S22, S40).
b) Utilizan una unidad de esfuerzo común, denominada ESP o *Estimated Story Points* (ESP), a excepción del S22 que no lo menciona explícitamente.
c) Definen un proceso que consiste de una serie de pasos o algoritmo que permite determinar el esfuerzo del proyecto.

Otras técnicas que hacen uso intensivo de datos son *Machine Learning* (S11, S14, S19, S23, S25, S42), *Neural Networks* (S2, S24, S31, S41, S46), técnicas de regresión (S20, S39) e inteligencia artificial (S5, S13). Entre estos destacan aquellos estudios que utilizan más de una técnica o modelo, realizando así la evaluación comparativa de los resultados de los mismos. Por ejemplo:

- S19 y 25 investigan el uso de algoritmos de clasificación (Amancio et al., 2014) para determinar un modelo de generación de valores de estimación. El primero está basado en datos obtenidos de historias de usuario, mientras que el segundo se enfoca en la estimación del esfuerzo requerido para resolver problemas de funcionamiento del sistema (*issues*). *Support Vector Machine* (SVM), *K-Nearest Neighbor* (KNN), *Naive Bayes*, *Decision Tree* y *Random Forest* se encuentran entre las técnicas más conocidas y utilizadas en minería de datos (Witten, Frank, Hall, y Pal, 2016) y *Supervised Machine Learning* (Kotsiantis, 2007).
- S23 y 24, publicados por los mismos autores, utilizan diferentes técnicas para la evaluación cualitativa de un modelo de estimación basado en *story points*. El primero se enfoca en el uso de ML, mientras que el segundo evalúa una serie de modelos basados en Redes Neuronales (*Neural Networks*).

La Figura 4.6 relaciona los resultados obtenidos para los dos estudios, considerado únicamente los principales métodos o técnicas de estimación identificados en ambos casos. *Planning Poker* duplica exactamente la cantidad resultante en el estudio anterior para esta categoría, mientras que *Expert Judgement* y *UCP Method* disminuyen en relevancia en las publicaciones de los últimos años respecto a modelos de estimación

de esfuerzo. También ha aumentado relativamente el uso de métodos algorítmicos, *Neural Networks* y COSMIC. Es importante resaltar que Usman et al. (2014) no indica haber identificado ningún estudio basado en ML, por lo que esta categoría no se ve aquí representada.

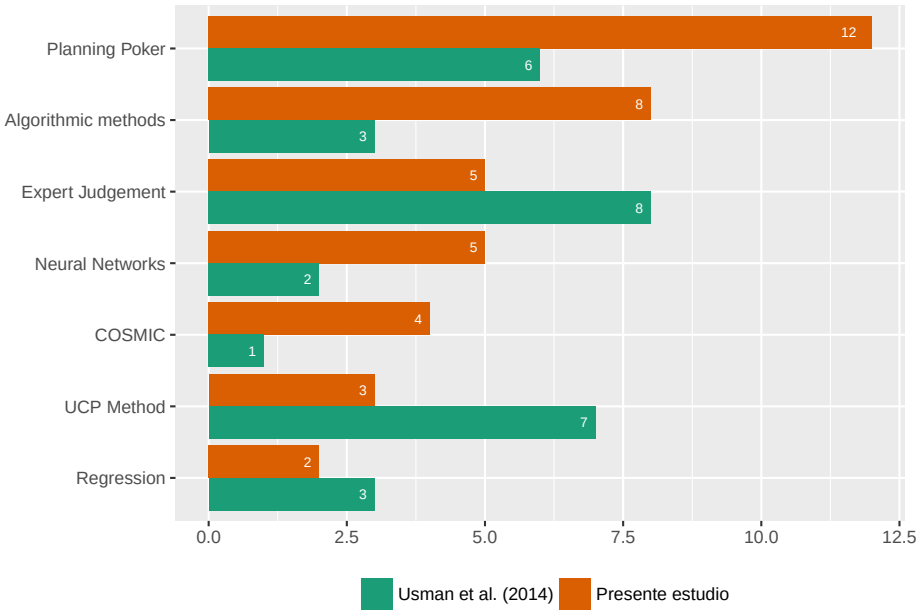

Figura 4.6: Técnicas de estimación: comparación con resultados de Usman et al. (2014).

RQ1a: ¿Qué métricas han sido utilizadas para determinar la precisión de tales métodos?

En relación al cálculo de la precisión de los modelos utilizados, la Tabla 4.9 resume los resultados encontrados. Las métricas que utilizan la magnitud de error relativo (MRE, MMRE, MdMRE) son las más utilizadas, con un total de 17 estudios que indican haber implementado algunas de estas métricas. MdMRE (*Median MRE*) se emplea específicamente en S16, S23 y S29.

PRED(x) es usado en conjunto con MRE en 8 de los estudios primarios. Por otro lado, S4 presenta tres métricas precisión basadas en modelos de regresión múltiple, las cuales son *Constrained Minimum Sum of Squared Error* (CMSE), *Constrained Minimum Sum of Absolute Error* (CMAE) y *Constrained Minimum Sum of Relative Error* (CMRE). Además, S4 compara los resultados con los siguientes modelos de regresión:

- *Stepwise Subset Selection* (Miller, 2002): utilizado para predecir los parámetros que mejor se ajustan a los valores de respuesta.
- *Ridge* (Hoerl y Kennard, 1970): utilizado para el análisis de datos que presentan multicolinealidad (fuerte correlación entre las variables independientes del

modelo), reduciendo sus efectos adversos.
- *Lasso* (Tibshirani, 1996): utilizado para reducir la variabilidad de las estimaciones mediante la reducción de los coeficientes.

Tabla 4.9: Métricas de precisión utilizadas

Métrica de precisión	Estudios	Cantidad	%
MRE/MMRE/MdMRE	S1, S4, S5, S9, S16, S17, S19, S20, S23, S24, S25, S29, S34, S37, S41, S42, S46	17	36.96 %
PRED(x)	S4, S9, S17, S20, S23, S24, S34, S42	8	17.39 %
Mean(BRE)	S14, S32	2	4.35 %
Median(BRE)	S32, S43	2	4.35 %
MAE	S2, S11	2	4.35 %
R2	S24, S39	2	4.35 %
Otras	S2 (Standardized Accuracy, SA), S4 (Subset Selection, Ridge, Lasso, CMSE, CMAE, CMRE), S11 (MBE, RMSE), S24 (MSE), S43 (Median(Brebias)	5	10.87 %
No mencionado	S6, S7, S8, S10, S12, S13, S18, S21, S22, S26, S27, S28, S30, S31, S33, S38, S44, S45	18	39.13 %

RQ1b: ¿Cuál es nivel de precisión de tales métodos?

En la Tabla 4.10 se presentan los valores de precisión alcanzados por cada una de las métricas utilizadas, según las técnicas de estimación empleadas. Se observa que ninguno de los estudios clasificados en la categoría de métodos algorítmicos describe la métrica de precisión empleada para la validación del modelo presentado. Por otro lado, la mayoría de estudios presenta valores aceptables de MMRE y PRED, lo cual sería menor o igual que 25 % para MMRE, y mayor o igual que 75 % para PRED (Samuel Daniel Conte, Dunsmore, y Shen, 1986; Port y Korte, 2008).

Tabla 4.10: Nivel de precisión de las técnicas de estimación utilizadas.

Técnica de estimación	Nivel de precisión
Basados en expertos:	
Planning Poker	MRE: 7.1 % (S1) MMRE: 0 % - 204.4 % (S17, S19, S25, S29, S34) MdMRE: 24 % (S29) PRED(x): 83 % - 88.89 % (S17, S34) Mean(BRE): 0.37 - 0.51 (S32) Median(BRE): 0.40 - 1.792 (S32, S43) Median(BREbias): from -2.128 to -0.250 (S43)
Expert Judgement	MMRE: 12.4 % - 29.0 % (S1, S37)
Wideband Delphi	MRE: 7.6 % (S1) MMRE: 15.3 % (S34) PRED(x): 75 % (S34)
Basados en datos:	
FSM	MMRE: 5.8 % - 27 % (S4, S16, S17, S20, S29) MdMRE: 9.35 % - 24 % (S16, S29) PRED(x): 8.06 % - 100 % (S4, S17, S20) Mean(BRE): 0.32 - 0.70 (S14) Subset Selection: 0.29 (S4) Ridge: 0.28 (S4) Lasso: 0.29 (S4) CMSE: 0.28 (S4) CMAE: 0.27 (S4) CMRE: 0.23 (S4)
Machine Learning	MAE: 7.61 - 8.002 (S11) MBE: 1.07 - 2.89 (S11) RMSE: 11.46 - 13.67 (S11) MMRE: 0 % - 204.4 % (S19, S23, S25, S42) MdMRE: 11 % - 29 % (S23) PRED(x): 38.1 % - 95.90 % (S23, S42)
Neural Networks	MAE: 2.09 (S2) SA: 52.66 % (S2) MSE: 0.0244 - 0.0317 (S24) R2: 0.6259 - 0.9303 (S24) MMRE: 35.81 % - 157.76 % (S24) PRED(x): 85.92 % - 94.76 % (S24) Time MRE: 4 % - 84 % (S41) MRE: 12 % - 34 % (S46)
Regression	MMRE: 3 % - 22 % (S20) PRED(x): 78 % - 100 % (S20) R2: 0.9723 (S39)

Swarm Intelligence	Effort MMRE: 19.88 % (S5) Time MMRE: 23.10 % (S5)
Híbridos:	
UCP Method	Mean(BRE): 0.325 - 0.700 (S14) MRE: 12 % - 34 % (S46)
Change Effort Prediction Model	MMRE: 22 % (S9, S37) PRED(x): 73.13 % (S9)

4.4.2.2. RQ2: ¿Qué predictores de esfuerzo han sido utilizados en estudios sobre estimación de esfuerzo en ASD?

Esta pregunta de investigación está relacionada con los predictores de esfuerzo. Tal y como se presenta en la sección 3.1.2 del Capítulo 3, los principales predictores de esfuerzo son el tamaño y los factores de coste. A continuación se evalúa cada uno de estos en relación a los 46 estudios primarios.

Métricas de tamaño

La Tabla 4.11 muestra que los puntos de historia (*story points*) son las métricas más utilizadas para determinar el tamaño del software. El 67.39 % de los estudios indica haber utilizado *story points*, en algunos casos, en combinación con otras métricas como NESMA (S6), COSMIC (S17, S20), LOC (S12, S20) y UCP (S46).

Tabla 4.11: Métricas de tamaño utilizadas

Métricas de tamaño	Estudios	Cantidad	%
Story Points	S2, S3, S6, S7, S8, S10, S11, S12, S13, S17, S18, S19, S20, S21, S22, S23, S24, S25, S26, S27, S28, S30, S32, S33, S35, S38, S40, S42, S43, S44, S46	31	67.39 %
Function Points:			
COSMIC	S14, S17, S20, S36, S39	5	10.87 %
SiFP	S16, S29	2	4.35 %
FPA*	S12	1	2.17 %
IFPUG	S29	1	2.17 %
NESMA	S6	1	2.17 %
Quick Function Points	S46	1	2.17 %
LOC	S9, S12, S20, S31	4	8.70 %
Use Case Point (UCP)	S45, S46	2	4.35 %
eXtreme Unit (XU)	S4	1	2.17 %

No mencionado	S1, S5, S15, S34, S37, S41	6	13.04 %

Los puntos de función (*function points*) ocupan el segundo lugar con un total de 9 estudios (S6, S14, S16, S17, S20, S29, S36, S39, S46), considerando que S29 se repite en las categorías de SiFP e IFPUG. Entre este grupo de métricas se encuentran COSMIC, IFPUG y NESMA, los cuales han sido mencionados en el apartado 3.1.2.1 entre los principales estándares de medición de tamaño funcional. S12 utiliza *Function Points Analysis* para la estimación del tamaño, aunque no se especifica el uso de ninguna de las métricas estándares. Por otro lado, se introduce el uso de las siguientes métricas:

a) SiFP, *Simplified Function Points*, (S16, S29), como una versión simplificada de IFPUG (Lavazza y Meli, 2014), que cuenta además con su propia asociación independiente para promover su uso[3]; y
b) *Quick Function Points* (S46), utilizado para medir el tamaño del software en las etapas iniciales del proceso de desarrollo (Santillo, Conte, y Meli, 2005), y que también está basado en IFPUG.

S4 presenta el uso de *eXtreme software size Units* (XU) para determinar el tamaño de proyectos que utilizan *Xtreme Programming* como metodología de desarrollo. Esta métrica es definida como una versión modificada de *COSMIC Full Function Points*, adaptada para este tipo de proyectos y más fácil de entender. El valor de XU de una historia de usuario dependerá, entre otros elementos, del número de campos en la interfaz y tablas creadas por dicha historia, así como también del número de variables que el programa deberá manejar, según describen los autores de este estudio.

La Figura 4.7 muestra la comparación con los resultados obtenidos por Usman et al. (2014).

[3]www.sifpa.org/en/

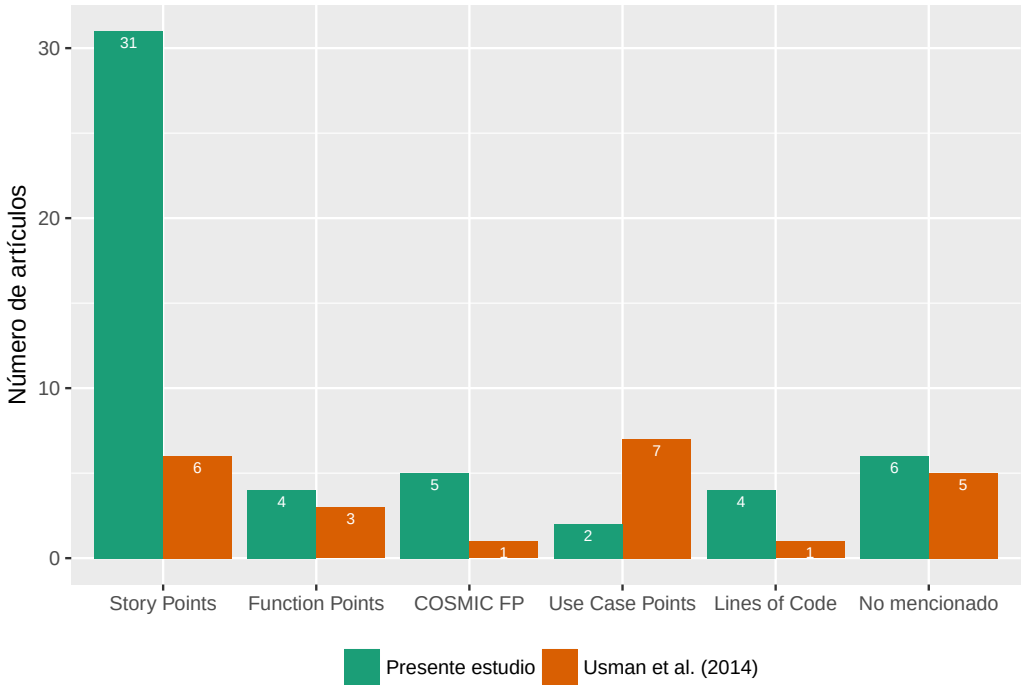

Figura 4.7: Métricas de tamaño: comparación con resultados de Usman et al. (2014).

Story Points supera hasta cinco veces los resultados del estudio anterior con 31 publicaciones que utilizan está métrica, mientras que solo 2 estudios indican el uso de *Use Case Points*, que anteriormente ocupaba el primer lugar con 7 estudios resultantes para esta categoría en (Usman et al., 2014). Se observa un aumento en el uso de COSMIC FP y LOC, que representaban los menos utilizados con tan solo una publicación para cada uno. Se destaca, además, que en esta gráfica se ha decidido presentar COSMIC FP separado del resto de métricas basadas en puntos de función para poder establecer la comparativa con los resultados del estudio anterior. Por otro lado, los 6 estudios que no indican el uso de ninguna métrica para medir el tamaño del software representan el 13.04 % en este conjunto, en contraste con el 20 % para el estudio de referencia.

La Figura 4.8 muestra la relación entre las principales métricas de tamaño y el año de publicación del estudio. El uso de *Story Points* en los modelos evaluados se presenta considerablemente por encima del resto de métricas en todos los años, con una mayor cantidad de estudios relacionados en 2014 y menor cantidad en 2015. *Function Points* también presenta un mayor número de estudios en el 2014 (5), mientras que UCP y LOC se mantienen como las menos utilizadas para todos los años. Tanto en 2014 como en 2016 se han investigado todas estas métricas, en relación a estimación de esfuerzo en entornos de desarrollo ágil.

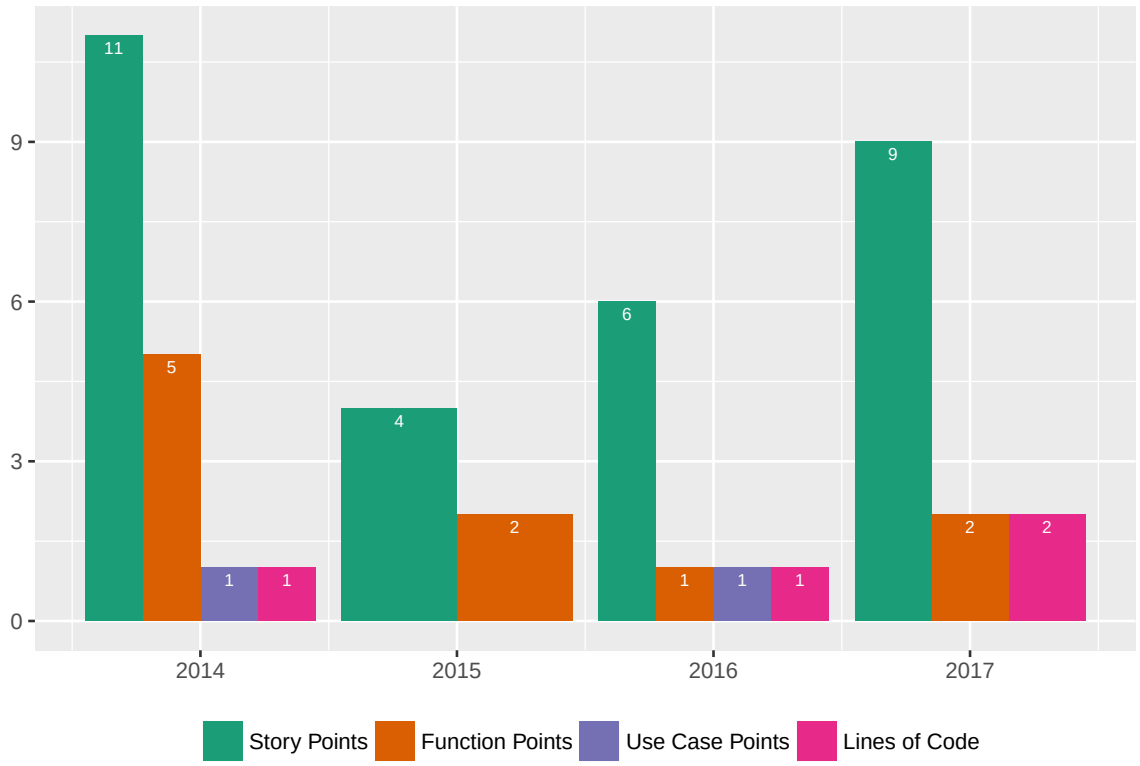

Figura 4.8: Número de métricas de tamaño utilizadas por año

Otro aspecto importante en relación a las métricas de tamaño es su relación con las técnicas de estimación utilizadas. La mayoría de estudios que usan *Planning Poker* o juicio de expertos para la estimación del esfuerzo indican haber empleado *story points* como métrica de tamaño, a excepción de S1, S34 y S37 que no mencionan explícitamente la métrica de tamaño, y S29 que utiliza IFPUG y SiFP en su lugar. Todos los métodos algorítmicos utilizan *story points*.

Factores de coste

Dado que se ha identificado una amplia variedad de factores de coste, se ha decidido agrupar entre factores del proyecto, del equipo, técnicos, de historias de usuario (*user stories*), sintácticos/textuales y otros. La Tabla 4.12 muestra los resultados para grupo.

Tabla 4.12: Factores de coste utilizados

Factores de coste	Estudios	Cantidad	%
Factores del proyecto:			
Complexity	S3, S13, S15, S21, S22, S28, S36, S40, S44, S45	10	21.74 %
Risk taking	S22, S36, S40	3	6.52 %
Novelty	S15, S40	2	4.35 %
Priority	S34, S44	2	4.35 %
Quality	S15, S40	2	4.35 %
Clarity of requirements	S44	1	2.17 %
Different sites	S22	1	2.17 %
Learnt lessons	S34	1	2.17 %
Project deceleration	S5	1	2.17 %
Relevant projects	S34	1	2.17 %
Task size	S3	1	2.17 %
Factores del equipo:			
Team/Programmer's experience	S3, S11, S22, S31	4	8.70 %
Developer skills, capacity or knowledge	S4, S15, S44	3	6.52 %
Familiarity within teams/project	S22, S28	2	4.35 %
Working hours/days	S5, S15	2	4.35 %
Communication (timezones and languages)	S22	1	2.17 %
Developers availability	S31	1	2.17 %
Team size	S36	1	2.17 %
Team's technical and managerial ability	S22	1	2.17 %
Factores técnicos:			
GUI or existing systems impact	S16, S29, S44	3	6.52 %
Software type	S4, S36	2	4.35 %
Software domain	S34	1	2.17 %
Software & tools used	S4	1	2.17 %
Database used	S4	1	2.17 %
Development platform	S36	1	2.17 %
Operating systems	S36	1	2.17 %
Process maturity	S36	1	2.17 %
Programming languages	S36	1	2.17 %
Source code	S9	1	2.17 %
Factores de historias de usuario:			
Priority of the story	S11, S19	2	4.35 %
Story type	S16, S29	2	4.35 %
Number of subtasks of the story	S11	1	2.17 %

Sensitivity of the story	S40	1	2.17 %
Sprint of the story	S11	1	2.17 %
Factores sintácticos/textuales:			
Frequency of syntactic linguistic features	S14	1	2.17 %
Term frequency	S19	1	2.17 %
Textual information	S25	1	2.17 %
Words in title and description	S2	1	2.17 %
Otros:			
Popli & Chauhan's factors	S7, S8, S12, S27	4	8.70 %
COCOMO II cost drivers	S9, S37	2	4.35 %
Type/Level of implementation	S13, S44	2	4.35 %
Actor type factors	S45	1	2.17 %
Business value	S26	1	2.17 %
Change request type/priority	S9	1	2.17 %
Development status	S9	1	2.17 %
Resistance factors	S27	1	2.17 %
Use case type factors	S45	1	2.17 %
No mencionado	S1, S6, S10, S17, S18, S20, S23, S24, S30, S32, S33, S35, S38, S39, S41, S42, S43, S46	18	39.13 %

Esta tabla evidencia la ausencia de un patrón común en el uso de factores de coste específicos y generales, igual que en el estudio anterior. Esto se debe a que los factores de coste dependen principalmente de las características de los proyectos analizados a la hora de proponer un modelo de estimación de esfuerzo. S9 y S37 son los únicos que explícitamente mencionan el uso del modelo COCOMO, brevemente presentado en el apartado 3.1.2.2. En cambio, cuatro estudios (S7, S8, S12, S27) consideran un conjunto específico de factores relacionados a las personas y al proyecto (presentados en dichos estudios como *People and Project related factors*), por lo que se los ha incluido dentro la categoría de "Otros" como *Popli and Chauhan's factors*, en referencia al nombre de los autores.

Los factores relacionados al proyecto y a las personas o equipo del proyecto aparecen con mayor frecuencia que el resto. Esto concuerda con los resultados de Usman et al. (2014), donde se destaca que las habilidades y experiencia del equipo de desarrollo juegan un papel importante en el proceso de estimación. La complejidad es el factor del proyecto más utilizado, seguido de otros aspectos como la experiencia/habilidades del equipo del proyecto, el riego asociado al proyecto, el impacto en los sistemas existentes, entre otros. La mayoría de factores técnicos han sido introducidos en los estudios S4 y S36 (tipo de software/aplicación, base de datos utilizada, plataforma de desarrollo, lenguaje de programación, entre otros). Por otro lado, los factores relativos

a las historias de usuario conforman un pequeño subconjunto por sí mismos dada su vinculación específica con este elemento del proyecto.

Los factores relacionados con el contenido textual y descriptivo de los requerimientos del software a desarrollar han sido agrupados como factores sintácticos/textuales. Se observa que estos son introducidos por técnicas de estimación basadas en el uso intensivo de datos como *Machine Learning* (S14, S19, S25) y *Neural Networks* (S2). Por ejemplo, S14 presenta una adaptación del método HKO (Hussain, Kosseim, y Ormandjieva, 2013) para la clasificación de requerimientos informalmente escritos en relación a su tamaño funcional según COSMIC, mediante casos de uso y herramientas de procesamiento del lenguaje natural (NLP, *Natural Language Processing*).

A pesar de la separación en categorías diferentes, es posible encontrar intersecciones entre las mismas. Por ejemplo, la comunicación es un factor que depende tanto de las personas como del entorno en el que el proyecto es desarrollado. Asimismo, las historias de usuarios son un artefacto específico del proyecto (en particular de las metodologías ágiles), por lo que podrían ser incluidos dentro de esta categoría por igual. Por último, en aproximadamente el 40 % de los estudios no se ha podido identificar el uso de ningún factor de coste.

4.4.2.3. RQ3: ¿Cuáles son las características del dataset utilizado en estudios sobre estimación de esfuerzo/tamaño en ASD?

En esta pregunta de investigación se evalúan las características del dataset utilizado en cada estudio. Este aspecto es importante a la hora de considerar la fiabilidad y aplicabilidad de los modelos presentados por los estudios considerados en esta muestra. Básicamente, permite identificar si los modelos presentados han sido validados con uno o múltiples casos, reales o simulados, para fines académicos o aplicados en la industria, y contando con datos internos de una compañía o incluyendo múltiples organizaciones dedicadas a la implementación de proyectos de desarrollo de software. A continuación se describen estos aspectos relacionados con dominio y tipo de dataset.

Dominio del dataset

La mayoría de estudios considerados han utilizados datos de proyectos de desarrollo de software de compañías específicas, identificándose 26 estudios con datasets de dominio industrial y 5 estudios con datos relacionados con proyectos de instituciones educativas. S14 es el único que emplea un repositorio mixto, considerando un dataset compuesto por 5 proyectos implementados en la Universidad Tecnológica de Poznan y un proyecto llevado a cabo por una compañía de desarrollo de software. Por otro lado, solo dos casos indican explícitamente el uso de datos extraídos de una simulación (S31, S40), los cuales corresponden a modelos de estimación basados en el uso intensivo de datos. La Tabla 4.13 refleja todo esto.

Tabla 4.13: Dominio del dataset utilizado

Dominio del dataset	Estudios	Cantidad	%
Industrial	S1, S3, S4, S6, S9, S10, S17, S2, S20, S21, S23, S24, S25, S26, S28, S30, S32, S34, S35, S36, S37, S39, S41, S42, S44, S46	26	56.52 %
Académico	S11, S13, S15, S16, S43	5	10.87 %
Industrial/Académico	S14	1	2.17 %
Simulado	S31, S40	2	4.35 %
No mencionado	S5, S7, S8, S12, S18, S19, S22, S27, S29, S33, S38, S45	12	26.09 %

Un total de 12 estudios no mencionan el dominio del dataset utilizado, a pesar de que en todos estos existe alguna forma de validación del modelo. S19, por ejemplo, indica el uso de datos recolectados mediante *IBM Rational Team Concert*; sin embargo, no es posible identificar para qué fines están destinados los proyectos de desarrollo de software analizados. En algunos casos, el modelo es validado tan solo mediante uno o varios escenarios específicos, sin indicar el uso de datos de un repositorio (S7, S8, S12, S18, S22, 27).

La Figura 4.9 muestra la comparación obtenida con respecto a los resultados de Usman et al. (2014). La principal diferencia se produce con respecto a los estudios que no mencionan el dominio del dataset utilizado, 12 artículos en el presente estudio frente a 1 en el anterior. Por otro lado, los datasets de dominio industrial ocupan el primer lugar y solo un estudio ha sido identificado con un repositorio mixto (industrial/académico) en ambos casos.

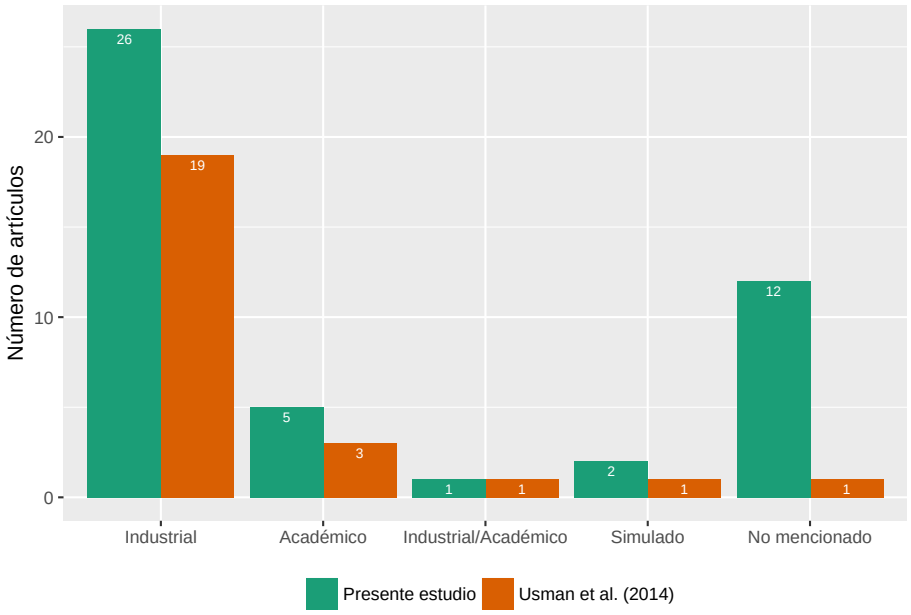

Figura 4.9: Dominio del dataset: comparación con resultados de Usman et al. (2014).

Tipo de dataset

El 50 % de los estudios presenta modelos de estimación evaluados mediante datos de proyectos internos de una compañía. Solo 9 estudios indican el uso de datasets con proyectos de diferentes organizaciones (*cross-company*), incluyendo repositorios de JIRA (S2, S10, S25) y CSBSG[4] (S9). La Tabla 4.14 muestra esta relación.

Tabla 4.14: Tipo de dataset utilizado

Tipo de dataset	Estudios	Cantidad	%
Interno	S1, S3, S4, S6, S11, S13, S14, S15, S16, S17, S20, S21, S26, S28, S30, S32, S34, S35, S39, S41, S43, S44, S46	23	50.00 %
Cross-company	S2, S9, S10, S23, S24, S25, S36, S37, S42	9	19.57 %
Simulado	S31, S40	2	4.35 %
No mencionado	S5, S7, S8, S12, S18, S19, S22, S27, S29, S33, S38, S45	12	26.09 %

S23, S24 y S42, que comparten autoría, utilizan el mismo dataset de 21 proyectos

[4] *Chinese Software Benchmarking Standard Group*

de software desarrollados por 6 compañías diferentes. Las categorías "Simulado" y "No mencionado" contienen los mismos estudios en las tablas 4.13 y 4.14. Asimismo, se observa que todos los estudios que utilizan un dataset académico son de carácter interno. Por otro lado, los resultados son consistentes con los del estudio anterior. Tan solo se destaca que en el presente estudio, el número de artículos correspondientes a repositorios *cross-company* (con datos de proyectos de diferentes compañías) es más significativo, representando el 19.57 % del total de estudios. La Figura 4.10 muestra la comparación de los resultados para cada uno.

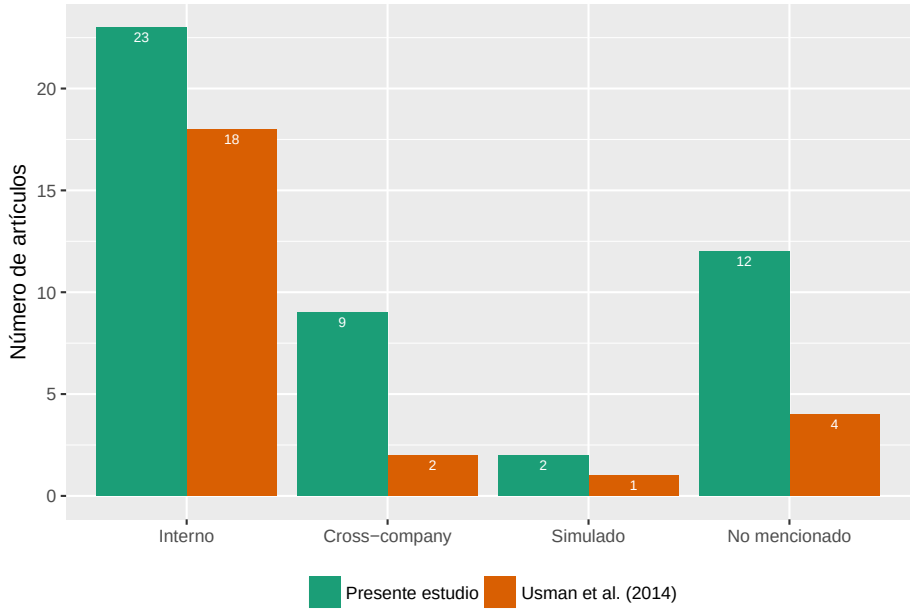

Figura 4.10: Tipo de dataset: comparación con resultados de Usman et al. (2014).

4.4.2.4. RQ4: ¿Qué metodologías ágiles han sido utilizadas en estudios sobre estimación de esfuerzo/tamaño?

Esta pregunta va relacionada a uno de los aspectos fundamentales de este estudio, ya que ubica el objeto de investigación (estimación de esfuerzo) en un contexto específico (desarrollo ágil). Esto significa que todos los modelos de estimación de esfuerzo que han sido revisados cumplen con la condición de ser implementados en entornos de ASD (ver proceso de selección de estudios primarios en el apartado 4.3). Recordemos que en el Capítulo 2 se ha realizado una introducción general de las metodologías de desarrollo ágil.

La Tabla 4.15 muestra el resumen de los resultados de la extracción de datos en relación a esta pregunta de investigación.

Tabla 4.15: Metodologías ágiles utilizadas

Metodologías ágiles	Estudios	Cantidad	%
Scrum	S1, S3, S6, S9, S11, S12, S15, S16, S17, S20, S21, S22, S23, S24, S26, S28, S29, S30, S31, S32, S33, S34, S35, S37, S39, S41, S42, S43, S44	29	63.04 %
XP	S4, S9, S25, S37, S44	5	10.87 %
TDD	S39, S44	2	4.35 %
Agile Unified Process (AUP)	S9, S37	2	4.35 %
Kanban	S39	1	2.17 %
Distributed Agile Software Development	S40	1	2.17 %
No mencionado	S2, S5, S7, S8, S10, S13, S14, S18, S19, S27, S36, S38, S45, S46	14	30.43 %

De las principales metodologías ASD presentadas previamente, solo Scrum, *Xtreme Programming* (XP) y *Test Driven Development* (TDD) son referenciadas en los estudios primarios analizados. Entre las nuevas menciones se encuentran:

a) *Agile Unified Process*, una versión simplificada de *Rational Unified Process* (RUP), que aplica técnicas ágiles como refactorización, TDD, modelado ágil, desarrollo incremental e iterativo, entre otros (Ambler, 2005);
b) Kanban, caracterizado como un modelo más orientado a lo visual y menos prescriptivo (en comparación con otras metodologías ASD), además de estar basado en el mecanismo *just-in-time* de la empresa Toyota (Brechner, 2015; Kniberg y Skarin, 2010); y
c) *Distributed Agile Software Development* (DASD), que resulta de la aplicación de los principios, técnicas y prácticas ágiles en entornos de desarrollo de software distribuido (Shrivastava y Date, 2010).

El 63.04 % de los estudios indica haber utilizado Scrum. El resto de estudios emplea Scrum en combinación con otro método ASD, exceptuando S4 y S25, cuyos proyectos han sido desarrollados con XP únicamente, y S40, donde se presenta DASD. Por otro lado, 14 estudios no indican la metodología ágil utilizada, lo que representa el 30.43 % de los estudios primarios.

En el estudio original solo se han identificado dos metodologías ágiles: Scrum (8 estudios) y XP (7 estudios). El resto de estudios no ha identificado la metodología. Además, se observa que en la SLR anterior no se han encontrado estudios en los que se emplee más de una metodología de desarrollo ágil a la vez.

RQ4a: ¿Qué actividades de desarrollo han sido objeto de estimación?

La actividad de desarrollo está relacionada con la etapa del proceso de desarrollo que ha sido considerada en el modelo de estimación esfuerzo. Teniendo en consideración las actividades comunes a todo proceso de desarrollo de software (análisis, diseño, desarrollo/implementación y prueba), la Tabla 4.16 muestra las actividades de desarrollo identificadas en los estudios primarios considerados.

Tabla 4.16: Actividad de desarrollo

Actividad de desarrollo	Estudios	Cantidad	%
Todo el proyecto	S1, S11, S12, S17, S20, S21, S22, S26, S32, S34, S37, S41	12	26.09 %
Desarrollo	S9, S31	2	4.35 %
Desarrollo y Prueba	S30, S46	2	4.35 %
No mencionado	S2, S3, S4, S5, S6, S7, S8, S10, S13, S14, S15, S16, S18, S19, S23, S24, S25, S27, S28, S29, S33, S35, S36, S38, S39, S40, S42, S43, S44, S45	30	65.22 %

A simple vista se observa que la mayoría de estudios (un 65.22 %) no menciona la actividad de desarrollo objeto de estimación. Además, ninguno de los estudios ha considerado la estimación del esfuerzo de las fases de análisis y diseño en particular, sino que se hayan integradas en un esfuerzo conjunto de todo el proyecto (26.09 % de los estudios primarios). En contraste con Usman et al. (2014) (ver Figura 4.11), no se ha encontrado ningún estudio que realice la estimación de esfuerzo de la fase de prueba únicamente, sino que se encuentra combinada con la etapa de desarrollo o el proyecto en general.

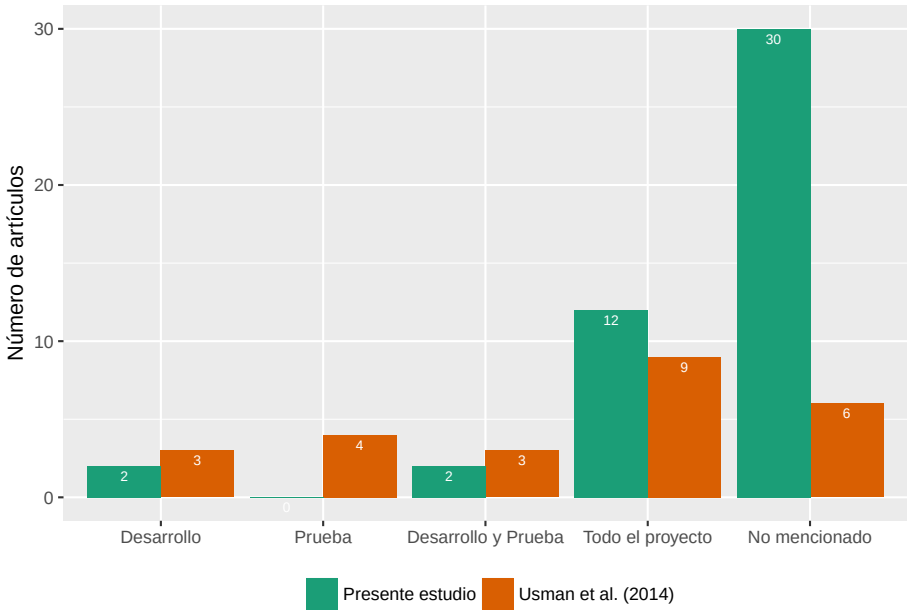

Figura 4.11: Actividad de desarrollo: comparación con resultados de Usman et al. (2014).

Teniendo en cuenta el carácter dinámico de los proyectos en ASD, es comprensible el hecho de que una gran cantidad de artículos no mencionen la actividad de desarrollo considerada en la estimación. En definitiva, los ciclos cortos en ASD normalmente incluyen todas las fases del proceso de desarrollo y, además, estas fases suelen solaparse y reajustarse dinámicamente durante la iteración. Por ello tiene sentido asumir que muchos de los autores han construido sus modelos de estimación desde un punto de vista general (todo el proyecto). Sin embargo, a no ser que haya quedado explícito en el estudio, no se han clasificado como tal.

RQ4b: ¿A qué niveles de planificación se realiza la estimación?

La estimación de esfuerzo en ASD se realiza principalmente a tres niveles de planificación: *release*, iteración y planificación diaria. En ninguno de los estudios se ha indicado la implementación del modelo de estimación de esfuerzo a nivel diario, sino por iteración o *sprint* (56.52 %) y por proyecto o *release* (32.61 %). En 8 estudios no se indica el nivel de planificación. La Tabla 4.17 muestra estos resultados.

Tabla 4.17: Nivel de planificación

Nivel de planificación	Estudios	Cantidad	%
Iteración (Sprint)	S3, S5, S7, S8, S11, S12, S13, S14, S15, S19, S20, S21, S22, S26, S28, S29, S30, S31, S32, S34, S35, S40, S41, S43, S44, S45	26	56.52 %
Proyecto (Release)	S1, S4, S6, S9, S18, S20, S23, S24, S25, S26, S27, S32, S33, S42, S46	15	32.61 %
No mencionado	S2, S10, S16, S17, S36, S37, S38, S39	8	17.39 %

S20, S26 y S32 consideran tanto la iteración como el proyecto en general. Para establecer la comparación con los resultados de Usman et al. (2014), se presentan por separado esto estudios mixtos que trabajan a la vez a nivel de *sprint* y de *release*. También destaca la cantidad de estudios que indican el *sprint* como principal nivel de planificación. Por otro lado, en el estudio anterior se ha observado que el 48 % de los casos no menciona el nivel de planificación, comparado con el 17.39 % de los estudios primarios aquí considerados. La Figura 4.12 presenta la comparación entre ambos estudios.

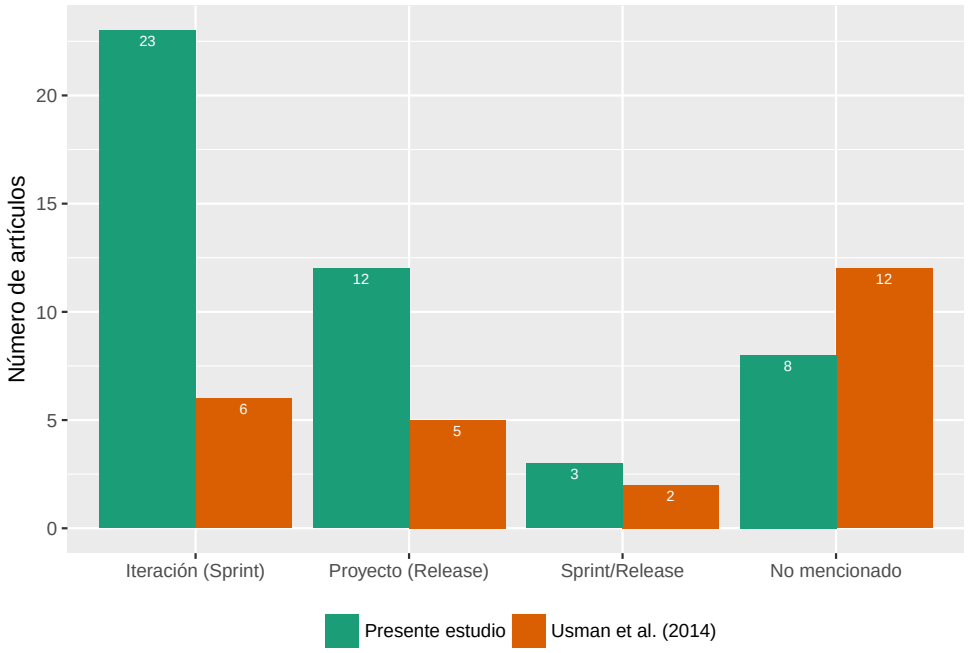

Figura 4.12: Nivel de planificación: comparación con resultados de Usman et al. (2014).

4.5. Discusión de los resultados

La Tabla 4.18 permite comparar los resultados obtenidos en la SLR anterior con los del presente estudio. El primer estudio abarca un periodo de 13 años y emplea 8 motores de búsqueda para recopilar las publicaciones relacionadas con estimación de esfuerzo en ASD. En cambio, en el presente estudio se han utilizado solo 5 motores de búsqueda, abarcando cuatro años más los estudios publicados en diciembre 2013 que no fueron incluidos en la búsqueda anterior.

Tabla 4.18: Resumen comparativo de las SLRs

	Usman et al. (2014)	Presente estudio
Periodo considerado	2001 - 2013	2013* - 2017
Número de motores de búsqueda utilizados	8	5
Resultados de búsqueda inicial (con duplicados)	787	1068
Resultados de búsqueda inicial (sin duplicados)	443	638
Número de estudios primarios	25	46

A pesar de haber considerado un rango menor de años y utilizado una menor cantidad de motores de búsqueda o librerías digitales, los resultados del presente estudio sobrepasan considerablemente los anteriores, con aproximadamente un 35 %, 44 % y 84 % más de publicaciones en la búsqueda inicial con duplicados, la búsqueda inicial sin duplicados y el subconjunto final de estudios primarios, respectivamente. Esto indica que el interés de los investigadores en relación al tema en cuestión ha aumentado significativamente en los últimos años, en coherencia con la expansión del uso de metodologías ágiles en la industria de desarrollo software.

En relación a las preguntas de investigación aquí planteadas, se observa que cada una de estas desarrolla uno de los aspectos importantes del estudio y permiten abordarlo desde cuatro puntos de vista distintos. Los resultados para cada uno de estos puntos son analizados en los párrafos siguientes.

4.5.1. Métodos de estimación de esfuerzo

En primer lugar, se ha observado que las técnicas de estimación basadas en expertos ocupan una posición importante, siendo *Planning Poker* la más utilizada. No obstante, considerando la subdivisión en tres principales conjuntos de métodos o técnicas de estimación (basadas en expertos, basadas en datos e híbridos), se muestra una mayor tendencia hacia el desarrollo de modelos basados en la manipulación y uso intensivo de datos para el cálculo o estimación del esfuerzo de un proyecto de software. Un total

de 33 estudios (71.74 %) ha utilizado al menos una técnica de estimación de este tipo. Esto sugiere que, a pesar de que la valoración subjetiva de expertos continua siendo uno de los factores de gran relevancia en entornos de desarrollo ágil, la investigación en los últimos años sobre modelos de estimación ha estado más orientada hacia el desarrollo de técnicas que utilicen herramientas sistematizadas y datos históricos de proyectos para la generación de las estimaciones.

En 14 de los estudios se han implementado modelos basados en datos y valoración de expertos a la vez, ya sea dentro de los categorizados como "híbridos" o por la combinación de dos o más modelos dentro del mismo estudio. De manera que ha habido un interés en el uso combinado de estos grupos de métodos, ya sea para poder obtener mejores resultados como para realizar comparaciones entre los modelos utilizados en términos de precisión y aplicabilidad.

Las métricas de precisión basadas en la magnitud del error relativo (MRE) son las utilizadas con mayor frecuencia, tal y como se presenta en la SLR anterior. Asimismo, se observan algunos estudios que implementan BRE (*Mean(BRE)*, *Median(BRE)*). Además, se detecta cierto interés en técnicas basadas en modelos de regresión para el calculo del nivel de precisión de modelos de estimación de esfuerzo, tales como CMSE, CMAE, CMRE, entre otras.

En general, se puede decir que se ha expandido el uso de diferentes métricas de precisión diferentes de MMRE y PRED(x), aunque aun persiste un número significativo de estudios que no calculan el nivel de precisión del modelo presentado, representando un 39.13 % del total de estudios primarios. Entre estos, la mayoría corresponde a estudios basados en métodos algorítmicos. En fin, se ha observado cierta negligencia entre los investigadores a la hora de presentar modelos de estimación sin proponer ningún mecanismo que permita verificar el nivel de exactitud y fiabilidad de tales modelos.

4.5.2. Predictores de esfuerzo

Los *story points* se mantienen como la principal medida de tamaño de proyectos de software. Asimismo, se observa la relación existente entre el principal método de estimación (*Planning Poker*), la medida de tamaño utilizada con mayor frecuencia (*story points*) y la forma en que los requerimientos de desarrollo aparecen especificados en ASD (*user story*), tal y como sugieren Usman et al. (2014).

Por otro lado, se destaca un creciente interés en el uso de métricas estándares basadas en FP (COSMIC, IFPUG, NESMA, etc.) para la determinación del tamaño. Este esfuerzo principalmente va de la mano con un deseo de introducir estas métricas de medición del tamaño funcional en entornos de desarrollo de software ágil. UCP, que aparecía entre las principales métricas de tamaño para el estudio anterior, ahora se muestra entre las menos utilizadas, dado que se han encontrado una cantidad mínima de estudios en los que se utilizan los casos de uso como la forma de especificación de los requerimientos funcionales.

En cuanto a los factores de coste, se han observado varios aspectos importantes. En primer lugar, se mantiene la gran diversidad de factores empleados y la falta de estandarización a nivel general, con al menos unos 47 factores de coste diferentes que han sido utilizados en los estudios primarios. Segundo, se han podido realizar agrupaciones de estos factores y establecer relaciones entre los mismos, lo que indica que estos no son utilizados de manera aislada y que van estrechamente ligados a las características de los proyectos evaluados en cada modelo. En este mismo sentido, el uso de factores de coste ha ido de la mano con los principios de desarrollo ágil, por ejemplo en el empleo de factores del equipo y del proyecto por encima de la consideración de los aspectos más técnicos. Por último, una gran cantidad de estudios, aproximadamente al 40 % de los artículos analizados, no indican los factores de coste considerados.

4.5.3. Características del repositorio de datos o dataset

Respecto a las características del dataset utilizado, se puede afirmar que se mantiene la tendencia presentada en el estudio anterior. La mayoría de los datasets son de dominio industrial y de carácter interno a una compañía específica. No obstante, existe un incremento significativo en el uso de repositorios *cross-company*, lo cual coincide con la extensión en el uso de metodologías ágiles para el desarrollo de software a nivel global en la última década. Además, esto extiende la aplicabilidad de los modelos presentados y la generalización de los resultados encontrados.

4.5.4. Contexto del proyecto

En el presente estudio, al menos 6 metodologías ágiles han sido identificadas (Scrum, XP, TDD, *Agile Unified Process*, Kanban y *Distributed Agile Software Development*), mientras que la SLR anterior solo menciona el uso de 2 metodologías ágiles (Scrum y XP). Evidentemente, el desarrollo de modelos de estimación de esfuerzo se ha extendido para abarcar los principales ámbitos de desarrollo de software ágil, de los cuales Scrum continua siendo el más utilizado entre los estudios primarios. Esto coincide, además, con el hecho de que esta sea la metodología ágil más utilizada en la industria de desarrollo en la actualidad.

Con respecto a la actividad de desarrollo y el nivel de planificación investigado, se observa que la mayoría de los estudios para los que se indican estos valores aborda la estimación de esfuerzo desde una perspectiva general (incluyendo todas las actividades del proyecto) y a nivel de cada iteración. Esto coincide con el enfoque iterativo e incremental de las metodologías ágiles, en el que las actividades de desarrollo no aparecen como bloques separados por fases estrictamente secuenciales, sino que se da mayor valor a la interacción entre las distintas actividades y al posible solape de las mismas.

Un 65.22 % de los estudios no menciona la actividad de desarrollo considerada en la estimación del esfuerzo. Esta resulta ser la pregunta de investigación con mayor porcentaje de estudios para los que no se ha identificado una respuesta concreta. Para el resto de preguntas de investigación, el porcentaje de estudios que no especifican alguna de las respuesta en relación a las preguntas de investigación analizadas se mantiene por debajo del 40 %.

Capítulo 5

Resultados experimentales con ISBSG

Existen diversos repositorios de datos en el campo de la ingeniería del software, con diversos grados de accesibilidad a la información (Boetticher, Menzies, y Ostrand, 2007). Los repositorios públicos ofrecen la oportunidad de disminuir los costes del proceso de recolección de datos que son inabordables para la mayoría de organizaciones (Top et al., 2011). ISBSG es uno de los más conocidos, utilizados y referenciados en la literatura sobre ingeniería de software, con una colección de un amplio número de variables discretas y una cantidad considerable de información descriptiva sobre distintas características de los proyectos de software (Cheikhi, Abran, y Desharnais, 2012).

5.1. ISBSG: Características generales

ISBSG[1] (International Software Benchmarking Standards Group) es una organización sin ánimo de lucro cuyo propósito es contribuir a la mejora de procesos de gestión de recursos en el sector de la ingeniería del software y de las tecnologías de la información, proveyendo recursos que permitan realizar multitud de análisis, estimaciones, comparaciones y/o benchmarking. Se formó en el año 1997 a partir de años de colaboración de varias asociaciones ya existentes en distintos países, con los mismos propósitos.

El objetivo principal de ISBSG es la acumulación de un cuerpo de conocimientos sobre cómo los proyectos de software son llevados a cabo, para poder contribuir con lecciones aprendidas mediante el análisis e investigación, y diseminar estas lecciones como buenas prácticas de desarrollo de software (C. Lokan, Wright, Hill, y Stringer, 2001). Los datos son evaluados y validados según las propias directrices de ISBSG (que son una derivación del estándar ISO/IEC Standard 15939:20074, adaptado a procesos

[1] isbsg.org

de software), de manera que sólo se admiten proyectos de empresas reconocidas con un cierto nivel de madurez en el sector. La Figura 5.1 presenta el proceso de recolección y almacenamiento de los datos en este repositorio. La versión pública (en formato de documento Excel) está disponible para investigadores y profesionales del área a un coste mínimo.

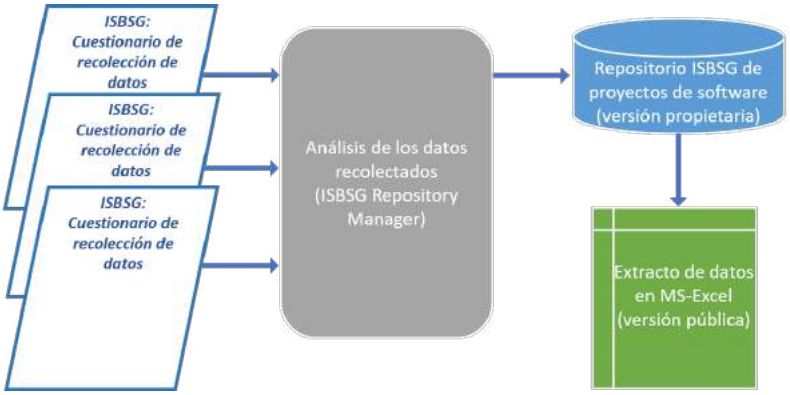

Figura 5.1: Proceso de recolección y almacenamiento de datos en el repositorio ISBSG. Adaptado de (Abran, 2015).

ISBSG mantiene dos repositorios distintos[2]:

a) *Software Maintenance and Support*. Cuenta con datos recogidos de más de 1000 proyectos, que contienen información de aplicaciones que ya están en producción y requieren soporte y mantenimiento.
b) *Software Development and Enhancement*. ISBSG ha ido recopilando datos de proyectos de desarrollo de software de industrias de diferentes sectores, tamaños e incluso países. Este repositorio se organiza en versiones. La versión actual (2017 R1), publicada en mayo del 2017, cuenta con 8012 proyectos y 252 variables descriptivas de cada proyecto.

La Figura 5.2 muestra el número de proyectos en función de la versión de la base de datos ISBSG. Se puede observar que la última versión (2017 R1) registra un crecimiento significativo respecto a los años anteriores, esto es, se han añadido un total de 2,006 proyectos nuevos, lo que representa un crecimiento del 33.40 %. A su vez, esta cantidad de proyectos nuevos representa aproximadamente el doble de lo registrado para la mayoría de las versiones anteriores.

[2]isbsg.org/software-project-data

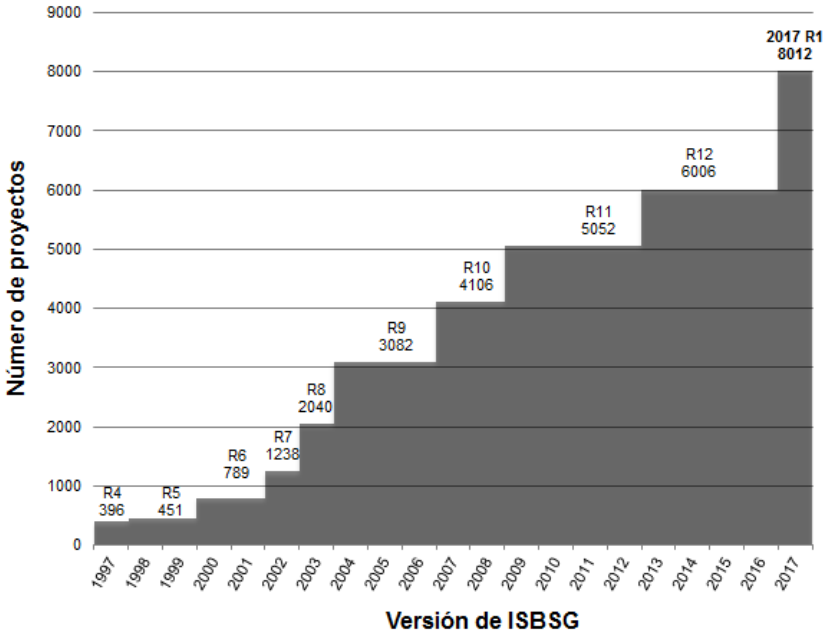

Figura 5.2: Versiones de ISBSG.

La base de datos ISBSG está reconocida como una de las más amplias, fiables, sistemáticas y representativas de su clase. Al contrario que otras bases de datos con objetivos similares, los datos recogidos por ISBSG provienen de proyectos de software de múltiples empresas (*cross-company*), de distintos tamaños, de diferentes sectores de la industria, y de distintos países, aplicaciones y tipos de desarrollo, por lo que suponen una muestra objetiva y variopinta, capaz de cubrir un amplio espectro de las necesidades de los usuarios. Esta información relativa a proyectos, prácticas, herramientas, metodologías y entornos organizativos proyectuales facilita la estimación del tamaño de un proyecto, esfuerzo, duración y coste (Top et al., 2011). A pesar de que, en comparación con otros repositorios de datos, ISBSG presenta un gran potencial que puede ser explotado por la comunidad científica, no deja de presentar una serie de limitaciones (Fernández-Diego y González-Ladrón-De-Guevara, 2014). Entre estas se encuentran:

- Los proyectos no son seleccionados de forma aleatoria, sino que las organizaciones participantes presentan únicamente sus mejores proyectos para ser incluidos en el repositorio, lo que supone un evidente sesgo.
- Los datos recolectados provienen de una gran variedad de entidades con trasfondos, culturas de negocio, nivel de experiencia del personal y nivel de madurez diferentes. Por lo tanto, la información puede resultar inconsistente y dificultar la obtención de predicciones precisas, requiriendo de un proceso de preparación y filtrado de los datos para obtener un subconjunto apropiado.
- La gran cantidad de datos perdidos resulta en una disminución significativa de la muestra de proyectos que pueden ser utilizados para la construcción de modelos

de estimación.
- La existencia de valores aislados (*outliers*) puede provocar un incremento de la magnitud de error relativo.

En la actualidad este repositorio de datos está siendo utilizado por varios grupos de interés:

- Para los profesionales del sector, este repositorio de datos es de interés para llevar a cabo una planificación y gestión realista de sus propios proyectos, para proporcionar un marco con el que comparar sus proyectos con los estándares de la industria y como asistencia a la hora de evaluar los beneficios de un posible cambio de software o hardware en sus entornos de desarrollo.
- Para los investigadores, estos datos son utilizados como base para la investigación y desarrollo de modelos matemáticos que mejoren esta industria y sus procesos en el futuro.

5.1.1. Selección de los proyectos del repositorio ISBSG

Para la estimación de esfuerzo en proyectos de desarrollo de software, en (González-Ladrón-de-Guevara et al., 2016) se identifican tres aspectos clave a la hora de seleccionar y filtrar los proyectos del repositorio ISBSG para su uso en la construcción de modelos de estimación. Estos aspectos son: (1) la calidad de los datos, (2) el método de medición del tamaño del proyecto y (3) los distintos valores de esfuerzo registrados. Además de estos, el presente estudio añade un cuarto factor, que es la metodología de desarrollo de software utilizada.

5.1.1.1. Calidad de los datos

Una de las razones principales por las que se realiza el filtrado de proyectos en el repositorio ISBSG es para garantizar el uso de datos de alta calidad (González-Ladrón-de-Guevara et al., 2016, p. 193). Liebchen y Shepperd (2008) resaltan la importancia de considerar explícitamente la calidad de los datos del repositorio utilizado como base de una investigación, dado que la mala calidad puede afectar la validez de las conclusiones obtenidas en el estudio.

En el repositorio ISBSG existen dos campos relativos a la calidad de la información almacenada: *Data Quality Rating* (DQR) y *Unadjusted Function Point Rating* (UFPR). Ambas variables son valoradas mediante un sistema alfabético (A, B, C, D), donde A representa la mejor valoración y D representa la peor. DQR permite determinar la confiabilidad, completitud y consistencia del registro almacenado, mientras que UFPR sirve para identificar la integridad y precisión de la medición del tamaño funcional mediante UFP. Para ambos casos, ISBSG recomienda utilizar aquellos proyectos cuya calificación en estas variables de calidad sea A o B, y que el resto de registros sea excluido (González-Ladrón-de-Guevara et al., 2016, p. 194).

5.1.1.2. Método de medición del tamaño del proyecto

El tamaño es uno de los factores de costo y tiempo más significativos e importantes en proyectos de desarrollo de software (Galorath y Evans, 2006, p. 53). Una de las razones del uso de esta variable como condición de filtro en ISBSG es la necesidad de que los proyectos seleccionados por los investigadores, especialmente para fines de propuesta de modelos de estimación de esfuerzo y tamaño, sean comparables entre sí, dado que cada método de conteo utiliza métricas diferentes en sus cálculos del tamaño del objeto de estimación.

La estimación del tamaño puede realizarse mediante opinión de expertos, estimación por analogía o metodologías formalizadas a través de algoritmos y/o herramientas automatizadas (Galorath y Evans, 2006, pp. 58–60). En ISBSG, la variable *Count Approach* (CA) es utilizada para indicar la métrica de medición del tamaño de los proyectos.

5.1.1.3. Valores de esfuerzo registrados

El tercer factor a tener en cuenta hace referencia a los valores asignados a las distintas variables de ISBSG relativas al esfuerzo. En concreto, esta base de datos utiliza tres campos distintos: *Normalised Work Effort* (NWE), *Normalised Level 1 Work Effort* (NL1WE) & *Summary Work Effort* (SWE). Esta última variable representa el esfuerzo total del proyecto y es medida en horas de personal (*staff hours*). Las dos anteriores son añadidas por ISBSG para tener en cuenta quiénes (personal de trabajo) han sido incluidos en SWE y qué fases del ciclo de vida del proyecto han sido consideradas (González-Ladrón-de-Guevara et al., 2016). En dicho estudio, se identifica una tendencia de los investigadores a seleccionar aquellos proyectos para los cuales se cumple que NWE es igual o muy aproximado a SWE. De igual forma, se recomienda considerar la variable *Resource Level* (RL) igual a 1, lo que implica que el esfuerzo almacenado en la variable corresponde exclusivamente al equipo de desarrollo.

5.1.1.4. Metodología de desarrollo

Tal y como se comenta anteriormente, en este estudio se añade la metodología de desarrollo de software como condición de filtro, dado que el objetivo es considerar únicamente aquellos proyectos desarrollados mediante el uso de metodologías ágiles. La variable de ISBSG para realizar este filtrado es *Development Methodologies* (DM), que describe la metodología empleada para el desarrollo de software. Para ISBSG, los métodos identificados en este campo son aplicables a todo el proceso desarrollo del proyecto; esto lo diferencia de la variable *Development Techniques*, que describe las técnicas aplicadas en actividades específicas dentro del proceso de desarrollo (ISBSG, 2013).

5.2. Resultados experimentales con ISBSG R12

El *dataset* ISBSG R12 cuenta con un total de 6006 registros (proyectos) y 126 variables o campos que identifican las características principales de los mismos. Estos proyectos están comprendidos entre los años 1989 y 2012. Según la Figura 5.3, entre 1998 y 2010 se registran más de 200 proyecto al año. La mayor cantidad de proyectos corresponde al 2004, con un total de 707 proyectos.

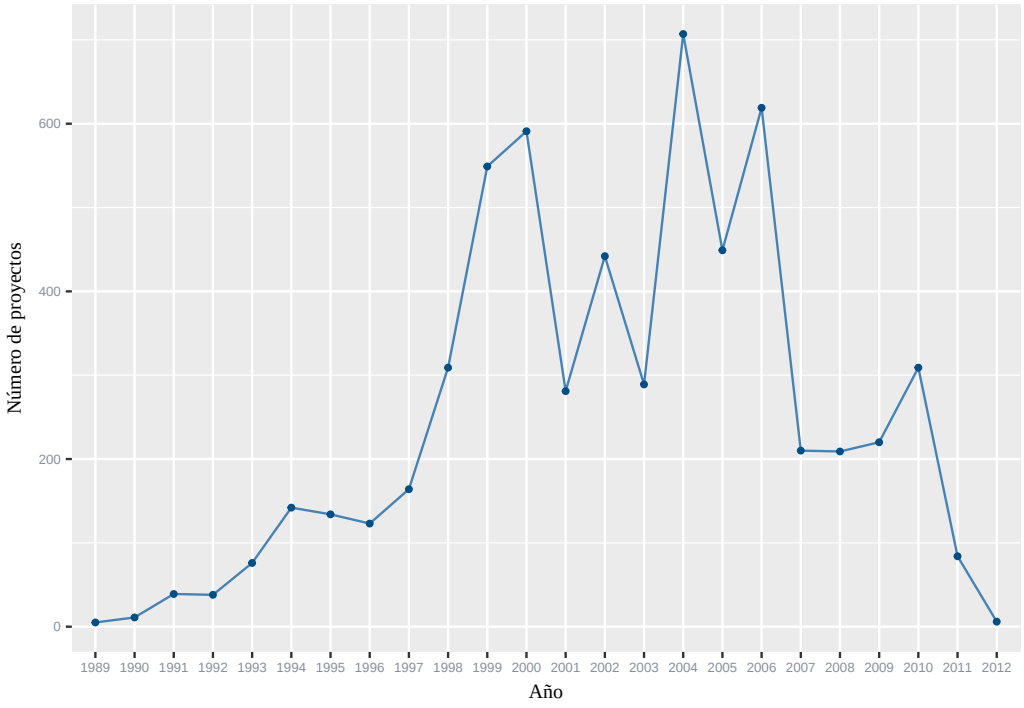

Figura 5.3: Número de proyectos por año

La variable utilizada para determinar el año del proyecto es el campo *Year of Project*; este campo, a su vez, ha sido derivado de la fecha de implementación del proyecto, en caso de ser conocida. Para el resto de casos en que dicho valor se desconozca, se han considerado otros datos del dataset para calcular el año, tales como la fecha de inicio o final del proyecto, la fecha estimada de implementación, la fecha de recopilación de los datos o, en su defecto, la fecha en que se registró dicho proyecto en el repositorio (ISBSG, 2013).

5.2.1. Metodologías de desarrollo

El análisis de la variable DM ha permitido identificar tres aspectos importantes. Por un lado, el porcentaje de datos perdidos de esta variable es bastante significativo, lo que implica que una gran parte de los proyectos ha de ser excluida de la muestra. Además,

la variable contiene tanto paradigmas y modelos de proceso de desarrollo de software, como prácticas y técnicas específicas. Por último, los proyectos pueden contener más de una metodología de desarrollo. Todo esto se analiza a continuación.

5.2.1.1. Valores perdidos

La cantidad de proyectos que no indican la metodología utilizada es de 4,158, esto es, aquellos registros para los cuales se cumple que $DM = \text{“}NA\text{”}$. Esta cantidad representa el 69,23 % del total de proyectos en el repositorio ISBSG R12. Esto significa que más de dos terceras partes de estos datos proyectos no serían útiles para la muestra considerada en el presente estudio.

La Figura 5.4 muestra la relación entre los proyectos con valores perdidos en la variable DM y el total de proyectos para cada año. Se puede observar un comportamiento relativamente proporcional, siendo nuevamente el año 2004 el de mayor cantidad de proyectos con valores perdidos (668).

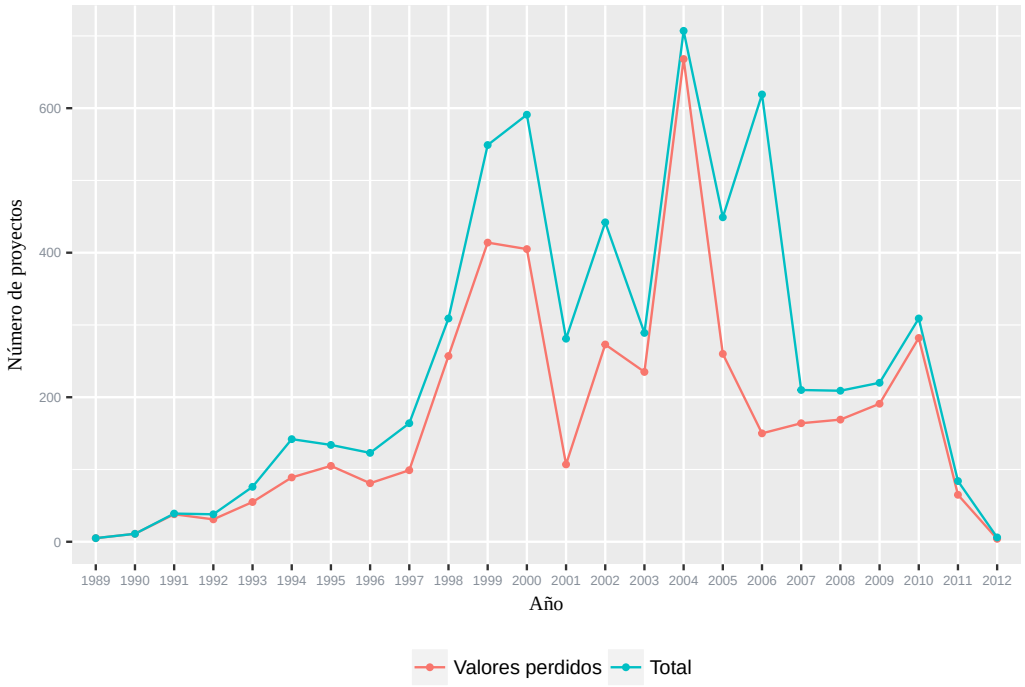

Figura 5.4: Relación entre los proyectos con valores perdidos en la variable DM y el total de proyectos para cada año.

5.2.1.2. Categorías de la variable DM

Al analizar los valores de la variable DM, se ha optado por realizar la siguiente clasificación de las categorías encontradas:

- **Modelos/Paradigmas/Procesos**. En (Sommerville, 2010), un modelo de proceso de desarrollo de software se define como una "representación simplificada de un proceso de software." Estos modelos y marcos generales son extendidos y aplicados mediante diferentes enfoques de desarrollo de software. *Waterfall, Iterative, y Spiral* son algunos ejemplos de estos modelos, así como también *Personal Software Process* (PSP) (Humphrey, 1996) y *Unified Process* (UP) (Scott, 2002). *Agile*, por su parte, suele ser considerado como un enfoque o paradigma dentro de los modelos de desarrollo incremental e iterativo (Larman, 2003), siendo definido por un conjunto de principios y valores en lugar de un modelo de proceso específico. Este subgrupo se ha extendido para incluir otros modelos de procesos de desarrollo de softwaren como *Rapid Application Development* (RAD) (Martin, 1991) y *Joint Application Development* (JAD) (Wood y Silver, 1995).
- **Metodologías**. Según (Cockburn, 2000), una metodología está compuesta de un conjunto de elementos (personas, roles, habilidades, equipos, herramientas, técnicas, procesos, actividades, hitos, productos de trabajo, estándares, medidas de calidad, y valores del equipo), que definen cómo se produce y se entrega un sistema. *Scrum*, *Extreme Programming* y *Lean Software Development* son ejemplos específicos de metodologías de desarrollo que se han podido identificar en este repositorio.
- **Prácticas/Técnicas**. Constituyen las herramientas de soporte para el proceso de desarrollo de software, siendo implementadas en actividades específicas dentro de dicho proceso. *Multifunctional Teams* y *Timeboxing* son técnicas características de las metodologías ágiles.

En resumen, los modelos de proceso de desarrollo de software definen el marco general, mientras que las metodologías permiten aplicar un determinado modelo a un proyecto de desarrollo de software mediante el uso de determinadas prácticas y técnicas. En ISBSG R12 se identifican 13 valores o categorías distintas en la variable DM. Estas categorías incluyen tanto modelos de proceso como metodologías y prácticas específicas de desarrollo de software. La Tabla 5.1 muestra para cada una de estas categorías el número de proyectos asociados y su porcentaje respecto a los 1,848 proyectos con valor en esta variable.

Tabla 5.1: Categorías de la variable DM en ISBSG R12.

Categorías	Número de proyectos	Porcentaje
Modelos/Paradigmas:		
Iterative	4	0.22 %
Spiral	4	0.22 %
Personal Software Process	12	0.65 %
Unified Process	19	1.03 %
Agile Development	38	2.06 %
Joint Application Development	152	8.23 %

Rapid Application Development	152	8.23 %
Waterfall	1423	77.00 %
Metodologías:		
Extreme Programming	1	0.05 %
Lean	2	0.11 %
Scrum	2	0.11 %
Prácticas/Técnicas:		
Timeboxing	38	2.06 %
Multifunctional Teams	138	7.47 %

Entre los modelos de proceso de desarrollo de software representados en este *dataset*, el más utilizado es *Waterfall* (77 %), mientras que los modelos Iterative y Spiral son los menos utilizados (0.22 %). Por otro lado, *Rapid Application Development* (8.23 %), *Joint Application Development* (8.23 %) y *Multifunctional Teams* (7.47 %) presentan los porcentajes más altos entre el resto de modelos, metodologías y prácticas de desarrollo, la mayoría de estos con menos de 2 % respecto al total de proyectos que indican algún valor en la variable DM.

No obstante, es posible reorganizar la tabla anterior agrupando las categorías que corresponden a metodologías ágiles. Por ejemplo, *Extreme Programming, Lean* y *Scrum* son metodologías ASD, según se ha observado en el Capítulo 2, así como también *Multifunctional Teams* y *Timeboxing* son prácticas características de ASD. En la Tabla 5.2 se presentan nuevamente las categorías con los ajustes realizados.

Tabla 5.2: Categorías de la variable DM en ISBSG R12 con agrupación de las metodologías ASD.

Categorías	Número de proyectos	Porcentaje
Iterative	4	0.22 %
Spiral	4	0.22 %
Personal Software Process	12	0.65 %
Unified Process	19	1.03 %
Joint Application Development	152	8.23 %
Rapid Application Development	152	8.23 %
Agile Development	203	10.98 %
Waterfall	1423	77.00 %

Ahora se observa que la categoría *Agile Development* cuenta con un total de 203 proyectos (10.98 %). Es importante resaltar que los porcentajes son respecto al total de proyectos que contienen algún valor en la variable DM (un 30.77 % del total de proyectos en este repositorio). Además, no se ha realizado una suma de los totales de las categorías aquí agrupadas, debido a que la multiplicidad de valores de esta variable

impide dicho tratamiento, lo cual se analiza en el punto a continuación.

5.2.1.3. Combinaciones de categorías

En total, se han encontrado 32 combinaciones de las categorías mencionadas en el punto anterior. La Tabla 5.3 muestra esta distribución, la cantidad de proyectos y el porcentaje respecto al total de proyectos con valores en la variable DM.

Tabla 5.3: Combinaciones de categorías en la variable DM de ISBSG R12.

	Categorías	Número de proyectos	Porcentaje	Grupo
1	Agile Development;Iterative;	1	0.05 %	Agile
2	Agile Development;Joint Application Development (JAD);Multifunctional Teams;	1	0.05 %	Agile
3	Agile Development;Multifunctional Teams;Scrum;	1	0.05 %	Agile
4	Agile Development;Scrum;	1	0.05 %	Agile
5	Agile Development;Unified Process;	1	0.05 %	Agile
6	Extreme Programming (XP);	1	0.05 %	Agile
7	Joint Application Development (JAD);Multifunctional Teams;Timeboxing;	1	0.05 %	Otros
8	Joint Application Development (JAD);Timeboxing;	1	0.05 %	Otros
9	Multifunctional Teams;Rapid Application Development (RAD);Timeboxing;	1	0.05 %	Otros
10	Personal Software Process (PSP);	1	0.05 %	Otros
11	Lean;	2	0.11 %	Agile
12	Multifunctional Teams;Unified Process;	2	0.11 %	Otros
13	Rapid Application Development (RAD);Timeboxing;	2	0.11 %	Otros
14	Iterative;	3	0.16 %	Otros
15	Joint Application Development (JAD);Rapid Application Development (RAD);Timeboxing;	3	0.16 %	Otros
16	Joint Application Development (JAD);Multifunctional Teams;Rapid Application Development (RAD);	4	0.22 %	Otros
17	Multifunctional Teams;Timeboxing;	4	0.22 %	Otros
18	Spiral;	4	0.22 %	Otros

19	Multifunctional Teams;Waterfall (includes Linear Processing);	5	0.27 %	Otros
20	Personal Software Process (PSP);Unified Process;	5	0.27 %	Otros
21	Unified Process;	5	0.27 %	Otros
22	Agile Development;Personal Software Process (PSP);Unified Process;	6	0.32 %	Agile
23	Joint Application Development (JAD);Multifunctional Teams;Rapid Application Development (RAD);Timeboxing;	6	0.32 %	Otros
24	Multifunctional Teams;Rapid Application Development (RAD);	6	0.32 %	Otros
25	Timeboxing;	20	1.08 %	Otros
26	Joint Application Development (JAD);Rapid Application Development (RAD);	24	1.30 %	Otros
27	Agile Development;	27	1.46 %	Agile
28	Joint Application Development (JAD);Multifunctional Teams;	33	1.79 %	Otros
29	Multifunctional Teams;	74	4.00 %	Otros
30	Joint Application Development (JAD);	79	4.27 %	Otros
31	Rapid Application Development (RAD);	106	5.74 %	Otros
32	Waterfall (includes Linear Processing);	1418	76.73 %	Otros
	Total	**1848**	**100.00 %**	

Esta variable presenta uno de los principales problemas de normalización de base de datos. Al tratarse de un campo multivalorado (Teorey et al., 2008), se dificulta la búsqueda y selección de registros cuando se utiliza esta variable como referencia. En el presente estudio, el objetivo principal es identificar qué proyectos de este repositorio han sido implementados mediante metodologías ASD, para así poder realizar un análisis y comparación de sus atributos principales. Para ello ha sido necesario reagrupar nuevamente estas categorías en dos grandes grupos de categoría:

- **Agile**: aquellos proyectos implementados mediante el uso de una o varias metodologías ágiles, incluyendo los casos en los que se utilicen en combinación con otras metodologías o técnicas de desarrollo.
- **Otros**: aquellos proyectos implementados mediante modelos o metodologías diferentes de Agile (JAD, RAD, Iterative, PSP, Spiral, UP, Waterfall, o la combinación de estos). También se considerarán en esta categoría:
  - Casos en los que se muestre una o varias de las categorías aquí agrupadas, utilizadas junto con alguna de las técnicas identificadas (*Timeboxing y/o Multifunctional Teams*) y sin especificación de ninguna metodología ágil.

- Casos en los que se muestre el uso de algunas de las técnicas identificadas (*Timeboxing y/o Multifunctional Teams*) sin la especificación de ninguna metodología ágil.

La columna "Grupo" de la Tabla 5.3 muestra como se ha seguido esta regla para cada una de las posibles combinaciones de categorías en este repositorio. Siguiendo esta nueva clasificación, se obtienen los resultados recogidos en la Tabla 5.4.

Tabla 5.4: Grupos de categorías de la variable DM de ISBSG R12.

Grupos de categorías	Número de proyectos	Porcentaje
Agile	41	2.22 %
Otros	1807	97.78 %
Total	**1848**	**100.00 %**

De un total de 1,848 proyectos, únicamente el 2.22 % corresponde a los que han sido desarrollados mediante el uso de alguna metodología ASD, mientras que un 97.78 % ha sido implementado con otras metodologías (de las cuales el mayor porcentaje corresponde a *Waterfall*, según se observó en la Tabla 5.2).

La Figura 5.5 muestra el número de proyectos correspondientes para cada año según los dos grupos definidos para la variable DM. Debido a la cantidad mínima de proyectos desarrollados con metodologías ágiles, apenas se logra observar un leve crecimiento entre los años 2008 y 2010.

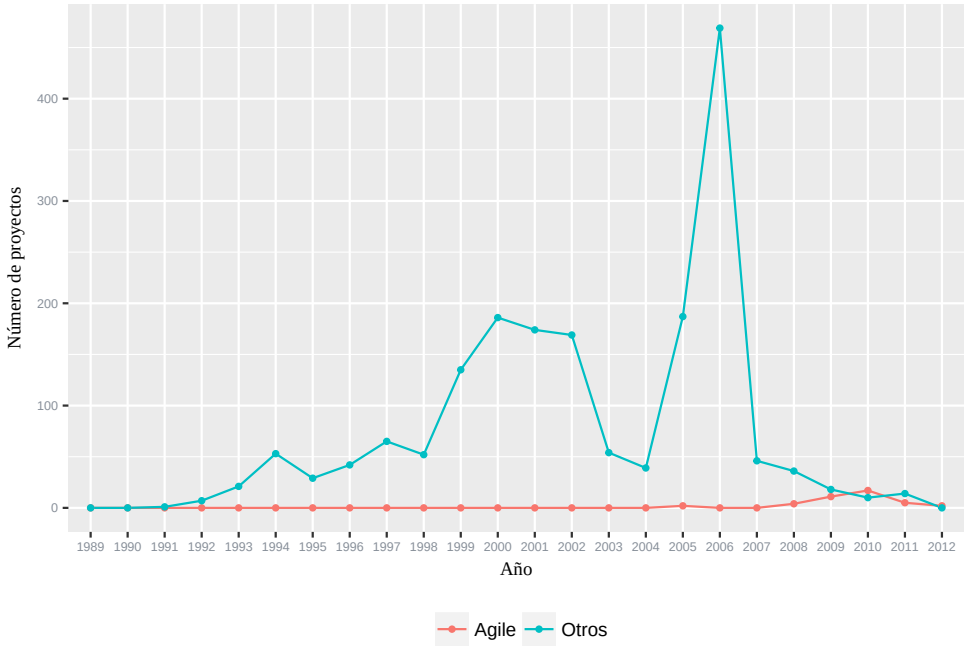

Figura 5.5: Número de proyectos por año según el tipo de metodología.

5.2.2. Calidad de los datos

En la Tabla 5.5 se muestra la distribución de los proyectos acorde al tipo de metodología de desarrollo utilizado y el valor del campo DQR. Considerando únicamente los proyectos cuyo DQR sea igual a 'A' o 'B', un total de 1740 proyectos quedarían seleccionados, lo que representa el 94.16 % del total de proyectos en esta base de datos que indican la metodología de desarrollo utilizada. Sin embargo, sólo el 2.16 % corresponde a proyectos desarrollados con metodologías ágiles.

Tabla 5.5: Valor de DQR por grupos de categorías en ISBSG R12.

	DQR			
	A	**B**	**C**	**D**
Agile	21	19	1	0
Otros	255	1445	45	62
Total	**276**	**1464**	**46**	**62**

Por otro lado, al considerar la variable UFPR se obtiene la distribución presentada en la Tabla 5.6. En este caso, 1263 proyectos han sido calificados con UFPR igual a 'A' o 'B', esto equivale al 68.34 % de los proyectos que serían aptos para ser seleccionados en caso de utilizar esta variable para el filtrado y selección. De este total, 'Agile' representa el 1.19 %. Además, es importante destacar que esta variable posee datos perdidos (NA), a diferencia de DQR. Por este motivo se ha incluido en esta tabla una columna adicional para identificar los datos perdidos en esta variable para cada uno de los grupos de categorías definidos, representando un 3.41 % del total de proyectos aquí identificados.

Tabla 5.6: Valor de UFPR por grupos de categorías en ISBSG R12.

	UFPR				
	A	**B**	**C**	**D**	**NA**
Agile	9	13	0	0	19
Otros	859	382	518	4	44
Total	**868**	**395**	**518**	**4**	**63**

Una vez se han considerado estas dos variables de calidad de forma independiente, el subconjunto resultante de proyectos cuyo DQR y UFPR es igual a 'A' o 'B', para cada uno de los grupos de categorías definidos, se muestra en la Tabla 5.7.

Tabla 5.7: Número de proyectos tras el filtrado de calidad en ISBSG R12.

Grupos de categorías	Número de proyectos	Porcentaje
Agile	21	1.14 %
Otros	1146	62.01 %
Total	**1167**	**63.15 %**

Según se observa, un 63.15 % de los proyectos con valor en la variable DM cumple con estos criterios de calidad de los datos, de los cuales el 1.14 % ha sido desarrollado con metodologías ágiles.

5.2.3. Medidas del tamaño del proyecto

La *Release* 12 del repositorio ISBSG contiene 17 categorías diferentes en la variable *Count Approach* (CA), la cual almacena la métrica utilizada para la medición del tamaño del proyecto. La mayor parte de proyectos en ISBSG utilizan métodos FSM (ISBSG, 2013). Según se presenta en la Figura 5.6, un 71.89 % de los proyectos registrados indican haber utilizado IFPUG 4+ para la estimación del tamaño funcional del software. Del resto de métodos utilizados, FiSMA (9.66 %) y COSMIC (7.54 %) ocupan el segundo y tercer lugar, respectivamente.

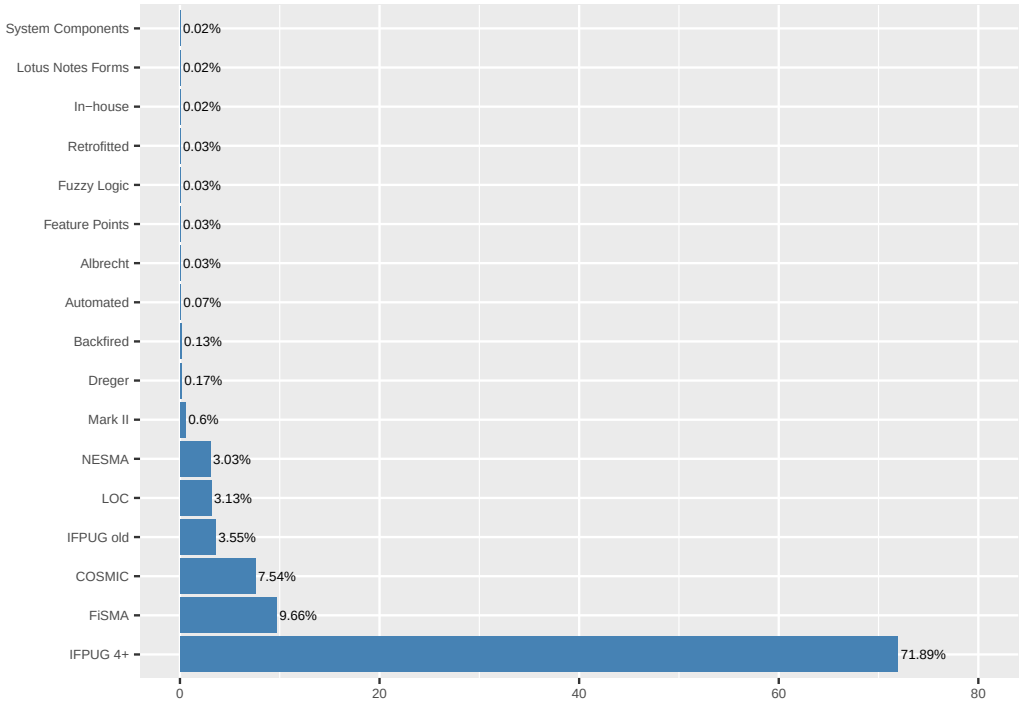

Figura 5.6: Porcentaje de proyectos según la métrica de tamaño utilizada.

Considerando aquellos proyectos que hayan utilizado *IFPUG 4+* como métrica, la distribución correspondiente a los grupos de categorías previamente definidos se muestra en la Tabla 5.8. Un total de 1,481 proyectos indican la metodología de desarrollo utilizada y la métrica de tamaño funcional es *IFPUG 4+*. Sólo el 1.19 % ha sido desarrollado con metodologías ágiles, frente a un 78.95 % de los proyectos desarrollados con otras metodologías diferente de ASD.

Tabla 5.8: Número de proyectos que han utilizado IFPUG 4+ como métrica de tamaño en ISBSG R12.

Grupos de categorías	Número de proyectos	Porcentaje
Agile	22	1.19 %
Otros	1459	78.95 %
Total	**1481**	**80.14 %**

5.2.4. Medidas del esfuerzo estimado

En esta condición de filtro sólo se ha de evaluar que los valores de esfuerzo registrados para los proyectos correspondan al equipo de desarrollo, esto es, *Resource Level == 1*. Del total de proyectos en este repositorio, solamente el 92.42 % cumple con esta condición. Es importante destacar que se han excluido los proyectos cuya variable de esfuerzo no es conocida, es decir, solo se han considerado aquellos para los cuales SWE es diferente de "NA". Asimismo, se observa que 1.84 % de estos proyectos han utilizado metodologías ágiles, frente a un 90.58 % que ha empleado otras metodologías de desarrollo.

Tabla 5.9: Número de proyectos de proyectos con 'RL == 1' en ISBSG R12.

Grupos de categorías	Número de proyectos	Porcentaje
Agile	34	1.84 %
Otros	1674	90.58 %
Total	**1708**	**92.42 %**

5.2.5. Muestra final de proyectos seleccionados

En resumen, debido al carácter heterogéneo de los datos provistos por ISBSG, se recomienda a los investigadores que utilicen ciertas condiciones que permitan filtrar y seleccionar muestras de proyectos contenidos en este repositorio, cuyos valores sean confiables, comparables y útiles para el desarrollo de modelos de estimación. La Figura

5.7 resume los criterios de filtrado considerados.

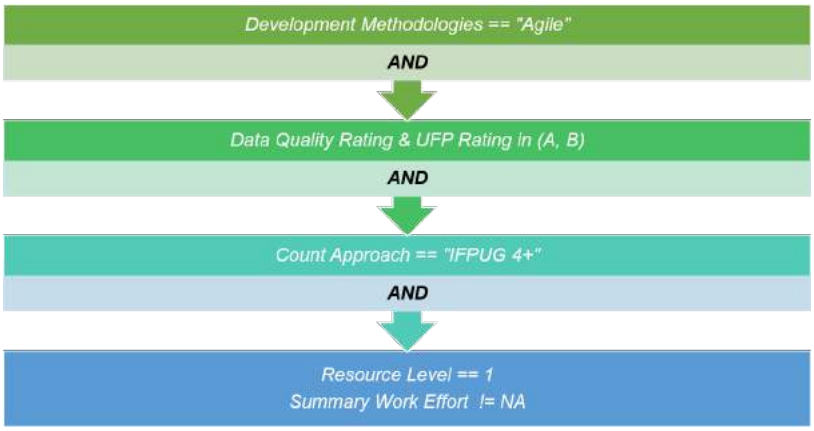

Figura 5.7: Criterios de filtrado del repositorio ISBSG R12.

Una vez agregadas estas condiciones, se obtiene la Tabla 5.10 como resultado de proceso realizado. Estos resultados se muestran para los dos grupos de categorías previamente definidos, donde se identifica que, de los proyectos que indican la metodología de desarrollo, un 48.54 % cumple con los criterios de filtrado descritos previamente. De los mismos, aquellos desarrollados con metodologías ágiles representan el 1.14 %, frente al 47.40 % que ha utilizado otras metodologías diferentes de ASD. Otro elemento a resaltar de esta tabla es que los proyectos dentro del grupo "Agile" presentan cierta consistencia respecto a la cantidad eliminada en cada uno de los filtros (Calidad, Tamaño y Esfuerzo), mientras que los desarrollados con otras metodologías varían considerablemente para cada filtro aplicado. Estos últimos, además, pasan de un promedio de 1,426 (teniendo en cuenta los criterios individualmente) a un total de 876 luego de agrupar los criterios (38.57 % menos que el promedio).

Tabla 5.10: Cantidad de proyectos tras aplicar los criterios de filtro.

	Calidad		Tamaño		Esfuerzo		Todos	
Grupos de categorías	No.	(%)	No.	(%)	No.	(%)	No.	(%)
Agile	21	1.14 %	22	1.19 %	34	1.84 %	21	1.14 %
Otros	1146	62.01 %	1459	78.95 %	1674	90.58 %	876	47.40 %
Total	**1167**	**63.15 %**	**1481**	**80.14 %**	**1708**	**92.42 %**	**897**	**48.54 %**

Capítulo 6

Conclusiones

El presente trabajo de investigación estudia el uso de modelos de estimación de esfuerzo para proyectos de software desarrollados mediante metodologías ágiles. Como marco de referencia del estudio, primero se abordan los conceptos fundamentales relacionados con el desarrollo de software ágil y el estado del arte de las principales metodologías que forman parte de este movimiento. En segundo lugar, se desarrollan los principales aspectos de la estimación de esfuerzo en proyectos de software, presentando los esquemas de clasificación de los procedimientos y métodos para la estimación de esfuerzo, el uso de métricas de precisión y predictores de esfuerzo (en relación al tamaño y los factores de coste del proyecto de software) y la aplicación de estos conceptos en entornos de *Agile Software Development* (ASD).

La revisión sistemática de literatura se realiza con el objetivo de analizar el concepto de estimación de esfuerzo en el ámbito específico de desarrollo de software ágil. Esta revisión está basada en un estudio anterior presentado en 2014, por lo que permite ampliar el marco temporal e identificar como ha evolucionado el uso de modelos de estimación de esfuerzo en el ámbito de la investigación. Una búsqueda en 5 librerías digitales ha arrojado un total de 638 artículos únicos, de los cuales solo 46 estudios cumplieron con los criterios de inclusión/exclusión definidos para este estudio. Estos 46 estudios primarios son luego analizados en base a una serie de preguntas de investigación que permiten extraer los datos relevantes de cada estudio en relación al método de estimación implementado, los predictores de esfuerzo utilizados, el contexto de los proyectos de software analizados en el estudio y la metodología de desarrollo ágil implementada en tales proyectos.

En general, se puede afirmar que la investigación sobre modelos de estimación de esfuerzo en entornos desarrollo ágil ha ganado gran interés en los últimos 4 años. Una mayor variedad de modelos han sido investigados y/o desarrollados desde un punto de vista empírico, incluyendo tanto los basados en la valoración subjetiva de expertos como los que van enfocados hacia el manejo de datos para la generación de un valor estimado. Un punto que resalta en el presente estudio es como el uso de *Machine Learning* ha capturado gran interés por parte de los investigadores, relegando

a últimas posiciones aquellos métodos que emplean casos de uso e incluso compitiendo a la par con *Planning Poker*, *Wideband Delphi* y el resto de modelos basados en juicio de expertos. Esto demuestra una nueva tendencia que vale la pena observar, dado que un esfuerzo importante ha sido dedicado a la adaptación de estos modelos robustos a un entorno más flexible y dinámico, que es característico del desarrollo ágil. Todo ello apunta hacia modelos automatizados, aunque el carácter heterogéneo de los proyectos en ASD dificulta la generalización de los resultados, por lo cual se constituye en el principal reto para esta clase de modelos de estimación.

El nivel de precisión de los modelos de estimación investigados es relativamente aceptable para la mayoría de estudios según los valores de MMRE que es la métrica de precisión más utilizada entre los distintos métodos investigados. Es decir, la mayoría de estudios presenta resultados por debajo del umbral del 25 % para MMRE. Sin embargo, aun se presentan algunos valores que carecen de claridad al analizar el contenido del estudio, lo cual genera cuestiones respecto a la fiabilidad de los modelos planteados. A todo ello se suman los debates referentes a la interpretación de las medidas de precisión basadas en la magnitud de error relativo (M. Jørgensen, 2007; Kitchenham et al., 2001).

Story points se mantiene como la principal métrica para medir el tamaño en proyectos de desarrollo ágil, estrechamente relacionada con la forma en que los requerimientos de estos proyectos vienen expresados. Por otro lado, existe una mayor gama de factores de coste utilizados como predictores de esfuerzo. Esto demuestra una falta de acuerdo general en el uso de tales factores, por lo que se afirma que aun existe una brecha considerable a tomar en cuenta, tanto a nivel de estandarización de los principales factores de coste relevantes para la estimación de esfuerzo en ASD como en la difusión del uso de los mismos.

El uso de repositorios industriales y la creciente incorporación de datasets *cross-company* son valores indicativos del progreso hacia modelos de estimación de esfuerzo que puedan ser aplicados por los profesionales de la industria de software. No obstante, aun hace falta una mayor cantidad de estudios en este respecto, en los que se pueda identificar con mayor claridad como las características particulares de los proyectos pueden afectar el proceso de estimación y las variables consideradas.

Aproximadamente una tercera parte de los estudios primarios analizados en la SLR no indica la metodología de desarrollo implementada. Debido a que *Agile* es un paradigma o modelo de desarrollo abstracto, basado en valores y principios que pueden ser aplicados directamente en proyectos de software, es difícil adscribir un determinado conjunto de proyectos a una de las metodologías ágiles a menos de que se especifique explícitamente en el estudio realizado. Por ende, se requiere mayor claridad por parte de los investigadores a la hora de contextualizar sus modelos, así como también la especificación de las actividades de desarrollo incluidas en la estimación y los niveles de planificación en el que se realiza dicho proceso.

6.1. Limitaciones del estudio

La principal amenaza a la validez de los resultados presentados en este estudio está relacionada con el número elevado de datos perdidos resultantes luego de la extracción de datos de cada una de las preguntas de investigación, y que podrían ser relevante para el presente estudio. Esto dificulta significativamente el proceso de análisis comparativo y puede resultar en una representación parcial de la realidad para aquellos casos en los que más del 30 % de los estudios no aporta información para alguna de las variables relevantes del estudio, tales como el entorno de desarrollo, el nivel de aplicación o la precisión de los resultados del modelo presentado.

En relación con el proceso de desarrollo de la SLR, la selección de los estudios se ha realizado de forma consensuada entre tres investigadores, lo que reduce la posibilidad de que los resultados se vean afectados por un sesgo de selección, es decir, influenciados por el criterio de un único investigador. No obstante, debido a la gran variedad de publicaciones, no es posible asegurar que todos los estudios relevantes han sido seleccionados. A esto se le suma el hecho de que tres de los motores de búsqueda digital propuestos en la SLR original no han podido ser accedidos, lo cual reduce sin duda el número de estudios seleccionados. Sin embargo, la selección en base a *ACM Digital Library, IEEE Xplore, Science Direct, SCOPUS* y *WOS* se considera adecuada al estar estas ampliamente reconocidas por la comunidad científica y proporcionar un conjunto exhasutivo de resultados.

Por otro lado, un análisis más exhaustivo a nivel de grupos de autores y temáticas desarrolladas por los mismos habría permitido obtener datos interesantes sobre la tendencia de los investigadores con mayor número de publicaciones en el periodo y el objeto de investigación bajo estudio. A esto se añadiría la posibilidad de establecer relaciones cruzadas entre las distintas variables observadas en los estudios analizados, con el objetivo de identificar el impacto que las mismas ejercen en la precisión y usabilidad del modelo.

Por último, otro de los factores limitantes se encuentra en el uso del repositorio ISBSG para extraer el estado actual del uso de las metodologías ágiles y los modelos de estimación de esfuerzo, principalmente debido al reducido número de proyectos ASD contenidos. El análisis de la nueva versión (ISBSG R1-2017) podrá arrojar datos más relevantes en relación al uso de metodologías de desarrollo ágil en la industria.

6.2. Futuras líneas de investigación

Dado que los autores de la SLR anterior han desarrollado recientemente una taxonomía para la clasificación y organización del conocimiento sobre modelos de estimación de esfuerzo en entornos de desarrollo ágil (Usman et al., 2017), un próximo paso sería analizar los estudios primarios encontrados en esta SLR utilizando dicha taxonomía. Asimismo, esta taxonomía podría ser útil para la clasificación de

los proyectos recuperados desde el repositorio ISBSG en cada una de las cuatro dimesiones consideradas (contexto de estimación, técnica de estimación, predictores de esfuerzo y valor de estimación).

Otro aspecto a considerar en el futuro es la influencia de los factores de coste respecto al nivel de precisión de los modelos. Es decir, sería un aporte interesante considerar cómo el uso de ciertos factores pueden mejorar la exactitud de las estimaciones y en qué contextos específicos pueden ser aplicados.

Por último, el presente estudio puede servir como paso previo para la propuesta de un nuevo modelo de estimación de esfuerzo basado en los datos históricos de ISBSG R1-2017. Para ello, sería necesario realizar un análisis pormenorizado de las variables relevantes y las relaciones entre ellas. Siendo este un repositorio *cross-company*, con datos sobre proyectos software desarrollados en diferentes países del mundo, las oportunidades de generalización y aplicabilidad de los resultados son considerables.

Apéndice 1: Estudios Primarios

[S1] Gandomani, T. J., Wei, K. T., y Binhamid, A. K. (2014). A Case Study Research on Software Cost Estimation Using Experts' Estimates, Wideband Delphi, and Planning Poker Technique. International Journal of Software Engineering and Its Applications, 8(11), 173-182.

[S2] Choetkiertikul, M., Dam, H. K., Tran, T., Pham, T., Ghose, A., y Menzies, T. (2016). A deep learning model for estimating story points. arXiv:1609.00489 [cs, stat]. Recuperado de http://arxiv.org/abs/1609.00489

[S3] Alostad, J. M., Abdullah, L. R. A., y Aali, L. S. (2017). A Fuzzy based Model for Effort Estimation in Scrum Projects. International Journal of Advanced Computer Science and Applications, 8(9), 270-277.

[S4] Karunakaran, E., y Sreenath, N. (2015). A method to effort estimation for XP projects in clients perspective. International Journal of Applied Engineering Research, 10(7), 18529-18550.

[S5] Manga, I., y Blamah, N. V. (2014). A particle Swarm Optimization-based Framework for Agile Software Effort Estimation. The International Journal Of Engineering And Science (IJES), 3(6), 30–36.

[S6] Huijgens, H., y Solingen, R. van. (2014). A Replicated Study on Correlating Agile Team Velocity Measured in Function and Story Points. En Proceedings of the 5th International Workshop on Emerging Trends in Software Metrics (pp. 30-36). New York, NY, USA: ACM. https://doi.org/10.1145/2593868.2593874

[S7] Popli, R., y Chauhan, N. (2014a). Agile estimation using people and project related factors. En 2014 International Conference on Computing for Sustainable Global Development (INDIACom) (pp. 564-569). https://doi.org/10.1109/IndiaCom.2014.6828023

[S8] Popli, Rashmi, y Chauhan, N. (2013). An Agile Software Estimation Technique based on Regression Testing Efforts. En 13th Annual International Software Testing Conference in India (pp. 04–05).

[S9] Basri, S., Kama, N., Sarkan, H. M., Adli, S., y Haneem, F. (2016). An Algorithmic-Based Change Effort Estimation Model for Software Development. En 2016 23rd Asia-Pacific Software Engineering Conference (APSEC) (pp. 177-184).

https://doi.org/10.1109/APSEC.2016.034

[**S10**] Arifin, H. H., Daengdej, J., y Khanh, N. T. (2017). An Empirical Study of Effort-Size and Effort-Time in Expert-Based Estimations. En 2017 8th International Workshop on Empirical Software Engineering in Practice (IWESEP) (pp. 35-40). https://doi.org/10.1109/IWESEP.2017.21

[**S11**] Malgonde, O., y Chari, K. (2014). An ensemble-based model for predicting agile software development effort. En Workshop on Information Technology and Systems. New Zealand: University of Auckland Business School. Recuperado de https://www.scopus.com/inward/record.uri?eid=2-s2.0-84924690220&partnerID=40&md5=c510a3adce368c51493cfde914f5fff4

[**S12**] Dhir, S., Kumar, D., y Singh, V. b. (2017). An estimation technique in agile archetype using story points and function point analysis. International Journal of Process Management and Benchmarking, 7(4), 518-539. https://doi.org/10.1504/IJPMB.2017.086933

[**S13**] Kumar C, S., Kumari, A., y Ramalingam, S. P. (2014). An Optimized Agile Estimation Plan Using Harmony Search Algorithm. International Journal of Engineering and Technology, 6, 1994-2001.

[**S14**] Ochodek, M. (2016). Approximation of COSMIC Functional Size of Scenario-Based Requirements in Agile Based on Syntactic Linguistic Features #x2014;A Replication Study. En 2016 Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement (IWSM-MENSURA) (pp. 201-211). https://doi.org/10.1109/IWSM-Mensura.2016.039

[**S15**] Dragicevic, S., Celar, S., y Turic, M. (2017). Bayesian network model for task effort estimation in agile software development. Journal of Systems and Software, 127(Supplement C), 109-119. https://doi.org/10.1016/j.jss.2017.01.027

[**S16**] Lenarduzzi, V., y Taibi, D. (2014). Can Functional Size Measures Improve Effort Estimation in SCRUM? En ICSEA-International Conference on Software Engineering and Advances. Nice, France.

[**S17**] Ungan, E., Çizmeli, N., y Demirörs, O. (2014). Comparison of Functional Size Based Estimation and Story Points, Based on Effort Estimation Effectiveness in SCRUM Projects. En 2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications (pp. 77-80). https://doi.org/10.1109/SEAA.2014.83

[**S18**] Popli, R., y Chauhan, N. (2014b). Cost and effort estimation in agile software development. En 2014 International Conference on Reliability Optimization and Information Technology (ICROIT) (pp. 57-61). https://doi.org/10.1109/ICROIT.2014.6798284

[**S19**] Moharreri, K., Sapre, A. V., Ramanathan, J., y Ramnath, R. (2016). Cost-Effective Supervised Learning Models for Software Effort Estimation

in Agile Environments. En 2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC) (Vol. 2, pp. 135-140). https://doi.org/10.1109/COMPSAC.2016.85

[S20] Salmanoglu, M., Hacaloglu, T., y Demirors, O. (2017). Effort Estimation for Agile Software Development: Comparative Case Studies Using COSMIC Functional Size Measurement and Story Points. En Proceedings of the 27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement (pp. 41-49). New York, NY, USA: ACM. https://doi.org/10.1145/3143434.3143450

[S21] Tanveer, B., Guzmán, L., y Engel, U. M. (2017). Effort estimation in agile software development: Case study and improvement framework. Journal of Software: Evolution and Process, 29(11), n/a-n/a. https://doi.org/10.1002/smr.1862

[S22] Owais, M., y Ramakishore, R. (2016). Effort, duration and cost estimation in agile software development. En 2016 Ninth International Conference on Contemporary Computing (IC3) (pp. 1-5). https://doi.org/10.1109/IC3.2016.7880216

[S23] Satapathy, Shashank Mouli, y Rath, S. K. (2017). Empirical assessment of machine learning models for agile software development effort estimation using story points. Innovations in Systems and Software Engineering, 13(2-3), 191-200. https://doi.org/10.1007/s11334-017-0288-z

[S24] Panda, A., Satapathy, S. M., y Rath, S. K. (2015). Empirical Validation of Neural Network Models for Agile Software Effort Estimation based on Story Points. Procedia Computer Science, 57(Supplement C), 772-781. https://doi.org/10.1016/j.procs.2015.07.474

[S25] Porru, S., Murgia, A., Demeyer, S., Marchesi, M., y Tonelli, R. (2016). Estimating Story Points from Issue Reports. En Proceedings of the The 12th International Conference on Predictive Models and Data Analytics in Software Engineering (pp. 2:1-2:10). New York, NY, USA: ACM. https://doi.org/10.1145/2972958.2972959

[S26] Torrecilla-Salinas, C. J., Sedeño, J., Escalona, M. J., y Mejías, M. (2015). Estimating, planning and managing Agile Web development projects under a value-based perspective. Information and Software Technology, 61(Supplement C), 124-144. https://doi.org/10.1016/j.infsof.2015.01.006

[S27] Popli, R., y Chauhan, N. (2014c). Estimation in agile environment using resistance factors. En 2014 International Conference on Information Systems and Computer Networks (ISCON) (pp. 60-65). https://doi.org/10.1109/ICISCON.2014.6965219

[S28] Grapenthin, S., Poggel, S., Book, M., y Gruhn, V. (2014). Facilitating Task Breakdown in Sprint Planning Meeting 2 with an Interaction Room: An Experience Report. En 2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications (pp. 1-8). https://doi.org/10.1109/SEAA.2014.71

[**S29**] Lenarduzzi, V., Lunesu, I., Matta, M., y Taibi, D. (2015). Functional Size Measures and Effort Estimation in Agile Development: A Replicated Study. En Agile Processes in Software Engineering and Extreme Programming (pp. 105-116). Springer, Cham. https://doi.org/10.1007/978-3-319-18612-2_9

[**S30**] Ahmed, A. R., Tayyab, M., Bhatti, S. N., Alzahrani, A. J., y Babar, M. I. (2017). Impact of Story Point Estimation on Product using Metrics in Scrum Development Process. International Journal of Advanced Computer Science and Applications, 8(4), 385-391.

[**S31**] Štrba, R., Štolfa, J., Štolfa, S., y Košinár, M. (2014). Intelligent software support of the SCRUM process, 272. https://doi.org/10.3233/978-1-61499-472-5-408

[**S32**] Zabkar, N., Hovelja, T., Urevc, J., y Mahnic, V. (2015). Introducing agile software development: lessons learned from the first scrum project in a Slovenian company. En International Conference on Advances in Management Engineering and Information Technology. https://doi.org/10.2495/AMEIT140781

[**S33**] Popli, R., y Chauahn, N. (2015). Managing uncertainty of story-points in Agile software. En 2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom) (pp. 1357-1361).

[**S34**] Adnan, M., y Afzal, M. (2017). Ontology based Multiagent Effort Estimation System for Scrum Agile Method. IEEE Access, PP(99), 1-1. https://doi.org/10.1109/ACCESS.2017.2771257

[**S35**] Taibi, D., Lenarduzzi, V., Diebold, P., y Lunesu, I. (2017). Operationalizing the Experience Factory for Effort Estimation in Agile Processes. En Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering (pp. 31-40). New York, NY, USA: ACM. https://doi.org/10.1145/3084226.3084240

[**S36**] Garg, S., y Gupta, D. (2015). PCA based cost estimation model for agile software development projects. En 2015 International Conference on Industrial Engineering and Operations Management (IEOM) (pp. 1-7). https://doi.org/10.1109/IEOM.2015.7228109

[**S37**] Basri, Sufyan, Kama, N., Haneem, F., y Adli, S. (2016). Predicting Effort for Requirement Changes During Software Development. En Proceedings of the Seventh Symposium on Information and Communication Technology (pp. 380-387). New York, NY, USA: ACM. https://doi.org/10.1145/3011077.3011096

[**S38**] Popli, R., Chauhan, N., y Sharma, H. (2014). Prioritising user stories in agile environment. En 2014 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT) (pp. 515-519). https://doi.org/10.1109/ICICICT.2014.6781336

[**S39**] Dumas-Monette, J. F., y Trudel, S. (2014). Requirements Engineering Quality Revealed through Functional Size Measurement: An Empirical Study in an Agile

Context. En 2014 Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement (pp. 222-232). https://doi.org/10.1109/IWSM.Mensura.2014.43

[**S40**] Aslam, W., Ijaz, F., Lali, M. I., y Mehmood, W. (2017). Risk Aware and Quality Enriched Effort Estimation for Mobile Applications in Distributed Agile Software Development. Journal of Information Science and Engineering, 33(6), 1481-1500. https://doi.org/10.6688/JISE.2017.33.6.6

[**S41**] Septian, W., y Gata, W. (2017). Software development framework on small team using Agile Framework for Small Projects (AFSP) with neural network estimation. En 2017 11th International Conference on Information Communication Technology and System (ICTS) (pp. 259-264). https://doi.org/10.1109/ICTS.2017.8265680

[**S42**] Satapathy, S.M., Panda, A., y Rath, S. K. (2014). Story point approach based agile software effort estimation using various SVR kernel methods. En Proceedings of the International Conference on Software Engineering and Knowledge Engineering, SEKE (Vol. 2014-January, pp. 304-307).

[**S43**] Grapenthin, S., Book, M., Richter, T., y Gruhn, V. (2016). Supporting Feature Estimation with Risk and Effort Annotations. En 2016 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA) (pp. 17-24). https://doi.org/10.1109/SEAA.2016.24

[**S44**] Tanveer, B., Guzmán, L., y Engel, U. M. (2016). Understanding and Improving Effort Estimation in Agile Software Development: An Industrial Case Study. En Proceedings of the International Conference on Software and Systems Process (pp. 41-50). New York, NY, USA: ACM. https://doi.org/10.1145/2904354.2904373

[**S45**] Khatri, S. K., Malhotra, S., y Johri, P. (2016). Use case point estimation technique in software development. En 2016 5th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO) (pp. 123-128). https://doi.org/10.1109/ICRITO.2016.7784938

[**S46**] Hamouda, A. E. D. (2014). Using Agile Story Points as an Estimation Technique in CMMI Organizations. En 2014 Agile Conference (pp. 16-23). https://doi.org/10.1109/AGILE.2014.11

Apéndice 2: Lista de abreviaturas

Abreviatura	Significado
ASD	Agile Software Development
COCOMO	Constructive Cost Model
DSDM	Dynamic Systems Development Method
FDD	Feature Driven Development
FP	Function Points
FPA	Function Point Analysis
FSM	Functional Size Measurement
ISBSG	International Software Benchmarking Standards Group
JAD	Joint Application Development
ML	Machine Learning
MMRE	Mean Magnitude of Relative Error
PM	Project Manager
PMBoK	Project Management Body of Knowledge
PMI	Project Management Institute
RAD	Rapid Application Development
RQ	Research Question
RUP	Rational Unified Process
SiFP	Simplified Function Points
SLOC	Source Lines of Code
SLR	Systematic Literature Review
SP	Story Points
TDD	Test Driven Development
UCP	Use Case Points
UML	Unified Modeling Language
WBS	Work Breakdown Structure
XP	eXtreme Programming
XU	eXtreme software size Unit

Referencias

Abrahamsson, P., Conboy, K., y Wang, X. (2009). "Lots done, more to do": The current state of agile systems development research. *European Journal of Information Systems*, *18*(4), 281–284. https://doi.org/10.1057/ejis.2009.27

Abrahamsson, P., Salo, O., Ronkainen, J., y Warsta, J. (2017). Agile software development methods: Review and analysis. *arXiv Preprint arXiv:1709.08439*.

Abrahamsson, P., Warsta, J., Siponen, M. T., y Ronkainen, J. (2003). New directions on agile methods: A comparative analysis. In *Proceedings of the 25th international conference on software engineering* (pp. 244–254). Washington, DC, USA: IEEE Computer Society. Obtenido de http://dl.acm.org/citation.cfm?id=776816.776846

Aljahdali, S., Sheta, A. F., y Debnath, N. C. (2015). Estimating software effort and function point using regression, support vector machine and artificial neural networks models. In *2015 IEEE/ACS 12th international conference of computer systems and applications (AICCSA)* (pp. 1–8). https://doi.org/10.1109/AICCSA.2015.7507149

Alostad, J. M., Abdullah, L. R. A., y Aali, L. S. (2017). A fuzzy based model for effort estimation in scrum projects. *International Journal of Advanced Computer Science and Applications*, *8*(9), 270–277.

Alsmadi, I. M., y Nuser, M. (2013). Evaluation of cost estimation metrics: Towards a unified terminology. *Journal of Computing and Information Technology*, *21*(1), 23–34.

Amancio, D. R., Comin, C. H., Casanova, D., Travieso, G., Bruno, O. M., Rodrigues, F. A., y Costa, L. da F. (2014). A systematic comparison of supervised classifiers. *PLoS ONE*, *9*(4), e94137. https://doi.org/10.1371/journal.pone.0094137

Ambler, S. (2005). The agile unified process (AUP). Obtenido July 6, 2018, de http://www.ambysoft.com/unifiedprocess/agileUP.html

Anandhi, V., y Chezian, R. M. (2014). Regression techniques in software effort estimation using COCOMO dataset. In *2014 international conference on intelligent computing applications* (pp. 353–357). https://doi.org/10.1109/ICICA.2014.

79

Anderson, D. (2012). Lean software development. Msdn.microsoft.com. Obtenido March 3, 2018, de https://msdn.microsoft.com/en-us/library/hh533841(v=vs.120).aspx

AXELOS. (2015). PRINCE2 agile PRINCE2. AXELOS. Obtenido March 19, 2018, de https://www.axelos.com/best-practice-solutions/prince2/prince2-agile

Azzeh, M., Neagu, D., y Cowling, P. (2008). Adjusting analogy software effort estimation based on fuzzy logic. In *ICSOFT 2008 - proceedings of the 3rd international conference on software and data technologies* (Vol. SE, pp. 127–132).

Barcelos Tronto, I. F. de, Simoes da Silva, J., y Sant'Anna, N. (2008). An investigation of artificial neural networks based prediction systems in software project management. *Journal of Systems and Software*, *81*(3), 356–367. https://doi.org/10.1016/j.jss.2007.05.011

Beck, K. (1999). Embracing change with extreme programming. *Computer*, *32*(10), 70–77. https://doi.org/10.1109/2.796139

Beck, K. (2000). *Extreme programming explained: Embrace change*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Benala, T. R., Dehuri, S., Satapathy, S. C., y Madhurakshara, S. (2012). Genetic algorithm for optimizing functional link artificial neural network based software cost estimation. In S. C. Satapathy, P. S. Avadhani, y A. Abraham (Eds.), *Proceedings of the international conference on information systems design and intelligent applications 2012 (INDIA 2012) held in visakhapatnam, india, january 2012* (Vol. 132, pp. 75–82). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-27443-5_9

Bilgaiyan, S., Mishra, S., y Das, M. (2016). A review of software cost estimation in agile software development using soft computing techniques. In *2016 2nd international conference on computational intelligence and networks (CINE)* (pp. 112–117). https://doi.org/10.1109/CINE.2016.27

Bilgaiyan, S., Sagnika, S., Mishra, S., y Das, M. (2017). A systematic review on software cost estimation in agile software development. *Journal of Engineering Science & Technology Review*, *10*(4).

Boehm, B. (1981). *Software engineering economics*. Pearson Education.

Boehm, B. (1986). A spiral model of software development and enhancement. *SIGSOFT Softw. Eng. Notes*, *11*(4), 14–24. https://doi.org/10.1145/12944.12948

Boehm, B., Abts, C., y Chulani, S. (2000). Software development cost estimation

approaches—A survey. *Annals of Software Engineering*, *10*(1), 177–205.

Boehm, B., Madachy, R., y Steece, B. (2000). *Software cost estimation with cocomo II with cdrom*. Prentice Hall PTR.

Boetticher, G., Menzies, T., y Ostrand, T. (2007). PROMISE repository of empirical software engineering data. *West Virginia University, Department of Computer Science*.

Borade, J. G., y Khalkar, V. R. (2013). Software project effort and cost estimation techniques. *International Journal of Advanced Research in Computer Science and Software Engineering*, *3*(8). Obtenido de https://pdfs.semanticscholar.org/40f3/691eec1ebd1b46f58ad4c1ff30ae0ab986db.pdf

Bossavit, L. (2012, October 15). A tour of the agile tribes. Agile alliance. Obtenido February 26, 2018, de https://www.agilealliance.org/a-tour-of-the-agile-tribes/

Brechner, E. (2015). *Agile project management with kanban*. Microsoft Press.

Britto, R., Mendes, E., y Börstler, J. (2015). An empirical investigation on effort estimation in agile global software development. In *2015 IEEE 10th international conference on global software engineering* (pp. 38–45). https://doi.org/10.1109/ICGSE.2015.10

Britto, R., Usman, M., y Mendes, E. (2014). Effort estimation in agile global software development context. In *Agile methods. large-scale development, refactoring, testing, and estimation* (pp. 182–192). Springer, Cham. https://doi.org/10.1007/978-3-319-14358-3_15

Bundschuh, M., y Dekkers, C. (2008). *The IT measurement compendium: Estimating and benchmarking success with functional size measurement*. Springer Science & Business Media.

Cheikhi, L., Abran, A., y Desharnais, J. M. (2012). Analysis of the ISBSG software repository from the ISO 9126 view of software product quality. In *IECON 2012 - 38th annual conference on IEEE industrial electronics society* (pp. 3086–3094). https://doi.org/10.1109/IECON.2012.6389405

Chemuturi, M. (2009). *Software estimation best practices, tools & techniques: A complete guide for software project estimators*. J. Ross Publishing.

Coad, P., Luca, J. de, y Lefebvre, E. (1999). *Java modeling color with uml: Enterprise components and process with cdrom* (1st ed.). Upper Saddle River, NJ, USA: Prentice Hall PTR.

Cobb, C. G. (2015). *The project manager's guide to mastering agile: Principles and practices for an adaptive approach*. John Wiley & Sons.

Cockburn, A. (2000). Selecting a project's methodology. *IEEE Softw.*, *17*(4), 64–71.

https://doi.org/10.1109/52.854070

Cockburn, A. (2002). *Agile software development* (Vol. 177). Addison-Wesley Boston. Obtenido de http://www.academia.edu/download/7332870/lec1d-short-agile-sd.ps

Cockburn, A. (2008). Using both incremental and iterative development. *STSC CrossTalk (USAF Software Technology Support Center)*, *21*(5), 27–30.

Coelho, E., y Basu, A. (2012). Effort estimation in agile software development using story points. *International Journal of Applied Information Systems (IJAIS)*, *3*(7).

Cohn, M. (2005). *Agile estimating and planning*. Pearson Education.

Conboy, K. (2009). Agility from first principles: Reconstructing the concept of agility in information systems development. *Information Systems Research*, *20*(3), 329–354. https://doi.org/10.1287/isre.1090.0236

Conte, S. D., Dunsmore, H. E., y Shen, V. Y. (1985). Software effort estimation and productivity. In M. C. Yovits (Ed.), *Advances in computers* (Vol. 24, pp. 1–60). Elsevier. https://doi.org/10.1016/S0065-2458(08)60364-2

Conte, S. D., Dunsmore, H. E., y Shen, V. Y. (1986). *Software engineering metrics and models*. Benjamin-Cummings Publishing Company.

Dejaeger, K., Verbeke, W., Martens, D., y Baesens, B. (2012). Data mining techniques for software effort estimation: A comparative study. *IEEE Transactions on Software Engineering*, *38*(2), 375–397. https://doi.org/10.1109/TSE.2011.55

Dingsoyr, T., Dybå, T., y Moe, N. B. (2010). *Agile software development current research and future directions*. Heidelberg [Germany]: Springer.

Dolan, S. (2007). Project management for agile software projects. In. Presented at the PMI® global congress 2007, North America, Atlanta, GA: Project Management Institute. Obtenido de https://www.pmi.org/learning/library/project-management-agile-software-projects-7242

Dragicevic, S., Celar, S., y Turic, M. (2017). Bayesian network model for task effort estimation in agile software development. *Journal of Systems and Software*, *127*(Supplement C), 109–119. https://doi.org/10.1016/j.jss.2017.01.027

Dybå, T., y Dingsøyr, T. (2008). Empirical studies of agile software development: A systematic review. *Information and Software Technology*, *50*(9), 833–859.

Erickson, J., Lyytinen, K., y Siau, K. (2005). Agile modeling, agile software development, and extreme programming: The state of research. *Journal of Database Management*, *16*(4), 88.

Fernández-Diego, M., y González-Ladrón-De-Guevara, F. (2014). Potential and

limitations of the ISBSG dataset in enhancing software engineering research: A mapping review. *Information and Software Technology*, *56*(6), 527–544.

Finnie, G. R., Wittig, G. E., y Desharnais, J.-M. (1997). A comparison of software effort estimation techniques: Using function points with neural networks, case-based reasoning and regression models. *Journal of Systems and Software*, *39*(3), 281–289. https://doi.org/10.1016/S0164-1212(97)00055-1

Fowler, M. (1997). Refactoring: Improving the design of existing code. In *11th european conference. jyväskylä, finland*.

Galorath, D. D., y Evans, M. W. (2006). *Software sizing, estimation, and risk management: When performance is measured performance improves*. CRC Press.

Gandomani, T. J., Wei, K. T., y Binhamid, A. K. (2014). A case study research on software cost estimation using experts' estimates, wideband delphi, and planning poker technique. *International Journal of Software Engineering and Its Applications*, *8*(11), 173–182. Obtenido de http://www.earticle.net/article.aspx?sn=235323

Gilb, T. (1985). Evolutionary delivery versus the "waterfall model". *SIGSOFT Softw. Eng. Notes*, *10*(3), 49–61. https://doi.org/10.1145/1012483.1012490

González-Ladrón-de-Guevara, F., Fernández-Diego, M., y Lokan, C. (2016). The usage of ISBSG data fields in software effort estimation: A systematic mapping study. *Journal of Systems and Software*, *113*, 188–215. https://doi.org/10.1016/j.jss.2015.11.040

Grenning, J. (2002). Planning poker or how to avoid analysis paralysis while release planning. *Hawthorn Woods: Renaissance Software Consulting*, *3*, 22–23.

Hamdy, A. (2012). Fuzzy logic for enhancing the sensitivity of COCOMO cost model. *Journal of Emerging Trends in Computing and Information Sciences*, *3*(9), 1292–1297.

Hamouda, A. E. D. (2014). Using agile story points as an estimation technique in CMMI organizations. In *2014 agile conference* (pp. 16–23). https://doi.org/10.1109/AGILE.2014.11

Hazzan, O., y Dubinsky, Y. (2014). The agile manifesto. In *Agile anywhere* (pp. 9–14). Springer, Cham. https://doi.org/10.1007/978-3-319-10157-6_3

Highsmith, J. (2000). *Adaptive software development: A collaborative approach to managing complex systems*. New York, NY, USA: Dorset House Publishing Co., Inc.

Highsmith, J. (2002). *Agile software development ecosystems*. Boston, MA, USA:

Addison-Wesley Longman Publishing Co., Inc.

Highsmith, J., y Cockburn, A. (2001). Agile software development: The business of innovation. *Computer*, *34*(9), 120–127. https://doi.org/10.1109/2.947100

Hoerl, A. E., y Kennard, R. W. (1970). Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, *12*(1), 55–67.

Huijgens, H., Bruntink, M., Deursen, A. van, Storm, T. van der, y Vogelezang, F. (2016). An exploratory study on functional size measurement based on code. In *Proceedings of the international conference on software and systems process* (pp. 56–65). New York, NY, USA: ACM. https://doi.org/10.1145/2904354.2904360

Humphrey, W. S. (1996). *Introduction to the personal software process (sm)*. Addison-Wesley Professional.

Hussain, I., Kosseim, L., y Ormandjieva, O. (2013). Approximation of COSMIC functional size to support early effort estimation in agile. *Data & Knowledge Engineering*, *85*, 2–14. https://doi.org/10.1016/j.datak.2012.06.005

Idri, A., Abnane, I., y Abran, A. (2015). Systematic mapping study of missing values techniques in software engineering data. In *16th IEEE/ACIS international conference on software engineering, artificial intelligence, networking and parallel/distributed computing (SNPD 2015)* (pp. 1–8). https://doi.org/10.1109/SNPD.2015.7176280

ISBSG. (2013). Repository data release 12—Field descriptions. International Software Benchmarking; Standards Group.

ISO/IEC. (2012). ISO/IEC 14143-6:2012(en). ISO/IEC 14143-6:2012(en) [Online Browsing Platform (OBP)]. Obtenido March 14, 2018, de https://www.iso.org/obp/ui/#iso:std:iso-iec:14143:-6:ed-2:v1:en

Jacobson, I., Spence, I., y Kerr, B. (2016). Use-case 2.0. *Communications of the ACM*, *59*(5), 61–69.

Jørgensen, M. (2004). Regression models of software development effort estimation accuracy and bias. *Empirical Software Engineering*, *9*(4), 297–314. https://doi.org/10.1023/B:EMSE.0000039881.57613.cb

Jørgensen, M. (2007). A critique of how we measure and interpret the accuracy of software development effort estimation. In *First international workshop on software productivity analysis and cost estimation*. Citeseer.

Jørgensen, M. (2014). What we do and don't know about software development effort estimation. *IEEE Software*, *31*(2), 37–40. https://doi.org/10.1109/MS.2014.49

Kahn, S. K. (2003). *Systematic reviews to support evidence-based medicine: How*

to review and apply findings of healthcare research. Royal Society of Medicine Press.

Karner, G. (1993). Resource estimation for objectory projects. In *Objective systems SF AB*.

Kassab, M. (2015). The changing landscape of requirements engineering practices over the past decade. In *2015 IEEE fifth international workshop on empirical requirements engineering (EmpiRE)* (pp. 1–8). https://doi.org/10.1109/EmpiRE.2015.7431299

Khatri, S. K., Malhotra, S., y Johri, P. (2016). Use case point estimation technique in software development. In *2016 5th international conference on reliability, infocom technologies and optimization (trends and future directions) (ICRITO)* (pp. 123–128). https://doi.org/10.1109/ICRITO.2016.7784938

Kitchenham, B., Budgen, D., y Brereton, P. (2015). *Evidence-based software engineering and systematic reviews*. Chapman & Hall/CRC.

Kitchenham, B., Pearl Brereton, O., Budgen, D., Turner, M., Bailey, J., y Linkman, S. (2009). Systematic literature reviews in software engineering – a systematic literature review. *Information and Software Technology*, *51*(1), 7–15. https://doi.org/10.1016/j.infsof.2008.09.009

Kitchenham, B., Pickard, L. M., MacDonell, S. G., y Shepperd, M. J. (2001). What accuracy statistics really measure [software estimation]. *IEE Proceedings - Software*, *148*(3), 81–85. https://doi.org/10.1049/ip-sen:20010506

Kitchenham, B., y Charters, S. (2007). *Guidelines for performing systematic literature reviews in software engineering*. Obtenido de http://www.dur.ac.uk/ebse/resources/Systematic-reviews-5-8.pdf

Kniberg, H., y Skarin, M. (2010). *Kanban and scrum - making the most of both*. Lulu.com.

Kotsiantis, S. B. (2007). Supervised machine learning: A review of classification techniques. In *Proceedings of the 2007 conference on emerging artificial intelligence applications in computer engineering: Real word AI systems with applications in eHealth, HCI, information retrieval and pervasive technologies* (pp. 3–24). Amsterdam, The Netherlands, The Netherlands: IOS Press. Obtenido de http://dl.acm.org/citation.cfm?id=1566770.1566773

Laird, L. M., y Brennan, M. C. (2006). *Software measurement and estimation: A practical approach*. John Wiley & Sons.

Larman, C. (2003). *Agile and iterative development: A manager's guide*. Pearson Education.

Larman, C., y Basili, V. R. (2003). Iterative and incremental development: A brief history. *Computer*, *36*(6), 47–56. https://doi.org/10.1109/MC.2003.

1204375

Lavazza, L., y Meli, R. (2014). An evaluation of simple function point as a replacement of IFPUG function point. In *2014 joint conference of the international workshop on software measurement and the international conference on software process and product measurement* (pp. 196–206). https://doi.org/10.1109/IWSM.Mensura.2014.28

Leung, H. K. N. (2002). Estimating maintenance effort by analogy. *Empirical Software Engineering*, *7*(2), 157–175. https://doi.org/10.1023/A:1015202115651

Li, J., Ruhe, G., Al-Emran, A., y Richter, M. M. (2007). A flexible method for software effort estimation by analogy. *Empirical Software Engineering*, *12*(1), 65–106. https://doi.org/10.1007/s10664-006-7552-4

Liebchen, G. A., y Shepperd, M. (2008). Data sets and data quality in software engineering. In *Proceedings of the 4th international workshop on predictor models in software engineering* (pp. 39–44). New York, NY, USA: ACM. https://doi.org/10.1145/1370788.1370799

Lokan, C., Wright, T., Hill, P., y Stringer, M. (2001). Organizational benchmarking using the ISBSG data repository. *IEEE Software*, *18*(5), 26–32. https://doi.org/10.1109/52.951491

Martin, J. (1991). *Rapid application development*. Macmillan Publishing Company.

Miller, A. (2002). *Subset selection in regression*. CRC Press.

Moløkken-Østvold, K., Haugen, N. C., y Benestad, H. C. (2008). Using planning poker for combining expert estimates in software projects. *Journal of Systems and Software*, *81*(12), 2106–2117. https://doi.org/10.1016/j.jss.2008.03.058

Monika, y Sangwan, O. P. (2017). Software effort estimation using machine learning techniques. In *2017 7th international conference on cloud computing, data science engineering - confluence* (pp. 92–98). https://doi.org/10.1109/CONFLUENCE.2017.7943130

Nelson, E. A. (1967). *Management handbook for the estimation of computer programming costs*. System Development Corporation (SDC), Santa Monica, California. Obtenido de http://www.dtic.mil/docs/citations/AD0648750

Ochodek, M. (2016). Approximation of COSMIC functional size of scenario-based requirements in agile based on syntactic linguistic features #x2014;A replication study. In *2016 joint conference of the international workshop on software measurement and the international conference on software process and product measurement (IWSM-MENSURA)* (pp. 201–211). https://doi.org/10.1109/IWSM-Mensura.2016.039

Oliveira, A. L. (2006). Estimation of software project effort with support

vector regression. *Neurocomputing*, *69*(13), 1749–1753. Obtenido de http://www.sciencedirect.com/science/article/pii/S0925231205004492

Palmer, S. R., y Felsing, J. M. (2002). *A practical guide to feature-driven development*. Prentice Hall PTR.

Papatheocharous, E., y Andreou, A. S. (2007). Software cost estimation using artificial neural networks with inputs selection. In J. Cardoso, J. Cordoso, y J. Filipe (Eds.) (pp. 398–407). Presented at the 9th international conference on enterprise information systems (ICEIS 2007), Funchal, PORTUGAL: Insticc-Inst Syst Technologies Information Control & Communication.

PMI. (2017). *Agile practice guide*. Project Management Institute.

Popli, R., y Chauhan, N. (2014). Cost and effort estimation in agile software development. In *2014 international conference on reliability optimization and information technology (ICROIT)* (pp. 57–61). https://doi.org/10.1109/ICROIT.2014.6798284

Poppendieck, M. B., y Poppendieck, T. (2003). *Lean software development: An agile toolkit*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Port, D., y Korte, M. (2008). Comparative studies of the model evaluation criterions mmre and pred in software cost estimation research. In *Proceedings of the second ACM-IEEE international symposium on empirical software engineering and measurement* (pp. 51–60). ACM. Obtenido de http://dl.acm.org/citation.cfm?id=1414015

Qumer, A., y Henderson-Sellers, B. (2008). An evaluation of the degree of agility in six agile methods and its applicability for method engineering. *Information and Software Technology*, *50*(4), 280–295. https://doi.org/10.1016/j.infsof.2007.02.002

Razzak, M. A. (2016). An empirical study on lean and agile methods in global software development. In *2016 IEEE 11th international conference on global software engineering workshops (ICGSEW)* (pp. 61–64). https://doi.org/10.1109/ICGSEW.2016.22

Reddy, P., Sudha, K. R., Sree, P. R., y Ramesh, S. (2010). Software effort estimation using radial basis and generalized regression neural networks. *Journal of Computing*, *2*(5), 87–92. Obtenido de https://arxiv.org/abs/1005.4021

Rijwani, P., y Jain, S. (2016). Enhanced software effort estimation using multi layered feed forward artificial neural network technique. *Procedia Computer Science*, *89*, 307–312. https://doi.org/10.1016/j.procs.2016.06.073

Runeson, P., y Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, *14*(2), 131. https:

//doi.org/10.1007/s10664-008-9102-8

Santillo, L., Conte, M., y Meli, R. (2005). Early amp; quick function point: Sizing more with less. In *Software metrics, 2005. 11th IEEE international symposium* (pp. 1–6). https://doi.org/10.1109/METRICS.2005.18

Saroha, M., y Sahu, S. (2015). Tools & methods for software effort estimation using use case points model — a review. In *Communication automation international conference on computing* (pp. 874–879). https://doi.org/10.1109/CCAA.2015.7148498

Schwaber, K. (1995). Scrum development process, OOPSLA'95 workshop on business object design and implementation. *Austin, USA*.

Schwaber, K., y Beedle, M. (2002). *Agile software development with scrum* (1st ed.). Pearson International Edition. Obtenido de http://sutlib2.sut.ac.th/sut_contents/H129174.pdf

Schweighofer, T., Kline, A., Pavlič, L., y Heričko, M. (2016). How is effort estimated in agile software development projects? In (Vol. 1677, pp. 73–80). Presented at the CEUR workshop proceedings.

Scott, K. (2002). *The unified process explained*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Sehra, S. K., Brar, Y. S., Kaur, N., y Sehra, S. S. (2017). Research patterns and trends in software effort estimation. *Information and Software Technology*. https://doi.org/10.1016/j.infsof.2017.06.002

Shepperd, M., y Schofield, C. (1997). Estimating software project effort using analogies. *IEEE Transactions on Software Engineering*, *23*(11), 736–743. https://doi.org/10.1109/32.637387

Shrivastava, S. V., y Date, H. (2010). Distributed agile software development: A review. Obtenido de https://arxiv.org/abs/1006.1955

Singh, B. K., y Misra, A. K. (2012). Software effort estimation by genetic algorithm tuned parameters of modified constructive cost model for nasa software projects. *International Journal of Computer Applications*, *59*(9).

Singh, J. (2017). *Functional software size measurement methodology with effort estimation and performance indication*. John Wiley & Sons.

Sommerville, I. (2010). *Software engineering* (9th ed.). USA: Addison-Wesley Publishing Company.

Sommerville, I. (2011). *Software engineering*. Pearson Education.

Srinivasan, K., y Fisher, D. (1995). Machine learning approaches to estimating software development effort. *IEEE Transactions on Software Engineering*, *21*(2), 126–137.

https://doi.org/10.1109/32.345828

Stapleton, J. (1997). *Dynamic systems development method-the method in practice*. Addison Wesley Reading.

Stellman, A., y Greene, J. (2005). *Applied software project management*. "O'Reilly Media, Inc."

Sutherland, J. (2013, May 16). Story points: Why are they better than hours? Scrum inc. Obtenido March 14, 2018, de https://www.scruminc.com/story-points-why-are-they-better-than/

Sweet, M., y Moynihan, R. (2007). Improving population health: The uses of systematic reviews.

Tang, J., Zhang, G., Wang, Y., Wang, H., y Liu, F. (2015). A hybrid approach to integrate fuzzy c-means based imputation method with genetic algorithm for missing traffic volume data estimation. *Transportation Research Part C: Emerging Technologies*, *51*, 29–40. https://doi.org/10.1016/j.trc.2014.11.003

Taylor, K. J. (2016). Adopting agile software development: The project manager experience. *Information Technology & People*, *29*(4), 670–687. https://doi.org/10.1108/ITP-02-2014-0031

Teorey, T. J., Buxton, S., Fryman, L., Güting, R. H., Halpin, T., Harrington, J. L., ... Witt, G. (2008). *Database design: Know it all*. Morgan Kaufmann.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 267–288.

Top, O., Ozkan, B., Nabi, M., y Demirors, O. (2011). Internal and external software benchmark repository utilization for effort estimation. In *Joint conference of the 21st international workshop on software measurement and the 6th international conference on software process and product measurement (IWSM-MENSURA 2011)* (pp. 302–307). https://doi.org/10.1109/IWSM-MENSURA.2011.41

Trendowicz, A., Münch, J., y Jeffery, R. (2011). State of the practice in software effort estimation: A survey and literature review. In *Third IFIP TC 2 central and east european conference on software engineering techniques* (pp. 232–245). Berlin, Heidelberg: Springer-Verlag. Obtenido de http://dl.acm.org/citation.cfm?id=2040660.2040685

Trendowicz, A., y Jeffery, R. (2014). *Software project effort estimation: Foundations and best practice guidelines for success*. Springer International Publishing. Obtenido de //www.springer.com/gp/book/9783319036281

Tripp, S. D., y Bichelmeyer, B. (1990). Rapid prototyping: An alternative instructional design strategy. *Educational Technology Research and Development*, *38*(1),

31–44.

Tsui, F., Karam, O., y Bernal, B. (2016). *Essentials of software engineering*. Jones & Bartlett Learning.

Usman, M., Börstler, J., y Petersen, K. (2017). An effort estimation taxonomy for agile software development. *International Journal of Software Engineering and Knowledge Engineering*, *27*(4), 641–674. https://doi.org/10.1142/S0218194017500243

Usman, M., Mendes, E., Weidt, F., y Britto, R. (2014). Effort estimation in agile software development: A systematic literature review. In *Proceedings of the 10th international conference on predictive models in software engineering* (pp. 82–91). New York, NY, USA: ACM. https://doi.org/10.1145/2639490.2639503

Usman, M., Mendes, E., y Börstler, J. (2015). Effort estimation in agile software development: A survey on the state of the practice. In *Proceedings of the 19th international conference on evaluation and assessment in software engineering* (pp. 12:1–12:10). New York, NY, USA: ACM. https://doi.org/10.1145/2745802.2745813

Usman, M., y Britto, R. (2016). Effort estimation in co-located and globally distributed agile software development: A comparative study. In *2016 joint conference of the international workshop on software measurement and the international conference on software process and product measurement (IWSM-MENSURA)* (pp. 219–224). https://doi.org/10.1109/IWSM-Mensura.2016.042

VersionOne. (2018). *State of agile report* (No. 12). Obtenido de https://explore.versionone.com/state-of-agile/versionone-12th-annual-state-of-agile-report

Webb, C. (2016, March 7). Navigating the agile landscape deloitte australia. Blog deloitte australia. Obtenido February 26, 2018, de http://blog.deloitte.com.au/navigating-the-agile-landscape/

Wen, J., Li, S., Lin, Z., Hu, Y., y Huang, C. (2012). Systematic literature review of machine learning based software development effort estimation models. *Information and Software Technology*, *54*(1), 41–59. https://doi.org/10.1016/j.infsof.2011.09.002

Williams, L., y Cockburn, A. (2003). Agile software development: It's about feedback and change. *Computer*, *36*(6), 39–43. https://doi.org/10.1109/MC.2003.1204373

Witten, I. H., Frank, E., Hall, M. A., y Pal, C. J. (2016). *Data mining: Practical machine learning tools and techniques*. Morgan Kaufmann. Obtenido de https://books.google.es/books?hl=en&lr=&id=1SylCgAAQBAJ&oi=fnd&pg=PP1&dq=%22Data+Mining:+Practical+machine+learning+tools+and+

techniques%22&ots=8HINxdoDCg&sig=qZG2FpTnFGmvXwsGfWQr0j7kc3Y

Wohlin, C. (2014). Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th international conference on evaluation and assessment in software engineering* (pp. 38:1–38:10). New York, NY, USA: ACM. https://doi.org/10.1145/2601248.2601268

Wood, J., y Silver, D. (1995). *Joint application development*. Wiley.