Tipo de artículo: Artículo de revisión

Análisis de la información generada para mantener la escalabilidad y persistencia del proceso de desarrollo de software

Analysis of the information generated to maintain the scalability and persistence of the software development process

Patricia María Marcillo Sánchez [1*], **https://orcid.org/0000-0003-1421-1004**
Lugio David Román Barrezueta [2], **https://orcid.org/0000-0002-3081-8052**

[1] Universidad de Guayaquil, Ecuador. Correo electrónico: patricia.marcillos@ug.edu.ec
[2] Universidad de Guayaquil, Ecuador. Correo electrónico: lugio.romanb@ug.edu.ec

*** Autor para correspondencia: patricia.marcillos@ug.edu.ec**

Resumen

Los proyectos de desarrollo de software, especialmente cuando son a gran escala, incorporan una variedad de artefactos que se desarrollan, usan, mantienen, evolucionan y que dependen unos de otros. En la presente investigación se realiza una revisión sistemática de la literatura para identificar cuáles son los artefactos ágiles más generados por los desarrolladores, y las herramientas automatizadas que se emplean para garantizar la integración y escalabilidad en el desarrollo de sistemas de software. La investigación realizada confirma que los equipos ágiles a gran escala utilizan una gran variedad de artefactos tanto de procesos ágiles como convencionales. Los artefactos ágiles más desarrollados son: historias de usuarios, solicitudes de nuevas funciones, diseño detallado, criterios de prueba, pruebas unitarias, código fuente, estimaciones de puntos de historia, cuadro de incendio, tableros de estado, pila de sprint y pila de producto. Es consenso general en la literatura revisada, que la trazabilidad e integración entre diferentes artefactos ayuda a reducir el tiempo y el costo de desarrollo y mantenimiento, mejorando así la calidad del sistema.

Palabras clave: artefactos ágiles; integración continua; trazabilidad; desarrollo ágil a gran escala.

Abstract

Software development projects, especially when they are large-scale, incorporate a variety of artifacts that are developed, used, maintained, evolved, and depend on each other. In the present investigation, a systematic review of the literature is carried out to identify which are the agile artifacts most generated by developers, and the automated tools that are used to guarantee integration and scalability in the development of software systems. Research confirms that large-scale agile teams use a wide variety of artifacts from both agile and conventional processes. The most developed agile artifacts are: user stories, new feature requests, detailed design, test criteria, unit tests, source code, story point estimates, fire chart, status boards, sprint backlog, and product backlog. . It is a general consensus in the reviewed literature that traceability and integration between different artifacts help to reduce the time and cost of development and maintenance, thus improving the quality of the system.

Keywords: *agile artifacts; continuous integration; traceability; large-scale agile development.*

Recibido: 25/06/2022
Aceptado: 09/08/2022
En línea: 19/08/2022

Introducción

El desarrollo de software se ha comportado con un crecimiento exponencial sostenido por lo que constantemente surgen pequeñas y grandes empresas que han apostado a este revolucionario campo y por tanto han redefinido los paradigmas de producción creando sistemas de software cada vez más complejos, orientados hacia el intercambio de información entre sistemas y la reutilización de componentes que permita el desarrollo en un menor tiempo y esfuerzo (Ajayi et al., 2019). Estos adelantos se centran fundamentalmente en la consolidación de la experiencia alcanzada, en la robustez de las soluciones desarrolladas y en las estrategias de reutilización y de infraestructura tecnológicas definidas (Kessel & Atkinson, 2018).

Las Metodologías de Desarrollo de Software evolucionan como una respuesta al vertiginoso crecimiento de la industria informática. Los métodos de desarrollo de software han sido objeto de estudio por diferentes autores (Castro et al., 2014; Harvie & Agah, 2016; Lunesu et al., 2021; Moyo & Mnkandla, 2020) y constituyen objetivos obligatorios del programa de formación de varias instituciones y programas académicos.

Existen varios métodos para el proceso de desarrollo software. Los métodos están conformados por etapas que son generales a todos los enfoques (Rashid et al., 2021). Las diferencias están básicamente en los tiempos en los cuales se realizan dichas etapas, la simultaneidad, la prioridad que se le da a cada etapa, la cantidad de información generada, entre otros elementos. De la selección certera y oportuna de estos métodos de desarrollo depende en gran medida el éxito de los proyectos de software que se emprenden.

La necesidad de definir un entorno tecnológico robusto es una prioridad de todas las organizaciones dedicadas al desarrollo de productos y servicios de software, así como de las Instituciones de la Educación Superior (IES), que tienen la responsabilidad de formar profesionales en el campo de la ingeniería de software, y que además, crean sus propias soluciones para fortalecer el proceso de enseñanza aprendizaje. Debido al tamaño y complejidad crecientes de los sistemas de software, la coordinación del trabajo, de los artefactos y de los desarrolladores se ha convertido en un desafío.

Si se analiza sistemáticamente cada uno de los Reportes *CHAOS del Standish Group*, se puede observar cómo la cantidad de proyectos de software no exitosos mantienen estabilidad en un rango elevado de entre el 44% y el 53% (Johnson, 2018) . Uno de los principales detonantes de estas estadísticas es el incremento del tamaño y la complejidad

de los productos de software, cuando el núcleo tecnológico que soporta estas grandes soluciones aún no ha alcanzado los niveles de madurez necesarios (Li et al., 2014).

Los proyectos de desarrollo de software, especialmente cuando son a gran escala, incorporan una variedad de artefactos que se desarrollan, usan, mantienen, evolucionan y que dependen unos de otros. El producto de software final como tal constituye un artefacto, así como los modelos del producto de software, sus especificaciones, certificados de seguridad y protección, planes de gestión de proyectos, manuales de usuario, el código y muchos más. En los proyectos de software, estos artefactos generalmente se manifiestan en forma de documentos impresos o electrónicos en el sentido de colecciones de información estructurada que sirve para distintos propósitos (Lucas et al., 2020).

Los artefactos juegan un papel vital en el desarrollo de software y sistemas. Sin embargo, lo que realmente es un artefacto y cómo está estructurado, no está definido de manera única. De acuerdo con (Méndez Fernández et al., 2019), un artefacto es un producto de trabajo producido, modificado o utilizado por una secuencia de tareas que tienen valor para un rol. Los artefactos están sujetos a garantía de calidad y control de versiones y tienen tipos específicos. Están estructurados jerárquicamente en elementos de contenido que definen áreas únicas de responsabilidad y que son el resultado de una sola tarea. Un artefacto es una unidad almacenable individualmente y con un nombre único que cumple un propósito específico en el contexto de un proceso de desarrollo.

Con la creación de artefactos, los miembros de los equipos de producción de software logran capturar la evolución del conocimiento de los desarrolladores sobre los elementos del proyecto de software, y de esta manera logran mantener la persistencia y escalabilidad de los productos de software (Alam, 2019). Los artefactos de software, también referenciados como productos de trabajo, representan no solo el conocimiento que los desarrolladores tienen sobre los elementos del sistema, sino que también capturan la información necesaria para lograr que este conocimiento prevalezca con el tiempo, disminuyendo las posibilidades de olvido, pérdida de información informal, retrabajo, e inconsistencias en las políticas de control de cambio. Incorporar estas garantías de escalabilidad y evolución es imprescindible tanto en el Desarrollo de Software Tradicional como en el Desarrollo Ágil de Software (Keshta & Morgan, 2017).

El Desarrollo Ágil de Software (ASD, por sus siglas en inglés) tiene como objetivo entregar continuamente software valioso a los clientes. Kieran Conboy define la agilidad como la preparación continua de un método de desarrollo ágil de software para crear un cambio de forma rápida o inherente; adoptar el cambio de forma proactiva o reactiva; y aprender del cambio mientras contribuye al valor percibido por el cliente (economía, calidad y simplicidad), a través de sus componentes colectivos y relaciones con su entorno (Conboy, 2009). La producción de software normalmente

implica la creación y gestión de numerosos artefactos de desarrollo. Si bien uno de los valores principales del manifiesto ágil (Beck et al., 2001) indica: "*Software funcionando por encima de la documentación*", los profesionales relacionados con el desarrollo ágil de software, aunque no es su fuerte producir gran cantidad de artefactos, reconocen su importancia en la reutilización, escalabilidad y mantenimiento del software.

Algunos de los artefactos más referenciados por la literatura del desarrollo ágil de software son: los planes de proyecto, documentos de requisitos, planes de prueba, documentos de diseño, código fuente, registros de inspección de código, información de construcción y registros de defectos o no conformidades (Patkar et al., 2022). Así mismo, existen diferentes medios de colaboración entre las partes interesadas de un sistema de software. Uno de esos medios es la colaboración basada en artefactos (Hillemacher et al., 2021).

Aunque lo métodos de desarrollo ágil (Anderson, 2010; Cockburn, 2001; Larman & Vodde, 2017) generalmente definen y proponen los artefactos de software que deben ser creado por cada una de las fases o etapas del ciclo de vida del desarrollo de software, en la implementación de entornos ágiles de desarrollo, los equipos de proyecto suelen eliminar la creación de algunos, e incorporar nuevos artefactos que consideran necesarios para el proyecto, independientemente del método que hayan seleccionado para el desarrollo ágil. Estos elementos fueron identificados en el estudio realizado por Wińska y Dąbrowski en 2020 sobre los artefactos de desarrollo de software en grandes organizaciones ágiles (Wińska & Dąbrowski, 2020).

Identificar los artefactos de desarrollo de software que garantizan la escalabilidad y persistencia del proceso de desarrollo de software, no es solo de interés de las organizaciones que se dedican al desarrollo de software como actividad profesional, sino que es indispensable para incluir dentro de las actividades formativas de las Instituciones de la Educación Superior (IES) para graduar profesionales con cultura de creación, mantenimiento y evolución de los sistemas de software. Siendo así, se define como objetivo de la presente investigación analizar la información generada en metodologías ágiles de software para mantener la escalabilidad y persistencia del proceso de desarrollo de software.

Materiales y métodos

Para lograr el resultado deseado de esta investigación, se realizó una búsqueda de estudios empíricos y documentos de diseño e implementación de artefactos de desarrollo de software generados en entornos de desarrollo ágiles para mantener la escalabilidad y persistencia del proceso de desarrollo de software. Se analizaron con énfasis artículos originales y de revisiones recientes, para abordar preguntas sobre cuáles son los artefactos que deberían crearse durante el ciclo de vida del desarrollo ágil de software. Se tomaron en cuenta las siguientes consideraciones:

- Los artículos de investigación, capítulos de libros, conferencias y artículos originales y de revisión son los principales materiales de origen para preparar esta investigación.
- Se realizaron búsquedas bibliográficas en las bases de datos IEEE Xplore, Web of Science, SpringerLink, Google Scholar y ACM Digital library.
- Se priorizaron artículos de los últimos 5 años (2018 a 2022) para rastrear los estudios más recientes. También se incluyeron trabajos fuera de los últimos 5 años si la información provista no tiene un límite de tiempo.
- La búsqueda bibliográfica automatizada se realizó empleando las siguientes palabras clave: *Software Engineering Artefacts*, *Syntax of artefacts*, *Semantics of artefacts*, *Consistency of artifacts*, *Artefacts in Software Engineering*, *development artifacts*.

Los recursos repetidos que aparecen en varias bases de datos se eliminaron durante una primera línea de selección. Se realizó una lectura completa del resumen/introducción y una inspección de los recursos. La principal exclusión en esta etapa se debe a los registros que no están relacionados con desarrollo de artefactos de software, o que profundizan en métodos más allá del desarrollo ágil de software. Este proceso fue realizado por los dos autores de forma independiente antes de presentar un registro de inclusiones y exclusiones provisionales. La lista de exclusiones acordadas por unanimidad no fue considerada para esta investigación.

Los materiales identificados para incluir como referencias en esta investigación se examinaron con el fin de identificar los artefactos más comunes en el desarrollo ágil de software, así como identificar las herramientas para la trazabilidad e integración continua de artefactos. Las preguntas de la investigación definidas, fueron las siguientes:

1. ¿Cuáles son los métodos de desarrollo ágiles más utilizados por las organizaciones de desarrollo de software?
2. ¿Cuáles son los artefactos ágiles más generados por los desarrolladores?
3. ¿Qué herramientas automatizadas se emplean para garantizar la integración y escalabilidad en el desarrollo de sistemas de software?

El propósito de la investigación es abordar el desafío de la persistencia y escalabilidad de los artefactos de software en la perspectiva amplia de la organización que es posible dentro de los marcos ágiles para escalar las organizaciones de desarrollo que trabajan de acuerdo con la cultura ágil y sus resultados materializados como artefactos. Los métodos ágiles que se han elegido para la investigación son: Scaled Agile Framework (SAFe) (Leffingwell et al., 2016), Large

Scale Scrum (LeSS) (Larman & Vodde, 2017), Scrum (Schwaber & Beedle, 2002) y eXtreme Programming (XP) (Beck, 1999).

El objetivo de esta selección es discutir la comparación de los métodos ágiles desde una perspectiva de entregables. Durante la investigación, los autores se han dado cuenta de que no existe un enfoque estricto de los artefactos en los artículos científicos. El proceso de investigación se ha dividido en:

- Especificación de la lista de métodos ágiles que se tendrán en cuenta durante el proceso de investigación.
- Investigación y revisión de publicaciones existentes relacionadas con el desarrollo ágil de software y la creación de artefactos definidos en los métodos ágiles seleccionados.
- Análisis de los sistemas de trazabilidad, integración y mejora continua de artefactos.

Resultados y discusión

Métodos y marcos de desarrollo ágil de software

Las organizaciones de desarrollo de software han utilizado marcos o métodos ágiles para trabajar colectivamente para desarrollar y entregar una versión estable de software. Los últimos 10 años han visto el surgimiento de una serie de métodos de desarrollo ágil que se han implementado en diversas organizaciones de desarrollo de software como Cisco (Power, 2016), Nokia Networks (Larman & Vodde, 2017), Intel (Conboy & Carroll, 2019), Ericsson (Conboy & Carroll, 2019), Dell (Conboy & Carroll, 2019). Algunos de los de métodos de desarrollo ágil más populares incluyen:

Tabla 1. Métodos y marcos de desarrollo ágil.

ID	Denominación	Autor	Referencia
XP	eXtreme Programming	Kent Beck	(Beck, 1999)
DSDM	Dynamic Systems Development Method	Jennifer Stapleton	(Stapleton, 1997)
Scrum	Scrum	Ken Schwaber, Jeff Sutherland	(Schwaber & Beedle, 2002)
Crystal	Crystal	Alistair Cockburn	(Cockburn, 2001)
AM	Agile Modeling	Scott Ambler	(Ambler, 2002)
FDD	Feature Driven Design	Peter Coad y Jeff de Luca	(Benoit et al., 1999)
LSD	Lean Software Development	Mary Poppendieck, Tom Poppendieck	(Poppendieck & Poppendieck, 2003)
Kanban	Kanban (development)	David J. Anderson	(Anderson, 2010)
SAFe:	Scaled Agile Framework	Dean Leffingwell	(Leffingwell et al., 2016)
LeSS:	Large Scale Scrum	Craig Larman, Bas Vodde	(Larman & Vodde, 2017)

Los métodos que se han elegido para la investigación son: Scaled Agile Framework (SAFe) (Leffingwell et al., 2016), Large Scale Scrum (LeSS) (Larman & Vodde, 2017), Scrum (Schwaber & Beedle, 2002) y eXtreme Programming (XP) (Beck, 1999).

Scaled Agile Framework® (SAFe®)

Los métodos de desarrollo ágil se han vuelto muy populares en las organizaciones de software. Los principios y prácticas del desarrollo ágil se diseñaron originalmente para equipos pequeños y ubicados en el mismo lugar. Para aprovechar los beneficios potenciales también en empresas más grandes, las prácticas ágiles deben escalarse. Para admitir el escalado, se han propuestos nuevos marcos como Scaled Agile Framework (SAFe). Aunque SAFe es un marco para escalar ágil a grandes empresas, varias reportes indicaron que se estaban alejando del desarrollo ágil. Así mismo, este desafío está respaldado por los argumentos de varios autores como Ken Schwaber (Schwaber & Beedle, 2002), Ron Jeffries (Jeffries, 2014) y Stephen Denning (Denning, 2018) los cuales han analizados problema de SAFe en los enfoques centralizados y la planificación fundamentalmente, y afirman que implementa mecanismos de la de la burocracia jerárquica que resulta improductivo. Teniendo en cuenta estos hallazgos, en la presente investigación se decidió analizar los principales artefactos que se generan dentro de este marco. Como resultado del análisis y síntesis de datos se identificaron las categorías de beneficios y desafíos de SAFe más destacadas, las cuales fueron:

Beneficios

- Transparencia
- Alineación
- Productividad
- Previsibilidad
- Tiempo de comercialización

Desafíos

- Resistencia al cambio
- Planificación del primer incremento del programa
- Alejamiento de ágil.

Large Scale Scrum (LeSS)

LeSS es un marco para la gestión del desarrollo de software ágil, que muestra cómo implementar Agile y Scrum a escala; es un Scrum aplicado a muchos equipos que trabajan juntos en un solo producto (Larman & Vodde, 2017). Aunque LeSS continúa el proceso de reinventar fundamentalmente la gestión mediante la incorporación de las lecciones de experiencia ganadas en la ampliación de los métodos de gestión de Agile y Scrum, aún está deliberadamente incompleto, dado que no ofrece respuestas definitivas, formuladas o de enfoques aparentemente seguros y disciplinados que ofrecen una guía de control predecible (Conboy & Carroll, 2019). LeSS se enfoca en la esencia mínima requerida al escalar, incluida la atención continua a la excelencia técnica y una mentalidad de experimentación continua. LeSS

no es un proceso o una técnica para construir productos, es un marco dentro del cual se pueden adaptar procesos y técnicas para satisfacer las necesidades de la situación particular (Almeida & Espinheira, 2022).

Extreme Programming (XP)

La Programación Extrema (XP) ha evolucionado a partir de los problemas causados por los largos ciclos de desarrollo de los modelos de desarrollo tradicionales. Inicialmente comenzó como una manera simple de hacer el trabajo con prácticas que se habían encontrado efectivas en los procesos de desarrollo de software durante las décadas anteriores (Borman et al., 2020). Después de varios ensayos exitosos en la práctica, la metodología XP fue formalizada sobre los principios clave y las prácticas utilizadas. Si bien las prácticas individuales de XP no son nuevas como tales, en XP se han recopilado y alineado para que funcionen entre sí de una manera novedosa, formando así una nueva metodología para el desarrollo de software. XP tiene como objetivo permitir el desarrollo de software exitoso a pesar de los requisitos vagos o en constante cambio en equipos de tamaño pequeño a mediano (Shrivastava et al., 2021). Las iteraciones cortas con versiones pequeñas y comentarios rápidos, la participación del cliente, la comunicación y la coordinación, la integración y las pruebas continuas, la propiedad colectiva del código, la documentación limitada y la programación en pareja se encuentran entre las principales características de XP.

Scrum

Es una de las metodologías de desarrollo ágil de Software más reconocidas a nivel mundial, en la cual se resalta el trabajo en equipo para el desarrollo de productos y la autonomía que estos deben tener. SCRUM ayuda a solventar estos riesgos involucrando al cliente en el proceso de desarrollo. El cliente, en conjunto con el equipo de desarrollo, es quién define qué se hace y cuándo se hace. El equipo es el que se compromete con qué puede entregar en la duración del sprint -ciclo de desarrollo- e involucra al cliente en la inspección. Scrum hace parte integral de la inspección al cliente, permitiéndole realizar revisiones tempranas de los desarrollos y una revisión general del proceso, con el fin de que él mismo, con sus ideas, aporte al proceso, ayude a mejorar la sinergia del equipo y a que efectivamente se entregue lo que él espera (Srivastava et al., 2017). El concepto de artefactos en Scrum representa el trabajo o valor que conduce a una provisión más fácil de transparencia. Además, los artefactos brindan oportunidades para la inspección y la adaptación, siendo estos dos conceptos fundamentales en los modelos de gestión empírica.

Con una fuerte colaboración entre los equipos de desarrollo (*Development Teams*) y los propietarios de productos (*Product Owners*) y con frecuentes ciclos de retroalimentación, Scrum tiende a producir artefactos de alta calidad. Además, la naturaleza de inspección y adaptación de Scrum fomenta que los equipos busquen continuamente mejores

formas de trabajar. Con la reducción del tiempo de comercialización y los artefactos de mayor calidad, también aumenta la satisfacción del cliente. Además, el refinamiento frecuente de la cartera de productos (*Program Backlog*) y la repriorización permiten responder más rápidamente a los cambios, lo que conduce a una mejor satisfacción de los clientes (Matthies et al., 2016).

La selección de estos cuatro métodos se debe a que estos marcos son comúnmente reconocidos en la industria de desarrollo de software. La revisión de la literatura científica identificó que existen estudios de casos, trabajos de investigación y publicaciones científicas que permiten una comparación precisa.

Artefactos de desarrollo de software

Los involucrados en el desarrollo de software entienden que la profesión tiene un fuerte componente de gestión de la información. La información se puede definir como el conjunto de datos que, transformados o modificados, tienen un valor para aquellos usuarios que hacen uso de ellos. Cualquier sistema de software exitoso, requiere la creación y gestión de muchos artefactos: requisitos, diseños, informes de no conformidades y código fuente con documentación integrada, por nombrar solo algunos. Para realizar el trabajo en el sistema, un desarrollador de software a menudo debe leer y comprender los artefactos asociados con el desarrollo del sistema (Behutiye et al., 2022).

Un artefacto es un resultado de trabajo autónomo, que tiene un propósito específico del contexto y constituye una representación física, una estructura sintáctica y un contenido semántico. El flujo de trabajo de desarrollo de software ágil está centrado en las personas y la mayor parte del tiempo se lleva a cabo en colaboración. Por lo general, este flujo de trabajo sugiere que los desarrolladores sigan rutinas diarias que involucran la ejecución de un conjunto de tareas, la interacción con otros desarrolladores y la manipulación de artefactos como el código fuente, la documentación y los archivos de configuración (Böhmer et al., 2017). Como resultado, los desarrolladores interactúan de forma continua y rutinaria entre sí, y con artefactos, surgiendo las interacciones Desarrollador-Desarrollador y las interacciones Desarrolladores-Artefactos, respectivamente. Las interacciones Desarrollador-Desarrollador ocurren cara a cara o a través de herramientas de comunicación, cuando se trabaja en un entorno distribuido. Las interacciones Desarrollador-Artefacto ocurren cuando un desarrollador interpreta, actualiza o crea un artefacto (Elamin & Osman, 2017).

Un artefacto es un producto del desarrollo de software que ayuda a describir la arquitectura, el diseño y la función del software. Los artefactos son como hojas de ruta (*roadmaps*) que los desarrolladores pueden usar para rastrear todo el proceso de desarrollo de software. Los artefactos pueden ser bases de datos, modelos de datos, documentos impresos,

diagramas, estudios de casos, modelos de datos, documentos de diseño, scripts y lenguajes de modelado unificados (UML). Su creación y evolución ayudan al mantenimiento y la actualización del software, ya que los desarrolladores pueden usarlos como material de referencia para ayudar a resolver problemas (Gonen & Sawant, 2020). Los artefactos se documentan y almacenan en un repositorio para que los desarrolladores de software puedan recuperarlos cuando lo soliciten; pueden incluir lo siguiente:

- Imágenes: Estas imágenes de diseño o de referencia ayudan a desarrollar el software.
- Conjunto de caracteres (*charset*)
- Traducción (i18n): se refiere a la capacidad de internacionalización y localización, centrado en la traducción.
- Interfaz de usuario (*UI*)
- Multimedias
- Prototipos: Esta versión completamente funcional del software ayuda a los desarrolladores a crear una versión funcional básica de su proyecto.
- Documentos de desarrollo (*Devel-doc*): Estos artefactos realizan un seguimiento de los documentos relevantes, incluidos diagramas, acuerdos de usuario final, documentación interna o guías escritas. Describen las características y atributos del software.
- Diagramas: Estos ayudan a los desarrolladores a trazar la estructura del software.
- Notas de la reunión (*Meeting notes*): Estas son opciones de diseño y aspectos escritos como transcripciones completas o parciales de las reuniones.
- Código Fuente: Este es el sistema diseñado y fundamental que permite que el código funcione. Este código actúa como base para el software y permite al desarrollador probar el software antes de ejecutarlo. Los artefactos de código pueden incluir código compilado, scripts de configuración, conjuntos de pruebas, objetos generados y registros generados durante las pruebas y el control de calidad.

Artefactos creados por equipos de desarrollo ágiles

La revisión de la literatura científica permitió identificar los artefactos más comunes que se cran durante el desarrollo ágil de software. El estándar IEEE 1074 (IEEE, 2006) especifica los Procesos para el Desarrollo del Ciclo de Vida del Software, en particular, el estándar distingue las siguientes fases en los Procesos Orientados al Desarrollo (Panizzi et al., 2016): Procesos de Pre-Desarrollo, los Procesos de Desarrollo y los Procesos de Post-Desarrollo del software. La Tabla 1 muestra el conjunto de artefactos identificados, clasificados para cada uno de estos procesos.

Tabla 1. Artefactos del desarrollo ágil de software para cada Proceso Orientado al Desarrollo.

Artefactos		
Pre-Desarrollo	**Desarrollo**	**Post-Desarrollo**
Análisis de mercado	Análisis de impacto	Guía de implementación
Caso de negocio	Caso de prueba	Liberación (*Release*)
Contactos	Criterios de prueba	Lista de verificación de liberación
Contrato	Cuadro de incendio (*Burndown chart*)	Manual de usuario
Evaluación de riesgos	Código fuente	Nota de liberación (*Release note*)
Hoja de ruta (*Roadmap*)	Criterios de aceptación (*Acceptance criteria*)	Registro de liberación (*Release log*)
Legislación/Regulación	Caso de uso	Script de implementación
Normas ISO	Diseño técnico	(*Deployment script*)
Portafolio de productos	Documentación técnica	
Plan de negocios	Definición de hecho (*Definition of done*)	
Plan de despliegue	Definición de listo (*Definition of ready*)	
Plazo	Diseño funcional	
Requisitos	Descripción funcional de la plantilla	
Requisitos del mercado	Estándar de arquitectura	
Requisitos del producto	Estándar de codificación	
Solicitud de Información	Historias de usuario	
Visión/Estrategia	Hoja de ruta técnica (*Technical roadmap*)	
	Informe de no conformidades	
	Informe retrospectivo (*Retrospective report*)	
	Informe de prueba	
	Informe de prueba de unidad	
	Incremento	
	Pila de Producto (*Product backlog*)	
	Pila de sprint (*Sprint backlog*)	
	Prueba de aceptación del usuario	
	Plan de prueba	
	Requerimientos técnicos	
	Tablero de estado (*Status board*)	
	Tareas (*Task*)	

La Tabla 1 mostró un conjunto de artefactos identificados en la literatura científica, independientemente del método o marco ágil que se halla empleado en el desarrollo. Los artefactos más comunes que han sido reportados, los cuales se muestran en la Figura 1.

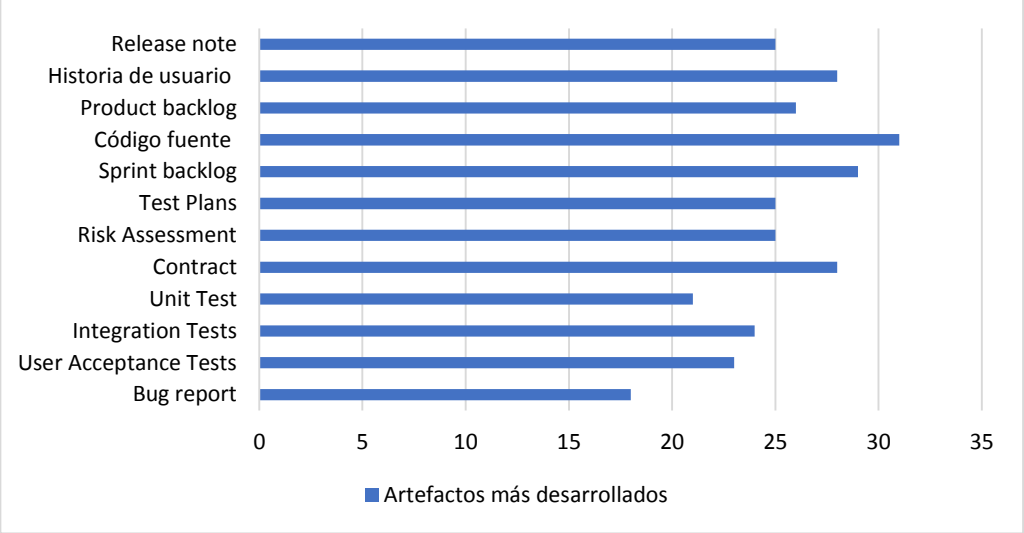

Figura 1. Artefactos más comunes identificados en la investigación.

El estudio realizado permitió clasificar los artefactos generados para mantener la escalabilidad y persistencia del proceso de desarrollo de software, según el enfoque para el cual resulten más valiosos, a saber: Seguimiento externo, Comunicación interna, Garantía de calidad, Gestión y Agilidad. La tabla 2 muestra esta clasificación.

Tabla 2. Artefactos clasificados según el enfoque.

Seguimiento externo	Comunicación interna	Garantía de calidad	Gestión	Agilidad
– Guía de implementación – Guía de implementación – Liberar – Lista de verificación de lanzamiento – Registro de liberación – Nota de lanzamiento – Manual de usuario	– Análisis de impacto – Bosquejo – Diseño técnico – Documentación técnica – Requerimiento técnico – Hoja de ruta técnica – Caso de uso	– No conformidades – Caso de prueba – Criterios de prueba – Plan de prueba – Informe de prueba – Informe de prueba de unidad – Prueba de aceptación del usuario	– Criterios de aceptación – Estándar de arquitectura – Estándar de codificación – Descripción funcional de la plantilla	– Cuadro de incendio – Épico – Pila de Producto – Informe retrospectivo – acumulación de sprint – Tablero de estado – Tarea – Historia del usuario – Definición de hecho – Definición de listo – Código fuente

La tabla 3 muestra el rol responsable de generar cada uno de los artefactos generados, y la fuente de información que utiliza como entrada para la creación del artefacto de salida.

Tabla 3. Identificación del rol responsable de generar artefactos y las fuentes de información que utiliza.

Rol responsable	Fuentes de información	Artefacto
– Propietario del producto proxy	– Patrocinador de productos	– Plan de despliegue
– Propietario del producto proxy	– Patrocinador de productos	– Historias de usuarios
– Product Owner	– Tendencias tecnológicas, Pila de Producto	– Arquitectura de referencia
– Product Owner	– Tendencias tecnológicas	– Estándar de arquitectura
– Propietario del producto proxy	– Contrato	– Evaluación de riesgos
– Patrocinador de productos	– Pila de Producto, Cliente	– Contrato
– Propietario del producto proxy	– Cliente	– Criterios de prueba
– Scrum Master	– Funciones binarias, pruebas unitarias	– Cuadro de incendio
– Scrum Master	– Funciones binarias, pruebas unitarias	– Tablero de estado
– Scrum Master	– Pila de Producto	– Pila de Sprint
– Equipo de desarrollo	– Archivos binarios liberados	– Archivos binarios
– Probador (Tester)	– Pila de Producto, , Criterios de prueba, Implementación de Arquitectura	– Test Plans
– Probador (*Tester*)	– Sprint Binario, Criterios de prueba	– Pruebas de integración
– Probador (*Tester*)	– Sprint Binario, Criterios de prueba	– Pruebas de regresión
– Equipo de desarrollo	– Sprint Binario	– Liberación binaria
– Equipo de desarrollo	– Código fuente	– Sprint Binario
– Patrocinador de productos	– Mercado, Clientes Existentes, Clientes Potenciales	– Pila de Producto (Product backlog)
– Probador (*Tester*)	– Historia de usuario, Criterios de prueba	– Pruebas unitarias
– Probador (*Tester*)	– Pruebas unitarias, Pruebas de integración Pruebas de regresión, Pruebas de aceptación del usuario (UAT)	– No conformidades
– Equipo de desarrollo	– Historia de usuario, Implementación de Arquitectura	– Estimación de Historia de usuario
– Equipo de desarrollo	– Historia de usuario, Implementación de Arquitectura	– Código fuente
– Equipo de desarrollo	– Diseño Detallado, Historia de usuario	– Código fuente
– Equipo de desarrollo	– Código fuente	– Funciones binarias

Diferentes partes interesadas, como gerentes de productos, gerentes de proyectos, analistas comerciales, desarrolladores y evaluadores, están involucradas en el desarrollo de sistemas de software y en la creación de artefactos para mantener la escalabilidad y persistencia del proceso de desarrollo de software. Los gerentes de productos investigan, seleccionan y desarrollan productos para una organización. Los directores de proyecto son asignados por la organización ejecutora para lograr los objetivos del proyecto. Los analistas de negocios o los ingenieros de sistemas desarrollan los requisitos del proyecto, los desarrolladores implementan los requisitos del sistema y los probadores (*tester*s) verifican estos

requisitos. El Product Owner, tiene como principal función la de priorizar aquellos elementos que tienen más valor en cada etapa y detallarlos para que el equipo de desarrollo sea capaz de valorarlos y ejecutarlos (Hess et al., 2017).

Descripción de artefactos

Los autores de este estudio identificaron un conjunto de artefactos que son recurrentes independientemente del método ágil adoptado. Estos artefactos se describen brevemente a continuación, clasificados según fases en los Procesos Orientados al Desarrollo, definidos en el Estándar IEEE 1074:

Descripción de artefactos de Pre-Desarrollo

- **Contratos:** Los contratos se realizan entre los proveedores de subcontratación y los clientes. Son siempre delicados y a veces se crea un conflicto abierto para su aceptación. La complejidad y la duración de los programas en el desarrollo ágil, significan que el cambio es inevitable. En los reportes técnicos sobre desarrollo ágil, no es común encontrar un proceso de licitación en el que el entregable haya sido lo que se solicitó inicialmente (Imran & Soomro, 2022). Los dos principales tipos de contratos en el desarrollo ágil son: precio fijo; y tiempo y materiales (T&M). Ambos modelos de contrato se pueden utilizar para adaptarse a los cambios durante el contrato. Un contrato de T&M permite la facturación continua por el trabajo completado, mientras que en un contrato de precio fijo se pacta el precio final del producto.
- **Evaluación de riesgos:** La evaluación de riesgos se utiliza para identificar fuentes potenciales de riesgo y estimar la probabilidad de ocurrencia. Para cada riesgo identificado, se enumeran y describen las acciones de mitigación. La evaluación de riesgos normalmente no se asocia con métodos de desarrollo ágiles, no es realmente algo específico de Agile. Es para cualquier tipo de proyecto, este artefacto incluye los riesgos potenciales y las fallas de un componente de software, lo que puede ayudar a los nuevos desarrolladores a determinar los riesgos potenciales y las soluciones a los problemas, a partir de su revisión (Lunesu et al., 2021). Mientras que los riesgos del producto pueden revisarse al final de cada sprint, los riesgos del cliente se revisarán con menos frecuencia, tal vez trimestralmente. Los riesgos operativos y de entrega se revisan solo después de hitos importantes, como la entrega del producto. El diseño arquitectónico se utiliza para mitigar cualquier complejidad técnica identificada durante el proceso de evaluación de riesgos (Leite, 2017).
- **Hojas de ruta (*Roadmap*):** Las hojas de ruta del producto estimulan la reflexión sobre las evoluciones inmediatas y futuras del producto. Este artefacto proporciona al equipo una dirección clara en la que deben desarrollar el producto, de acuerdo con la dirección y brindan información importante a las partes interesadas (*stakeholders*) sobre los planes futuros.

- **Plan de despliegue (*Release plan*):** En el desarrollo ágil de software, un plan de lanzamiento es un diagrama de flujo en evolución que describe qué características se entregarán.

Descripción de artefactos de Desarrollo

- **Historias de usuario:** Impulsan el proceso de desarrollo del código fuente. Se desarrollan durante el análisis de requisitos por parte de los propietarios del producto (*Product Owner*). Se necesitan diseños detallados para cumplir con los requisitos complejos aprobados antes de pasar a desarrollo. Una historia de usuario comprende elementos de interfaz de usuario, funcionalidad, lógica de negocio y almacenamiento de datos (Li et al., 2021).
- **Código fuente:** El código fuente generalmente sigue un enfoque de desarrollo basado en funciones. Las funciones se utilizan para fomentar la cohesión de las funciones estrechamente relacionadas y la independencia de las funciones no relacionadas, para evitar el acoplamiento indeseable, no debe haber interdependencia entre dos funciones diferentes. Las funciones están estrechamente relacionadas con la arquitectura del proyecto.
- **Criterios de prueba *(test criteria)*** se definen para cada historia de usuario y, a veces, los criterios de prueba se registran en tarjetas de historias físicas. En algunos equipos los criterios de aceptación son colocados en la propia historia de ususario.
- **Informe de No conformidades:** Se refiere a los defectos y mejoras de las funciones. En los programas de desarrollo de software a menudo hay muchos más problemas y errores conocidos de los que se pueden solucionar con el tiempo y los recursos disponibles. En consecuencia, los problemas se priorizan según la gravedad y el posible impacto en los usuarios finales. Luego, las no conformidades seleccionadas se solucionan dentro de los equipos de desarrollo y las soluciones se prueban posteriormente. Existen herramientas de software automatizadas para gestionar estas no conformidades como JUnit (Acharya, 2014), PhpUnit (Zandstra, 2016), NUnit (Puri-Jobi, 2015), Sonar (Arapidis, 2012), Selenium (Ramya et al., 2017), FitNesse (Lenka et al., 2018), PHPDepend (Warnars et al., 2017), PHPCodesniffer (Eddy et al., 2017) entre otras.
- **Pruebas unitarias:** se llevan a cabo para mejorar la calidad del software producido por un equipo. Luego, el código de función se puede cargar en un repositorio de integración, solo cuando se haya logrado que todos los puntos lógicos en una historia se han cumplido, y se hayan ejecutado casos de prueba para asegurarse de que todos estén bien. Después se debe verificar nuevamente la historia, y es entonces que se pasa al desarrollo estable del repositorio, para luego integrar.
- **Pila de Producto (*Product Backlog*):** es un inventario que contiene todo tipo de trabajo que haya que hacer en el desarrollo ágil del producto, ya sea requerimientos, casos de uso, tareas y dependencias. Es la principal fuente

de información sobre el producto. Aunque los desarrolladores que implementan otros marcos ágiles, también lo usan, específicamente en Scrum, se implementa como una lista en cualquier formato, que contiene todos los requerimientos que se necesitan para implementar en el producto y refleja el estado real del trabajo pendiente de implementación en el producto, así como el ya realizado. Es una lista ordenada, que contiene todo lo que podría ser necesario en el producto y es la única fuente de requisitos para hacer cambios en el producto. Los ítems que aparecen en el *Product Backlog* tienen los siguientes atributos: Descripción, Ordenación, Estimación y Valor.

- **Estándar de arquitectura:** se utilizan para garantizar que los sistemas de software estén bien estructurados, sean coherentes internamente, sean fáciles de entender, fáciles de mantener y satisfagan los requisitos no funcionales. Por lo tanto, las mejoras propuestas a los sistemas existentes se verifican y están sujetas a aprobación para garantizar el cumplimiento de la estrategia técnica corporativa. En este contexto el proceso de gestión de cambios es muy importante. Se desaconseja realizar cambios no autorizados en los sistemas ya que puede haber ciertas implicaciones graves, si no existe una gobernanza. Las decisiones arquitectónicas deben registrarse en un formulario que pueda difundirse entre los miembros del equipo (Matthies et al., 2019).
- **Implementación de arquitectura**: La implementación de la arquitectura es considerada como un diseño de alto nivel. Los cambios en la implementación de la arquitectura pueden surgir de nuevos requisitos funcionales. Los cambios en la implementación de la arquitectura precipitados por un nuevo requisito pueden afectar el trabajo de varios equipos de desarrollo y, por lo tanto, están sujetos a escrutinio fuera de cualquier equipo específico. En muchos equipos ágiles, la implementación de la arquitectura se realiza antes de las iteraciones de desarrollo, al mismo tiempo que la planificación del lanzamiento.
- **Plan de prueba**: Los planes de prueba describen el enfoque general de las pruebas para el programa de desarrollo. Articula los recursos necesarios para las diferentes etapas de la prueba. El cronograma para realizar los diferentes tipos de pruebas se describe en el plan, que es particularmente importante cuando se requiere personal de un equipo de pruebas de aceptación de usuarios de terceros, como es más común en los programas de desarrollo ágil a gran escala que en los proyectos más pequeños. El plan de prueba también describe el despliegue de personal, ya sea centralizado en un equipo de prueba dedicado o distribuido en equipos de desarrollo multifuncionales.
- **Pruebas de regresión:** Las pruebas de regresión funcional se utilizan para confirmar el comportamiento correcto del software implementado, después de agregar nuevas funciones. Se realiza para asegurarse de que la nueva función no afecte las funciones anteriores, y lógicamente, que estas nuevas funciones también se ejecutan

correctamente. Aunque este es un proceso automatizado, ejecutar una prueba de regresión completa puede demorar mucho tiempo, esta es una razón importante por la cual el código no se entrega a los clientes al final de cada iteración en los programas de desarrollo a gran escala. El tiempo que demora en ejecutarse una prueba de regresión es una de las principales debilidades, ya que en el desarrollo ágil, muchos equipos eligen no ejecutar una iteración de regresión al final de cada sprint.

- **Pruebas de integración**: Las pruebas de integración generalmente se usan para identificar problemas con los datos o controlar los flujos a través del sistema de software. Con estas pruebas se valida que lo que se está creando es correcto, y genera una visibilidad temprana de los problemas. De esta manera se comprueba que los productos creados por cada uno de los equipos de desarrollo son compatibles e integrables, y funcionan correctamente como un único producto.
- **Pruebas de Aceptación del Usuario (UAT):** Se realizan después de las pruebas de integración y regresión. Generalmente, después de dos o tres sprints, se realizan las pruebas de lanzamiento de UAT durante dos semanas aproximadamente. En ocasiones, las UAT se llevan a cabo dentro del equipo de desarrollo, lo que implica que todos los desarrolladores y el control de calidad ejecutan los scripts de prueba automatizados, aunque existen muchos equipos que prefieren las UAT sean realizadas por un equipo externo, que represente a los clientes, de esta manera, una vez que el código es liberado por el equipo de desarrollo, el equipo de UAT lo recoge y comienza a hacer sus pruebas.
- **Planes de lanzamiento:** Los proyectos grandes requieren un plan de lanzamiento para programar la entrega del producto, a menudo coordinado con eventos externos, como campañas publicitarias en televisión o medios impresos. Generalmente, un lanzamiento consistirá en una serie de iteraciones, que a menudo involucran a varios equipos. Se prepara una estrategia para cada lanzamiento, en base a las nuevas funcionalidades que se han implementado y se propone una estrategia que dependerá de cuánto tiempo requiere un equipo para hacer el lanzamiento. Para garantizar lanzamientos de productos de calidad y minimizar cualquier insatisfacción del cliente que surja de lanzamientos, en las empresas de desarrollo de software, los planes de lanzamiento, los códigos binarios de integración y los candidatos de lanzamiento, se producen antes del lanzamiento de cualquier producto a los clientes.
- **Pila de Sprint:** Este artefacto permite visualizar en cada Sprint, aquellos elementos que aún no han empezado a desarrollarse, aquellos que ya comenzaron su implementación y quiénes están trabajando en los mismos. También gestiona y actualiza la información sobre los productos que están esperando a desplegarse, o aquellos que están completamente terminados. Este artefacto es un elemento para visualizar el trabajo a realizar durante

cada Sprint y está gestionado por el equipo de desarrollo. Analizando este artefacto se puede entender cuál es la evolución del trabajo durante el Sprint, así como hacer un análisis de riesgos. Dado que cada Sprint tiene una meta específica, el Sprint Backlog permite analizar hasta donde se ha cumplido el objetivo, y qué se podría eliminar, para maximizar el retorno de la inversión en desarrollo. De manera general, se trata de una lista de elementos en los cuales trabajar durante la etapa de Sprint. Estos elementos normalmente se componen de tareas técnicas más pequeñas que permiten conseguir un incremento de software terminado.

- **Incremento:** El incremento ocurre como resultado del Sprint. Constituye la suma de todas las tareas, casos de uso, historias de usuario y cualquier elemento que se haya desarrollado durante el Sprint, y que será puesto a disposición del usuario final en forma de software, aportando un valor de negocio al producto que se está desarrollando. Gestionar adecuadamente la información del incremento, permite la obtención de productos estables que pueden ser reutilizados posteriormente.
- **Estimaciones de historias de usuario:** En el desarrollo ágil, los puntos de la historia de usuario se usan a menudo para estimar el trabajo. Se asigna un punto de la historia estimada para cada característica. Una ventaja de usar puntos de historia es que se puede generar un *Burndown chart* durante cada sprint y se puede calcular una velocidad para el equipo al final de la iteración. Sin embargo, otro enfoque común para la estimación se conoce como T-Shirt sizing, que divide las estimaciones de esfuerzo en categorías amplias de pequeño, mediano y grande, aunque este enfoque se considera menos preciso. La estimación de la tarea realizada por los miembros del equipo luego se comunica a los gerentes, sin embargo, la práctica ha demostrado que en una serie de programas de desarrollo, los gerentes realizan estimaciones, en negociación con los clientes, y las presentan a los equipos.
- **Cuadro de incendio (*Burndown chart*):** Este artefacto se utiliza para registrar y difundir la finalización de las historias de usuario; es propio del desarrollo con Scrum. Generalmente el *Burndown chart* se actualiza diariamente.
- **Tablero de estado (*Status board*):** Es un tablero físico que se utiliza para difundir el progreso de las funciones a lo largo del proceso de desarrollo. Las tarjetas sobre el panel indican cómo va la iteración actual. Las historias de usuario pueden estar representadas en este tablero en diferentes etapas de desarrollo. Así ocurre en el tablero de Kanban, en el que se definen columnas de estado para producto de trabajo. La principal debilidad de este artefacto es que para los equipos de desarrollo distribuidos geográficamente, un tablero físico es menos útil. Para difundir información a los miembros dispersos del equipo, se crea un tablero de estado virtual para representar en línea.

- **Definición de hecho (*Definition of done, DoD*)**: La *DoD* es un documento que define qué se considera hecho en un equipo agile. La idea es establecer una serie de criterios comunes para especificar cuando un ítem está completamente terminado y que aplique a todos los ítems que forman parte del incremento.
- **Definición de listo (*Definition of ready, DoR*)**: El DoR es un documento que define cuándo un requerimiento (historia de usuario o similar) se considera listo para que el equipo de desarrollo pueda entenderlo, valorarlo e incluirlo en un Sprint Planning con idea de acometerlo en un Sprint.
- **Cuadro de incendio (*Burndown Chart*):** El Burndown Chart es un gráfico de trabajo pendiente a lo largo del tiempo que muestra la velocidad a la que se están completando los objetivos, requisitos, o historias de usuarios. Permite extrapolar si el equipo podrá completar el trabajo en el tiempo estimado.

Descripción de artefactos de Post-Desarrollo

La justificación para el uso de artefactos posteriores al desarrollo es algo diferente de las otras categorías, se trata del enfoque de seguimiento externo del producto. Su uso no es solo una decisión de un equipo ágil. Estos artefactos son especialmente apreciados por partes externas, fuera del equipo ágil. Puede ser un cliente externo a la organización u otra parte de la organización. Sin embargo, el equipo ciertamente está involucrado con ellos, especialmente como su productor.

- **Manual de usuario**: Las partes interesadas externas e internas requieren el manual de usuario para conocer las nuevas características del producto y cómo funcionan.
- **Release**: Un lanzamiento es la distribución de la versión final o la versión más nueva de una aplicación de software. Un lanzamiento de software puede ser público o privado y generalmente significa el lanzamiento de una versión nueva o mejorada de la aplicación. En el desarrollo de software ágil, una versión es un paquete de software implementable que culmina en varias iteraciones y se puede realizar antes del final de una iteración.

Todos los artefactos posteriores al desarrollo, el script de implementación, la guía de implementación, la versión, la lista de verificación de la versión, el registro de la versión, la nota de la versión y el manual del usuario se encabezan bajo el fundamento de seguimiento externo del producto. Una vez analizados y descritos los principales artefactos que son generados durante el desarrollo ágil de software para mantener la escalabilidad y persistencia del proceso de desarrollo de software, se realiza una comparación de los artefactos comunes que implementan los cuatro métodos ágiles objetos de estudio en esta investigación: Scaled Agile Framework (SAFe), Large Scale Scrum (LeSS), Scrum y eXtreme Programming (XP).

Tabla 4. Artefactos comunes que implementan los métodos ágiles estudiados.

Artefacto	Método ágil			
	(SAFe)	(LeSS)	Scrum	XP
Product Backlog	x	x	x	
Sprint Backlog	x	x	x	
Incremento		x	x	
Program Increment	x			
Roadmap	x			
Impediment Backlog	x	x	x	
Historias de Usuario	x	x	x	x
Tarjetas de historias (*story cards*)				x
Arquitectura del sistema	x	x	x	x
Plan de la iteración				x
Código fuente	x	x	x	x
Documentos del diseño	x	x	x	x
Estimaciones de historias	x	x	x	x
Informe de no conformidades	x	x	x	x
Estándar de codificación				x
Documento Visión			x	
Manual de usuario		x	x	
Caso de uso			x	

Artefactos definidos por Scrum

Durante la revisión de la literatura, los autores identificaron un considerable número de trabajos que reportan la implementación de Scrum en los procesos de desarrollo de software en los últimos cinco años (Briatore & Golkar, 2021; Friess, 2019; Masood et al., 2022; Rodr et al., 2021), es por tal motivo que se decidió, revisar brevemente la información generada específicamente por esta metodología de desarrollo ágil de software. En Scrum, se denomina artefacto a aquellos elementos físicos que se producen como resultado de la implementación de Scrum. Los tres principales artefactos generados son: el Product Backlog, Sprint Backlog y el Incremento, sin embargo existen otros artefactos que aunque sean poco referenciados en la literatura, si son muy implementados dentro de los equipos de desarrollo.

Los artefactos definidos por Scrum están diseñados específicamente para maximizar la transparencia de la información clave que se necesita para el éxito del producto. También se pone un fuerte énfasis en la comprensión amplia y común de la organización de cada artefacto. A medida que la complejidad de los esfuerzos de entrega crece con el crecimiento de la organización, los modelos de gobernanza que incorporan artefactos que capturan los resultados del trabajo que se entrega, ya que reflejan ideas de transparencia, inspección y adaptación, juegan un papel principal. Las decisiones para optimizar el valor y controlar el riesgo se hacen con base en estos artefactos, por lo tanto si no son lo suficientemente transparentes se puede incurrir en decisiones erróneas.

Muchos artefactos de software se crean, mantienen y evolucionan como parte de un proyecto de desarrollo de software. A medida que los desarrolladores de software trabajan en un proyecto, interactúan con los artefactos del proyecto existente y realizan actividades como leer informes de errores archivados previamente en busca de informes duplicados. Estas actividades a menudo requieren que un desarrollador examine una cantidad sustancial de texto. La tabla 5 muestra la asignación de artefactos a Scrum que fue identificada durante la revisión de la literatura.

Tabla 5. Asignación de artefactos seleccionados a Scrum.

Programa de desarrollo	Sprint	Lanzamiento del producto
– Plan de prueba – Plan de despliegue – Estándar de arquitectura – Evaluación de riesgos – Contrato – Definición de hecho – Definición de listo	– Pila de Sprint – Estimación de Historia de usuario – Cuadro de incendio – Sprint binarios – Tablero de estado – Historias de usuario – Diseño detallado – Código fuente – Criterios de prueba – Prueba unitaria – No conformidades	– Pila de Producto – Pruebas de aceptación del usuario – Arquitectura de referencia – Implementación de arquitectura – Productos binarios – Diseño de alto nivel – Pruebas de integración – Pruebas de regresión – Liberación de archivos binarios

En los programas de desarrollo a gran escala, los niveles más altos de complejidad provocan un mayor número de dependencias técnicas entre los elementos de la cartera de productos. Los maestros de Scrum ayudan al propietario del producto a identificar las dependencias técnicas entre los elementos de la cartera de pedidos y ayudan a priorizar los elementos, sin embargo, los miembros de los equipos de desarrollo no suelen tener la oportunidad de influir en los elementos de la cartera de productos. En este contexto, las diversas comunidades de partes interesadas en el desarrollo ágil, aumentan la probabilidad de requisitos conflictivos. Esto ocurre porque los propietarios de productos crean y revisan continuamente la cartera de productos durante el programa de desarrollo de software, y los maestros de Scrum brindan asesoramiento técnico sobre las dependencias entre funciones.

Herramientas automatizadas para garantizar la integración y escalabilidad

Sistema de trazabilidad de artefactos

La trazabilidad basada en artefactos, se puede definir como una forma de colaboración asíncrona en la que diferentes partes interesadas crean o utilizan artefactos en diferentes puntos del ciclo de vida del desarrollo de software. Hay dos tipos de colaboraciones entre las partes interesadas de un sistema de software. Uno es la colaboración sincrónica, que

incluye reuniones de grupo y otros enfoques típicos de trabajo en equipo (Perera et al., 2015). La otra es la colaboración asíncrona, en la que, por ejemplo, un programador trabaja con un analista para implementar una función determinada. No se encuentran y es posible que ni siquiera se comuniquen entre sí directamente. La colaboración basada en artefactos se logra mediante el intercambio y la coautoría colaborativa de artefactos comunes de un proyecto. La colaboración basada en artefactos se puede mejorar a través de un sistema de trazabilidad que vincula explícitamente los artefactos del sistema (Jeong et al., 2018).

Un sistema de trazabilidad vincula los artefactos de un proyecto y, por lo tanto, brinda soporte para la colaboración entre las partes interesadas. Diferentes partes interesadas están interesadas en diferentes tipos de enlaces de trazabilidad entre los requisitos, el diseño, el código, las inspecciones de código, las construcciones, los defectos y los artefactos de prueba. Un sistema de trazabilidad se compone de un espacio de artefactos vinculados en el que las partes interesadas atraviesan vínculos entre artefactos. Los enlaces en el sistema de rastreabilidad se utilizan para colaboraciones basadas en artefactos y se denominan enlaces de rastreabilidad (Mustafa & Labiche, 2015).

Los documentos de requisitos especifican, por ejemplo, todos los requisitos que un sistema en desarrollo debe cumplir durante su vida útil, mientras que las arquitecturas de sistemas escritas en el Lenguaje de Modelado de Sistemas (SysML) (Kapos et al., 2021) permiten a los arquitectos de sistemas describir la arquitectura lógica y técnica del sistema. Si el comportamiento esperado de un sistema y sus componentes también se modela en SysML, los casos de prueba que son generados automáticamente, se puede utilizar para verificar que el sistema cumpla con estos requisitos del sistema. En consecuencia, la creación de estos artefactos de desarrollo se extiende a través de todas las fases del desarrollo del sistema y, por lo tanto, durante toda la duración del proyecto (Nistala & Kumari, 2013). Cuando diferentes desarrolladores crean los requisitos del sistema, la arquitectura y los artefactos de prueba utilizando diversas herramientas del dominio de la aplicación respectiva, se debe comprobar la coherencia de todos los artefactos.

Repositorios de artefactos

Los métodos de desarrollo ágil proporcionan un conjunto de normas, reglas, principios, procesos y guías para la construcción de software. Sin embargo, cuando más de un equipo de desarrollo está trabajando con el mismo artefacto o proyecto, y en la misma base de código para un producto, se enfrentan a una disminución de la productividad del desarrollador individual debido a la sobrecarga de integración y comunicación. Si los desarrolladores trabajan en diferentes equipos, ¿cómo integrarán su trabajo y probarán la versión del software integrado? Estos desafíos aparecen cuando dos o más equipos de desarrollo están integrando su trabajo en una sola versión de software que se puede enviar

y se vuelven significativamente más difíciles cuando más equipos de desarrollo integran su trabajo en una versión de software.

Un repositorio de artefactos almacena artefactos de compilación creados mediante integración continua y los pone a su disposición para la implementación automatizada en los entornos de prueba, puesta en escena y producción. Estos repositorios ofrecen una ubicación central para almacenar esas compilaciones, y la mayoría expone una Interfaz de Programación de Aplicaciones (API) para implementar automáticamente compilaciones en los entornos de su proceso. Como parte de su lógica de proceso, puede especificar cuánto tiempo deben mantenerse las compilaciones en el repositorio, así como las condiciones para eliminar artefactos para liberar espacio.

Herramientas de control de versiones

La utilización de un sistema de control de versiones es una buena práctica para los proyectos aún si no se realiza el proceso de IC. Por ello, se hace imprescindible la aplicación de los mismos, mediante herramientas que permiten a los programadores trabajar simultáneamente sobre una versión igual del código, estas son:

Sistema de Versiones Concurrentes (CVS): Es un sistema de control de versiones centralizado que se encuentra bajo una licencia de tipo GNU/GPL, (Licencia Pública General de GNU). Se conoce por ser el primer sistema de control de versiones de código abierto con acceso a redes de área amplia para desarrolladores y en brindar *checkouts* anónimos de sólo lectura. Esta herramienta no versiona directorios ni soporta grandes cambios, pero ofrece ramificaciones, etiquetado y un buen rendimiento en la parte del cliente. Los ficheros binarios los trata internamente como si fueran de texto, razón por la cual no los maneja muy bien. Está basado en el modelo cliente-servidor y existen versiones del software para varias plataformas. Trabaja con código fuente de programas y documentos en los que su formato sea completamente de texto como ficheros xml, html y sgml (Kapos et al., 2021).

Subversion (SVN): Es una herramienta de código abierto que fue diseñada para reemplazar el CVS y mejorar sus funcionalidades. Es un software libre bajo licencia de tipo Apache/Berkeley Software Distribution (BSD); se le conoce además como SVN pues es el nombre de la herramienta de la línea de órdenes. Este sistema centralizado recuerda todos los cambios realizados en él, posibilitando un historial con datos específicos sobre estos cambios desde una ubicación principal. El número de revisión de los archivos versionados es el mismo, identificando un único estado para estos en un momento determinado. Realiza los cambios que permiten detectar errores en poco tiempo y agilizar la construcción del software. Facilita modificaciones atómicas y no posee problemas para leer archivos binarios. Su información es

almacenada en forma de un árbol de archivos y los clientes conectados al repositorio podrán compartir dicha información. Permite la creación de ramas y etiquetas de forma sencilla (Chen et al., 2017).

Mercurial: Es un sistema de control de versiones implementado con Python aunque contiene una implementación binaria escrita en C. En sus inicios funcionaba sólo con Linux pero ha sido adaptado para otros sistemas: Unix, Windows y Mac OS X. Esta herramienta de software libre brinda una interfaz web autónoma integrada, completa indexación cruzada de ficheros y conjuntos de cambios, protocolos de sincronización SSH (Secure SHell) para acceso remoto seguro y HTTP. Mercurial es sencillo pues no tiene base de datos especiales, ni servidor central y todos los datos son locales por lo que no se necesita red. Los metadatos se comparten entre los repositorios locales porque el sistema es distribuido y posee una arquitectura de archivos que reduce las búsquedas.

GitHub: Es una herramienta libre y de código abierto muy fácil de aprender que maneja proyectos de cualquier tamaño de manera rápida. Es un sistema de control de versiones distribuido ya que sus directorios de trabajo son capaces de gestionar las revisiones sin necesitar red o un servidor central. Posee ramificaciones locales y áreas de estacionamiento ventajosas, es compatible con disímiles flujos de trabajo y permite utilizar extensiones para perfeccionar su funcionalidad (Chatziasimidis & Stamelos, 2015).

Integración Continua de software

Para el procedimiento a realizar en la aplicación de la Integración Continua (IC), teniendo en cuenta las características de los proyectos en la empresa, se selecciona un repositorio centralizado porque permite identificar las versiones existentes en el mismo y controlar el trabajo del equipo de desarrollo. Por tanto, se debe escoger una herramienta de control de versiones, como las descritas anteriormente en esta investigación, para que se integre y mejore las funcionalidades del servidor de integración continua (Seth & Khare, 2015; Soni, 2015). El servidor de IC implementa tres funciones fundamentalmente:

- Actualización de la cartera de artefactos. Los archivos del proyecto deben monitorearse en busca de adiciones y cambios que resulten del desarrollo y la evolución del sistema, y la memoria del proyecto debe actualizarse en consecuencia para reflejar las adiciones y los cambios.
- Identificación de enlaces. A medida que se agregan artefactos a la memoria de un proyecto, los vínculos entre los artefactos relacionados deben identificarse y agregarse a la memoria. Estas adiciones pueden causar cambios o eliminaciones de los enlaces existentes para algunos tipos de relaciones.

- Integración de las funciones. Se realizan pruebas automatizadas de integración y compatibilidad.

En esta investigación se realiza una breve descripción de los servidores de IC más reportados por la literatura científica, dentro de los que se encuentran Jenkins, Travis CI y BuildBot entre otros.

Jenkins: Jenkins es un servidor de IC escrito en Java. Se trata de un único servidor maestro con varios esclavos de compilación que pueden ejecutarse en Windows o en Secure Sheel en cualquier sistema que lo admita. No es necesaria ninguna configuración adicional. El sistema en sí es muy simple de usar. Al usuario se le proporciona una interfaz web para controlar todos los aspectos del sistema, incluida la gestión de esclavos de compilación y la definición de trabajos de compilación. Jenkins admite complementos y ya hay muchos de ellos, implementando soporte para varios pasos de compilación y acciones posteriores a la compilación. Para definir trabajos, hay una página de configuración por trabajo donde se puede configurar cómo se activa el trabajo, cuáles son los pasos de compilación y qué notificaciones enviar una vez que finaliza el trabajo. Un trabajo se puede activar manualmente haciendo clic, usando la API RESTful o sondeando el servicio de alojamiento de código (Seth & Khare, 2015).

Travis CI: Es principalmente un servidor de integración continua alojado, diseñado específicamente para trabajar de cerca con repositorios alojados en GitHub. Travis instalará ganchos para detectar cambios y desencadenar compilaciones. Está escrito en Ruby y utiliza el intermediario de mensajes RabbitMQ para escalar y desacoplar los componentes entre sí (Gallaba & McIntosh, 2018).

Drone: Es un servidor de integración continua escrito en el lenguaje Go, y utiliza una nueva plataforma de virtualización basada en contenedores para Linux llamada Docker. Drone está inspirado en Travis CI. Permite definir qué entorno usar y qué comando ejecutar. Implementa una interfaz web moderna. A diferencia de Travis CI, que usa máquinas virtuales para crear entornos, Drone usa virtualización basada en contenedores, que es mucho más liviana y le permite a Drone asignar menos recursos más rápido. La principal limitación es que Drone solo admite entornos que se pueden ejecutar en Docker, que son básicamente distribuciones de Linux que ejecutan una versión reciente de Linux (Beaulieu-Jones & Greene, 2016).

BuildBot: Es un marco que se puede utilizar para implementar procesos de integración continua para flujos de trabajo personalizados, pero no es un servidor de integración continua completo. La idea central es que el sistema no debe imponer ninguna restricción sobre lo que se puede lograr con él, el usuario simplemente debe usar el marco para

implementar sus propios procesos. Una instancia de BuildBot se compone de un maestro de compilación y una cierta cantidad de esclavos de compilación (Acharya, 2019).

Discusiones

Los marcos o métodos ágiles como Scrum, SAFe, LeSS y Extreme Programming son ampliamente adoptados y utilizados por pequeños equipos dentro de grandes organizaciones y empresas emergentes. Las empresas establecidas y los gigantes tecnológicos tienen como objetivo introducir e implementar métodos ágiles que ayuden a organizar los procesos en organizaciones de alta complejidad. Los principales desafíos debido a una mayor complejidad se relacionan, entre otros con:

- La integración de múltiples flujos de valor.
- Las comunicaciones entre varios equipos distribuidos.
- La gestión de dependencias de artefactos entre equipos.
- El flujo de integración de artefactos entre varios equipos.

El desarrollo de software es una actividad colaborativa en la que interactúan analistas de negocios, clientes, ingenieros de sistemas, arquitectos y desarrolladores. En este sentido, el Principio 6 del Manifiesto Ágil plantea:

> "*El método más efectivo y eficiente de compartir información a, y dentro de un equipo de desarrollo, es la conversación cara a cara.*"

Sin embargo, la edición simultánea de artefactos y procesos requiere una colaboración sincrónica entre arquitectos y desarrolladores que no pueden estar físicamente presentes en una ubicación común. La integración continua de software requiere control de concurrencia en tiempo real, lo que permite a los desarrolladores dispersos geográficamente editar y discutir los mismos artefactos y mejorar la productividad al proporcionar un medio a través del cual capturar y modelar fácilmente conceptos difíciles a través de espacios de trabajo virtuales y la edición colaborativa de artefactos por medio de las herramientas descritas en el presente estudio que permiten interacciones sincronizadas. Se puede decir que las grandes organizaciones necesitan entregar, adaptar o introducir nuevos métodos para administrar las organizaciones de desarrollo y entregar los artefactos necesarios.

Otro de los artefactos generados para mantener la escalabilidad y persistencia del proceso de desarrollo de software, es la generación de artefactos de documentación, y aunque los valores fundamentales de Agile, plantean como factor de éxito la reducción de documentación, en las investigaciones realizadas se muestra la necesidad de generar este tipo de

artefactos sin alejarse se agile. Mantener y actualizar documentación valiosa también puede mejorar el proceso de colaboración del equipo de desarrollo mientras se utilizan prácticas ágiles. Los equipos ágiles no pueden depender de notas adhesivas en un tablero de tareas o un gráfico de trabajo pendiente en la pared para el seguimiento del proyecto si no están en la misma sala. Del mismo modo, los diseños y diagramas deben compartirse en múltiples ubicaciones. La decisión de distribuir un equipo debe ir acompañada del compromiso de proporcionar al equipo los artefactos que necesita para maximizar la comunicación y la expectativa de que les llevará algún tiempo optimizar a su alrededor.

Las experiencias, métodos, decisiones y habilidades de los miembros del equipo deben acumularse durante el proceso de desarrollo a través de mecanismos efectivos de intercambio de información, de modo que cada miembro del equipo pueda utilizar la experiencia de su predecesor y la experiencia del equipo acumulada durante el desarrollo, y así ahorrando costes y tiempo al evitar el trabajo redundante (Wagenaar et al., 2017).

En el trabajo realizado por (Wagenaar et al., 2017) identificaron que en el desarrollo de software ágil, se utilizan artefactos ágiles, como historias de usuarios y productos atrasados; así como artefactos no ágiles, como diseños y planes de prueba. En el citado estudio, Wagenaar y colegas encontraron que la madurez se correlacionó negativamente con la relación de los artefactos no ágil; lo que quiere decir que cuanto más madura es la gestión de productos de software, menos artefactos no ágiles se utilizan en el desarrollo ágil. Esto sugiere que un factor organizativo influye en un equipo ágil en el uso de sus artefactos, en contradicción con el concepto de equipos ágiles autoorganizados.

Limitaciones del estudio

La principal limitación que enfrentaron los autores del presente estudio es la traducción al español de los distintos artefactos que se generan en el desarrollo ágil. Ya que muchos de estos artefactos son asumidos originalmente en inglés, mientras que otros se traducen y utilizan comúnmente en español. Es por eso que las tablas y figuras de esta investigación, muestran artefactos en uno u otro idioma indistintamente. Es opinión personal de estos autores, que existe la necesidad de homologar y estandarizar la nomenclatura de estos artefactos para llegar a un entendimiento común en todas las publicaciones y evitar errores de duplicación.

Conclusiones

El presente artículo utilizó el método de revisión sistemática de la literatura para investigar los artefactos de software utilizados en el desarrollo ágil a gran escala. Esta investigación confirma que los equipos ágiles a gran escala utilizan una gran variedad de artefactos de procesos ágiles convencionales. Los artefactos ágiles más desarrollados son: historias

de usuarios, solicitudes de nuevas funciones, diseño detallado, criterios de prueba, pruebas unitarias, código fuente, estimaciones de puntos de historia, cuadro de incendio, tableros de estado, pila de sprint y pila de producto.

El aumento en la cantidad de artefactos generados también ayuda a lograr una mejor clarificación de los requisitos de coordinación con casos de uso e historias de usuarios. Por lo tanto, se recomienda la flexibilidad y el ajuste del proyecto en lugar de seguir estrictamente los principios ágiles, esto dependerá de la madurez del equipo de desarrollo. Las investigaciones de este estudio combinan hábilmente artefactos ágiles con artefactos convencionales basados en planes, como arquitecturas de referencia, evaluaciones de riesgos, pruebas de regresión, estándares de arquitectura, pruebas de aceptación de usuarios, planes de lanzamiento y lanzamientos de productos.

Es consenso general en la literatura revisada, que la trazabilidad entre diferentes artefactos ayuda a reducir el tiempo y el costo de desarrollo y mantenimiento, mejorando así la calidad del sistema. Si bien la información en los repositorios de software se puede extraer para estudiar la evolución del software con el fin de realizar un análisis retrospectivo, también es directamente útil para un desarrollador que se dedica a tareas de mantenimiento y evolución del software.

Conflictos de intereses

Los autores declaran que no poseen conflicto de intereses.

Contribución de los autores

1. Conceptualización: Patricia María Marcillo Sánchez
2. Curación de datos: Patricia María Marcillo Sánchez, Lugio David Román Barrezueta
3. Análisis formal: Patricia María Marcillo Sánchez, Lugio David Román Barrezueta
4. Adquisición de fondos: Patricia María Marcillo Sánchez, Lugio David Román Barrezueta
5. Investigación: Patricia María Marcillo Sánchez, Lugio David Román Barrezueta
6. Metodología: Patricia María Marcillo Sánchez, Lugio David Román Barrezueta
7. Administración del proyecto: Patricia María Marcillo Sánchez
8. Recursos: Patricia María Marcillo Sánchez, Lugio David Román Barrezueta
9. Software: Patricia María Marcillo Sánchez, Lugio David Román Barrezueta
10. Supervisión: Patricia María Marcillo Sánchez
11. Validación: Patricia María Marcillo Sánchez, Lugio David Román Barrezueta
12. Visualización: Patricia María Marcillo Sánchez, Lugio David Román Barrezueta

13. Redacción – borrador original: Patricia María Marcillo Sánchez, Lugio David Román Barrezueta
14. Redacción – revisión y edición: Patricia María Marcillo Sánchez, Lugio David Román Barrezueta

Financiamiento

La investigación fue financiada por los autores.

Referencias (

Acharya, P. (2019). Design and Implementation of a Continuous Integration System. https://www.theseus.fi/bitstream/handle/10024/167981/Acharya_Prakash.pdf?sequence=2

Acharya, S. (2014). *Mastering unit testing using Mockito and JUnit*. Packt Publishing Ltd. https://www.google.com/books?hl=es&lr=&id=zVoHBAAAQBAJ&oi=fnd&pg=PT11&dq=junit+testing&ots=6fAuv8-BJP&sig=cb53hVa97Muq7lfuDgI16JEhFWI

Ajayi, O. O., Chiemeke, S. C., & Ukaoha, K. C. (2019). Assessing the stability of selected software components for reusability. *2019 IEEE AFRICON*, 1-7. https://doi.org/10.1109/AFRICON46755.2019.9133973

Alam, O. (2019, 25-25 May 2019). Towards an Agile Concern-Driven Development Process. 2019 IEEE/ACM International Conference on Software and System Processes (ICSSP),

Almeida, F., & Espinheira, E. (2022). Adoption of Large-Scale Scrum Practices through the Use of Management 3.0. Informatics,

Ambler, S. (2002). *Agile modeling: effective practices for extreme programming and the unified process*. John Wiley & Sons. https://books.google.com/books?hl=es&lr=&id=uh_jSk2FSa0C&oi=fnd&pg=PR5&dq=Agile+Modeling+%2BAmbler+2002&ots=sU81IKcyie&sig=bmQQC0thInso1IhMdhDLtd8_pds

Anderson, D. J. (2010). *Kanban: successful evolutionary change for your technology business*. Blue Hole Press.

Arapidis, C. (2012). *Sonar code quality testing essentials*. Packt Publishing Ltd. https://www.google.com/books?hl=es&lr=&id=VbWugJpp5tMC&oi=fnd&pg=PT16&dq=Sonar+testing&ots=AIk67kI4y-&sig=_yxbIHvbDYp3QS28T6n5gbaGpM0

Beaulieu-Jones, B. K., & Greene, C. S. (2016). Reproducible computational workflows with continuous analysis. *bioRxiv*, 056473. https://www.biorxiv.org/content/10.1101/056473.abstract

Beck, K. (1999). Embracing change with extreme programming. *Computer*, *32*(10), 70-77. https://ieeexplore.ieee.org/abstract/document/796139/

Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., & Jeffries, R. (2001). Manifesto for agile software development. https://moodle2019-20.ua.es/moodle/pluginfile.php/2213/mod_resource/content/2/agile-manifesto.pdf

Behutiye, W., Rodr, P., x00Ed, guez, & Oivo, M. (2022). Quality Requirement Documentation Guidelines for Agile Software Development. *IEEE Access*, *10*, 70154-70173. https://doi.org/10.1109/ACCESS.2022.3187106

Benoit, M., Anthony, R., & Wee, L. B. (1999). Feature-driven development. http://csis.pace.edu/~marchese/CS616/Agile/FDD/fdd2.pdf

Böhmer, A. I., Meinzinger, M., Hostettler, R., Knoll, A., & Lindemann, U. (2017, 27-29 June 2017). Towards a framework for agile development of physical products influence of artifacts and methods. 2017 International Conference on Engineering, Technology and Innovation (ICE/ITMC),

Borman, R. I., Priandika, A. T., & Edison, A. R. (2020). Implementasi metode pengembangan sistem Extreme Programming (XP) pada aplikasi investasi peternakan. *JUSTIN (Jurnal Sistem Dan Teknologi Informasi)*, *8*(3), 272-277. https://jurnal.untan.ac.id/index.php/justin/article/view/40273

Briatore, S., & Golkar, A. (2021). Estimating Task Efforts in Hardware Development Projects in a Scrum Context. *IEEE Systems Journal*, *15*(4), 5119-5125. https://doi.org/10.1109/JSYST.2021.3049737

Castro, S. J. B., Crespo, R. G., & Medina, V. H. (2014). Software Processes and Methodologies Modeling Language SPMML, A Holistic Solution for Software Engineering. *IEEE Latin America Transactions*, *12*(4), 818-824. https://doi.org/10.1109/TLA.2014.6868888

Cockburn, A. (2001). Crystal Clear: A human-powered software development methodology for small teams. *Reading: Addison-Wesley*.

Conboy, K. (2009). Agility from first principles: Reconstructing the concept of agility in information systems development. *Information systems research*, *20*(3), 329-354. https://ulir.ul.ie/bitstream/handle/10344/574/L+1.pdf?sequence=2

Conboy, K., & Carroll, N. (2019). Implementing large-scale agile frameworks: challenges and recommendations. *Ieee Software*, *36*(2), 44-50. https://ieeexplore.ieee.org/abstract/document/8648270/

Chatziasimidis, F., & Stamelos, I. (2015, 6-8 July 2015). Data collection and analysis of GitHub repositories and users. 2015 6th International Conference on Information, Intelligence, Systems and Applications (IISA),

Chen, H., Leng, H., Tang, H., Zhu, J., Gong, H., & Zhong, H. (2017, 15-17 Dec. 2017). Research on model management method for Micro-grid. 2017 IEEE 2nd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC),

Denning, S. (2018). How major corporations are making sense of Agile. *Strategy & Leadership*. https://www.emerald.com/insight/content/doi/10.1108/SL-11-2017-0104/full/html

Eddy, B. P., Wilde, N., Cooper, N. A., Mishra, B., Gamboa, V. S., Patel, K. N., & Shah, K. M. (2017). CDEP: Continuous delivery educational pipeline. Proceedings of the SouthEast Conference,

Elamin, R., & Osman, R. (2017, 4-8 July 2017). Towards Requirements Reuse by Implementing Traceability in Agile Development. 2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC),

Friess, E. (2019). Scrum Language Use in a Software Engineering Firm: An Exploratory Study. *IEEE Transactions on Professional Communication*, *62*(2), 130-147. https://doi.org/10.1109/TPC.2019.2911461

Gallaba, K., & McIntosh, S. (2018). Use and misuse of continuous integration features: An empirical study of projects that (mis) use travis ci. *IEEE Transactions on Software Engineering*, *46*(1), 33-50. https://ieeexplore.ieee.org/abstract/document/8360943/

Gonen, B., & Sawant, D. (2020, 9-12 March 2020). Significance of Agile Software Development and SQA Powered by Automation. 2020 3rd International Conference on Information and Computer Technologies (ICICT),

Harvie, D. P., & Agah, A. (2016). Targeted Scrum: Applying Mission Command to Agile Software Development. *IEEE Transactions on Software Engineering*, *42*(5), 476-489. https://doi.org/10.1109/TSE.2015.2489654

Hess, A., Diebold, P., & Seyff, N. (2017, 4-8 Sept. 2017). Towards Requirements Communication and Documentation Guidelines for Agile Teams. 2017 IEEE 25th International Requirements Engineering Conference Workshops (REW),

Hillemacher, S., Jäckel, N., Kugler, C., Orth, P., Schmalzing, D., & Wachtmeister, L. (2021). Artifact-Based Analysis for the Development of Collaborative Embedded Systems. In *Model-Based Engineering of Collaborative Embedded Systems* (pp. 315-331). Springer. https://link.springer.com/chapter/10.1007/978-3-030-62136-0_17

IEEE. (2006). IEEE Standard for Developing a Software Project Life Cycle Process. *IEEE Std 1074-2006 (Revision of IEEE Std 1074-1997)*, 1-110. https://doi.org/10.1109/IEEESTD.2006.219190

Imran, R., & Soomro, T. R. (2022, 16-17 Feb. 2022). Mapping of Agile Processes into Project Management Knowledge Areas and Processes. 2022 International Conference on Business Analytics for Technology and Security (ICBATS),

Jeffries, R. (2014). SAFe–Good But Not Good Enough. *RonJeffries. com [online]*, 2014-2002.

Jeong, S., Cho, H., & Lee, S. (2018, 27 May-3 June 2018). Agile requirement traceability matrix. 2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion),

Johnson, J. (2018). *CHAOS report: decision latency theory: it is all about the interval*. Lulu. com. https://www.google.com/books?hl=es&lr=&id=WV1QDwAAQBAJ&oi=fnd&pg=PA1&dq=The+chaos+report&ots=9_GSRIBUZj&sig=XybjqBoyFwXk7Ue5-5A4EU_mbh0

Kapos, G. D., Tsadimas, A., Kotronis, C., Dalakas, V., Nikolaidou, M., & Anagnostopoulos, D. (2021). A Declarative Approach for Transforming SysML Models to Executable Simulation Models. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, *51*(6), 3330-3345. https://doi.org/10.1109/TSMC.2019.2922153

Keshta, N., & Morgan, Y. (2017, 3-5 Oct. 2017). Comparison between traditional plan-based and agile software processes according to team size & project domain (A systematic literature review). 2017 8th IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON),

Kessel, M., & Atkinson, C. (2018). Integrating Reuse into the Rapid, Continuous Software Engineering Cycle through Test-Driven Search. *2018 IEEE/ACM 4th International Workshop on Rapid Continuous Software Engineering (RCoSE)*, 8-11. https://ieeexplore.ieee.org/document/8452100/

Larman, C., & Vodde, B. (2017). *Large-Scale Scrum: More with LeSS*. Dpunkt. https://www.craiglarman.com/wiki/downloads/Book%20Large-Scale%20Scrum%20-%20More%20with%20LeSS/Large-Scale%20Scrum%20-%20More%20with%20LeSS%20-%20Chapter%202%20-%20LeSS.pdf

[Record #600 is using a reference type undefined in this output style.]

Leite, A. I. M. (2017, 4-8 Sept. 2017). An Approach to Support the Specification of Agile Artifacts in the Development of Safety-Critical Systems. 2017 IEEE 25th International Requirements Engineering Conference (RE),

Lenka, R. K., Kumar, S., & Mamgain, S. (2018). Behavior driven development: Tools and challenges. 2018 International Conference on Advances in Computing, Communication Control and Networking (ICACCCN),

Li, Y., Zhou, Y., Stafford, T., & Wang, X. (2021). Significant Stakeholders: Toward an Agile Knowledge Management System in the Time of Coronavirus Crisis. *IEEE Engineering Management Review*, *49*(1), 38-49. https://doi.org/10.1109/EMR.2020.3036816

Li, Z., Brien, L. O., & Yang, Y. (2014, 7-10 April 2014). Impact of Product Complexity on Actual Effort in Software Developments: An Empirical Investigation. 2014 23rd Australian Software Engineering Conference,

Lucas, E. M., Oliveira, T. C., Schneider, D., & Alencar, P. S. C. (2020). Knowledge-Oriented Models Based on Developer-Artifact and Developer-Developer Interactions. *IEEE Access*, *8*, 218702-218719. https://doi.org/10.1109/ACCESS.2020.3042429

Lunesu, M. I., Tonelli, R., Marchesi, L., & Marchesi, M. (2021). Assessing the Risk of Software Development in Agile Methodologies Using Simulation. *IEEE Access*, *9*, 134240-134258. https://doi.org/10.1109/ACCESS.2021.3115941

Masood, Z., Hoda, R., & Blincoe, K. (2022). Real World Scrum A Grounded Theory of Variations in Practice. *IEEE Transactions on Software Engineering*, *48*(5), 1579-1591. https://doi.org/10.1109/TSE.2020.3025317

Matthies, C., Huegle, J., Dürschmid, T., & Teusner, R. (2019, 25-31 May 2019). Attitudes, Beliefs, and Development Data Concerning Agile Software Development Practices. 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET),

Matthies, C., Kowark, T., Richly, K., Uflacker, M., & Plattner, H. (2016, 16-16 May 2016). ScrumLint: Identifying Violations of Agile Practices Using Development Artifacts. 2016 IEEE/ACM Cooperative and Human Aspects of Software Engineering (CHASE),

Méndez Fernández, D., Böhm, W., Vogelsang, A., Mund, J., Broy, M., Kuhrmann, M., & Weyer, T. (2019). Artefacts in software engineering: a fundamental positioning. *Software & Systems Modeling*, *18*(5), 2777-2786. https://link.springer.com/article/10.1007/s10270-019-00714-3

Moyo, S., & Mnkandla, E. (2020). A Novel Lightweight Solo Software Development Methodology With Optimum Security Practices. *IEEE Access*, *8*, 33735-33747. https://doi.org/10.1109/ACCESS.2020.2971000

Mustafa, N., & Labiche, Y. (2015, 9-11 Feb. 2015). Towards traceability modeling for the engineering of heterogeneous systems. 2015 3rd International Conference on Model-Driven Engineering and Software Development (MODELSWARD),

Nistala, P., & Kumari, P. (2013, 19-19 May 2013). Establishing content traceability for software applications: An approach based on structuring and tracking of configuration elements. 2013 7th International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE),

Panizzi, M. D., Hossian, A., & García Martínez, R. (2016). Implantación de sistemas: estudio comparativo e identificación de vacancias en metodologías usuales. XXII Congreso Argentino de Ciencias de la Computación (CACIC 2016).

Patkar, N., Chis, A., Stulova, N., & Nierstrasz, O. (2022, 15-18 March 2022). First-class artifacts as building blocks for live in-IDE documentation. 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER),

Perera, I., Meedeniya, D., & Bandara, M. (2015, 18-20 Dec. 2015). A traceability management framework for artefacts in self-adaptive systems. 2015 IEEE 10th International Conference on Industrial and Information Systems (ICIIS),

Poppendieck, M., & Poppendieck, T. (2003). *Lean software development: an agile toolkit*. Addison-Wesley. https://books.google.com/books?hl=es&lr=&id=IJ1gAgAAQBAJ&oi=fnd&pg=PR7&dq=Poppendieck++%2B+Lean+programming.+&ots=BkRz4i9N__&sig=qUNx0vNMNkUhyEyhjLQXfLyTP1Q

Power, K. (2016). Sensemaking and Complexity in Large-Scale Lean-Agile Transformation: A Case Study from Cisco. 2016 49th Hawaii International Conference on System Sciences (HICSS),

Puri-Jobi, S. (2015). Test automation for NFC ICs using Jenkins and NUnit. 2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW),

Ramya, P., Sindhura, V., & Sagar, P. V. (2017). Testing using selenium web driver. 2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT),

Rashid, N., Khan, S. U., Khan, H. U., & Ilyas, M. (2021). Green-Agile Maturity Model: An Evaluation Framework for Global Software Development Vendors. *IEEE Access*, *9*, 71868-71886. https://doi.org/10.1109/ACCESS.2021.3079194

Rodr, G., x00Ed, guez, Gasparini, I., Kemczinski, A., & Matos, A. V. d. (2021). Students' Perception of Scrum in a Course Project. *IEEE Revista Iberoamericana de Tecnologias del Aprendizaje*, *16*(4), 329-336. https://doi.org/10.1109/RITA.2021.3136436

Schwaber, K., & Beedle, M. (2002). *Agile software development with scrum. Series in agile software development* (Vol. 1). Prentice Hall Upper Saddle River. https://www.lowrell.com/sites/default/files/webform/cv/agile-software-development-with-scrum-series-in-agile-software-d-ken-schwaber-mike-beedle-2e69aa1.pdf

Seth, N., & Khare, R. (2015, 21-22 Dec. 2015). ACI (automated Continuous Integration) using Jenkins: Key for successful embedded Software development. 2015 2nd International Conference on Recent Advances in Engineering & Computational Sciences (RAECS),

Shrivastava, A., Jaggi, I., Katoch, N., Gupta, D., & Gupta, S. (2021). A Systematic Review on Extreme Programming. Journal of Physics: Conference Series,

Soni, M. (2015, 25-27 Nov. 2015). End to End Automation on Cloud with Build Pipeline: The Case for DevOps in Insurance Industry, Continuous Integration, Continuous Testing, and Continuous Delivery. 2015 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM),

Srivastava, A., Bhardwaj, S., & Saraswat, S. (2017). SCRUM model for agile methodology. *2017 International Conference on Computing, Communication and Automation (ICCCA)*, 864-869. https://ieeexplore.ieee.org/abstract/document/8229928/

Stapleton, J. (1997). *DSDM, dynamic systems development method: the method in practice*. Cambridge University Press. https://books.google.com/books?hl=es&lr=&id=kp656t4p7soC&oi=fnd&pg=PR11&dq=Dynamic+Systems+Development+Method+%2B+Stapleton+1997&ots=TzIhPuZRVh&sig=AFGQT2jU2QGtGHLMslSh6VdwQlU

Wagenaar, G., Overbeek, S., Lucassen, G., Brinkkemper, S., & Schneider, K. (2017). Influence of software product management maturity on usage of artefacts in Agile software development. International Conference on Product-Focused Software Process Improvement,

Warnars, H. L. H. S., Gaol, F. L., & Randriatoamanana, R. (2017). Object oriented metrics to measure the quality of software upon PHP source code with PHP_depend study case request online system application. 2017 International Conference on Applied Computer and Communication Technologies (ComCom),

Wińska, E., & Dąbrowski, W. (2020). Software development artifacts in large agile organizations: a comparison of scaling agile methods. In *Data-Centric Business and Applications* (pp. 101-116). Springer. https://link.springer.com/chapter/10.1007/978-3-030-34706-2_6

Zandstra, M. (2016). Testing with PHPUnit. In *PHP Objects, Patterns, and Practice* (pp. 435-464). Springer. https://link.springer.com/chapter/10.1007/978-1-4842-1996-6_18