UNIVERSIDAD PERUANA UNIÓN

ESCUELA DE POSGRADO

Unidad de Posgrado de Ingeniería y Arquitectura

Marco de trabajo SCRUM extendido con prácticas de Clean Architecture para la mantenibilidad de software

Tesis para obtener el Grado Académico de Maestro en
Ingeniería de Sistemas con Mención en Ingeniería de Software

Autor:

Ing. Eduardo Cesar Arango Quincho

Asesor:

Mg. Omar Leonel Loaiza Jara

Lima, Julio de 2021

DECLARACIÓN JURADA DE AUTORÍA DE TESIS

Omar Leonel Loaiza Jara, de la Escuela de Posgrado, Unidad de Posgrado de Ingeniería y Arquitectura, de la Universidad Peruana Unión.

DECLARO:

Que la presente investigación titulada: **"Marco de trabajo SCRUM extendido con prácticas de Clean Architecture para la mantenibilidad de software"** constituye la memoria que presenta el Licenciado Eduardo Cesar Arango Quincho para aspirar al Grado Académico de Maestro en Ingeniería de Sistemas con Mención en Ingeniería de Software, cuyo trabajo de investigación ha sido realizada en la Universidad Peruana Unión bajo mi dirección.

Las opiniones y declaraciones en este informe son de entera responsabilidad del autor, sin comprometer a la institución.

Y estando de acuerdo, firmo la presente declaración en la ciudad de lima, a los 12 días del mes de Julio del año 2021

Mg. Omar Leonel Loaiza Jara

ACTA DE SUSTENTACIÓN DE TESIS DE MAESTRO(A) 315

En Lima, Ñaña, Villa Unión, a 02 días del mes de julio del año 2021, siendo las 10:00 a.m, se reunieron en la modalidad online sincrónica, bajo la dirección del Señor Presidente del Jurado: Mg. Sergio Omar Valladares Castillo, el secretario: Mg. Nemias Saboya Rios, los demás miembros: Mg. Abel Angel Sullon Macalupu y el M.Sc. Fredy Abel Huanca Torres y el asesor: Mg. Omar Leonel Loaiza Jara, con el propósito de administrar el acto académico de sustentación de Tesis de Maestro(a) titulada: Marco de trabajo scrum extendido con prácticas de clean architecture para el mantenimiento de software del Bachiller/Licenciado(a) Eduardo Cesar Arango Quincho

Conducente a la obtención del Grado Académico de Maestro(a) en: Ingeniería de Sistemas
(Nomenclatura del Grado Académico)
con Mención en Ingeniería de Software.

El Presidente inició el acto académico de sustentación invitando al candidato hacer uso del tiempo determinado para su exposición. Concluida la exposición, el Presidente invitó a los demás miembros del Jurado a efectuar las preguntas, cuestionamientos y aclaraciones pertinentes, los cuales fueron absueltos por el candidato. Luego se produjo un receso para las deliberaciones y la emisión del dictamen del Jurado. Posteriormente, el Jurado procedió a dejar constancia escrita sobre la evaluación en la presente acta, con el dictamen siguiente:

Bachiller/Licenciado (a): Eduardo Cesar Arango Quincho

CALIFICACIÓN	ESCALAS			Mérito
	Vigesimal	Literal	Cualitativa	
Aprobado	15	B-	Con nominación Bueno	Muy bueno

(*) Ver parte posterior

Finalmente, el Presidente del Jurado invitó al candidato a ponerse de pie, para recibir la evaluación final. Además, el Presidente del Jurado concluyó el acto académico de sustentación, procediéndose a registrar las firmas respectivas.

Presidente | Secretario

Asesor | Miembro | Miembro

Bachiller/Licenciado(a)

Marco de trabajo SCRUM extendido con prácticas de Clean Architecture para la mantenibilidad de software

Eduardo C. Arango[1][0000-0001-7027-3360]

[1, 2] Escuela de Posgrado, Unidad de posgrado de Ingeniería y Arquitectura. Universidad Peruana Unión,
Carretera Central Km. 19.5, Lurigancho-Chosica. Lima. Perú
eduardo.arango@upeu.edu.pe

Abstract. Este estudio consistió en la extensión del marco de trabajo Scrum con prácticas Clean Architecture (Scrum-CA) para mejorar la mantenibilidad de software. Metodológicamente, en este estudio: i) se identificaron los requerimientos que Scrum-CA debía satisfacer; ii) identificación de los aspectos (artefactos y/o actividades) Scrum a extender, iii) especificar el marco de trabajo extendido; y, iv) validar la propuesta mediante un caso de estudio. A nivel de marco de trabajo extendido, esta propuesta considera la formulación de un contexto de negocio como insumo al Product Backlog. La especificación de tareas en el Sprint backlog se delimito por su contexto para obtener una alta cohesión en el diseño de la arquitectura, así como la división en 4 espacios (infraestructura, controladores, aplicación y dominio) para que en el desarrollo de los Sprint se permita el logro de una arquitectura mantenible, cuestión que a priori los equipos Scrum deben definir empíricamente. Se añaden los principios SOLID en la implementación del producto dentro de los Sprint y se guarda coherencia entre la planificación de los requerimientos y el logro de la arquitectura del producto. Scrum-CA se validó a través de un proyecto de implementación en una entidad estatal peruana de alcance nacional. Los resultados concluyen una mejora de la mantenibilidad del software del proyecto, del Sprint1 al Sprint2, bajando la duplicidad de código de 4.8% a 3.5%, de 31 bloques duplicados a 9, una deuda técnica de 6.3% a 1.3%. Asimismo, es necesario repetir Scrum-CA en otros proyectos para generar un estándar.

Keywords: Scrum, Scrum extended, Clean architecture, software maintainability

1 Introducción

La tendencia tecnológica de la Industria 4.0 está impulsando cambios e innovaciones disruptivas en tiempos cada vez más cortos [1]. En ese sentido, existe hoy un sentido mayor de urgencia de adaptarse a los cambios de la sociedad y los mercados; como, por ejemplo, entre ellas por citar una, la coyuntura de la COVID 19, que obliga a las organizaciones a cambiar en la forma de interactuar con sus usuarios y/o clientes, socios de

negocios [2], debido a las condiciones sociales y sanitarias que se necesita para su mitigación. En ese orden de ideas, las tecnologías de la información son vitales para la transformación digital de las organizaciones, teniendo la cada vez más necesidad de crear y utilizar nuevos contenidos, aplicaciones y / o servicios mediante el uso de tecnologías que soporten sus procesos de negocios, con la finalidad de mantener una ventaja competitiva [3]. Es también cierto que, si las organizaciones crecen con el tiempo en tamaño y complejidad, el software también debe cambiar para adaptarse a las nuevas necesidades [4], esto quiere decir que el software para seguir siendo considerado un adecuado soporte debe cumplir con criterios de calidad basado en estándares que elonguen su vida operativa.

Todo sistema de información es el fruto de seguir un proceso de desarrollo [5]; donde los requerimientos llegan al final del ciclo de vida convertidos en producto. Cada actividad del ciclo de vida debe estar orientado al logro que se espera del producto para una adecuada operación en su vida útil. Como se decía anteriormente, todo software cambia en el tiempo puede ser durante el desarrollo del proyecto o cuándo está en producción [6]. Entonces si es cierto que es necesario que el software se mantenga en el tiempo, las actividades del ciclo de vida deberían tener en cuenta prácticas que permitan una adecuada mantenibilidad de su arquitectura. Un efecto de no conseguir un producto mantenible en el tiempo, definitivamente va a comprometer más recursos, se evidenciará poca integración con otros sistemas de información, hasta tener que crear nuevo software por el hecho que no es mantenible.

Es necesario mencionar que los requerimientos son el elemento principal que orienta todos los esfuerzos en los procesos desarrollo de software. En base a ellos se formula la arquitectura del software a producir, que evoluciona a medida que se desarrollan las actividades del ciclo de vida [7]. Existen cambios que pueden introducirse en cualquier iteración bajo un protocolo de gestión de cambios; pueden tener un impacto diferente en el proyecto, desde cambios en una clase hasta la redefinición de funcionalidades, lo que no cabe duda es que todas ellas afectan la arquitectura del producto, donde los efectos pueden ser indeseables como: alto el grado de acoplamiento entre componentes poca independencia de la interfaces de usuario, librerías externas y de base de datos, poca facilidad de probar funcionalidades, poca flexibilidad para añadir o remover funcionalidades, así como para corregir errores; o en todo caso, los efectos podrán verse en la futura mantenibilidad cuando el producto esté en producción [8].

Lo anterior coloca la idea de la necesidad de que se piense en la arquitectura y las tareas necesarias desde etapas más tempranas e intermedias del desarrollo de un software será mejor. En ese sentido, lo propuesto por Martin [9] con el tema de arquitecturas limpias permite pensar en la arquitectura como un tema transversal del desarrollo de software, y no como actividades puntuales y discretas.

Por otro lado, Scrum es el marco de trabajo más utilizado en la actualidad en el desarrollo de software [10]. Entre las bondades de Scrum tenemos que divide el trabajo en partes pequeñas, fomenta la interacción con el cliente y la entrega de un producto de

forma iterativa incremental permitiendo generar valor en plazos cortos [11] Es bueno indicar a priori que Scrum no tiene un enfoque a temas específicos concerniente a la arquitectura en software [12]. En ese sentido Martin [9] dice que de todos los aspectos de software, el mantenimiento es el más costoso [13], porque el software con el tiempo va cambiando en funcionalidades y van apareciendo defectos, librerías obsoletas que se tiene que actualizar, reemplazar y que requieren de una inversión grande en recursos humanos y tiempo. en ese sentido dedicar tiempo para diseñar una buena arquitectura al inicio del proyecto propenderá a garantizar una mejor mantenibilidad.

Por lo expresado anteriormente, el objetivo de este artículo es reportar la extensión del marco de trabajo Scrum con prácticas de Clean Architecture para la mantenibilidad del software; constituyéndose esta extensión en el aporte principal de esta investigación.

La importancia de esta investigación; porque proveerá de una alternativa que potencie Scrum en términos de arquitectura de software desde las etapas tempranas de los requerimientos y cumpla con criterios de mantenibilidad, y no posponer el tema para el análisis y diseño del producto, o peor aún, para el periodo de pruebas; así mismo, este estudio servirá de línea de base para posteriores investigaciones en gestión de requerimientos, arquitecturas de software, utilización de inteligencia artificial, madurez de procesos de software, etc. Por otro lado, los beneficios para cualquier organización que cumpla los principios de arquitectura limpia en un proyecto de software se pueden evidenciarse de forma indirecta así, por ejemplo, en el ahorro de recursos, en el tiempo utilizado para entender y mantener el software. De igual forma garantiza la posibilidad de posponer o desechar decisiones tecnológicas, garantizando un mínimo impacto en los cambios que pueda experimentar el software. Por último, los logros del objetivo se reseñan a través de la ejecución de un caso de estudio en una entidad estatal peruana.

1.1 Antecedentes

Taghi et.al [14], en su estudio “Adapting Scrum Process with 7C Knowledge Management Model” tuvo como objetivo gestionar el conocimiento en SCRUM adaptando el ciclo de desarrollo de Scrum con el modelo de gestión del conocimiento de las 7C, obteniendo como resultado que el modelo de las 7C se podría adaptar al marco de trabajo de Scrum.

Alotaibi [15] en su estudio de “Modelling Security Requirements Through Exten-ding Scrum Agile Development Framework” tuvo como objetivo en desarrollar un método para extender SCRUM, añadiendo las mejores prácticas de seguridad en los requisitos iniciales y durante el ciclo de desarrollo de Scrum, obteniendo como resultado que la inclusión de requerimientos de seguridad y el Rol de Security Owner tuvo un efecto positivo en el ciclo de desarrollo de Scrum.

Según Robert et al [16], en su investigación “Agile in distress: Architecture to the rescue” que tuvo como finalidad demostrar que un enfoque temprano y continuo en la

arquitectura del software permite a la organización desarrollar el enfoque ágil y minimizar el impacto de los riesgos en los constantes cambios. obteniendo como resultado que la arquitectura permite un desarrollo ágil que es un factor clave para el éxito del proyecto, la dedicación de esfuerzo en diseñar una arquitectura que pueda soportar los cambios persistentes se realiza en cada iteración de las practicas agiles.

1.2 Mantenibilidad de software

La calidad del producto software se puede definir como el grado en que dicho producto satisface los requisitos de los usuarios y además de los atributos no funcionales del software. La mantenibilidad del software está definida como la facilidad con la que se puede añadir funcionalidades, corregir errores de un producto de software [7].

Además Martínez, et al [13] afirman que las actividades de mantenimiento consume una gran parte del esfuerzo en el ciclo de vida software. De acuerdo con Udya [17], afirman que el diseño de la arquitectura es el primer paso en el desarrollo de software por consiguiente medir la calidad en el diseño de la arquitectura ayudará a determinar la calidad del software

Alkharabsheh et al [18], en su estudio de "Software Design Smell Detection a systematic mapping study" tuvo como objetivo detectar los defecto de diseño que afecten negativamente a los atributos de calidad del software, como la comprensibilidad, la capacidad de prueba, la extensibilidad, la reutilización y la capacidad de mantenimiento, obteniendo como resultado que Design Smell Detection influye positivamente en los atributos de calidad del software.

1.3 Arquitectura Limpias

La arquitectura de software es la forma como se organiza y se interrelacionan entre los componentes que lo conforman [7].

Las Clean Architecture son una serie de reglas y principios para que una Architecture de software sea considerado limpia, cuya finalidad es que sea mantenible. Para ello se basan en los principios de separación de responsabilidades, permitiendo separar los elementos de la arquitectura en capas interdependiente las cuales son: capa de Frameworks y Drivers que es la capa más externa donde están los componentes que pueden cambiar fácilmente, seguido de la capa de Adaptadores de Interfaz, seguidamente la capa de Caso de Usos del Sistema y finalmente la capa de Entidades, cada capa debe cumplir la regla de dependencia de afuera hacia el centro y no al revés, en ese sentido la capa de Entidad no sabe nada de la infraestructura ni de las IU, las CA tiene como fundamento los principios SOLID que nos permite gestionar los problemas de diseño de software que se presentan en el desarrollo del mismo, la finalidad de las CA es diseñar aplicaciones modulares con bajo acoplamiento y alta cohesión, independientes de la infraestructura, como bases de datos y UI, Web, etc. de esa manera la aplicación satisface las necesidades de los clientes proporcionando flexibilidad de mantener y extender funcionalidades así como también de probar [9].

Las Clean Architecture se caracterizan por ser:

Independiente de frameworks: la arquitectura no depende de librerías ni framework, estos solo son herramientas y el cambio de estas no deben impactar en la lógica de negocio.
Comprobable: las reglas de negocio se deben probar independientemente de la UI, base de datos.
Independiente de la UI: la UI puede cambiar sin afectar a la lógica del negocio
Independiente de la base de datos: se puede cambiar de base de datos, con un mínimo de esfuerzo, porque la lógica de negocio no depende de las bases de datos.
Independiente de cualquier agencia externa: la regla de negocio no sabe de las interfaces externas.

Las CA están dividido por capas, las cuales están definidos por círculos externos que forman los mecanismos, los círculos internos son políticas de negocios, la dependencia debe ser hacia el centro y los elementos de círculos internos no deben saber nada sobre los elementos externos, así mismo las CA tiene como base los principios de diseño SOLID [9].

Las capas de Clean Architecture son:

Entidades: Es la capa central de la arquitectura CA, es la que contiene la lógica de negocio de la empresa, donde están definido las reglas de alto nivel las mismas, que son menos propenso cambiar ni tampoco le afectaría si existiera un elemento de la capa de Framework y Driver

Casos de Usos: Esta capa contiene todas las reglas específicas de la aplicación, donde se implementa todos los casos de uso del sistema. Además, esta capa permite el flujo de datos hacia y desde las entidades.

Adaptadores de Interfaz: Esta capa evita que exista una dependencia directa hacia los elementos internos y externos de la aplicación.

Frameworks y Drivers: Esta capa es la más externa del modelo de Clean Architecture, donde están los elementos que tiendan a cambiar con mayor frecuencia, como por ejemplo las bases de datos, frameworks, el cambio de uno de estos elementos no afecta a las reglas de negocio.

1.4 Marco de trabajo Scrum

Scrum es un marco de trabajo para el desarrollo Ágil de productos de software. Se basa en los principios, prácticas y valores ágiles, para la entrega de un producto terminado y con un alto valor en el menor tiempo, además permite rápidamente y repetidas veces

inspeccionar el trabajo realizado así mismo brinda autonomía al equipo para determinar la mejor manera de entregar funcionalidades [11].

1.5 Prácticas de Arquitecturas Limpias y marco de trabajo de Scrum

El modelo de desarrollo de software tradicionales tales como cascada, prototipos, iterativo incremental, proceso unificado, etc., se caracteriza por seguir un plan definido que no cambian son rígidos, donde se tienen como primera actividad el identificar los requisitos y en base a ello se hace un plan fijando hitos, y por cada hito existe un conjunto de artefactos a entregar [7]. En estos modelos de proceso de desarrollo son complejos carecen de flexibilidad, genera muchas documentaciones y se corre el riesgo de tener la documentación desactualizada, estos procesos son muy burocráticos y están orientados para proyecto donde los requisitos no cambian con frecuencia.

Para proyectos que se caracterizan por tener requisitos cambiantes, y poca claridad al principio están los procesos de desarrollo ágil entre ellos tenemos a Scrum que es un marco de trabajo para desarrollar productos complejos que tiene requisitos cambiantes [11]. en ese sentido Staron et al [8] nos dice que los principios del desarrollo ágil cada son vez son más utilizados para proyecto grandes de desarrollo de software, pero que al tener equipos autoorganizados que desarrollan estos proyecto existe el riesgo que las dependencias arquitectónicas entre los componentes de software se vuelvan cada vez más difícil de mantener. En ese sentido se requiere extender el marco de trabajo de Scrum con prácticas de Clean Architecture para facilitar el desarrollo de una arquitectura con bajo acoplamiento y alta cohesión que se adapte a estos entornos de trabajo cambiante y que no requieran un alto esfuerzo para su implementación conforme va creciendo en funcionalidades el software.

En la presente investigación se realizó con Clean Architecture porque los principios y reglas aportan para diseñar de una arquitectura de software limpia, porque nos permite separar la lógica del dominio de toda la infraestructura así como la organización de los componentes por su contexto delimitado, permitiendo dividir el modelo en pequeñas piezas para obtener una alta cohesión y bajar el acoplamiento entre componentes, de tal manera que la lógica del negocio se traslade a la arquitectura del producto, con la finalidad de que tanto las entidades y casos de uso lo define el propio negocio, y puede ser entendió por los usuarios de negocio así también por el equipo de desarrollo de esta forma se habla un mismo lenguaje [9] las prácticas de Clean Architecture se puede aplicar para para cualquier tipo de arquitectura de software así como en las arquitecturas de Event-driven Architecture, Event Sourcing o arquitectura microservicios, las CA brindan principios y reglas para que una arquitectura sea llamada limpia, cabe resaltar que la implementación con estos paradigmas de Architecture puede llevar una curva de aprendizaje mediamente alto, así como un cambio de paradigmas a la hora de programar en algunos como por ejemplo Event Sourcing requiere de un complejo diseño gestión de errores para que no haya inconsistencias por alguna caída de unos de los servicios interrelacionado [19], o las tradicionales forma de trabajo (3 capas, N capas, etc) que tienen alto acoplamiento que añade una complejidad para la mantenibilidad del

software [19]. así mismo las CA se puede aplicar para desarrollos de aplicaciones web, móvil. Una de las bondades que obtenemos con las prácticas de Clean Architecture es que no importa donde lo vayas a desarrollar, más conocido como cross platform.

2 Metodología

Esta investigación consistió en el diseño de 4 etapas: i) identificación de requerimientos y prácticas de Clean Architecture; ii) identificación de aspectos de Scrum a extender; iii) especificación el marco de trabajo extendido; y, iv) validación del marco de trabajo extendido en un caso de estudio.

En lo concerniente a la primera etapa, se efectuó una revisión de fuentes de referencia de forma indagatoria y que estén relacionados a Scrum y arquitectura de software, para luego consolidar los resultados y extraer de ellos los requerimientos que sean necesarios para la extensión materia de este artículo. Asimismo, se efectuó una consulta a profesionales con el mismo propósito.

Respecto a la etapa 2, se relacionaron los requerimientos identificados con los aspectos Scrum (artefactos y actividades), de modo tal que se desprenda de dicho análisis la acción a emprender, como agregar un elemento nuevo a un artefacto o actividad (A), o extender (E) el propósito mediante una especificación.

En la etapa 3, se detalla en que consiste la especificación de los elementos agrega-dos y/o extendidos y que están articulados al bosquejo del marco de trabajo Scrum extendido con Clean Architecture que esta investigación propone y presenta. Así mismo, se describe el flujo de trabajo del marco de trabajo extendido. Cabe precisar que esta investigación no propone un nuevo Scrum, o desnaturaliza su propósito; sino que, basándose en los principios ágiles y flujo de trabajo conocido de él, lo extiende para potenciarlo. Se entiende, que para esta propuesta sea estándar, necesita que se lo experimente múltiples veces.

Por último, en la etapa 4, para ver el comportamiento del marco de trabajo extendido es necesario verlo en acción en un caso de estudio; a través de un caso de negocio, el cual es presentado en la sección 4.4 que cuenta con una descripción como tal y de su alcance también; luego, se ejecuta el marco de trabajo; finalmente, se reportan los resultados obtenidos de su aplicación y sus efectos en la mantenibilidad del software construido.

3 Resultados

3.1 Identificación de requerimientos y prácticas de Clean Architecture

Scrum es un marco de trabajo que parte de la identificación de requerimientos, pudiendo quedar por entendido el contexto de negocio.

Scrum es un marco de trabajo empírico, poco prescriptivo y altamente flexible [18], de modo que la formación del Product Backlog podría partir por amplios y diferentes caminos y desde el inicio no contemplar un contexto de negocio común entre las partes contratantes de un proyecto de desarrollo de software; más un si lo que se entrega no necesariamente garantiza que se contemple información antecedente de la arquitectura de un software o de otros con los que interopere, produciendo potenciales problemas de mantenibilidad futuros. De modo que contar con una comprensión, aunque mínima del contexto (REQ1) sería muy útil.

Como se sabe el resultado y principal razón, tanto de un proceso de desarrollo como un marco de trabajo como Scrum orientado a software, es el producto en manos del cliente para su explotación. Cabe precisar que dicho producto se logra en base a requerimientos. Hay que indicar que todo software, basa su estructura en una arquitectura guiada por requerimientos. Las premisas anteriores son razonables para pensar que la arquitectura se logra de forma progresiva y es una idea transversal a lo largo de todo un proceso de desarrollo. Podría decirse de algún modo que la idea no es tan reciente, no al menos de la forma tradicional en procesos de desarrollo más ceremoniosos como RUP, Open Up entre otros; pero si en Scrum, donde parece que la variabilidad de los tiempos y los requerimientos tienden a requerir ajustes recurrentes en la arquitectura, perjudicando su mantenibilidad.

Entonces queda sentada la idea que los requerimientos son básicos y orientadores Si el Product backlog es la piedra angular que guía todo el desarrollo iterativo e incremental de software en Scrum, y del cual se deriven sprint backlogs con tareas identificadas, ¿no sería mejor explicitar y pensar en la arquitectura desde estadíos tempranos (REQ2 a REQ5), en aquellos aspectos que varían más y aquellos que no? ¿por qué pensar lo variable y menos variable recién en la implementación dentro de un Sprint? Queda claro entonces, que se debe trabajar la arquitectura de una forma diferente, pasar por un lado un trabajo en capas acopladas a capas interdependiente; y por otro lado, comenzar con las tareas que expresen lo menos variable de la arquitectura e ir incorporando progresivamente en aquellos aspectos más variables.

Ahora, Clean Architecture, permite pensar en la arquitectura trabajando con capas interdependientes y mejor especificadas desde estadios tempranos, visibilizando lo más variante es la capa de infraestructura (REQ2), dado que la tecnología que permite la interacción entre el usuario y la aplicación o manejo computacional de la información puede variar por cuestiones de obsolescencia, eficiencia tecnológica o coyuntura de uso. En tanto que la capa menos variante es la que contiene las reglas de negocio empresarial en la capa de dominio (REQ5). En el intermedio se encuentran las capas de comportamiento de cómo se usan las reglas de negocio en la aplicación (REQ4) y cómo los resultados (REQ3) de estas se adaptan en la capa de infraestructura. A la clasificación anterior de las tareas en el sprint backlog en 4 categorías, plasmadas en los requerimientos REQ2 a REQ5 se le denomina en este trabajo de investigación, Backlog de arquitectura

Cabe precisar que la implementación de una arquitectura debe ser fruto de una planificación previa consideradas en el Product y Sprint backlog, ¿qué pasa si ello no ocurre?, se corre el riesgo de lograr un producto poco mantenible. La mejor opción de reducir ese riesgo es verificando que las tareas están clasificadas convenientemente y se orientan a la arquitectura por capas independientes (REQ6). Los requerimientos resultantes son:

Table 1. Requerimientos identificados para la extensión de Scrum

Código	Descripción
REQ01	El marco de trabajo extendido debe explicitar el contexto de negocio.
REQ02	Identificar tareas orientadas a la infraestructura relacionada al producto.
REQ03	Identificar tareas orientadas a los controladores que son necesarios para el producto.
REQ04	Identificar tareas orientadas al logro de la aplicación.
REQ05	Identificar tareas relacionadas al dominio relacionado con la aplicación.
REQ06	Se debe verificar que las tareas identificadas permitirán lograr la arquitectura del producto por capas interdependientes, las cuales tenemos la capa de dominio, capa de casos de usos, capa de controladores/presentadores y capa de infraestructura. Además, las Clean Architecture permite que las entidades y casos de uso lo define el negocio, permitiendo hablar un mismo idioma entre las personas del negocio y equipo scrum de esa manera estamos brindando transparencia al proyecto
REQ07	La programación del producto debe considerar los principios SOLID en las clases que constituyen la arquitectura del producto.
REQ08	Verificar adhesión a principios SOLID en el desarrollo de cada Sprint
REQ09	Las lecciones aprendidas de cada Sprint deben incorporar la reflexión de temas relacionados a la arquitectura.
REQ10	La revisión técnica del Sprint debe incorporar en su análisis el cumplimiento de los principios de arquitectura.

3.2 Identificación de aspectos de Scrum a extender

Uno de los aspectos identificados en Scrum, es la necesidad que el Product backlog tenga como insumo de entrada un Contexto de negocio se debe a que este artefacto cobre mayor sentido y dominio (control) en el equipo [20], les permita una mejor planificación [20] al familiarizarse y crecer con él al cambiar el contexto o este explique porque suceden los cambios en el Product Backlog [21], o ayudar a relacionar el efecto de un cambio en el producto (aunque este venga de forma tardía) [22] con el contexto cuando se comuniquen en los Daily meeting o en las retrospectivas. Este contexto debe

constar de una descripción breve del caso de negocio, narrar una situación problemática relacionado a aspectos de una realidad de estudio, así como denotar los interlocutores válidos, que ayuden a comprender los requerimientos listados en el Product backlog.

Por otro lado, en los Sprint backlog, los problemas más palpitantes en el cambio continuo de un poco claro Product backlog redunda definitivamente en los Sprint backlog y en el producto. Problemas como corrección de defectos, mejoras por errores o inconsistencias en el diseño [23], conflictos en la operación del producto, etc. los cuales claramente tienen que ver con la mantenibilidad del producto y su arquitectura [24]. Esta propuesta enriquece el Sprint backlog, al clasificar las tareas los 4 estancos (infraestructura, dominio, aplicación y reglas de negocio) de la configuración de capas de Clean Architecture [9]; de forma que el equipo puede ocuparse las tares más y menos cambiantes, en secuencia o paralelo, tal como se trabaja tradicionalmente en Scrum. Esta división permite un trabajo enfocado en la arquitectura con la agilidad acostumbrada, mejor mantenible.

En lo que se refiere a los Sprint tienen como objetivo producir incrementos para el cliente y sus objetivos son fijado en equipo para cumplir compromisos pactados. Es la etapa donde se producen los modelos orientados a la arquitectura y su implementación.

En cuanto a implementación dentro de los Sprint, los principios SOLID son buenas prácticas que guían la forma de diseñar los sistemas, que ayudan a reducir la complejidad del código además de ser más explícito y directo reduciendo el acoplamiento y generando un código mantenible y fácil de extender [9]. La responsabilidad única quiere decir que una clase debe hacer una única responsabilidad y no implementar otras funciones que no sean de su dominio de ese modo brinda una facilidad de mantenimiento por tener las responsabilidades separadas. El principio de abierto y cerrado proporciona un módulo o clase que solo debe estar abierto para extender funcionalidades más no para su modificación, eso quiere que se puede añadir código para extender funcionalidades, pero no para modificar. El principio de sustitución de liskov, permite la utilización de herencias, permitiendo que las clases hijas puedan ser utilizado como padre sin alterar su comportamiento, este principio nos ayuda a tener un sistema modular, que va de la mano con los otros principios. El Principio de segregación de interfaz, no dice que es mejor tener muchas interfaces en lugar de tener una interfaz que tenga muchas funcionalidades, como se utiliza en la forma tradicional se tiene una interfaz y se añaden muchas funcionalidades el tener muchas funcionalidades cuando se requiera utilizar la interfaz habrá métodos que no se utilizaran. El Principio de inversión de dependencia, nos dicen que las abstracciones no deberían depender de los detalles sino los detalles de las abstracciones

En lo que se refiere a las retrospectivas, tanto en el Sprint Retrospective como en el Sprint Review, se promueve un análisis en función a como la arquitectura de producto que se logra se articula a los requerimientos; en el primer caso a través de lecciones aprendidas en el flujo de trabajo Scrum, y en el otro caso, a través de una mirada técnica

a lo efectuado en los Sprint. Esto permite retro alimentar el trabajo y hacer que el producto no pierda el enfoque de mantenibilidad y logro de requerimientos.

Table 2. Aspectos Scrum a ser extendidos

Aspectos Scrum	Requerimiento	Acción	Elemento a adicionar o ampliar especificación
Product backlog	REQ1	Insertar contexto de negocio en el marco de trabajo	Contexto de negocio
Sprint backlog	REQ2	Clasificar el product backlog en términos de arquitectura	Product backlog divido en 4 compartimentos: infraestructura, controladores, aplicación y dominio
	REQ3		
	REQ4		
	REQ5		
Sprint Planning	Ninguna	Ninguna	Ninguna
Sprint	REQ6	Especificar procedimiento para necesidades de modelamiento e implementación respecto a los principios SOLID	
	REQ7	Insertar sub estado de verificación en sección To Do de Sprint	Sub estado en el To Do de tablero Scrum
Daily meeting	Ninguna	Ninguna	Ninguna
Sprint Review	REQ8 REQ10	Adicionar ítem en la agenda del Sprint review	Sprint review
Sprint Retrospective	REQ9	Adicionar ítem en la agenda del Sprint retrospective.	Sprint retrospective

3.3 Especificación del marco de trabajo Scrum extendido (Scrum-CA)

El marco de trabajo extendido comienza con la formulación del enunciado de negocio a cargo del Product Owner y que posteriormente es socializado al equipo Scrum. De este enunciado se extraen los requerimientos para el Product backlog, clasificándolos según riesgo, costo y tiempo. A continuación, en el Sprint Planning, se definieron los requisitos de mayor valor para el usuario y según la capacidad de trabajo del equipo para generar el Sprint Backlog.
En el sprint backlog el equipo de desarrollo, asume los requisitos priorizados por el Product Owner, descomponiéndolos en las diferentes capas de Clean Architecture (Dominio, Casos de Usos, controladores/presentadores, infraestructura y IU), así mismo se realiza una estimación del tiempo para completar las tareas.

Luego, el equipo de desarrollo ejecuta las actividades de implementación de dichas tareas en cada Sprint, monitoreando el desarrollo en un tablero con tres estados: To do, Doing y Done. Para las definiciones de los métodos y clases del software se aplican los principios SOLID. En paralelo a los Sprint y de forma diaria, se lleva a cabo el Daily

meeting con enfoque en la arquitectura limpia del software, para ver si hubo dificultades en llevar acabo, para ello el equipo debieron responder las siguientes preguntas ¿que he realizado?, ¿qué voy hacer? y ¿qué impedimento he tenido?

El Sprint Retrospective, es la reunión que se lleva acabo al final de cada Sprint, y participan el equipo de desarrollo donde se evalúa que mejoras hay que realizar al proceso en el siguiente sprint, considerando las prácticas de Clean Architecture se hayan llevado a cabo. Del mismo modo en el Sprint Review se revisa técnicamente el Sprint en términos de producto con enfoque en arquitectura.

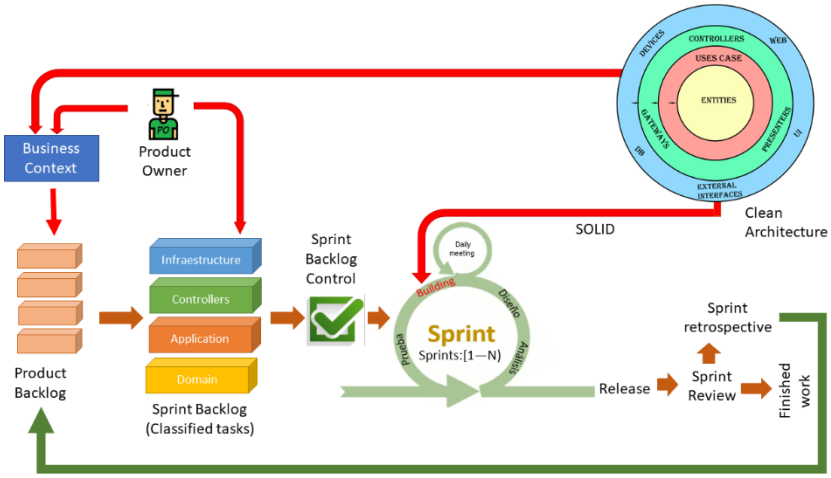

Fig. 1. Marco de trabajo extendido Scrum + Clean Architecture

3.4 Validar del marco de trabajo extendido en caso de estudio

La implementación del caso de estudio del marco de trabajo extendido se realizó en una entidad estatal del Perú. El Scrum Master llevo a cabo las actividades de incepción, la misma que consistió en brindar una inducción sobre el marco de trabajo Scrum extendido (Scrum-CA), unificar los objetivos del proyecto. Cabe precisar que el Product Owner elaboró el contexto de negocio, que brevemente se expone a continuación:
La entidad estatal donde se realizó el caso de estudio, en el contexto de la crisis sanitaria por el COVID-19, se tuvo la necesidad de implementar una plataforma virtual para la recepción de solicitudes de los ciudadanos vía internet, ya que antes la recepción era en forma física y presencial, en ese sentido la entidad estatal realizó las modificatorias a las normas legales, cuyo propósito es habilitar la recepción de documentos en formato digital. El Portal Digital es una plataforma web, donde el ciudadano pueden realizar solicitudes y adjuntar la documentación requerida según el tipo de solicitud. Cada solicitud tiene un costo, el Portal debe validar el pago realizado para dicho trámite según medio de pago, ya sean en el banco estatal, recaudación propia, además de la interoperabilidad con otros sistemas internos, en este contexto se requiere tener el sistema que

tenga una arquitectura modular que en el diseño de su arquitectura se consideraron los principios y reglas de CA, para el diseño de la arquitectura, al cumplir las regla de dependencia y los principios SOLID en la arquitectura facilitó las tareas de mantenimiento, para el caso de estudio se desarrolló un componente independiente que se comunica con diversos aplicativos, no está dentro de una diseño de microservicios por que la Entidad Estatal no tiene la capacidad técnica e infraestructura para la orquestación de microservicios, es allí donde se rescata una de las bondades que tiene la CA, que se puede posponer decisiones tecnológicas y en cuanto haya la infraestructura de microservicios la migración sea fácil y transparentes.

3.5 Actividades orientadas a descubrir las Clean Architecture

Para el desarrollo del Sprint Backlog se consideró la clasificación de las tareas según las capas de Clean Architecture, en ese sentido se realizó la agrupación de las capas por su contexto en el negocio, las cuales están dividido por capas del diseño de la arquitectura limpias las cuales consta de capa de dominio, capa de casos de usos, capa de controladores, así como la capa de infraestructura [9]. Una vez clasificado y agrupado las actividades definidas en a la siguiente tabla. Este Sprint backlog fue verificado y autorizado para su ejecución. Por motivos de espacio se reproduce solo una parte de una tabla con mayor contenido.

Table 3. Definición de arquitectura de Clean Architecture en capas

Módulo	Capas	Casos de Usos
Arquetipo de la arquitectura Clean Architecture		
Administrative-Request	Domain	
	Application	Create, Search, CustomResearch, SendProcedure, AttatRequrement, CorrectObservatio, SendEmail, CallApiRestServicesSGD, CallApiRestServicesSGD, CallApiRestServicesROP, CallApiRestServicesSecurity, SearchRequirementByAdministrativeProcedureId, CreateRequirement
	Controller	GetController,PutController,PostController
	Infrastructure	….

El proyecto para el caso de estudio estuvo dividido en 2 Sprint. En el primer Sprint el portal permitió registrar solicitudes del ciudadano vía internet, adjuntar la documentación requeridos según el tipo de solicitud, así como también la validación del pago realizado en la entidad estatal, pero en el contexto de crisis sanitaria del COVID-19 se requirió la integración con una pasarela de pago estatal y los pagos realizados en un Banco estatal; una vez identificado el nuevo requerimiento se añadió al Sprint Backlog las validaciones para los pagos vía WebApi con la pasarela de pago y banco estatal.

Para el segundo Sprint el portal digital, tendría la funcionalidad de revisión, observación, subsanación de observaciones, además de las validaciones de los pagos externos a la entidad estatal, para ello el equipo de desarrollo organizó las actividades según las capas de CA, considerando en el diseño los principios SOLID.

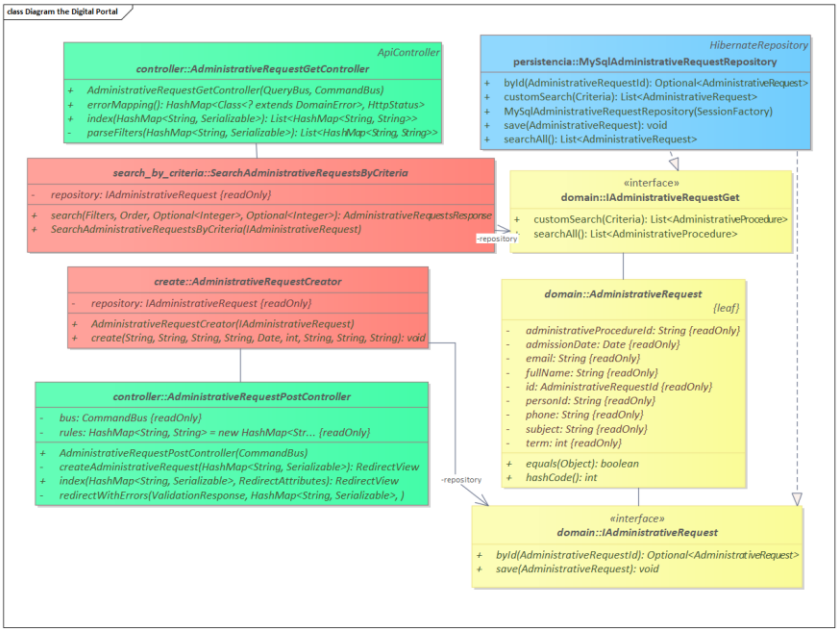

Fig. 2. Diagrama de clase reflejando las capas de Clean Architecture

SCRUM-CA ayudó en el desarrollo del caso de estudio, brindado las buenas prácticas para desarrollar una arquitectura que sea mantenible, así como con la definición de las tareas en el sprint backlog, para facilitar en el desarrollo de la construcción del software.

Para validar la mantenibilidad del software Portal Digital se realizó con la herramienta SonarQube, la misma que proporciona métricas de mantenibilidad, duplicidad, seguridad, etc. En la evaluación se consideró 342 archivos que contienen 5503 línea de código, cuyos resultados emitidos por SonarQube fueron:

La primera medición sobre duplicidad de código: se obtuvo un 4.80% de líneas duplicadas (porcentaje de duplicidad = cantidad de líneas duplicadas /cantidad de líneas* 100) así como 31 bloques duplicados, para proyectos java se acepta hasta un 3% de código duplicados y/o 10 bloques duplicados que es al que se debe llegar en la siguiente iteración. Con respecto a la mantenibilidad: se obtuvo una ratio de deuda técnica de 6.3%, según la escala de clasificación de mantenibilidad de SonarQube (<= 5% la calificación es A, entre el 6 y el 10% la calificación es B, entre el 11 y el 20% la calificación es una C, entre el 21 y el 50% la calificación es D, cualquier valor por encima del 50%

es una E) se entiende a menor ratio de deuda técnica mejor es el estado de salud del software,

Para la segunda medición el porcentaje de duplicidad de código paso a 3.5% y 9 bloques duplicados y con respecto a la mantenibilidad el ratio deuda técnica fue de 1.3% del código analizado, por lo tanto SCRUM-CA influyó en la mantenibilidad del software. Estas mejoras en la mantenibilidad de explican por una mejor comprensión y aplicación del flujo de trabajo del marco de trabajo Scrum-CA

Al final de Sprint se llevó acabo la reunión para realizar mejoras al proceso llevado a cabo, así como las revisiones de lecciones aprendidas (Sprint retrospective) y en las revisiones técnicas (Sprint review) capturando nuevos requerimientos y realimentado el Product Backlog y refrescando el contexto de negocio, en la cual se creó un plan para evitar la generación de anti patrones.

Conclusiones

En este caso de estudio se propone ampliar el marco de trabajo de SCRUM con prácticas de Clean Architecture para incorporar las buenas prácticas de diseño de arquitectura en el proceso de construcción de software que refleje el contexto del negocio, además este centrado en los caso de uso del negocio, así mismo para evitar el alto grado de complejidad que muchas veces se llega cuando se requiere cumplir con la implementación SCRUM y no se tiene un diseño que delimite el desarrollo del software para asegurar el mantenimiento así como soportar los constantes cambios que se presentan durante el desarrollo del proyecto, se propone que las prácticas de Clean Architecture guie el diseño de una arquitectura limpia, delimitado por su contexto permitiendo que haya una alta cohesión y por consiguiente bajar el acoplamiento de los componentes así como la facilidad de probar cada componente aun sin la necesidad de depender de una IU.

En el caso de estudio se ha extendido el marco de trabajo Scrum con prácticas de arquitectura limpia de Clean Architecture para mejorar la mantenibilidad de un software desarrollado para una entidad estatal peruana. Lo resultados concluyen que dicha mejora, del Sprint1 al Sprint2 en el proyecto, se baja la duplicidad de código de 4.8% a 3.5%, de 31 bloques duplicados a 9 y una deuda técnica de 6.3% a 1.3%.
El marco de trabajo Scrum extendido con prácticas Clean Architecture, por un lado, ha seguido un diseño metodológico lógico y validable en futuras experiencias a esta investigación o incluso extrapolarlas a otros contextos en lo posible; por otro lado, se ha cumplido con todos sus requerimientos (Tabla 1), y cuyos puntos de extensión o especificación se ha logrado a través de la identificación de aquellos aspectos que permitan que Scrum apuntale el logro de software mejor mantenible (Tabla 2).

La extensión del marco de trabajo SCRUM, con prácticas de Clean Architecture orienta el desarrollo de software hacia la mantenibilidad, en base a la inclusión del contexto de

negocio como artefacto, la clasificación de los sprint backlog por bloques arquitecturales de Clean Architecture, y el énfasis en el sprint Review y el sprint Retrospective.

Esta investigación por último concluye, que los resultados de este estudio son perfectibles en términos de que se necesitan más ejecuciones en otros proyectos para estandarizar el marco de trabajo.

Referencias

[1] D. F. Orellana-Daube, "El efecto global de la actual revolución tecnológica 4ª revolución industrial y la industria 4.0 en acción," *Rev. GEON (Gestión, Organ. y Negocios)*, vol. 7, no. 2, 2020, doi: 10.22579/23463910.194.

[2] A. Isaksen, M. Trippl, N. Kyllingstad, and J. O. Rypestøl, "Digital transformation of regional industries through asset modification," *Compet. Rev.*, 2020, doi: 10.1108/CR-12-2019-0140.

[3] C. C. Hsu, R. H. Tsaih, and D. C. Yen, "The evolving role of IT Departments in digital transformation," *Sustain.*, vol. 10, no. 10, 2018, doi: 10.3390/su10103706.

[4] Riquelme Matias, "Mejora Continua (Proceso, Importancia Y Caraterísticas)," *Web y Empres.*, 2020.

[5] Y. D. Amaya Balaguera, "Metodologías ágiles en el desarrollo de aplicaciones para dispositivos móviles. Estado actual," *Rev. Tecnol.*, vol. 12, no. 2, 2015, doi: 10.18270/rt.v12i2.1291.

[6] M. D. Papamichail and A. L. Symeonidis, "A generic methodology for early identification of non-maintainable source code components through analysis of software releases," *Inf. Softw. Technol.*, vol. 118, 2020, doi: 10.1016/j.infsof.2019.106218.

[7] Ian Sommerville, *Ingeniería de Software*. 2011.

[8] M. Staron, W. Meding, C. Hoglund, P. Eriksson, J. Nilsson, and J. Hansson, "Identifying implicit architectural dependencies using measures of source code change waves," 2013, doi: 10.1109/SEAA.2013.9.

[9] R. C. Martin, *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. 2017.

[10] Digital.ai, "14th State of Agile," 2020, [Online]. Available: https://explore.digital.ai/state-of-agile/14th-annual-state-of-agile-report.

[11] K. Schwaber and J. Sutherland, "La Guía de Scrum. La Guía Definitiva de Scrum: Las Reglas del Juego.," *scrum.org*, p. 22, 2017, [Online]. Available: http://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-Spanish-SouthAmerican.pdf#zoom=100.

[12] E. Hossain, M. Ali Babar, and H. Y. Paik, "Using scrum in global software development: A systematic literature review," in *Proceedings - 2009 4th IEEE International Conference on Global Software Engineering, ICGSE 2009*, 2009, pp. 175–184, doi: 10.1109/ICGSE.2009.25.

[13] J. E. Martínez, A. F. Gómez, and F. J. Pino, "Generando productos software mantenibles desde el procesode desarrollo: El modelo de referencia MANTuS," 2016. doi: 10.4067/S0718-33052016000300007.

[14] T. J. Gandomani, Z. Tavakoli, M. Z. Nafchi, and M. Najafi Sarpiri, "Adapting Scrum Process with 7C Knowledge Management Model," *2019 IEEE 5th Conf. Knowl. Based Eng. Innov. KBEI 2019*, pp. 56–59, 2019, doi: 10.1109/KBEI.2019.8735008.

[15] M. Alotaibi, "Modelling Security Requirements Through Extending Scrum Agile Development Framework," no. February, 2016.

[16] R. L. Nord, I. Ozkaya, and P. Kruchten, "Agile in distress: Architecture to the rescue," *Lect. Notes Bus. Inf. Process.*, vol. 199, 2014, doi: 10.1007/978-3-319-14358-3_5.

[17] U. Dayanandan and K. Vivekanandan, "An empirical evaluation model for software architecture maintainability for object oriented design," *ACM Int. Conf. Proceeding Ser.*, vol. 25-26-Augu, pp. 1–4, 2016, doi: 10.1145/2980258.2980459.

[18] K. Alkharabsheh, Y. Crespo, E. Manso, and J. A. Taboada, "Software Design Smell Detection: a systematic mapping study," *Softw. Qual. J.*, vol. 27, no. 3, pp. 1069–1148, 2019, doi: 10.1007/s11219-018-9424-8.

[19] A. Debski, B. Szczepanik, M. Malawski, S. Spahr, and D. Muthig, "A scalable, reactive architecture for cloud applications," *IEEE Softw.*, vol. 35, no. 2, 2018, doi: 10.1109/MS.2017.265095722.

[20] S. C. Misra, V. Kumar, and U. Kumar, "Identifying some important success factors in adopting agile software development practices," *J. Syst. Softw.*, vol. 82, no. 11, pp. 1869–1890, 2009, doi: 10.1016/j.jss.2009.05.052.

[21] "Manifesto for Agile Software Development." .

[22] R. F. Dan Turk and B. Rumpe, "Limitations of Agile Software Processes," *J. Environ. Prot. Ecol.*, vol. 18, no. 3, pp. 1259–1267, 2002.

[23] T. Sedano, P. Ralph, and C. Peraire, "The Product Backlog," *Proc. - Int. Conf. Softw. Eng.*, vol. 2019-May, no. Section IV, pp. 200–211, 2019, doi: 10.1109/ICSE.2019.00036.

[24] A. M. Alsalemi and E. T. Yeoh, "A survey on product backlog change management and requirement traceability in agile (Scrum)," *2015 9th Malaysian Softw. Eng. Conf. MySEC 2015*, pp. 189–194, 2016, doi: 10.1109/MySEC.2015.7475219.