Centro de Enseñanza Técnica y Superior

Con reconocimiento de validez oficial de estudios del Gobierno del Estado de Baja California según Acuerdo de fecha 10 de octubre de 1983

“MODERNIZACIÓN DE LA HERRAMIENTA DE CONFIGURACIÓN DE CLIENTES, APLICANDO METODOLOGÍA SCRUM”

Tesis/Proyecto de aplicación

para cubrir parcialmente los requisitos necesarios para obtener el grado de Maestro en Ingeniería e Innovación

Presenta:

César Gustavo Guzmán Flores

Director:

M.C. Amanda Nieto Sánchez
Centro de Enseñanza Técnica y Superior (CETYS Universidad)

Ensenada, Baja California, México,
5 de enero de 2020

MODERNIZACIÓN DE LA HERRAMIENTA DE CONFIGURACIÓN DE CLIENTES, APLICANDO METODOLOGÍA SCRUM

Tesis/Proyecto de aplicación que para obtener el grado de Maestro en Ingeniería e Innovación

Presenta:

César Gustavo Guzmán Flores

y aprobada por el siguiente Comité

_____

M.C. Amanda Nieto Sánchez
Director de tesis

_____

Mtra. Lucía Beltrán Rocha
Codirector de tesis

_____

Dra. Dalia Holanda Chávez García
Miembro del comité

Resumen de la tesis que presenta César Gustavo Guzmán Flores como requisito parcial para la obtención del grado de Maestro en INGENIERÍA E INNOVACIÓN.

MODERNIZACIÓN DE LA HERRAMIENTA DE CONFIGURACIÓN DE CLIENTES, APLICANDO METODOLOGÍA SCRUM

Resumen aprobado por:

_____
M.C. Amanda Nieto Sánchez

El presente documento describe los principales conceptos y fundamentos de la metodología Scrum y su aplicación en un proyecto real, se presenta el contexto general del sistema de Herramienta de Configuración de Clientes (Customer Setup Tool) que se pretende migrar de su actual arquitectura(monolítica) a un modelo de Microservicios.

Palabras clave: Scrum, metodologías agiles, microservicios, servicios web, angular**.**

Abstract of the thesis presented by César Gustavo Guzmán Flores as a partial requirement to obtain the Master or Doctor of Science degree in ENGINEERING AND INNOVATION with orientation in

MODERNIZACIÓN DE LA HERRAMIENTA DE CONFIGURACIÓN DE CLIENTES, APLICANDO METODOLOGÍA SCRUM

Abstract approved by:

_____

M.C. Amanda Nieto Sánchez

This document describes the main concepts and foundations of Scrum methodology and it's application in a real project, presents the general context of the Customer Setup Tool system that is intended to migrate from its current architecture to a model of Microservices.

Keywords: Scrum, agile methodologies, microservices, web services, angular.

Dedicatoria

A mi tío y mentor M.C. Antonio Luis Álvarez Oval por ser fuente de motivación y guía, por ser ejemplo para seguir como profesionista y humano.

Agradecimientos

Al CONACYT por el apoyo económico brindado mediante el programa CONACYT para la Industria.
A CETYS Universidad campus Ensenada, por abrirme sus puertas y brindarme la oportunidad de continuar mis estudios de Postgrado.
A la Dra. Dalia Chávez por el apoyo y las facilidades para continuar la maestría aún con las barreras de la distancia.
A la M.C. Amanda nieto y Mtra. Lucía Beltrán por su tiempo y dedicación en la asesoría y revisión de la presente investigación.

Tabla de contenido

Lista de figuras

Lista de tablas

1. Introducción

Staples Inc. es una compañía que suministra productos básicos para negocios, impresoras, tinta, computadoras, muebles de oficina, servicios de impresión, productos promocionales, etc., al por menor y por mayor con sede en Framingham, Massachusetts. La Herramienta de Configuración del Cliente (del inglés Customer Setup Tool) es un sistema Negocio a Negocio (B2B del inglés Business to Business)[1] de administración de clientes empresariales de Staples, fue desarrollado con tecnología Progress, esta tecnología ya es obsoleta, y esto implica dos problemas, primero debido a la escasez de mano de obra especializada en esta tecnología, no habrá soporte para este sistema, en segundo la corporación está moviendo sus sistemas hacia una arquitectura de microservicios (del inglés microservices) y el diseño actual del sistema no se alinea a esta arquitectura.

Los microservicios son una tendencia reciente en el diseño de software. Una arquitectura de microservicio simplifica el desarrollo de sistemas de escalamiento horizontal complejos que son altamente flexible, modulares e independientes del lenguaje. Definimos un microservicio como un pequeño servicio autónomo especializado que se comunica a través de un límite de red (Otterstad & Yarygina, 2017).

El tiempo que demora un usuario en registrar/actualizar información de un cliente es de aproximadamente 15 minutos, esta actividad es recurrente alrededor de 20 veces al día con tendencia a incrementar de acuerdo con las encuestas con los operadores. Se requiere aumentar la capacidad de atención a clientes con el mismo personal capacitado al eliminar tiempos muertos generados por la interfaz de usuario (UI) actual.

El tiempo de instalación del software por cada actualización toma alrededor de 20 minutos por máquina, se cuentan con alrededor de 200 usuarios operativos ubicados en diversas ciudades de USA, lo cual representa un inconveniente a la hora de actualizar este sistema.

[1] Negocio a negocio (del inglés business-to-business o B2B) hace referencia a las transacciones comerciales entre empresas.

En ambiente de producción de este sistema necesita 5 servidores físicos y alrededor de 12 servidores de aplicaciones, aumentar la cantidad de servidores solo ayuda para mitigar la disponibilidad del sistema, pero no mejora la latencia[2] de hasta 30 segundos en muchas peticiones del sistema, resultando en una mala experiencia de uso para el usuario.

1.1 Antecedentes

1.1.1 Netflix y el escalamiento de sus sistemas

En 2008, Adrian Cockcroft[3], en ese entonces arquitecto de la nube de Netflix entendió que Netflix necesitaba alejarse de los centros de datos monolíticos de escala vertical y avanzar hacia una arquitectura de microservicios basada en la nube. Los beneficios potenciales en una revisión estructural eclipsaron crudamente cualquier desventaja previsible. En tan solo 8 años Netflix se había convertido en una de las primeras grandes corporaciones en existir completamente en la nube pública. Este cambio arquitectónico reveló mejoras significativas en el rendimiento, el desarrollo y la escalabilidad, lo que dio como resultado un notable período de crecimiento. El éxito de Netflix con una arquitectura de microservicios basada en la nube ha sido tan notable que, en retrospectiva, es difícil imaginar que Netflix se haya movido en otra dirección (Cockroft, Hicks , & Orzell, 2011).

Técnicamente, debería ser posible crear módulos independientes bien factorizados dentro de un solo proceso monolítico. Y, sin embargo, rara vez vemos que esto suceda. Los módulos en sí mismos pronto se unen estrechamente con el resto del código, renunciando a uno de sus beneficios clave. Tener una separación de límite de procesos impone una higiene a este respecto (o al menos hace que sea más difícil hacer lo incorrecto). Newman no sugiere que este sea el principal impulsor de la separación del

[2] Se define como el tiempo que ocurre entre que envías una petición hasta que recibes el primer bit de respuesta.

[3] Adrian Cockcroft es vicepresidente de estrategia de arquitectura en la nube en Amazon Web Services https://www.oreilly.com/talent/a9220-adrian-cockcroft

proceso, por supuesto, pero es interesante que las promesas de separación modular dentro de los límites del proceso rara vez se cumplan en el mundo real (Newman, 2015).

Entonces, aunque la descomposición modular en un límite de procesos es posible, descomponer el sistema en servicios, por sí solo no ayudará a resolver todo.

1.1.2 Metodología ágil distribuido

Los proyectos de globalización de software a menudo satisfacen no solo los desafíos internos, sino también los externos, como los horarios ajustados y cambiantes, el avance del alcance, los presupuestos inflexibles y las expectativas poco realistas de los clientes. Estos desafíos generalmente se deben a la falta de conocimiento y apoyo del proceso por parte de las divisiones de desarrollo, ventas y administración (Flarup, 2007). Con el desarrollo global del software, los proyectos distribuidos globalmente se están convirtiendo en la norma para los grandes sistemas de software. Si bien esta es una tendencia en la industria de TI, también hay muchas organizaciones que ahora se dan cuenta de que los procesos pesados tradicionales son lentos y carecen de interacción entre los clientes y los equipos de desarrollo de productos (Schwaber, 2007).

El grupo Global Mi Yahoo! experimentó los mismos desafíos mencionados anteriormente, cuando lanzaron 17 versiones internacionales de la antigua My Yahoo! ‘Zorro’ entre 2005 y 2006. El equipo de desarrollo de productos en Sunnyvale, California, proporcionó a los equipos internacionales una plataforma poco internacionalizada y un enorme manual de instalación internacional junto con una lista de verificación de instalación. Los equipos internacionales se distribuyeron en Asia Pacífico, Europa y las Américas y fueron responsables de localizar el producto global. El equipo de desarrollo de productos se distribuyó en Sunnyvale y Bangalore, y apoyó el trabajo de localización principalmente en línea. Cada proyecto de localización demoró entre 6 y 12 meses y dependió de una serie de módulos y recursos locales para implementar cada versión localizada de Zorro. Como resultado, el proyecto de despliegue internacional para Zorro no tenía suficientes recursos ni suficiente soporte, y no funcionó de manera eficiente después de todo el esfuerzo realizado. Aquí es donde los métodos ágiles demostrarán ser un beneficio.

El informe “Despliegue ágil en una gran empresa” (Benefield, 2008) presenta un estudio de caso industrial que muestra prácticas exitosamente integradas de desarrollo de software ágil (ASD) y desarrollo de software distribuido (DSD).

El equipo de producto global en Sunnyvale había adoptado Scrum y Agile en 2006. El equipo de producto global y los equipos internacionales lanzaron las siguientes versiones de My Yahoo! "Camaleón" para EE. UU. Y 17 países internacionales entre 2007 y 2008. Se ha logrado una gran mejora tanto en el proceso como en la calidad del producto. La plataforma Chameleon ha fortalecido la función de configuración y programación local más allá de las nuevas características de lujo. Además, el rendimiento general y la satisfacción con el despliegue internacional de Chameleon en comparación con el Zorro aumentaron en más del 30%, aunque solo se siguieron algunas prácticas ágiles. (Benefield, 2008)

1.2 Justificación

Cada día escasea más el personal capacitado en tecnología Progress y conforme pasa el tiempo aumenta el riesgo de dejar sin soporte al sistema actual, esto obliga a actualizar la tecnología donde exista mayor oferta de mano de obra profesional e incluso disminuir en menor medida costos en salarios.

La interfaz de usuario existente hace perder alrededor de 3 mil horas hombre semanales por tiempos muertos en latencia y fallas del sistema.

De acuerdo con el reporte de Modernization-Backend retirement plan de Staples (Staples Inc., 2019), los beneficios de esta migración se reflejan en más de 500mil dólares de ahorro de EBITDA[4], es por esto por lo que modernizar este sistema es relevante.

[4] El **EBITDA** es un indicador financiero que hace referencia a las ganancias de la compañía antes de intereses, impuestos, depreciaciones y amortizaciones. Se puede entender, pues, que el **EBITDA** es el beneficio bruto de explotación calculado antes de la deducibilidad de los gastos financieros

1.3 Hipótesis

Utilizando la metodología de desarrollo ágil Scrum incrementa la productividad en los equipos de trabajo en al menos 20%.

1.4 Objetivos

1.4.1 Objetivo general

- Describir el grado de impacto en la modernización (migración) de Customer Setup Tool, al adoptar una metodología ágil para la gestión de proyectos de software.

1.4.2 Objetivos específicos

- Analizar las metodologías de desarrollo de software Scrum y tradicional de cascada
- Realizar una comparativa entre la metodología tradicional cascada versus Scrum
- Adoptar la metodología ágil Scrum a la gestión de un proyecto real
- Describir el impacto en la productividad y competitividad después de adoptar Scrum

2. Marco teórico

2.1 Metodología cascada (Waterfall model)

El modelo en cascada es un proceso secuencial de etapas a seguir para el desarrollo de un producto, este modelo se originó para usar en las industrias de construcción y manufactura, aunque la primera descripción formal de este modelo fue en un artículo realizado por Winston W Royce (Winston, 1970) y a consecuencia de no existir un modelo para el desarrollo de software se adoptó este como el primer modelo. La figura 1 muestra las etapas de este proceso o ciclo de vida.

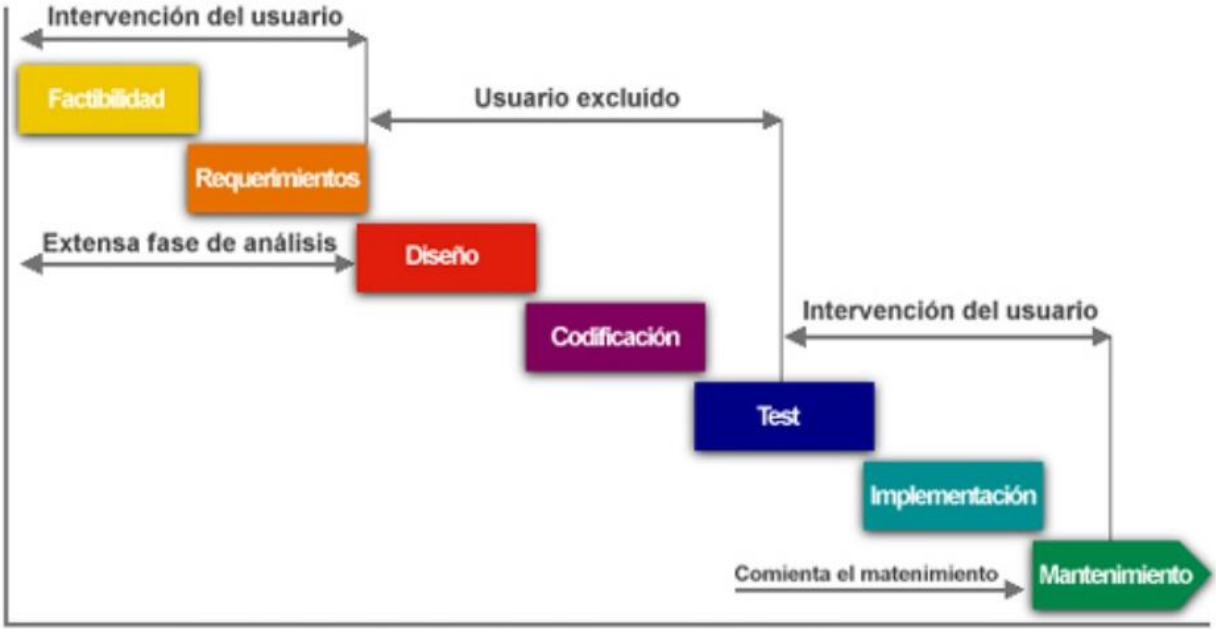

Figura 1. Modelo cascada. Quiroz Gustavo (2012). Métodos ágiles, pasado, presente y futuro. Recuperado de https://www.slideshare.net/gquiroz/mtodos-agiles-pasado-presente-y-futuro

Análisis de requerimientos.

Esta etapa consiste en recabar la información relacionada a la solución deseada y a las necesidades del usuario final. Este análisis incluye la definición del problema, las metas y un claro entendimiento del negocio y sus reglas, así como también identificar las funciones que el producto final debe considerar. Algunas de las técnicas a utilizar para reunir esta información se basan en entrevistas, observaciones, creación de prototipos,

casos de usos y lluvia de ideas. Los resultados obtenidos son plasmados en un documento conocido como Especificación de Requerimientos de Software (SRS), el cual servirá como entradas para la siguiente etapa del modelo. Los requerimientos descritos en el SRS deben considerar las funcionales, no funcionales y la descripción de comportamientos (Leffingwell & Widrig, 2003).

Diseño.

Esta etapa toma como inicio el SRS de la etapa anterior y en base a ello define como el software debe ser construido. El diseño del software consiste en elaborar la arquitectura de software y hardware necesaria, especificar parámetros de rendimiento y segundad, seleccionar la plataforma y el lenguaje de desarrollo, gestión de recursos y conectividad de las interfaces. También en esta etapa se define el diseño de la interfaz de usuario, la navegación y el acceso al sistema. La salida de esta etapa es un documento llamado Descripción del Diseño del Software (SDD) el cual servirá de entrada a la siguiente etapa del modelo (Society, 1998).

Codificación.

En esta etapa es donde inicia la construcción del producto tomando como base el SDD y el SRS de las etapas anteriores. En esta etapa el trabajo es realizado por los desarrolladores, diseñadores de interfaces y otros expertos usando como herramientas las plataformas y lenguajes definidos en el SDD. La salida de esta etapa es el producto y componentes construidos.

Pruebas.

En esta etapa se realizan las pruebas para cada componente y pruebas al producto integrado. Los casos de prueba son escritos en un documento los cuales evalúan si el sistema cumple con la definición inicial de los requerimientos. Las pruebas pueden ser categorizadas en pruebas unitarias, pruebas integrales y pruebas de aceptación. Cualquier defecto encontrado deberá ser notificado a los desarrolladores para que sea reparada. En esta etapa se deberá generar el manual de usuario (Mathur, 2011).

Despliegue.

En esta etapa se prepara el sistema o producto para la instalación y uso en un ambiente productivo. El resultado es un documento de instalación que define las configuraciones y pasos a ejecutar para instalar el producto.

Mantenimiento.

Esta última etapa del modelo consiste en realizar ajustes, mejoras y reparación de defectos al sistema o componentes individuales para mejorar el producto o el rendimiento de este.

Una característica de este modelo es que para continuar con la fase siguiente se debe completar la fase predecesora y cuando surgen cambios en una de las fases es necesario retornar a las fases anteriores para aplicar los nuevos cambios. Este tipo de modelo es efectivo cuando no suceden cambios después de finalizar cada etapa de otra manera se tendrá que retornar, lo que produce el efecto de proyectos que nunca se concluyen.

Otra característica de este modelo es que es obligatoria la generación de documentación extensa con la finalidad de generar conocimiento al final de cada etapa, pero nuevamente si existe algún cambio se deberá actualizar toda la documentación generada, lo que requiere invertir grandes cantidades de tiempo en documentación que generalmente nadie lee.

En general este modelo de proceso para la construcción de software puede ser adecuado para proyectos donde los requisitos no cambian y existe un alto grado de predictibilidad del producto final.

En la compañía de tecnología de información donde actualmente se realiza este estudio, muchos proyectos siguen este modelo y muchas de las problemáticas que se presentan en los proyectos son los siguientes:

- Proyectos que se concluyen, pero al final el producto ya no coincide con el proceso
- Después de varios meses sin ver resultados se pierde el interés por parte del cliente
- Desconfianza para la realización de proyectos
- Equipos de trabajo desmotivados

2.2 Metodologías ágiles

En el año 2001 varios expertos que había creado y probado las metodologías ágiles se reunieron con el objetivo de acordar y definir valores y principios que ayudarían a los equipos de trabajo a desarrollar software de manera eficiente, rápida y con adaptación ante los cambios, este documento se conoce como el "Manifiesto ágil"[5] el cual da el verdadero significado a la palabra ágil y consiste en 4 valores:

- Individuos y sus interacciones sobre procesos y herramientas
- Software funcionando sobre documentación extensa
- Colaboración con el cliente sobre negociaciones contractuales
- Respuesta al cambio sobre seguir un plan

Estos valores conforman la médula principal de cualquier metodología ágil y marca la diferencia sobre las metodologías tradicionales.

2.3 Scrum

Ken Shwaber y Jeff Sutherland presentaron conjuntamente por primera vez Scrum en la conferencia COPSLA en 1995[6]. Scrum es un marco de trabajo para el desarrollo y el mantenimiento de productos complejos que ha sido utilizado desde los años 90, en el cual las personas pueden afrontar complejos problemas adaptativos, a la vez que entregan productos del máximo valor posible de forma productiva y creativa.

El marco de trabajo de Scrum consiste en los equipos Scrum y en sus roles, eventos, artefactos y reglas asociadas. Cada componente dentro del marco de trabajo sirve a un propósito específico y es esencial para el éxito de Scrum y pasa su uso. La figura 2 muestra los elementos que componen la metodología Scrum.

[5] Fuente: https://agilemanifesto.org/iso/es/manifesto.html leído 20 de abril de 2019
[6] Fuente: https://www.scrumguides.org/ leído el 28 de mayo de 2019

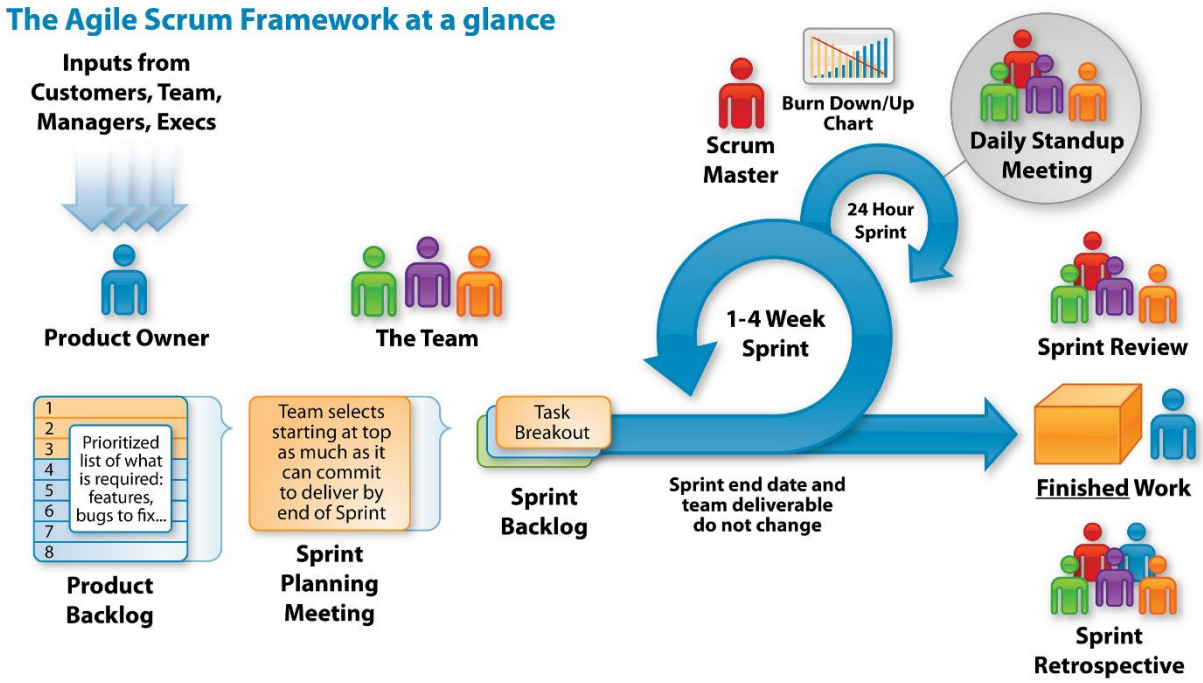

Figura 2. Modelo Scrum. Neon Rain Interactive (2019). AGILE SCRUM FOR WEB DEVELOPMENT. [Figura]. Recuperado de https://www.neonrain.com/agile-scrum-web-development/

Scrum se fundamenta en la teoría empírica de control de procesos o empirismo. El empirismo asegura que el conocimiento procede de la experiencia y de tomar decisiones basándose en lo que se conoce. Scrum emplea una aproximación iterativa e incremental para optimizar la predictibilidad y controlar el riesgo.

Roles.

El equipo Scrum consiste en un dueño de producto (product owner), el equipo de desarrollo (development team) y un Scrum master. Los equipos Scrum son autoorganizados y multifuncionales. Los equipos autoorganizados eligen la mejor forma de llevar a cabo su trabajo y no son dirigidos por personas externas al equipo. Los equipos multifuncionales tienen todas las competencias necesarias para llevar a cabo el trabajo sin depender de otras personas que no son parte del equipo. El modelo de equipo en Scrum está diseñado para optimizar la flexibilidad, la creatividad y la productividad. El equipo de Scrum ha demostrado ser cada vez más efectivo para todos los usos anteriores y cualquier trabajo complejo. Los equipos Scrum entregan productos de forma iterativa e incremental, maximizando las oportunidades de obtener retroalimentación. Las entregas

incrementales del producto “terminado” aseguran que siempre estará disponible una versión potencialmente útil y funcional del producto.

El dueño del producto (Product Owner)

Es el responsable de maximizar el valor del producto resultante del trabajo del equipo de desarrollo. El cómo se lleva a cabo esto podría variar ampliamente entre distintas organizaciones, equipos Scrum e individuos. El dueño del producto es la única persona responsable de gestionar la lista del producto (product backlog). La gestión de la lista del producto incluye:

- Expresar claramente los elementos de la lista del producto
- Ordenar los elementos en la lista del producto para alcanzar los objetivos y misiones de la mejor manera posible
- Optimizar el valor del trabajo que el equipo de desarrollo realiza
- Asegurar que el product backlog es visible, transparente, clara para todos y que muestra aquello en lo que el equipo trabajará a continuación
- Asegurar que el equipo de desarrollo entiende los elementos de la lista del producto al nivel necesario

El dueño del producto podría representar los deseos de un comité en el product backlog, pero aquellos que quieran cambiar la prioridad de un elemento de la lista deben hacerlo a través del dueño de producto.

El equipo de desarrollo (development team)

El equipo de desarrollo consiste en los profesionales que realizan el trabajo de entregar un incremento de producto “terminado” que potencialmente se pueda poner en producción al final de cada iteración (del inglés sprint). Una iteración “terminada” es obligatoria en la revisión del sprint. Solo los miembros del equipo de desarrollo participan en la creación del sprint. La organización es la encargada de estructurar y empoderar a los equipos de desarrollo para que estos organicen y gestionen su propio trabajo. La

sinergia resultante optimiza la eficiencia y efectividad del equipo de desarrollo (Gonçalves, 2018).

El Scrum Master

Es responsable de promover el proceso ayudando a todos a entender la teoría, prácticas, reglas y valores de Scrum. Es un líder que está al servicio del equipo Scrum, proporciona los siguientes servicios:

- Asegurar que los objetivos, el alcance y el dominio del producto sean entendidos por todos en el equipo de la mejor manera posible
- Asegurar que el dueño de producto conozca cómo ordenar la lista de producto para maximizar el valor
- Facilitar los eventos de Scrum según se requiera o necesite
- Guiar al equipo de desarrollo en ser autoorganizado y multifuncional
- Eliminar impedimentos para el progreso del equipo de desarrollo

Eventos de Scrum

Existen eventos predefinidos con el fin de crear regularidad y minimizar la necesidad de reuniones no definidas en Scrum. Todos los eventos son bloques de tiempo (time-boxes), de tal modo que todos tienen una duración máxima. Una vez que comienza un sprint, su duración es fija y no puede acortarse o alargarse. Los demás eventos pueden terminar siempre que se alcance el objetivo del evento, asegurando que se emplee una cantidad apropiada de tiempo sin permitir desperdicio en el proceso.

El sprint

El corazón de Scrum es el sprint, es un bloque de tiempo (time-box) de un mes o menos durante el cual se crea un incremento del producto “terminado” utilizable y potencialmente desplegable. Es más conveniente si la duración de los sprints es consistente a lo largo del esfuerzo de desarrollo. Cada nuevo sprint comienza inmediatamente después de la finalización del sprint anterior. Los sprints contienen y consisten en la planificación del sprint (sprint planning), los scrums diarios (daily scrums), el trabajo de desarrollo, la

revisión del sprint (sprint review), y la retrospectiva del sprint (sprint retrospective). Durante el sprint:

- No se realizan cambios que puedan afectar al objetivo del sprint (sprint goal);
- Los objetivos de calidad no disminuyen;
- El alcance puede clarificarse y renegociarse entre el dueño de producto y el equipo de desarrollo a medida que se va aprendiendo más.

Planificación del sprint (sprint planning)

El trabajo por realizar durante el sprint se planifica en la planificación del sprint. Este plan se crea mediante el trabajo colaborativo del equipo Scrum completo. La planificación de sprint tiene un máximo de duración de ocho horas para un sprint de un mes. Para sprints más cortos el evento es usualmente más corto. La planificación del sprint responde a las siguientes preguntas:

- ¿Qué puede entregarse en el Incremento resultante del sprint que comienza?
- ¿Cómo se conseguirá hacer el trabajo necesario para entregar el incremento?

Scrum diario (daily scrum)

Es una reunión con un bloque de tiempo de 15 minutos para el equipo de desarrollo. El Scrum diario se lleva a cabo cada día del sprint. En él, el equipo de desarrollo planea el trabajo para las siguientes 24 horas. El Scrum diario se realiza a la misma hora y en el mismo lugar todos los días para reducir la complejidad. Algunos equipos de desarrollo usarán preguntas, algunos se basarán más en discusiones. Aquí hay un ejemplo de lo que podría usarse:

- ¿Qué hice ayer que ayudó al equipo de desarrollo a lograr el objetivo del sprint?
- ¿Qué haré hoy para ayudar al equipo de desarrollo a lograr el objetivo del sprint?

- ¿Veo algún impedimento que evite que el equipo de desarrollo o yo logremos el objetivo del sprint?

Los scrum diarios mejoran la comunicación, eliminan la necesidad de realizar otras reuniones, identifican impedimentos a remover relativos al desarrollo, resaltan y promueven la toma rápida de decisiones y mejoran el nivel de conocimiento del equipo de desarrollo.

Retrospectiva de sprint (sprint retrospective)

La retrospectiva de sprint es una oportunidad para el equipo scrum de inspeccionarse a sí mismo y de crear un plan de mejoras que sean abordadas durante el siguiente sprint.

El propósito de la retrospectiva de sprint es:

- Inspeccionar cómo fue el último sprint en cuanto a personas, relaciones, procesos y herramientas;
- Identificar y ordenar los elementos más importantes que salieron bien y las posibles mejoras;
- Crear un plan para implementar las mejoras a la forma en la que el equipo Scrum desempeña su trabajo.

Artefactos de Scrum

Representan trabajo o valor en diversas formas que son útiles para proporcionar transparencia y oportunidades para la inspección y adaptación. Los artefactos definidos por Scrum están diseñados específicamente para maximizar la transparencia de la información clave, necesaria para asegurar que todos tengan el mismo entendimiento del artefacto.

Lista de producto (product backlog)

Es una lista ordenada de todo lo que se conoce que es necesario en el producto. Es la única fuente de requisitos para cualquier cambio a realizarse en el producto. El dueño de producto (product owner) es el responsable de la lista de producto, incluyendo su

contenido, disponibilidad y ordenación. Enumera todas las características, funcionalidades, requisitos, mejoras y correcciones que constituyen cambios a realizarse sobre el producto para entregas futuras. Tienen como atributos la descripción, el orden, la estimación y el valor.

El equipo de desarrollo es el responsable de proporcionar todas las estimaciones. El dueño de producto podría influenciar al equipo ayudándoles a entender y seleccionar sus compromisos, pero las personas que harán el trabajo son las que hacen la estimación final.

Seguimiento del progreso hacia los objetivos

En cualquier momento es posible sumar el trabajo total restante para alcanzar el objetivo. El dueño de producto hace seguimiento de este trabajo restante total al menos en cada revisión de sprint. El dueño de producto compara esta cantidad con el trabajo restante en revisiones de sprint previas, para evaluar el progreso hacia la finalización del trabajo proyectado en el tiempo deseado para el objetivo.

Lista de pendientes del sprint (sprint backlog)

Es el conjunto de elementos de la lista de producto seleccionados para el sprint, más un plan para entregar el Incremento de producto y conseguir el objetivo del sprint. La lista de pendientes del sprint es una predicción hecha por el equipo de desarrollo acerca de qué funcionalidad formará parte del próximo incremento y del trabajo necesario para entregar esa funcionalidad en un incremento “terminado”. La lista de pendientes del sprint hace visible todo el trabajo que el equipo de desarrollo identifica como necesario para alcanzar el objetivo del sprint. La lista de pendientes del sprint es un plan con un nivel de detalle suficiente como para que los cambios en el progreso se puedan entender en el Scrum diario. El equipo de desarrollo modifica la lista de pendientes del sprint durante el sprint y esta lista de pendientes del sprint emerge a lo largo del sprint. Cuando se requiere nuevo trabajo, el equipo de desarrollo lo adiciona a la lista de pendientes del sprint. A medida que el trabajo se ejecuta o se completa se va actualizando la estimación de trabajo restante. Cuando algún elemento del plan se considera innecesario, es eliminado.

Solo el equipo de desarrollo puede cambiar su lista de pendientes del sprint durante un sprint. La lista de pendientes del sprint es una imagen visible en tiempo real del trabajo que el equipo de desarrollo planea llevar a cabo durante el sprint y pertenece únicamente al equipo de desarrollo.

3. Metodología

En el presente capítulo se presenta la descripción de la metodología a seguir para realizar la modernización del sistema Customer Setup Tool aplicando metodología Scrum para responder la pregunta planteada ¿Scrum incrementará la productividad en un 20% con respecto a la metodología en cascada? Y cumplir con los objetivos del proyecto.

Este estudio es no experimental cuantitativo de tipo transeccional o transversal descriptivo, donde se pretende estudiar y analizar la productividad en cada sprint durante el desarrollo del proyecto.

Para ello se elabora un estado del arte para dar sustento a la investigación y el cual es fundamental para entender las aportaciones de este trabajo. En él se presentan y analizan dos metodologías de desarrollo de software, aportando una guía inicial en la investigación y conocimiento para interpretar los resultados del estudio.

Las variables medidas en este estudio se muestran en la tabla 1.

Variables	Sub-variables	Relación
La metodología Scrum		Independiente
Productividad	Cantidad de código Calidad	Dependiente

Tabla 1. Variables a medir

La productividad es la relación entre el producto obtenido y los recursos empleados en su producción. En desarrollo de software, podemos entender la productividad como la cantidad de código que se produce con respecto a un tiempo determinado.
Una de las premisas de Scrum es entregar software de valor en un periodo de no más de 30 días. En la presente investigación la productividad se midió a través de software de valor (historias o stories en inglés) entregadas al final de cada sprint.
En el desarrollo de software la calidad se puede describir de diferentes maneras, para este estudio de investigación la calidad la mediremos en términos de defectos(bugs) levantados y atendidos a través del tiempo.

3.1 Técnica de recolección de datos

La técnica de recolección de datos consistió en la aplicación y uso de la herramienta Jira, en la cual nos apoyamos durante todo el desarrollo del proyecto.

Cada requerimiento, defecto, estatus, estimación y asignación fue capturado esta herramienta, se capturaron puntos de esfuerzo a cada requerimiento y cada dos semanas se hizo el corte de los requerimientos entregados y se asignaban nuevos requerimientos para trabajar en las siguientes dos semanas.

3.2 Población y muestra

La población de este estudio fue un proyecto de software de alto grado de importancia para el negocio. La muestra fue un proyecto que cumpliera con los parámetros de la tabla 2.

DIMENSIÓN	VALOR DESEADO
TAMAÑO	Mediano
DURACIÓN	12 meses
IMPORTANCIA	Alta

Tabla 2. Población y muestra

3.3 Cronograma de actividades

La tabla 3 muestra el cronograma de actividades.

		2018 (mes)		Calendario 2019(mes)											
	Actividad	11	12	1	2	3	4	5	6	7	8	9	10	11	12
1	Documentar metodologías de software de cascada y de SCRUM	X													
2	Planeación del proceso con Scrum	X													
3	Elaborar documentación del proyecto		X												
4	Selección y preparación de herramientas de apoyo		X												

5	Definición del equipo de trabajo	x												
6	Capacitación de la metodología Scrum		x											
7	Elaboración del product backlog		x	x	x									
8	Estimación del proyecto con metodología Cascada					x								
9	Revisión de métricas del sprint		x	x	x	x	x	x	x	x	x	x	x	x
10	Documentar los resultados												x	x

Tabla 3. Cronograma de actividades. Nota. Tabla de elaboración propia

3.4 Planeación del proceso SCRUM

La metodología SCRUM requiere una planeación bien definida, se estableció un equipo de trabajo, unos objetivos y estos objetivos se dividieron en ciclos(iteraciones) de dos semanas, realizando reuniones cortas de 15 minutos diarios para verificar el avance, analizar los problemas y determinar rápidamente un plan de acción para solucionarlos.

3.4.1 Definición del equipo de trabajo

El equipo está distribuido en cuatro localidades, Framingham MA, Englewood NJ, Ciudad de México y Ensenada. El equipo está confirmado por:

Product Owner: Es el dueño del producto, determina los objetivos del producto, este rol fue desempeñado por Tina Cahan, analista de negocios de Staples, con sede en Englewood NJ.

SCRUM Master: Se asegura que las etapas de SCRUM se lleven a cabo, Francisco Nuñez fue el encargado de esta labor, con sede en Ensenada.

Development Team: El equipo de desarrollo se encarga de realizar todo el proceso de creación y pruebas del producto, la tabla 4 muestra el detalle de las personas que desempeñaron estos roles.

Name	Role	Location
César Guzmán	Lead Developer	Ensenada, Mexico
Francisco Nuñez Miranda	Developer	Ensenada, Mexico
Anairene Ishihara	UI Developer	Ensenada, Mexico
Sarahi Flores Carreon	UI Developer	Ensenada, Mexico
Jorge Gonzalez	Developer	Ensenada, Mexico
Ernesto Murillo	Developer	Ensenada, Mexico
Beatriz Perez (Betty)	QA	Mexico City, Mexico
Dutta Arghya	Developer	Framingham, MA

Tabla 4. Equipo de desarrollo

Ceremonias: En la figura 3 se muestra la frecuencia de las ceremonias de la metodología SCRUM aplicada como el Scrum diario, planeación y retrospectiva.

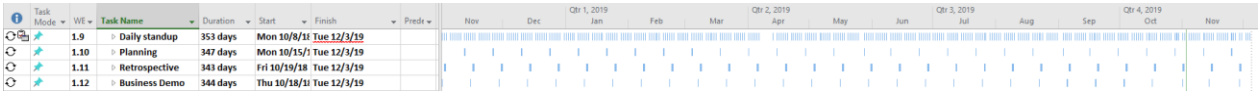

Figura 3. Línea del tiempo de las ceremonias SCRUM. Nota. Figura de elaboración propia.

3.4.2 Definición del backlog

Al inicio del proyecto se definieron alrededor de 600 requerimientos o historias de usuario, con una velocidad de sprint esperada de 35 puntos. Las figuras 4,5 y 6 muestran el conjunto de épicas iniciales que engloban las historias o requerimientos del proyecto.

T	Key	Summary	Assignee	Status
	CAMS1-17	Refactoring and new functionality to support Diversity ShipTo features	*Unassigned*	DONE
	CAMS1-18	Refactoring and changes necessary for the new Palletization process in OMS	*Unassigned*	DONE
	CAMS1-20	Manage ShipTo revision number	*Unassigned*	DONE
	CAMS1-21	Create a new Ship To	*Unassigned*	IN PROGRESS
	CAMS1-24	Add, Maintain, and Validate Contract # assignment on ShipTo Accounts	*Unassigned*	DONE
	CAMS1-26	Reporting Tools/Processes related to Assortment Management Application	*Unassigned*	IN PROGRESS
	CAMS1-185	Migration of SSO solution from SiteMinder to ForgeRock	*Unassigned*	DONE
	CAMS1-295	Small modifications and bug fixes for FY18Q2.	*Unassigned*	DONE
	CAMS1-296	Nexus MVP: Nexus CSS UI - Shared	*Unassigned*	DONE
	CAMS1-297	Displaying and editing Master Account details.	*Unassigned*	DONE
	CAMS1-298	Displaying and editing BillTo details.	*Unassigned*	IN PROGRESS
	CAMS1-299	Displaying and editing ShipTo details.	*Unassigned*	IN PROGRESS
	CAMS1-300	New Master, BillTo, & ShipTo account creation	*Unassigned*	DONE
	CAMS1-321	Q2 Enhancements and Fixes for PCard application.	*Unassigned*	TO DO
	CAMS1-323	Inactive Backlog - Stories not to be worked and we can't reuse	*Unassigned*	TO DO
	CAMS1-485	18Q3 Support and Maintenance Activities for AMH Application	*Unassigned*	DONE

Figura 4. Épicas de los requerimientos capturados en Jira. Nota. Figura de elaboración propia.

CAMS1-505	FY18 Q3 CAMS - ShipTo Maintenance Fixes and Enhancements	*Unassigned*	DONE
CAMS1-488	CAMS Q3 - ShipTo Maintenance Fixes and Enhancements	*Unassigned*	DONE
CAMS1-489	CAMS Q3 - General Tab for Master / BillTo / ShipTo maintenance	*Unassigned*	DONE
CAMS1-491	CAMS Q3 - Web Tab for Master / BillTo / ShipTo maintenance	*Unassigned*	DONE
CAMS1-498	FY18 Q3 CAMS: Allow Tab for Master / BillTo / ShipTo maintenance	*Unassigned*	DONE
CAMS1-500	TBD - Design P-Card application replacement UI	*Unassigned*	DONE
CAMS1-534	Stories that were created in error - Can be recycled	*Unassigned*	DONE
CAMS1-602	19Q2 - CAMS Trans-Ship Tab	*Unassigned*	DONE
CAMS1-604	Creation of Rulesets Tab for Master / Billto	*Unassigned*	DONE
CAMS1-605	FY18 Q4 CAMS Sales Rep Tab for Master / Billto / Shipto	*Unassigned*	DONE
CAMS1-606	19Q2 CAMS Acquisitions Tab	*Unassigned*	DONE
CAMS1-607	FY18 Q4 CAMS Assortment Management Tab for Master / Billto / Shipto	*Unassigned*	DONE
CAMS1-608	FY18 Q4 CAMS Billing/Payment Tab for Master/Billto	*Unassigned*	DONE
CAMS1-609	19Q2 CAMS Invoicing	*Unassigned*	DONE
CAMS1-610	19Q1 - CAMS Shopping and Ordering Tab - Backend Integration	*Unassigned*	DONE
CAMS1-611	FY18 Q4 CAMS Limits Tab for Shipto	*Unassigned*	DONE
CAMS1-612	FY18 Q4 CAMS Shipping and Delivery Tab for Shipto	*Unassigned*	DONE
CAMS1-613	Creation of Convenience Card Tab for Shipto	*Unassigned*	DONE
CAMS1-614	Creation of Budget Center Tab	*Unassigned*	DONE

Figura 5. Épicas de los requerimientos capturados en Jira. Nota. Figura de elaboración propia.

CAMS1-615	Creation of Purchase Order Tab	*Unassigned*	DONE
CAMS1-616	Creation of Release Tab	*Unassigned*	DONE
CAMS1-617	19Q4 Blasts Functionality Discovery	*Unassigned*	TO DO
CAMS1-618	Implementation of Diversity Functionality for Billto / Shipto	*Unassigned*	DONE
CAMS1-619	FY18 Q4 CAMS Reporting Fields Tab	*Unassigned*	DONE
CAMS1-620	19Q1 CAMS Notes Tab - Nexus UI and Behavior	*Unassigned*	DONE
CAMS1-621	Q4 CAMS List Pages and Tree Modification	*Unassigned*	DONE
CAMS1-622	FY18 Q4 CAMS Creation Flow for Master/Billto/Shipto	*Unassigned*	DONE
CAMS1-623	ZZZAccount Service Integration	*Unassigned*	DONE
CAMS1-625	FY18 Q4 CAMS Enhancements and Fixes	*Unassigned*	DONE
CAMS1-717	19Q1 - For current CAMS and Nexus fixes or Enhancements	*Unassigned*	DONE

Figura 6. Épicas de los requerimientos capturados en Jira. Nota. Figura de elaboración propia.

3.5 Estimación del proyecto con metodología Cascada

La figura 7 muestra el resultado del ejercicio de estimación del proyecto en metodología de cascada. La técnica de estimación utilizada fue a Juicio de expertos, dando como resultados una proyección de 11 meses para 600 requerimientos recibidos al inicio del proyecto.

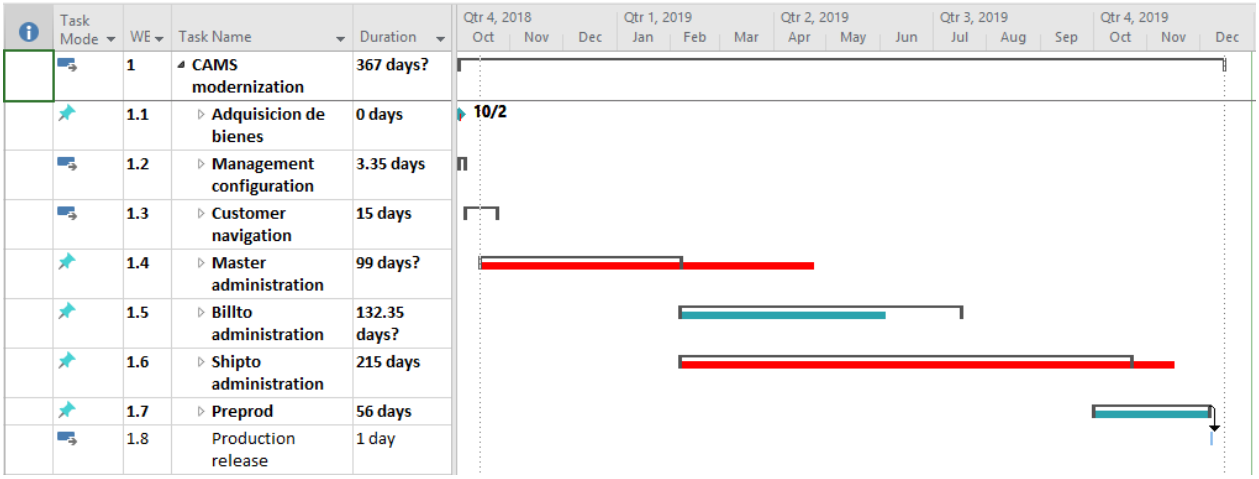

Figura 7. Línea del tiempo de estimación de las épicas (macro funcionalidades) iniciales. Nota. Figura de elaboración propia.

4. Resultados

Los resultados demuestran una constante evolución de los requerimientos a través del tiempo, al inicio del proyecto se contaban con 600 requerimientos, con el paso del tiempo los requerimientos se fueron afinando al punto de crearse nuevas historias y épicas. En el mes 11 ya se contaban documentados alrededor de 1050 requerimientos en total, lo que equivale a un 75% más de los requerimientos iniciales.

Se percibe el esfuerzo en requerimientos esperados de 35 puntos en los primeros 3 sprints, aumentando hasta 45 puntos en los sprints consecutivos.

Los resultados demuestran el decremento de la producción de código funcional en los primeros 3 meses, así como la constante aparición de bugs que se han atendido con 1 sprint de diferencia.

Cabe resaltar que la primera entrega a producción(piloto) ocurrió finalizando el 3er mes, se utilizaron cuentas reales por 3 personas para probar su comportamiento en producción y detectar fallas en tiempo temprano. A partir de esa entrega, hubo entregas recurrentes con calidad productiva 1 ves por mes.

Hasta agosto de 2019 se cubrieron 900 de los 1050 requerimientos a la fecha.

En 7.3 meses se cubrieron 600 requerimientos, en cascada se proyectaron 11 meses para atender esa cantidad, notese un 33% de diferencia entre una metodología y otra. Debido a la naturaleza evolutiva del proyecto no se puede comparar de manera equitativa ambas metodologías, cascada proyectó 11 meses en base a requerimientos inamovibles, mientras en ágil los requerimientos cambiaron, por lo que esos 600 requerimientos no fueron los mismos que los proyectados en cascada.

Fue necesario hacer una proyección de los mismos requerimientos en cascada con la finalidad de compararlos de manera más justa y equitativa. Por tal motivo se realizó otro ejercicio de estimación para la metodología cascada de los 900 requerimientos que se atendieron en el desarrollo del proyecto con ágil, las figuras 8 y 9 muestran esta proyección.

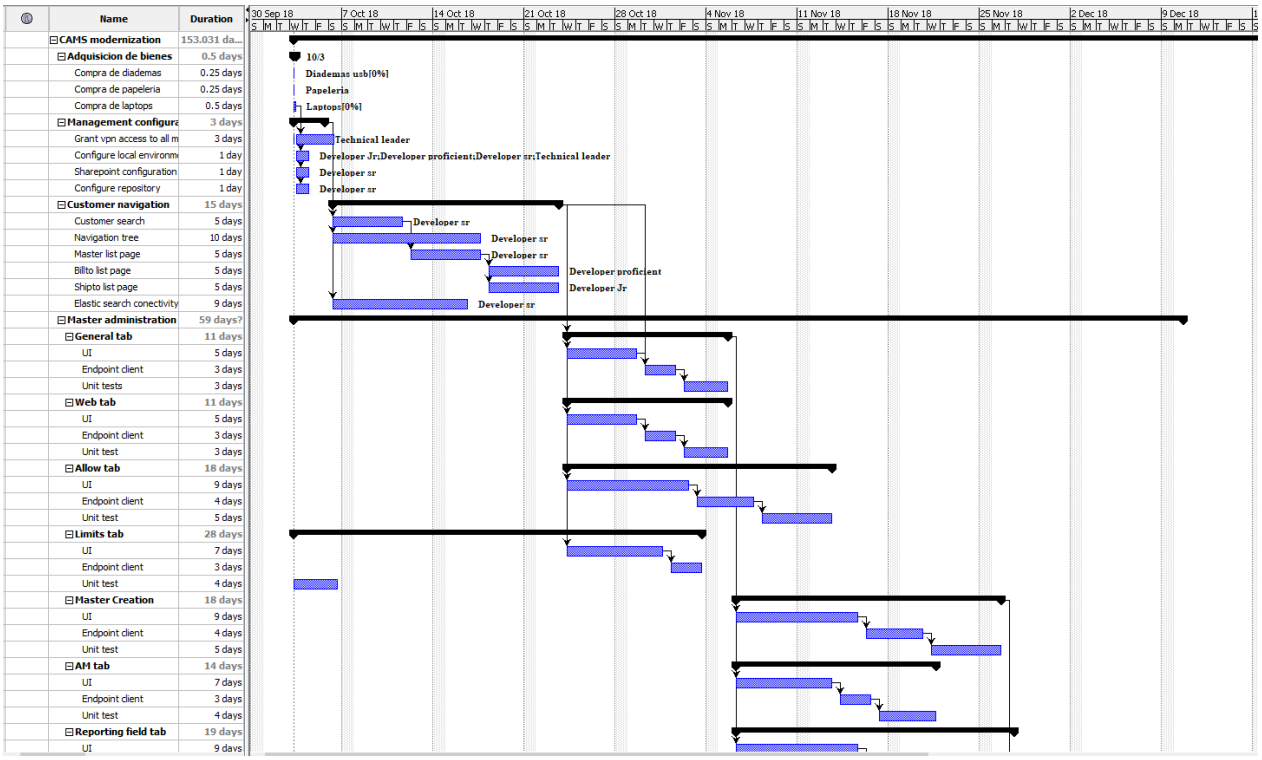

Figura 8.Línea del tiempo de estimación de las épicas (macro funcionalidades) proyección en cascada. Nota. Figura de elaboración propia.

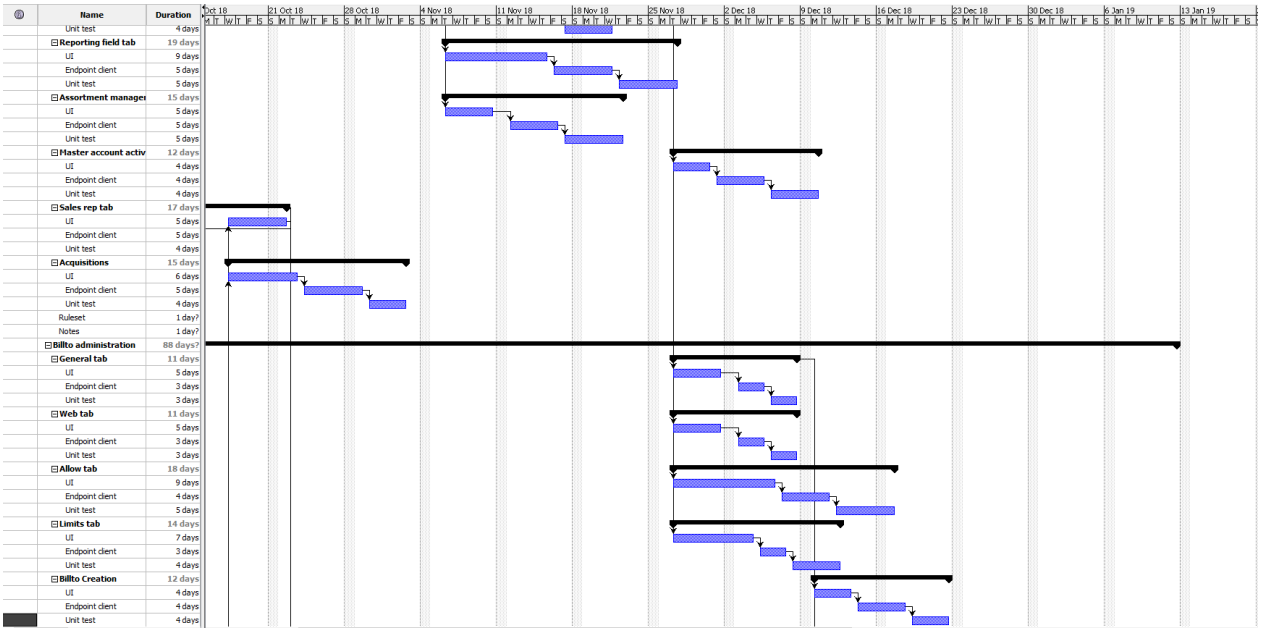

Figura 9. Línea del tiempo de estimación de las épicas (macro funcionalidades) proyección en cascada. Nota. Figura de elaboración propia.

4.1 Cumplimiento de objetivos

El objetivo de este estudio es demostrar que mediante la adopción de la metodología ágil Scrum, se puede mejorar la productividad de los equipos de trabajo, entregando valor en menos tiempo y productos con mayor calidad.

Los objetivos específicos en este estudio fueron los siguientes:

a) Analizar las metodologías de desarrollo de software Scrum y tradicional de cascada.
   En este documento se analizó de manera detallada el proceso y los roles que participan en las dos metodologías Cascada y Scrum.
b) Realizar una comparativa entre la metodología tradicional versus ágil.
   Esta comparativa es importante ya que permite identificar las principales diferencias entre ambas metodologías y conocer en qué momento se debe aplicar una u otra.
c) Adoptar la metodología ágil Scrum a la gestión de un proyecto real.
   Se llevó a cabo una descripción detallada de la adopción de Scrum en el proyecto de modernización de Customer Setup Tool.
d) Describir el impacto en la productividad y competitividad después de adoptar Scrum.
   Se realizó un análisis y mediciones de cómo la adopción de Scrum tiene un impacto sobre la productividad, efectividad, calidad y el valor generado.

4.2 Métricas del proyecto

En desarrollo de software, podemos entender la productividad como la cantidad de código que se produce con respecto a un tiempo determinado. Para medir la productividad se tomaron 3 estadísticas como apoyo: las gráficas de trabajo pendiente, la gráfica de flujo acumulativo de requerimientos y la gráfica de velocidad del equipo.

Gráfica de trabajo pendiente (del inglés burndown chart).

Describe el esfuerzo en requerimientos que está realizando el equipo para construir la parte incremental del producto. En las figuras 10-14 se muestran las mediciones de historias completadas en 5 sprints.

En la figura 10 la disminución de la barra indica la disminución de los requerimientos pendiente, es decir cumplimiento de los requerimientos aumenta. Se puede observar que no llegó a 0, esto quiere decir que el trabajo estimado no se completó en su totalidad para este sprint.

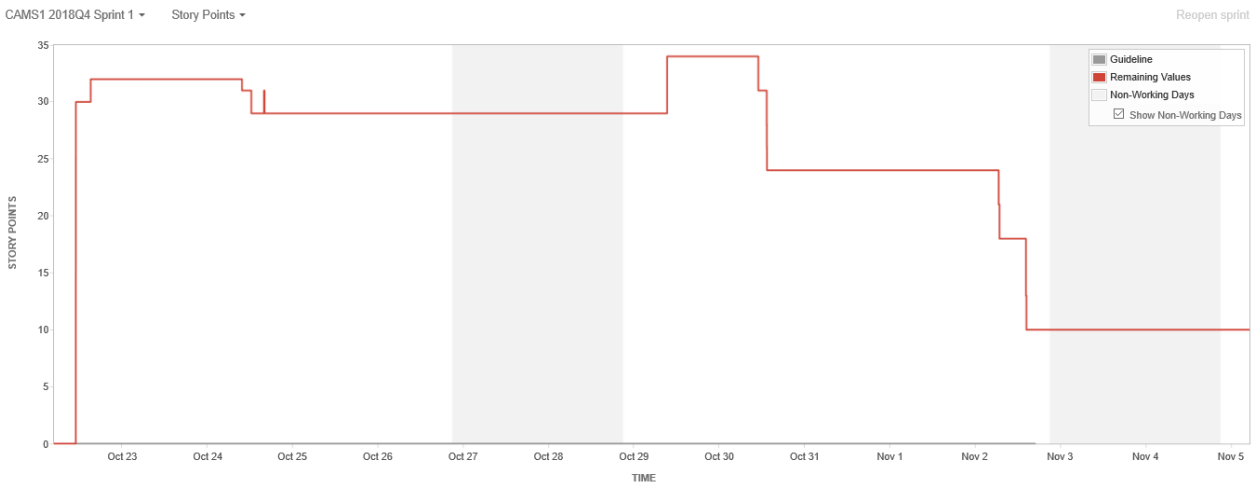

Figura 10. Burndown chart del cuatrimestre 4 de 2018, sprint 1.

En la figura 11 Observamos un mayor decremento en los requerimientos pendientes, a comparación del sprint anterior, esta vez se acercó más a lo estimado.

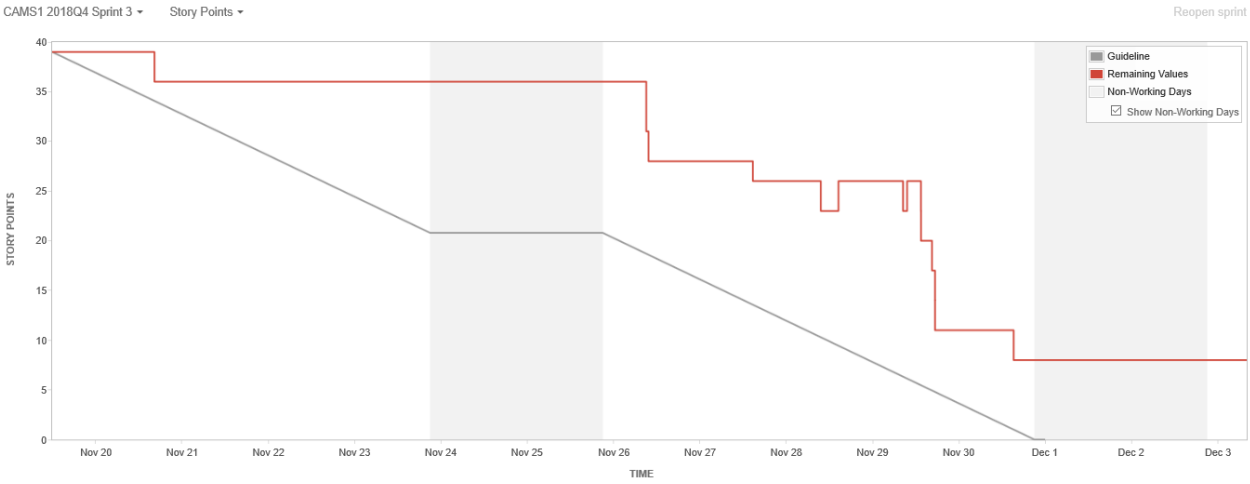

Figura 11. Burndown chart del cuatrimestre 4 de 2018, sprint 3.

La figura 12 muestra un problema, los requerimientos no se fueron entregando paulatinamente, sino al final del sprint, esto puede indicar que los requerimientos fueron complejos esta vez o que hubo algún tipo de bloqueo en el equipo.

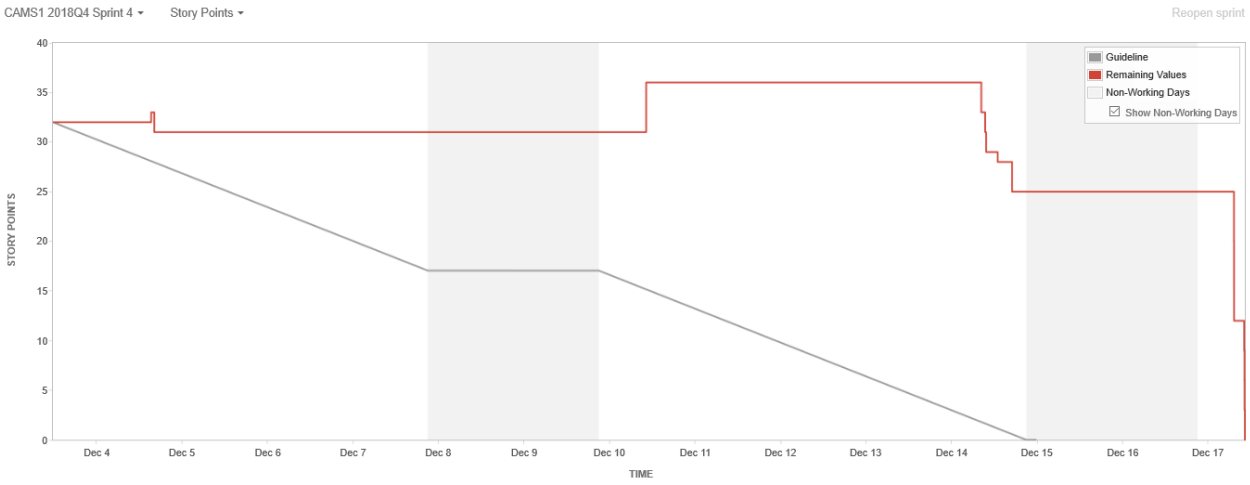

Figura 12. Burndown chart del cuatrimestre 4 de 2018, sprint 4.

La figura 13 muestra un comportamiento más cercano a lo esperado, los requerimientos se fueron entregando paulatinamente y no todo al final del sprint.

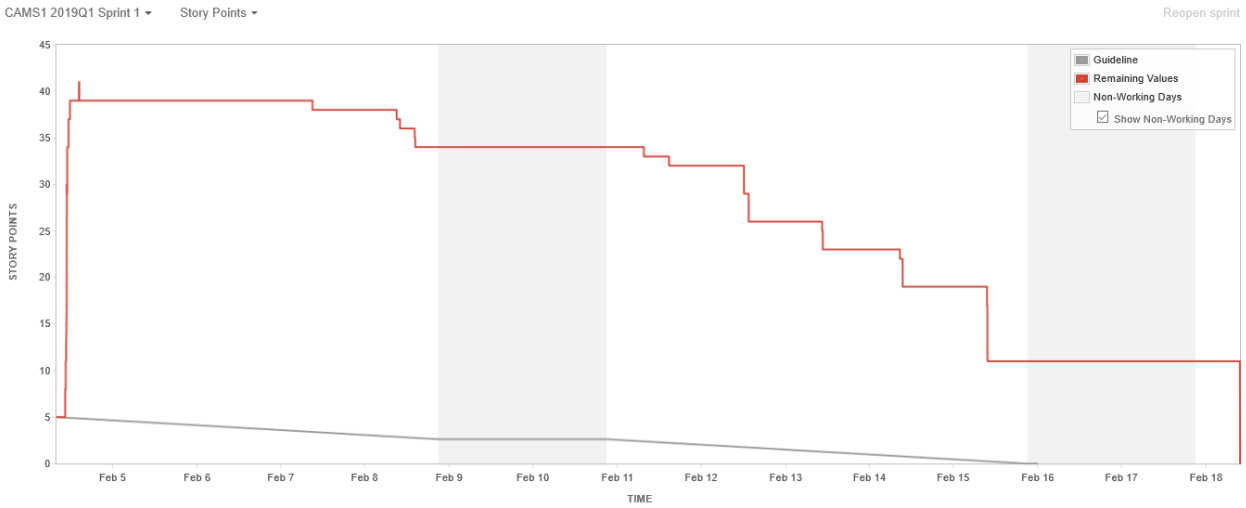

Figura 13. Burndown chart del cuatrimestre 1 de 2019, sprint 1.

La figura 14 muestra uno de los mejores sprints, con un comportamiento más cercano a lo esperado, los requerimientos se fueron entregando desde el inicio del sprint y se mantuvo el ritmo hasta entregar todo en su totalidad.

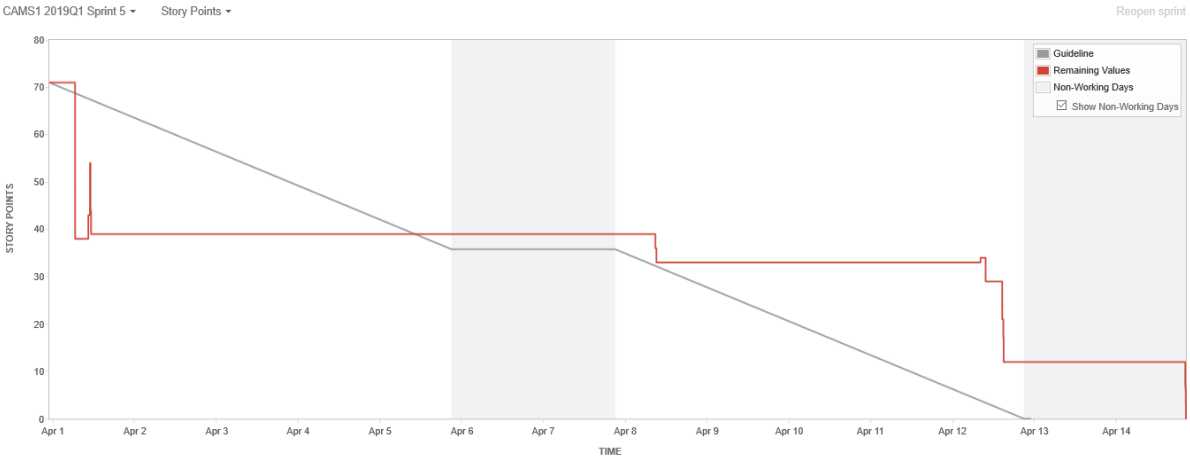

Figura 14 Burndown chart del cuatrimestre 1 de 2019, sprint 5.

Diagrama de flujo acumulativo

Un diagrama de flujo acumulativo es una herramienta utilizada en la teoría de colas. Es un gráfico de área que representa la cantidad de trabajo en un estado determinado, que muestra las llegadas, el tiempo en cola, la cantidad en cola y la salida. En la figura 15 se

muestran 600 requerimientos al inicio del proyecto incrementando paulatinamente hasta llegar a 1050.

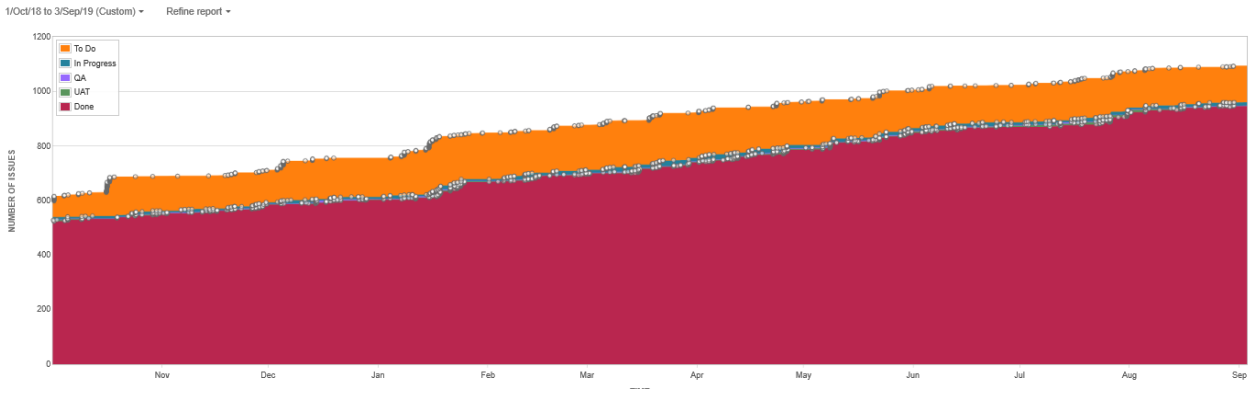

Figura 15. Gráfica de flujo acumulativo desde octubre de 2018 hasta septiembre de 2019.

Velocidad del equipo

Esta estadística de velocidad ayuda a saber ¿cuánto esfuerzo puede completar el equipo en un sprint? ¿cuál es la velocidad máxima o mínima del equipo?

Responder estas preguntas ayuda a estimar en base a los puntos de esfuerzo la velocidad necesaria que requerimos para cumplir una determinada cantidad de historias por sprint. Se esperaba una velocidad inicial de 35 puntos, esta velocidad se mantuvo en los primeros 3 sprints, aumentando hasta 45 puntos en los sprints consecutivos. La figura 16 muestra la velocidad del equipo de 5 sprints.

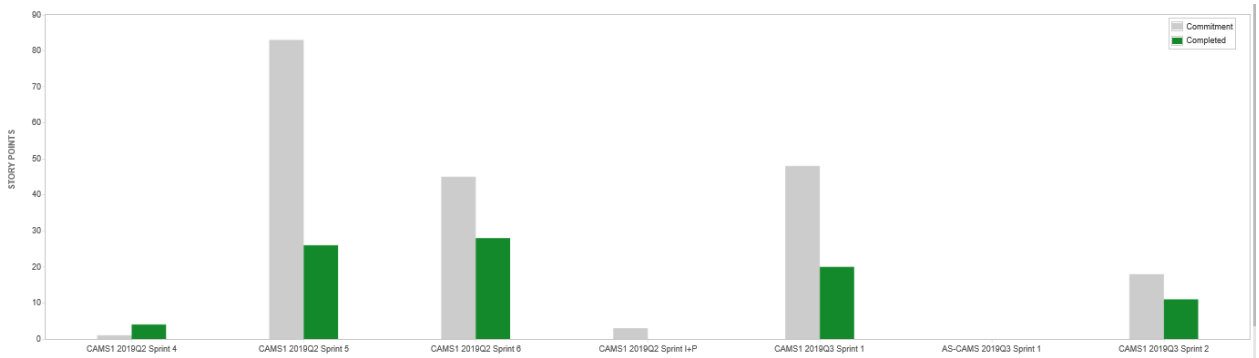

Figura 16. Gráfica de velocidad del equipo.

Eficiencia de calidad

La figura 17 muestra una gráfica con el número de problemas creados frente al número de problemas resueltos en los últimos 320 días.

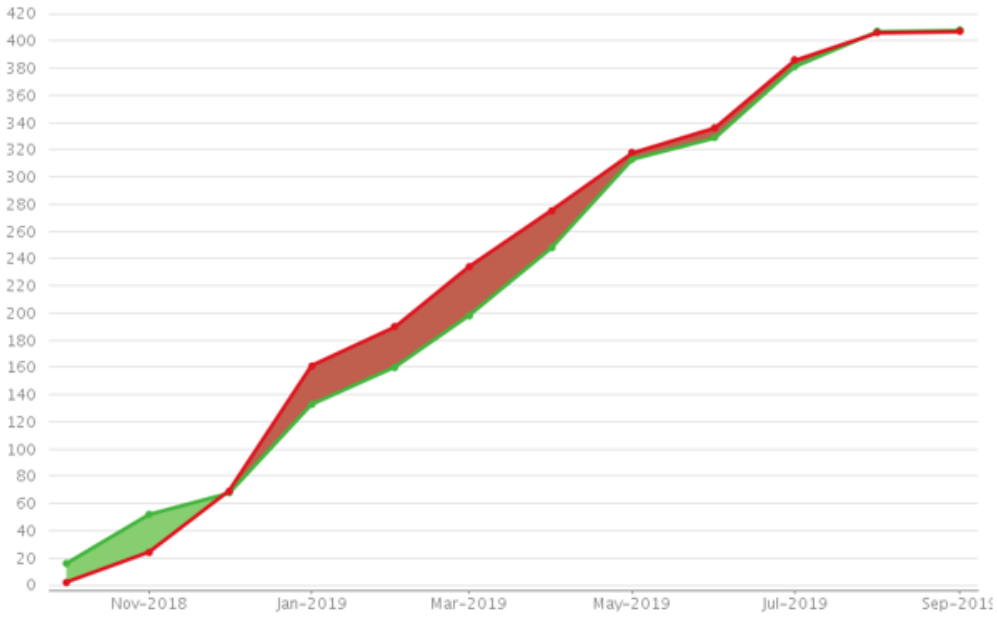

Figura 17. Gráfica lineal comparando bugs creados y bugs resueltos.

La tabla 5 muestra la cantidad de bugs creados y solucionados en cada mes.

Data Table

Period	Created	Resolved
October 2018	2	16
November 2018	22	36
December 2018	45	16
January 2019	92	65
February 2019	29	27
March 2019	44	38
April 2019	41	50
May 2019	43	65
June 2019	18	16
July 2019	50	52
August 2019	20	26
September 2019	1	1

Tabla 5. Bugs creados y solucionados. Nota. Tabla de elaboración propia

5. Conclusiones

Se observó un time to market (plazo de lanzamiento) de 30 días promedio, 45 días máximo. La retroalimentación constante en cada iteración (sprint review) ayuda a mitigar la evolución constante de los requerimientos, lo que contribuye a una continua reducción de bugs hasta estabilizar el sistema. En 11 meses se cubrieron 900 requerimientos en metodología ágil, mismos que se tomaron en cuenta en el ejercicio de estimación para cascada, se proyectaron 16 meses, equivalente a un 45% de diferencia, comprobando de esta manera la hipótesis “Utilizando la metodología de desarrollo ágil Scrum incrementa la productividad en los equipos de trabajo en al menos 20% con respecto a la metodología en cascada”.

El propósito de esta investigación consistió en observar y confirmar el incremento de la productividad y la eficiencia de los equipos de trabajo que participan en proyectos de desarrollo de software, a través de la adopción de la metodología ágil Scrum. Para poder medir el grado de impacto en la organización se obtuvieron métricas cuantitativas y cualitativas de productividad, efectividad, calidad y valor generado. Cascada no es candidato para proyectos con requerimientos evolutivos o con cierta incertidumbre.

Rentabilidad y efectividad.

Esto se traduce a la entrega de valor al negocio en un lapso máximo de 30 días, es decir el tiempo para comercializar (time to market), esto se logró:

- Estableciendo prioridades de valor en los requerimientos.
- Realizando entregas parciales funcionales.
- Empleando técnicas que mejoren la calidad del producto.

Productividad

La productividad se logró mediante el uso de prácticas y herramientas como:

- Empleo de pruebas automatizadas.
- Empleando técnicas de refactorización del código.

- Las reuniones de estatus diarias.
- La motivación y confianza que se le otorga al equipo de desarrollo en la toma de decisiones mediante la pertenencia de responsabilidades (del inglés ownership).

Calidad

La entrega del producto se puede considerar de alta calidad, en el transcurso del desarrollo se identificaron defectos que fueron atendidos en los sprints consecutivos, esto se logró mediante:

- Creación y ejecución de pruebas unitarias.
- Integración continua del código.
- Retroalimentación constante en cada iteración (sprint review).
- Retrospectivas del proceso.

Satisfacción del cliente

El resultado final de un producto funcional y de valor entregado en un lapso corto de tiempo impacta en un cliente cuyas necesidades y expectativas fueron correspondidas de manera positiva.

Hoy en día las organizaciones se enfrentan a retos cada vez más grandes, se les exige, no solamente tener una economía sólida, producir productos con calidad y con el mínimo coste sino también tiene que ser ágil para adaptarse al mercado, aprovechar las oportunidades y generar valor a una gran velocidad que la impulse a tener una ventaja competitiva sobre sus competidores.

En el estudio se demostró que la adopción de la metodología Scrum para la gestión de proyectos de software es una opción viable que puede llevar a una organización a incrementar sus niveles de agilidad y competitividad.

Aunque entender la metodología resulte sencillo, la experiencia en este estudio demuestra que es difícil dominarla, existen varios aspectos que se tienen que madurar y mejorar con la práctica y el cambio de viejos hábitos.

No hay una respuesta simple a por qué la metodología en cascada es tan propensa a fallas, pero está fuertemente relacionada con una suposición falsa clave que subyace a muchos proyectos de software fallidos: que las especificaciones son predecibles y estables y se pueden definir correctamente desde el principio, con bajas tasas de cambio. Esto resulta estar lejos de ser preciso, y un malentendido costoso. Un estudio realizado por Boehm y Papaccio mostró que un proyecto de software típico experimentó un cambio del 25% en los requisitos (Papaccio, 1988). Y esta tendencia se corroboró en otro estudio importante de miles de proyectos de software, con tasas de cambio que son aún más altas: 35% (figura 18) a 50% para proyectos grandes. (Jones, 1997)

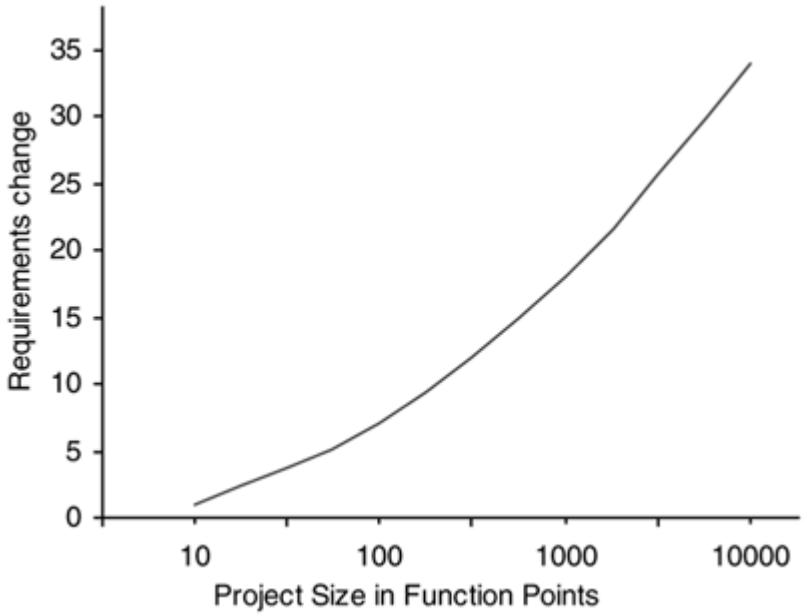

Figura 18. Porcentaje de cambio en los proyectos de software de diferentes tamaños.

Si comparamos con la metodología en cascada, solo la fase de diseño representa un esfuerzo considerable en trabajo que se tiene que modificar ya sea en la fase de construcción y comúnmente en las fases finales como pruebas o despliegue, un punto importante es que, si se mueve el diseño, el código se tiene que refactorizar.

Los resultados que se obtendrían de usar una metodología tradicional no eran aceptables en un escenario donde los requerimientos y las necesidades van cambiando con gran velocidad y donde el esfuerzo que se tenía que aplicar en los procesos y artefactos la mayoría de las veces termina como un desperdicio de tiempo y dinero.

Lista de referencias bibliográficas

Referencias

Arreola Rubalcava, E. y. (2007). *Manual del usuario. Método 52 de formulación y evaluación de proyecto.*

Benefield, G. (2008). *Rolling out Agile in a Large Enterprise.* Yahoo!

Cockroft, A., Hicks , C., & Orzell, G. (29 de 04 de 2011). *Lessons Netflix Learned from the AWS Outage*. Obtenido de https://netflixtechblog.com/lessons-netflix-learned-from-the-aws-outage-deefe5fd0c04

Flarup, E. (2007). *best practices in software localization.* San José, California.

García, A. J. (2017). A microservice-based architecture for enhancing the user experience in cross-device distributed mashup UIs with multiple forms of interaction. *Universal Access in the Information Society*, 1-24.

Gonçalves, L. (2018). Scrum The methodology to become more agile. *Controlling & Management Review*, 40–42.

Harrington, H. J. (1991). *Business Process Improvement.* McGraw-Hill.

Haupt F, L. F.-H. (2017). A framework for the structural analysis of REST APIs. *Proceedings of the IEEE international conference on software architecture.*

Haupt, F. (2017). API governance support through the structural analysis of REST APIs. *Computer Science - Research and Development*, 1-13.

Jones, C. (1997). *Applied Software Measurement.* Nueva York: McGraw-Hill.

Karl T. Ulrich, S. D. (2016). *PRODUCT DESIGN AND DEVELOPMENT.* New York: Mc Graw Hill.

Kettunen, P. (2017). Future software organizations – agile goals and roles. *European Journal of Futures Research*, 5-16.

Larman, C. (2004). *Aplicando UML y Patrones.* Prentice Hall.

Leffingwell, D., & Widrig, D. (2003). *Managing Software Requirements: A Use Case Approach.* Addison-Wesley.

Maleshkova M, P. C. (2010). Investigating web APIs on the World Wide Web. *The 8th IEEE European conference on web services (ECOWS 2010)* (págs. 1-3). Ayia Napa: Cyprus.

Mathur, A. P. (2011). *Foundations of Software Testing.* Pearson.

Newman, S. (2015). *Building Microservices.* Boston: O'Reilly.

Otterstad, C., & Yarygina, T. (2017). *Low-level Exploitation Mitigation by Diverse Microservices.* Oslo: Springer.

Palma F, D. J. (2014). etection of REST patterns and antipatterns: a heuristics-based approach. *ICSOC.* Berlin: Springer.

Palma F. Gonzalez-Huerta J, M. N. (2015). re restful apis well-designed? Detection of their linguistic (anti) patterns. *International conference on service-oriented computing.* Berlin: Springer.

Papaccio, B. a. (1988). *Understanding and Controlling Software Costs.* IEEE Transactions on Software Engineering.

Petrillo F, M. P. (2016). Are REST APIs for Cloud Computing Well-Designed? An Exploratory Study. *International Conference on Service-Oriented Computing* (págs. 157-170). Malaga: Springer, Cham.

Schwaber, K. (2007). *The Enterprise and Scrum.* Microsoft.

Shelly Cashman Series. (1990). *Systems Implementations.*

Society, I. C. (1998). *IEEE Recommended Practice for Software Design Descriptions.* New Yori: Institute of Electrical and Electronics Engineers, Inc.

Staples Inc. (2019). *Modernization Backend retirement plan, Canada Abandonment, LOMS Abandonment, PDB Retirement, DB2 Retirement.* Framingham, Massachusetts.

Villamizar, M. (2017). Cost comparison of running web applications in the cloud using monolithic, microservice, and AWS Lambda architectures. *Service Oriented Computing and Applications*, 233-247.

Vliet, M. A. (2009). *Software Architecture Knowledge Management.* Berlin: Springer.

Waddington, S. (2013). Cloud repositories for research data – addressing the needs of researchers. *Journal of Cloud Computing: Advances, Systems and Applications*, 2-13.

Winston, R. (1970). *Mananging the development of large software systems.* Los Angeles: WesCon.

Yang, X. (2014). Cloud computing in e-Science: research challenges and opportunities. *The Journal of Supercomputing*, 408-464.

Zheng, T. (2018). SmartVM: a SLA-aware microservice deployment framework. *World Wide Web*, 1-19.

Zimmermann, O. (2016). Architectural refactoring for the cloud: a decision-centric view on cloud migration. *Computing*, 129-145.

Zimmermann, O. (2017). Microservices tenets. *Computer Science - Research and Development*, 301–310.