UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

TÍTULO: Migrando un proceso de desarrollo clásico a Scrum.
AUTOR: Ezequiel Simón Nápoli
DIRECTOR ACADÉMICO: Dr. Leandro Antonelli
DIRECTOR PROFESIONAL: Lic. Diego Fernando Tarrío
CARRERA: Licenciatura en Sistemas

Resumen

Las metodologías de desarrollo de Software evolucionan con el tiempo, acompañando la aparición de nuevas tecnologías y requisitos cada vez más demandantes. El tiempo de desarrollo y la calidad son dos elementos que usualmente tienen metas extremadamente ambiciosas. Con lo cual, la metodología elegida debe ayudar a cumplir los objetivos planteados. Esta elección no debe ser tomada a la ligera, se debe analizar el contexto en donde se va a implementar, planificar los pasos para llevar a cabo el cambio y adaptar lo que se detecte como inconveniente. Por otra parte, los beneficios que la metodología puede aportar se deben contraponer con los desafíos que conlleva la implementación de la misma. Estos desafíos no son solo técnicos, sino que involucran a toda la organización. En esta tesina se analizarán dichos puntos en el contexto de una migración de metodología clásica a una ágil, realizada en la empresa en la que trabajo.

Palabras Clave

Metodologías de desarrollo de software, metodologías clásicas, metodologías tradicionales, metodologías estructuradas, metodologías ágiles, modelo en cascada, Scrum, migración, desafíos.

Trabajos Realizados

En primer lugar, se describieron las metodologías tradicionales y ágiles, haciendo una comparación entre ambas. Luego se hizo una revisión de la literatura sobre los posibles desafíos al realizar una migración hacia una metodología ágil. Finalmente, se analizaron estos desafíos y la implementación de Scrum en una migración de estas características realizada en la empresa a la que pertenezco.

Conclusiones

Al migrar a una metodología ágil hay que tener en cuenta todos los desafíos que pueden presentarse en cualquier nivel de la organización. En el caso de estudio los más importantes son la resistencia al cambio y el poder romper con la cultura de trabajo anterior. La implementación de Scrum se fue adaptando con el tiempo para parecerse cada vez más a la metodología previa. Como resultado, se perdieron muchos elementos de Scrum que hacen a su esencia.

Trabajos Futuros

Para acercar la implementación de Scrum a la propuesta por la literatura se deberían analizar distintas acciones: en primer lugar, realizar una nueva capacitación a todo el personal, remarcando los errores cometidos. En segundo lugar, es necesario realizar un trabajo de adaptación a la metodología con los clientes involucrados. Por último, y teniendo en cuenta las dificultades del contexto actual, se podría realizar una implementación correcta de Scrum en algún proyecto de menor tamaño y complejidad. Dicha implementación debería ser guiada por expertos y funcionaría como prueba piloto.

Fecha de la Presentación: Marzo de 2019

Universidad Nacional de La Plata

Facultad de Informática

Migrando un proceso de desarrollo clásico a Scrum

Director Académico:
Dr. Leandro Antonelli

Director Profesional:
Lic. Diego Fernando Tarrío

Alumno:
Ezequiel Simón Nápoli - 7555/7

Resumen

Las metodologías de desarrollo de Software evolucionan con el tiempo, acompañando la aparición de nuevas tecnologías y requisitos cada vez más demandantes. El tiempo de desarrollo y la calidad del software son dos elementos que usualmente tienen metas extremadamente ambiciosas. Con lo cual, la metodología elegida debe ayudar a cumplir los objetivos planteados. Esta elección no debe ser tomada a la ligera, se debe analizar el contexto en donde se va a implementar, planificar los pasos para llevar a cabo el cambio de metodología y adaptar lo que se detecte como inconveniente. Por otra parte, los beneficios que la metodología puede aportar se deben contraponer con los desafíos que conlleva la implementación de la misma. Estos desafíos no son solo técnicos, sino que involucran a toda la organización. En esta tesina se analizarán dichos puntos en el contexto de una migración de metodología clásica a una ágil, realizada en la empresa en la que trabajo.

Palabras Claves: Metodologías de desarrollo de software, metodologías clásicas, metodologías tradicionales, metodologías estructuradas, metodologías ágiles, modelo en cascada, Scrum, migración, desafíos.

Índice General

Índice de Tablas

Índice de Figuras

Capítulo 1

Introducción

1.1. Motivación

El desarrollo de software es una actividad compleja que involucra a una gran cantidad de actores, los cuales, a su vez, tienen variadas características y conocimientos. Es menester tener un proceso que ordene todas las actividades relacionadas a la construcción del software para disminuir los riesgos de que el proyecto fracase o se demore más de lo esperado.

Un proceso de desarrollo de software es el conjunto de actividades requeridas para realizar un sistema de software. Estas actividades son: especificación de requerimientos, diseño, codificación, validación (pruebas) y mantenimiento. Estos procesos se pueden dividir en estructurados (también llamados tradicionales o pesados), que son los que promueven la disciplina por medio de la planificación y la comunicación escrita, y las metodologías ágiles, que dan prioridad a la interacción entre los individuos y a la comunicación con el cliente. [3]

Una de las metodologías ágiles más difundidas es Scrum. Scrum define un proceso empírico, iterativo e incremental de desarrollo que intenta obtener ventajas respecto de los procesos estructurados, optimizando la predictibilidad y el control de riesgos. Scrum no es un proceso o método definido, sino que es un *framework* dentro del cual se pueden emplear varios procesos y técnicas [6]. Al principio del proyecto se define el *Product Backlog*, que contiene todos los requerimientos funcionales y no funcionales que deberá satisfacer el sistema a construir. Los mismos estarán especificados de acuerdo a las convenciones de la organización ya sea mediante *features*, casos de uso, diagramas de flujo de datos, incidentes, tareas, etc. El *Product Backlog* será definido durante reuniones de planeamiento con los *stakeholders*. A partir de ahí se definirán las iteraciones, conocidas como *Sprints*, en las que irá evolucionando la aplicación. Los *Sprints* pueden abarcar de una a cuatro semanas y cada uno tendrá su propio *Sprint Backlog,* que será un subconjunto del *Product Backlog* con los requerimientos a ser construidos en el *Sprint* correspondiente. Según Sutherland y Schwaber [6], Scrum es liviano, fácil de entender, pero difícil de dominar.

Las organizaciones deben buscar el proceso de desarrollo que más se ajuste a sus características, necesidades y proyectos. Si bien los procesos de desarrollo fueron evolucionando con el tiempo, no es estrictamente necesario cambiar permanentemente por las novedades del mercado. Los beneficios de las metodologías ágiles pueden ser muy atractivas, pero deben circunscribirse

e integrarse con las actividades existentes en la organización [10]. Todo cambio de la metodología de trabajo tiene un costo y genera nuevos riesgos, por lo que la decisión debe ser bien pensada y adecuada a los objetivos que se persiguen.

1.2. ¿Qué tipos de desafíos podemos encontrar al migrar a una metodología ágil?

Para poder realizar una migración de la metodología, primero se debe analizar el contexto en donde se va a implementar, planificar los pasos para llevar a cabo el cambio y luego adaptar lo que se detecte como inconveniente. Por otra parte, los beneficios que la metodología puede aportar se deben contraponer con los desafíos que conlleva la implementación de la misma. Estos desafíos no son solo técnicos, sino que involucran a toda la organización. Almeida [11] define un agrupamiento de todos los desafíos mencionados en diferentes informes. Estos desafíos se pueden clasificar en:

- Organización y administración: toda la estructura de la organización debe comprender los objetivos y el funcionamiento de la nueva metodología. Este tipo de cambios suele producir mucha resistencia.
- Personas: Debe existir una capacitación y distribución de conocimiento para que todos los actores puedan cumplir correctamente su rol dentro de la nueva metodología. Se debe trabajar fuertemente en el trabajo en equipo y en el compromiso con el nuevo rol.
- Procesos: el cambio de metodología conlleva un cambio en los procesos de la organización. Fundamentalmente se debe asegurar la mayor calidad posible en todos los procesos.
- Herramientas: se debe tener en cuenta la complejidad de los sistemas que se construyen y utilizar las herramientas adecuadas que le den soporte. Es fundamental poder evaluar el proyecto y realizar un buen seguimiento de los errores reportados.

1.3. El caso de estudio

Para analizar estos desafíos, se estudiará la implementación de Scrum en la empresa en la que trabajo. Allí se utilizaba una metodología de trabajo similar al modelo en cascada para el desarrollo de un sistema para *brokers* de seguros. Este producto es una aplicación de escritorio de un gran tamaño y complejidad. A pesar de que continúa en producción en muchos clientes, se decidió hace algunos años realizar una migración hacia una tecnología actual y hacia un sistema web orientado a servicios. Al comienzo, el desarrollo no tenía prácticamente ningún marco de trabajo, dado que no existían proyectos

concretos. Conocer el detalle del trabajo realizado y el trabajo pendiente era prácticamente una tarea imposible. Al conseguir el primer cliente para el nuevo producto, se decidió realizar también una migración de la metodología, utilizada prácticamente desde los comienzos de la empresa. Se eligió Scrum y se implementó rápidamente y con poca capacitación. Además de los desafíos propios del cambio del paradigma tradicional, la implementación de Scrum tuvo muchas adaptaciones que continúan actualmente. Se analizarán entonces los desafíos propios de la migración de una metodología tradicional a una ágil, aplicados al caso de estudio, junto con los errores y aciertos en la implementación de Scrum.

1.4. Objetivos de la Tesina

Los objetivos planteados en esta tesina son los siguientes:

- Objetivo General:
  - Describir las lecciones aprendidas en la migración de un proceso de desarrollo clásico a uno ágil, como así también en la implementación definitiva de Scrum.
- Objetivos específicos:
  - Analizar distintos procesos de desarrollo clásicos.
  - Analizar distintas metodologías ágiles y en particular Scrum.
  - Describir el proceso de desarrollo utilizado anteriormente en la empresa.
  - Describir las dificultades encontradas en el proceso de migración a Scrum.
  - Describir las diferencias entre la implementación definitiva de Scrum y la recomendada por la literatura.

1.5. Estructura del Informe

El informe de la Tesina se compone de los siguientes capítulos:

- **Capítulo 1**. Introducción.
- **Capítulo 2**. Marco conceptual: Se comienza con una breve descripción de distintas metodologías tradicionales y ágiles. En particular, se hace hincapié en la descripción de Scrum.
- **Capítulo 3**. Revisión de Literatura: En primer lugar, se presenta una comparación de ambos tipos de metodologías, la cual se puede encontrar en diversos informes. Luego se describen una serie de dificultades que suelen encontrarse en la migración de una metodología tradicional a una ágil.
- **Capítulo 4**. Contexto: Se describen las características principales de la empresa en la cual se basa el caso de estudio.

- **Capítulo 5**. Migrando a Scrum: Se explica cómo fue todo el proceso de migración de un modelo en cascada a Scrum, junto con las dificultades encontradas y las diferencias con lo aconsejado normalmente en la teoría de Scrum.
- **Capítulo 6**. Conclusiones: se presentan las conclusiones del caso de estudio en base a dos cuestiones fundamentales: el proceso de migración y la implementación de Scrum.

Capítulo 2

Marco conceptual

2.1. Ciclo de Vida

En primer lugar, es necesario definir qué es el Ciclo de Vida del Software. El Ciclo de Vida describe el desarrollo de un sistema de software desde la fase inicial hasta la fase final y el orden y coordinación de las tareas involucradas en el desarrollo. Se utiliza para garantizar que el software resultante cumpla con los requisitos de la aplicación y para verificar que los métodos utilizados son los apropiados.

Según la Norma IEEE 1040, el Ciclo de Vida del Software es “una aproximación lógica a la adquisición, suministro, desarrollo, explotación y mantenimiento del Software”. Por otro lado, la Norma ISO 12207-1 define el Ciclo de Vida del Software como “un marco de referencia que contiene los procesos, las actividades y las tareas involucradas, en el desarrollo, la explotación y el mantenimiento de un producto de Software, abarcando la vida del sistema desde la definición de los requisitos hasta la finalización de su uso”.

2.2. Modelo de Ciclo de Vida

El Modelo del Ciclo de Vida determina la manera en que se van realizando las actividades dentro del Ciclo de Vida del Software. La elección de un modelo depende de las características del producto a desarrollar, de los costos y de los requerimientos del cliente.

Se pueden agrupar los distintos modelos de Ciclo de Vida, por sus características más sobresalientes, de la siguiente forma:

- Evolutivo: Se dice que un Ciclo de Vida es Evolutivo cuando acompaña la evolución del producto con el tiempo. Es decir, a medida que transcurre el tiempo contempla los cambios en requerimientos.
- Incremental: Se dice que un Ciclo de Vida es Incremental cuando está dividido en etapas y al final de cada una se genera una versión o incremento usable del producto en cuestión.
- Iterativo: Se dice que un Ciclo de Vida es Iterativo cuando existen etapas que se repiten en cada ciclo de trabajo. Por ejemplo: Análisis / Diseño / Desarrollo / Pruebas, se repiten en la construcción de cada incremento de un producto.

2.3. Metodología de desarrollo de Software

Según el INTECO [13], una metodología es un conjunto integrado de técnicas y métodos que permite abordar de forma homogénea y abierta cada una de las actividades del ciclo de vida de un proyecto de desarrollo. Es un proceso de software detallado y completo.

Las metodologías se basan en una combinación de los modelos de ciclo de vida. Dan un marco de trabajo desde el momento en que se idea un proyecto hasta que se finaliza. El objetivo de la metodología es poder llevar a cabo un proyecto con la mayor probabilidad de éxito posible.

En síntesis, una metodología de desarrollo de software:

- Optimiza el proceso y el producto de software.
- Provee métodos que guían en la planificación y en el desarrollo del software.
- Define qué hacer, cómo y cuándo durante todo el desarrollo y mantenimiento de un proyecto.

Existen diversas metodologías de desarrollo de software que fueron evolucionando con los años, cada una con sus ventajas y desventajas. A su vez, no toda metodología es aplicable a cualquier contexto, ya que depende de las características inherentes a ésta.

2.4. Metodologías tradicionales

Las metodologías estructuradas son consideradas como la forma tradicional de desarrollar software. Proponen un proceso disciplinado con el objetivo de hacer que el desarrollo de software sea lo más predecible y eficiente posible.

Awad [9] destaca 4 características principales de las metodologías tradicionales:

- Enfoque predictivo: tienen una tendencia a planificar de antemano una gran parte del proceso de software con mucho detalle y por un largo período de tiempo.
- Documentación exhaustiva: las metodologías tradicionales tienen como pieza fundamental a la documentación de los requerimientos. Entienden que es posible definir y documentar todos los requerimientos de un sistema antes de comenzar cualquier tipo de desarrollo.
- Orientación a procesos: El objetivo de las metodologías tradicionales es definir un proceso que funcione bien para quienquiera que lo utilice. El proceso consiste de ciertas tareas que llevarán a cabo gerentes, diseñadores, desarrolladores, testers, etc. Para cada una de estas tareas existe un proceso definido.
- Orientación a herramientas: Deben utilizarse herramientas para cumplimentar cada una de estas tareas (herramientas para gestión de proyectos, editores de código, compiladores, etc.).

2.4.1. Modelo en Cascada

El modelo en Cascada propone una división en etapas que deben seguir un orden secuencial y estricto. El inicio de cada etapa debe esperar a la finalización de la etapa anterior. La primera aproximación de este modelo fue presentada por Winston Royce en 1970, aunque allí no se lo nombraba con el término “cascada”. En el modelo de Royce se definían las siguientes fases:

1. Especificación de requisitos
2. Diseño
3. Construcción (Implementación o codificación)
4. Integración
5. Pruebas
6. Instalación
7. Mantenimiento

Actualmente al modelo se lo conoce con las siguientes fases:

- Ingeniería del Sistema: en esta etapa se investiga y comprende el sistema a desarrollar. Aquí se definen los requisitos que el software debe cumplir.
- Análisis: Se especifica la función del software, la descripción de las interfaces con los otros elementos del sistema y las restricciones de diseño. Como resultado se obtienen las especificaciones de los requerimientos.
- Diseño: Diseñar cómo el sistema cumplirá con los requerimientos. Aquí se toman decisiones sobre la arquitectura y tecnología a utilizar.
- Codificación: En base a lo diseñado, se construye el sistema en un lenguaje de programación concreto.
- Prueba: Una vez que el software ha sido desarrollado, se realizan pruebas para descubrir los errores que puedan existir en la función, la lógica o en la implementación.
- Mantenimiento: Aquí se modifica el sistema para adaptarlo a nuevas necesidades o para corregir errores detectados.

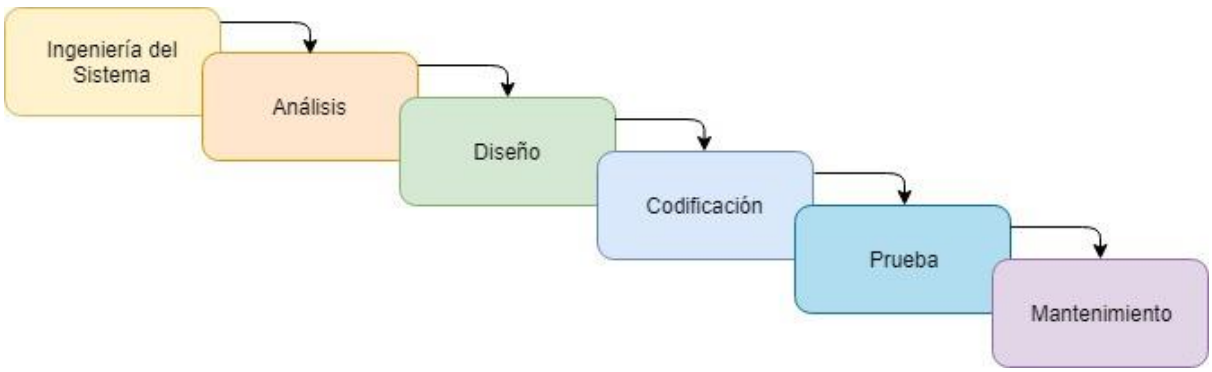

***Figura 2.1**. Modelo en Cascada*

El modelo en cascada presupone que es posible definir todos los requerimientos al inicio del proyecto, lo cual no siempre concuerda con la realidad. Además, al tener una etapa de prueba recién al final, hace que encontrar un error se torne un grave problema. Es un modelo que quizás pueda ser acorde a proyectos chicos o de objetivos muy específicos, en los que es posible conocer todos los requerimientos de antemano.

2.4.2. Modelo en V

El modelo en V surge para solucionar algunas de las problemáticas del modelo en cascada. El cambio fundamental es el de integrar las pruebas con todas las fases del ciclo de vida y no solo con el de desarrollo, además de que pueden realizarse en paralelo. De esta forma, las pruebas se incorporan en etapas tempranas y abarcan todas las actividades del proceso. El nombre proviene de la representación gráfica del modelo y también de los términos “validación” y “verificación”. Si bien las etapas que se definen individualmente pueden ser muy similares a las del modelo en cascada, hay una diferencia fundamental: en lugar de ser lineales, luego de la etapa de codificación se retoma el ciclo hacia arriba.

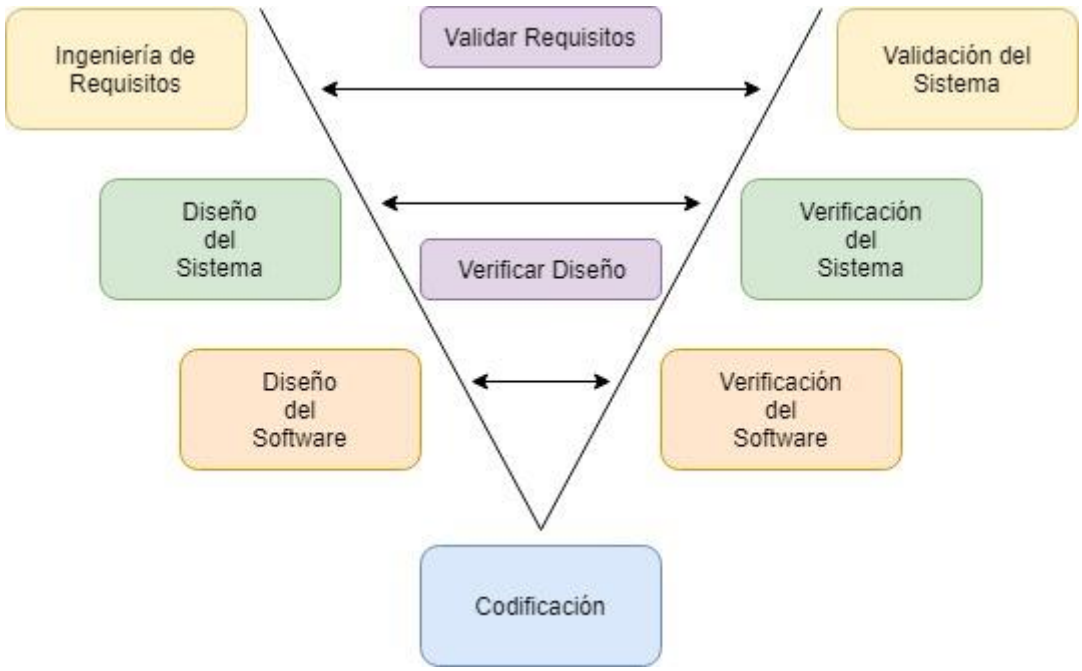

***Figura 2.2**. Modelo en V*

2.4.3. Modelo de Prototipación

Este modelo se basa en la técnica de construir y experimentar con versiones pre-armadas del sistema. Esto permite comprender mejor la funcionalidad, los requerimientos pretendidos y la interfaz que va a tener el sistema. El modelo tiene las siguientes fases:

1. Recolección y refinamiento de requerimientos: Se definen los objetivos globales y todos los requerimientos conocidos. Se identifican las áreas donde será necesaria una mayor definición.
2. Diseño rápido: Diseño del prototipo a construir.
3. Construcción del prototipo: Desarrollo del prototipo planteado.
4. Evaluación del prototipo por el cliente: El cliente evalúa el prototipo construido y sugiere las modificaciones necesarias para cumplir con sus necesidades.

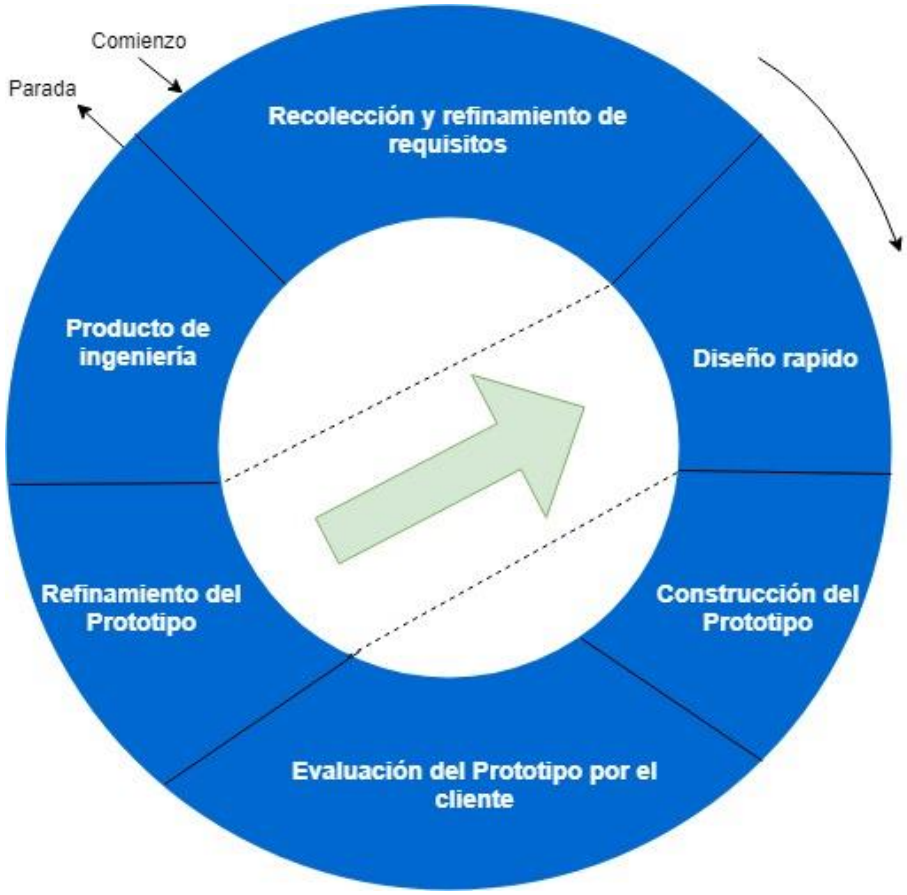

***Figura 2.3**. Modelo de Prototipación*

Pressman [1] sugiere que cuando el cliente tiene una necesidad legítima, pero está desorientado en los detalles, el primer paso es desarrollar un prototipo.

2.4.4. Modelo RAD (*Rapid Application Development*)

Pressman [1] explica que la metodología de desarrollo rápido de aplicaciones (RAD) se constituyó para responder a la necesidad de entregar sistemas más rápidamente. El enfoque RAD no es apropiado para cualquier proyecto; debe tenerse en cuenta el alcance, el tamaño y el contexto. Se trata

de un modelo en cascada con un ciclo de desarrollo extremadamente corto y basado en componentes. Si se tiene un conjunto de requerimientos bien definido y acotado, se puede llevar a cabo el proyecto en tiempos de 60 a 90 días aproximadamente. Sus fases son:

- Modelado de Gestión: Este modelo intenta responder algunas cuestiones como: ¿Qué información conduce el proceso de gestión? ¿Qué información se genera? ¿Quién la genera? ¿A dónde va la información? ¿Quién la procesa?
- Modelado de datos: Se definen los atributos de cada uno de los objetos y sus relaciones.
- Modelado del proceso: Se definen procesos para agregar, modificar, eliminar, o recuperar un objeto de datos.
- Generación de aplicaciones: El RAD propone utilizar herramientas que faciliten el desarrollo del software, reutilizando componentes ya existentes o generando componentes reutilizables.
- Pruebas y entrega: Finalmente se prueban todas las componentes nuevas. Dado que se hace hincapié en la reutilización, deberían existir muchas componentes ya probadas, con lo que, en ese caso, bastaría con pruebas de integración.

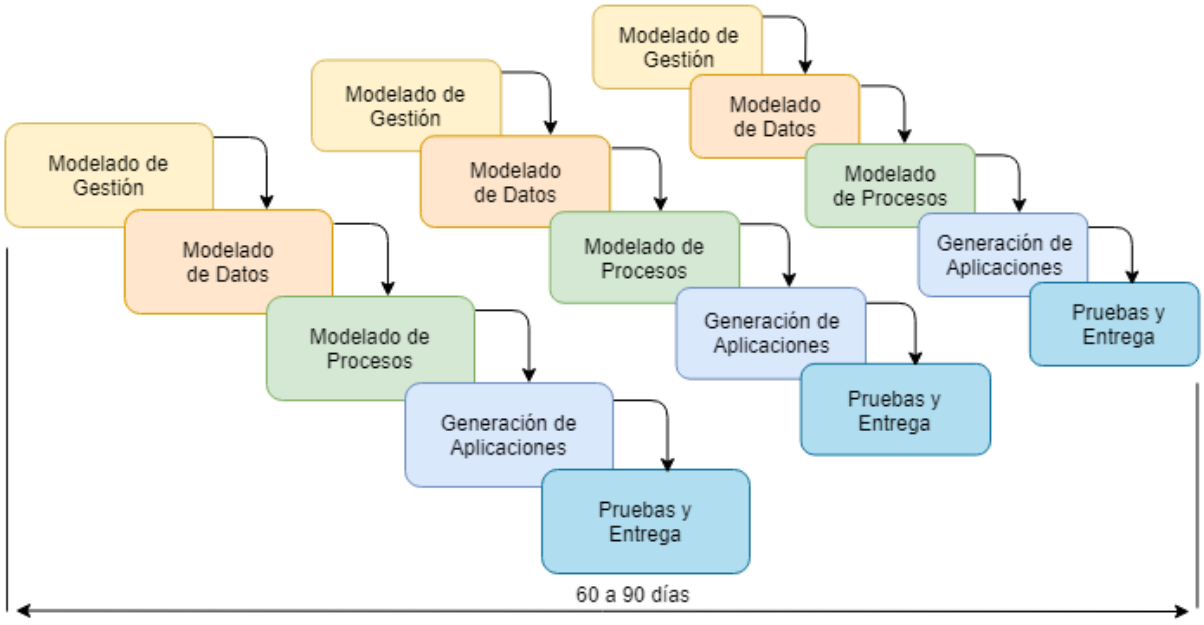

***Figura 2.4**. Modelo RAD*

2.4.5. Modelos Evolutivos del Proceso de Software

El software, al igual que todos los sistemas complejos, evoluciona con el tiempo. Esto se ve reflejado en la variación que sufren los requerimientos, incluso luego de haberse iniciado el desarrollo. Asimismo, las restricciones del mercado imponen limitaciones tanto de recursos como de tiempo, lo cual hace que sea necesario realizar entregas parciales.

Es necesario que el modelo de proceso esté diseñado para acompañar a dicha evolución. Los modelos de proceso evolutivos son iterativos y se caracterizan por la capacidad de permitir desarrollos cada vez más completos.

2.4.6. Modelo Incremental

Pressman [1] explica que el modelo incremental combina elementos del modelo en cascada (aplicados repetidamente) con ideas de la construcción de prototipos. A medida que avanza el tiempo calendario, se aplican secuencias de forma escalonada. Cada secuencia lineal aporta un incremento al producto de software y en cada uno es posible incorporar un prototipo. El objetivo es que en cada secuencia se obtenga un producto funcional que permita evaluar el avance.

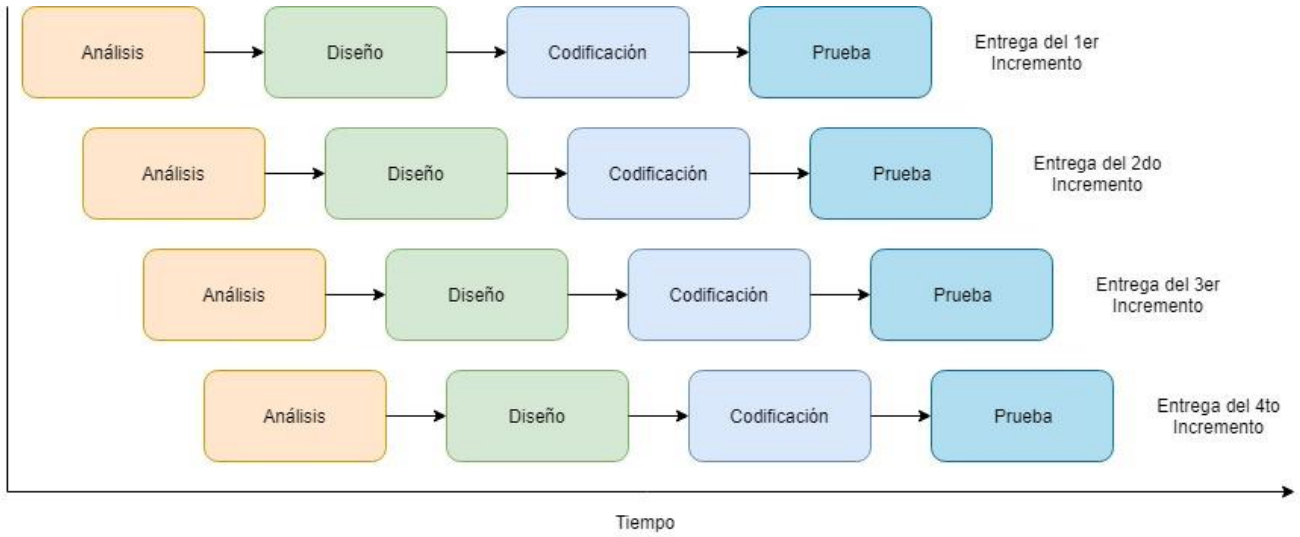

***Figura 2.5**. Modelo Incremental*

2.4.7. Modelo en Espiral

El modelo en espiral, desarrollado por Barry Boehm en 1985, es un modelo iterativo e incremental en el que se hace fuerte hincapié en el análisis de riesgos. Las actividades conforman una espiral en el que en cada bucle se realiza un incremento en el sistema a desarrollar. Las actividades no se establecen desde el comienzo, sino que se definen a partir de un análisis de riesgos realizado en el bucle anterior. Las fases de este modelo son:

- Comunicación con el cliente: comprende las tareas necesarias para la comunicación entre el desarrollador y el cliente.
- Planificación: comprende las tareas necesarias para definir recursos, tiempos, fechas, etc.
- Análisis de riesgos: comprende las tareas necesarias para el análisis de riesgos técnicos y de gestión.

- Ingeniería: comprende las tareas necesarias para construir una o más representaciones del sistema.
- Construcción y Entrega: comprende las tareas necesarias para construir, probar, instalar y proporcionar soporte al usuario.
- Evaluación del cliente: comprende las tareas necesarias para obtener una evaluación del cliente, basada en las representaciones e implementaciones realizadas.

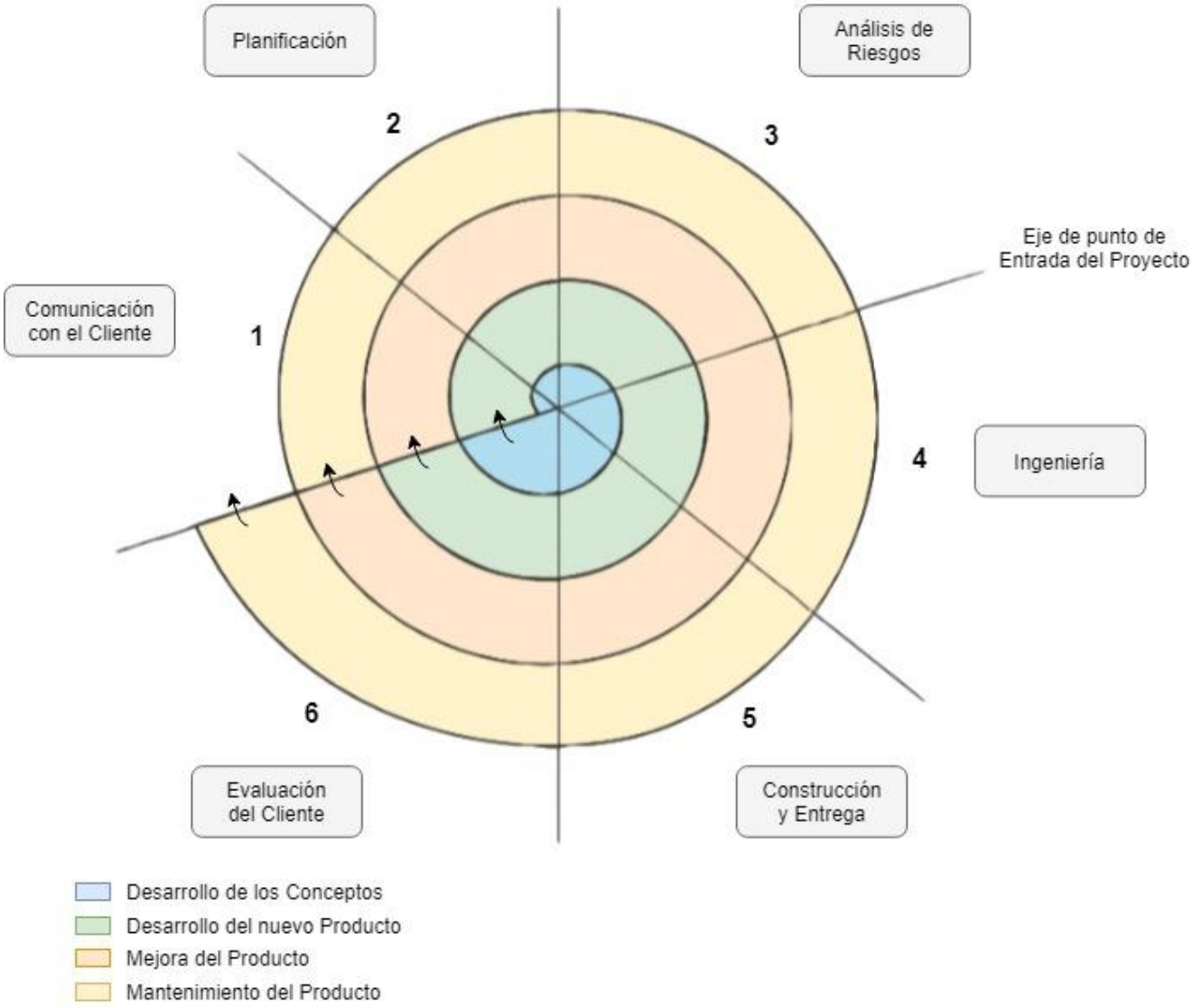

***Figura 2.6**. Modelo en Espiral*

2.5. Metodologías ágiles

Las metodologías clásicas o estructuradas se basaban en el control del proceso de desarrollo a través de una rigurosa definición de roles, actividades y artefactos, incluyendo modelado y documentación detallada. Este esquema podía ser adecuado para proyectos grandes, pero no era lo mejor para

adaptarse a la nueva naturaleza del software, mucho más cambiante y dinámica. A grandes rasgos, las metodologías ágiles promueven:

- Retrospección
- Adaptación a los cambios
- Trabajo en equipo
- Autogestión
- Entregas rápidas
- Comunicación personal por sobre la documentación
- Alineación entre el área de desarrollo y el cliente

En las metodologías ágiles se busca realizar incrementos pequeños con planificaciones acotadas, en lugar de realizar una planificación de todo el proyecto. Ante cada fin de iteración se debe poder obtener una versión cerrada, que por más que no vaya a ser productiva, debe estar disponible para evaluar los avances realizados. La metodología propone organizar iteraciones cortas (en general de 1 a 4 semanas) de las que se obtiene este incremento. Un equipo está compuesto normalmente por entre 5 y 9 personas y son multidisciplinares. En una iteración, cada equipo se ocupa de realizar un ciclo de desarrollo completo, incluyendo planificación, análisis de requerimientos, diseño, codificación y pruebas. Los miembros de los equipos deciden ellos mismos cómo y quienes van a encarar cada tarea. La documentación pasa a cumplir un papel secundario: solo debe generarse cuando un actor lo requiera. Asimismo, se pondera la comunicación cara a cara rutinaria, diaria y formal entre los miembros del equipo y en algunos casos involucrando a un representante del cliente. Al final de cada iteración, las personas involucradas en el negocio y el representante del cliente revisan el progreso y reevalúan las prioridades, asegurando la alineación con las necesidades del cliente y los objetivos de la compañía.

2.5.1. Resumen de las Metodologías Ágiles más importantes

En el cuadro siguiente puede verse un resumen de las metodologías ágiles más difundidas:

***Tabla 2.1**. Resumen de metodologías ágiles*

Metodología	Acrónimo	Creación
Adaptive Software Development	ASD	Highsmith 2000
Agile Modeling	AM	Ambler 2002
Cristal Methods	CM	Cockbum 1998

Aguile RUP	dX	Booch, Martin, Newkirk 1988
Dynamic Solutions Delivery Model	DSDM	Stapleton 1997
Evolutionary Project Managment	EVO	Gilb 1976
eXtreme Programming	XP	Beck 1999
Feature-Driven Development	FDD	De Luca & Coad 1998 Palmer & Felsing 2002
Lean Development	LD	Charette 2001, Mary & Tom Poppendieck 2003
Rapid Development	RAD	McConnell 1996
Microsoft Solutions Framework	MSF	Microsoft 1994
Scrum	Scrum	Sutherland 1994 Schwaber 1995

2.5.2. eXtreme Programming (XP)

XP es considerada la primera metodología ágil y de ella se desprenden muchos de sus fundamentos: potenciar relaciones interpersonales, trabajo en equipo, retroalimentación con el cliente y adaptación a los cambios. En XP se definen cuatro variables para cualquier proyecto de software: costo, tiempo, calidad y alcance. De estas cuatro variables, al menos una debe ser establecida por el equipo de desarrollo. Por ejemplo, si el cliente define la calidad y el alcance y el jefe del proyecto el precio, entonces el equipo de desarrollo debe tener la libertad de determinar el tiempo que durará el proyecto.

XP propone un ciclo de vida dinámico, de iteraciones cortas y con un entregable al finalizar cada ciclo. Las fases son las siguientes:

- Exploración: Aquí se define el alcance general del proyecto, mediante las historias de usuario. Además, el equipo de desarrollo estima primariamente los costos de tiempo.
- Planificación: En esta fase se realiza un plan de entregas acordado entre el cliente, los líderes del proyecto y el equipo de desarrollo. Para esto, se deben ordenar las historias de usuario definidas y en consecuencia los entregables.

- Iteraciones: Esta es la fase fundamental de XP ya que aquí se realizan las iteraciones que incluyen análisis, diseño, desarrollo y pruebas. El cliente está involucrado ya que debe colaborar con el refinamiento de la definición de cada historia de usuario.
- Puesta en producción: Aquí se realizan todas las tareas de ajustes necesarios previo a la puesta en producción. No deben incluirse nuevas funcionalidades.
- Mantenimiento: Soporte al cliente una vez que el sistema ya está en producción.
- Muerte del proyecto: Ocurre cuando ya no hay historias de usuario para incluir en el proyecto. Pueden hacerse mejoras de rendimiento, pero sin modificar la arquitectura. Además, se genera la documentación final del sistema.

2.5.3. Crystal Methods (CM)

Se trata de un conjunto de metodologías que se caracterizan por estar centradas en las personas que componen el equipo y en la comunicación. La particularidad es que las metodologías dependen de dos características fundamentales de un proyecto: tamaño y criticidad. Para esto se hace una división por colores: los proyectos más grandes o críticos, que necesitan más coordinación y comunicación, se asocian con colores más oscuros. De esta forma se define que estos proyectos van a requerir mayor coordinación y rigurosidad en el proceso.

2.5.4. Dynamic Systems Development Method (DSDM)

El método de desarrollo de sistemas dinámico (DSDM) está basado en la metodología RAD y tiene un enfoque iterativo e incremental en el que se remarca la participación del usuario. Su objetivo es adaptarse a los cambios en los requerimientos durante el proceso de desarrollo para poder cumplir con las metas de tiempo y presupuesto del proyecto. DSDM consta de tres fases: fase de pre-proyecto, fase de ciclo de vida del proyecto y fase de post-proyecto. La fase de ciclo de vida del proyecto está subdividida en 5 etapas: estudio de viabilidad, estudio de negocio, iteración de modelo funcional, iteración de diseño y construcción e implementación. [16]

2.5.5. Adaptive Software Development (ASD)

La principal diferenciación que hace esta metodología es la revisión de los componentes de software para adaptarse a los cambios y corregir errores a partir de las lecciones aprendidas en iteraciones previas. El ciclo de vida tiene tres fases:

- Especulación: Se da inicio al proyecto y se planifican las características del software. Se dice que es especulativo ya que se asume que todos los actores tienen alguna dificultad para definir estos elementos.
- Colaboración: Aquí se desarrollan las características del software definidas. Es colaborativo ya que se intenta balancear el trabajo bajo condiciones predecibles e impredecibles.
- Aprendizaje: Aquí se revisa la calidad de lo desarrollado y se entrega al cliente. El aprendizaje se da fundamentalmente a partir de la corrección de los errores detectados.

2.5.6. Feature-Driven Development (FDD)

El desarrollo basado en funcionalidades define un proceso iterativo en 5 fases:

- Desarrollar un modelo global: se realiza una inspección de alto nivel del alcance del sistema y su contexto. Se realizan pequeños modelos de distintas partes del sistema para luego integrarlos en un modelo global.
- Construir una lista de funcionalidades: A partir del modelo anterior se define una lista de funcionalidades.
- Planificar por funcionalidad: Se planifica el desarrollo de las funcionalidades por separado, estableciendo un encargado para cada una.
- Diseñar por funcionalidad: Se realiza un diseño de cada funcionalidad agrupando aquellas que puedan construirse en conjunto.
- Construir por funcionalidad: se construye y prueba cada funcionalidad.

2.5.7. Lean Development (LD)

El desarrollo *Lean* (término que hace referencia a algo fino o esbelto) está basado en el proceso de producción utilizado originalmente por compañías automotrices. LD puede definirse bajo siete principios:

- Eliminar los desperdicios.
- Amplificar el aprendizaje.
- Decidir lo más tarde posible.
- Entregar tan rápido como sea posible.
- Capacitar al equipo.
- Construir integridad intrínseca.
- Ver todo el conjunto.

2.5.8. Scrum

2.5.8.1. Breve historia

En 1986 Hirotaka Takeuchi e Ikujiro Nonaka describieron un enfoque integral para incrementar la velocidad y flexibilidad del desarrollo de nuevos productos comerciales. Allí utilizaron distintas analogías con el rugby. En 1991 DeGrace y Stahl definieron a este enfoque como Scrum, haciendo una comparación con la formación del rugby en donde los compañeros de un equipo unen fuerzas para empujar hacia adelante.

A principios de la década de los ´90, Ken Schwaber y Jeff Sutherland aplicaron esta metodología en sus respectivas empresas. En 1995, ambos presentaron en forma conjunta un artículo describiendo Scrum [17]. En 2001, Schwaber y Mike Beedle se asociaron para clarificar la definición de la metodología en el libro Agile Software Development with Scrum [7].

2.5.8.2. Definición de SCRUM

En la Guía de Scrum [6], se define a esta metodología como un *framework* en el que las personas pueden hacer frente a problemas de adaptación complejos. El resultado debe ser la entrega de productos con el mayor valor posible, de forma productiva y creativa. La idea no es definir un proceso o una técnica, sino más bien un marco en el cual se puedan combinar distintas técnicas o procesos. Este framework se conforma por distintos equipos, roles, eventos, artefactos y reglas. Sutherland y Schwaber [6] promocionan Scrum con estos 3 conceptos:

- Liviano
- Simple de comprender
- Difícil de dominar

Scrum fomenta la creación de equipos auto organizados y multidisciplinares. Idealmente, todos los miembros de un equipo deben trabajar en la misma ubicación, para favorecer la comunicación verbal y cara a cara.

Uno de los principios claves de Scrum es el de reconocer que es imposible definir antes de comenzar un proyecto todo lo que el cliente necesita. Es decir que se trata de un proceso de descubrimiento dinámico en el que en cada iteración se acerca más a la solución deseada. Otro de los principios en el que se apoya es en el empirismo: se deben maximizar los conocimientos, habilidades y experiencias del equipo para afrontar los problemas que surjan a lo largo del proyecto. Por más capacidad que se tenga, no es posible diseñar una solución completa antes de iniciar el proyecto.

2.5.8.3. Fundamentos de Scrum

Scrum utiliza un enfoque iterativo e incremental en el que se le da mucho valor al conocimiento que procede de la experiencia. En [6] se destacan

tres pilares que soportan toda la implementación del control de procesos empírico:

- Transparencia: Los aspectos más importantes del proceso deben ser visibles para todos aquellos que son responsables del resultado.
- Inspección: Se deben inspeccionar frecuentemente los artefactos de Scrum y el progreso hacia un objetivo, para detectar variaciones.
- Adaptación: Si luego de la inspección se detecta que algún aspecto se desvió más allá de los límites aceptables, debe ajustarse lo más rápido posible.

2.5.8.4. El Equipo de Scrum

Los equipos de Scrum son auto organizados, ya que eligen la mejor forma de llevar a cabo su propio trabajo, y son multifuncionales ya que sus integrantes tienen la capacidad de cumplir todos los roles necesarios. Un equipo de Scrum está compuesto por:

- El Dueño de Producto (*Product Owner*): es el responsable de maximizar el valor del producto y del trabajo del Equipo de Desarrollo. Es la única persona responsable de gestionar la Lista del Producto (*Product Backlog*), tanto de su contenido como de su priorización.
- El equipo de Desarrollo (*Development Team*): está compuesto por los profesionales que se encargan de realizar un "incremento" en el producto que potencialmente puede ponerse en producción al finalizar un *Sprint*. Todos los integrantes son "desarrolladores", independientemente de la actividad que realicen. No pueden existir "sub-equipos" dentro del equipo de desarrollo. Normalmente se aconseja que la cantidad de integrantes en el equipo de desarrollo sea de entre 4 y 8 personas. Debe ser lo suficientemente pequeño como para permanecer ágil y lo suficientemente grande como para poder cumplir con todas las aptitudes necesarias para desarrollar un incremento significativo.
- El Scrum Master: Es el encargado de asegurar que Scrum sea comprendido y llevado a cabo bajo su teoría, reglas y prácticas. Es un líder que se debe ocupar de ayudar a optimizar las interacciones tanto de miembros del equipo como de personas externas. Además, debe interactuar con el *Product Owner* para optimizar la gestión del *Product Backlog*.

2.5.8.5. Eventos de Scrum

Scrum plantea una serie de eventos predefinidos con el objetivo de generar cierta regularidad y minimizar las reuniones no planificadas. Cada evento tiene un fin específico, conocido por todos los integrantes del equipo, y una duración máxima. Algunos eventos, como el *Sprint*, tienen una duración

fija, es decir que no pueden alargarse ni acortarse. Otros eventos pueden terminar cuando se alcance su objetivo, pero no pueden superar el tiempo establecido. Cada uno de los eventos de Scrum es visto como una oportunidad formal para inspeccionar y adaptar algún aspecto del proceso.

El Sprint

El *Sprint* es un período de tiempo normalmente recomendado de entre 2 a 4 semanas en el que se genera un incremento "terminado" del producto. Dicho incremento debe poder utilizarse y potencialmente ponerse en producción. Los *Sprints* se suceden de forma continua, es decir, que cada uno comienza inmediatamente después de la finalización del *Sprint* anterior. El *Sprint* tiene un objetivo planificado al comienzo del mismo y no puede modificarse en ningún momento.

Los *Sprints* contienen y consisten de la Reunión de Planificación del *Sprint* (*Sprint Planning Meeting*), los Scrums Diarios (*Daily Scrums*), el trabajo de desarrollo, la Revisión del *Sprint* (*Sprint Review*), y la Retrospectiva del *Sprint* (*Sprint Retrospective*).

Reunión de Planificación de *Sprint* (*Sprint Planning Meeting*)

Se trata de una reunión de la que participa todo el equipo de Scrum y tiene como fin planificar lo que se desarrollará durante el *Sprint* que comienza. Para un *Sprint* de un mes, el tiempo máximo de la reunión está pautado en 8 horas. Para iteraciones más cortas, la duración debería ser proporcional a este tiempo.

El *Scrum Master* debe liderar la reunión asegurando que todos los integrantes del equipo entiendan el motivo y la dinámica de la misma, y propiciar la participación activa. Además, debe velar porque la reunión pueda concluirse dentro del tiempo establecido.

Para poder llevar a cabo esta reunión se debe contar con la siguiente información: la Lista de Producto priorizada, el último incremento del producto, la capacidad proyectada del equipo de Desarrollo y su rendimiento en el *Sprint* anterior.

La reunión de planificación debe determinar dos cuestiones: qué puede entregarse como incremento resultante del *Sprint* y cómo se realizará este trabajo. En primer lugar, el Dueño de Producto expone un posible objetivo junto con los elementos de la Lista de Producto que lo cumplirían. El Equipo de Desarrollo debe definir si es posible alcanzar a completar todos estos elementos. Para determinar esto, se tiene en cuenta la capacidad proyectada y el rendimiento pasado del equipo de desarrollo. Solo el Equipo de Desarrollo puede determinar la cantidad de elementos que es posible desarrollar en el tiempo que dura el *Sprint*. Una vez hecho esto, todo el equipo de Scrum elabora un Objetivo del *Sprint* (*Sprint Goal*), el cual proveerá una idea al equipo de Desarrollo de por qué se está construyendo el incremento.

Luego de esto, el equipo de Desarrollo debe determinar cómo llevará a cabo la implementación de los elementos de la Lista de Producto definidos. Los elementos de la Lista de Producto seleccionados para este *Sprint*, más el plan para terminarlos, se denomina Lista de Pendientes del *Sprint* (*Sprint Backlog*). Para elaborar el plan, se deben llevar a cabo durante la reunión debates de diseño de la solución y trabajos necesarios para convertir la Lista de Producto en un Incremento funcional. En estos debates puede ser necesaria la participación del Dueño de Producto para explicar algunas cuestiones o incluso de otras personas por fuera del equipo que puedan brindar asesoría técnica o de dominio. Durante la reunión se debe descomponer cada uno de los elementos de la Lista de Producto en unidades que puedan concretarse en un día o menos. Al finalizar la reunión, el Equipo de Desarrollo debería ser capaz de explicar al Dueño de Producto y al *Scrum Master* cómo pretende llevar a cabo el trabajo de forma auto organizada para, de esta manera, lograr el Objetivo del *Sprint* y crear el Incremento esperado.

Objetivo del *Sprint* (*Sprint Goal*)

Es la meta establecida para el *Sprint* durante la reunión de Planificación. El cumplimiento del Objetivo del *Sprint* comprende la implementación de un conjunto de elementos de la Lista de Producto. Además, sirve como guía para que el equipo de desarrollo entienda por qué está construyendo el incremento.

Scrum Diario (*Daily Scrum*)

El Scrum Diario es una reunión llevada a cabo por los miembros del Equipo de Desarrollo para sincronizar, revisar y planificar el trabajo del día a día. Tiene un tiempo máximo preestablecido de 15 minutos o de 2 minutos por cada integrante. El *Scrum Master* no debe liderar la reunión, pero debe asegurar que se lleve a cabo en tiempo y forma. La hora y lugar de esta reunión debe ser siempre igual para lograr una regularidad y acostumbramiento.

Durante la reunión, cada uno de los integrantes debe exponer la respuesta a las siguientes preguntas:

- ¿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*?

Siempre se deben respetar los dos minutos por integrante, por lo que, de existir alguna discusión o duda que amerite más tiempo, las personas involucradas deben reunirse inmediatamente después de la reunión.

El Scrum Diario permite evaluar durante todo el *Sprint* cómo es el avance hacia el Objetivo y tomar decisiones rápidas ante eventuales

problemas. Además, mejoran la comunicación y la distribución de conocimiento.

Revisión de *Sprint* (*Sprint Review*)

En el último día del *Sprint*, se debe llevar a cabo una reunión de Revisión del *Sprint* para inspeccionar el Incremento y actualizar el avance de la Lista de Producto. El tiempo de la reunión está establecido en 4 horas para una iteración de un mes. Como en otras reuniones, el *Scrum Master* principalmente debe velar por el cumplimiento del tiempo máximo y el entendimiento del objetivo de la reunión. Los participantes de esta reunión son: el *Scrum Master*, el Equipo de Desarrollo, el Dueño de Producto y todos los interesados en el Incremento que se va a presentar.

Durante la Revisión del *Sprint* básicamente se debe definir qué elementos de la Lista de Producto se han "terminado" y cuáles no. Para esto, los integrantes del Equipo de Desarrollo deben explicar las dificultades que tuvieron y cómo las resolvieron. Además, deben responder todas las consultas hechas al respecto por el Dueño de Producto o los invitados interesados. Por otro lado, en base al avance hecho sobre la Lista de Producto, se debe planificar un potencial próximo Incremento. Esta información será valiosa para la reunión de Planificación del *Sprint* siguiente.

Retrospectiva de *Sprint* (*Sprint Retrospective*)

La Retrospectiva de *Sprint* es una reunión de todo el Equipo de Scrum para inspeccionarse a sí mismo y planificar mejoras para las siguientes iteraciones. Debe realizarse después de la Revisión de *Sprint* y antes de la reunión de Planificación del siguiente *Sprint*. El tiempo máximo preestablecido es de 3 horas para un *Sprint* de un mes. El *Scrum Master* debe cumplir su rol habitual de controlar el tiempo de la reunión y asegurar el entendimiento de la misma, pero además debe participar activamente como un miembro más del equipo ya que la responsabilidad del proceso de Scrum depende de él.

El objetivo de la reunión es inspeccionar el último *Sprint* con respecto a los integrantes del equipo, sus relaciones, procesos y herramientas. Se deben destacar los aspectos positivos y negativos de cada elemento, proponiendo posibles mejoras. A su vez, las propuestas deben estar acompañadas de un plan para poder implementarlas.

2.5.8.6. Artefactos de Scrum

Los artefactos de Scrum son elementos pensados para maximizar la transparencia de la información, la cual es necesaria para asegurar que todos los individuos que participan del proceso vean cada elemento de la misma forma. Además, tal cual es el espíritu de Scrum, los artefactos deben facilitar oportunidades para la inspección y adaptación del mismo proceso.

Lista de Producto (*Product Backlog*)

La Lista de Producto es una lista ordenada de elementos que describen necesidades de cambios en el producto. Estos cambios pueden ser nuevas características, funcionalidades, requisitos, mejoras o correcciones. La Lista de Producto es el único repositorio en donde se pueden encontrar todos los cambios a realizar sobre el Producto, por más que exista más de un Equipo de Scrum. En ese caso, es posible diferenciar los elementos a través de algún atributo. Como mínimo, cada elemento de la lista debe tener una descripción, un orden, una estimación y un valor. El Dueño de Producto es el responsable de gestionar su contenido, pero el refinamiento de la lista puede debatirse con el Equipo de Scrum. Refinar la lista significa dar mayor granularidad y detalle a las definiciones, estimar y ordenar los elementos. Este es un proceso continuo, que evoluciona a medida que avanzan los *Sprints* y se agregan elementos a la lista. El ordenamiento de la lista es una cuestión fundamental, ya que se considera que los elementos más prioritarios se tomarán primero a la hora de la planificación. El nivel de detalle de estos elementos debe permitir que pueda desarrollarse dentro de un *Sprint*.

Nunca puede considerarse que la Lista de Producto está completa, ya que se trata de una lista que cambia permanentemente. Puede verse afectada por distintas cuestiones como modificaciones del negocio, condiciones del mercado, cambios tecnológicos, etc. La retroalimentación proporcionada por el uso del producto también genera nuevos elementos en la lista.

Para realizar un seguimiento del avance hacia un objetivo, se suelen utilizar algunas prácticas de proyección, como por ejemplo el trabajo consumido (*burndown*), avanzado (*burnup*) y el flujo acumulado (*cumulative flow*). De todas formas, en Scrum se hace hincapié en la importancia del empirismo para la toma de decisiones.

Lista de Pendientes del *Sprint* (*Sprint Backlog*)

La Lista de Pendientes del *Sprint* es un subconjunto de elementos de la Lista de Producto más un plan para desarrollarlos y conseguir, de esta forma, el Objetivo del *Sprint*. Todo esto se obtiene al finalizar la reunión de Planificación del *Sprint*. El Equipo de Desarrollo es quien identifica si es posible generar un Incremento con todos estos elementos dentro de un *Sprint*.

El nivel de detalle del plan debe permitir un seguimiento de forma sencilla, el cual se realiza en la reunión Diaria. El Equipo de Desarrollo puede modificar la Lista de Pendientes a medida que avanza el *Sprint*, ya que puede encontrarse con eventualidades o simplemente porque conoce en mayor profundidad los elementos. Es posible que se incorporen nuevos elementos a la lista o que incluso se eliminen algunos por ser considerados innecesarios. Solo el Equipo de Desarrollo puede cambiar su Lista de Pendientes del *Sprint*. A medida que se van completando los elementos, se actualiza la estimación del trabajo restante.

La Lista de Pendientes del *Sprint* permite ver en tiempo real cómo es el avance del Equipo de Desarrollo hacia la concreción del Objetivo del *Sprint*. Este seguimiento puede revisarse fácilmente en cada reunión Diaria para detectar posibles desvíos.

Incremento

El Incremento es el conjunto de todos los elementos de la Lista de Producto que son considerados como "Terminados" en la reunión de Revisión de un *Sprint*. Todos estos elementos deben estar en condiciones de ser utilizados sin importar si se decide implementarlos en producción.

2.5.8.7. Transparencia de los Artefactos

La Transparencia es un aspecto fundamental de los artefactos de Scrum. El concepto hace referencia a que todos los integrantes de un Equipo deben entender a los artefactos de la misma forma. A mayor transparencia, mayor seguridad habrá en la toma de decisiones para optimizar el valor y controlar el riesgo. Todos los integrantes del Equipo de Scrum deben trabajar para lograr que los artefactos sean completamente transparentes.

2.5.8.8. Definición de "Terminado" (*"Done"*)

Un concepto importante en Scrum es el de "Terminado" ("*Done*"). Se utiliza para determinar que se ha completado el trabajo de un elemento de la Lista de Producto y ya es posible utilizarlo o ponerlo en producción. Todo el Equipo de Scrum debe acordar y entender cuándo debe considerarse un elemento como "Terminado". Todos los integrantes de un Equipo, o todos los Equipos de Scrum que trabajan sobre una Lista de Producto, deben acordar de forma clara lo que significa el concepto de "Terminado". Ésta es una de las maneras de aumentar la Transparencia en la metodología.

El concepto de "Terminado" puede ampliarse o mejorarse para obtener una mayor calidad del producto. Esto se logra a través del tiempo y de la maduración de los Equipos de Scrum. Siempre debe tenerse en cuenta que cualquier cambio debe ser comprendido por todos los integrantes.

Capítulo 3

Revisión de literatura

3.1. Comparación entre ambas metodologías

Las metodologías ágiles nacieron en los años 90 en contraposición a las metodologías tradicionales. Se conocen como metodologías pesadas o tradicionales, a aquellas que se centran en la definición detallada de los procesos, tareas, y herramientas a utilizar, y requieren de extensa documentación. Algunas (como el modelo en cascada) pretenden prever todo de antemano.

Este tipo de metodologías fueron utilizadas durante mucho tiempo. Su objetivo era que la construcción de software sea lo más predecible y eficiente posible. Fowler marca que su principal desventaja es la gran burocracia que agregan al proceso. Esto hace que en muchos casos se deban realizar tareas que no aporten a la velocidad ni a la calidad del resultado.

En cambio, de las metodologías ágiles se suelen destacar las siguientes características (extraídas de los valores del Manifiesto Ágil [21]):

- Se encargan de valorar al individuo y las iteraciones del equipo más que a las herramientas o los procesos utilizados.
- Se hace mucho más importante crear un producto de software que funcione que escribir mucha documentación.
- El cliente está en todo momento colaborando en el proyecto.
- Es más importante la capacidad de respuesta ante un cambio realizado que el seguimiento estricto de un plan.

La agilidad proviene de no atarse estrictamente a la generación de documentación ni al seguimiento de un plan, sino, adaptarse flexiblemente a los cambios y al desarrollo de un producto funcional.

Canós y Letelier [4] ofrecen el siguiente cuadro que compara ambas metodologías:

Tabla 3.1. *Diferencias entre metodologías Ágiles y Tradicionales*

Metodologías Ágiles	Metodologías Tradicionales
Basadas en heurísticas provenientes de prácticas de producción de código	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo

Especialmente preparados para cambios durante el proyecto	Cierta resistencia a los cambios
Impuestas internamente (por el equipo)	Impuestas externamente
Proceso menos controlado, con pocos principios	Proceso mucho más controlado, con numerosas políticas/normas
No existe contrato tradicional o al menos es bastante flexible	Existe un contrato prefijado
El cliente es parte del equipo de desarrollo	El cliente interactúa con el equipo de desarrollo mediante reuniones
Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio	Grupos grandes y posiblemente distribuidos
Pocos artefactos	Más artefactos
Pocos roles	Más roles
Menos énfasis en la arquitectura del software	La arquitectura del software es esencial y se expresa mediante modelos

3.2. Cuestiones a tener en cuenta al migrar de una metodología clásica a una ágil

Almeida [11] hace una revisión de la literatura sobre las cuestiones a tener en cuenta cuando se migra de una metodología clásica a una ágil. Luego del análisis de los casos de estudio de mayor relevancia, se define una agrupación de los desafíos en 4 categorías principales:

- Organización y administración
- Personas
- Procesos
- Herramientas

3.2.1. Organización y administración

La migración hacia una metodología ágil afecta a toda la estructura de la organización. Las áreas de gestión, dirección, desarrollo, etcétera, están compuestas por personas que tienen diferentes costumbres y conocimientos.

Es común no solo encontrar resistencia al cambio, sino también un acostumbramiento al seguimiento de planes y tareas muy estrictas.

Las metodologías ágiles proponen una gran participación y comunicación de todos los integrantes de la organización. La distribución de conocimiento es un valor clave para conseguir los objetivos. Sin embargo, es común encontrar problemas de comunicación, sobre todo entre distintos sectores, como así también problemas para trabajar en equipo y relaciones ad-hoc entre los mismos. De esta forma, se aíslan las áreas y aumenta la resistencia al cambio de metodología.

Las barreras culturales son otro desafío importante a vencer, debido a los largos períodos de tiempo en los que se utilizaron metodologías tradicionales. Sin embargo, la propuesta de cambio puede funcionar como un factor motivacional.

Por último, suelen encontrarse inconvenientes de origen legislativo, debido a los cambios en reglas y procedimientos. Sobre todo, en grandes proyectos en los que hay muchas compañías involucradas o incluso con organizaciones gubernamentales. El cambio de metodología debe respetar estas reglas y en ocasiones debe hacer algunas concesiones. Otro punto relacionado a esto son las auditorías externas, que normalmente están pensadas para metodologías más tradicionales.

En la figura 3.1 se puede ver un mapa conceptual con un resumen de estos desafíos:

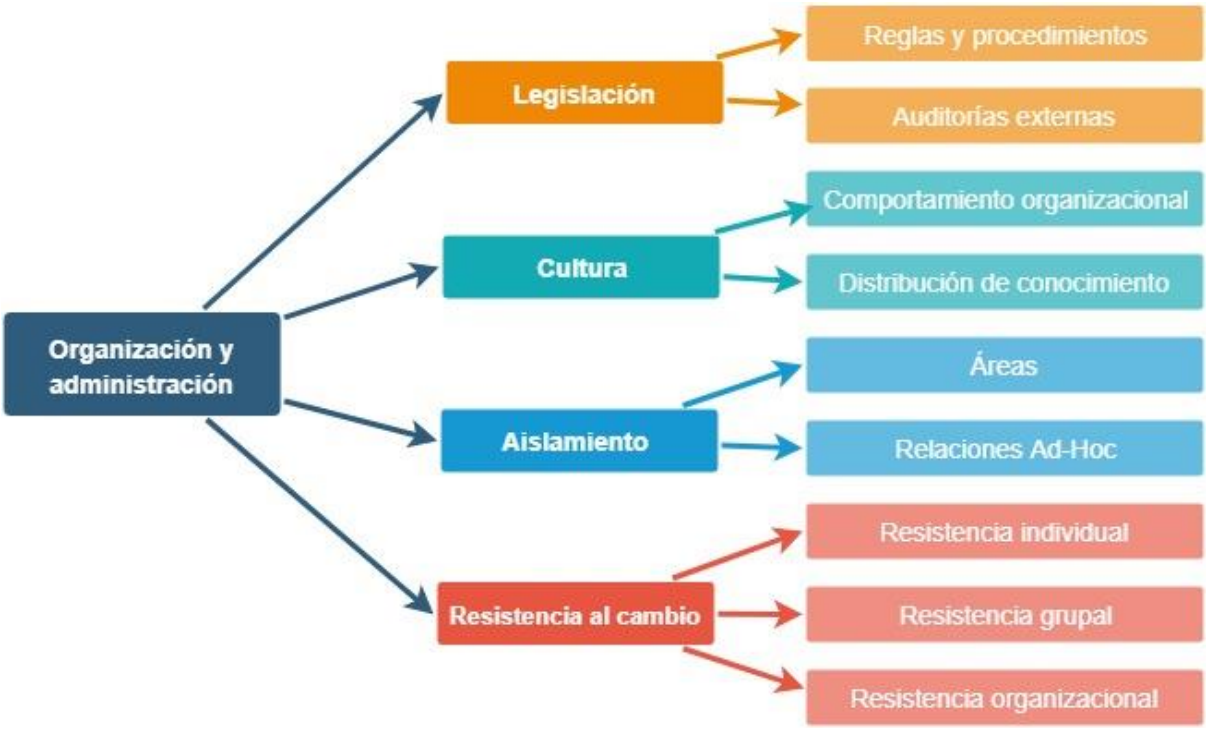

***Figura 3.1**. Mapa conceptual de desafíos de "Organización y administración"*

3.2.2. Personas

La falta de conocimiento sobre las metodologías ágiles es una de las primeras problemáticas. No es común que en ambientes universitarios se introduzcan demasiados contenidos relacionados. Aún hoy se sigue haciendo foco en metodologías más tradicionales. Para mitigar estos problemas, se suelen utilizar cursos introductorios, aunque no son tan comunes y el tiempo que consumen se convierte en otro obstáculo.

La experiencia es un factor muy importante en toda metodología. Como las metodologías ágiles todavía son novedosas, es muy difícil encontrar personas con mucha experiencia u obtener información de proyectos previos que las hayan utilizado. Otro factor a tener en cuenta es el compromiso de los empleados en la implementación de la metodología. Para llevar a cabo esta tarea se requiere mucha adaptabilidad, de manera que una persona pueda cumplir diferentes roles. La mejor alternativa para lograr esto es conseguir que los equipos se auto organicen, eligiendo el rol que cada uno va a cumplir, en lugar de que sea asignado por un directivo externo al equipo.

Otro de los desafíos a enfrentar se da cuando miembros del equipo u otras personas interesadas se encuentran en distintos lugares geográficos. La cohesión del equipo en las metodologías ágiles es una propiedad fundamental, por lo que hay que buscar formas de mitigar esta dificultad. Algunas opciones son las visitas regulares, la participación en todas las reuniones, interacción frecuente entre los miembros del equipo o la utilización de herramientas que favorezcan la comunicación.

Por último, la participación en todo el proceso de las personas interesadas y la correcta identificación de las necesidades del cliente, son factores fundamentales para el éxito de las metodologías ágiles.

En la figura 3.2 se puede ver un mapa conceptual con un resumen de estos desafíos:

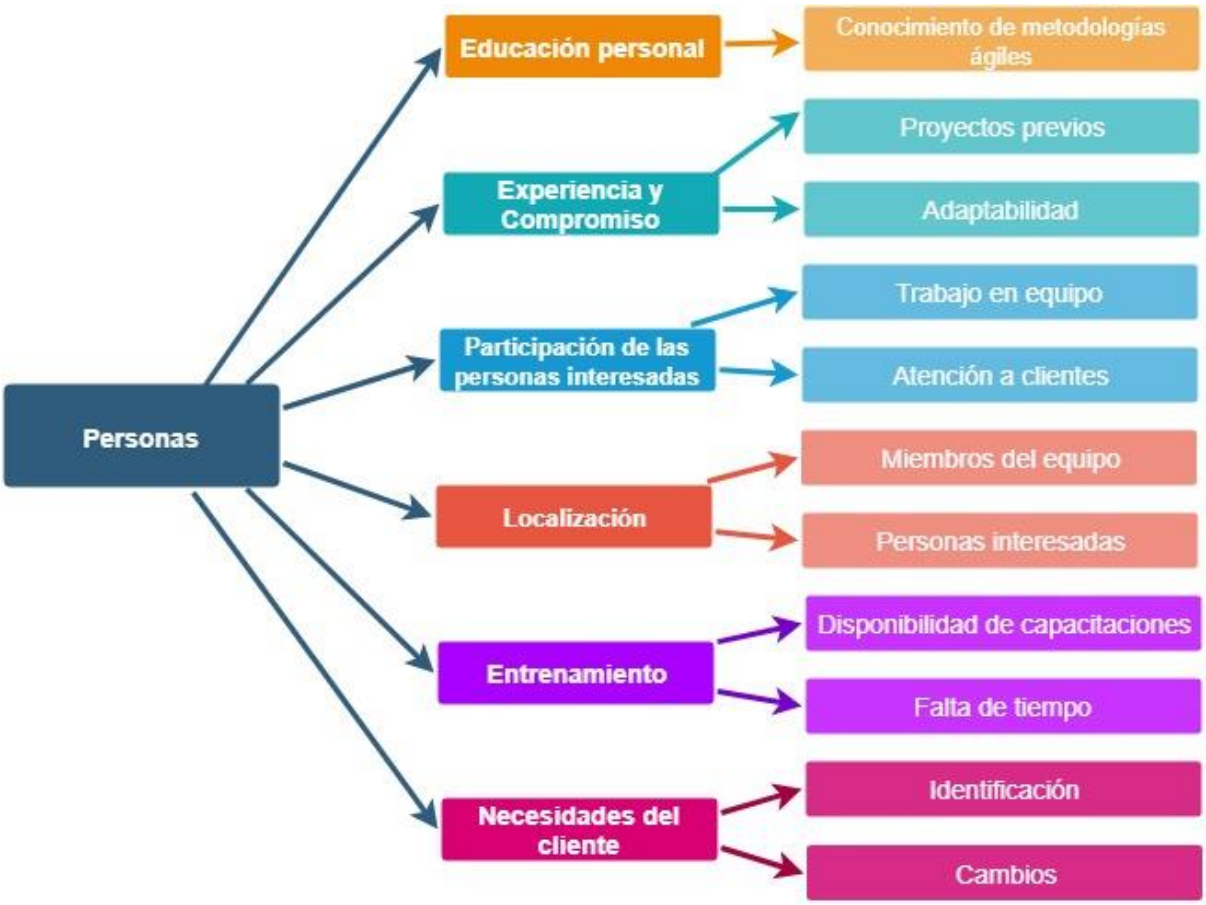

***Figura 3.2**. Mapa conceptual de desafíos relacionados con las personas.*

3.3.3. Procesos

En las metodologías ágiles se requiere que los equipos puedan dividirse los distintos tipos de tareas para poder cumplir con el objetivo. El desafío principal es realizar esta tarea teniendo en cuenta las distintas personalidades y conocimientos de los integrantes del equipo. Otro punto importante es encontrar a uno de ellos con las aptitudes necesarias para llevar a cabo el liderazgo del equipo, las cuales suelen ser naturales y no pueden forzarse.

La identificación de los requerimientos en las metodologías ágiles es un proceso dinámico que se realiza a lo largo de todo el proyecto. El objetivo es encontrar todos los requerimientos funcionales y no funcionales para cumplir con las necesidades del cliente. Se debe enfatizar que es necesario identificar ambos tipos de requerimientos, ya que los segundos muchas veces son dejados en segundo plano. Para evitar que falte identificar algún requerimiento, es necesaria la participación de todos los interesados en el producto durante todo el proceso. Otro desafío suele ser lidiar con ciertos requerimientos conflictivos. Esto no debe tomarse como un problema ya que es natural que suceda. En cambio, debe tomarse como una oportunidad para asegurar que el requerimiento cumpla con la expectativa del cliente.

Ponderar las tareas productivas por sobre la documentación es un fundamento conocido de las metodologías ágiles. Sin embargo, este concepto muchas veces se toma de forma equivocada, relegando completamente a la documentación. Lejos de tener que eliminarla, es necesario construir un proceso de documentación de forma colaborativa. Para esto, es posible utilizar herramientas que faciliten el trabajo o incluso tener un rol específico que se encargue de esta tarea.

Cuando se trabaja en grandes proyectos, la interacción y la dependencia entre los equipos se convierte en un desafío. Más aún, cuando los equipos que participan son heterogéneos en cuanto a capacidades y conocimientos. Una buena opción es tener equipos que puedan desarrollar funcionalidades de forma "*full-stack*", sin dependencias externas.

Otra de las necesidades en todo proyecto es la de poder medir y estimar. Se pueden encontrar diversos estudios referidos al tema, pero como rasgo común se puede mencionar la necesidad de involucrar a todo el equipo, con un fuerte compromiso para llevar a cabo ambas tareas.

La calidad de todo el proceso de desarrollo debe ser un objetivo esencial de toda metodología. En las metodologías ágiles se utilizan distintas estrategias de calidad como la refactorización, revisión e inspección, además del seguimiento de estándares y lineamientos conocidos. Otro aspecto importante que favorece a la calidad es la entrega rápida de pequeños incrementos del producto, facilitando el testing interno y externo.

La gestión de riesgos es una tarea que no suele encontrarse en la implementación de metodologías ágiles ya que en general se toma como un trabajo burocrático que no va con el espíritu de este tipo de metodologías. Sin embargo, siempre se recomienda identificar, priorizar y gestionar los riesgos posibles.

Por último, es importante plantear el desafío de la escalabilidad del proceso en relación al tamaño o la complejidad del proyecto. Algunas cuestiones a tener en cuenta cuando el proyecto crece en tamaño son: cómo mantener la velocidad de desarrollo, cómo coordinar las dependencias entre equipos o cómo mantener la misma calidad.

En la figura 3.3 se puede ver un mapa conceptual con un resumen de estos desafíos:

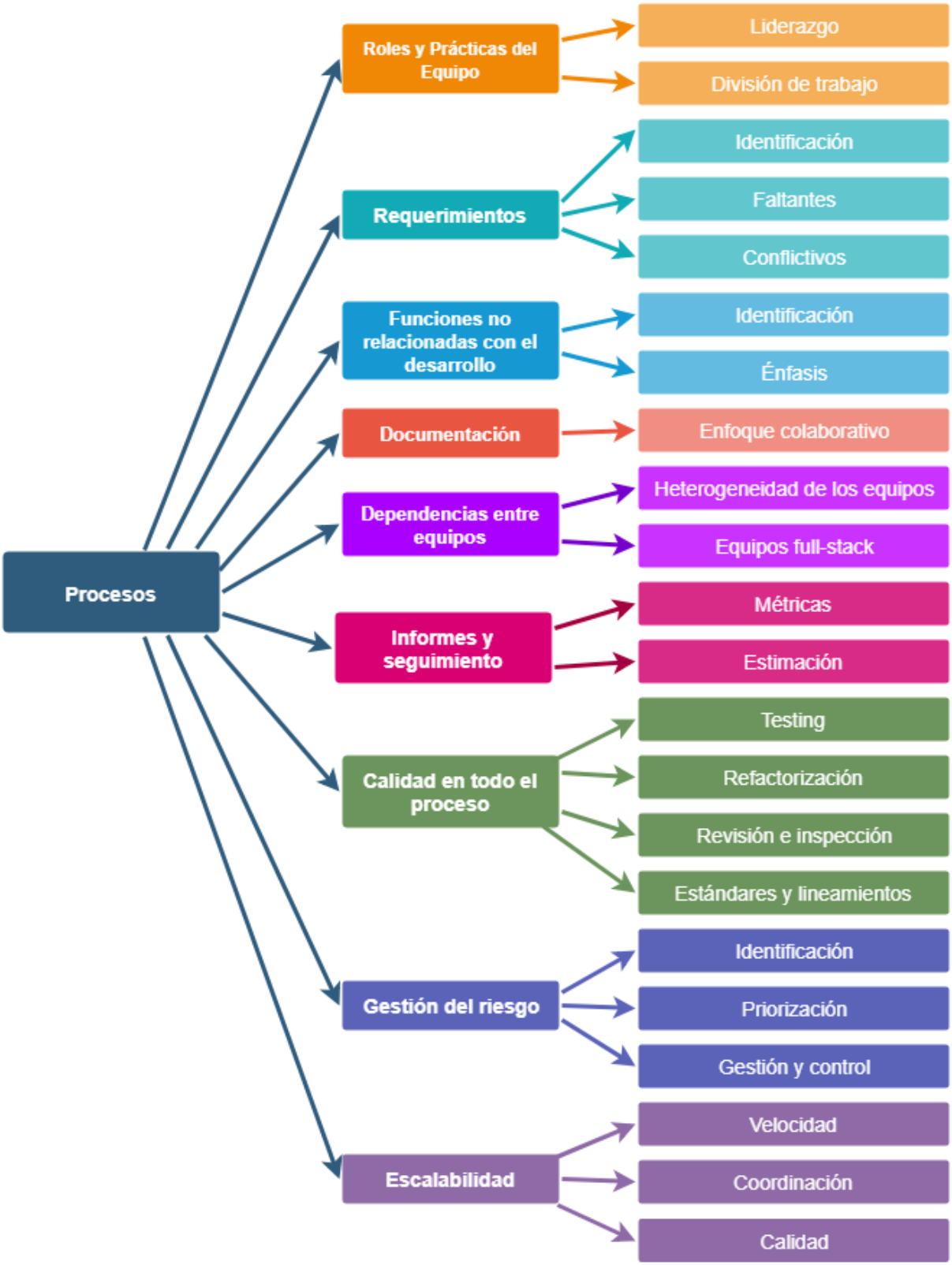

***Figura 3.3**. Mapa conceptual de desafíos relacionados con los Procesos.*

3.3.4. Herramientas

La complejidad de la arquitectura del software que se está construyendo se debe tener en cuenta a la hora de implementar una metodología ágil. A medida que se incrementa la complejidad técnica se va a requerir una mejor comunicación entre los equipos y una integración de las distintas partes del sistema que se construye. Para esto se requiere mayor gestión y control que en proyectos más simples. Por otra parte, es común que en proyectos de esta envergadura se presenten otros desafíos como la integración con proveedores externos (librerías, *frameworks*, herramientas, etc.) y la necesidad de interactuar con múltiples dueños de producto.

En una metodología ágil, la identificación de los requerimientos es un proceso constante que ocurre a medida que avanza el desarrollo. Un problema común suele ser que nunca llegan a lograrse los resultados esperados. Con lo cual, es menester definir criterios de aceptación que permitan avanzar en la construcción de los incrementos del producto. Por otra parte, para poder evaluar el avance del proyecto, se deben utilizar algunos mecanismos como por ejemplo la integración continua. La idea es tener un proceso automatizado que permita ser ejecutado de forma recurrente para así poder detectar rápidamente posibles fallas.

Por último, el seguimiento de los problemas reportados es un desafío en una metodología ágil, ya que normalmente un error encontrado en un *Sprint* corresponde a un desarrollo de un *Sprint* anterior. Para esto, se debe tener alguna herramienta que centralice el reporte y que pueda ser utilizada tanto por los integrantes del equipo como por personas externas.

En la figura 3.4 se puede ver un mapa conceptual con un resumen de estos desafíos:

***Figura 3.4**. Mapa conceptual de desafíos relacionados con las Herramientas.*

Capítulo 4

Contexto

En este capítulo se describirán algunas características de la empresa a la que pertenezco y de los productos que elabora. Estas características dan el marco en el cual se realizó una migración de una metodología tradicional a una ágil y que será la base de este estudio.

4.1. Acerca de la empresa

Se trata de una empresa que desarrolla software de seguros desde principios de los ‘90. Es el principal proveedor de este tipo de productos en América Latina, en donde tiene a los clientes más importantes del rubro.

4.1.1. Organigrama

En la empresa se pueden encontrar las siguientes gerencias:

- Gerencia Comercial: Esta área se encarga de la comercialización de todos los productos que desarrolla la empresa.
- Gerencia de Proyectos: Se encarga de la planificación y seguimiento de todos los proyectos ya sean nuevos o de mantenimiento. Contiene dos sectores:
    - Operaciones: Se encarga de la elicitación de requerimientos. Es el sector que interactúa con los clientes para definir el marco del proyecto o alguna modificación posterior a la implementación. También realiza QA funcional.
    - Soporte: Se encarga de interactuar con el cliente ante eventuales problemas con el uso de los productos.
- Gerencia de Producto: Se encarga de definir las características tanto funcionales como de diseño que van a tener los productos. También valida que las modificaciones solicitadas por los clientes puedan incorporarse al producto.
- Gerencia de Desarrollo: Se encarga de la construcción de todos los productos de la empresa. Dentro del área de desarrollo existe un sector de QA que valida la correcta implementación de todos los requerimientos.

4.1.2. Sobre los productos que desarrolla

La empresa se especializa en el desarrollo de productos relacionados al mundo de los seguros. Si bien se desarrollan diversos productos, los troncales son 3: para brokers de seguros, para reaseguros y para bancos. El primero de ellos es el más importante ya que tiene su raíz en los primeros años de la empresa con una versión en DOS. Con el tiempo tuvo dos grandes migraciones: a una aplicación de escritorio desarrollada en Delphi y luego a una aplicación Web desarrollada en .NET. En esta tesina se hará hincapié solo en los proyectos relacionados a este producto, no solo porque es la práctica en la que trabajo, sino también porque es el producto más complejo de la empresa.

El valor fundamental de los productos es el conocimiento del dominio que adquirió la empresa a lo largo de los años. Más allá de las migraciones tecnológicas, la lógica de dominio se mantuvo y fue creciendo con cada proyecto.

Un proyecto nuevo siempre se basa en la implementación del *core* de un producto (utilizando cuestiones de seguridad para ocultar determinados módulos), al cual se le agregan todas las customizaciones necesarias del cliente (por ejemplo, mediante parámetros del sistema o *plugins*). Estos agregados pueden incorporarse al *core* si el área de Producto decide que puede ser útil para otros clientes.

Con el paso de los años el producto fue creciendo en complejidad técnica y en tamaño. Ya que siempre se trabaja sobre el mismo producto, crece prácticamente de forma indefinida. Actualmente, se trata de un sistema web en N capas, orientado a servicios, que abarca distintos módulos relacionados a la gestión de pólizas, cobranzas, honorarios, facturaciones, etc. De la misma forma, mantiene un modelo de base de datos creciente que contiene más de mil tablas. Desde el comienzo se diseñó una arquitectura robusta haciendo hincapié en separar la lógica de negocio de los artefactos propios de la arquitectura. El *BackEnd* se compone de las siguientes capas: persistencia (con un ORM), DAO, *Business Logic* (utiliza *Business Entities*), *Models* y *Services* (utiliza DTOs). La intención del *FrontEnd* es que sea sumamente liviano y consuma toda la lógica a través de los servicios expuestos por el *BackEnd*.

Al ser un único producto que se utiliza en diversos clientes, se tiene una gran cantidad de parametrizaciones que permiten customizarlo. Si bien existen manuales y documentaciones sobre las funcionalidades principales del producto, no abarcan todas estas particularidades, ni llegan al detalle fino de cada una. Se trata de un producto de gran tamaño y complejidad.

4.2. Sobre los Clientes

Sin dudas que el principal del valor de la empresa es el conocimiento que tiene sobre los negocios de seguros. Seguramente sea este el motivo principal por el que las grandes empresas de la región relacionadas con este negocio la eligen como proveedor. La mayor parte de los clientes lleva muchos años utilizando el mismo producto y esto hace que gran parte de sus áreas y procesos lo tengan muy arraigado, generando una importante dependencia. Es normal y comprensible que, ante la migración hacia una nueva tecnología, los clientes esperen encontrar prácticamente el mismo sistema, incluso sabiendo que se trata de una migración desde una aplicación de escritorio hacia un sistema Web.

Este acostumbramiento se puede ver también en otros niveles, como por ejemplo en la gestión de los proyectos, en las relaciones interpersonales, en el conocimiento de fortalezas y debilidades. Es por eso que se puede decir que se generó a lo largo de los años una especie de cultura de trabajo en varios niveles y que es difícil de modificar.

4.3. Sobre la metodología de trabajo previa a la migración

Antes de la decisión de migrar el proceso de desarrollo se utilizaba un modelo de ciclo de vida en donde el flujo de un requerimiento pasaba por un proceso lineal y estricto similar al definido por el modelo en cascada.

4.3.1. El modelo en Cascada

Para ejemplificar cómo era el proceso de desarrollo, se describe a continuación el flujo de un requerimiento ante un pedido de cambio (SPC):

1. El Cliente solicita la nueva funcionalidad o soporte a través de un documento.
2. El sector de Operaciones realiza un “SPC” con dicho pedido.
3. El líder de Proyecto genera el requerimiento.
4. El líder del equipo de Desarrollo estima las horas que implica la construcción del requerimiento.
5. El líder de Proyecto aprueba la estimación y genera el presupuesto.
6. El Cliente responde con la aprobación del valor del presupuesto.
7. El líder del equipo de Desarrollo asigna la/s tarea/s correspondientes al requerimiento junto con los lineamientos del diseño.
8. El programador lleva a cabo el desarrollo.
9. El líder del equipo realiza un QA técnico.

10. El sector de Operaciones realiza un QA funcional.
11. El Cliente realiza el UAT y la puesta en vivo de la nueva funcionalidad.
12. El Cliente reporta algún problema con la funcionalidad.

Si clasificamos estos pasos dentro del modelo en cascada tenemos lo siguiente:

- Análisis: pasos del 1 al 6.
- Diseño: paso 7.
- Implementación: paso 8.
- Pruebas: pasos 9, 10 y 11.
- Mantenimiento: paso 12.

Para el caso de un proyecto nuevo, el flujo es similar, pero con N requerimientos que deben cumplir cada etapa. El sector de Operaciones se encarga de generar un documento de alcance con todos los requerimientos que debe incluir el proyecto (etapa de Análisis). El área de desarrollo estima y diseña todas las soluciones que deben modificar al *core* del producto (aquí se realiza una interacción con el área de Producto en donde también se incluye el diseño visual). Luego, el líder el equipo de desarrollo genera las tareas para los programadores. Una vez concluidos todos los requerimientos, se genera una versión que primero es testeada por el sector de Operaciones y luego entregada el Cliente para el UAT (etapa de pruebas). Una vez concluido el proyecto, se pasa a la etapa de mantenimiento, en la que todos los reportes de errores pasan por el área de Soporte.

4.3.3. Herramienta de gestión del proceso de desarrollo

La empresa tiene un software propietario para dar soporte a toda la gestión de proyectos. Se trata de una aplicación de escritorio a la que pueden acceder todos los empleados de la empresa. Allí se manejan distintos elementos como: Clientes, Proyectos, Etapas, Subetapas, SPCs, SPSs, Requerimientos, Tareas y Actividades.

Un SPC está compuesto por uno o más requerimientos que cumplen con el pedido del Cliente. El requerimiento tiene los siguientes estados: Falta de Definición, Revisión interna de Producto, Controlado por Producto, Rechazado, Reemplazado por otro Requerimiento, Suspendido, Pendiente de Presupuestación, Presupuestado, En Estimación, Estimado, Aprobado, Listo para Probar, Prueba del Cliente, Finalizado. El flujo de estados es estricto, y solo determinados actores pueden realizar un cambio de estado. Por ejemplo: Un líder de producto puede indicar el estado “Controlado por Producto” o “Rechazado”. Un *Team Leader* de Desarrollo pueda indicar el estado “Estimado” y “Listo para Probar”.

Cada requerimiento tiene asociada una serie de tareas, entre las que se suelen encontrar una para la estimación, una para el análisis y diseño, una o más para el desarrollo y una para pruebas. Los estados que puede tener una tarea son los siguientes: Aprobada, Rechazada, Suspendida, Comenzada, Para prueba de QA, Finalizada. Las tareas están asociadas a una persona en particular que es la encargada de llevarla a cabo. Cada tarea debe tener Actividades asociadas que indican el tiempo dedicado para realizarla. Una Actividad tiene solo dos estados: Comenzada y Finalizada.

4.3.4. Problemas de la metodología

Con la metodología utilizada anteriormente se presentaban varios problemas típicos del modelo en Cascada. El flujo de estados con la intervención de distintos actores genera cuellos de botella importantes. Los recursos nunca son ilimitados, lo cual se hace más visible cuando es un líder quien debe modificar un estado. En un circuito tan estricto, se necesita demasiada disponibilidad de las personas como para que fluya sin demoras, lo cual nunca sucede por diversos motivos como por ejemplo restricciones de seguridad, licencias, ocupación del responsable en otras actividades, etc. Además, en cada etapa debían generarse ciertos documentos y esperar a ser aprobados para continuar con el flujo. Por ejemplo, no puede comenzarse un desarrollo hasta no estar el requerimiento en estado “Aprobado”, con las Tareas creadas y estimadas. Esto trae aparejado que, ante urgencias de tiempo, algunos de los filtros se cumplan con menor calidad de la necesaria. Por ejemplo, aprobaciones de Producto que en realidad no deberían formar parte del *core*, inicio de desarrollos sin la definición necesaria, estimaciones erradas por falta de análisis, pruebas insuficientes, etc.

Un punto importante es que no existían iteraciones de tiempo predeterminado. Más bien, se utilizaba un versionado bajo demanda, en el que distintos clientes terminaban compitiendo a la hora de priorizar los requerimientos a incluir. Esto se debe a que se trabajaba siempre sobre un mismo producto y un único versionado. Un cliente podía ver retrasada su entrega por motivos ajenos a sus necesidades: por ejemplo si se decidía esperar a concluir con requerimientos de otros clientes antes del cierre de versión.

Capítulo 5

Migrando a Scrum

5.1. ¿Por qué modificar la metodología?

La empresa utilizaba una metodología de desarrollo muy antigua, aunque con buenos resultados. El producto principal llevaba mucho tiempo sólo con proyectos de mantenimiento (salvo pequeñas excepciones) con lo cual no había demasiados motivos para modificarlo. Sin embargo, se trataba de un producto con una tecnología también antigua (se trata de una aplicación de escritorio desarrollada con Delphi 4) con lo cual una actualización comenzaba a ser una exigencia de clientes viejos y nuevos.

Si bien la decisión de una modificación de la metodología no fue en simultáneo con el comienzo de la migración de los productos, sí comenzaba a vislumbrarse que la metodología estructurada no era lo más adecuado. La migración comenzó sin proyectos concretos (es decir, sin las exigencias de los clientes) por lo que no era necesario cumplimentar todos los pasos requeridos por el sistema propietario de la empresa que se describió anteriormente. Para evitar esto, se fue abandonando la registración de los requerimientos, con la consecuente pérdida de orden, documentación y trazabilidad. La metodología de trabajo se tornaba cada vez más caótica y difusa.

Por lo tanto, era claramente necesario un cambio que permita dar un marco de trabajo a la migración. El problema se hizo más visible con la llegada del primer proyecto de implementación del nuevo sistema. En ese momento, se decidió optar por Scrum, una metodología mucho más actual y ampliamente utilizada. Es innegable que el mercado influye en las decisiones de la empresa: Scrum estaba (y está) muy en boga en muchas empresas de la región y también suele ser conocido por la mayoría de los desarrolladores, lo cual hace que la transición sea más natural y rápida. La característica de ágil, además, era lo más atrayente para el momento de la empresa.

5.2. Beneficios esperados

El beneficio principal que se buscó con la modificación de la metodología era dar un orden y dirección a la migración del producto en conjunto con la gestión del primer proyecto sobre el nuevo sistema.

Como punto de partida, Scrum invita a definir el *Product Backlog*. Para el proyecto de migración era extremadamente necesario poder conocer todas las funcionalidades pendientes, lo cual en el comienzo era bastante incierto. A su vez, por primera vez se podía tener una diferenciación de funcionalidades *core*

y no *core*, cuáles habían sido pedidas por determinados clientes y cuáles eran más prioritarias. Además, este registro permitía conocer el avance de la migración. En el *Backlog* también se incorporaron las historias de usuario necesarias para cumplir con el primer proyecto concreto, con lo cual se podía tener un panorama completo del trabajo pendiente.

Uno de los pilares fundamentales de la empresa es la especialización en el dominio del mundo de los seguros. Este valor en la experiencia se ve reflejado también en sus empleados: aquellos que tienen mayor tiempo de trabajo y conocimiento de los productos son los que tienen más preponderancia. La opinión del experto es muchas veces la que tiene mayor valor. Este concepto se ve reflejado en los fundamentos de Scrum. A su vez, el trabajo en equipo que propone esta metodología favorece la distribución del conocimiento.

5.3. Desafíos encontrados al migrar a una metodología ágil

En esta sección se presentará un detalle de los desafíos encontrados en la migración a Scrum, enmarcados en el agrupamiento definido por Almeida [11] que se describió anteriormente. Solo se detallarán aquellos puntos en donde se encontraron problemas relacionados con la implementación de la metodología.

5.3.1. Organización y administración

5.3.1.1. Resistencia al cambio

Uno de los problemas principales es claramente la resistencia el cambio en todos los niveles de la empresa:

- Individual: cuando empleados, fundamentalmente de otros sectores distintos al de desarrollo, no comprenden o no tienen interés en cambiar la metodología. También puede darse que afecten a sus intereses personales. Por ejemplo, quien se encarga solo de la facturación, tiene interés en entregar desarrollos cuanto antes, priorizando los que den mayor rédito económico. Otro ejemplo es cuando distintos líderes de proyecto quieren priorizar el propio. Como en la empresa no hay equipos de Scrum dedicados, esta presión termina afectando la planificación de los *Sprints*.
- Grupal: La empresa está divida en distintas áreas, las cuales tienen diferentes formas de trabajo, conocimientos y objetivos. Esto hace que, dependiendo del área, haya mayor o menor interés en la aplicación de Scrum. Siempre se vio como un cambio que afecta solo al sector de desarrollo, cuando debería haber sido comprendido por toda la empresa de la misma forma. Por ejemplo, muchas veces el área de “Operaciones”

posee mayor presión por parte de los clientes para acelerar algún requerimiento o adelantar una entrega, lo cual perjudica la planificación de un *Sprint*, sobre todo cuando los clientes requieren con urgencia un SPC.

- Organizacional: La empresa en su conjunto no tuvo desde el comienzo la idea de implementar Scrum a todo nivel. Esto se puede ver en que no se tomaron acciones de capacitación y planificación de la migración de forma interna y fundamentalmente externa: los clientes no tienen ningún interés en la metodología ni demasiada información al respecto.

5.3.1.2. Cultura

La cultura de trabajo de la organización es otro de los desafíos más importantes al migrar la metodología de trabajo, ya que deben deshacerse de los vicios arraigados. Todos los actores involucrados deben comprender los fundamentos de la nueva metodología y comprometerse a cumplirlos. Ya sea por inercia o desinterés, es común que los cambios sean resistidos.

En el caso de estudio podemos encontrar un ejemplo claro: en la implementación que se hizo de Scrum no hay equipos dedicados por proyecto, al igual que sucedía con la metodología anterior. Un mismo equipo trabaja en simultáneo sobre múltiples proyectos y sobre el mismo producto. Cada incremento de un *Sprint* contiene el avance de una iteración para múltiples proyectos. Cada uno se subordina incluso a la cultura o las políticas de cada cliente. Es decir que puede haber proyectos de larga duración, de clientes mucho más estructurados, compitiendo con otros mucho más cortos, simples o de clientes más permeables al dinamismo de la metodología.

Otro ejemplo puede verse en el flujo de estados de los requerimientos, que en principio se había dejado de lado, pero con el tiempo y de forma paulatina se fue incorporando de forma muy similar a las historias de usuario. Esto se dio por distintos motivos: el área de desarrollo no podía indicar que se necesitaban redefiniciones, se hacían desarrollos sin el control de producto, se iniciaban historias de usuario sin la aprobación de una estimación, etc. Con estas justificaciones se agregaron a la herramienta de Scrum prácticamente los mismos estados que se utilizaban con la herramienta anterior, lo cual arrastró sus problemas inherentes.

También puede verse un problema similar con la cultura de los clientes. Al no haber habido una comunicación, ni haber aplicado un plan de adaptación, los clientes esperan manejarse de la misma forma que lo hicieron siempre. En general se trata de grandes *brokers* y compañías de seguros que tienen un presupuesto ajustado para este tipo de proyectos. Por lo tanto, es normal que requieran una presupuestación inicial antes de iniciarlo, que incluya una definición detallada de todos los requerimientos, costo y estimación de tiempo.

5.3.1.3. Aislamiento

La comunicación es un factor fundamental en Scrum que se ve seriamente comprometido ya que, como quedó dicho, la migración profundizó diferencias entre sectores. Cuando los sectores tienen diferentes visiones sobre la metodología y, por lo tanto, objetivos diferentes, es normal que se produzcan problemas de comunicación y se generen competencias entre los intereses de cada área.

5.3.2. Personas

5.3.2.1. Educación personal

Si bien es cierto que Scrum es una metodología muy conocida, en general se trata de un conocimiento superficial, en el que no se logra comprender completamente el espíritu. Más aún en los sectores no vinculados al desarrollo. La capacitación que se hizo a toda la empresa no alcanzó para subsanar este problema, ya que se trató de una sola jornada y no hubo nuevos encuentros o auditorías para revisar cómo se estaba efectuando la implementación. No hubo ningún experto que guíe el proceso, o ayude a comprender los errores cometidos.

5.3.2.2. Experiencia y compromiso

No se hizo hincapié tampoco en el compromiso necesario para el éxito de la metodología. Cada quién tomó a su manera la capacitación y su rol posterior. Con lo cual, era de esperarse que este rol no sea otro que el mismo que venía desempeñando antes de la migración.

5.3.2.3. Participación de las personas interesadas

Relacionado a los puntos anteriores, tampoco hubo capacitación hacia las personas interesadas en el producto, sean clientes o personas internas a la empresa. Esto es fundamental ya que son quienes determinan cuáles son las necesidades reales y definen cuándo un incremento está listo para ser incorporado al producto. Sin esa participación, se pierde un circuito fundamental en Scrum.

5.3.2.4. Localización

Otro problema, aunque con menor importancia que los anteriores, es que hay muchos integrantes de equipos en ubicaciones variadas: distintas ciudades, provincias y países. Incluso hay quienes trabaja de forma remota permanentemente. Cómo se remarca siempre en Scrum, la comunicación es muy importante e idealmente debe realizarse de forma presencial, aunque sea una vez cada cierto tiempo.

5.3.3. Procesos

5.3.3.1. Identificación de requerimientos

El principal problema está relacionado con la identificación de requerimientos y se debe a que no hay participación de los clientes en los procesos de Scrum. Ésta se circunscribe solo a una comunicación con el área de Operaciones quien se encarga de plasmar los requerimientos en historias de usuario. Por lo tanto, toda la participación del cliente siempre es indirecta. Si es necesario repasar prioridades, resolver dudas de la definición, o revisar el funcionamiento de algún desarrollo, siempre se hace con personas internas a la empresa, las cuales, en ocasiones, deben trasladar la consulta al cliente.

5.3.3.2. Documentación

El proceso de documentación prácticamente no tuvo modificaciones, continuó con las virtudes y defectos de la metodología anterior. En general, se genera poca documentación técnica y funcional y no hay responsables que deban encargarse de esta tarea.

5.3.3.3. Dependencia entre equipos

Los equipos de desarrollo son heterogéneos en cuanto a conocimientos y capacidades. Además, se encuentran en ubicaciones distintas. Si bien se trata de realizar algunas divisiones de tareas por módulos, todos trabajan sobre el mismo producto y el mismo código, con lo cual esta heterogeneidad suele causar problemas. Cuando el trabajo se concentra sobre un módulo, un equipo queda sobrecargado, con lo cual debe distribuir tareas a los demás (que no son expertos en esa parte del dominio). Por otro lado, como también existen diferencias de conocimientos técnicos, es normal que se requiera mucha comunicación o que se complique unificar criterios.

5.3.3.4. Informes y seguimientos

La tarea de estimación nunca tuvo un proceso muy definido y formal. Incluso, no hubo una estandarización entre los distintos equipos. Al principio se utilizaba el póquer de estimación con todo el equipo de Scrum, luego se tomó solo la opinión de los expertos y actualmente la realiza solo el líder del equipo. En algunos casos, un desarrollador realiza su propia estimación, lo cual claramente no es conveniente.

Tampoco se definieron métricas claras y conocidas por todo el equipo. En general, las mediciones que se hacen son solo por parte de directores y gerentes, quienes las definen y utilizan para sus propósitos. Además, no se hicieron demasiadas customizaciones a la herramienta utilizada para la gestión de Scrum, las cuales deberían ser necesarias para tomar las métricas de forma más fiable.

5.3.3.5. Roles y prácticas del equipo

La elección de líderes de Scrum fue realizada de forma paulatina, intentando encontrar aptitudes de liderazgo antes de formalizar un rol. Sin embargo, eso se realizó solo en el inicio del cambio de metodología. Los equipos crecieron y no se busca (ni surgen) nuevos líderes. De esta forma, tenemos un equipo de más de 15 integrantes, lo cual no es recomendado por la guía de Scrum.

5.3.3.6. Calidad en todo el proceso

Todo lo desarrollado tiene básicamente tres etapas de pruebas: una por el equipo de QA interno, otra por el sector de Operaciones y otra por el cliente. Sin embargo, como se mencionó anteriormente, no hay prácticamente participación del cliente en la reunión de Revisión de los incrementos del producto. Tampoco lo hacen sectores internos de la empresa como podría ser Operaciones. De esta forma, la reunión de Revisión, pierde sentido, y los reportes de errores llegan más tarde lo de debido. Pueden llegar incluso con más de una iteración de demora. Por este motivo, directamente ya no se realiza la reunión de Revisión. El incremento solo tiene el testing interno por el área de QA al incorporarse a una versión.

5.3.4. Herramientas

5.3.4.1. Complejidad de la arquitectura del software

Como se describió anteriormente, el producto es un sistema de gran tamaño con más de 10 años de desarrollo y continúa creciendo. Más allá de esto, la complejidad radica fundamentalmente en la lógica de negocio y la gran cantidad de entidades que maneja. Mantener un código prolijo y estandarizado no es una tarea sencilla por la cantidad de desarrolladores que trabajan, los cuales tienen distintos criterios y conocimientos. La complejidad de la arquitectura hace que además los desarrollos llevan más tiempo de lo ideal, lo cual repercute en una sensación de demora por parte de los clientes o personas externas al sector de desarrollo. Para aplicar Scrum en este contexto es necesario tener personas con mucha experiencia en la metodología, que guíe los procesos en todos los niveles de la organización. La coordinación de los equipos se torna más crítica cuando se trata de proyectos de esta envergadura.

5.3.4.2. Integración del sistema

En general se utilizan diversas librerías, herramientas y *frameworks* para el desarrollo, con muy buenos resultados. Sin embargo, uno de los problemas a nivel técnico más importante fue la mala elección de una librería para el *FrontEnd*, que se realizó casi al comienzo del proyecto de migración. La misma

quedó obsoleta hace mucho tiempo y poder reemplazarla es un trabajo de un costo altísimo. Este trabajo es poco productivo desde el punto de vista externo, lo cual genera mayor disconformismo con la velocidad de desarrollo.

5.3.4.3. Evaluación del proyecto

Las historias de usuario no tienen definidos prácticamente nunca sus criterios de aceptación. La herramienta TFS que se utiliza no tiene por defecto algún campo que incite a los responsables a generarlos. Lo más cercano a esto, fue proponer una suerte de esquema de carga de historias de usuario que los incluya. El resultado fue que se completaba solo por compromiso, sin aportar información adicional a la descrita en la misma historia de usuario. Con este escenario, la carga de criterios de aceptación fue perdiendo valor y dejó de utilizarse.

5.3.4.4. Seguimiento de problemas

Se utiliza sin mayores inconvenientes la herramienta de TFS para el seguimiento de problemas (*Issues*). Sin embargo, es solo visible para las personas que trabajan en la empresa. Por este motivo, la interacción con un cliente se hace a través de otros sistemas (por ejemplo, *Jira)*. Es necesario hacer un mapeo entre ambos sistemas, con lo cual muchas veces se producen problemas del tipo teléfono descompuesto: se pierde información importante, se sintetiza la información, se cambian prioridades, etc.

5.4. Primera implementación de Scrum

En esta sección se describirá cómo fue la primera experiencia de Scrum en la empresa, comenzando por la capacitación inicial y luego por los detalles de la implementación.

5.4.1. Capacitación

Antes de poner en práctica Scrum, se organizó una capacitación para todos los integrantes de la empresa, la cual fue realizada por una empresa especializada. Se trató de una jornada completa en la que se formaron equipos de manera arbitraria. El espíritu era fomentar el trabajo en equipo compuestos por integrantes de toda la empresa que no se conocían o que no solían tener demasiada comunicación.

En líneas generales, la capacitación se dividió en 3 etapas:

- Actividades de integración para ejemplificar distintos valores de Scrum, a través de diversos juegos didácticos.
- Explicación de todos los elementos de Scrum.
- Consultas y conclusiones de la jornada.

Luego de esto, la misma empresa brindó soporte durante los primeros días, pero solo en algunos aspectos relacionados al área de desarrollo. No se realizaron más capacitaciones grupales, ni charlas de ningún tipo.

5.4.2. Características generales de la primera implementación

Se crearon equipos de Scrum internos al área de Desarrollo. La estructura del área tenía anteriormente un Director y un Gerente con un equipo de programadores por cada oficina. Se decidió entonces mantener la misma estructura, pero eligiendo un *Scrum Master* dentro de los programadores más experimentados. Los equipos tenían un máximo de 8 integrantes cada uno.

Los *Product Owners* eran generalmente integrantes del sector de Operaciones (cuando se trata de un pedido de un cliente) o del área de Producto (cuando se trata de un cambio del producto decidido por el área).

Los *Sprints* tenían una duración de 2 semanas, en el que al cierre de cada uno se generaba una versión interna. Cada 3 *Sprints* se hacía el cierre de una versión *release*. Es decir que se tenían por ejemplo las versiones 5.01.01, 5.01.02 y 5.01.03 como cierres *beta* para cada *Sprint*, y al finalizar el último se obtenía la versión 5.02.00 como cierre *release*, el cual podía llegar a instalarse en algún cliente.

5.4.3. Elementos de Scrum aplicados

Al ser el comienzo de la experiencia, se intentó cumplir con casi todos los eventos y artefactos propuestos por la metodología, aunque con algunas diferencias con respecto a la teoría. A continuación, se detallan las características generales de cada uno al comienzo de la implementación de Scrum:

- *Sprints* de 2 semanas de duración.
- Equipos de Scrum compuestos solamente por desarrolladores y un Scrum Master para cada uno.
- Reuniones de Planificación al comienzo del *Sprint*: Actualmente no se realiza en todos los *Sprints*, por motivos que se describirán más adelante. Sin embargo, al comienzo se realizaba sin mayores inconvenientes. Todo el equipo de Scrum se reunía para analizar las historias de usuario, estimarlas, y asignar las tareas para cada desarrollador.
- Scrum Diario: Esta reunión se mantiene de la misma forma actualmente. Todos los días, a una hora prefijada (siempre la misma) todo el equipo de Scrum se reúne para comentar con qué tarea trabajó el día anterior y qué tiene pensado realizar hoy. Además, puede comentar cualquier problema o consideración

que atañe a las tareas del equipo. Cada orador tiene un tiempo máximo de 2 minutos.

- Scrum de Scrum: Si bien la reunión continúa realizándose, actualmente tiene otras características que se describirán más adelante. Un integrante designado de cada equipo de Scrum se reunía 2 veces por semana con los demás equipos del sector. En el sector de desarrollo había 3 equipos de Scrum por lo que se juntaban 3 personas a comentar sucintamente las tareas que estaba realizando cada equipo y los inconvenientes que fueron surgiendo.
- Reunión de Revisión: Esta reunión dejó de realizarse desde hace tiempo. Al finalizar el *Sprint*, un integrante del sector de Operaciones, haciendo las veces de "Product Owner", junto con el *Scrum Master*, repasaban todas las historias de usuario desarrolladas en la iteración.
- Reunión de Retrospectiva: Esta reunión se realiza actualmente, aunque con baja frecuencia. Cada equipo de Scrum se reúne al final del *Sprint* para revisar la iteración que pasó. Los temas son libres y pueden incluir cuestiones técnicas, organizativas, de infraestructura, etc. La duración de la reunión es de 2 horas máximo. Se comienza revisando la retrospectiva anterior y luego proponiendo los nuevos temas.
- Definición de la Lista de Producto: El área de Producto comenzó a generar historias de usuario basándose en requerimientos del *core* del Producto. A eso se le sumaron las historias de usuario propias del primer proyecto.
- Definición de la Lista de *Sprint*: Luego de cada reunión de Planificación quedaba definida la lista de *Sprint*.

Como diferencia con lo definido por la Guía de Scrum [6], podemos ver que no se aclaraba un Objetivo del *Sprint*. La reunión de planificación derivaba en un conjunto de historias de usuario a definir, las cuales dependían solamente de la capacidad del equipo de desarrollo. Si algún elemento no llegaba a completarse se continuaba en el siguiente *Sprint*. Por otra parte, se definió realizar una reunión de integración de los equipos de Scrum que, si bien no se encuentra definida por la guía de esta forma, sí es una práctica común [12].

5.5. Herramienta Team Foundation Server

A continuación, se describirá la herramienta *Team Foundation Server* (TFS) utilizada en la empresa como soporte a la metodología de trabajo desde el mismo momento en que se decidió migrar a Scrum. TFS tiene varias funcionalidades: repositorio de código fuente, gestión de proyectos, gestión de requerimientos, integración continua y testing. Los beneficios fundamentales

por las cuales se utiliza esta herramienta son la gestión de Scrum y la trazabilidad del producto.

5.5.1. Gestión de Scrum

La implementación de SCRUM se hizo a través de una única herramienta utilizada por todos los sectores de la empresa: Team Foundation Server (TFS). De esta forma se logra tener centralizados todas las historias de usuario, Issues y Bugs. Cada usuario tiene distintos permisos para poder consultar o crear estos elementos.

A continuación, se muestra el *Home Page* que brinda TFS:

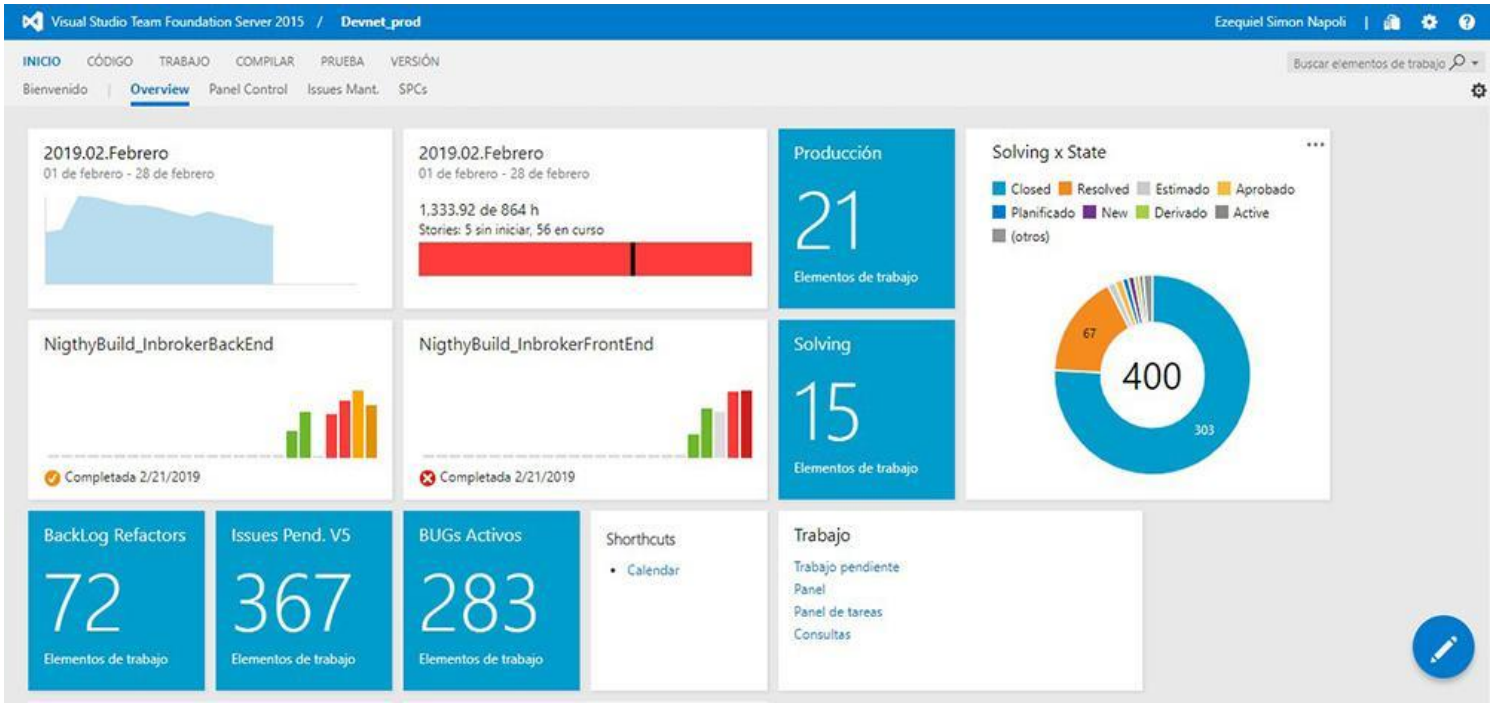

***Figura 5.1**. Home Page de Team Foundation Server.*

Este *Home Page* es totalmente configurable, por lo que varía a medida que se requiere nueva información o que muestre información obsoleta. En los dos cuadros superiores, se puede ver información del *Sprint* actual. A la izquierda el gráfico de *Burndown* del *Sprint* actual, y a la derecha una barra que indica la cantidad de horas remanentes del *Sprint* contra la cantidad de horas pendientes de desarrollo (en este ejemplo, las horas pendientes superan ampliamente a la capacidad restante). En los dos cuadros centrales, se puede ver información sobre la compilación automatizada del código (integración continua). Por debajo hay 3 contadores: historias de usuario de *refactor*, *Issues* pendientes, y *BUGs* activos. A la derecha de estos recuadros hay información relacionada a los *issues* más críticos, que son aquellos que están en producción.

En la siguiente imagen se puede ver con más detalle el gráfico de *Burndown*:

***Figura 5.2**. Ejemplo de Burndown del Sprint.*

El eje Y representa las horas de desarrollo estimadas para las tareas de la iteración actual, mientras que el eje X muestra las fechas que comprenden dicha iteración. El gráfico en azul muestra como fue el progreso durante el *Sprint*. Con una línea verde se indica la capacidad disponible del equipo, mientras que con una línea negra se muestra cómo debería ser el progreso "ideal". Es decir, cómo se deberían ir consumiendo todas las horas asignadas. Cuando el gráfico supera esta línea, podemos deducir que hay más trabajo pendiente del que puede realizarse. Si se encuentra por debajo, tenemos que hay más capacidad disponible como para agregar elementos al *Sprint*. Claro está, dicho ideal pocas veces se cumple (o nunca), pero sirve para tener una idea del estado de situación del progreso de las tareas asignadas al comienzo del *Sprint*.

Al aplicar Scrum es muy frecuente el uso de un panel o pizarrón en el que todo el equipo puede ver el estado de las tareas del *Sprint* actual. En muchos casos se siguen utilizando pizarrones reales con *post-it's*. La Web de *Team Foundation Server* provee un panel como el siguiente:

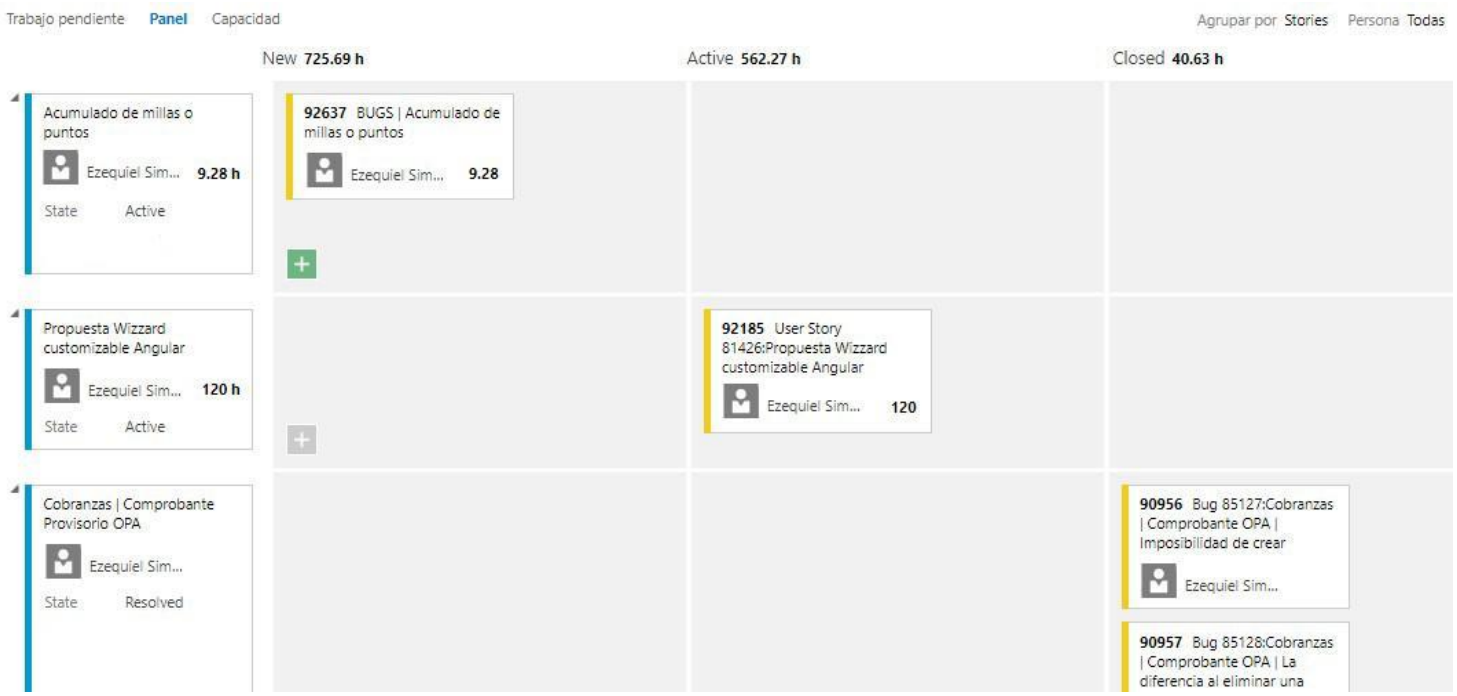

***Figura 5.3**. Ejemplo del Panel de Tareas.*

En cada columna se muestra el estado, mientras que en cada fila se ve el título de la historia de usuario. En la intersección de ambas se encuentra el "*Post-It*" con el título de la tarea, el tiempo restante y la persona encargada de llevarla a cabo. Es importante destacar que las columnas son configurables, por lo que además de las 3 que se pueden ver (Nueva, Activa, Cerrada) podrían agregarse otra como por ejemplo las correspondientes al Testing.

Por otro lado, la herramienta también permite ver la "capacidad" de cada persona para el *Sprint* actual. Esto es, cuánto tiempo disponible tiene la persona para realizar tareas y cuánto trabajo tiene asignado.

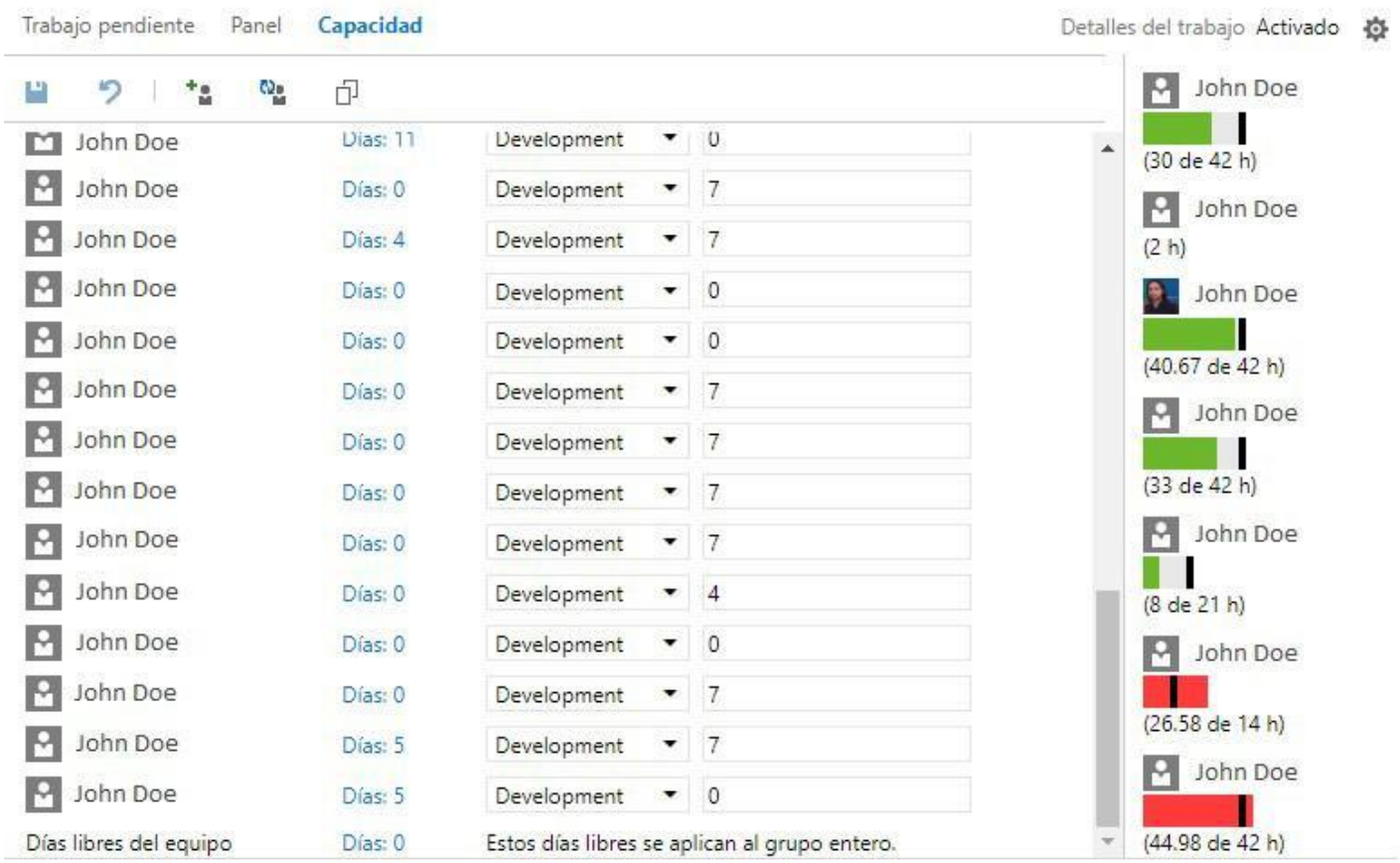

***Figura 5.4**. Capacidad disponible por persona.*

Se puede ver a la izquierda la configuración de la capacidad disponible para cada persona. Se pueden indicar los días libres previstos por licencias y los días libres del equipo, por ejemplo, por feriados. Luego se puede indicar la cantidad de horas disponibles por día, teniendo en cuenta el factor de dedicación. Como la jornada laboral tiene 8 horas, se considera lo normal restar 1 hora debido a las reuniones de Scrum, el tiempo para descanso, ir al baño, tomar un café, etc. A la derecha, se puede ver el estado actual de cada persona en cuanto a su tiempo disponible contra el trabajo pendiente asignado.

En cuanto a la Lista de Producto, *Team Foundation Server* permite definir elementos y consultarlos en todo momento. Para una mejor búsqueda se provee un motor de consultas (que también sirve para buscar historias de usuario, tareas, bugs, *issues*, o cualquier otro elemento). A continuación, se muestra un listado de historias de usuario, con el estado actual, la iteración asignada (si la tiene) y *Tags* para facilitar su categorización.

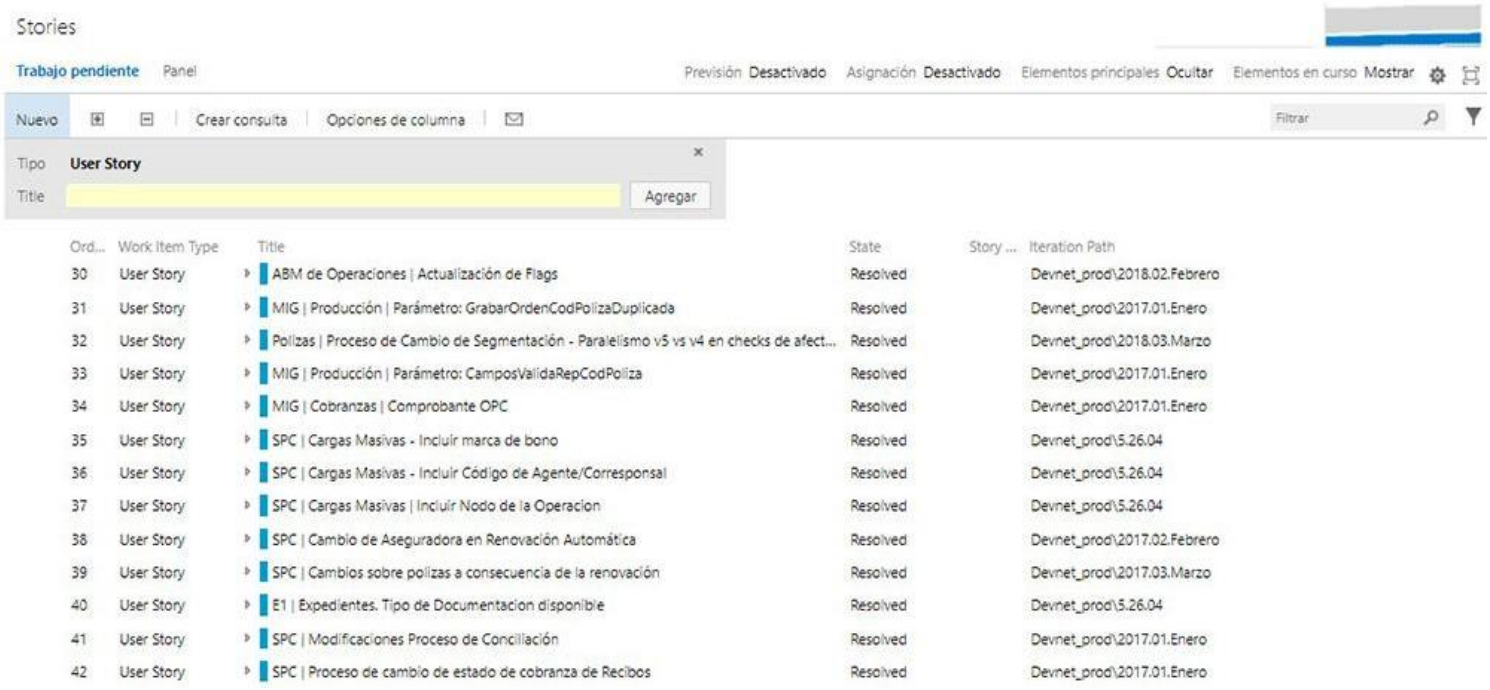

***Figura 5.5**. Lista de Producto.*

También se provee un gráfico para mostrar el estado y el progreso de la Lista de Producto:

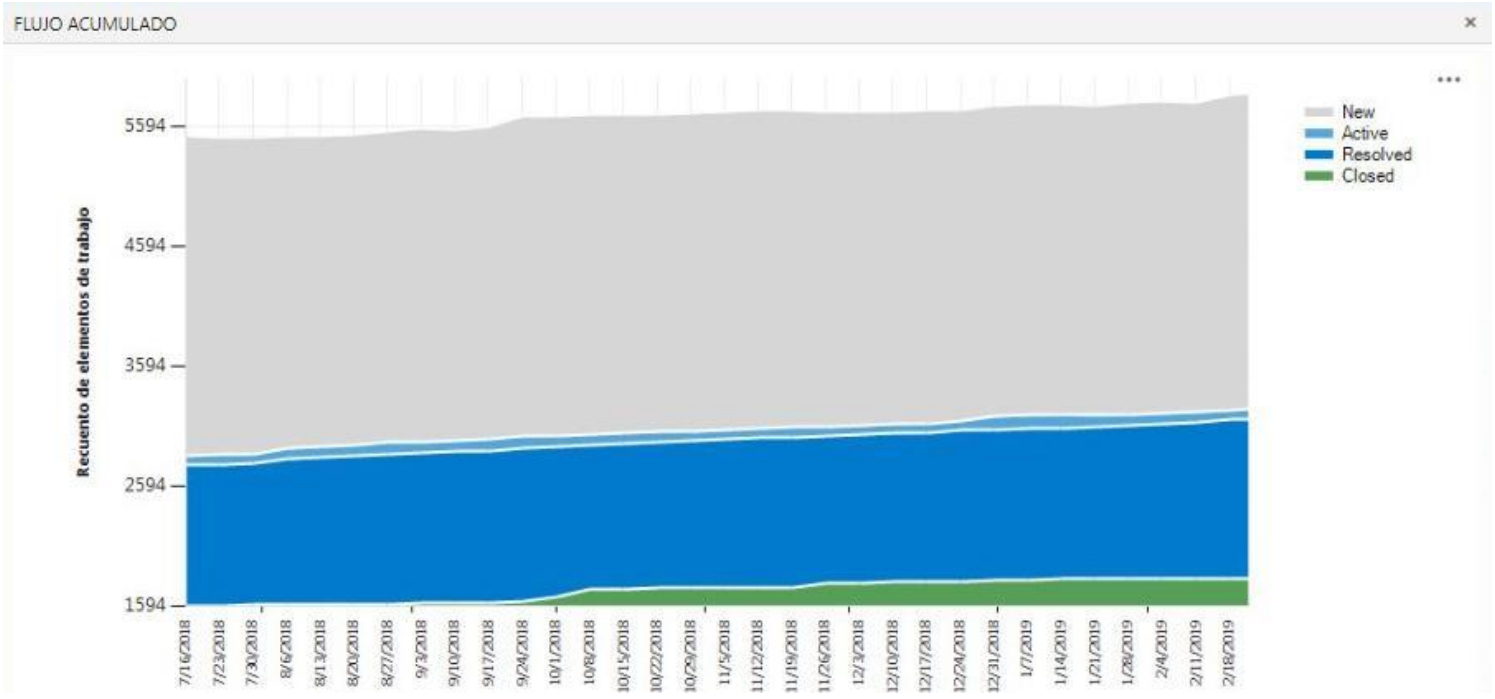

***Figura 5.6**. Gráfico de progreso de la Lista de Producto.*

En el eje Y se muestran la cantidad de elementos, mientras que en el X se indican las fechas. Con colores se representan los distintos estados de las

historias de usuario: Nuevas en gris, Activas en celeste, Resueltas en azul y Cerradas en verde.

La carga de historias de usuario puede o no estar relacionada a algún proyecto. Dependiendo del sector, tenemos los siguientes grupos de historias de usuario:

- “Producto”: para funcionalidades propias del sistema, generalmente relacionadas a un requerimiento de la versión de escritorio, o visuales, como por ejemplo mejoras en la interfaz de usuario.
- “Operaciones”: para funcionalidades que provienen de pedidos del cliente.
- “Desarrollo”: para revisiones de código, *refactors*, optimizaciones, etc.

A continuación, se muestra la pantalla de carga para una nueva historia de usuario:

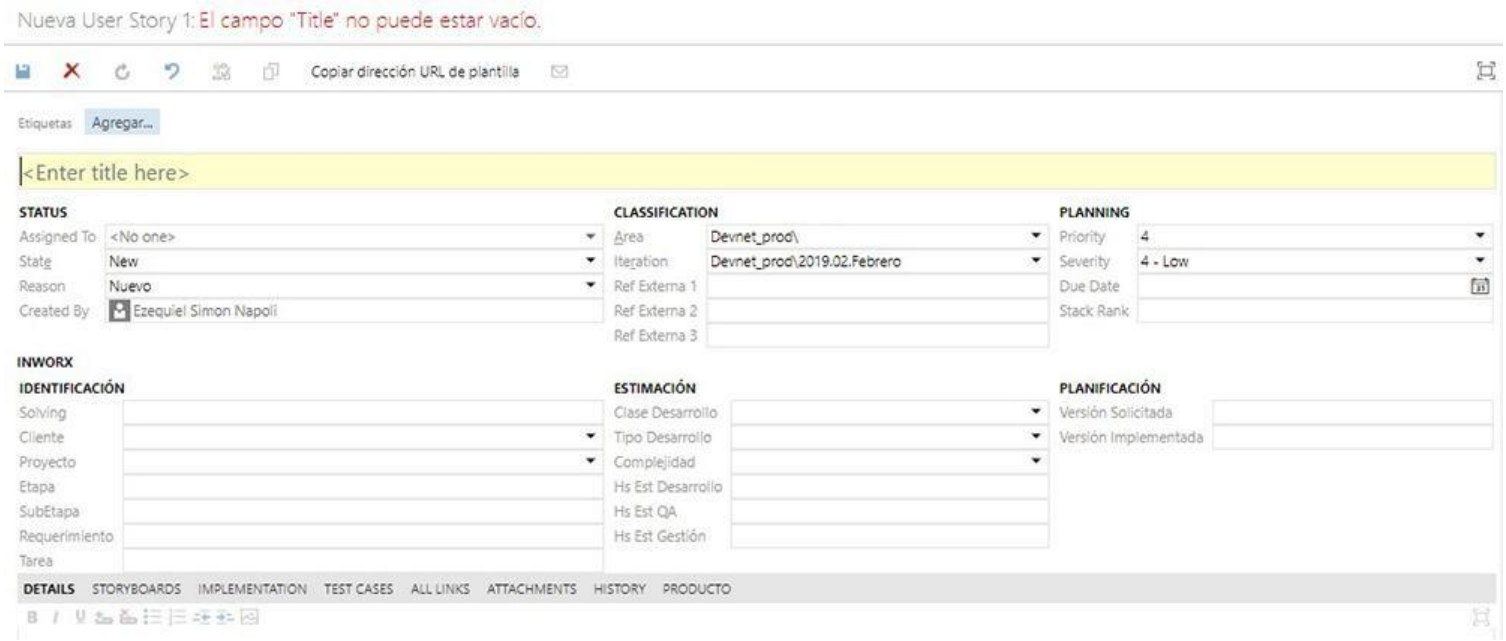

***Figura 5.7**. Formulario actual para la carga de una Historia de Usuario.*

Como se puede ver, hay una gran cantidad de datos relacionados a la historia de usuario, los cuales se fueron agregando a medida que evolucionaba la implementación de Scrum. Básicamente, tiene un título, una persona asignada (no se utiliza para el Dueño de Producto sino para el *Team Leader*), un Estado, un Motivo, una Iteración (que puede ser el *Backlog* de Producto o un *Sprint* concreto), una prioridad, una criticidad y una fecha límite. Luego hay muchos campos relacionados con el proyecto al que pertenece e información de la versión en la que se incluye. Por debajo, se encuentra la descripción de la historia de usuario, el cual es un campo de texto libre.

Las historias de usuario se resuelven mediante Tareas, las cuales se crean generalmente durante la reunión de Planificación. La pantalla de carga de una tarea muestra lo siguiente:

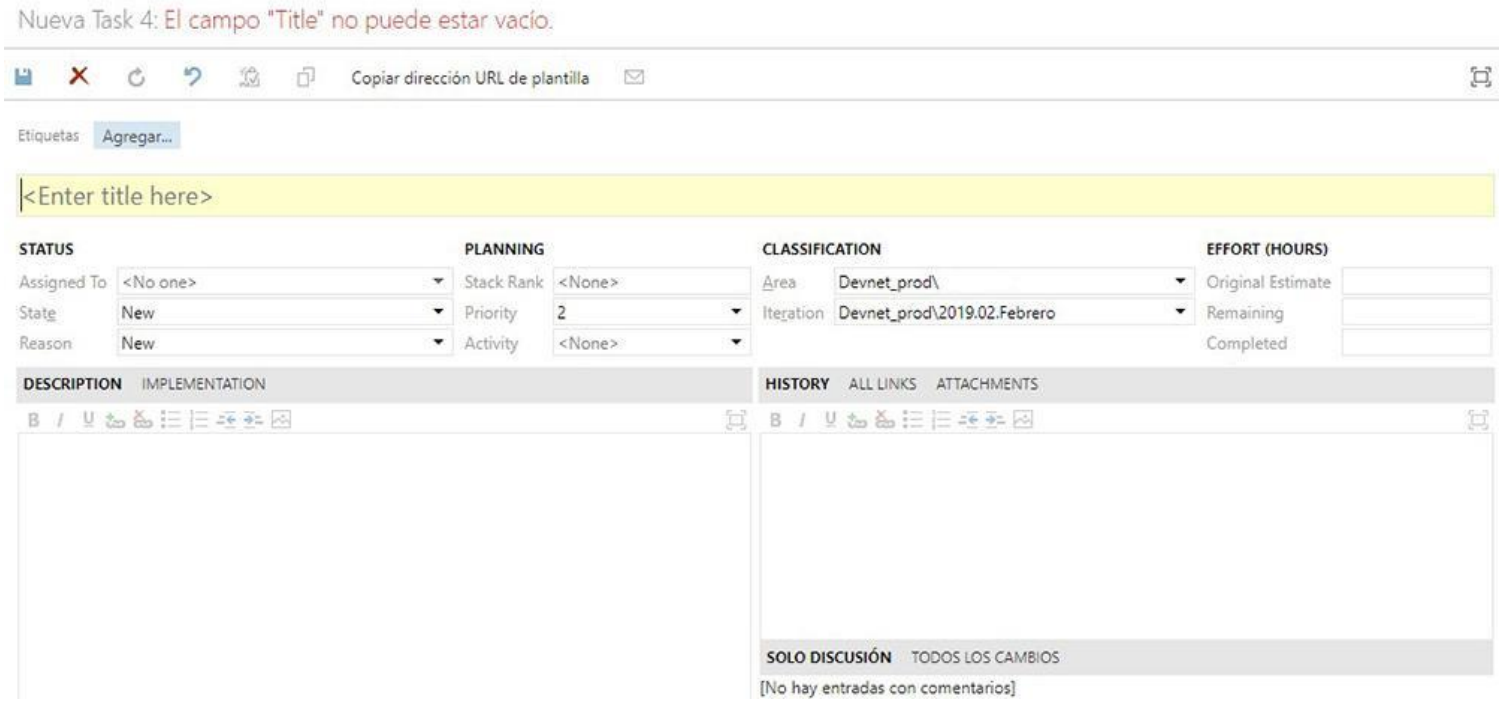

***Figura 5.8**. Formulario de carga para una tarea.*

Una Tarea tiene un título, una persona asignada (la responsable de resolverla), un Estado, un Motivo, una Prioridad, un Tipo, la iteración en la que se va a realizar, un tiempo estimado, un tiempo restante y un tiempo completado. La información del tiempo es actualizada solo por la persona que tiene la tarea asignada. Luego tiene un campo de texto libre en el que se describe el objetivo de la Tarea.

5.5.2. Trazabilidad del Producto

Team Foundation Server es una herramienta que permite obtener trazabilidad del producto, tanto por el registro de todos los requerimientos que lo componen, como por el seguimiento del código fuente.

Cada historia de usuario tiene relacionada todas la Tareas necesarias para cumplimentarla. A su vez, en las tareas se pueden asociar los cambios realizados en el código fuente (si los hubiera). Lo mismo sucede con los elementos de tipo *Issue* (utilizado en la empresa para registrar los reportes de los clientes) como por los *Bugs* (informados por el área de Testing). De esta forma, es posible buscar cualquier elemento de TFS y reconocer fácilmente cuáles fueron los cambios realizados. Para la búsqueda, TFS brinda la posibilidad de crear consultas. Por ejemplo, si quisiéramos obtener todas las

historias de usuario resueltas de una determinada iteración tendríamos que crear la siguiente consulta:

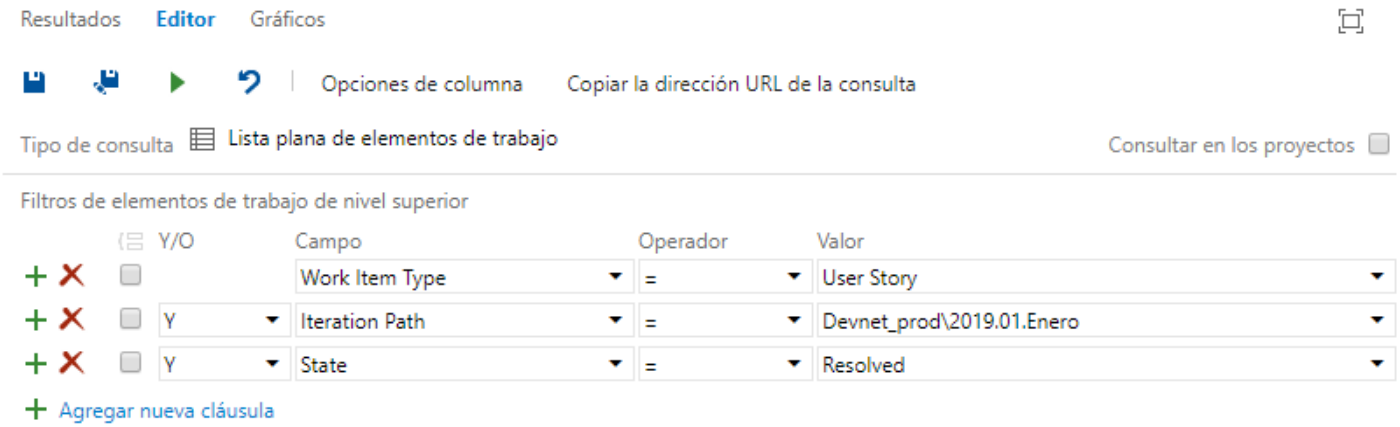

***Figura 5.9**. Ejemplo de Consulta de TFS*

5.6. Evolución de la implementación de Scrum

La implementación realizada al comienzo sufrió diversos cambios que fueron adaptando algunos elementos de Scrum a la cultura de la empresa o a distintas necesidades que fueron surgiendo. En primer lugar, los *Sprints* modificaron la duración de 2 semanas a 3, ya que los cierres internos tenían poco sentido y consumían un trabajo innecesario (el costo de realizar un cierre de versión era en ese momento para nada despreciable). Luego de un tiempo, se extendió el *Sprint* a un mes calendario, el cual se mantiene actualmente. En primer lugar, por una cuestión de orden y facilidad de comprensión de toda la empresa. Cualquier área puede conocer cuándo comienza y cuándo termina un *Sprint*, sin necesidad de consultar a una persona o sistema. En segundo lugar, para dar tiempo suficiente a un test interno previo al cierre de la versión. Los elementos que se van finalizando durante el *Sprint* puede tener la verificación del área de QA.

En cuanto a las versiones, actualmente se liberan dos tipos:

- Versión nueva: cierre que se realiza al finalizar el *Sprint* con todos los desarrollos nuevos incluidos. Esta versión se considera como *Beta*. Luego de 15 días de regresión por parte del área de QA se realiza el cierre *Release*. Es decir que la versión recién está disponible pasados 15 días de la iteración siguiente.
- Branch de Versión: por necesidad del cliente, se genera un branch de versión que contiene correcciones de *Issues* o SPCs. Estas versiones no tienen ninguna relación con el *Sprint*, es decir que pueden suceder en cualquier momento. Se intenta que todo SPC se incluya dentro de la reunión de planificación, aunque en la práctica muchas veces es difícil de cumplir, ya sea por presiones externas o compromisos pre acordados.

Uno de los cambios más importantes tiene que ver con el concepto de *Scrum Master*. Al principio, se habían definido 3 equipos de Scrum, divididos en dos oficinas distintas, los cuales tenían cada uno su *Scrum Master*. Luego, se cambió el concepto por el de *Team Leader* (el cual se utilizaba en la metodología anterior). Actualmente existen 3 oficinas, que tiene cada una un *Team Leader*, el cual hace las veces de *Scrum Master*. En una de las oficinas trabajan más de 15 desarrolladores, por lo que se subdivide en dos equipos de Scrum, aunque sólo a efectos de la reunión Diaria y la reunión de Planificación. El motivo es meramente temporal: las reuniones con tantos participantes se hacían extremadamente largas.

Por último, la reunión de Scrum de Scrum cambió completamente su fisonomía. En primer lugar, ahora incluye a los *Team Leader* de los equipos de desarrollo, a Directores de distintas áreas y Gerentes. Primero se realizaba 3 veces por semana, luego 2 y actualmente solo 1. El motivo de la disminución de la frecuencia es que, al haber tantos actores, la reunión consume demasiado tiempo. En teoría, era una reunión pensada para 45 minutos, aunque muchas veces demora más de una hora. Por otra parte, las variadas características de las personas que la componen (y los distintos intereses) hace que muchas veces se traten temas que no interesen a la mayoría. Para poder suplir la reunión de Scrum de Scrum con las características anteriores, se creó una Diaria de Líderes. Todos los días a la misma hora se reúnen los líderes de cada oficina para coordinar el trabajo, reasignar tareas y revisar el avance de la planificación.

En TFS también se fueron generando cambios para adaptarse a distintas necesidades, o para eliminar campos que dejaron de utilizarse. Por ejemplo, en cuanto a las historias de usuario se eliminaron los *Story Points* y el *Stack Rank* (utilizado para la priorización o “nivel de importancia”). En la figura 5.7 podíamos ver el formulario actual de la carga de una historia de usuario, la cual difiere mucho de la primera versión:

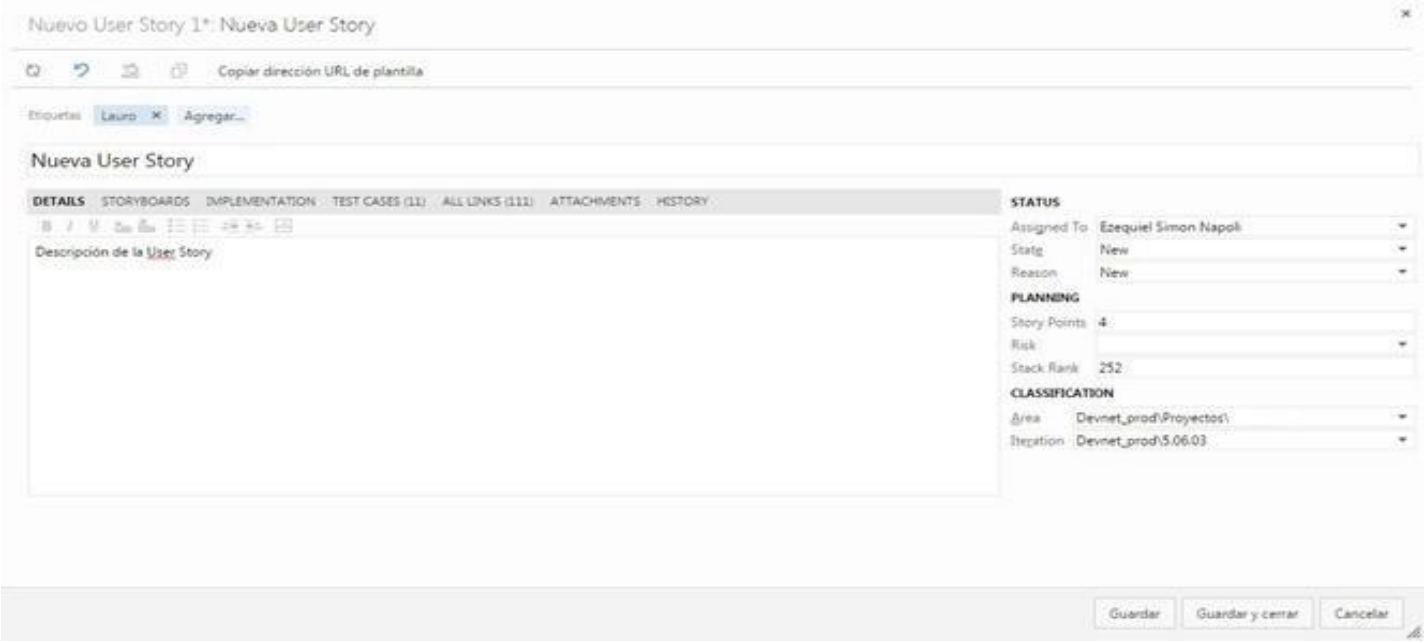

***Figura 5.10**. Primer formulario de carga para una Historia de Usuario.*

Como se puede ver, más allá de los campos mencionados que se eliminaron, actualmente hay mucha información nueva. La mayoría de estos datos tienen que ver con información de proyecto, la cual es necesaria no solo para seguimiento sino también para la facturación de las horas imputadas a cada historia de usuario.

5.7. Diferencias con el Scrum teórico

En esta sección se describirán todos los elementos de Scrum en los que se detectan diferencias con el propuesto por la Guía de Scrum [6] o con los consejos dados por diversas implementaciones de Scrum.

Como bien menciona Kniber [12], la reunión de planificación es el corazón de Scrum. Cumplir con todos los principios de esta reunión es la clave para poder implementar la metodología de forma exitosa. En el caso de estudio podemos encontrar algunas diferencias fundamentales que minimizan el sentido de la reunión. En primer lugar, la planificación suele variar durante una iteración debido a cambios de estrategia de la empresa, ya sea por decisiones de la Dirección, por prioridades establecidas por un cliente, por compromisos ajenos al área de desarrollo, SPCs prioritarios comunicados de manera informal, etc. Por otra parte, no se suele fijar un objetivo para un *Sprint*, lo cual es coincidente con la posibilidad de modificar la planificación. Los objetivos suelen ser del estilo “cumplir lo mayor posible en la mayor cantidad de proyectos”, lo cual no es un objetivo concreto. Otra diferencia fundamental es que nunca se incorporó un dueño de Producto a las reuniones de planificación. Las mismas siempre son definidas íntegramente por el área de desarrollo, tomando todas las decisiones de forma autónoma e intentando resolver las

dudas que puedan surgir a partir de la colaboración de los más experimentados (quienes ofician de alguna manera de *Product Owner*). Además, por las urgencias de tiempo, muchas veces la reunión de planificación no se realiza con todo el equipo. En esos casos, es tarea del *Team Leader/Scrum Master* analizar, estimar y asignar tareas para un subconjunto de historias de usuario de la lista de producto. La decisión de cuáles son esas historias de usuario es propia del *Team Leader*. Para esta tarea se suelen definir cotas de horas por proyecto, aunque no siempre es una información clara y concreta. Tampoco es habitual contar con una priorización (o nivel de importancia) de las historias de usuario. Al comienzo de la implementación de Scrum se utilizó, pero luego fue perdiendo valor al no contar normalmente con personas con el conocimiento o la disponibilidad suficiente como para gestionar la lista.

Cómo se comentó previamente, el concepto del *Scrum Master* se delegó a manos de un *Team Leader*, el cual en algún caso tiene a su cargo hasta 15 desarrolladores. Como no es práctico para las reuniones, este equipo se divide en dos, aunque con el mismo *Scrum Master*. El problema de esto es que el *Team Leader* no solo se enfoca en la gestión de Scrum, sino que también debe realizar diversas tareas: soporte técnico a todo su equipo, soporte funcional a otras áreas de la empresa, seguimiento del avance de un proyecto, colaboración en la definición y estimación de historias de usuario e *issues*, evaluación del rendimiento individual de cada uno de los desarrolladores a cargo y reporte a directores y gerentes.

Un problema común es que pueden existir historias de usuario cuya estimación supere a un *Sprint*. La complejidad de este tipo de elementos hace que no sea posible subdividirlo. No es aconsejable tampoco realizar entregas parciales, ya que todo *Sprint* debe derivar en un incremento que potencialmente pueda ser utilizado. Por lo tanto, en estos casos, se decide crear una tarea por cada *Sprint* necesario, sabiendo que el resultado de la primera no podrá incorporarse a la versión. A excepción de este caso, en las planificaciones no se suelen dividir las historias de usuario en N tareas. Normalmente es una sola tarea que la resuelve por completo. El motivo de esto es que muchas veces no contamos con suficiente definición en la historia de usuario como para determinar en poco tiempo cuestiones técnicas. Esta es una tarea que termina definiendo el programador al comienzo de su desarrollo.

La estimación de cada historia de usuario en general también es realizada por el *Team Leader/Scrum Master*. En un principio se utilizaba el conocido póquer de estimación, el cual daba resultados diversos. Con el tiempo, se fue abandonando esta práctica, para reemplazarla por una estimación realizada por los más experimentados. Esta tarea debe realizarse antes de la planificación, para poder determinar cuánto trabajo podrá resolverse en el *Sprint*. La forma de estimación fue siempre en horas. No se utilizan *Story Points*, ni otras formas de medición (como por ejemplo días-hombre, mencionada por Kniberg [12]).

Para realizar la planificación, se asigna una capacidad de desarrollo en horas a cada integrante del equipo, teniendo en cuenta feriados y licencias. Se asignan las tareas con su tiempo estimado, y se completa aproximadamente un 80% de la capacidad de cada uno. El restante se toma como reserva para la corrección de errores. Esto es similar a lo comentado por Kniberg [12], aunque la reserva que allí se define es mucho mayor (50%). Lo cual se corresponde con un problema recurrente que tenemos: la cantidad de tiempo que insume la corrección de errores es superior al tiempo reservado y, por lo tanto, se compromete la culminación de las tareas planificadas en el *Sprint*.

Otras reuniones de Scrum fundamentales también se fueron dejando de lado con el tiempo. La reunión de revisión, por ejemplo, se realizó al comienzo de la implementación, pero fue dejada de lado rápidamente. El motivo es que era difícil tener a los dueños de producto u otras personas interesadas disponibles. Por otra parte, la retrospectiva de *Sprint*, si bien se realiza de forma intermitente, casi se ha dejado de lado actualmente. La razón de esto es que se prioriza dedicar el tiempo de la reunión a cuestiones de desarrollo. En las retrospectivas, además, se ha revisado poco la metodología de trabajo, lo cual es fundamental para la retroalimentación aconsejada en Scrum.

Otra diferencia importante con lo aconsejado normalmente en Scrum es que no se definen equipos de Scrum por proyecto, lo cual genera, además, una contradicción con la idea de generar objetivos de *Sprint*. Existen N equipos de Scrum que pueden trabajar en N proyectos en un *Sprint*. El beneficio que brinda esta situación es la de poder reasignar recursos dinámicamente, una necesidad creada por los habituales cambios de prioridades a nivel de gerencia de proyectos.

Hay otras diferencias respecto de los consejos dados por Kniberg [12], aunque de menor importancia tal vez que las anteriores. Por ejemplo, no se tienen tarjetas físicas, ni tableros reales para la gestión de Scrum, todo se realiza a través del sistema web de TFS. Las planificaciones se realizan con proyector y con todos los integrantes viendo la carga de tareas realizada por el *Scrum Master*. Tampoco se juntaron los puestos de trabajo de los integrantes de un mismo equipo. Cada uno mantiene siempre su puesto, y los equipos pueden variar en cada *Sprint*. Nunca se realizan “descansos” entre *Sprints*. Siempre comienza uno a continuación del otro, y el día puede coincidir con cualquier día de la semana ya que los *Sprints* se realizan por mes calendario. Por último, contamos con historias de usuario de tipo técnicas. Un problema mencionado ampliamente es que, si tienen la misma forma y tratamiento que cualquier historia de usuario, entonces no suelen priorizarse. Esto sucede exactamente de esta forma, ya que en las planificaciones en general no son tenidas en cuenta.

5.8. ¿Qué se hizo bien?

Más allá de todas las dificultades y errores cometidos, seguramente podemos rescatar algunos puntos positivos. En primer lugar, se definió una Lista de Producto, que, como se mencionó, era un punto de partida fundamental. Para poder ordenar la migración del sistema, debía registrarse de alguna manera todas las funcionalidades que tenía la versión anterior.

Otro aspecto que se puede destacar, es que Scrum tiene como un valor fundamental al empirismo. Más allá de la metodología a utilizar la experiencia es un valor fundamental en la empresa. El producto tiene un tamaño y una complejidad que no se encuentra documentada al máximo detalle. Por esto, las personas que más lo conocen tienen siempre mayor importancia y poder de decisión.

Siempre se hace hincapié en que Scrum es un Marco de Trabajo y no una metodología en sí. Hay muchos aspectos que dependen del contexto y que deben optimizarse basándose en la retroalimentación. Un ejemplo de esto puede verse en cómo fue evolucionando la duración de los *Sprints* hasta encontrar un balance. Las iteraciones no son tan cortas como para no poder resolver un Incremento considerable ni muy largas como para tener que esperar demasiado por una nueva versión y perder agilidad. Otro ejemplo de esta retroalimentación positiva es el manejo de horarios de las reuniones. Al principio era bastante costoso mantener los horarios predefinidos. Por ejemplo, la reunión de planificación difícilmente se concluía en un solo día. Actualmente, y a partir de distintas lecciones aprendidas, las reuniones pueden realizarse en el tiempo establecido.

Un aspecto que menciona Kniberg [12], es que los *Sprints* de los distintos equipos deben estar sincronizados. Incluso se menciona que el propio Ken Schwaber lo recomendó. Esto siempre se hizo de esta manera.

Otro elemento que se implementó de forma satisfactoria es la manera en que se calcula la velocidad de desarrollo. Si bien no está medido en *Story Points* sino en horas, el cálculo funciona bien para poder determinar cuánto trabajo es posible asignar en un *Sprint*. Además, se tiene en cuenta el factor de dedicación, para no sobrecargar la capacidad de cada desarrollador.

Capítulo 6

Conclusiones

6.1. Sobre el proceso de migración de una metodología clásica a una ágil

Claramente la mayor dificultad fue romper con la cultura de trabajo de todos los sectores de la organización y de sus clientes. Luego de muchos años utilizando la misma metodología y herramientas, se hizo muy difícil realizar un cambio radical. El único sector que pudo implementar diversos cambios en un primer momento fue el de Desarrollo. Sin embargo, al no tener la correspondencia en toda la organización, se fueron perdiendo elementos valiosos de la metodología. Hasta el momento, resulta muy difícil adaptar el esquema de trabajo en el resto de la empresa, ya sea por inercia, falta de capacitación o por la necesidad de adaptar cuestiones comerciales. La cultura de los clientes también influye en la metodología. Como se mencionó en este informe, se trata de grandes compañías que en muchos casos llevan varios años envueltos en esta relación comercial. En general requieren contratos y presupuestos acordados al inicio de un proyecto que puede llevar años, lo cual afecta a la agilidad necesaria en la nueva metodología. En este contexto, es muy difícil que un cambio de metodología resulte realmente exitoso.

El cambio se intentó realizar quizás sin demasiado análisis del contexto. El desarrollo de un producto tan grande y complejo requiere de mucha experiencia en la metodología como para implementarla de la mejor manera. Una opción mucho menos riesgosa tal vez hubiese sido realizar una suerte de prueba piloto en algún producto más pequeño, de equipos más reducidos y proyectos de corto plazo. Lejos de esto, el cambio intentó realizarse radicalmente, con escaso tiempo e involucrando a todos los proyectos de la empresa en simultáneo.

La retroalimentación es un valor fundamental en las metodologías ágiles y claramente en el caso de estudio no se realiza para ajustar la metodología de manera adecuada. Por el contrario, pareciera ser que la mayoría de las adaptaciones realizadas tienen como objetivo combinar Scrum con la metodología de trabajo previa.

La resistencia al cambio también es uno de los problemas que más afectaron a la migración, ya sea a nivel grupal o individual. En la empresa existen personas con muchos años de antigüedad y habitualmente resulta difícil modificar sus roles o responsabilidades. Lo mismo sucede a nivel sectores, ya que la forma de trabajo que “funciona” es la que venían realizando anteriormente.

Si bien todos participaron de la capacitación inicial, claramente no fue suficiente. Hubiese sido deseable tener una persona o grupo de personas expertas en esta clase de migraciones que ayuden a guiar todo el proceso, involucrando a todas las personas y a todos los sectores por igual.

Creo que la decisión de migrar hacia una metodología ágil no fue desacertada. Las empresas deben acompañar la evolución tecnológica y las necesidades de los clientes aprovechando las virtudes de metodologías probadas. El modelo en cascada anterior tenía varios defectos y cuellos de botella que podían convertirse en problemas, sobre todo al comienzo de la migración del sistema. Una metodología ágil podía favorecer el inicio de una migración muy compleja y de mucho tiempo de desarrollo. Sin embargo, no todo es aplicable a cualquier contexto. O, mejor dicho, en algunos casos es posible aplicarlo de forma más directa y en otras se requiere mucho mayor análisis. Luego se deben planificar todos los cambios necesarios, capacitar a toda la organización, plantear objetivos propios de la metodología y una vez implementada, adaptar lo que sea necesario para conseguir los objetivos planteados.

6.2. Sobre la implementación de Scrum

El resultado de la implementación de Scrum tiene claramente diversas dificultades. En primer lugar, muchos de los elementos que no se realizan como se deben, hacen al funcionamiento y el espíritu de Scrum. Por ejemplo, la reunión de planificación es clave para poder tener orden, agilidad y transparencia. Sin embargo, no solo no se realiza con todos los actores necesarios, sino que habitualmente no llega a realizarse. Además, una vez definido una lista de pendientes del *Sprint*, difícilmente sea respetado. Las presiones por el cumplimiento de historias de usuario fuera de plan o la resolución de una gran cantidad de errores críticos en el corto plazo, hace que muchos elementos originales de la planificación no puedan completarse. La retroalimentación también es fundamental en Scrum y, sin realizar la reunión de retrospectiva, se pierde una buena oportunidad para tenerla. Por último, tampoco se realiza la reunión de revisión, que permite evaluar los avances realizados.

Como se mencionó anteriormente, aún no se logró implementar Scrum en el resto de los sectores de la empresa, lo cual complica aún más el poder respetar los pocos elementos de Scrum que sí se implementaron. Muchas veces se requiere la participación en reuniones de personas ajenas a desarrollo, lo cual desde el inicio fue una tarea casi imposible de cumplir. Esto deviene en que dichas reuniones pierdan sentido.

Para el sector de desarrollo, la interacción con los clientes es siempre de forma indirecta, con lo cual se pierde muchas veces información de primera mano o se complica la resolución de dudas que puedan surgir antes o durante un desarrollo.

Si bien se aclara que Scrum es un marco de trabajo y no una metodología en sí, es necesario cumplir con todos los artefactos y eventos que se definen. El detalle de la implementación de cada uno puede adaptarse al contexto, pero bajo ningún punto de vista pueden eliminarse o modificar su sentido. En la Guía de Scrum [6] hay una nota final que indica que Scrum solo puede implementarse como un todo. Si se implementa sólo una parte, entonces no es Scrum. Por lo expuesto en este trabajo, faltarían muchos elementos en la metodología implementada como para que sea considerado Scrum. Más allá de esto, no se puede decir que los proyectos fracasen o se vean en riesgo por culpa del cambio de metodología o por cómo se implementó Scrum. Más bien, se obtienen resultados similares a los de la metodología anterior, con casi todas sus ventajas y desventajas.

6.3. Trabajos Futuros

Para poder obtener los beneficios que ofrece una metodología ágil como Scrum, es necesario realizar diversos cambios. En primer lugar, debería realizarse a nivel de direcciones y gerencias un análisis de cómo funciona la metodología y cuál es el objetivo a seguir. Si la idea es implementar realmente Scrum, debería comenzarse por una recapacitación general haciendo hincapié en los errores cometidos. Esto debería realizarse tanto a nivel grupal (por ejemplo, con capacitaciones por sector) como individual. Una vez hecho esto, es posible realizar una implementación de Scrum en alguno de los productos de la empresa. Podría tomarse como prueba piloto alguno de los productos más pequeños y que trabaje un equipo independiente de cada sector enfocado sólo en ese producto. Luego se deberían definir los equipos de Scrum y los roles que va a cumplir cada uno. No solo el *Scrum Master* debería controlar la implementación de la metodología, sino que algún agente externo, experto en la materia, debería auditar todo el proceso. Recién con esta experiencia exitosa, y si los resultados son los esperados, podría ir escalando hacia productos y proyectos más ambiciosos. Por otra parte, es necesario realizar un trabajo de adaptación a la metodología con los clientes involucrados. Se debe exponer el funcionamiento de Scrum con sus beneficios y acordar un proyecto que se pueda amoldar a este marco de trabajo.

La metodología actual implementada en los productos para *brokers* de seguros difícilmente pueda adecuarse en el contexto actual. Quizás necesite tener menores presiones, con una baja en la cantidad de proyectos, como para poder realizar los cambios que se requieren. De todas formas, en paralelo a la experiencia con otros productos, podrían ir recuperándose algunas reuniones claves: planificación, revisión y retrospectiva. En la reunión de retrospectiva debería aprovecharse (como nunca se hizo) para revisar el funcionamiento de la metodología y para analizar cómo podemos acercarnos a la definición tradicional de Scrum, teniendo en cuenta todas las dificultades planteadas. Para ayudar al control del proceso, también se requiere definir equipos de

Scrum más chicos y que cada uno cuente con un *Scrum Master*, enfocado solo en su equipo. De esta forma, al menos el área de Desarrollo puede ir corrigiendo algunas diferencias y favorecer la aplicación de Scrum en el resto de la compañía.

Referencias bibliográficas

1. Pressman, R. S., & Troya, J. M. (1988). Ingeniería del software. Un enfoque práctico.
2. Sommerville, I. (2007). *Software engineering*. Addison-wesley.
3. Cervantes Ojeda, J., & Gómez Fuentes, M. D. C. (2012). Taxonomía de los modelos y metodologías de desarrollo de software más utilizados. *Universidades*, *62*(52).
4. Canós, J. H., & Letelier, M. C. P. P. (2012). Metodologías ágiles en el desarrollo de software.
5. Highsmith, J. A., & Highsmith, J. (2002). *Agile software development ecosystems* (Vol. 13). Addison-Wesley Professional.
6. Sutherland, J., & Schwaber, K. (2017). The scrum guide. *The definitive guide to scrum: The rules of the game*.
7. Schwaber, K., & Beedle, M. (2002). *Agile software development with Scrum* (Vol. 1). Upper Saddle River: Prentice Hall.
8. Tinoco Gómez, O., Rosales López, P. P., & Salas Bacalla, J. (2010). Criterios de selección de metodologías de desarrollo de software. *Industrial Data*, *13*(2).
9. Awad, M. A. (2005). A comparison between agile and traditional software development methodologies. *University of Western Australia*.
10. Nerur, S., Mahapatra, R., & Mangalaraj, G. (2005). Challenges of migrating to agile methodologies. *Communications of the ACM*, *48*(5), 72-78.
11. Almeida, F. (2017). Challenges in Migration from Waterfall to Agile Environments. *World Journal of Computer Application and Technology*, *5*(3), 39-49.
12. Kniberg, H. (2015). *Scrum and XP from the Trenches. How we do Scrum.*
13. INTECO. (2009). Ingeniería del Software: Metodologías y Ciclos de Vida. *Laboratorio Nacional de Calidad Del Software*.
14. Fowler, M. (2005) The New Methodology. https://www.martinfowler.com/articles/newMethodology.html
15. Fowler, M., Beck, K., Brant, J. (1999). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley.
16. Stapleton J. (1997). Dsdm Dynamic Systems Development Method: The Method in Practice. Addison-Wesley.
17. Schwaber, K. (1995). Scrum Development Process. OOPSLA'95 Workshop on Business Object Design and Implementation. Austin, USA.
18. Schwaber, K. (2004). Agile project management with Scrum. Microsoft press.

19. Gallego, M. T. (2012). Metodología scrum. Universitat Oberta de Catalunya.
20. Sutherland, J. (2004). Agile development: Lessons learned from the first scrum. *Cutter Agile Project Management Advisory Service: Executive Update*, *5*(20), 1-4.
21. Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., ... & Kern, J. (2001). Manifesto for agile software development.
22. Boehm, B., & Turner, R. (2005). Management challenges to implementing agile processes in traditional development organizations. *IEEE software*, *22*(5), 30-39.
23. Sureshchandra, K., & Shrinivasavadhani, J. (2008, August). Moving from waterfall to agile. In *Agile 2008 conference* (pp. 97-101). IEEE.