Trabajo Fin de Máster

Organización Industrial y Gestión de Empresas

# Estudio sobre Metodologías Ágiles en los Proyectos Software. Propuesta de Plan de Implantación para PYMES.

Autor: Nora Tatiana Quesada Reyes

Tutor: José Luis Andrade Pineda

**Dep. de Organización Industrial y Gestión de Empresas I**

Sevilla, 2020

Trabajo Fin de Máster
Organización Industrial y Gestión de Empresas

# Estudio sobre Metodologías Ágiles en los Proyectos Software. Propuesta de Plan de Implantación para PYMES.

Autor:

Nora Tatiana Quesada Reyes

Tutor:

José Luis Andrade Pineda

Profesor Sustituto Interino

Dep. de Organización Industrial y Gestión de Empresas

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2020

Proyecto Fin de Máster: Estudio sobre Metodologías Ágiles en los Proyectos Software. Propuesta de Plan de Implantación para PYMES.

Autor: Nora Tatiana Quesada Reyes

Tutor: José Luis Andrade Pineda

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2020

El Secretario del Tribunal

# AGRADECIMIENTOS

Un sincero agradecimiento a mi tutor, el Prof. José Luis Andrade Pineda por su ayuda y orientación durante la realización de este trabajo.

A mi prima Daysita por confiar en mí y haber hecho posible la realización de este Máster.

A mi familia en Sevilla: Ana, Norberto, Stephany y Lucas, por su inmenso apoyo y cariño.

A mi pareja, César por su amor, comprensión, entrega y motivación sobre todo, en este periodo de mi vida.

A mis padres y hermanos, por su amor incondicional y estar siempre a mi lado brindándome apoyo y consejos. Pero en especial a mi madre, por inspirarme a ser mejor cada día y ayudarme a convertirme en la mujer que soy.

.

# RESUMEN

Las metodologías ágiles se han extendido en los últimos años en la gestión de los proyectos software debido a su capacidad de adaptación frente a la complejidad e incertidumbre presentes en estos proyectos. Gracias a su enfoque de entrega temprana y continua de valor asegura un resultado que satisface las necesidades del cliente.

En este trabajo, se realiza un estudio de las metodologías ágiles más utilizadas y las ventajas competitivas que éstas aportan a las organizaciones que las adoptan. Se demuestra que el método de trabajo propuesto por estas metodologías y el uso de las diferentes prácticas ágiles genera mejores resultados en comparación a los métodos predictivos.

Finalmente se presenta una propuesta de plan para la implantación de una metodología ágil en una PYME. Para este trabajo se eligió Scrum como metodología a implantar. En este plan se indican los factores que se deben considerar para aumentar las probabilidades de éxito de la adopción de Scrum, entre los que destacan: la cultura organizacional, el uso de un modelo de cambio, la necesidad de una capacitación y acompañamiento de personal experto de la metodología.

Palabras claves: Metodologías ágiles, Gestión de proyectos software, Gestión de proyectos ágiles, Adaptación ágil y Gestión del Cambio.

# ABSTRACT

Agile methodologies have been extended in recent years in software projects management because of their ability to adapt to the complexity and uncertainty of these projects. Due to the early and continuous delivery of value approach, it ensures a result that meets customer needs.

This work conducts a study of the most widely used agile methodologies and the competitive advantages they provide to the organizations that adopt them. It is demonstrated that the processes and tools of these methodologies and the use of the different agile practices produce better results compared to predictive methodologies.

Finally, a plan for the implementation of an agile methodology in a SME is suggested. Scrum was chosen as the agile methodology to be implemented. This plan lists the factors that should be considered to increase the chances of success of the adoption of Scrum, among which are: the organizational culture, the use of a change model, the need for training and accompaniment of agile methodology expert staff.

Keywords: Agile Methodologies, Software Project Management, Agile Project Management, Agile Adaptation and Change Management.

# ÍNDICE

# ÍNDICE DE TABLAS

# ÍNDICE DE FIGURAS

# ABREVIATURAS

| | |
|---|---|
| ADAPT | Awareness, Desire, Ability, Promote, Transfer. |
| AENOR | Asociación Española de Normalización y Certificación. |
| AM | Agile Modeling. |
| CPI | Índice de desempeño del Costo. |
| DDJ | Dr. Dobb's Journal. |
| EDT | Estructura de Desglose del Trabajo. |
| I D | Investigación y Desarrollo. |
| IEEE | Institute of Electrical and Electronics Engineers |
| IPMA | International Project Management Association. |
| ISO | International Organization for Standardization. |
| LSD | Lean Software Development. |
| PMBOK | Guía de los fundamentos para la dirección de proyectos. |
| PMI | Project Management Institute. |
| PO | Product Owner. |
| PU | Proceso Unificado. |
| PYME | Pequeña y Mediana Empresa. |
| RUP | Proceso Unificado Racional. |
| SM | Scrum Master. |
| SMART | Specific, Measurable, Achievable, Relevant, Time-Oriented |
| SPI | Desempeño del Cronograma. |
| UML | Unified Model Language. |
| UNE | Asociación Española de Normalización. |
| WIP | Work In Progress. |
| XP | Extreme Programming. |

# 1 INTRODUCCIÓN

En los últimos años la globalización y los continuos avances tecnológicos han generado un proceso sin precedentes de integración y desarrollo a escala mundial. Las organizaciones han modificado su forma de hacer negocios y se han visto forzadas a adaptarse de forma rápida a los continuos cambios que generan las nuevas tecnologías. Hoy en día las empresas son capaces de crear productos y procesos innovadores en ciclos muy cortos. Una de las principales tecnologías que está impulsando esta tendencia y cuyo crecimiento se ha incrementado en los últimos años es el ***Software***. Roger S. Pressman afirma que el software es la tecnología más importante que existe a nivel mundial porque se encuentra embebido en toda clase de sistemas y forma parte de la actividad cotidiana de las personas (Pressman, 2010, p. 2). Las ventajas que ofrece el uso del software a la sociedad son ciertamente innumerables, por eso Barry Boehm nombra al siglo XXI como el Siglo del Software y asegura que "las próximas décadas definirán al software como el principal elemento que impulsa la vida humana" (Boehm, 2008, p. 78).

Es evidente, que el mundo vive un apogeo del software, la curva de la demanda es creciente y los Ingenieros de Software tienen el desafío de crear un producto fiable, rentable y rápido. Este reto es difícil debido a la complejidad inherente del software. Desde sus inicios, los proyectos software se han caracterizado por tener problemas de calidad, retraso y sobrecoste. La gravedad de esta situación llevó a declarar en 1968 "La Crisis del Software". A partir de ese momento, la comunidad del software se ha dedicado a investigar y crear diversas técnicas y métodos para mejorar sus procesos y obtener resultados favorables. Aunque se han conseguido avances importantes, los problemas continúan, y la tasa de fracasos sigue siendo alta (Pressman, 2010, p. 556).

Existen muchos factores que afectan al éxito de los proyectos software, siendo admitido que el fracaso se debe típicamente a "una combinación de fallos en decisiones técnicas, gestión de proyectos y de negocios" (Charette, 2005, p. 45). Por otro lado, algunos autores afirman que la causa más común de los problemas en los proyectos software es el uso de los procesos predictivos que no se adaptan a la naturaleza del software (Schwaber & Sutherland, 2012, p. 7). Es decir, los procesos predictivos funcionan en proyectos con un entorno estable, tienen especificaciones muy definidas que difícilmente son alteradas, y el flujo de las actividades es secuencial, algo que en el ámbito tecnológico, específicamente del software, es poco probable que suceda. Los proyectos software tienen un alto grado de incertidumbre, el entorno no solamente cambia, sino que lo hace rápidamente, además, existe la probabilidad de añadir modificar o quitar requisitos durante su desarrollo. Por lo tanto, implementar un modelo predictivo a un proyecto software limita su capacidad de adaptación al cambio.

En contraposición a los modelos tradicionales, surgen una serie de metodologías que en los últimos años han obtenido una mayor aceptación en la comunidad del software. Estas

metodologías son llamadas Ágiles y se caracterizan por su naturaleza iterativa e incremental, con periodos cortos de entrega y flexibilidad ante los cambios. Su objetivo principal es la entrega temprana y continua de valor al cliente. Además, fomentan una alta cooperación entre los interesados y el equipo del proyecto, de esta forma, existe una mayor comunicación y se consigue disminuir la tasa de errores y los retrasos.

## 1.1 Justificación del Proyecto

El sector TIC, especialmente del software, es uno de los grandes impulsores de la economía mundial. La importancia del software en todos los aspectos de la vida cotidiana lo hace merecedor de invertir esfuerzos en investigación sobre cómo mejorar su proceso de desarrollo. Desafortunadamente gestionar este tipo de proyectos no es fácil: el software posee propiedades que lo hace particularmente complejo. Los métodos clásicos o prescriptivos de gestión no se adaptan al entorno del software que se caracteriza por los continuos cambios e incertidumbre. Por lo tanto, profundizar en las Metodologías Ágiles cómo alternativa a los métodos actuales de gestión, es una oportunidad para innovar en los procesos de gestión, aumentar la productividad y obtener mejores resultados en los proyectos software.

La adopción de una metodología ágil trae consigo muchas ventajas competitivas para la organización. Esta afirmación ha sido demostrada en diversos estudios y encuestas[1] realizadas a empresas que han utilizado una metodología ágil en sus proyectos software. Las mejoras más relevantes se vieron reflejadas en un aumento de: la productividad, el compromiso de los empleados y la satisfacción laboral, la calidad del software y la satisfacción de las partes interesadas (Cohn, 2010, pp. 11-17)

[1] Las fuentes principales se basan en los siguientes estudios:

- QSMA (Mah, 2008): informe ejecutivo basado en un estudio comparativo entre 26 proyectos ágiles y 7.500 proyectos de desarrollo principalmente tradicionales.
- DDJ (Ambler, 2008): encuesta realizada por Dr. Dobb's Journal a 642 personas.
- VersionOne (VersionOne, 2008): resultado de encuesta a más de 3.000 personas.
- SalesForce.com (Greene & Fry, 2008): reflejan los resultados éxitos de la empresa después de 7 años de haber adoptado Scrum en la organización.
- David Rico (Rico, et al., 2009) ha recopilado trabajos académicos y de investigación. Además de una investigación propia de 51 encuestas de proyectos ágiles.

## 1.2 Objeto del Proyecto

**Objetivo General**

El objetivo general de este trabajo es realizar un estudio sobre las Metodologías Ágiles y las ventajas que ofrece en la Gestión de Proyectos Software. Adicionalmente, se elabora una propuesta para la implantación de las Metodología Ágiles en una PYME del sector de desarrollo de software.

**Objetivos Específicos**

- Introducir al lector en el contexto de los proyectos Software. Resumir aspectos teóricos del software, su proceso y tipos modelos de proceso.
- Estudiar sobre el estado actual de los Proyectos Software.
- Analizar las Metodologías Ágiles y el beneficio que aportan a la Gestión de Proyecto Software.
- Realizar una comparación entre las Metodologías Ágiles.
- Ofrecer un plan de implantación de Metodologías Ágiles en una PYME.
- Realizar las conclusiones sobre la investigación realizada.

## 1.3 Estructura del Proyecto

Una vez presentada la visión general de la justificación y objetivos con que se planteó este TFM, se presenta a continuación los elementos correspondientes a los sucesivos capítulos:

1. El Capítulo 2 se reserva a los **Aspectos Teóricos - Proyecto Software:** se definen conceptos generales de la Gestión de Proyectos Software. También se hace una descripción de los diferentes Modelos del Proceso desarrollo, sus ciclos de vida, ventajas y desventajas. Finaliza con una breve reseña de la situación actual de los proyectos software.
2. El Capítulo 3 es una **Revisión del Estado del Arte**, en que además se realiza una introducción más fundamentada a las características del Movimiento Ágil. Se lleva a cabo una revisión bibliográfica acerca las Metodologías Ágiles utilizadas en la actualidad. Se realiza una comparación entre ellas.
3. El Capítulo 4 se propone un **Plan de Implantación de Metodologías Ágiles**. Se lleva a cabo un análisis acerca de aspectos importantes que se deben tomar en cuenta para adoptar una metodología ágil en una PYME y desarrolla un Plan para su debida implantación.
4. Finalmente, en el Capítulo 5 **Conclusiones** presenta el análisis de los resultados obtenidos, y los aspectos más reseñables del proyecto realizado.

# 2 PROYECTOS SOFTWARE

En este capítulo se introduce la disciplina Gestión de Proyectos software, se definen conceptos generales, características y restricciones. También se hace una breve explicación de los modelos de Proceso de Desarrollo Software. Se concluye con un análisis de la situación actual de la Gestión de Proyectos Software.

## 2.1 Aspectos Teóricos

El origen de un proyecto puede tener diferentes causas: satisfacer una necesidad, dar solución a un problema o actuar como agente de cambio. Cuando se concibe la idea del proyecto es importante definir los objetivos que expresan su propósito. Según Doran (Doran, 1981), para lograr tener un enfoque en los problemas y dar un sentido claro a la dirección de proyecto, los objetivos que rijan el proceso de gestión de un proyecto deben ser específicos, medibles, asignables, realistas y temporales (S.M.A.R.T. como acrónimo en lengua inglesa). Así, se posibilita un mecanismo para hacer el correcto control y monitoreo del progreso del proyecto, pero además se fija un marco de referencia en el que el responsable de proyecto se asegura que todas las personas involucradas en el mismo comprenden los objetivos. Esto es especialmente relevante en entornos complejos, en los que para asegurar el éxito, además de definir los objetivos, debe asegurarse de que haya una comunicación clara entre todos los actores del proyecto.

Existen muchas definiciones del significado de proyecto, para este trabajo se toma como referencia tres fuentes importantes:

- Según la Guía del PMBOK (Guide to the Project Management Body of Knowledge) un proyecto es un esfuerzo temporal que se lleva a cabo para crear un único producto, servicio o resultado (PMI, 2013a, p. 3).
- El IPMA (Institute Project Management Association) indica que un proyecto es una operación limitada en tiempo y coste para materializar un conjunto de entregables definidos (el alcance para cumplir los objetivos el proyecto) de acuerdo con unos requisitos y estándares de calidad (IPMA, 2009, p. 31).
- La norma UNE ISO 21500:2013 Directrices para la dirección y gestión de proyectos define un proyecto como un conjunto único de procesos que consta de actividades coordinadas y controladas, con fechas de inicio y fin, que se llevan a cabo para lograr los objetivos del proyecto (AENOR, 2013, p. 8).

Estas definiciones reflejan que, indistintamente de su naturaleza, todos los proyectos poseen características comunes (Villafiorita, 2014, pp. 1-3; Maley, 2012, pp. 4-5; PMI, 2013a, p. 3):

- **Un proyecto es temporal,** con un inicio y un final, y tiene un propósito único.
- **Produce una salida,** ya sea un producto o servicio que puede ser tangible o intangible.

- **Necesita recursos y son limitados**. Es el conjunto de los elementos que se requieren para realizar un proyecto (p.e., tiempo, personas, equipo, materiales, etc). El éxito del proyecto depende de su acertada estimación y eficiente utilización.
- **Requiere una elaboración progresiva.** Durante el proyecto se realizan diferentes actividades que son monitoreadas continuamente para medir su avance. A medida que el proyecto avanza, se reduce la libertad de realizar cambios y aumenta su coste.
- **Su desarrollo implica riesgo e incertidumbre.** Un proyecto posee un nivel de riesgo que depende de factores internos, como su complejidad y restricciones, y de factores externos, como políticas gubernamentales, toma de decisiones de los interesados[2], entre otros.
- Otro concepto a definir es el **Software**. John W. Tukey (Tukey, 1958, p. 2)introdujo por primera vez el término *software*, en un artículo publicado en 1957 en la revista American Mathematical Monthly[3] y lo define como un programa que proporciona instrucciones a un ordenador para realizar funciones específicas. Por su parte el IEEE ofrece un concepto más completo y establece que el software es un conjunto de programas de computadora, procedimientos y posiblemente documentación asociada y datos relacionados con la operación de un sistema de computadora (IEEE, 1990, p. 69). Esta definición señala que, desarrollar software es un trabajo altamente intelectual que no solo se limita a dar una instrucción mediante una simple escritura de líneas de código, también involucra actividades de gestión. Crear un producto software es considerada como una de las tareas más complejas que realiza el ser humano (Villafiorita, 2014, p. xv), ya que es el resultado de los conocimientos, experiencias y continuo aprendizaje de las personas involucradas en el proyecto (Ruhe & Wohlin, 2014, p. 2).

Así pues, en este TFM el término **proyecto software** se refiere al conjunto de actividades necesarias para el desarrollo del software. La disciplina que se encarga de todos los aspectos del desarrollo software es la Ingeniería de Software[4], que se define como la aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento de software (IEEE, 1990, p. 70). Debemos indicar que la Ingeniería del Software no se limita a la programación, sino que involucra todas las etapas de la producción de la salida esperable 'componente o producto software', desde la especificación de requisitos hasta su entrega final, incluyendo su mantenimiento futuro. El término proceso de desarrollo software se usa comúnmente para referirse al conjunto de actividades que están relacionadas, siguen un flujo (es decir, una secuencia

[2] Cuando se menciona a la parte interesada de un proyecto nos referimos al individuo, grupo u organización que puede afectar o ser afectado por las decisiones o resultados del proyecto (PMI, 2017, p. 550).

[3] En el artículo John W. Tukey menciona: "Hoy *en* día el *'software'* que comprende las rutinas interpretativas cuidadosamente planificadas, compiladores y otros aspectos de la programación automática son al menos tan importantes a la calculadora electrónica moderna como su 'hardware' de tubos, transistores, cables, cintas y similares".

[4] El nombre de esta disciplina fue utilizado por primera vez en la conferencia de la Organización del Tratado del Atlántico Norte (OTAN) de 1968 y "fue escogido deliberadamente por ser provocativo, pues implicaba la necesidad de manufacturar software según las bases teóricas y las disciplina prácticas que son tradicionales en otras ramas de ingeniería" (Bauer, et al., 1969, p. 8).

en un tiempo específico) y da como resultado entregables del proceso de adición de valor a la solución software en desarrollo (documentación, componentes, modelos, entre otros).

Si en general los proyectos suponen retos que requieren de esfuerzo, dedicación y la coordinación de varios elementos para conseguir un objetivo, los proyectos software en particular, son un desafío debido a su naturaleza abstracta, la falta de claridad en los requisitos, los continuos cambios a los que están expuestos, entre otros factores que se explicarán más adelante. Aún hoy día, en que hay una madurez de varias décadas de Ingeniería del Software, los proyectos software siguen caracterizándose con frecuencia por los retrasos, sobrecostes y problemas de calidad. Por eso es necesario el uso de técnicas, herramientas y/o metodologías que faciliten la organización y coordinación de los procesos involucrados en los proyectos software.

Como en todo proceso de gestión, el proceso software une a las personas, procedimientos y herramientas (O'Regan, 2011, p. 14). Puede concluirse por tanto que, para el éxito en la ejecución de los proyectos software, es importante poseer tanto habilidades de gestión de proyectos y como de ingeniería de software (Ahmed, 2011, p. 6).

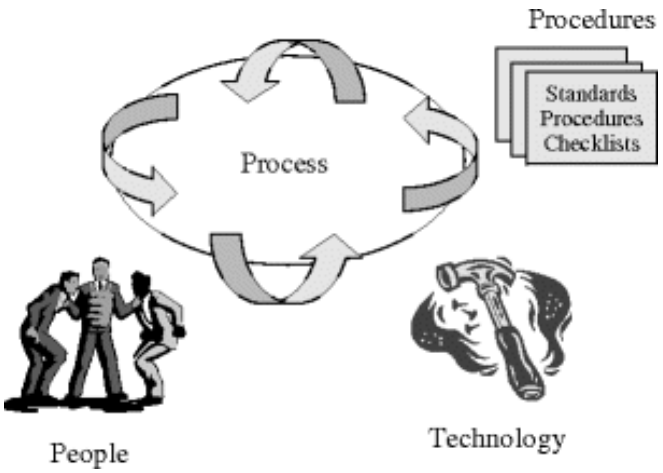


Figura 2-1 El proceso software como pegamento para personas, procedimientos y herramientas. (O'Regan, 2011, p. 3).

## 2.2 Gestión de Proyectos Software

Los proyectos software poseen ciertas características que los hace particularmente complejos en comparación a otro tipo de proyectos. Una de ellas, es su naturaleza intangible. El software es el resultado de un proceso cognitivo y de colaboración entre un equipo de personas. De aquí se derivan ciertas dificultades, algunas de ellas son mencionadas por Marco y Lister, dos consultores con experiencia en proyectos de desarrollo software concluyeron que, los principales problemas que enfrentaron cuando trabajaban en estos proyectos no fueron tanto a nivel tecnológico sino sociológico. Los autores se referían a problemas relacionados al equipo de trabajo, por ejemplo:

problemas de comunicación, la falta de motivación, el ambiente laboral, la alta rotación, entre otros (DeMarco & Lister, 2013, p. 4). Estas dificultades son originadas principalmente por las propiedades esenciales del software y otros factores relacionados al entorno tecnológico.

### 2.2.1 Propiedades esenciales del Software

El desarrollo software es un proceso complejo que necesita ser gestionado. La creación de software exige un gran esfuerzo intelectual y creativo. Generalmente, cuando una organización decide innovar en su negocio y tener una mejora competitiva, estas acciones conllevan a la introducción de nuevas tecnologías, como puede ser una nueva solución software. La innovación es la creación de algo nuevo y diferente, lo que a su vez significa incertidumbre y riesgo.

En su origen, la idea de introducir un elemento software en las organizaciones típicamente parte de personas que patrocinan la idea por estar interesadas en la explotación de la ventaja competitiva que resultará de él. Después de un proceso en el que estos interesados en la idea identifican las necesidades requeridas del software, conciben un resultado final de lo que se espera de él. Sin embargo esta concepción misma está necesariamente abierta a modificaciones importantes, pues la mayoría de los clientes (sean internos a la organización o externos a la misma) no están seguros al 100% de lo que quieren del software hasta que lo ven e interactúan con él.

En su artículo *No Silver Bullet. Essence and Accidents of Software Engineering,* Frederick Brooks enumera las cuatro propiedades esenciales del software y explica por qué el software es diferente y dificultoso en comparación a otros productos creados por el ser humano (Brooks, 1987). Estas propiedades son las siguientes:

1. **Complejidad:** El desarrollo software requiere mucho esfuerzo debido a su naturaleza abstracta. A diferencia del hardware[5], el software no posee elementos repetidos, éstos son únicos y están encapsulados en funciones, rutinas u objetos que son invocados tantas veces como sea necesario (Fairley, 2009, p. 3). Además, pueden tener diferentes estados e interacciones en un sistema. Por esta razón, se dice que no existen cambios pequeños, cuando se modifican los requerimientos del software, aunque sea muy sencillo: este cambio puede alterar varios elementos y la forma en la que interactúan con otros componentes y en sus diferentes estados (Fairley, 2009, pp. 3-4). El software carece de propiedades físicas de manera que es difícil dimensionar su tamaño. A esto se añade, que la ampliación del desarrollo del software no sigue un crecimiento lineal, es decir, sus elementos no se repiten en tamaños más grandes, más bien, es el número de elementos diferentes los que se incrementan e interactúan en el sistema. Según Brooks, la complejidad del software deriva en problemas administrativos como la

[5] Brooks expone este enunciado haciendo una comparación entre el hardware y software: "Los computadores digitales son en sí más complejos que la mayor parte de las cosas que las personas construyen: tienen un enorme número de posibles estados. Esto hace que concebir, describir y probarlas sea tan difícil. Los sistemas de software tienen órdenes de magnitud más estados que los computadores" (Brooks, 1987, p. 11).

comunicación eficaz entre las personas de un equipo y la dificultad de supervisar el trabajo. Como consecuencia, existe una probabilidad muy alta de cometer errores y que éstos sean identificados muy tarde, ya cuando el software está en producción. El desarrollo software es un trabajo con mucha carga mental, estrés y desgaste en las personas, lo que provoca un aumento en la rotación del personal (Brooks, 1987, p. 11).

2. **Conformidad:** El software debe cumplir con cada una de las funcionalidades establecidas en el proyecto. Asegurar la conformidad del software no es una tarea fácil, principalmente porque es realizado por personas con diferentes criterios y que utilizan distintas metodologías de trabajo. En esto se basa Brooks cuando argumenta que, la gestión de la mayor parte de la complejidad del software es arbitraria. En la actualidad existen herramientas que ayudan a detectar errores de sintaxis en una línea de código, en cambio, no hay ninguna para detectar un error de la lógica del programa empleada por el desarrollador software (Fairley, 2009, p. 4). El problema de la conformidad radica en que muchos de estos errores son detectados en una etapa avanzada debido: a una especificación incorrecta o incompleta, una cobertura insuficiente de las pruebas[6] o la interpretación errónea de las especificaciones por parte del desarrollador. La dificultad de lograr la conformidad del software aumenta cuando éste debe adaptarse a otras interfaces de un sistema, ya que requiere de mucho esfuerzo y múltiples pruebas para comprobar que el software cumple con los requisitos y que la interacción con otras interfaces se ejecuta sin ningún contratiempo (Brooks, 1987, pp. 11-12).

3. **Mutabilidad-Cambio:** El cambio es parte esencial del software. Los sistemas en el entorno en que ha de encajarse una solución software se encuentran en continuos cambios, que en particular pueden derivar en nuevas necesidades de los usuarios (sea para satisfacer un requerimiento nuevo del cliente, adaptarse a nueva ley o regulación, o por reacción a cambios en la tecnología o las tendencias del mercado) a las que típicamente se pide que la solución software se adapte. Al ser el software el elemento más flexible en un sistema, es el primero que se considera para ser modificado. Pues bien, tal y como sucede en cualquier gestión de proyecto, será importante que el director del proyecto software analice si los cambios propuestos están dentro o fuera del alcance actual del proyecto. En caso de estar fuera del alcance, se debe evaluar en qué grado afecta a las demás restricciones del proyecto (tiempo, coste, recursos, calidad, riesgos, etc.) para minimizar el impacto en cada una de ellas (Fairley, 2009, p. 5). Brooks indica que todo software que tiene éxito sufrirá alguna modificación a lo largo de su vida útil, ya sea para crear nuevas funcionalidades, demandas por los usuarios o porque el software debe adaptarse a nuevas versiones de ordenadores u otros dispositivos electrónicos (hardware) (Brooks, 1987, p. 12).

[6] Si bien es cierto que, las pruebas del software se realizan durante todo su ciclo de vida con el objetivo de detectar no conformidades, es imposible revisar el software por completo. Pressman lo explica muy claramente-que nunca se termina de probar el software sino que esta responsabilidad pasa del ingeniero de software al usuario final (Pressman, 2010, p. 388).

4. **Invisibilidad:** El software no posee propiedades físicas, es intangible y está compuesto por un conjunto de líneas de códigos que dan instrucciones para que determinadas órdenes se realicen. Aunque estos efectos se pueden observar cuando el software es ejecutado, el software en sí mismo no se puede visualizar porque es una entidad inmaterial. Los ingenieros de software tienen que apoyarse en modelos gráficos para representar los comportamientos del software y sus interacciones con otros elementos de un sistema.

   La determinación del avance exacto del desarrollo del proyecto software tiene precisamente complicación por esta invisibilidad. El director del proyecto debe utilizar diferentes técnicas que le ayuden a determinar el avance del proyecto, y evitar caer en lo que comúnmente se conoce como el síndrome del 90%, que es una forma de decir que el software "está casi completo" (Fairley, 2009, pp. 22-23; Brooks, 1987, p. 12). Durante el desarrollo muchas partes del software se terminan de forma individual, pero no es hasta el momento en que éstas se integran y superan las pruebas de validación, cuando se puede decir que el software está terminado y listo para ponerlo en producción.

### 2.2.2 Restricciones de los Proyectos Software

El director del proyecto es quien tiene que cumplir con los objetivos del proyecto dentro de sus limitaciones establecidas, con acciones que se ven constreñidas por el llamado triángulo de hierro *(véase figura 2-2)*. Los lados del triángulo representan un tipo de restricción al que todo proyecto se enfrenta durante su desarrollo. En la sexta edición del libro del PMBOK, una restricción es definida como un "factor limitante que afecta a la ejecución de un proyecto, programa, portafolio o proceso" (PMI, 2017, p. 723). El alcance, tiempo coste y calidad son las típicas restricciones en la gestión de proyectos tradicional. Todas estas variables están relacionadas, es decir, si una de ellas se modifica, ésta afecta de forma directa a las demás. Por su parte el PMBOK, en su cuarta edición, añade a este grupo de restricciones dos elementos importantes que también limitan a un proyecto como ser los recursos y el riesgo. A continuación, se explica cada una de una de ellas.


Figura 2-2 Triángulo de Hierro de la Gestión de Proyectos

**El Alcance** define los límites del proyecto y las tareas que necesitan ser completadas. El PMBOK define el alcance como "la suma de productos, servicios y resultados a ser proporcionados como un proyecto" (PMI, 2013a, p. 527). En la planificación, el director del proyecto junto con los interesados, recopilan la información de las necesidades de negocio que deben ser satisfechas con la realización del proyecto. Después de analizar esta información, toman la decisión de lo que se incluye y se descarta en el proyecto. Las necesidades de negocio identificadas se convierten en los requisitos del proyecto y forman parte del enunciado del Alcance que es documentado en el Plan para la Dirección del Proyecto.

La definición del alcance del proyecto es una parte crítica durante la planificación ya que define todo el trabajo que debe realizarse en el proyecto (Wisocky, 2014, p. 11). Por ello, esta información tiene que ser lo más precisa y completa posible para evitar confusiones en el equipo del proyecto que puedan provocar retrasos y pérdidas económicas.

Durante el proceso de ejecución, el director del proyecto debe asegurar el cumplimiento de las actividades de acuerdo al plan del proyecto. Sin embargo, en ocasiones el proyecto puede ser alterado debido a factores internos y externos que obligan a modificar el plan original. Los cambios en la planificación pueden ser solicitados por cualquiera de los interesados. El director del proyecto es el responsable de analizar cada solicitud de cambio y elaborar un informe acerca del posible impacto que puede tener la introducción de estos cambios en el proyecto. Este informe es presentado a las personas que tienen la autoridad para decidir sobre el proyecto y se encargan de aceptar o rechazar los cambios[7]. Una vez autorizados los cambios, el director del proyecto debe replantear el alcance y adaptarlo al nuevo contexto (Wisocky, 2014, pp. 11-12). En los proyectos software en particular, los cambios deben asumirse como algo inevitable (Fairley, 2009, p. 111).

**El coste** es una restricción muy importante que debe ser gestionada durante todo el ciclo de vida del proyecto. Una acertada estimación del coste total y un presupuesto detallado permiten al director del proyecto planificar y controlar todas las actividades económicas involucradas, y minimizar el nivel de riesgo del proyecto. La gestión de los costes de un proyecto involucra procesos de planificación y estimación, elaboración de presupuestos, evaluación y obtención de financiación, gestión y control de los costos para que el proyecto finalice dentro del presupuesto aprobado. Además, se debe incluir una reserva para la gestión de riesgos que ayude a solventar situaciones imprevistas y que pueden provocar el incremento del presupuesto (PMI, 2013a, pp. 109, 203).

**Tiempo o Cronograma.** La definición de la fecha de entrega del proyecto es el punto de partida para la realización del cronograma. El director del proyecto dispone de un tiempo establecido para cumplir con el cronograma y debe utilizarlo de forma eficiente (Wisocky, 2014, p. 13). El PMBOK

[7] Al inicio del proyecto se elabora el Acta de Constitución en el que se incluye un Marco de Gobernanza del Proyecto. En este Marco se establece un Comité Directivo que son las personas que tienen autoridad para tomar decisiones del proyecto. El Comité Directivo lo integran "partes interesadas que son clave, que representan las unidades organizativas afectadas por el cambio que trae el proyecto" (Wagner, 2015).

define el cronograma del Proyecto como la salida de un modelo de programación que presenta actividades vinculadas con fechas planificadas, duraciones, hitos y recursos (PMI, 2013a, p. 536).

Una herramienta muy utilizada para definir el alcance y el cronograma del proyecto es la Estructura de Desglose del Trabajo (EDT por sus siglas en español) que consiste en descomponer de forma jerárquica (de un nivel superior a niveles inferiores) el trabajo total en piezas más pequeñas que son agrupadas en paquetes de trabajo (Patel, 2008, p. 6). Entre los beneficios de la EDT se puede mencionar que facilita el control y monitoreo de las tareas gracias a una estimación adecuada tanto del tiempo, coste y recursos. En la EDT se identifican los responsables de cada tarea y se procura distribuir de forma equitativa la carga de trabajo entre los miembros del equipo del proyecto. Para obtener un mejor resultado, se recomienda que una EDT no debería ser muy restringida porque podrían obviarse detalles importantes o ser demasiada extensa que dificulte el seguimiento del trabajo. Por ejemplo, algunas organizaciones optan por el uso de reglas de descomposición, como puede ser la regla "4/40" que indica que un paquete de trabajo al menos debe durar 4 horas pero no puede sobrepasarse de 40 horas. De igual forma ocurre con la regla "8/80", la duración del paquete de trabajo oscila entre 8 y 80 horas (Caamaño, 2011, p. 136).

Durante la elaboración del cronograma del proyecto se deben identificar las interdependencias entre las tareas y su duración, siendo lo más importante, la identificación de aquellas tareas que tienen una mayor duración y que forman el camino más largo a través de un proyecto, esto es mejor conocido como la ruta crítica del proyecto. Un retraso en la ruta crítica afectará directamente la fecha de entrega del proyecto (Pressman, 2010, pp. 628-629). También, deben incluirse hitos que puedan ser revisados por medio de indicadores objetivos que sean capaces de evaluar el alcance y la calidad de las tareas que son finalizadas. Los hitos son eventos significativos en el proyecto que se utilizan como puntos de referencias para señalar un logro importante en alguna etapa del proyecto. Por ello, los hitos son considerados como otra forma de medir el avance del proyecto.

**La Calidad.** La ISO define la calidad como el grado en el que un conjunto de características inherentes cumple con los requisitos (ISO, 2015). En (Sommerville, 2011)un proyecto se consideran dos tipos de calidad: la calidad del producto que está enfocada a cumplir con todas las especificaciones requeridas y, la calidad del proyecto relacionada a la calidad del proceso mismo de gestión a lo largo del proyecto (Wisocky, 2014, p. 12). La forma en que se gestiona la calidad, y qué metodologías y técnicas se utilizan depende la naturaleza del proyecto.

En un proyecto, la calidad depende del balance y éxito en la gestión de factores como el alcance, tiempo y coste. Por ejemplo, en un proyecto que marcha retrasado y/o ha superado el presupuesto, el director del proyecto tendrá que tomar medidas para alinear el proyecto nuevamente al plan original. Esta acción podría llevar a comprometer el cumplimiento de todos los requisitos inicialmente considerados. Cuando esto se hace de puertas adentro (sin comunicación al cliente), se suele recurrir a la disminución de los controles, análisis y las pruebas para detectar fallos, lo que acaba afectando directamente a la calidad (Siegelaub, 2007).

**Los recursos** son activos, tanto internos como externos, necesarios para ejecutar un proyecto. Entre ellos están los recursos humanos especializados, equipos, servicios, suministros, materias primas,

materiales, presupuestos o fondos (PMI, 2013a, p. 560). Dependiendo de su necesidad, algunos recursos son utilizados durante todo el proyecto (recursos fijos) y otros solamente se requieren en alguna de sus etapas (recursos variables) (Wisocky, 2014, p. 13).

En la fase de planificación del proyecto se establecen las fechas de inicio y finalización de cada tarea y se define cuándo y cómo los recursos serán utilizado (Hughes & Cotterell, 2009, p. 131). La asignación de recursos puede ser una actividad muy compleja puesto que deben considerarse varios factores como: el orden de las tareas, su interdependencia y las limitaciones de los recursos en tiempo y cantidad[8] (p.e., si estos recursos son compartidos con otras tareas). El director del proyecto debe realizar una estimación correcta de los recursos, hacer una programación y distribución adecuada y velar por que se utilicen de forma eficiente para que estén disponibles cuando sean necesarios. Cabe resaltar, que la prioridad al momento de asignar los recursos siempre será atribuida a las tareas que forman parte de la ruta crítica del proyecto (Hughes & Cotterell, 2009, p. 199). En algunos casos, cuando los recursos son necesarios en varias tareas, se identifica aquella tarea que tiene holgura y puede retrasarse sin afectar a otra tarea posterior. De esta forma, la distribución de los recursos es balanceada. El director del proyecto debe considerar distintos escenarios en caso de que algún recurso pueda faltar durante el desarrollo del proyecto (Fuller, et al., 2008, p. 253).

**El riesgo** es un evento incierto que puede tener un efecto positivo o negativo en el proyecto. La guía del PMBOK recomienda crear una lista de riesgos identificados y elaborar un plan de respuesta para cada uno. Además, debe asignarse la probabilidad de que un riesgo ocurra y el grado de impacto positivo o negativo (riesgo muy bajo, bajo, moderado, alto, muy alto) que puede tener sobre el proyecto. De esta forma, el director del proyecto cuenta con un plan de contingencia para evitar o al menos disminuir un efecto negativo en el alcance, tiempo, coste, cronograma o calidad del proyecto. Si un riesgo llega a producirse, debe realizarse un seguimiento para verificar que el plan de respuesta se está ejecutando y evaluar su eficacia frente al riesgo. La actualización y retroalimentación sobre el estado del proyecto es muy importante para afrontar un riesgo, porque una toma de decisiones acertadas depende de la veracidad y exactitud de la información que le sea entregado al director de proyectos (PMI, 2013a, pp. 327-331, 349, 562).

Existen otras restricciones del ámbito tecnológico que limitan a los proyectos software y son mencionadas en la extensión de Software de la Guía PMBOK (PMI, 2013b) y son:

- Estado de la tecnología de hardware y software.
- Plataformas del hardware, plataforma del software, sistema operativo y protocolo de comunicación.
- Herramientas de desarrollo software.

[8] Cuando más de una tarea comparten recursos, se intenta balancear la distribución de estos. en algunos casos puede ser necesario flotar una tarea, es decir, retrasar para que no afecte a la tarea posterior.

- Arquitectura de Software.
- Requisitos de compatibilidad con versiones anteriores y posteriores.
- Reutilización de componentes de software de una biblioteca.
- Uso de componentes software de código abierto versus cerrado.
- Uso de componentes de software suministrados por el clientes.
- Interfaces para hardware y otro software
- Creación y uso de la propiedad intelectual.

Existen otros factores que dificultan la gestión de los proyecto de software. Estos factores están descritos en Anexo 6.1

## 2.3 El Proceso del Software

El proceso de desarrollo software es el proceso mediante el cual se crea una solución software en respuesta a unas necesidades, para ello se lleva a cabo una serie de actividades como "Traducir las necesidades a requerimientos, transformar requerimientos en diseño, implementar el código del diseño, probar el código, y en ocasiones, instalar y comprobar el software para su uso operativo" (IEEE, 1990, p. 67). Estas actividades son realizadas en ciclos iterativos e involucran el uso de diversos procedimientos y técnicas. No existe un único proceso o un proceso ideal para desarrollar software, la elección de un enfoque u otro depende del tipo de proyecto a realizar (Pressman, 2010, p. 26). En ocasiones, las organizaciones diseñan su propio proceso ajustándolo a las necesidades del proyecto (Fairley, 2009, p. 40; Sommerville, 2011, p. 28).

Sommerville establece que, independientemente del software a realizar, existen cuatro actividades comunes que están involucradas en un proceso software (Sommerville, 2011, pp. 28,36 - 43):

- **Especificación del software:** o la ingeniería de requerimientos es un proceso en el que se identifican las necesidades del negocio y sus limitaciones. Este proceso incluye siete tareas: concepción, indagación, elaboración, negociación, especificación, validación y administración (Pressman, 2010, p. 102). Una vez que las necesidades del negocio son discutidas y revisadas por los usuarios finales y el equipo de desarrollo, se recopila toda la información y se crea la documentación de requisitos[9]. Esta documentación incluye la definición de los requisitos de usuario que debe satisfacer el software y, una descripción más detallada para los desarrolladores acerca las funcionalidades que debe ofrecer el software.

[9] La documentación de requisitos describe cómo los requisitos individuales cumplen con las necesidades de negocio del proyecto. Los requisitos pueden comenzar a un alto nivel e ir convirtiéndose gradualmente en requisitos más detallados, conforme se va conociendo más información acerca de ellos (PMI, 2017, p. 147).

- **Diseño e implementación del software:** con los requisitos obtenidos en la etapa anterior, se continúa con el diseño del software, que es "la descripción de la estructura del software que se va a implementar, los modelos y las estructuras de datos utilizados por el sistema, las interfaces entre componentes del sistema y, en ocasiones, los algoritmos usados" (Sommerville, 2011, p. 38). En la implementación del software, los desarrolladores utilizan lenguajes de programación para desarrollar una versión ejecutable del software. En esta actividad se incluyen pruebas del código implementado para detectar errores y hacer su correspondiente depuración.
- **Validación del software:** es el conjunto de pruebas utilizadas para asegurar que el software cumple con los requisitos que ha especificado el cliente. Estas pruebas se dividen en tres etapas: (1) los componentes del software se someten a prueba de forma individual, después (2) son integrados al sistema de software y, en el que se observa y evalúa la interacción entre ellos. Una vez que se ha comprobado el correcto funcionamiento de los componentes, (3) se realizan las pruebas de aceptación por que consiste en la puesta en marcha del software con datos suministrados por el cliente, para que compruebe sí cada una de las funcionalidades cumple con los requisitos. Aquí se detectan los errores y omisiones realizados durante la definición de los requisitos del software. Cuando estos defectos son corregidos y el software es validado, se da por terminado el ciclo de desarrollo software.
- **Evolución del software:** el software se caracteriza por su flexibilidad y adaptación a los cambios, permitiendo realizar mejoras en cualquier momento. Según las necesidades del cliente se realizan modificaciones o se añaden nuevas funcionalidades al sistema para evitar que el software se vuelva obsoleto y sin utilidad. El continuo mantenimiento del software es una actividad importante ya que asegura su buen funcionamiento lo que permite extender su vida útil y obtener una mayor rentabilidad.

## 2.4 Modelos de proceso software

El proceso del software sigue un flujo de trabajo que especifica cómo están organizadas las actividades y tareas y define la secuencia y el tiempo en que se realizan (Pressman, 2010, p. 28). En un proyecto software es fundamental establecer el tipo de proceso o enfoque que va a ser utilizado. La elección del tipo de proceso dependerá de las características y necesidades del proyecto.

Un proceso de desarrollo software tiene un ciclo de vida asociado que está compuesto por varias fases: requisitos, diseño, implementación, pruebas y, a veces, fase de instalación y verificación. El ciclo de vida de desarrollo software comienza desde el momento en que se decide desarrollar un producto software y finaliza cuando es entregado (IEEE, 1990, p. 67). Mohapatra "el reconocimiento de dicho ciclo de vida de desarrollo de software es la clave para un desarrollo de software exitoso" (Mohapatra, 2010, p. 17).

Como se ha mencionado, el desarrollo software se considera una tarea compleja debido a que

involucra procesos intelectuales y creativos; es el resultado de los procesos cognitivos, del aprendizaje social de una o varias personas (PMI, 2013b, p. 6; Sommerville, 2011, p. 28; Pressman, 2010, p. 26). Por ello, es fundamental adoptar un modelo de proceso y adaptarlo a las necesidades del proyecto (Fairley, 2009, p. 53) ya que ofrece un marco de trabajo que ayuda a tener una visión global del proceso software. El modelo de un proceso es la representación abstracta de un proceso (Sommerville, 2011, p. 742); éste proporciona un punto de partida, una referencia en común para la definición y flujo de las actividades durante el ciclo de vida del software. Un modelo de proceso mejora la comprensión y comunicación entre las personas interesadas del proyecto y el equipo de desarrollo.

En la ingeniería de software existen diferentes modelos de proceso que se caracterizan por el flujo que siguen sus actividades que puede ser secuencial, iterativo, incremental o evolutivo.

### 2.4.1 Modelos de Proceso Prescriptivos o Tradicionales

Los modelos prescriptivos, también llamados modelos tradicionales, son aquellos que "prescriben un conjunto de elementos del proceso: actividades estructurales, acciones de ingeniería de software, tareas, productos del trabajo, aseguramiento de la calidad y mecanismos de control del cambio para cada proyecto" (Pressman, 2010, p. 33). Estos modelos son bastante estructurados y rígidos de modo que las actividades se realizan tal y como se han definido en el cronograma. El éxito de estos modelos radica en la ejecución precisa del Plan del Proyecto evitando cualquier tipo de variación. Los modelos prescriptivos tienen un enfoque sistemático, su proceso de desarrollo es secuencial, es decir, una fase debe ser terminar para continuar con la siguiente. El Modelo Cascada y el Modelo V son un ejemplo de modelos prescriptivos. A continuación, se realiza una breve explicación de ambos modelos.

#### 2.4.1.1 Modelo Cascada

El modelo cascada es un enfoque clásico en la gestión de proyectos y un ejemplo de un proceso *dirigido por un plan*[10]. Este modelo fue explicado por primera vez por Winston Royce en 1970 (Royce, 1970) y ha sido durante muchos años el modelo que se ha usado para gestionar el proceso de desarrollo software[11].

Como se puede observar en la figura 2-3, el flujo de las actividades en el modelo cascada es secuencial y se agrupan en cinco fases: Análisis y definición de requerimientos, diseño del sistema y del software, Implementación y prueba de unidad, integración y prueba de sistema y operación y

[10] Un proceso dirigido por un plan consiste en la planificación de toda las actividades (el plan incluye: el calendario de trabajo, los responsables, la documentación del trabajo que se realiza, entre otros) antes de dar inicio con el proceso (Pressman, 2010, p. 30).

[11] la Ingeniería de Software adoptó este modelo en los proyectos software siguiendo el ejemplo de otras disciplinas de ingeniería que lo utilizaban para organizar sus proyectos).

mantenimiento. Una fase se considera terminada cuando los hitos (productos de trabajo resultantes de cada fase), son revisados y validados. Estos productos se convierten en la línea de base[12] que sirve como punto de referencia para iniciar la siguiente fase del proyecto (Fairley, 2009). Los hitos funcionan como puertas de control, lo que significa que la puerta no se abrirá para pasar a la siguiente fase hasta que se corrijan los problemas identificados durante la revisión de hitos (Fairley, 2009, p. 57). Un punto relevante en el modelo cascada es su énfasis en la documentación. Al final de cada fase se genera la documentación del resultado obtenido, la cual debe ser autorizada para dar validez a los productos de trabajo generados. (Sommerville, 2011, p. 31).

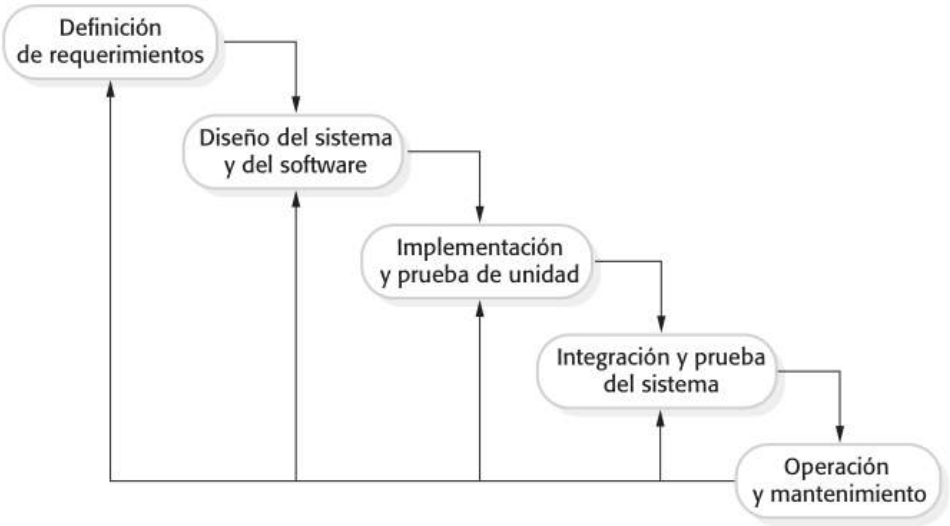


Figura 2-3 El modelo en cascada (Sommerville, 2011, p. 30).

El modelo cascada es adecuado para en proyectos software que poseen las siguientes características (O'Regan, 2011, p. 22; Pressman, 2010, p. 33; Royce, 1970, p. 328; Sommerville, 2011, p. 32) :

- Los requisitos son conocidos, muy bien definidos y establecidos desde el principio.
- El entorno es estable con poca probabilidad de que ocurran cambios.
- El nivel de riesgo e incertidumbre son bajos debido a que existe un conocimiento previo de lo que se va a realizar. Por ejemplo, cuando el objetivo de proyecto es implementar una mejora o adaptación a un software que ya existe, el nivel de comprensión del proyecto es mayor en comparación a un desarrollo software que empieza 'desde cero'.

[12] El IEEE define la línea de base como una especificación o producto que se revisó formalmente y con el que se estuvo de acuerdo, que a partir de entonces sirve como base para un mayor desarrollo y que puede cambiar sólo a través de procedimientos de control de cambio formal. (IEEE, 1990).

- Cuando se tiene un control suficiente de la información y no se requiere de la participación de terceros. Esto sucede en el caso de que el software resultante será utilizado por el mismo equipo de desarrollo.
- El modelo cascada es ideal para el desarrollo de un sistema riguroso que necesita del cumplimiento ordenado y controlado de todas las actividades, como ser proyectos con requerimientos inflexibles relacionados a la seguridad, fiabilidad o protección.

En el contexto de los proyectos software el modelo cascada presenta muchas desventajas, entre ellas:

- El desarrollo de software no sigue un flujo lineal, es un proceso que necesita de un ciclo de vida iterativo para obtener una retroalimentación continua del trabajo realizado (Royce, 1970, p. 328; Sommerville, 2011, p. 31). La intercalación y superposición de las fases permiten una revisión constante de los hitos. De esta forma, se identifican y corrigen los defectos en una etapa temprana (Fairley, 2009, p. 58).
- No se puede planificar al 100% el proyecto debido a que en la fase de inicio no se conocen todos los requisitos del software. El desarrollo software consiste en la creación de algo nuevo (a esto se debe añadir su naturaleza intangible), por lo tanto es difícil para el cliente o personas interesadas, expresar de forma clara y exacta todos los requisitos del software (Pressman, 2010, p. 34).
- Los requisitos se revisan durante todo el proyecto, siendo probable que sufran alguna modificación. El modelo cascada es rígido, y no se adapta ni a las necesidades cambiantes del cliente (Sommerville, 2011, p. 32) ni al entorno que envuelve la tecnología que también cambia con facilidad y rapidez. El modelo cascada es inflexible, los cambios no son bien recibidos (Murray, 2016, p. 33).
- Los hitos se validan hasta final de la fase. Según el tipo de proyecto, una fase puede durar meses lo que impide que las no conformidades sean detectadas y corregidas con prontitud durante la fase de desarrollo en la que se esté trabajando (Fairley, 2009, p. 58).
- El exceso de 'burocracia', refiriéndonos a la documentación, puede ralentizar la implementación de cambios y el avance en las actividades.
- El cliente no se integra durante el desarrollo de las fases. En el modelo cascada no se planifican demostraciones frecuentes de versiones preliminares del software, al contrario, el cliente debe esperar hasta una etapa muy avanzada para ver el software y validar sus funcionalidades (Pressman, 2010, p. 34). Esto es un problema cuando el cliente identifica no conformidades y el equipo de desarrollo debe regresar a una etapa anterior para corregir los errores (Sommerville, 2011, p. 31; Fairley, 2009, p. 58). Mientras tanto, otras actividades se detienen y producen bloqueos en el proceso; éstos pueden llegar a tener un tiempo de espera mayor al tiempo trabajado (Bradac, et al., 1994).

Se puede decir que Royce (Royce, 1970, p. 329) estuvo muy acertado cuando expuso que la implementación de este modelo en el desarrollo de software era riesgosa y que invitaba al fracaso.

### 2.4.1.2 Modelo V

El modelo de ciclo de vida V se deriva del modelo cascada y al igual que éste, describe de forma secuencial las fases del proyecto. En la figura 2-4 se puede observar que el lado izquierdo de la V está formado por las actividades de: modelado de los requisitos, diseño de la arquitectura, diseño de los componentes y generación del código. En el lado derecho del modelo, se encuentran las actividades de pruebas en las que se verifican y validan los productos de trabajo resultantes.

El modelo V se esfuerza para asegurar la calidad del software. Por ello, se llevan a cabo diferentes pruebas: pruebas unitarias, pruebas de integración, pruebas del sistema y por último las pruebas aceptación. Las pruebas son incluidas en cada fase de desarrollo del lado izquierdo de la V. En estas pruebas se utilizan los hitos como criterio para dar por finalizado una fase y pueda continuar la siguiente. (O'Regan, 2011). No existen grandes diferencias entre el modelo cascada y el modelo V, (Pressman, 2010, p. 34) solo que éste último hace mayor hincapié en la realización de pruebas para verificar la conformidad de los requisitos expresados por el cliente.

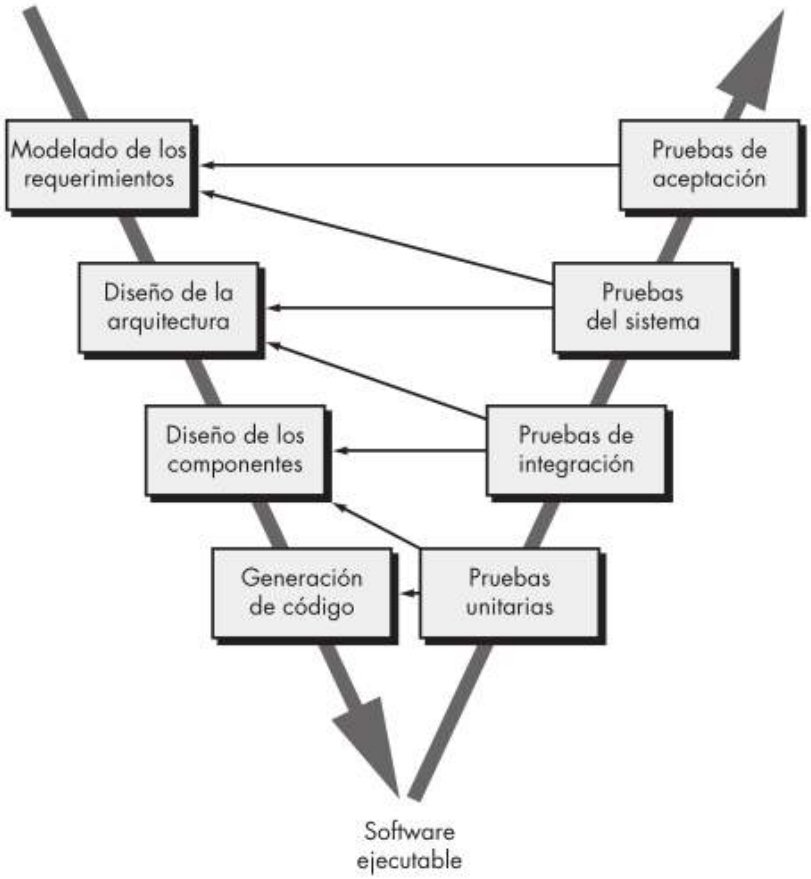


Figura 2-4 El Modelo en V (Pressman, 2010, p. 35).

### 2.4.2 Modelos de Proceso Iterativo

En un modelo de proceso iterativo las funcionalidades del software se agrupan en paquetes pequeños que se desarrollan en ciclos cortos. Durante cada ciclo o iteración el paquete de funcionalidades que se esté desarrollando pasa por las fases de requisitos, diseño, implementación y prueba. Este ciclo se repite las veces que sea necesario hasta completar cada funcionalidad. El proceso iterativo en el desarrollo software permite una mejora continua porque en cada iteración se añaden nuevas funcionalidades y se entrega una versión más completa del software. A continuación se explican algunos modelos iterativos.

#### 2.4.2.1 El modelo de proceso incremental

El desarrollo incremental es "una técnica en que la definición, el diseño, la implementación y las pruebas de los requisitos se producen de manera superpuesta, iterativa (en lugar de secuencial), lo que resulta en una finalización incremental del producto de software en general" (IEEE, 1990, p. 39). Al igual que los modelos prescriptivos, el modelo incremental define las fases de desarrollo software con la diferencia de que las actividades se entrelazan y 'pueden ir hacia adelante y hacia atrás'.

El modelo incremental comienza con el desarrollo de una primera versión del software que contiene las funcionalidades más esenciales y/o urgentes que ha especificado el cliente (Sommerville, 2011, p. 33). Una vez que la primera iteración es completada, se muestra al usuario final para conocer su opinión. El usuario evalúa cada una de las funcionalidades y valida aquellas que han sido desarrolladas correctamente y, manifiesta su no conformidad sobre las funcionalidades que no cumplen con las especificaciones acordadas en la documentación de los requisitos. Durante esta evaluación, el usuario puede solicitar cambios o añadir nuevas funcionalidades que no estaban definidas en el Alcance del proyecto (Pressman, 2010, pp. 35-36). Una vez que el incremento es validado[13] y se obtiene la retroalimentación por parte del usuario, el equipo se planifica el siguiente incremento, que puede consistir en:

- La continuación del plan original.
- La corrección de las no conformidades encontradas en el incremento actual.
- Modificar el software actual para cumplir nuevas necesidades del cliente.
- Añadir al próximo incremento nuevas funcionalidades.

Entre las ventajas del modelo incremental en los proyectos software están:

- Asegura la calidad del software debido a que los productos de trabajo se integran, verifican y

[13] El proceso de desarrollo de cada incremento se da por finalizado cuando el cliente verifica, valida, demuestra y acepta la versión final (Fairley, 2009, p. 60).

validan de forma constante.

- Es un modelo flexible que acepta con naturalidad el cambio y se adapta a los requerimientos cambiantes del cliente (Sommerville, 2011, p. 29). En los proyectos software se asume que en algún momento del proceso de desarrollo se tendrá que corregir el trabajo. Según Fairley, la reelaboración del software es "inevitable".
- La evaluación y retroalimentación continua del software por parte del usuario final permite la puesta en marcha de mejoras desde una etapa muy temprana. El usuario está habilitado para hacer sugerencias en cualquier momento del desarrollo del proyecto (Mohapatra, 2010, p. 26).
- Con la detección temprana de los errores, el nivel de riesgo disminuye y se evitan los costes que se derivan del reproceso. Esto es importante puesto que el coste de corregir errores crece de forma exponencial[14] según en la etapa en que se detectan (Fairley, 2009, p. 59).
- Al hacer entregas del software operativo en periodos cortos, el cliente puede ver los avances del proyecto continuamente. Esto acelera la entrega de valor al cliente y se obtienen beneficios económicos en menor tiempo. Asimismo, la confianza entre ambas partes aumenta y se fortalece la relación contractual.

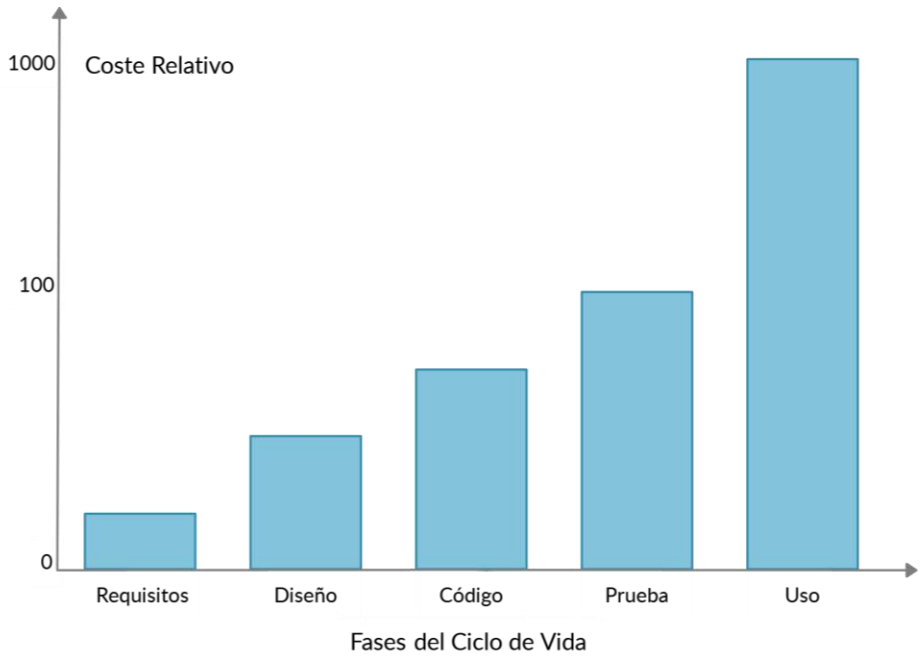


Figura 2-5 Coste relativo de encontrar y corregir un defecto de software (Fairley, 2009, p. 59).

[14] Según Boehm, encontrar y reparar un defecto en los requisitos durante la prueba del sistema puede costar 100 veces más que repararlo durante la fase de requisitos (Boehm, 1981, p. 40).

A pesar de las ventajas del modelo incremental, también presenta ciertos inconvenientes:

- No es recomendable para sistemas grandes y complejos que necesitan de un marco o una arquitectura estable (Sommerville, 2011, p. 34). La estructura del sistema tiende a degradarse conforme se tienen nuevos incrementos.
- Exige un esfuerzo en la planificación continua, debido a que se aceptan implementar cambios durante el desarrollo del software.
- Es difícil estimar el coste total con exactitud, ya que es un modelo flexible que permite realizar cambios durante el proyecto.
- Al inicio del proyecto, el proceso no es visible en su totalidad, lo que puede crear confusiones.

#### 2.4.2.2 El modelo evolutivo

Los proyectos software pueden tener un nivel de riesgo e incertidumbre muy alto, sobre todo cuando se trata del desarrollo de un producto software muy innovador que introduce una nueva tecnología u ofrece una nueva experiencia de usuario. En estos proyectos existe el inconveniente de que tanto los usuarios como el equipo de desarrollo no tienen conocimientos o una experiencia previa, lo que dificulta la elaboración del alcance y la planificación del proyecto en general. El nivel de riesgo de un proyecto también puede verse afectado por los cambios en el mercado o en las necesidades de los usuarios. Frente a estos factores de riesgo e incertidumbre es recomendable que un proyecto software utilice un modelo evolutivo en su proceso de desarrollo, ya que se adapta mejor a un entorno inestable y es adecuado para los procesos relacionados con las actividades de investigación y desarrollo (I+D) de nuevos productos.

Los modelos evolutivos comienzan con una fase experimental en la que pone a prueba las ideas iniciales del proyecto. Un modelo evolutivo sigue un proceso iterativo que inicia con el desarrollo de una primera versión del software que se entrega al usuario y que a través de su experiencia de uso, se obtiene información relevante que servirá para el desarrollo de las siguientes versiones del software. El objetivo de utilizar un modelo evolutivo en el proceso software es obtener los conocimientos suficientes para definir los requisitos de lo que podría ser el software final. Con los resultados obtenidos al final de cada iteración, validación y demostración (Fairley, 2009, p. 65) se espera esclarecer lo siguiente:

- Que el enfoque elegido es el adecuado y se ha generado suficiente información para continuar con el siguiente ciclo de desarrollo (análisis, diseño, implementación y evaluación)
- Los resultados de la iteración no han sido satisfactorios y debe elegirse un nuevo enfoque que se adapte mejor a las necesidades del proyecto.
- La iteración ha generado conocimientos suficientes para identificar los requisitos y definir un diseño del software, con lo cual se decide que el proyecto debe continuar con un enfoque incremental.

- Ha sido demostrada la inviabilidad del proyecto y debe cancelarse.

Entre los modelos evolutivos tenemos el Prototipo y Espiral. **El prototipo** tiene un ciclo de desarrollo iterativo y utiliza el método experimental basado en la prueba y error; es ideal para un proyecto software en el que se desconoce cuál será el resultado final. Los prototipos se caracterizan por tener un proceso de desarrollo rápido en el que se obtiene un software funcional en un ciclo muy corto. El objetivo de un prototipo es entregar lo antes posible, una versión funcional al usuario para que acepte, descarte o añada nuevas funcionalidades.

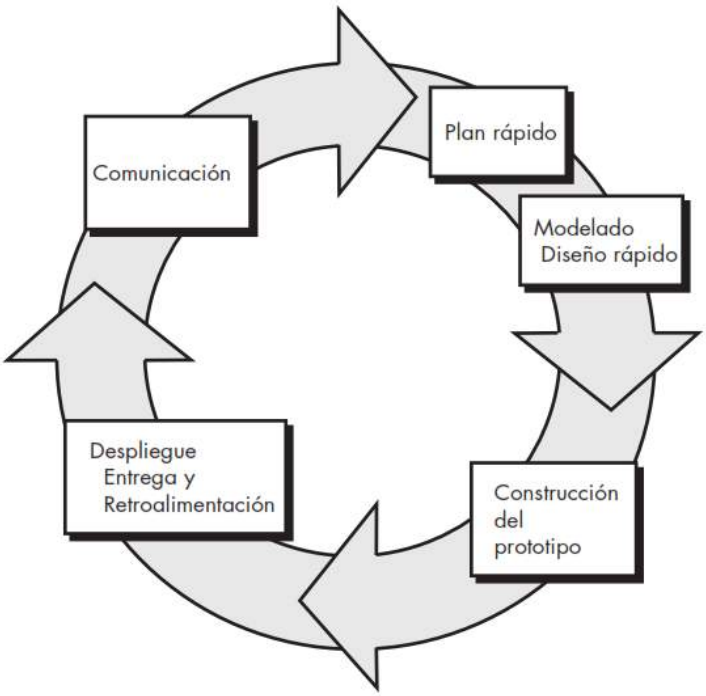


Figura 2-6 El paradigma de hacer prototipos (Pressman, 2010, p. 37).

Los modelos prototipos se pueden clasificar en desechables y evolutivos. El prototipo desechable tiene un ciclo de vida temporal; éste es entregado al usuario final para que realice suficientes pruebas y valide las funcionalidades desarrolladas hasta ese momento. Una vez que el equipo de desarrollo obtiene esta retroalimentación, el prototipo es desechado y se inicia con un nuevo ciclo de desarrollo. En cambio, el prototipo evolutivo sí es utilizado en el proyecto y es mejorado en cada iteración con la adición de nuevos requisitos. Las iteraciones finalizan hasta que se consigue desarrollar el software definitivo (Mohapatra, 2010, p. 27).

El uso de los prototipos en los proyectos software presenta las siguientes ventajas (Hughes & Cotterell, 2009, pp. 85-86; Sommerville, 2011, p. 45) :

- “Se aprende haciendo”, durante el proceso se obtienen conocimientos de mucho valor tanto de los aciertos como de los errores.
- Mejora la comunicación entre el usuario y el equipo de desarrollo. Es más fácil para los usuarios dar su opinión y sugerencias después de haber utilizado el prototipo del software.

- Los usuarios están más involucrados en las decisiones de diseño.
- Ayuda a definir el alcance. La lista preliminar de requisitos cambia y evoluciona a medida que progresa el desarrollo del prototipo. El nivel de detalle de los requisitos aumenta gracias a los conocimientos generados durante el ciclo de desarrollo. También son descubiertos nuevos requisitos y otros son descartados[15].
- Se reduce la documentación. Al tener un prototipo operativo, disminuye la necesidad de especificar detalles.
- Reducción en los costes de mantenimiento. Con el prototipo es posible anticiparse a posibles modificaciones ya que durante las pruebas se revelan los errores u omisiones.
- El avance del software es altamente visible.
- Es muy útil para encontrar soluciones específicas de software.

Algunas de desventajas del uso de los prototipos:

- Los usuarios pueden malinterpretar el papel del prototipo y expresar cierta inconformidad[16]. Alguno de ellos creerán que: se han omitido funcionalidades importantes, no existe suficiente documentación y no se ha considerado "la calidad general del software o la facilidad de darle mantenimiento a largo plazo" (Pressman, 2010, p. 38). Puede perderse el control del desarrollo del prototipo cuando el usuario presiona al equipo del proyecto para que se agreguen nuevas funcionalidades.
- La creación de un prototipo requiere de un gasto adicional en el proyecto. Por ello, se aconseja incluir en el prototipo aquellas funcionalidades que se consideran fundamentales y que formarán parte del software final.

**El Modelo Espiral** fue propuesto por Barry Boehm (como una alternativa al modelo cascada) para mejorar el proceso de desarrollo del software (Boehm, 1988). Tiene un ciclo de vida iterativo y su punto más fuerte es su consideración explícita (identificación y eliminación) de los riesgos del proyecto (Mohapatra, 2010, p. 32). En cada iteración o vuelta a la espiral, se hace un análisis del proyecto para identificar los riesgos potenciales. Luego, se elabora un plan de control de riesgos para disminuir el impacto negativo que podrían tener sobre el proyecto.

Boehm (Boehm, 2001) en su artículo *The Spiral Model as a Tool for Evolutionary Software Acquisition,* describe el modelo espiral como "un generador de modelo de un proceso impulsado

[15] Sommerville explica que "una función descrita en una especificación puede parecer útil y bien definida. Sin embargo, cuando dicha función se combina con otras operaciones, los usuarios descubren frecuentemente que su visión inicial era incorrecta o estaba incompleta" (Sommerville, 2011, p. 45).

[16] Pressman asegura que la clave para conseguir éxito de un prototipo es "definir desde el principio las reglas del juego; es decir, todos los participantes deben estar de acuerdo en que el prototipo sirva como el mecanismo para definir los requerimientos. Después se descartará (al menos en parte) y se hará la ingeniería del software real con la mirada puesta en la calidad (Pressman, 2010, p. 38).

por el riesgo". El autor identifica dos características principales del modelo espiral: (1) tiene un "enfoque cíclico para el crecimiento incremental del grado de definición de un sistema y su implementación, mientras que disminuye su grado de riesgo" y (2) posee "un conjunto de puntos de referencia de anclaje puntual para asegurar el compromiso del participante con soluciones factibles y mutuamente satisfactorias". En la espiral de Boehm se visualizan datos importantes del proyecto (*véase figura2-7*), por ejemplo: el progreso de desarrollo, es decir, el incremento de las funcionalidades se observa en el recorrido que hace la espiral en cada cuadrante; el radio de la espiral representa el coste acumulativo del proyecto; cada 'vuelta´ o ciclo significa que finaliza una fase del proceso software (a medida que se realizan más iteraciones, se aumenta el nivel de detalle); los cuadrantes de la espiral son el conjunto de actividades y tareas que deben ser realizadas en un momento específico del proceso de desarrollo (Mohapatra, 2010, p. 32).

El modelo espiral propuesto por Boehm se divide en cuatro sectores que representan las actividades que se realizan durante el proceso de desarrollo software (Sommerville, 2011, pp. 49-50):

1. Determinar objetivos, alternativas y restricciones. Los riesgos son identificados y se elabora un plan detallado para contrarrestarlos.
2. Evaluar alternativas, identificar y resolver riesgos. Después de un análisis, se establecen estrategias para cumplir con los objetivos y controlar los riesgos identificados.
3. Desarrollar, verificar producto del siguiente nivel. Se elige un modelo de proceso adecuado para el desarrollo del software (Prototipo, cascada, etc.) que mejor se adapte a las restricciones y riesgos del proyecto. Los resultados se someten a pruebas de verificación y validación (Mohapatra, 2010, p. 32).
4. Plan de la siguiente fase. Los resultados del ciclo finalizado son evaluados y se analiza el progreso del proyecto. Aquí se toma la decisión sobre sí el proceso de desarrollo del software ha terminado o, si es necesario realizar otro ciclo en la espiral. De ser esta última, se inicia con la planificación de la siguiente iteración.

Algunas ventajas de la espiral de Boehm en el proceso software:

- Adecuado para proyectos software complejos y de gran escala.
- Se adapta a las necesidades del usuario gracias a su enfoque iterativo. Es un proceso flexible que admite añadir, modificar o quitar funcionalidades.
- Es un enfoque que puede aplicarse a lo largo de la vida del software. El modelo en espiral permite la construcción de un prototipo en cualquier etapa del proyecto con el fin de reducir riesgos.
- Debido al reconocimiento explícito de los riesgos, tanto el equipo de desarrollo como el cliente comprenden y reaccionan de mejor forma ante los riesgos en cada ciclo del proyecto (Pressman, 2010, p. 40).

Entre sus desventajas se pueden mencionar:

- Dificultad en el control del alcance, tiempo y costes del proyecto. Los ciclos de la espiral se podrían extender más de lo que se espera.
- Los clientes podrían tener dudas sobre la seguridad de mantener el control del proyecto y alcanzar el éxito con la implementación de un enfoque evolutivo. El modelo en espiral es un muy complejo de utilizar, por lo que se requiere de personas calificadas y con mucha experiencia para la evaluación correcta de los riesgos (Pressman, 2010, p. 40) (Pressman, 2010, p. 40).

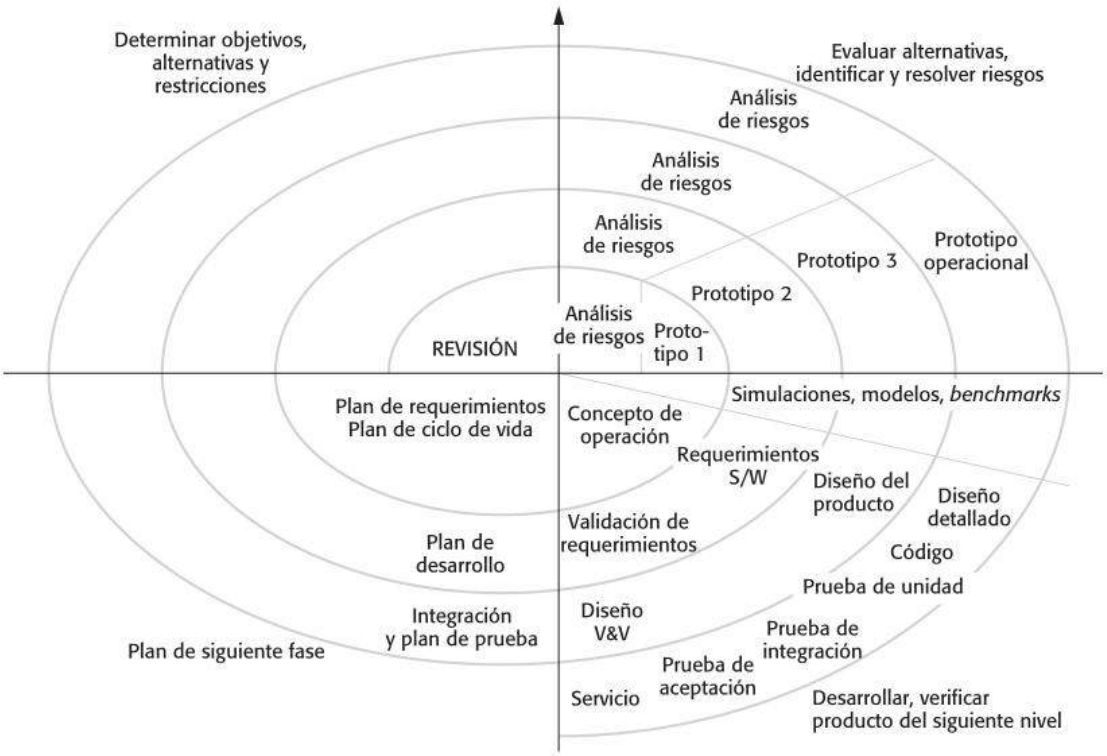


Figura 2-7 Modelo en espiral de Boehm del proceso de software (Sommerville, 2011, p. 49).

### 2.4.2.3 Proceso Unificado Racional (RUP)

La empresa Rational Software, actualmente propiedad de IBM, desarrolló un marco de trabajo basado en el Proceso Unificado (PU), llamado Rational Unified Process (RUP). El PU es un proceso de desarrollo software que está "dirigido por casos de uso, centrado en la arquitectura, e iterativo e incremental" (Jacobson, et al., 2000, p. 4). Se le llama proceso unificado porque une elementos de diferentes modelos de proceso (Sommerville, 2011, p. 50) con el "objetivo de asegurar la producción de software de alta calidad" (Kruchten, 2000). El PU utiliza el lenguaje Unificado de Modelado (Unified Model Language, UML) para representar de forma gráfica un sistema software. El UML "es un lenguaje estándar para el modelado de software – lenguaje para visualizar, especificar, construir y documentar los artefactos de un sistema con gran cantidad de software;

permite a los desarrolladores visualizar el producto de su trabajo (artefactos) en esquemas o diagramas estandarizados" (Jacobson, et al., 2000, p. 430).

Como se ha mencionado antes, el modelo PU tiene las siguientes características:

- Guiado por casos de uso: Un caso de uso es "un fragmento de funcionalidad del sistema que proporciona al usuario un resultado importante. Los casos de uso representan los requisitos funcionales" (Jacobson, et al., 2000, p. 5). El caso de uso es una forma simplificada de identificar las entidades que participarán en el sistema software y las interacciones entre ellos. En RUP son representados a través de modelos gráficos basados en UML.
- Centrado en la arquitectura: El proceso se enfoca en el desarrollo temprano del software y sobre todo, pone énfasis en crear una línea base de arquitectura ejecutable (IBM, 2003, p. 2) La arquitectura involucra los elementos más significativos del sistema; "es una vista del diseño completo con las características más resaltadas" (Jacobson, et al., 2000, p. 6).
- Es iterativo: los casos de uso se agrupan en paquetes pequeños de trabajo y se desarrollan de forma iterativa. En cada ciclo, los casos de uso pasan por las fases del proceso de desarrollo software (Análisis, Diseño, Implementación y Prueba) hasta obtener un software funcional.

A diferencia de otros modelos de proceso software, el RUP se puede describir desde tres perspectivas:

1. Es dinámica porque muestra la iteración de las fases del modelo (Concepción, elaboración, construcción y transición) durante el tiempo de desarrollo (Sommerville, 2011, p. 49).
2. Es estática, formada por cuatro elementos: trabajadores, actividades, artefactos y los flujos de trabajo. En el artículo "Rational Unified Process: Best Practices for Software Development Teams" (IBM, 2003, pp. 7-9) los elementos del RUP se explican de la siguiente forma:
    - El trabajador "define el comportamiento y las responsabilidades de un individuo o grupo de individuos que trabajan juntos como un equipo.
    - La actividad "es una unidad de trabajo que se le puede pedir a una persona que desempeñe ese rol y que produce un resultado significativo en el contexto del proyecto".
    - El artefacto "es una información que es producida, modificada o utilizada por un proceso".
    - El flujo de trabajo "es una secuencia de actividades que produce un resultado de valor observable"[17]. En RUP se definen nueve flujos (seis flujos de trabajo centrales y tres flujos

[17] En RUP, los flujos de trabajo no están asociados a ninguna fase, por el contrario, pueden presentarse en cada iteración, en diferentes fases, Modelado de negocios, Requisitos, Diseño, Implementación, Prueba, Implementación, Configuración, Gestión de cambios y Gestión de proyectos una o más pueden ocurrir dentro de cada iteración.

de trabajo de apoyo).

3. Es práctica porque sugiere buenas prácticas que deben usarse durante el proceso como ser: El desarrollo de software de manera iterativa, la gestión de requerimientos, el uso de una arquitectura basada en componentes, el software modelado visualmente, y la verificación de la calidad de software y control de los cambios de software (IBM, 2003, p. 1)

El RUP está compuesto por cuatro fases:

1. **Fase de concepción**: el cliente define las necesidades del negocio que debe satisfacer el software y se utilizan los casos de uso para describirlas. También es importante identificar "las entidades externas (personas y sistemas) que interactuarán con el sistema y definirán dichas interacciones" (Sommerville, 2011, pp. 50-51). En esta fase se realiza una evaluación sobre la viabilidad del proyecto (Mohapatra, 2010, p. 50), se establece una arquitectura preliminar y se elabora el plan del proyecto (Pressman, 2010, p. 47).
2. **Fase de elaboración**: Se añaden nuevos requisitos, es decir, aumentan los casos de uso y sus detalles. El diseño de la arquitectura es definido e incluye cinco vistas arquitectónicas del software que son los modelos de: casos de uso, análisis, diseño, implementación y despliegue. En esta fase se obtiene una línea base de la arquitectura (Jacobson, et al., 2000, p. 11) y se revisa el plan para el desarrollo software en el que se examina nuevamente el alcance con los últimos requisitos añadidos. Con esta información se estiman los recursos necesarios, se evalúan los riesgos más críticos del proyecto y se determina la fecha de entrega. En resumen, en esta fase debe confirmarse que el proyecto sigue siendo viable (Mohapatra, 2010, p. 61; Pressman, 2010, p. 47; Sommerville, 2011, p. 51).
3. **Fase construcción:** los componentes de software son desarrollados en cada iteración. Los modelos de requerimientos y diseño se completan. "La línea base de la arquitectura crece hasta convertirse en el sistema completo" (Jacobson, et al., 2000, p. 12). Esta fase incluye las actividades de codificación, pruebas unitarias, integración, es decir, se llevan a cabo todas las acciones necesarias para tener un software operativo y documentado (incluye manuales para entregar al usuario).
4. **Fase de Transición:** se realiza la entrega del software al usuario final para que realice las pruebas de validación de las funcionalidades e identifique las no conformidades del software. En esta fase de pruebas, se pueden solicitar cambios o la adición de nuevos requisitos. El equipo de desarrollo corrige los defectos e implementan las mejoras que han sido propuestas durante las pruebas. En esta fase también se incluyen las actividades de capacitación, soporte y mantenimiento del producto software y finaliza cuando los usuarios están satisfecho con el software (Kruchten, 2000).

Entre las ventajas de ventajas de RUP se pueden mencionar:

- La participación del usuario durante todo el proceso ayuda a disminuir los errores durante el desarrollo.
- La definición de los roles y asignación de las responsabilidades facilita la organización y el

control de las actividades del proyecto.

- Asegura la calidad del software y disminuye el nivel de riesgo gracias al ciclo de desarrollo iterativo.
- Es flexible ya que admite cambios durante el proceso de desarrollo. El proceso unificado "reconoce una realidad que a menudo se ignora – que las necesidades del usuario y sus correspondientes requisitos no pueden definirse completamente al principio" (Jacobson, et al., 2000, p. 7)
- Valora la comunicación con el cliente y utiliza métodos efectivos (casos de uso) para describir sus necesidades con respecto al software que se va a desarrollar (Pressman, 2010, p. 46). El empleo de los casos de uso para definir los requisitos del proyecto, mejora la comunicación y aumenta la comprensión tanto de los usuarios como del equipo de desarrollo
- Su énfasis en la arquitectura del software "ayuda a tener un control intelectual del proyecto, facilita la reutilización a gran escala y proporciona una base para la gestión de proyectos" (Kruchten, 2000).
- El modelo iterativo e incremental propuesto se puede adaptar a proyectos grandes y pequeños y satisface las necesidades concretas de cada proyecto (Mohapatra, 2010, p. 49; Pressman, 2010, p. 46).

RUP tiene algunas dificultades en los proyectos software:

- Aunque es un proceso que puede adaptarse a las necesidades específicas de un proyecto, tiene un grado de complejidad bastante alto y es muy exhaustivo y denso para ser utilizado en proyectos pequeños. Demanda una inversión de tiempo y esfuerzo en la organización del trabajo y documentación por consiguiente genera costes adicionales.
- Es necesario personas calificadas, con suficientes conocimientos y experiencia de RUP para lograr con éxito una adaptación del proceso a las necesidades del proyecto (Hanssen, et al., 2007). Según algunos autores (Hanssen, et al., 2005) es recomendable una formación previa de RUP para el equipo de desarrollo ya que consideran que "mejoraría el grado de uso y el efecto del uso de RUP y también serviría como un mecanismo de transferencia de experiencia".

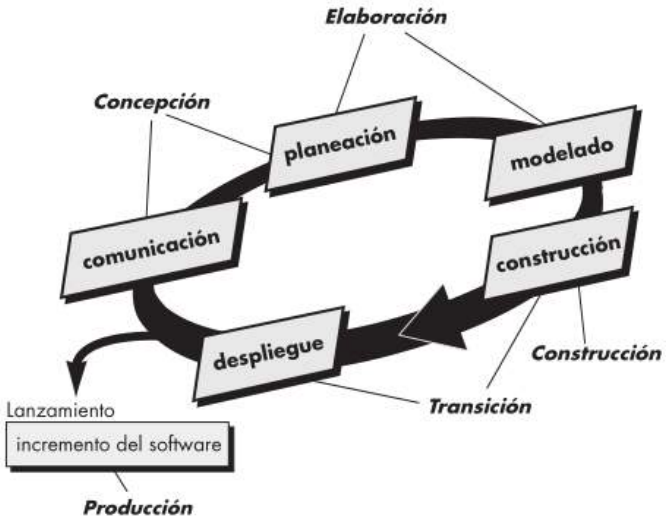


Figura 2-8 Proceso Unificado (Pressman, 2010, p. 47).

## 2.5 Situación actual de la Gestión de Proyectos Software

Hoy en día los directores de proyectos software afrontan a grandes desafíos que afectan la eficacia y eficiencia en sus tareas de gestión. Por un lado la complejidad de los sistemas de software ha ido en aumento debido al crecimiento de su contenido, tamaño y a su nivel de interacción con otros sistemas (Ruhe & Wohlin, 2014, p. 2). Por otra parte, el software debe satisfacer a los usuarios que demandan de forma creciente e interminable, nuevas funcionalidades que deben ser entregadas con rapidez. Esta complejidad aumenta en aquellos proyectos software que "a menudo implican cuestiones de seguridad, protección, fiabilidad y otros requisitos de calidad" (PMI, 2013b, p. 3).

Desde 1994, la compañía Standish Group se ha dedicado al estudio del estado de la industria del desarrollo software. Sus investigaciones son publicadas anualmente en el CHAOS Report y muestran estadísticas de los resultados de los proyectos software. En este informe, los proyectos software se clasifican, según sus resultados, en tres categorías:

- **Exitoso** El proyecto es completado a tiempo y dentro del presupuesto, con todas las características y funciones especificadas originalmente.
- **Desafiado** El proyecto está terminado y operacional, pero por encima del presupuesto, el tiempo estimado, y con pocas características y funciones especificadas inicialmente.
- **Fracasado** El proyecto es cancelado antes de ser terminado o nunca fue implementado.

En la figura 2-8 se puede observar un resumen de los CHAOS Report que muestran los resultados de más de 50.000 proyectos software en los últimos 25 años. Aunque esta información refleja una tendencia positiva en la tasa de éxito de los proyectos software, el porcentaje de los proyectos que tuvieron problemas y fueron canceladon sigue siendo muy alto. Una muestra de ellos, ha sido el

último informe “CHAOS 2020 Beyond Infinity”,en el que solo el 35% de los proyectos software fueron exitosos, un 47% tuvieron dificultades y un 19 % fracasaron.

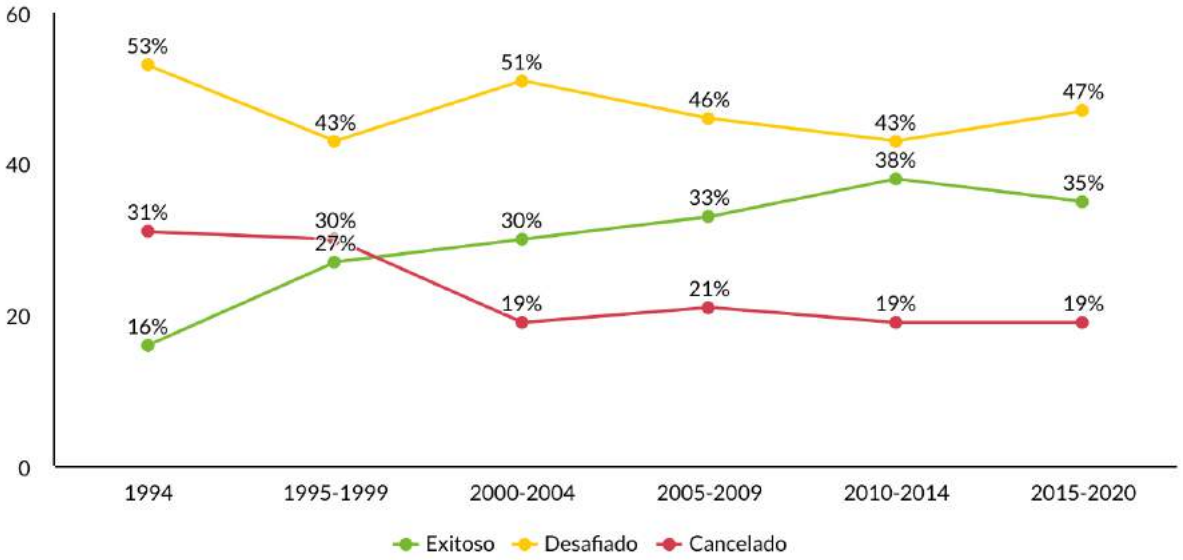


Figura 2-9 Investigación Longitudinal de los CHAOS Report

Uno de los objetivos de la compañía Standish Group es ofrecer a la comunidad del software información relevante que ayude a mejorar el estado actual de los proyectos. A través de sus informes CHAOS, se han identificado factores que predominan en el éxito o fracaso de los proyectos software (*Véase tabla 2-1).* Entre ellos destacan: el apoyo por parte de la dirección (tanto financiero como emocional), la madurez emocional del equipo de desarrollo y la implicación del usuario en el proyecto. Referente a los factores que generan problemas en los proyectos software se mencionan: los requisitos incompletos, la falta de participación de los usuarios y la falta de recursos.

El CHAOS Report también indica otras dos variables que influyen en la resolución de un proyecto software: su tamaño y el método de desarrollo del software utilizado. El tamaño de un proyecto es considerado por el CHAOS Report como el factor más importante en la resolución de un proyecto software. De forma lógica se comprende que un proyecto de gran escala es más complejo de gestionar debido a “la cantidad y variedad de elementos y la interconexión entre los elementos” del proyecto (Shenhar, et al., 2002). Los proyectos grandes tienen un nivel de riesgo mayor y una planificación más compleja en comparación a un proyecto pequeño que es más fácil de organizar, dirigir y controlar. El otro factor clave es la elección del método de desarrollo del software. En el CHAOS Report se hace una comparativa entre los proyectos que utilizaron un método predictivo y estructurado como ser el Cascada y, aquellos que implementaron un método más flexible e iterativo como el método Ágil.

| Factores de Éxito | Factores de Fracaso |
|---|---|
| • Apoyo de la dirección<br>• Madurez emocional<br>• Implicación del usuario<br>• Optimización<br>• Personal calificado<br>• Arquitectura estándar<br>• Proceso ágil<br>• Ejecución moderada y limitada<br>• Experiencia en gestión de proyectos<br>• Objetivos de negocios claros | • Requisitos incompletos<br>• Falta de participación de los usuarios<br>• Falta de recursos<br>• Expectativas poco realistas<br>• Falta de apoyo ejecutivo<br>• Cambio de requisitos y especificaciones<br>• Falta de planificación<br>• Ya no se necesitaba más<br>• Falta de gestión de TI<br>• Desconocimiento de la tecnología |

Tabla 2-1 Información adaptada del CHAOS Report 2015 (Standish Group, 2015).

Los métodos Ágiles "son métodos de desarrollo de software que se combinan para una entrega rápida del software. El software se desarrolla y se entrega en incrementos, y se minimizan la documentación del proceso y la burocracia. El foco del desarrollo está en el código en sí, y no en los documentos de apoyo" (Sommerville, 2011, p. 741). Según la tabla 2-2, los proyectos que tuvieron una mayor tasa de éxito fueron los que utilizaron un método ágil para el desarrollo del software.

| Tamaño | Método | Exitoso | Desafiado | Fracasado |
|---|---|---|---|---|
| **Todos los proyectos** | Ágil | 42% | 47% | 11% |
| | Cascada | 13% | 59% | 28% |
| **Proyectos de tamaño grande** | Ágil | 19% | 56% | 25% |
| | Cascada | 8% | 56% | 36% |
| **Proyectos de tamaño medio** | Ágil | 34% | 53% | 13% |
| | Cascada | 9% | 66% | 25% |
| **Proyectos de tamaño pequeño** | Ágil | 59% | 36% | 5% |
| | Cascada | 45% | 46% | 9% |

Tabla 2-2 La resolución de todos los proyectos de software de 1994-2020, segmentada por el método ágil y el método de cascada.

Como se ha mencionado antes, los modelos predictivos tienen un enfoque muy estructurado que no se adaptan a las necesidades de los proyectos software. Con respecto a esto, Highsmith (Highsmith, 2010, p. 9) asegura que "el pensamiento lineal, los procesos prescriptivos y las prácticas estandarizadas e invariables no son rival para el entorno volátil del actual desarrollo de productos". El uso de un método ágil garantiza una entrega temprana de valor gracias a su proceso incremental e iterativo. Desde la primera entrega se obtiene un producto funcional que aporta información valiosa tanto para el cliente como para el equipo del proyecto. Las metodologías ágiles son una respuesta a las organizaciones que necesitan desarrollar productos innovadores y lanzarlos al mercado lo más pronto posible. Estas ventajas y muchas otras son la razón por la cual estos métodos se han popularizado en las últimas dos décadas ya no solamente en la Ingeniería del Software sino también en otras disciplinas.

Con esta información se ha podido demostrar que los métodos predictivos no son lo suficientemente adecuados para ser utilizados en el desarrollo de software. Tienen elementos que limitan y dificultan el camino para conseguir el éxito del proyecto. En cambio, los métodos ágiles se adaptan de mejor a las necesidades particulares de los proyectos software y ofrecen ventajas competitivas gracias a prácticas de mejora continua y a su enfoque en la entrega temprana de valor al cliente. En el siguiente capítulo se realiza una explicación más detallada acerca del movimiento Ágil y de las metodologías ágiles que más se utilizan en la actualidad.

# 3 ESTADO DEL ARTE

## 3.1 Introducción a la Agilidad

La palabra agilidad es sinónima de moverse con rapidez, de pensar y actuar con prontitud. Ser ágil es tener la habilidad de estar siempre en alerta, con actitud proactiva para actuar ante situaciones adversas e inesperadas. En la Gestión de Proyectos, Highsmith (Highsmith, 2010, p. 13) se refiere a la agilidad como la "capacidad tanto de crear como de responder al cambio con el fin de obtener beneficios en un entorno empresarial turbulento".

Los inicios de la agilidad en la Gestión de Proyectos Software se remontan a la década de los 90's cuando el mundo experimentaba cambios acelerados y sustanciales a consecuencia del desarrollo de nuevas tecnologías y a la transferencia de la información gracias al uso del internet. Es una época que da comienzo a una transformación digital en la que el software es el elemento principal de los sistemas digitales que se estaban desarrollando (Broy, 2018). Esto generó un aumento en la demanda de los proyectos software. En ese momento era claro que los métodos de desarrollo predictivos muy utilizados en los proyectos software, no se adaptaban a la volatilidad e incertidumbre del entorno. Más bien, generaban dificultades y ralentizaban el avance del proyecto debido a: una planificación densa y exhaustiva, comunicación pobre e ineficiente y la entrega de "todo a la vez", refiriéndose a las entregas de software realizadas hasta finalizar por completo una fase (Cooke, 2016, p. 19). Esta situación llevó a los ingenieros de software a considerar la introducción de cambios en los métodos de trabajo con el fin de satisfacer las nuevas necesidades del mercado: entregas rápidas y adaptación constante.

En el 2001, un grupo de profesionales de la Ingeniería de Software se reúnen con el objetivo de proponer nuevas ideas y crear alternativas para mejorar el proceso de desarrollo software. Este grupo de pensadores se autonombró como "The Agile Alliance" y fueron los responsables de dar inicio al Movimiento Ágil que hoy conocemos. En esa reunión se acuñó por primera vez el término de Metodologías Ágiles para nombrar a los nuevos métodos, más ligeros y flexibles, que comenzaban a emerger en contraposición a los métodos predictivos (Beck, et al., 2001).

## 3.2 Manifiesto por el Desarrollo Ágil de Software

El grupo de "The Agile Alliance" creó el Manifiesto Ágil como una declaración de los valores y principios en los que se fundamenta la agilidad. En este manifiesto se expresa lo siguiente:

*"Estamos descubriendo formas mejores de desarrollar software tanto por nuestra propia experiencia como ayudando a terceros. A través de este trabajo hemos aprendido a valorar:*

*Individuos e interacciones sobre procesos y herramientas,*
*Software funcionando sobre documentación extensiva,*
*Colaboración con el cliente sobre negociación contractual,*

*Respuesta ante el cambio sobre seguir un plan.*

*Esto es, aunque valoramos los elementos de la derecha, valoramos más los de la izquierda"* (Beck, et al., 2001).

1. **Individuos e interacciones sobre procesos y herramientas:** esta frase resalta lo más importante en el proceso de desarrollo software: las personas. Como se ha dicho en el capítulo anterior, el software es el resultado del conocimiento generado durante el proceso creativo del desarrollo software. Los miembros del equipo comparten ideas y propuestas innovadoras para dar soluciones únicas a problemas complejos. Para ello, las metodologías ágiles proponen la creación de un ambiente de trabajo colaborativo con canales de comunicación adecuados para fomentar la participación de las personas y el trabajo en equipo. Esto contribuye enormemente al proceso de desarrollo del software ya que las personas se sienten integradas y libres para expresar sus ideas. Dicho lo anterior, el Manifiesto Ágil no quiere menospreciar la importancia que tienen los procesos y herramientas en el desarrollo software. Al contrario, la Ingeniería de Software reconoce el valor que aportan los procesos puesto que ayudan a organizar las actividades, ofrecen una visión global del trabajo y dan sentido de dirección al proyecto. De igual manera, el uso de herramientas ayuda a las personas a optimizar su tiempo y esfuerzo en muchas actividades importantes del proceso como el modelado del software, depuración del código, creación de la documentación, entre otras.

   En resumen, el uso de procesos y herramientas ayudan a las personas a ser más eficientes, pero no aseguran el éxito de un proyecto. En cambio, las personas, sus habilidades, conocimientos y sobre todo, el trabajo que en equipo, son determinantes para cumplir con los objetivos del proyecto.

2. **Software funcionando sobre documentación extensiva**: en este apartado, el valor prevalece en las actividades que están directamente ligadas al trabajo que realiza el desarrollador para obtener un software funcional. Lo más importante es asegurar la entrega de valor al cliente a través de un software que pueda utilizar. El manifiesto ágil promueve la entrega temprana y frecuente del software que permite al cliente observar el avance del proyecto y verificar el cumplimiento de los requisitos con frecuencia. A su vez, el equipo de desarrollo obtiene una retroalimentación constante de su trabajo. La comunicación y la información que se genera en esta interacción es lo que realmente aporta valor al software; ayuda a que ambas partes, el cliente y el equipo de desarrollo, tengan una mayor comprensión sobre el software en desarrollo.

   Acerca de la documentación, lo que destaca el Manifiesto Ágil es que el equipo debe valor que información es relevante para que sea debidamente documentada. En este punto, es conveniente recordar que los proyectos software son conocidos por sus retrasos en las fechas de entrega, por lo tanto, es clave identificar las tareas que aportan valor al producto final para darles prioridad y, descartar aquellas suponen un desperdicio de recursos. Sobre esto, Highsmith (Beck, et al., 2001)fue muy claro cuando expresó su punto de vista: "Aceptamos la documentación, pero no cientos de páginas de tomos que nunca se mantienen y que rara vez se usan". Por ejemplo, existe documentación que es obligatoria, ya sea porque se ha estipulado

en el contrato o lo exija alguna ley o normativa (Palacio, 2020, p. 13). También cabe destacar la importancia de registrar el trabajo que han realizado los miembros del equipo, puesto que permite hacer un seguimiento del proyecto y conocer el estado de las tareas. Documentar los conocimientos obtenidos durante el proceso de desarrollo facilita que éstos puedan ser compartidos y utilizados por otras personas.

Las metodologías ágiles reconocen la importancia de crear la documentación del proyecto, pero hacen hincapié en que el esfuerzo del equipo debe estar enfocado siempre en el desarrollo de un software que funcione y satisfaga las necesidades del cliente.

3. **Colaboración con el cliente sobre negociación contractual:** Una realidad en el desarrollo software es la necesidad de tener un ciclo de vida incremental e iterativo. Debido a la naturaleza abstracta e intangible del software no es posible que el cliente especifique completamente los requisitos al inicio del proyecto. La lista de requisitos evoluciona durante el proyecto según varían las necesidades del cliente. Por esta razón, se recomienda que en un proyecto software no se establezca un contrato cerrado puesto que limita al cliente la posibilidad para solicitar cambios durante el proyecto. En su lugar, se aconseja incluir cláusulas específicas de Cambios de Requisitos. De esta forma se define un procedimiento flexible y adaptativo que permite al cliente formalizar sus solicitudes de cambio. Aquí se recuerda el primer valor expresado en el Manifiesto Ágil: la importancia de las personas y sus interacciones. La continua comunicación con el cliente, su implicación y colaboración en el proyecto son factores importantes que influyen en el éxito del proyecto.

   En el desarrollo ágil el cliente no es solamente un espectador sino un miembro más del equipo del proyecto. La participación del cliente, sus ideas y cualquier otra contribución que realice durante el proceso de desarrollo, aportará más valor que lo definido al inicio cuando no tenía certeza del alcance total.

4. **Respuesta ante el cambio sobre seguir un plan:** este valor se fundamenta en la aceptación de que el cambio es un factor inherente en un proyecto software. El equipo de desarrollo asume que el cliente no puede definir desde el inicio todos los requisitos del software y que es inevitable que realice peticiones para añadir, modificar y/o eliminar requisitos. A diferencia de otro tipo de proyectos cuyo éxito depende del cumplimiento del plan, en los proyectos software recae en la capacidad y velocidad de adaptar el plan a los cambios solicitados por el cliente. En un desarrollo ágil, el plan de trabajo es revisado y actualizado en cada iteración.

## 3.3 Principios del Manifiesto Ágil

Además de los valores descritos anteriormente, el Manifiesto Ágil establece doce principios que caracterizan y marcan la diferencia entre una metodología ágil y una tradicional, y son:

1. **“Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor”.** El enfoque ágil fomenta que en ciclos cortos el cliente reciba un software

funcional que puede utilizar. El cliente queda satisfecho al ver que en poco tiempo tiene un retorno temprano de su inversión. Por otro lado, el equipo de desarrollo también se beneficia al recibir una retroalimentación constante de su trabajo. Las entregas frecuentes fomentan la confianza entre el cliente y el equipo de desarrollo. Otra ventaja de la entrega temprana y continua del software es que muestra de forma muy clara el avance del proyecto, ya que cada entrega significa un incremento del software, es decir, una parte del software que ya está terminada y operativa (Cobb, 2015, p. 25).

2. **"Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos Ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente".** Un proyecto ágil es adaptativo y flexible. Los cambios son bien recibidos porque representan una mejora competitiva al estado actual del proyecto. La lista de requisitos es dinámica y evoluciona; se añaden, modifican o eliminar requisitos con el único objetivo de maximizar la entrega del valor al cliente. Es importante que la lista de requisitos exprese lo que en realidad el cliente espera recibir. Por eso, durante el proyecto se hacen todos los ajustes y refinamientos que sean necesarios. En este proceso, es importante que haya un trabajo colaborativo por parte del cliente y el equipo de desarrollo.
3. **"Entregamos software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible".** Un proyecto ágil tiene un ciclo iterativo, con entregas de pequeños grupos de funcionalidades conocidos como incrementos. Se procura que cada iteración tenga una duración muy corta, preferiblemente que no exceda de un mes para facilitar que el proceso de aprendizaje sea rápido (Cobb, 2015, p. 26). Con cada entrega, tanto el cliente como el equipo de desarrollo tienen una mejor comprensión sobre el software que se está desarrollando, las ideas son más claras, los riesgos disminuyen y el proyecto avanza con mayor rapidez y seguridad.
4. "**Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto".** El éxito del proyecto depende del trabajo colaborativo entre los miembros del equipo del proyecto. Los equipos ágiles mantienen una comunicación constante y directa, y trabajan de forma conjunta durante todo el proyecto. Trabajar en equipo incrementa la calidad del proceso de desarrollo de software, porque fomenta la participación de las personas y el intercambio de conocimiento.
5. **"Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo".** Un equipo de trabajo motivado es un equipo de alto rendimiento, muy productivo y con espíritu de cooperación. El enfoque ágil fomenta a que el equipo se empodere del proyecto y así aumentar su nivel de compromiso y responsabilidad. Por esa razón, el director del proyecto delega liderazgo a los miembros del equipo incentivando la autonomía y libertad para tomar decisiones sobre cómo gestionar su trabajo.
6. **"El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara"**. Como se ha mencionado antes, las metodologías

ágiles fomentan la creación de canales de comunicación adecuados para que garantice que la información fluya de forma clara y rápida en el equipo del proyecto. Lo ideal y recomendable es que la comunicación entre los miembros del equipo se realice de "cara a cara" ya que a través de una conversación directa se transmite información de manera verbal como no verbal. Por ejemplo, la entonación de la voz o el lenguaje corporal pueden transmitir una emoción que difícilmente se puedo percibir leyendo un documento. Las conversaciones directas ayudan a esclarecer asuntos de forma rápida y eficaz y a comprender mejor la información que se quiere transmitir.

7. **"El software funcionando es la medida principal de progreso".** El progreso de un proyecto puede medirse de varias formas, una de ellas corresponde a los incrementos que se entregan en cada iteración. Cada incremento corresponde a una parte del software que ya está terminada, es decir, que un grupo de funcionalidades ya están desarrolladas, entregadas y validadas por lo que es fácil determinar el trabajo que queda pendiente.
8. **"Los procesos Ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma indefinida".** Cuando se habla de un desarrollo sostenible, se refiere a mantener un ritmo de trabajo que sea adecuado a las necesidades del proyecto pero sobre todo que sea acorde a las capacidades del equipo de desarrollo. Esto quiere decir, que en ningún caso el ritmo de trabajo debería ser tan lento como para no llegar a completar las tareas en el tiempo acordado, ni excesivamente rápido para que los miembros del equipo sufran estrés. Las estimaciones de tiempo y esfuerzo deben ser lo más realista posibles. Trabajar bajo mucha presión genera desgaste físico y mental que puede provocar bajas y rotación del personal, algo que afectaría negativamente el avance del proyecto. Además, repercute en la calidad del producto y en el presupuesto del proyecto ya que cuando una persona se retira, se lleva consigo todo el conocimiento y aprendizaje del proyecto. Esto implica contratar a otra persona, invertir tiempo y dinero en capacitarla (Measey, 2015, pp. 8-9).
9. **"La atención continua a la excelencia técnica y al buen diseño mejora la Agilidad".** Un equipo de desarrollo que no se esfuerza en aplicar buenas prácticas ni en seleccionar un diseño y arquitectura eficientes genera como resultado un producto frágil con deuda técnica y dificultades para el mantenimiento (Measey, 2015, p. 9). Resolver estos errores tienen un coste alto y directamente proporcional a la fase del proyecto en la que se esté trabajando. Este principio anima a que el trabajo se realice de manera correcta, siguiendo estándares de codificación desde el inicio para evitar repetir trabajo de forma innecesaria (Cobb, 2015, p. 29).
10. **"La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial".** El desarrollo de una funcionalidad debe basarse en cumplir con las expectativas del cliente (Measey, 2015, p. 9). Una de las ventajas del desarrollo incremental e iterativo es la oportunidad de realizar mejoras en cada iteración. En el caso que una funcionalidad necesite ser modificada, se planifica para que en una próxima iteración sea implementada (Cobb, 2015, p. 29). En otras palabras, el equipo debe enfocarse en un desarrollo simple y permitir que el cliente decida si el trabajo realizado cumple con sus requisitos, evitando invertir tiempo en un desarrollo

innecesario.

11. **“Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados”.** Un equipo capacitado trabajando de forma colaborativa por lo general ofrece mejores soluciones en comparación a personas que trabajen de forma aislada y no consideran la opinión del equipo (Cobb, 2015, p. 30).
12. **“A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia”.** Un proceso de desarrollo ágil se basa en el aprendizaje y la mejora continua. Los ciclos cortos de entrega permiten al equipo del proyecto revisar lo que se ha hecho y cómo se ha hecho. Esta inspección del trabajo ayuda a identificar las áreas del proceso que son necesarias mejorar para implementar acciones correctoras maximizando la entrega del valor al cliente.

## 3.4 Metodologías Ágiles

Dentro del marco de la Agilidad existen diferentes metodologías que siguen procedimientos específicos de trabajo. Aunque cada una posee elementos que las distingue de las demás, todas las comparten ciertas características (Lindvall, et al., 2002, p. 207):

- **Tienen un proceso de desarrollo iterativo e incremental**. Lo más común y recomendable es que cada iteración tenga una duración lo más corto posible (entre 2 a 4 semanas). Al final de cada iteración se entrega un incremento del software.
- **El equipo de desarrollo tiene autonomía para organizar su trabajo.** Los equipos ágiles son autónomos y colaborativos, además, comparten las decisiones y la responsabilidad del trabajo.
- **Aceptan la introducción de nuevos requisitos y tecnologías durante el ciclo de desarrollo**. Los cambios son bien recibidos siempre y cuando aporten valor.

Este trabajo se enfoca en el estudio de las seis metodologías ágiles más populares según las encuestas realizadas en el 2020 por VersionOne y Coding Sans:

- Scrum
- Kanban
- Extreme Programming
- Agile Modeling
- Lean Developmemt
- Scrumban

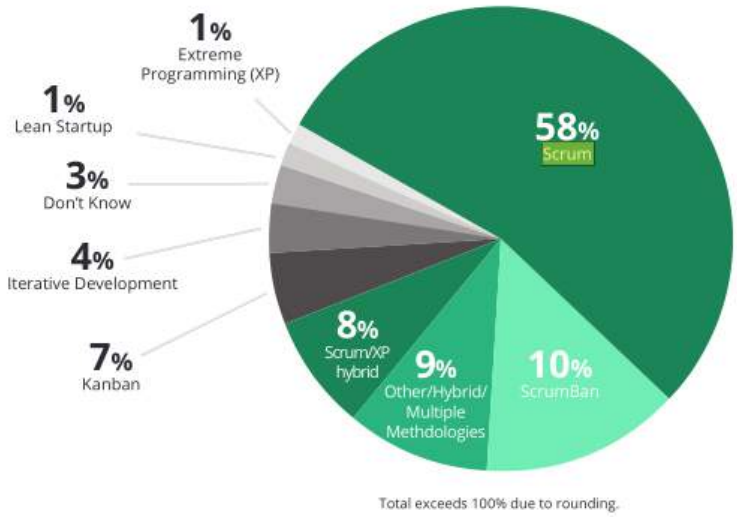


Figura 3-1 Resultado de Encuesta "14th Annual State of Agile Report" (VersionOne, 2020).

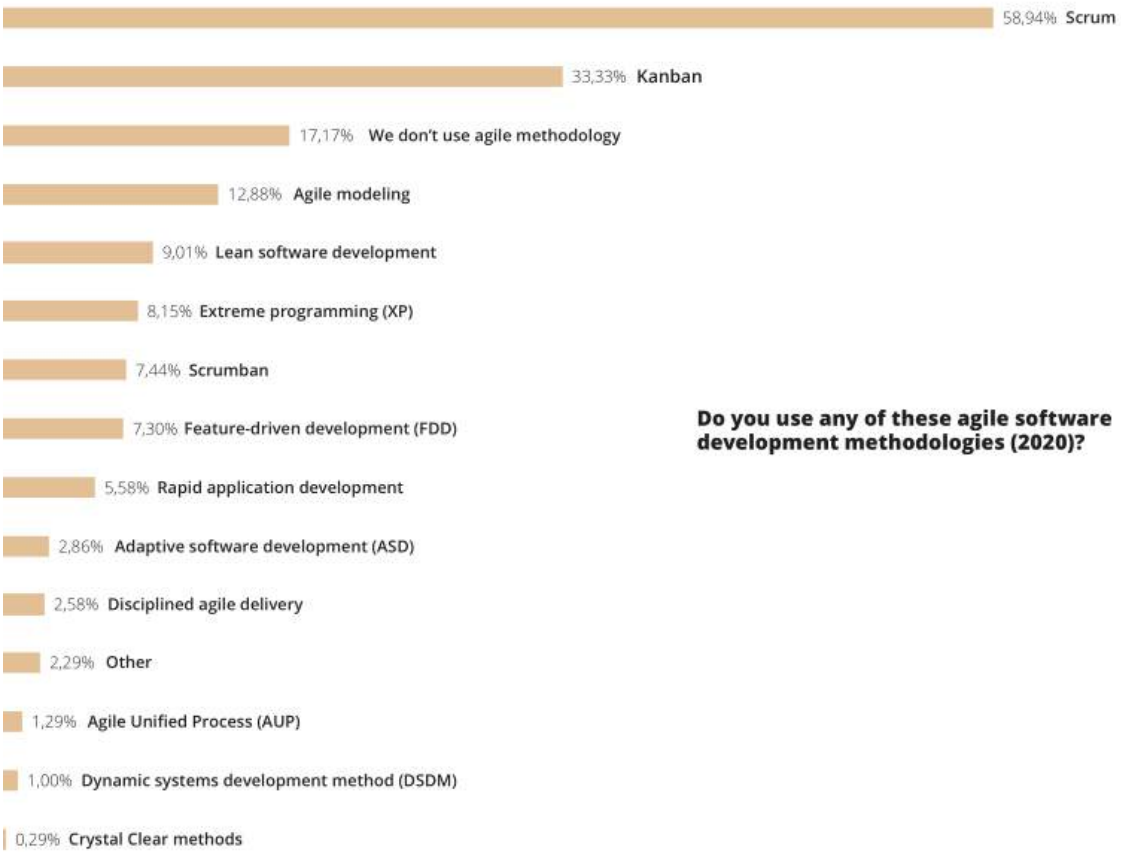


Figura 3-2 Resultado de Encuesta "State of Software Development" (Coding Sans, 2020).

### 3.4.1 Scrum

#### 3.4.1.1 Origen

A mediados de los 80, Hirotaka Takeuchi y Ikujiro Nonaka observaron que las empresas del sector tecnológico e innovación se caracterizaban por trabajar en un entorno muy competitivo e inestable. Después de analizarlas, encontraron que algunas de ellas habían cambiado su forma de trabajar y gestionar sus proyectos para adaptarse a la volatilidad del entorno y obtener mejores resultados. A raíz de esto, Takeuchi y Nonaka (Takeuchi & Nonaka, 1986) publicaron en 1986 el artículo The New New Product Development Game en el que describen el método de trabajo que utilizaban los equipos de proyectos de empresas americanas y japonesas (Fuji-Xerox, Canon, Honda, Nec, Epson, Brother, 3M y Hewlett-Packard) para desarrollar con éxito los nuevos productos que lanzaban al mercado. Estos equipos se caracterizaban por: trabajar juntos de principio fin, ser auto-organizados y multidisciplinarios. Otra característica que tenían en común era que solapaban las fases del ciclo de vida del producto, es decir, que de forma simultánea realizaban tareas de diferentes fases. Los autores compararon ese modelo de trabajo con la formación estratégica de rugby llamada SCRUM en el que el equipo avanza con el balón de forma conjunta hacia la misma dirección. En este artículo, fue utilizado por primera vez el término SCRUM como un nuevo enfoque para gestionar proyectos.

#### 3.4.1.2 Definición

Scrum es “un marco de trabajo por el cual las personas pueden abordar problemas complejos adaptativos, a la vez que entregar productos del máximo valor posible productiva y creativamente” (Schwaber & Sutherland, 2017, p. 3). Scrum es la metodología ágil más extendida y se basa en la adaptación continua y en la entrega de valor temprana al cliente. Además, se caracteriza por su desarrollo incremental e iterativo cuyas fases de desarrollo se solapan y se repiten las veces que sea necesario.

#### 3.4.1.3 Control del Proceso Empírico

El empirismo “asegura que el conocimiento procede de la experiencia y de tomar decisiones basándose en lo que se conoce” (Schwaber & Sutherland, 2017, p. 4). Una de las bases de Scrum es el trabajo colaborativo que permite a los miembros del equipo compartir sus experiencias, intercambiar conocimientos, revisar y reflexionar sobre su trabajo con la finalidad de mejorar. Como explica Sutherland (Sutherland, 2016, p. 16) Scrum ofrece un soporte al proceso de aprendizaje que se da en un proyecto software. En esta metodología, el control de procesos empíricos se considera como “Los tres pilares de Scrum” (Schwaber & Sutherland, 2017, pp. 4-5) y son:

- **Transparencia:** la información relevante del proyecto debe estar siempre actualizada, visible y accesible. Se aconseja utilizar una jerga o lenguaje en común para que la información del proyecto sea comprendida por todos.

- **Inspección: l**os miembros de equipo deben revisar de forma continua el trabajo. La inspección tiene la finalidad de comprobar que el proyecto se está desarrollando de acuerdo a la planificación, aumentar el control sobre el avance del proyecto y detectar cualquier desviación que pueda afectar.
- **Adaptación:** ante los posibles cambios que se presentan durante el proyecto, la planificación es revisada y ajustada al nuevo contexto, con la información obtenida gracias a los pilares anteriores.

#### 3.4.1.4 Prácticas de Scrum

El software cambia de forma constante para aumentar sobre el control sobre el proyecto, la metodología Scrum propone las siguientes actividades (Palacio, 2020, p. 28):

- **Revisión de las iteraciones**: cuando una iteración es completada, el software es probado para verificar que funciona correctamente. Luego se entrega al cliente para que valide si lo que se ha desarrollado es conforme a lo que en realidad necesita. Con esta práctica se asegura la satisfacción del cliente y también da la oportunidad para hacer propuestas de mejora.
- **Desarrollo incremental:** facilita la evaluación continua de las funcionalidades que se van desarrollando e integrando al producto final.
- **Solapamiento de las fases:** agiliza la adaptación a los cambios así como correcciones durante una iteración *(Véase Figura 3-3).*
- **Auto-organización del equipo:** toman decisiones sobre la gestión del trabajo. Scrum fomenta un ambiente idóneo para que surja el liderazgo y la pro actividad en las personas.
- **Colaboración:** los miembros del equipo de Scrum son cooperativos y dinámicos. Hay una conciencia colectiva acerca del valor del trabajo en equipo. El apoyo mutuo y su esfuerzo en conjunto facilitan a que el equipo consiga el objetivo.

#### 3.4.1.5 El Equipo Scrum. Roles y responsabilidades.

El equipo Scrum está formado por todas las personas que participan de forma directa en el proyecto. Cada miembro del equipo tiene un rol asignado con responsabilidades específicas. En Scrum existen tres roles: el dueño del producto, el scrum master y el equipo de desarrollo.

**El dueño del producto (Product Owner):** es el representante del cliente y tiene autoridad para tomar decisiones sobre el proyecto. Su principal objetivo es asegurar que el cliente obtenga el máximo valor posible (Palacio, 2020; Schwaber & Sutherland, 2017). El dueño del producto es el único responsable de priorizar, añadir, modificar o eliminar las historias de usuario en la Pila del Producto (Palacio & Ruata, 2016).

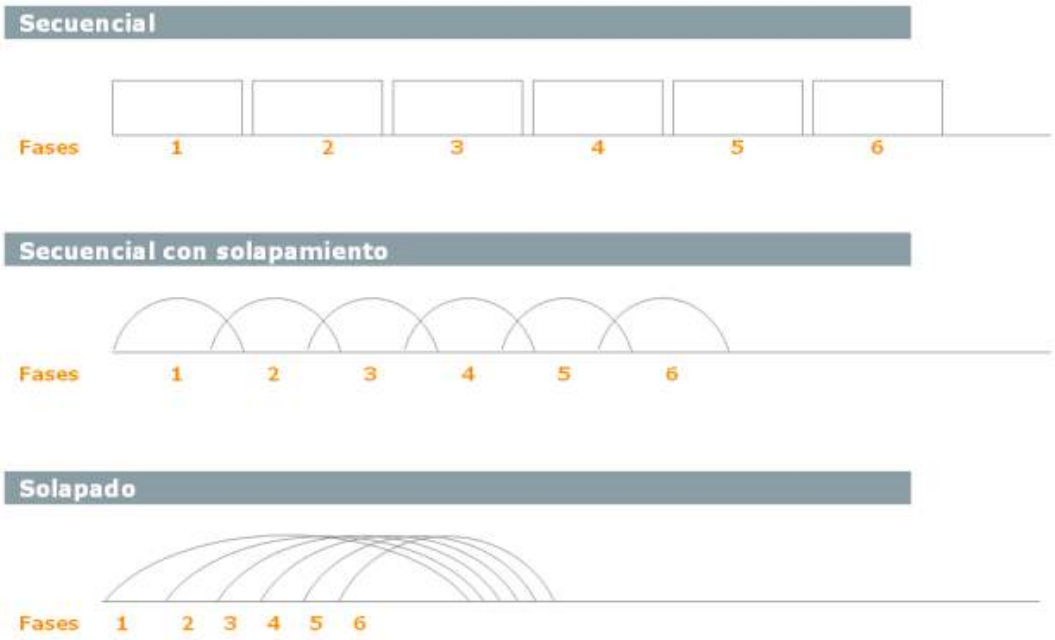


Figura 3-3 Producción con fases secuenciales o solapadas (Palacio & Ruata, 2011, p. 36).

Entre sus principales responsabilidades (SCRUMstudy, 2017; Schwaber & Sutherland, 2017; Palacio, 2020; Vanderjack, 2015) se pueden mencionar:

- Definir los requisitos del negocio y expresarlos de forma clara en la pila del producto. En Scrum, estos requisitos son llamados “historias de usuario”, en ellas se deben reflejar las necesidades de negocio que el software debe satisfacer.
- Ordenar las historias de usuario según nivel de prioridad (en función a las necesidades del negocio y al valor aportado).
- Asegurar que los miembros del equipo comprenden las historias de usuario descritas en la pila del producto.
- Establecer criterios de aceptación para cada historia de usuario.
- Verificar el cumplimiento de los requisitos en las funcionalidades desarrolladas.
- Validar los productos entregables del sprint y darlo por finalizado.
- Guiar el trabajo del equipo de desarrollo para que entreguen el máximo valor.
- Garantizar que el equipo de desarrollo cumpla según los plazos previstos.
- Trabajar de forma conjunta con el equipo de desarrollo. Debe participar en las reuniones programadas por el equipo del proyecto.
- Estar disponible para atender las consultas del equipo de desarrollo.
- Dar por finalizado el proyecto.

Aunque el dueño del producto es la única persona que representa al cliente, en empresas grandes

que desarrollan proyectos complejos, la responsabilidad del proyecto recae en un grupo de personas o comité (Palacio, 2020, p. 30). De igual forma, este comité debe elegir a un portavoz (Dueño del Producto) que los represente ante el equipo de desarrollo. Una restricción en Scrum es que únicamente el Dueño del Producto puede tener contacto directo con el equipo de desarrollo. Si el comité del proyecto necesita modificar algún elemento de la Pila del Producto, éste debe acudir al Dueño del Producto para que sea él quien transmita las nuevas necesidades (Schwaber & Sutherland, 2017, p. 6).

**Scrum Master:** es "un líder que está al servicio del Equipo Scrum" (Schwaber & Sutherland, 2017) , es el responsable de guiar al equipo para que cumplan las reglas y procesos de la metodología Scrum. Para ello, debe asegurarse que el equipo tiene los conocimientos suficientes y formar a los miembros del equipo. El Scrum Master tiene las siguientes responsabilidades (Palacio, 2020; Schwaber & Sutherland, 2017; Vanderjack, 2015):

- Asegurar que las reuniones Scrum son programadas y realizadas. También es el moderador y mediador en caso de una discrepancia entre los miembros del equipo. El Scrum Master debe buscar alternativas para encontrar la mejor solución a cualquier dificultad que se presente.
- Colaborar con el dueño del producto en la gestión de la Pila del Producto sugiriendo técnicas que ayuden a priorizar los requisitos, hacer modificaciones y añadir nuevos elementos. También verifica que las historias de usuarios sean claras y estén detalladas.
- Ayudar al equipo Scrum a comprender el proceso empírico que se desarrolla en Scrum. Les hace ver la importancia que tiene la participación de cada miembro del equipo durante todas las etapas del proyecto.
- Delegar autoridad a los miembros del equipo y los orienta para puedan auto-organizarse y ser multifuncionales.
- Ayudar al equipo a ser más eficaces y eficientes en su trabajo, y aseguren la entrega del máximo valor al cliente.
- Asegurar que no existan en el entorno del equipo, factores que dificulte su trabajo de desarrollo.
- Verificar que los objetivos y el alcance sean comprendidos por el equipo Scrum.
- De forma continua, revisar el progreso del equipo y verifica que se esté cumpliendo con el trabajo planificado en cada sprint.

**Equipo de Desarrollo:** es el grupo de personas responsables del desarrollo del software. Al final de la iteración se entrega un software funcional que puede ser utilizado por el cliente. El equipo de desarrollo está formado por un grupo de 4 a 8 personas. Scrum considera que un equipo de trabajo menor de cuatro personas podría limitar el nivel de interacción y perder la oportunidad de compartir conocimientos e ideas. Por otra parte, un grupo mayor a ocho personas dificulta la coordinación y comunicación (Schwaber & Sutherland, 2017, p. 7).

Para que un equipo sea verdaderamente Ágil y saque provecho del uso de la metodología Scrum es

imprescindible que el Scrum Master delegue liderazgo y autoridad al equipo. Esto los empodera del proyecto y otorga libertad para tomar decisiones y gestionar sus actividades. Otro punto importante para garantizar el éxito, es asegurar que todos los miembros del equipo conozcan y comprendan los objetivos del proyecto. En Scrum, los miembros del equipo de desarrollo se caracterizan por (Schwaber & Sutherland, 2017; SCRUMstudy, 2017; Palacio, 2020):

- Ser auto-organizados. Los miembros del equipo deciden la forma en cómo gestionan su trabajo.
- Ser multidisciplinarios. Tienen conocimientos especializados y complementarios que en conjunto son suficientes para llevar al cabo el proyecto.
- Trabajar unidos desde el inicio hasta el final del proyecto. Aunque cada miembro del equipo tiene asignado un conjunto de tareas, todos asumen la responsabilidad de finalizar el proyecto en tiempo, coste y con los requisitos especificados por el cliente.
- Trabajan con un alto nivel de compromiso y se esfuerzan por cumplir con los objetivos del proyecto, siendo el más importante: la entrega temprana y continua de valor al cliente.
- No tener jerarquía. Todos los miembros tienen voz y voto en las decisiones que se tomen como equipo.
- Mantienen una comunicación activa y directa entre todos los miembros.

Las principales responsabilidades del equipo de desarrollo son (Vanderjack, 2015; SCRUMstudy, 2017):

- Desarrollar las funcionalidades del software y entregar en plazo el incremento del software al final de cada sprint.
- Definir las tareas a realizar en cada sprint.
- Actualizar diariamente la información del avance del proyecto.
- Trabajar junto al dueño del producto en la creación de la pila del producto.
- Mantener una comunicación constante con el Dueño del Producto.
- Estimar el esfuerzo requerido de cada historia de usuario, y crear junto al dueño del producto los criterios de aceptación de cada una.
- Compartir el conocimiento y aprendizaje para mejorar el proceso de desarrollo software.

#### 3.4.1.6 Artefactos

Los artefactos de Scrum son herramientas utilizadas para mostrar el trabajo del proceso Scrum.

**Pila del producto:** es la lista priorizada de los requisitos del cliente. En ella se expresan todas las necesidades de negocio que se espera que el software satisfaga. También incluye información de mejoras técnicas, actividades de investigación, corrección de errores, entre otras (Deemer, et al., 2012, pp. 6-7). La pila del producto es una herramienta de comunicación que contiene toda la información del trabajo que el equipo de desarrollo tiene que realizar (Palacio, 2020, p. 34).

Cuando se crea una historia de usuario se realiza una descripción muy general de la idea principal. Durante la evolución del proyecto, el dueño del producto junto al equipo de desarrollo van aclarando y añadiendo información a cada historia de usuario. En Scrum, a esta actividad se le llama Refinamiento de la Lista de producto y es definida como "el acto de añadir detalle, estimaciones y orden a los elementos de Producto" (Schwaber & Sutherland, 2017, p. 16). El objetivo es que cada historia de usuario tenga la información detallada y suficiente para que pueda descomponerse en varias tareas. Esta información aumenta la precisión de las estimaciones de esfuerzo y tiempo necesario para realizarlas. La estimación de las historias de usuario es una responsabilidad exclusiva del equipo de desarrollo.

Las historias de usuario que tienen mayor prioridad son las primeras que deben ser detalladas; éstas se ubican en un nivel alto de la pila del producto. Cada una contiene una descripción detallada, su nivel de prioridad, estimación del esfuerzo y los criterios de aceptación. El dueño del producto es el encargado de asignar la prioridad a cada historia de usuario. Es fundamental que la historia de usuario sea comprendida y aceptada tanto por el dueño del producto como por el equipo de desarrollo para ser categorizada como "Preparada", es decir, que está lista para ser incluida en la siguiente reunión de planificación del sprint (Palacio, 2020, p. 35; Schwaber & Sutherland, 2017, p. 16). La pila del producto no es rígida sino que evoluciona y debe adaptarse a las necesidades que vayan emergiendo durante el proyecto.

**Pila de sprint:** son las historias de usuario a realizar durante el sprint. Éstas son seleccionadas durante la planificación del sprint por todos los miembros del equipo Scrum. Para esta selección, se toma en cuenta la prioridad asignada y las estimaciones del equipo de desarrollo. De manera que se asegura la entrega temprana de valor al cliente y que el equipo de desarrollo sea capaz de hacer el trabajo en el plazo establecido.

El equipo de desarrollo descompone las historias de usuario en tareas definidas con el suficiente nivel de detalle que permita una estimación de esfuerzo realista. También es el único autorizado a modificar la pila del sprint una vez que haya iniciado (Palacio, 2020, p. 36).

La pila del sprint es una herramienta visual que sirve al equipo de desarrollo para conocer el progreso del sprint por lo que ante cualquier modificación se debe actualizar. La metodología recomienda que esta información esté accesible (a través de un tablero, un software de gestión de proyectos, u otros medios que se consideran más apropiados). Según el libro de Scrum Master (Palacio, 2020, p. 37) es importante que la pila del producto incluya:

- Sólo la información relevante: la lista de tareas, el responsable de cada una y estado actual.
- En ella se debe registrar diariamente el avance de cada tarea, así como el esfuerzo pendiente.
- Tiene que ser una herramienta que ayude a mantener una comunicación eficaz y directa entre los miembros del equipo.

**Incremento:** es una mejora, una versión más completa del software con respecto al sprint anterior. Según Roche (Roche, 2020) es el resultado de cada sprint y se define como "la suma de todas las tareas, casos de uso, historias de usuario y cualquier elemento que se haya desarrollado durante el

Sprint y que será puesto a disposición del usuario final en forma de software, aportando un valor de negocio al producto que se está desarrollando".

Para que un incremento sea considerado como finalizado, el equipo Scrum debe acordar una "Definición de Terminado". Esta definición puede variar, pero en términos generales, es establecer criterios de aceptación, que pueden ser una lista de actividades o comprobaciones, con los que se demuestre que el incremento está completo, es funcional, cumple con el objetivo del sprint y añade valor al software.

#### 3.4.1.7 Eventos

Scrum establece eventos recurrentes de actividades como planificación, desarrollo del software, pruebas, reuniones, entre otros. Estos eventos son bloques de tiempo (en inglés Time-boxes) con una duración establecida y son utilizados para facilitar la organización de tareas y coordinación.
Los bloques de tiempo tienen un horizonte de trabajo corto y alcanzable, lo que facilita al equipo a administrar los recursos, aumentado el rendimiento y disminuyendo los desperdicios (SCRUMstudy, 2017, p. 36). Los eventos de Scrum son:

**El Sprint:** El Sprint es considerado como la pieza clave de la metodología de Scrum, algunos autores (Schwaber & Sutherland, 2017, p. 9) lo nombran como "el corazón de Scrum". Un sprint es una iteración con una duración máxima de un mes. Se procura que todos los sprints tengan la misma duración para mantener una constancia en las entregas del proyecto. Un Sprint contiene todos los eventos de Scrum.

En cada Sprint se establece un objetivo específico por el cual todo el equipo de desarrollo se esfuerza por cumplir (Objetivo del Sprint). Este objetivo se consigue a través de las funcionalidades que se desarrollan y entregan al final de la iteración.

Una vez inicia el sprint ya no se admiten cambios que puedan entorpecer la planificación o dificultar el cumplimiento del Objetivo del Sprint. En situaciones específicas, el dueño del producto puede solicitar la cancelación del sprint si considera que el objetivo del sprint pierde sentido y no aporta ningún valor para el cliente (Schwaber & Sutherland, 2017, p. 10).

**Planificación del Sprint (Sprint Planning):** es el primer evento que se celebra el equipo Scrum; se le llama también "sprint 0". Los participantes de esta reunión son todos los miembros del equipo Scrum. La reunión tiene asignado un bloque de tiempo específico que varía según la duración del sprint que se va a planificar. Por ejemplo, si el sprint que se está planificando es de un mes, se estima que la reunión tendrá una duración máxima de ocho horas. El scrum master es el responsable de facilitar esta reunión y de asegurar que todos los interesados estén presentes y comprendan su objetivo. (Schwaber & Sutherland, 2017, p. 10). La planificación del sprint se divide en dos secciones (Cobb, 2015, pp. 41-42; Schwaber & Sutherland, 2017, pp. 10-11):

1. En la primera parte de la reunión se establecen las funcionalidades se van desarrollar. También se define el objetivo del sprint, el cual debe reflejar el valor que recibirá el cliente con ese incremento. El dueño del producto dirige esta primera parte de la reunión en la que expone la Pila del Producto con las historias de usuario ordenadas según su nivel de prioridad. Una vez

que el equipo de desarrollo conoce y comprende el objetivo del sprint y las prioridades del trabajo, lleva a cabo una estimación de tiempo y esfuerzo que se requiere para desarrollar las historias de usuarios. Tras la estimación, el equipo evalúa su capacidad de entrega y define la cantidad de historias de usuario que puede completar en ese sprint.

2. En la segunda parte, se traza una estrategia de trabajo para lograr el objetivo del sprint. El equipo de desarrollo se auto-organiza y decide cómo llevará a cabo el trabajo. Para esta planificación, el equipo de desarrollo comienza a descomponer cada historia de usuario en pequeñas tareas, procurando que la duración máxima sea de un día y balanceando la carga de trabajo del equipo. Es decir, que el trabajo planificado no sea ni excesivo o por el contrario, sea insuficiente para la duración del sprint. El resultado de esta reunión es la Pila del Sprint.

**Scrum Diario (Daily Scrum):** tiene un bloque de tiempo de 15 minutos. Esta reunión se realiza todos los días, a la misma hora y en el mismo lugar. Los miembros del equipo permanecen de pie, preferiblemente frente a un tablero u otro elemento en el que puedan visualizar la información actualizada del avance del Sprint (Cobb, 2015, p. 42; Palacio, 2020, p. 43). Esto puede ser un tablero de tareas donde se vea reflejada la Pila del Sprint con el estado de cada historia de usuario. Aquí puede añadirse el Gráfico de Avance o (Burndown en inglés) para observar el trabajo pendiente por hacer con respecto al número de días que faltan para terminar el sprint.

El objetivo de esta reunión es conocer de forma directa y rápida el estado actual del proyecto. Los miembros del equipo hablan sobre el trabajo que han hecho desde la última reunión. Cada uno explica las tareas que van a realizar en las siguientes horas y comentan si consideran que pueden tener algún problema para cumplir con la programación. Esta información facilita el monitoreo y control del proyecto, ya que diariamente los miembros del equipo revisan el trabajo y verifican si los avances del proyecto están encaminados al cumplimiento del objetivo del sprint. Con esta actualización de datos, se planifica el trabajo de las próximas 24 horas.

El Scrum Master vigila que estas reuniones diarias se realicen, pero es el equipo de desarrollo el responsable en organizarlas y dirigirlas. Aunque esta reunión es exclusiva para los miembros del equipo de desarrollo, pueden asistir otras personas que tengan interés en el proyecto, pero no pueden participar en ella.

**Revisión del Sprint (Sprint Review):** se realiza al final de cada sprint para mostrar el incremento del software desarrollado. La reunión tiene asignado un bloque de tiempo específico que varía según la duración del sprint. Por ejemplo, si el sprint ha sido de un mes, se estima que la reunión tendrá una duración máxima de cuatro horas. En ella participa todo el equipo Scrum y otras personas interesadas en el proyecto.

El equipo de desarrollo hace una demostración sobre el incremento del software y resume las funcionalidades que han sido desarrolladas y entregadas en ese sprint. También comparten la experiencia que tuvieron durante del sprint, destacando los aspectos buenos, los problemas a los

que se enfrentaron y las soluciones que implementaron. El dueño del producto muestra la pila del Producto actualizada con las historias de usuario que han sido terminadas y las que están pendientes[18]. Una vez que se acaba la presentación se responden a las preguntas y resuelven dudas acerca del incremento (Schwaber & Sutherland, 2017, p. 13).

La reunión de revisión de mucho valor para el proyecto porque se obtiene una retroalimentación clara y directa acerca del software. Con la información actualizada del proyecto se abre una consenso acerca de lo que podría ser el siguiente sprint: un objetivo, una posible pila de sprint, unas fechas tentativas de inicio y finalización del sprint. La reunión de revisión viene a ser un preparativo para la siguiente reunión de planificación. En resumen, la revisión del sprint no es únicamente para mostrar el incremento del producto y revisar el estado actual del proyecto, si no que sirve como un nuevo punto de referencia para tomar decisiones sobre el futuro y establecer una estrategia de negocio (Roche, 2020)**.**

**Retrospectiva del Sprint:** es una reunión que tiene la finalidad reflexionar sobre el trabajo y el resultado del último sprint. Se realiza después de la revisión del sprint y antes de la planificación del siguiente sprint La reunión tiene asignado un bloque de tiempo específico que varía según la duración del sprint. Por ejemplo, si el sprint ha sido de un mes, se estima que la reunión tendrá una duración máxima de tres horas (Schwaber & Sutherland, 2017, p. 14). En esta reunión participan todos los miembros del equipo Scrum y comparten su experiencia con el objetivo de aprender y mejorar para el siguiente sprint. Los miembros del equipo hacen una inspección global acerca de su rendimiento y destacan aquellas fortalezas que han sido importantes y que han influido positivamente en el trabajo. También identifican sus puntos más débiles y vulnerables que han afectado al proceso de desarrollo (Palacio, 2020, p. 46). Esta inspección incluye tanto aspectos técnicos (procesos y herramientas) como sociales (trabajo en equipo y comunicación) (SCRUMstudy, 2017).

El resultado de esta reunión es la creación de un plan de mejora a implantar tan pronto como sea posible. La retrospectiva una de las actividades más importantes en el proceso ágil.

[18] Es importante destacar que, antes de la Reunión de Revisión, el equipo de desarrollo ha mostrado al dueño del producto el incremento del software (Demo del Sprint). Él ha revisado cada una de las funcionalidades y ha determinado si cumplen o no con los criterios de aceptación y con el objetivo del sprint (Cobb, 2015, p. 43).

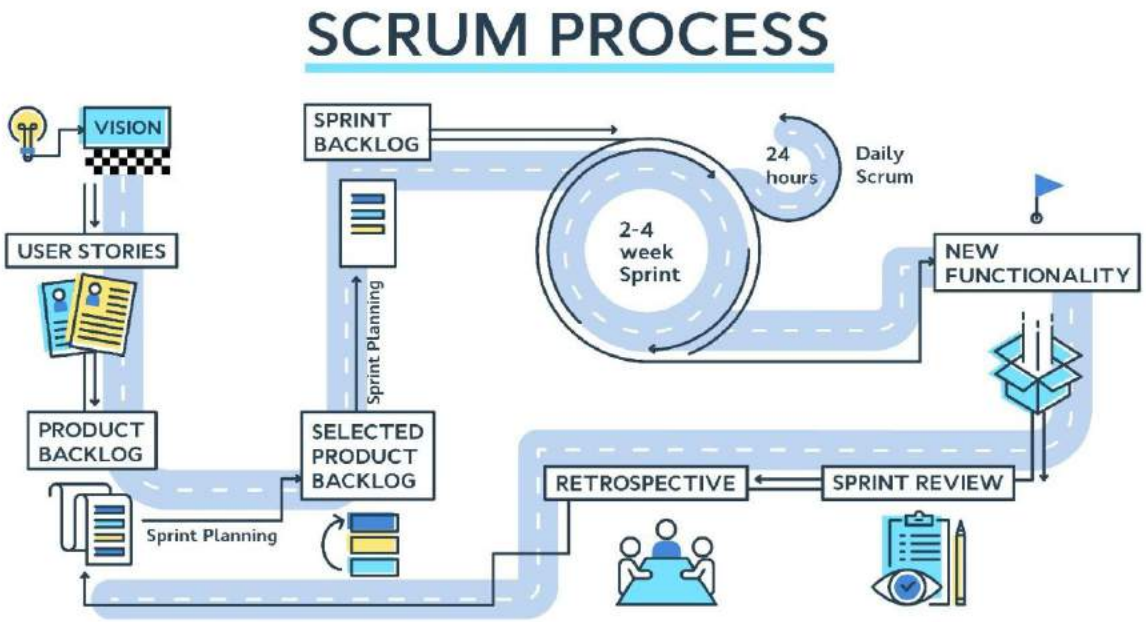


Figura 3-4 Proceso Scrum (Crystalloids, s.f.).

#### 3.4.1.8 Medición

En la gestión de proyectos las medidas son fundamentales para conocer el estado real del proyecto. Con las medidas se puede cuantificar diversos aspectos como la satisfacción del cliente, el valor aportado, el rendimiento del equipo, la calidad del producto, el progreso entre otros. Al medir y comparar aumentamos la base de conocimientos para la toma de decisiones y permite realizar un control y seguimiento sobre el proyecto y la organización.

Existe un principio de la agilidad directamente relacionado con las medidas: "El software funcionando es la principal medida de progreso". Al realizar una entrega se puede comparar el trabajo terminado con el trabajo pronosticado para obtener la medida del desempeño del equipo. También se puede calcular el valor aportado por el desarrollo al comparar el objetivo del sprint con los resultados obtenidos tras la entrega. Si el equipo tiene un alto desempeño pero el desarrollo no aporta el valor previsto podemos modificar la hoja de ruta y replantear soluciones.

En un proyecto ágil se toman en cuenta tres magnitudes para medir el avance del proyecto: velocidad, trabajo y tiempo. En Scrum se define la velocidad como "la cantidad de trabajo realizada por unidad de tiempo" (Palacio & Ruata, 2011, p. 95).

$$Velocidad = \frac{Trabajo}{Tiempo}$$

Es vital que la medida sea concisa, veraz, relevante y lo más precisa posible, de lo contrario, puede alterar los resultados y ser contraproducente, aparte de consumir recursos ya que algunas prácticas para la toma de medidas pueden ser muy costosas.

Un factor importante en la satisfacción del cliente es que el equipo de desarrollo cumpla con la planificación en la fecha acordada. Para ello, es importante realizar una estimación lo más acertada posible. En cada Planificación del sprint se estima el esfuerzo necesario para cumplir con el objetivo del sprint y en la revisión del Sprint se verifica el esfuerzo que invertido.

**Estimación de historias de uso**

Tradicionalmente se ha usado unidades de tiempo para la estimación de tareas. Esta unidad de medida es muy rígida y supone un compromiso adquirido con un alto nivel de incertidumbre. Para reducir la tasa de error en la estimación se recurre a un intervalo de tiempo pesimista y optimista. Esta práctica puede ser útil solo si la tarea se asigna a un miembro del equipo cuyas capacidades son conocidas, ya que existen múltiples factores que afectan su rendimiento (experiencia, carga de trabajo, motivación), por lo que el tiempo de implementación de una tarea puede variar entre diferentes desarrolladores.

Scrum propone el esfuerzo como unidad de medida para la estimación de historias de uso. El esfuerzo es una unidad relativa que se establece por consenso del equipo basado en la experiencia de un desarrollo previo con el cual se pueda comparar. Este desarrollo previo se usa como referencia para la estimación de las historias de uso, denominado puntos de historia. El uso de puntos de historia aporta flexibilidad en la estimación dado que los esfuerzos son comparados con una referencia, de esta forma se puede estimar si una historia de uso supone dos, tres o cuatro veces más esfuerzo. Al tratarse de una medida relativa, puede dificultar la estimación de historias de uso complejas, ya que se puede presentar la duda si una historia de uso requiere 31 o 32 veces más esfuerzo. Para facilitar la estimación, reducir la complejidad y mejorar la precisión se recomienda usar una escala no continua para los valores de puntos de historia, lo cual reduce las opciones de comparación. Una escala muy extendida es la sucesión de Fibonacci *(Veáse Anexo 6.3).*

**Planning Poker**

Una herramienta muy extendida para la estimación de las historias de uso es el planning poker. A cada miembro del equipo se le entrega un juego de cartas cuyos valores han sido establecidos previamente. El equipo debate sobre los aspectos particulares de la historia de usuario seleccionada y asignan una estimación que mantienen en privado. Cuando todos los miembros del equipo confirmen que han elegido la estimación, levantan sus cartas de forma simultánea para hacer pública las estimaciones. Si existen valores muy extremos (altos y bajos) cada miembro del equipo expone las razones que justifican su elección. Se realiza un pequeño

debate y vuelven a estimar la historia de usuario. Según los resultados, llegan a un consenso sobre la cantidad de puntos de historias que son asignados a la tarea en cuestión.

Para aprovechar los beneficios de esta actividad, los autores (Palacio & Ruata, 2011, p. 104) recomiendan:

- Descomponer las historias de usuario en tareas pequeñas, para facilitar la estimación e identificar los riesgos.
- Las tareas no deben tener una duración menor a media jornada de trabajo.

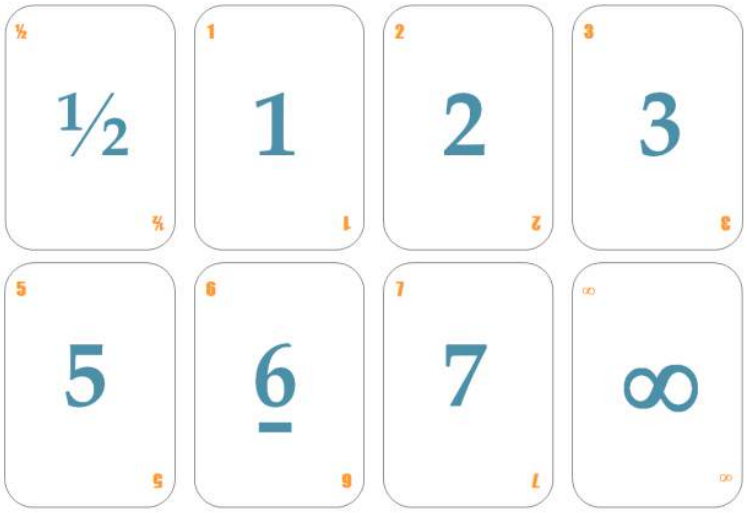


Figura 3-5 Ejemplo de cartas para estimación de póquer (Palacio & Ruata, 2011, p. 104).

## 3.4.2 Extreme Programming

### 3.4.2.1 Definición

Extreme Programming (XP) es una metodología ágil que se enfoca en la entrega de valor al cliente y en el trabajo en equipo (Crawford & Leon de la Barra, 2008). Fue desarrollada por Kent Beck en 1999 y publicada en su libro "Extreme Programming Explained: Embrace Change". Beck (Beck, 2005, p. 1) define a XP como un "cambio social" refiriéndose a que es una metodología que invita a modificar el pensamiento individualista para formar parte de una comunidad que trabaja unida para conseguir un objetivo. Hace hincapié en la importancia de fomentar las buenas relaciones interpersonales y lo califica como un factor que afecta directamente a la productividad y la confianza dentro del ámbito laboral. También menciona otro elemento de mucho valor, y es la buena técnica de programación. Beck insiste en la búsqueda de la excelencia, invita a los desarrolladores a esforzarse para hacer cada vez mejor su trabajo a través de prácticas simples pero llevadas al "extremo" como las entregas en ciclos cortos para una retroalimentación rápida, utilizar un enfoque incremental o implementar un desarrollo flexible que se adapta a las necesidades emergentes.

#### 3.4.2.2 Valores

XP basa su método de trabajo en cinco valores:

1. **Comunicación:** XP hace énfasis en mantener una comunicación continua y eficaz entre los miembros del equipo de desarrollo, con el cliente u otra parte interesada del proyecto. Beck atribuye a que muchos de los problemas que se dan durante el desarrollo software son debido a problemas de comunicación (Beck, 1999). XP propone prácticas como el cliente in situ y la programación en parejas mediante la cual la comunicación es directa.

2. **Simplicidad:** hacer un desarrollo lo más simple posible para cumplir exactamente con lo que el cliente necesita y no invertir tiempo en el desarrollo innecesario. Beck (Beck, 1999) hace una reflexión acerca del beneficio mutuo que entre la comunicación y la simplicidad, entre mejor sea la comunicación habrá un entendimiento más claro y específico sobre el trabajo, lo que evitará dudas, errores y reprocesos. De la misma forma la simplicidad agiliza el proceso, y no hay necesidad de una comunicación excesiva.

3. **Retroalimentación:** Durante el desarrollo software se realizan diferentes prácticas en las que se transmite y recibe información valiosa acerca del estado del proyecto y la calidad del software. En las revisiones del trabajo, el equipo de desarrollo obtiene información necesaria para tomar decisiones rápidas en caso que el proyecto no esté cumpliendo con los objetivos del cliente. Las pruebas unitarias es otra fuente de información sobre el correcto funcionamiento del software.

4. **Coraje:** en XP el coraje está relacionado con la toma de decisiones cuando el proyecto se ha desviado lo suficiente como para no cumplir con los objetivos. Por ejemplo, reformular incluso eliminar un código cuando el proyecto está muy avanzado. Esto requiere un gran esfuerzo y mucha dedicación por parte de los desarrolladores. No es para nada una tarea sencilla, por el contrario hay que ser muy perseverante hasta lograr obtener los resultados que se esperan. Aunque puede significar tirar a la basura muchas horas de trabajo (depende del defecto a corregir), esto puede ser más productivo que invertir horas identificando y arreglando los errores.

5. **Respeto**: es un valor muy promovido por XP. Las relaciones basadas en el respeto son más cooperativas y productivas. Durante el desarrollo del software se respetan las opiniones e ideas de todos los interesados. Cada aportación es muy bien recibida y valorada. Realizar el trabajo de la mejor manera posible es otra muestra de respeto hacia el cliente y los demás miembros del equipo

#### 3.4.2.3 Ciclo de vida de un Proyecto XP

Beck (Beck, 1999) propone un ciclo de vida ideal para llevar a cabo un proyecto software *(Véase Figura 3-6)* utilizando la metodología de XP y está formado por seis fases:

**Exploración:** en esta fase el cliente expone las necesidades de negocio y se transforman en historias

de usuario que son recogidas en documentos con formato de tarjetas. El equipo de desarrollo evalúa las tecnologías y herramientas que serán utilizadas y realizan un primer planteamiento acerca de la arquitectura del software.

**Planificación de la Entrega:** el cliente asigna el nivel de prioridad a cada historia de usuario y selecciona aquellas que serán implementas para la primera entrega. El equipo de desarrollo estima el esfuerzo requerido y las tareas necesarias para desarrollar cada historia de usuario. Se elabora un cronograma con la lista de tareas y fecha de entrega.

**Iteraciones: C**ada ciclo tiene una duración máxima de tres semanas. El cliente indica el orden en que serán desarrolladas las historias de usuario. En cada iteración se establecen pruebas funcionales que el cliente realizará para validar cada funcionalidad desarrollada.

**Producción:** el software es sometido a múltiples pruebas para confirmar que todas las funcionalidades se han desarrollado correctamente. Una vez comprobado el estado del software es desplegado al entorno del cliente para que pueda ser utilizado por los usuarios.

**Mantenimiento:** esta actividad se da durante toda el ciclo del software y su objetivo es garantizar el buen funcionamiento del software, en particular cuando el equipo de desarrollo despliega una nueva versión del mismo. Se verifica el rendimiento y se le da soporte al cliente para la introducción de mejoras o corrección de errores.

**Muerte del Proyecto:** es la finalización del proyecto. Se puede presentar por haber alcanzado los objetivos y la satisfacción del cliente o, por el contrario el cliente no recibió lo que esperaba y el proyecto perdió el sentido para continuar, o por cuestiones del presupuesto.

#### 3.4.2.4 Prácticas XP

XP utiliza doce prácticas:

1. **El juego de la planificación:** es un diálogo entre el cliente y el equipo de desarrollo. Se definen las historias de usuario y se determina el valor a cada una de ellas y las ordena según su prioridad. Por otro lado el equipo de desarrollo estima el esfuerzo necesario por cada historia de usuario, define qué cantidad puede entregar en cada iteración y coordina el trabajo. Al final se determina una fecha de entrega.
2. **Entregas pequeñas:** el objetivo es desarrollar las suficientes funcionalidades para entregar un software operativo en corto tiempo, es decir, entregar el máximo valor en el menor tiempo posible.

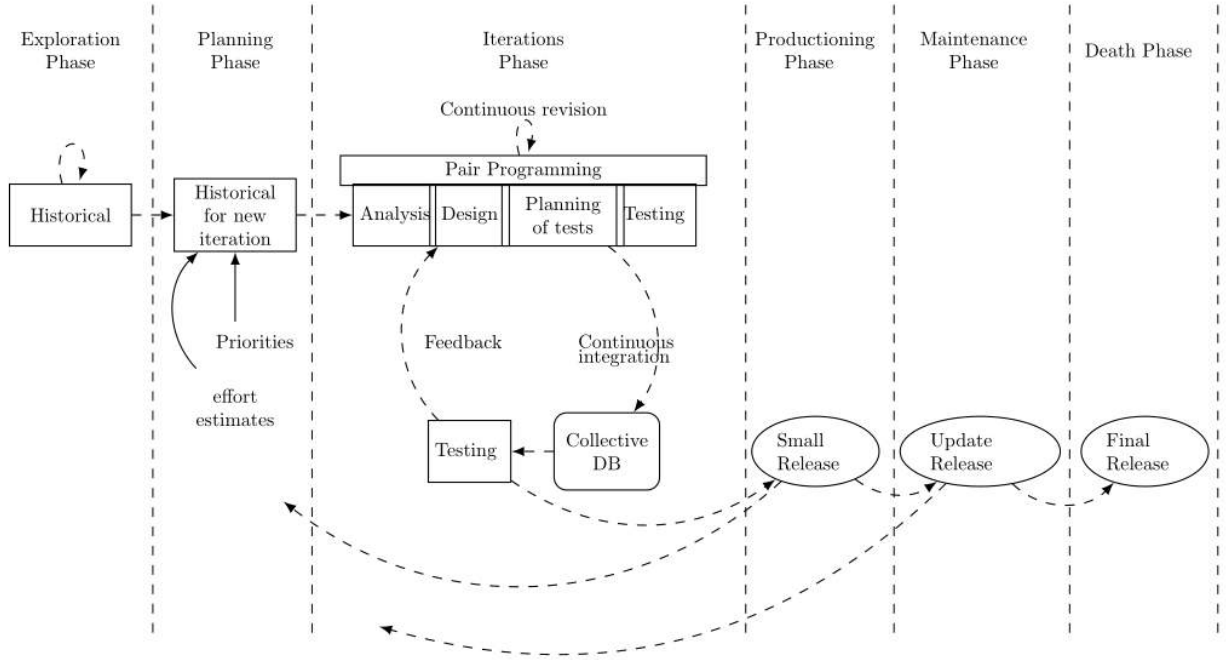


Figura 3-6 Ciclo de Vida XP (Crawford & Leon de la Barra, 2008).

3. **Metáfora:** es un método para explicar las funcionalidades del software de forma simple y comprensible para los interesados, facilitando las explicaciones y tecnicismos (clases, métodos, relaciones, objetos) al cliente. (Beck, 1999) Es mayormente utilizada al inicio del proyecto debido a que la arquitectura y el diseño no están definidos, y se va refinando a medida que avanza el desarrollo. Se debe elegir una metáfora apropiada para evitar confusiones.
4. **Diseño simple:** es evitar la complejidad y el trabajo innecesario. El objetivo es desarrollar únicamente los requisitos expresados en las historias de usuario.
5. **Pruebas:** el equipo de desarrollo define las pruebas unitarias que ayuden a verificar si las funcionalidades han sido desarrolladas correctamente. Por otra parte, el cliente define las pruebas de validación para determinar si las funcionalidades son correctas, es decir, si cumplen con lo que el expresó en la historia de usuario.
6. **Refactorización:** es una actividad que se realiza durante el desarrollo de software para producir un código sencillo y fácil de mantener (que los cambios en el futuro sean más fáciles de implementar) (Sommerville, 2011, p. 66).
7. **Programación en parejas:** consiste en que una persona se encarga de escribir el código y la otra persona revisa su trabajo y a la vez piensa en opciones de cómo se puede mejorar. Algunos de los beneficios obtenidos son la detección rápida de errores, mejora en el diseño, líneas de código más cortas, mayor aprendizaje, aumento en la calidad del software y reducción de costes (Cockburn & Williams, 2001)
8. **Propiedad colectiva del código**: cualquier desarrollador puede cambiar parte del código en cualquier momento. En los proyectos de XP esta práctica se permite porque todos los desarrolladores son responsables del código, tienen la capacitación requerida y están muy

involucrados en el proyecto.

9. **Integración continua:** cuando una tarea es finalizada, el nuevo código se integra al sistema y se realizan pruebas para verificar su buen funcionamiento. La práctica de integración se realiza varias veces al día.
10. **40 horas por semana:** se respeta la jornada laboral de un máximo de 40 horas a la semana. Es importante que no haya un exceso en la carga laboral con jornadas interminables de trabajo. Beck (Beck, 1999) insiste en esta práctica manifestando que es aceptable trabajar horas extras en una semana, pero que no puede existir una segunda semana trabajando demasiadas horas. De ser así, está claro que el proyecto tiene un problema y la planificación necesita ser ajustada.
11. **Cliente in-situ:** el cliente es parte del equipo de desarrollo (Sommerville, 2011, p. 66), debe estar presente y trabajando junto a los desarrolladores respondiendo preguntas y aclarando dudas. Esta práctica es muy importante porque se asegura la entrega de valor al cliente.
12. **Estándares de programación:** la comunicación entre los miembros del equipo de desarrollo es través del código, por lo cual es necesario establecer un estándar de programación para que el código sea legible y comprensible. De igual forma, tener los mismos lineamientos para la descripción de las historias de usuarios, la metáfora, las pruebas y otros elementos importantes del software (Holcombe, 2008, p. 32). Lo más importante es tener una comunicación clara y fluida.

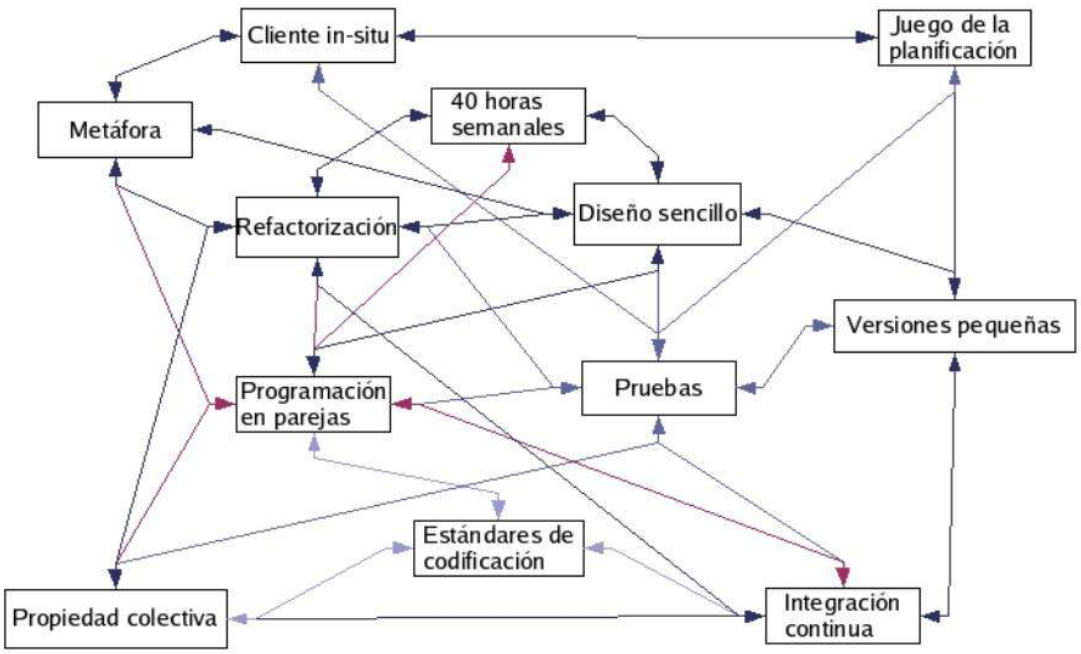


Figura 3-7 Las prácticas se soportan mutuamente (Letelier & Penadés, 2006; Beck, 1999).

#### 3.4.2.5 Roles

Un equipo extremo tiene roles específicos que describen la responsabilidad de cada miembro. Algunos de estos roles hacen su trabajo de forma individual y otros necesitan trabajar de forma conjunta con otras personas. Según las necesidades del proyecto, algunos roles pueden rotar entre los miembros del equipo. La metodología XP propone los siguientes roles (Beck, 1999):

- **Programador:** considerado por Beck como "el corazón de XP"; es el responsable del desarrollo software y de escribir las pruebas unitarias
- **Cliente:** escribe y detalla las historias de usuario y les asigna un valor para luego priorizarlas. También es el responsable de definir los criterios de validación y hacer las pruebas funcionales a cada historia de usuario.
- **Encargado de Pruebas (Tester):** ayuda al cliente a seleccionar y escribir las pruebas funcionales. También ejecuta las pruebas de forma continua, comparte los resultados y verifica el estado del las herramientas de prueba.
- **Encargado de seguimiento (Tracker):** se encarga de hacer las estimaciones del proyecto y de llevar un control y monitoreo de su avance. Debe vigilar que el equipo de desarrollo está cumpliendo con el cronograma del proyecto. Ofrece una retroalimentación continua al equipo de desarrollo sobre el estado del proyecto. También es responsable de notificar cuando se presenta una desviación significativa en la que se necesite introducir un cambio para nuevamente encauzar el proyecto. Adicionalmente, lleva un histórico de las funcionalidades desarrolladas, de los errores encontrados y del responsable en corregirlos.
- **Entrenador (coach):** es un experto de la metodología XP. Se asegura que todo el equipo comprenda la metodología XP, y lo guía para que implementen las prácticas y desarrollen el proceso correctamente.
- **Consultor:** es una persona externa al equipo de desarrollo con conocimientos técnicos muy específicos que guía al equipo para resolver un problema. Algo importante es que el consultor no resuelve el problema directamente, ofrece sus conocimientos, da su opinión y alternativas para que sea el mismo equipo el que implemente la solución.
- **Gestor (Big Boss):** coordina las actividades del equipo de desarrollo y le ayuda a resolver los problemas que se presenten durante el desarrollo software. El gestor conoce el estado actual del proyecto, revisa continuamente los resultados e interviene en el proceso cuando considere que el equipo está fallando y es necesario realizar un cambio.

### 3.4.3 KANBAN

#### 3.4.3.1 Definición

Kanban es un término japonés que significa tarjeta de señal. El método Kanban fue desarrollado por Taiichi Ono para mejorar la eficiencia del proceso de producción de Toyota. Se basa en tarjetas de

señales que dan aviso sobre la cantidad de suministro que hay disponible para abastecer el proceso de producción. Kanban ayuda a tener un mejor control de las existencias, de manera que solo se produzca lo que se necesita y que esté listo en el momento requerido.

David J. Anderson (Anderson, 2010) fue el responsable de introducir Kanban como una metodología para el desarrollo de software. En su libro "Kanban: Successful Evolutionary Change for Your Technology Business" Anderson explica que el método Kanban es utilizado para ayudar al equipo del proyecto a visualizar el trabajo y conocer en tiempo real el estado o la etapa en la que se encuentra cada tarea. En el método Kanban las tarjetas representan tareas que el equipo de desarrollo tiene que completar. Otra de sus funciones es que ayuda a limitar el trabajo que está en progreso.

#### 3.4.3.2 Valores

Los valores en Kanban pueden se concentran en un único valor: el respeto (Anderson & Carmichael, 2017, pp. 3-4).

- **Transparencia:** la información del proceso es visible y está al alcance de todos.
- **Equilibrio:** tener un balance en la carga de trabajo.
- **Colaboración:** fomenta la participación y el trabajo entre los miembros del equipo y el cliente.
- **Enfoque en el cliente:** los esfuerzos están orientados a la entrega de valor al cliente. El equipo comprende el objetivo del cliente y trabaja por cumplirlo.
- **Flujo:** el flujo del proceso representa el flujo de valor. Cada vez que una tarea avanza en cada etapa del proceso obtiene más valor.
- **Liderazgo:** aunque no existe un rol de líder en el método Kanban, si se espera que el liderazgo surja entre los miembros del equipo para promover la iniciativa, dirigir el trabajo y motivar al resto de equipo.
- **Entendimiento:** se refiere a tener un autoconocimiento a nivel personal como de la organización.
- **Acuerdo:** es el compromiso de trabajar en equipo para alcanzar los objetivos propuestos.
- **Respeto:** tener consideración por las personas, respetar sus opiniones y valorar su trabajo.

#### 3.4.3.3 Principios

Los autores (Anderson & Carmichael, 2017, pp. 9-10) establecen dos grupos de principios:

1. Principios de Gestión del Cambio: son utilizados para minimizar la resistencia al cambio.
    - "Empiece con lo que haces ahora: comprender los procesos actuales, tal como se practican realmente, y respetar los roles, responsabilidades y cargos existentes".
    - "Acuerde buscar la mejora a través del cambio evolutivo".

- "Fomente los actos de liderazgo en todos los niveles, desde el contribuyente individual hasta la alta dirección".

2. Principios de Servicio de Entrega: están centrados en entregar valor al cliente.
   - "Entender y enfocarse en las necesidades y expectativas del cliente"
   - "Gestiona el trabajo; permita que la gente se auto-organice".
   - "Desarrollar políticas para optimizar los beneficios del cliente y del negocio".

#### 3.4.3.4 Proceso

Kanban no sigue un proceso específico de actividades, pero si ofrece directrices sobre cómo realizar el trabajo. Tienen un flujo iterativo, incremental y evolutivo. Utiliza medios visuales, como una pizarra con tarjetas o un tablero electrónico para mostrar y hacer seguimiento a cada una de las tareas del proyecto. La pizarra o tablero *(Véase figura 3-2)* se divide en varias columnas las cuales representan diferentes etapas por las que puede pasar una tarea, por ejemplo: en espera, en análisis, en desarrollo, en prueba o entregado (las etapas pueden variar en cada organización según el tipo de proyecto). En la primera columna se ubica la Pila del Producto con todas las historias de usuario que deben ser completadas. Las historias de usuario se representan en tarjetas o notas que son colocadas en el tablero Kanban. Estas tarjetas tienen una breve descripción de la tarea, su prioridad asignada, estimación de esfuerzo, criterios de validación u otra información que el equipo de desarrollo considere importante.

Kanban utiliza un sistema "Pull" o de arrastre (Anderson, 2010), en el que solo se trabaja bajo pedido. En el caso de un proyecto software, antes de iniciar una iteración el cliente crea y selecciona las historias de usuario que deben ser entregadas al final del ciclo. El equipo de desarrollo trabaja únicamente según en lo que el cliente ha solicitado.

#### 3.4.3.5 Roles

El método Kanban carece de roles obligatorios, sin embargo Anderson (Anderson & Carmichael, 2017, p. 33) ha identificado que durante la implementación de Kanban surgen dos roles en el proceso y son:

- **El gestor de solicitudes de servicio:** es el responsable de comunicarse con el cliente y comprender los requisitos de negocio. Durante la reunión de reabastecimiento, ayuda a seleccionar y priorizar las historias de usuario para cada iteración.
- **El gestor de entrega de servicio:** es el responsable de asegurar que se realice la entrega de las historias de usuarios seleccionadas por el cliente y organiza la Reunión Kanban y la Planificación de Entrega.

#### 3.4.3.6 Prácticas

Las seis prácticas básicas del método Kanban son (Anderson & Carmichael, 2017, pp. 17-26):

**Visualización del Flujo de Trabajo:** en Kanban es fundamental que la información el trabajo esté accesible a los interesados del proyecto para que conozcan que su estado actual, observar el progreso de cada tarea y asegurar que la carga de trabajo sea adecuada. La información reflejada en un tablero Kanban ayuda al equipo a tener una mayor comprensión del trabajo y tomar decisiones para mejorar la coordinación.

**Limitación del Trabajo en Progreso:** en cada etapa del proceso (columna del tablero) se asigna una cantidad específica de tareas que el equipo de desarrollo es capaz de realizar. A este límite de tareas se le llama WIP (del inglés Work In Progress). Esta práctica mejora la organización del trabajo, ya que la cantidad de tareas a realizar es controlable ayuda a "r reduce los costos de coordinación (menos para coordinar), reduce la multitarea y aumenta la concentración" (Measey, 2015, p. 149).

**Gestión del Flujo:** el objetivo es balancear la carga de trabajo y mantener un ritmo de trabajo constante. El equipo de desarrollo inspecciona el rendimiento del proyecto midiendo el flujo de trabajo. Para ello, Kanban utiliza dos métricas : "lead time" es el tiempo que transcurre desde que el cliente solicita el trabajo hasta que se le entrega y el "cycle time" mide el tiempo desde que se inicia una tarea hasta que finaliza (Anderson & Carmichael, 2017, pp. 49,52).

**Hacer las políticas explícitas: l**as políticas son directrices o procedimientos que el equipo debe seguir en situaciones concretas. Por ejemplo es importante tener respuestas a preguntas como: ¿Cuándo una tarea se considera terminada?, ¿Quién es responsable de actualizar el tablero? o ¿Qué se debe hacer cuando introduce al proceso una tarea que no estaba planificada?.
Las políticas optimizan el proceso porque evitan tiempos de espera, cada persona sabe cómo actuar en determinadas circunstancias. Anderson (Anderson & Carmichael, 2017, p. 22) aconseja que las políticas de proceso deben ser "escasas, simples, bien definidas, visibles, siempre aplicadas y fácilmente cambiables". Todas las políticas que se establezcan para la gestión de riesgos y procesos deben ser documentadas (Measey, 2015, p. 149).

**Implementación de Ciclos de Retroalimentación:** en Kanban se realizan, según las necesidades de retroalimentación, siete reuniones o "cadencias" que facilitan el flujo de información entre los interesados del proyecto: (1) Revisión de la estrategia, (2) Revisión de operaciones, (3) Revisión de riesgos, (4) Revisión de la prestación de servicios, (5) Reunión de reabastecimiento, (6) La reunión Kanban diaria y (7) Reunión de planificación de entregas (Anderson & Carmichael, 2017, p. 25).

**Mejorar colaborativamente, evolucionar experimentalmente:** El objetivo de utilizar Kanban es tener una mejora continua en el proceso para obtener un resultado de máxima calidad. Gracias a la transparencia y los ciclos de retroalimentación implementados, se puede conocer las fortalezas y debilidades del proceso que pueden producir "los cuellos de botella, colas, variabilidad y desperdicios" (Kniberg & Skarin, 2010, p. viii). En respuesta a las debilidades se aplica de forma gradual acciones de mejora que son consensuadas entre los interesados. La colaboración y implementación de las prácticas de Kanban ayudan a optimizar el proceso, aumenta la productividad de las personas y asegura la entrega de valor al cliente.

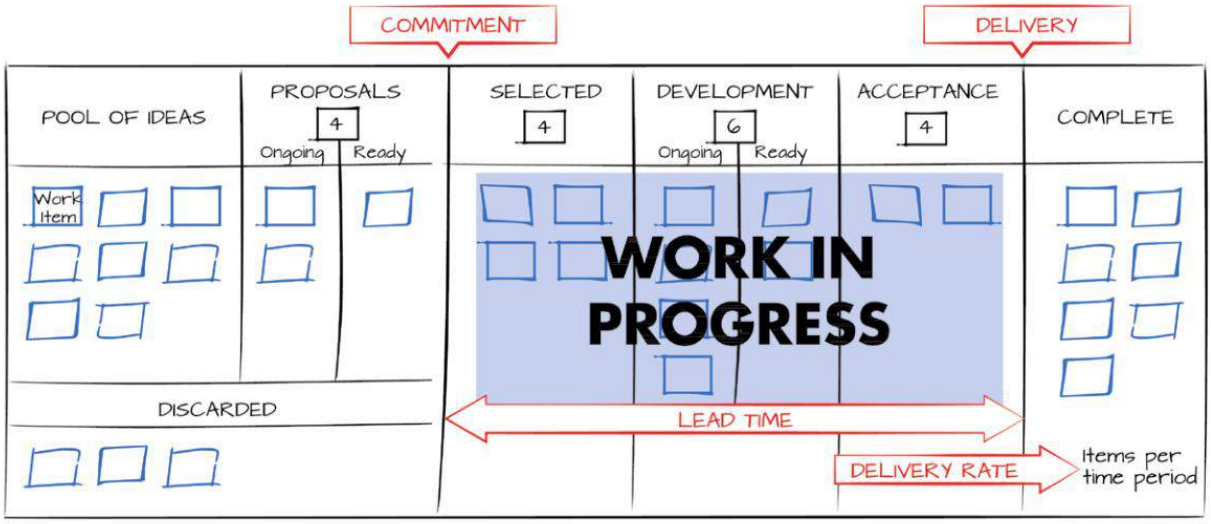


Figura 3-8 Un ejemplo de un tablero Kanban (Anderson & Carmichael, 2017, p. 13).

### 3.4.4 Lean Software Development

#### 3.4.4.1 Definición

Lean Software Development (LSD) es una metodología ágil basada en los principios de Lean Manufacturing[19]. Fue desarrollada y publicada en el 2003 por Mary y Tom Poppendieck. En su libro "Lean Software Development: An Agile Toolkit" los autores explican como adaptaron y aplicación los principios de Lean Manufacturing al desarrollo software.

#### 3.4.4.2 Principios

LSD se basa en siete principios y cada uno se soporta en 22 herramientas (*Véase Anexo 6.2*) que facilitan su implementación.

1. **Eliminar el desperdicio:** En Lean se considera desperdicio a toda actividad que no añade valor al producto. El término de valor va asociado a las expectativas deseadas por el cliente. El proceso de desarrollo software debe estar enfocado a comprender ampliamente lo que el cliente necesita y realizar una entrega temprana. Toda actividad que impida o ralentice ese proceso debe ser eliminado (Poppendieck & Poppendieck, 2003). Por ejemplo, algunos

[19] Lean Manufacturing es una filosofía japonesa de organización del trabajo creada por Taiichi Ohno para la empresa Toyota (Womack, et al., 1990) Este enfoque tiene como objetivo optimizar un proceso mediante la eliminación de todas las actividades que no aportan valor al resultado final.

autores (Poppendieck & Cusumano, 2012) estiman que un 40 a 50 por ciento del tiempo de desarrollo es utilizado para identificar y corregir errores.

En el proceso de desarrollo software se identifican una serie de desperdicios[20] que son descritas en la Tabla 3-1.

| Desperdicio | Ejemplos |
|---|---|
| **Inventario - Trabajo parcialmente hecho**<br>• Tareas incompletas que no finalizan alguna fase del ciclo de desarrollo (implementación, integración o prueba).<br>• Puede suponer un obstáculo en el desarrollo de otras tareas.<br>• El mayor problema es la incertidumbre de no saber si va funcionar o no.<br>• Utiliza recursos que se pueden emplear en otras tareas. Si la tarea nunca se pasa a producción es necesario emplear recursos para eliminarla. | • Código incompleto.<br>• Código en espera de QA (Control de calidad).<br>• Especificaciones sin implementar. |
| **Procesamiento adicional - Procesos adicionales**<br>• Trabajo extra que no aporta valor o que puede ser pospuesto.<br>• Si una documentación aporta poco valor, pero es necesaria, existen tres recomendaciones: Ser Breve, de alto nivel y Offline.<br>• Es recomendado elegir un formato de tabla o plantilla para los documentos, lo cual simplifica la información y optimiza la validación y comprensión.<br>• Un buen indicador de que el documento aporta valor, es si existe una tarea que dependa de el para ser realizada. | • Documentación excesiva o innecesaria.<br>• Protocolos innecesarios de aprobación. |
| **Sobreproducción - Características adicionales**<br>• Características o funcionalidades implementadas no necesarias.<br>• Las características adicionales tienden en gran medida a la obsolescencia incluso antes de ser usadas. | • Nuevas capacidades técnicas |

[20] Después se introdujo "Reaprendizaje" y "Traspaso" definidos como: "Reaprendizaje es redescubrir algo que ya conocíamos y hemos olvidado" y Traspaso es la pérdida de conocimiento tácito cuando un trabajo es traspasado a otros" (Alahyari, et al., 2019).

| | |
|---|---|
| **Transporte - Cambiar de tarea**<br>• Asignar personas a múltiples proyectos produce ineficiencia y bajo rendimiento. Cada cambio de tarea genera un desperdicio significativo de tiempo debido a la recopilación de información de la nueva tarea.<br>• Pertenecer a múltiples equipos dificulta la comunicación y coordinación de actividades por las interrupciones de los cambios de tarea.<br>• Se debe evitar trabajar en varios proyectos de forma simultánea que comparten recursos. Lo ideal es trabajar en un solo proyecto. | • Realizar múltiples tareas de forma simultánea.<br>• Asignar múltiples proyectos a un equipo |
| **Espera – Espera / Retrasos**<br>• Las esperas son uno de los mayores problemas a los que se enfrenta un desarrollo software. Esto produce un retraso que puede afectar el cumplimiento del cronograma del proyecto, sobretodo, impedir que el cliente reciba valor lo antes posible. | • Retraso por bajo rendimiento del equipo.<br>• Retraso en la elaboración o actualización de la Pila del Producto.<br>• Retrasos en procesos administrativos. |
| **Movimiento – Movimiento**<br>• Es el esfuerzo que supone a un desarrollador obtener ayuda para resolver una necesidad.<br>• El movimiento también se produce en otros elementos como los requerimientos, los documentos o el código. Cada vez que un elemento se traspasa a otra persona se produce desperdicio del conocimiento tácito. Aunque se procura que a través de un documento se informe del trabajo realizado o a realizar, este elemento puede carecer de información necesaria para el receptor. | • La respuesta a una consulta.<br>• La solución a un problema técnico.<br>• Obtener los resultados de las pruebas.<br>• Contactar con el cliente o representantes del cliente. |
| **Defecto – Defecto**<br>• Es una no conformidad detectada en un requisito que puede afectar el cumplimiento de las expectativas del cliente. | • Defectos en requisitos<br>• Errores de código |

Tabla 3-1 Desperdicios de Lean Software Development. Adaptada de (Alahyari, et al., 2019; Griffiths, 2011; Poppendieck & Poppendieck, 2003) .

**Herramientas utilizadas:** Identificar desperdicios y Mapeo del flujo de Valor.

2. **Aumentar el aprendizaje:** el proceso de desarrollo software es un proceso de aprendizaje. A medida que el proyecto avanza, aumentan los conocimientos acerca del software en

desarrollo gracias al trabajo. Todos los miembros del equipo aportan ideas, experiencias y conocimientos para mejorar el proceso y cumplir con las expectativas del cliente.

**Herramientas utilizadas:** retroalimentación, iteración, sincronización y set-based development.

3. **Decidir lo más tarde posible:** en el desarrollo software es poco probable acertar con las decisiones tempranas, esto es debido a la incertidumbre provocada por el entorno volátil que rodea la tecnología, a los cambios en el mercado, a la continua evolución del producto y, principalmente, a que el cliente no dispone de toda la información necesaria para la implementación (requisitos, arquitectura, diseño, tecnología). Por ello LSD propone tomar las decisiones lo más tarde posible, es decir, esperar hasta disponer de la suficiente información.
   **Herramientas utilizadas:** Options Thinking, El Último Momento Responsable y Tomando decisiones.

4. **Entrega lo más rápido posible:** LSD fomenta la entrega temprana de valor al cliente. El ciclo de desarrollo iterativo permite que el cliente reciba de forma continua pequeños incrementos del software. Realizar entregas frecuentes ayuda a disminuir el nivel de riesgo, crear más conocimientos, detectar errores y asegurar la calidad del software.

   **Herramientas utilizadas:** Sistemas pull, Teoría de Cola y Coste de Retraso.

5. **Empoderar al equipo:** empoderar al equipo significa delegar liderazgo, esto motiva a las personas a tener un mayor compromiso y participación sobre el proyecto. Las personas motivadas generalmente son más creativas, proponen mejores ideas. Un equipo que se siente comprometido con el proyecto, reconoce que sobre ellos recaen la responsabilidad de la entrega de valor al cliente y se esfuerzan para conseguirlo (Poppendieck & Cusumano, 2012).

   **Herramientas utilizadas:** Autodeterminación, motivación, liderazgo y experiencia.

6. **Construir con integridad:** El usuario debe encontrar el producto como una solución coherente que resuelve las necesidades mediante procesos simples e intuitivos, esto supone que el producto tiene integridad percibida.

   Un producto tiene integridad conceptual si los diferentes procesos y conceptos del sistema funcionan en conjunto de forma coherente y armónica. Esto repercute de forma directa en la integridad percibida por los usuarios.

   La integridad de un software también se mide por aspectos técnicos, como la arquitectura, usabilidad, capacidad de respuesta, mantenibilidad, adaptabilidad y eficiencia con el paso del tiempo. Estos aspectos están directamente relacionados con la calidad, por ello es prioritario adoptar buenas prácticas para reducir la posterior optimización o corrección.

   Para diseñar soluciones con integridad es fundamental una comunicación efectiva con los interesados para comprender las necesidades desde una perspectiva global, en lugar de

recibir requisitos de forma independiente y secuencial. También es importante contar con miembros que tengan capacidad de liderazgo y un equipo con experiencia y disciplinado.

**Herramientas utilizadas:** Integridad Percibida, Integridad Conceptual, Refactorización y Pruebas.

7. **Ver el conjunto:** LSD invita a observar de manera global todo el proceso y no solamente enfocarse en las actividades que intervienen directamente en el desarrollo del software. El flujo de valor comienza desde el cliente y termina con él. Es importante analizar todos los elementos que participan en el proceso (las personas, herramientas, métodos de trabajo, etc.), sus interacciones y evaluar su rendimiento en conjunto, como un sistema (Poppendieck & Poppendieck, 2003).

   **Herramientas utilizadas:** Medidas y Contratos.

## 3.4.5 Agile Modeling

### 3.4.5.1 Definición

Agile Modeling (AM) es una metodología dedicada al modelado de sistemas y documentación de software basándose en la implementación de prácticas y principios ágiles para obtener resultados efectivos. Fue desarrollada por Scott Ambler y publicada en su libro "Agile Modeling" en el 2002. AM no define un proceso a seguir, más bien, ofrece un conjunto de recomendaciones que ayuden a las personas a diseñar buenos modelos (Ambler, 2002, p. 8). A diferencia de otras metodologías que son implementadas en todo el proceso desarrollo software, AM únicamente se enfoca en el modelado y documentación de algunas actividades del proceso. Tal y como lo explica el autor (Ambler, 2002, p. 10), AM no incluye actividades de programación, pruebas, o de gestión de proyectos. En un proyecto software, AM puede ser adoptada por otra metodología para mejorar su proceso (*Véase Figura 3-6).*

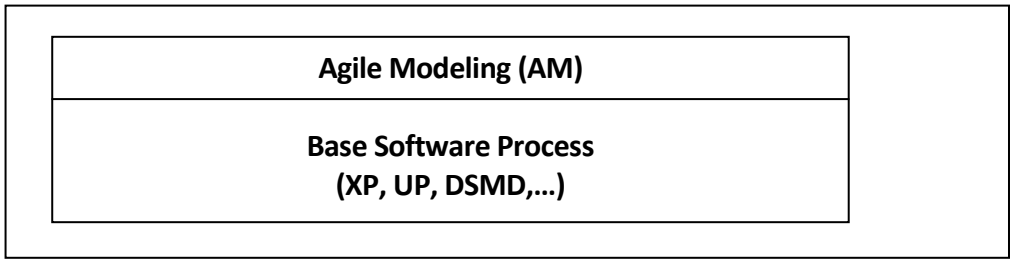


Figura 3-9 AM refuerza otros procesos de software. Obtenida de (Ambler, 2002, p. 10).

### 3.4.5.2 Valores y Principios

Los valores son adoptados de la metodología XP: comunicación, simplicidad, retroalimentación,

coraje y humildad. Este último es el único valor diferente con XP, pero mantiene un significado similar.

Con respecto a los principios, AM los divide en dos categorías (Ambler, 2005): Principios básicos que son necesarios implementar para poder aprovechar los beneficios de esta metodología y principios suplementarios que ayudan a adaptarse a las circunstancias específicas del proyecto.

| Principios Básicos | Principios Suplementarios |
|---|---|
| • Asumir la simplicidad<br>• Aceptar el cambio<br>• El objetivo secundario es facilitar el siguiente esfuerzo<br>• Cambio incremental<br>• Maximizar el ROI de las partes interesadas<br>• Modelar con un propósito<br>• Múltiples modelos<br>• Trabajo de calidad<br>• Retroalimentación rápida<br>• El software funcional es su objetivo principal<br>• Viajar ligero | • El contenido es más importante que la representación.<br>• Comunicación abierta y honesta. |

Tabla 3-2 Principios de Agile Modeling (Ambler, 2005).

#### 3.4.5.3 Prácticas

Ambler (Ambler, 2002, p. 9) expone dos razones por las cuales es importante modelar: tener una mayor comprensión sobre el trabajo que se está desarrollando y/o para facilitar y mejorar la comunicación entre los miembros del equipo o con las partes interesadas del proyecto. Para lograr esos dos objetivos recomienda un conjunto de prácticas que se pueden observar en la Tabla 3-3.

| Prácticas Principales | Prácticas suplementarias |
|---|---|
| • Participación activa de las partes interesadas<br>• Aplicar los artefactos adecuados<br>• Propiedad colectiva<br>• Crear varios modelos en paralelo<br>• Crea contenido simple<br>• Representa modelos simplemente<br>• Mostrar modelos públicamente<br>• Iterar a otro artefacto<br>• Modelar en pequeños incrementos<br>• Modelar con otros<br>• Probarlo con código<br>• Información de fuente única<br>• Utilice las herramientas más sencillas | • Aplicar estándares de modelado<br>• Aplicar patrones suavemente<br>• Descartar modelos temporales<br>• Formalizar modelos de contrato<br>• Actualizar solo cuando duela |

Tabla 3-3 Prácticas de Agile Modeling (Ambler, 2005).

### 3.4.6 SCRUMBAN

#### 3.4.6.1 Definición

Scrumban es una metodología híbrida derivada de la combinación de Scrum y Kanban. Utiliza el marco de trabajo y estructura que ofrece Scrum y añade las actividades de mejora de proceso y elementos visuales de Kanban.

#### 3.4.6.2 Comparativa entre Scrum y Kanban

Ambas metodologías tienen muchas fortalezas, pero “no son ni perfectas ni completas” (Kniberg & Skarin, 2010, p. 7). Por ejemplo, Scrum se enfoca en la entrega constante de historias de usuario y Kanban se esfuerza en mantener un flujo continuo del proceso eliminando cualquier actividad que lo ralentice (Rao, 2017). A continuación, los autores (Kniberg & Skarin, 2010, pp. 51-5 realizan una comparación entre ambas metodologías.

| Scrum | Kanban |
|---|---|
| Las iteraciones deben ser de tiempo fijo. | El tiempo fijo en las iteraciones es opcional. La cadencia puede variar en función del plan del producto y la mejora del proceso. Pueden estar marcadas por la previsión de los eventos en lugar de tener un tiempo pre-fijado. |
| El equipo asume un compromiso de trabajo por iteración. | El compromiso es opcional. |
| La métrica por defecto para la planificación y la mejora del proceso es la Velocidad. | La métrica por defecto para la planificación y la mejora del proceso es el Lead Time (tiempo de entrega o tiempo medio que tarda una petición en salir del ciclo) |
| Los equipos deben ser multifuncionales. | Los equipos pueden ser multifuncionales o especializados. |
| Las funcionalidades deben dividirse en partes que puedan completarse en un sprint. | No hay ninguna prescripción en cuanto al tamaño de las divisiones. |
| Deben emplearse gráficos Burndown. | No se prescriben diagramas de seguimiento concretos. |
| Se emplea una limitación WIP indirecta (por sprint). | Se emplea una limitación WIP directa (marcada por el estado del trabajo). |
| Se deben realizar estimaciones. | Las estimaciones son opcionales. |
| No se pueden añadir tareas en medio de una iteración. | Siempre que haya capacidad disponible, se pueden añadir tareas. |
| La pila del sprint pertenece a un equipo determinado. | Varios equipos o personas pueden compartir la misma pizarra Kanban. |
| Se prescriben 3 roles (PO/SM/Equipo). | No hay roles prescritos. |
| En cada sprint se limpia el tablero de seguimiento. | El tablero Kanban es persistente. |
| La pila del producto debe estar priorizada. | La priorización es opcional. |

Tabla 3-4 Comparativa entre Scrum y Kanban (Kniberg & Skarin, 2010, pp. 51-52).

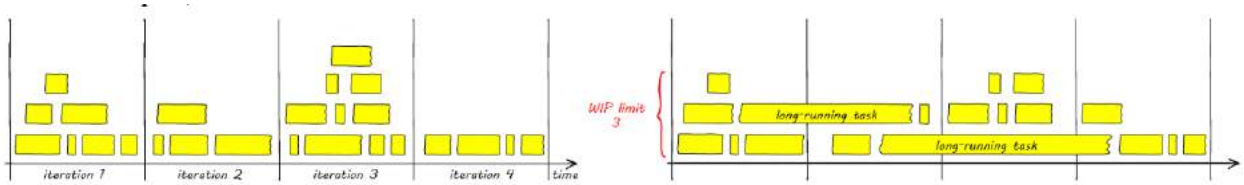


Figura 3-10 Comparativa de tareas en las iteraciones entre Scrum (izquierda) y Kanban (derecha) (Brezočnik & Majer, 2016).

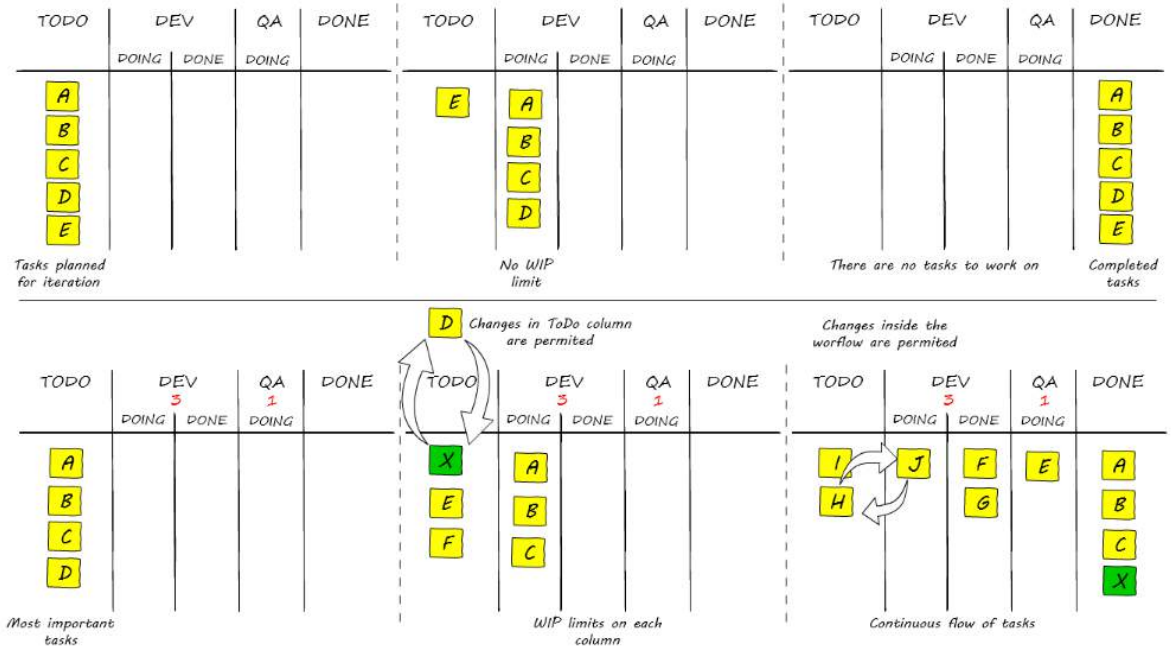


Figura 3-11 Comparativa de los cambios en el plan de trabajo entre Scrum (arriba) y Kanban (abajo) (Brezočnik & Majer, 2016).

### 3.4.6.3 Características

De forma individual, Scrum y Kanban tienen algunas limitaciones. Scrumban en cambio, aprovecha las mejores características (Bhavsar, et al., 2020; Brezočnik & Majer, 2016) de cada metodología:

**De Scrum:**

- Ceremonias: Planificación, revisión y retrospectiva del sprint (reuniones diarias recomendadas pero no obligatorias).
- Sistema Push: las tareas del proyecto son priorizadas y seleccionadas para ser entregadas en cada sprint.

- Sistema Pull: se trabaja bajo pedido. La planificación del trabajo depende de las necesidades del cliente.
- Artefactos: Pila del Producto.
- Iteraciones.

**De Kanban:**

- Límite WIP: limitar la cantidad de trabajo.
- No define roles específicos.
- Se centra en el proceso: mejora del flujo del proceso y reduce el plazo de entrega.
- Uso de tablero: se utiliza un tablero que se reinicia en cada sprint.
- Sistema Pull: las tareas son "arrastradas" en cada estado del proceso.

**De Scrumban** (Brezočnik & Majer, 2016)**:**

- Una característica singular de Scrumban es que utiliza el "bucket size planning" (Teamhood, s.f.) que incluye un plan de trabajo a largo plazo, cuya duración puede variar según las necesidades o el estado del proyecto. Por ejemplo:
  - Bucket de 1 año: se utiliza para que el equipo de trabajo establezca unos objetivos para un proyecto futuro.
  - Bucket de 6 meses: Se desarrolla un plan de trabajo y los requisitos principales se definen con más de detalle.
  - Bucket de 3 meses: la descripción de los requisitos es más completa, en cada uno se especifica el trabajo que se debe realizar (tareas) y cuáles son los objetivos (valor que recibirá el cliente)
  - Bucket actual: las tareas del proyecto están claramente definidas y listas para ser desarrolladas en la primera iteración.
- Permite desarrollar tareas cuya duración abarque más de una iteración, lo que otorga al equipo mayor flexibilidad.
- Dependiendo del tipo de proyecto permite elegir equipos multifuncionales o especializados.
- Acepta cambios durante una iteración.
- Contempla la posibilidad de corregir defectos durante una iteración en caso que tengan un nivel de prioridad mayor que cualquier otra tarea que se esté desarrollando.

## 3.5 Comparación entre las Metodologías Ágiles

Con la información recopilada sobre las diferentes metodologías ágiles, se ha llevado a cabo un resumen sobre sus características. En dicho resumen se decide no incluir Agile Modeling debido a que no cubre el proceso de software en su totalidad centrándose, en prácticas de modelado y documentación (Ambler, 2002, p. 10).

| | SCRUM | XP | KANBAN | LEAN SOFTWARE DEVELOPMENT | SCRUMBAN |
|---|---|---|---|---|---|
| **Representación de Requisitos** | Las historias de usuario. | Historias de usuarios. | Historias de usuario. | Historias, casos de uso, o ítems. | Historias de usuario. |
| **Rol del cliente** | El propietario del software juega el papel principal en la definición de los requisitos. | El cliente in situ debe participar en la definición, estimación y priorización de requisitos. | No especificado. | No tiene un rol específico. Es el encargado de definir los requisitos. | No tiene un rol específico. Es el encargado de definir los requisitos. |
| **Enfoque de Desarrollo** | Enfoque ágil y centrado en el cliente. | Enfoque ágil y centrado en el cliente. | Enfoque ágil y centrado en el cliente. | Enfoque ágil y centrado en el cliente. | Enfoque ágil y centrado en el cliente. |
| **Fases Propuestas** | • Elaboración de la Pila del Producto<br>• Planificación del Sprint.<br>• Ejecución del Sprint<br>• Revisión del Sprint<br>• Retrospectiva del Sprint | • Exploración.<br>• Planificación.<br>• Iteraciones<br>• Producción.<br>• Mantenimiento.<br>• Muerte del Proyecto. | No define fases, si no un flujo de trabajo en el que las tareas pasan por diferentes etapas. | No tiene fases propuestas. | Hereda de Scrum las fases pero permitiendo flexibilidad. |
| **Equipos con Conocimiento Multidisciplinar** | Si. | Parcialmente con roles definidos. | No es obligatorio pero puede incorporar roles adicionales. | Si. | Hereda de Kanban. |
| **Tiempo de Iteración** | De 1 a 4 semanas. | De 1 a 3 semanas. | Trabajo continuo con entregas pequeñas y frecuentes | De 1 a 4 semanas. Permite prolongar su duración. | De 1 a 4 semanas. |

| | | | | | |
|---|---|---|---|---|---|
| **Flexibilidad de requisitos** | Durante la planificación del sprint se permite modificar. | Si. Durante el Juego de Planificación. | Permite añadir historias de usuario en cualquier momento. | Si. Durante la planificación de la iteración se permite modificar. | Hereda de Scrum la restricción de permitir modificaciones durante la planificación. |
| **Roles y Responsabilidades** | Definido Específicamente. | Definido Específicamente. | No definido. | No definido. | No definido. |
| **Centrado en el proceso versus centrado en las personas** | Centrado en las personas (el proceso es importante pero secundario) | Centrado en las personas | Centrado en el proceso | Centrado en el proceso | Centrado en el Proceso (Hereda de Kanban) |
| **Soporte de equipo virtual** | Parcialmente compatible (no está diseñado específicamente para equipos distribuidos, pero puede adaptarse) | No admitido (el equipo debe ser equipos distribuidos de ubicación conjunta). | Parcialmente compatible (se puede adaptar a equipos distribuidos a través de tableros Kanban virtuales y otras herramientas ágiles. | No definido. | Parcialmente compatible (Hereda de Kanban). |
| **Interacción de cliente** | Frecuencia media | Frecuencia alta | Frecuencia media | Frecuencia media | Frecuencia media (Hereda de Scrum) |
| **Tamaño del equipo** | Equipos pequeños (entre 4 a 8 personas) | Equipos pequeños (entre 2 a 10 personas) | No especificado. | No especificado | No especificado (Hereda de Kanban) |

Tabla 3-5 Comparativa entre las Metodologías Ágiles. Adaptada de (Al-Zewairi, et al., 2017; Shaydulin & Sybrandt, 2017; Rojas Pino, 2017; Agile PrepCast, 2013).

## 3.6 Comparación entre Metodologías Tradicionales y Ágiles

El principal propósito de las metodologías agiles es alcanzar la satisfacción del cliente y ofrecer las mejores soluciones, para ello el enfoque plantea una gestión que se diferencia en diversos aspectos de las metodologías tradicionales. En la tabla 3-6 se presenta un resumen de las diferencias más relevantes.

| Metodologías Tradicionales | Metodologías Ágiles |
|---|---|
| Existe un contrato cerrado | Un contrato ágil es flexible e incluye cláusulas específicas para la gestión de cambios |
| Documentación exhaustiva y densa | Documentación mínima y relevante |
| La lista de requisitos se define al inicio | La lista de requisitos está en continua evolución |
| El cliente participa al inicio del proyecto en la gestión de requisitos. Durante la ejecución del proyecto tiene una participación limitada con reuniones periódicas y generalmente en etapas avanzadas. | El cliente tiene una participación activa y está involucrado con el equipo. |
| Planificación rígida y evita su alteración lo máximo posible. | Se esperan y admiten cambios durante el proyecto con naturalidad. |
| Se entrega la planificación del proyecto y en la etapa final se realiza la entrega del producto. | Se realizan entregas tempranas y continuas basadas en las prioridades del cliente. |
| El diseño y arquitectura son definidos en la planificación | El diseño y arquitectura se define y optimiza a lo largo del proyecto. |
| El plan del proyecto incluye una previsión de riesgos y costos, y actividades para su monitoreo y control. | Las iteraciones y revisiones permiten un control continuo del riesgo y costes |
| Prioriza el cumplimiento del contrato | Prioriza la satisfacción del cliente |
| Prioriza la comunicación a través de documentos | Fomenta la comunicación directa y presencial |
| Centrado en los procesos. Las personas se adaptan a los procesos | Centrado en las personas y el trabajo en equipo. Los procesos se adaptan a las personas |
| Control y coordinación de procesos | Control de procesos empíricos: Transparencia, |

| | |
|---|---|
| predictivos | inspección y adaptación |
| Enfocada a equipos de diversos tamaños y admite equipos distribuidos. | Enfocada a equipos pequeños y que se encuentren en el mismo lugar de trabajo. Permite la posibilidad de escalar a equipos distribuidos. |
| Más roles, más específicos | Pocos roles, más genéricos y flexibles |
| Validación al final del proyecto | Validación del incremento en cada iteración |

Tabla 3-6 Comparación entre Metodologías Tradicionales y Ágiles. Adaptada de (Letelier & Penadés, 2006; PMI, 2017)

# 4 PLAN DE IMPLANTACIÓN

## 4.1 Introducción

Diversos estudios y estadísticas han revelado un índice de mejora en diversos aspectos de las organizaciones que adoptan una metodología ágil, esto es gracias a los principios, prácticas y herramientas que facilitan la gestión de los proyectos software. Los resultados han animado a muchas organizaciones a iniciar la adopción de una metodología ágil, el cual es un proceso complejo e implica múltiples desafíos.

Existen diversos factores que las organizaciones deben tener en cuenta a la hora de adoptar una metodología ágil para incrementar las probabilidades de éxito. A continuación se presenta un plan de implantación para guiar a PYMES en este proceso de cambio.

## 4.2 Preparación para adoptar una Metodología Ágil

La adopción de una metodología ágil es un proceso que requiere de una preparación minuciosa para saber cómo afrontar los cambios inevitables y no fracasar durante su ejecución. Para conseguirlo, es importante anticiparse a los problemas y proponer soluciones que faciliten la adaptación y gestión del cambio. Para este trabajo se elige a Scrum como metodología a implantar por ser un marco de trabajo versátil y con características que lo hacen adaptable a la mayoría de las organizaciones.

### 4.2.1 Desafíos

Cuando la organización adopta una metodología ágil para mejorar su proceso de desarrollo software se enfrenta a desafíos y a la implantación de muchos cambios. La gestión de estos cambios es una tarea compleja que requiere mucho esfuerzo y tiempo para lograr los resultados deseados. Es importante prever estos obstáculos y tomar medidas para superarlos. Mike Cohn (Cohn, 2010, pp. 5-9) cinco puntos claves que generan conflictos durante la adopción de Scrum:

1. **El cambio exitoso no es completamente de arriba hacia abajo o de abajo hacia arriba**. Un cambio hacia la agilidad se consigue únicamente con el compromiso y la participación de todas las personas involucradas. Es importante que los directores, jefes intermedios u otras personas con cargos de responsabilidad sean portavoces y practicantes de la agilidad. Su apoyo en esta transición es vital; son los responsables de crear los cimientos para que la metodología ágil se implante eficazmente. Pero estos esfuerzos no servirían de mucho, si las personas que van utilizar directamente la metodología no comprenden o no están convencidos que este cambio puede traer beneficios, por lo que es igualmente importante su completa colaboración.
2. **El estado final es impredecible:** las metodologías ágiles están compuestas por diferentes prácticas y herramientas que se van utilizando y adaptando según las características y

necesidades de las personas y del proyecto. Alistair Cockburn (Cohn, 2010, p. 6) considera a este proceso de personalización como un factor clave para el éxito de adopción pero, es una tarea que requiere de tiempo y experimentación. Es imposible anticipar cómo reaccionarán las personas a los cambios necesarios para convertirse en ágiles. A esto se añade, que estas metodologías se basan en mantener un ciclo de mejora continua, por lo que es muy difícil prever el resultado final del proceso de adopción.

Lo que se recomienda es establecer objetivos a corto y medio plazo, para realizar diversas comparaciones del estado inicial con respecto al actual e ir tomando decisiones sobre cómo mejorar el proceso.

3. **Scrum está en todas partes.** Introducir la agilidad en un proyecto software trae consigo cambios que afectan al equipo de desarrollo y se extiende a otras áreas de una organización e incluso fuera de ella (clientes e interesados) (Boehm & Turrner, 2005). Al afectar a muchas personas, la resistencia al cambio puede ser mucho mayor. De forma directa, los miembros de un equipo de desarrollo se ven afectados ya que deben adoptar principios y prácticas de trabajo que son particulares de las metodologías ágiles. Con respecto a los clientes, su participación debe ser activa y comprometida durante todo el proyecto, esto representa un cambio respecto a la metodología tradicional. Sobre las diferentes áreas de la organización, por ejemplo, un departamento de ventas puede verse afectado ya que debe cambiar su método de negociación con los clientes y realizar un contrato ágil con clausulas de Cambios de Requisitos.
4. **Scrum es dramáticamente diferente:** es un proceso de desaprender para nuevamente aprender. Adoptar una metodología ágil requiere que las personas cambien la forma habitual con la que desempeñan su trabajo. Esto implica cambiar costumbres, realizar nuevas prácticas, utilizar nuevas herramientas, comunicarse de manera diferente, trabajar en equipo, entre otras. El proceso de aprendizaje puede ser difícil para muchas personas, aceptar los cambios y habituarse a ellos requiere de tiempo.
5. **El cambio llega más rápido que nunca.** Una vez iniciado el proceso de implantación de Scrum se experimentan cambios de forma inmediata. Esto puede crear problemas, incluso provocar rechazo por parte de las personas, lo que puede dificultar el proceso de adopción de la metodología. La resistencia al cambio es una reacción natural de las personas ya que pierden la sensación de estabilidad y seguridad causándole estrés, enfado y desorientación.
6. **Las mejores prácticas son peligrosas.** Una vez que se ha adoptado Scrum y los miembros del equipo comienzan a implementar las prácticas ágiles, es importante que evalúen constantemente si el trabajo puede ser realizado de forma diferente para obtener mejores resultados. Asumir que esas prácticas son las mejores puede ser un grave error, ya que lleva al equipo hacia el conformismo. Es responsabilidad del equipo realizar una mejora continua sobre el proceso y uso de estas prácticas.

**Cultura Organizacional**

Además de estos puntos clave, cabe destacar la cultura organizacional como factor crítico que determina el éxito o fracaso de la implantación. La última encuesta de "State of Agile 2020" (VersionOne, 2020) demostró que la cultura organizacional continua siendo el principal desafío que dificulta el proceso de adaptación y escalamiento de una metodología ágil en las organizaciones. Se entiende por cultura organizacional al "conjunto de creencias, supuestos y comportamientos compartidos en la organización. Está reforzado por las normas del grupo" (Combe, 2014). La resistencia al cambio, la falta de liderazgo, la cultura organizacional en contra de los valores ágiles, y el apoyo inadecuado de la gerencia son los principales desafíos derivados de una organización que no está preparada para la implantación.

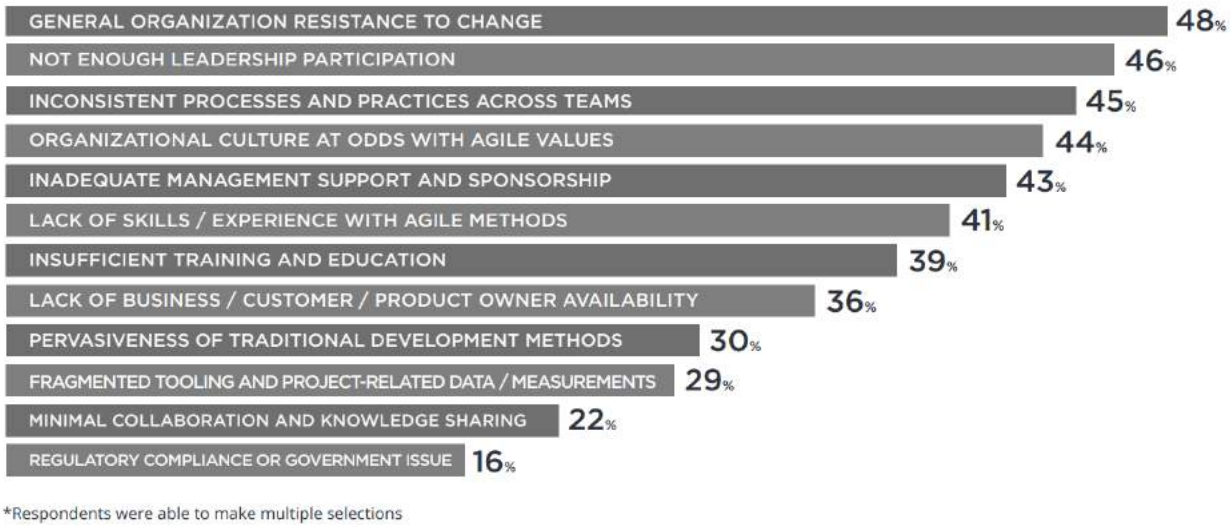


Figura 4-1 Desafíos experimentados al adoptar y escalar Ágil (VersionOne, 2020)

### 4.2.2 Adopción de la Metodología Ágil utilizando el Modelo ADAPT

Los desafíos descritos en la sección anterior demuestran que un proceso de adopción puede fracasar cuando la organización no está alineada al proceso de cambio. Una organización necesita apoyarse en un enfoque o modelo de cambio que ayude a planificar las actividades necesarias para la transición hacia la agilidad. Para la adopción de la metodología SCRUM se propone el modelo ADAPT[21] desarrollado por Mike Cohn.

ADAPT es un acrónimo en inglés y representa cinco actividades:

1. **A**wareness : "Conciencia de que el proceso actual no está generando resultados aceptables".

[21] El modelo ADAPT es una adaptación realizada por Mike Cohn del modelo ADKAR creado por Jeff Hiatt.

2. **D**esire: “Deseo de adoptar Scrum como una forma de abordar los problemas actuales”.
3. **A**bility: “Habilidad para tener éxito con Scrum”.
4. **P**romotion: “Promoción de Scrum a través del intercambio de experiencias para que podamos recordar y otros puedan ver nuestros éxitos”.
5. **T**ransfer: “Transferencia de las implicaciones del uso de Scrum a través de la compañía”.

Según el autor, para que una organización tenga éxito en la adopción de Scrum es importante que el proceso de cambio se realice en cuatro niveles:

1. Organizacional: debe existir una conciencia colectiva para lograr el cambio y la adopción de la metodología.
2. Individual: las personas tienen diferentes ritmos de aprendizaje las personas, por lo que el avance del proceso de adopción puede variar en cada persona.
3. Equipo: los miembros del equipo avanzan juntos en la adopción de la metodología.
4. Prácticas: las personas pueden crear una conciencia de que el método actual de trabajo no es aceptable y desean cambiarlo. Adquieren nuevas habilidades que ponen en práctica y el conocimiento generado es transferido hacia los demás miembros del equipo.

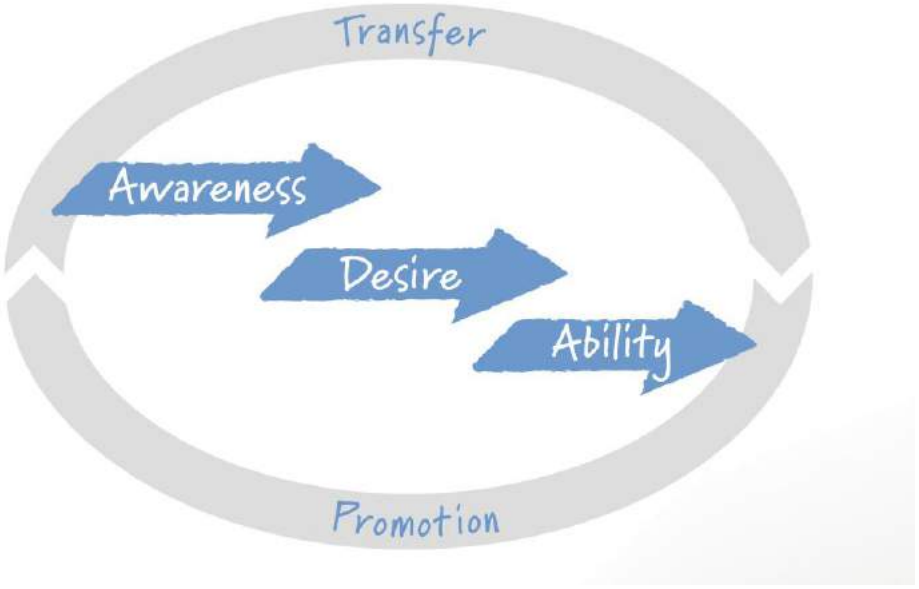


Figura 4-2 Modelo de Cambio ADAPT (Cohn, 2010, p. 22)

#### 4.2.2.1 Conciencia

Un cambio inicia cuando hay conciencia que la situación actual no es aceptable. Cada organización, departamento o equipo debe analizar si los resultados de sus procesos son satisfactorios o más importante aún, si pueden ser mejores. En ocasiones, realizar este auto-análisis puede ser complicado debido a varios factores:

- **Falta de exposición al panorama general:** carecer de la información necesaria para definir el estado actual del proceso.
- **Negarse a ver lo que está justo frente a nosotros:** no dar la suficiente importancia a los resultados negativos argumentando factores puntuales o pasajeros.
- **Confundir el movimiento con el progreso:** el cumplimiento de las tareas diarias puede dar la sensación de que las cosas caminan bien. Esto no se puede confundir con la oportunidad de realizar el trabajo de una mejor manera y obtener resultados con un nivel alto de calidad.
- **Escuchar nuestra propia propaganda:** resultado del conformismo, la complacencia y autoengaño. La ausencia total de tener un criterio objetivo que identifiquen las debilidades y en consecuencia, las amenazas de mantener un estado pasivo sin ánimos de mejorar.

Existen herramientas que son útiles para evitar los puntos anteriores y que ayudan al desarrollo de la conciencia:

- **Comunicar que existe un problema:** es fundamental que las personas sean capaces de identificar aquellos factores que dificulten el rendimiento de su trabajo y les impida obtener mejores resultados. Estos debe ser comunicados de forma inmediata para agilizar su resolución.
- **Utilizar métricas:** se considera una de las mejores herramientas para demostrar la necesidad de hacer un cambio. Proporcionan la información necesaria para comparar el estado actual con el deseado (avance del proyecto, velocidad del equipo en una iteración, tiempos de espera, bugs, etc).
- **Promover el contacto con nuevas personas y experiencias:** motivar a las personas a investigar e implementar nuevas técnicas, asistir a capacitaciones o conferencias, fomentar el contacto con nuevas personas para intercambiar conocimientos y experiencias. Promover estas prácticas es una estrategia que amplía la perspectiva de las personas para que evalúen su método de trabajo y valoren mejores alternativas.
- **Ejecutar un proyecto piloto:** una de las mejores formas de demostrar que una metodología ágil puede funcionar es ejecutando un proyecto que tenga éxito. Esta es una práctica que sin duda alguna convencerá a los escépticos que no creían en los beneficios de un enfoque ágil.
- **Centrar la atención en las razones más importantes para cambiar:** seguramente exista una lista larga que argumente la necesidad de realizar un cambio, lo más recomendable es enfocarse en aquellas que son clave y críticas en el proceso (insatisfacción del cliente, retrasos en entregas, baja calidad del software, sobrecostes).

#### 4.2.2.2 Deseo

Evaluar el proceso de desarrollo y confirmar que no es aceptable no significa desear el cambio. Desear el cambio es tener conciencia y querer adoptar una nueva forma de trabajar para mejorar esa situación. Muchas personas pueden abrazar esos cambios y adoptar la metodología con facilidad, pero es probable que un grupo, por diferentes razones, se resista o se nieguen rotundamente.

Entre las herramientas que pueden ser útiles para aumentar el deseo del cambio se pueden mencionar:

- **Comunicar que existe una mejor manera de hacer las cosas:** es importante concentrarse en la solución, y comunicar de forma apropiada los beneficios que puede traer consigo la adopción de la metodología ágil frente a los problemas actuales.
- **Crear un sentido de urgencia:** dejar claro que el estado actual tiene que cambiar y que las mejoras deben implementarse lo más pronto posible. Es fundamental transmitir a las personas las consecuencias graves que puede traer al equipo, departamento u organización si los cambios no son realizados a tiempo. Esto puede traducirse en clientes insatisfechos o pérdidas económicas.
- **Generar impulso:** Cohn (Cohn, 2010, p. 28) anima a dedicar el mayor tiempo y esfuerzo en el grupo de personas que aceptan y desean adoptar el cambio. El objetivo es aprovechar el apoyo de este grupo para generar un movimiento con suficiente fuerza que animará a los demás a unirse a él.
- **Conseguir que el equipo lleve Scrum a una prueba de manejo:** para facilitar el proceso de adopción es conveniente permitir al equipo experimentar de forma breve con la metodología. Al finalizar con la experiencia, se debe llevar a cabo una reunión de retrospectiva para que el equipo decida como van a trabajar los siguientes meses. Es probable que en esta reunión algunos estén satisfechos con los resultados y otros no se habrán convencido. De ser así, se puede prolongar el periodo de prueba.
- **Concentrarse en abordar el miedo:** las personas pueden tener diferentes razones para resistirse al cambio: experiencias negativas, inseguridad de sus capacidades o las del equipo, entre otras. Es necesario incentivar la comunicación para conocer las expectativas de cada persona y sus miedos frente al cambio, y ofrecer propuestas sobre cómo afrontarlos y superarlos.

**Otras herramientas que se pueden utilizar:**

- Alinear los incentivos (o al menos eliminar aquellos que desmotivan).
- Ayudar a la gente a soltarse.
- No desacreditar el pasado.
- Involucrar a los empleados en el esfuerzo.

- Involucrar a los escépticos también.

#### 4.2.2.3 Habilidad

Las personas necesitan aprender a ser agiles y comprender los valores y principios en los que se basa la agilidad. Al mismo tiempo, deben hacer a un lado sus prácticas habituales de trabajo. Este proceso incluye tres desafíos importantes de aprendizaje: nuevas habilidades técnicas, pensar y trabajar en equipo y crear software funcional en ciclos cortos (Cohn, 2010, p. 31). Para facilitar esta actividad se mencionan las siguientes herramientas:

- **Proporcionar orientación y entrenamiento:** el Equipo de desarrollo, el Scrum Master y el Dueño del Producto deben tener los conocimientos necesarios para ser un equipo ágil. Una adopción exitosa de cualquier metodología ágil requiere de una formación adecuada impartida por personal capacitado y con experiencia. Se recomienda que la organización lleva a cabo una formación inicial y posteriormente ofrecer capacitaciones para desarrollar habilidades más específicas.
- **Responsabilizar a las personas de las habilidades aprendidas:** establecer un compromiso para que las personas pongan en práctica los conocimientos adquiridos.
- **Compartir información:** motivar al equipo para que comparta sus conocimientos, experiencias, dudas, desafíos u otra información de interés para los demás. Fomentar reuniones o talleres internos para que las personas puedan compartir habilidades que hayan aprendido recientemente**.**
- **Establecer objetivos razonables:** los objetivos deben traducir en metas realistas y alcanzables en periodos cortos para motivar a las personas a continuar con el proceso de aprendizaje.
- **Simplemente hacerlo:** animar a las personas a que se atrevan a experimentar con el conocimiento adquirido. Como bien se dice “La práctica hace al maestro”, es importante insistir a las personas que no tengan miedo a equivocarse, toda acción genera más conocimientos sobre cómo hacer o no las cosas. Algo que siempre se debe recordar es que la mejora continua es una de las bases de las metodologías ágiles y hay que ponerlo en práctica en todo momento.

#### 4.2.2.4 Promoción

Una vez que las personas han tomado conciencia del cambio, comprobado y expresado su deseo de implementar mejoras y se han capacitado para realizar esos cambios con éxito, entonces es buen momento para realizar tres actividades importantes:

1. Planificar el siguiente paso para seguir implementando mejoras. La experiencia y casos de éxito obtenidos hasta ese momento facilitará el camino para la promoción de la metodología ágil adoptada.
2. No descuidar los avances alcanzados hasta ese momento. La motivación al cambio debe ser reforzada continuamente, sobre todo en las personas que ya están utilizando la metodología y

comienzan a tener resultados favorables.

3. Crear conciencia en aquellas personas que no han adoptado la metodología. La adopción exitosa de una metodología ágil en una organización se obtiene cuando de manera conjunta todas las personas comprenden y participan en el proceso de cambio.

Las herramientas que pueden ser utilizadas para la promoción de la metodología ágil son:

- **Dar a conocer las historias de éxito:** Exponer la experiencia, desafíos y logros alcanzados durante el proceso de implantación. Esta es una buena herramienta para informar a las personas sobre cómo se está implementado la metodología y que puedan comprobar los beneficios a través de resultados verídicos. Cuando las personas escuchan a sus propios compañeros hablar sobre los buenos resultados que han conseguido, será más fácil persuadir a los escépticos.
- **Organizar un safari ágil:** ofrecer la oportunidad para que otras personas se integren por periodos cortos a los equipos ágiles y trabajen junto con ellos utilizando las herramientas y prácticas ágiles. Esta experiencia puede aportar "de golpe" información muy valiosa, ya que las personas comprueban por sí mismas que todo lo que se ha dicho en las reuniones, formaciones, o lo que se rumorea en la organización es cierto**.**
- **Atraer la intención y el interés:** un "Safari ágil" puede resultar comprometedor para algunas personas. Una alternativa es invitarlas a conocer el lugar de trabajo de un equipo ágil. Esto les permite ser espectadores y observar la dinámica y su entorno. Se debe crear un ambiente de confianza que fomente la participación y se sientan libres de hacer preguntas ya que el objetivo es aclarar dudas para que desaparezcan ideas erróneas preconcebidas. Los espacios de trabajos ágiles son especiales: tableros con tarjetas y gráficos, mesas grandes de trabajo para el equipo, zona de reuniones diarias, y otros elementos pueden ser atractivos para las personas que solo han conocido los métodos tradicionales de trabajo. Esto puede despertar en ellas la curiosidad de aprender algo nuevo que "al parecer funciona".

#### 4.2.2.5 Transferir

La transferencia significa llevar la metodología ágil fuera del equipo o departamento de desarrollo hacia otras áreas de la organización. Este último paso del ciclo ADAPT es muy importante ya que asegura el éxito de la adopción de la metodología. Para ello, es importante citar a Cohn cuando asegura que "Es imposible que un equipo de desarrollo permanezca ágil por sí solo de forma permanente. Si las implicaciones del uso de Scrum no se transfieren a otros departamentos, la gravedad organizacional de esos departamentos eventualmente paralizará y matará el esfuerzo de transición. Con esto, no quiero decir que el resto de la organización necesite comenzar a usar Scrum. Lo que quiero decir es que el resto de la organización debe volverse al menos compatible con Scrum" (Cohn, 2010, p. 38).

Existe una sola herramienta que puede ayudar a transferir la metodología a través de la organización y es la comunicación directa con el resto de los departamentos.

Durante el proceso de transferencia es probable encontrarse con grupos o departamentos que presenten una mayor dificultad. Será necesario dedicarles más atención y tiempo para que comprendan el porqué se están realizando los cambios y cuáles son los beneficios para la organización. En estos casos, el objetivo no es convencerlos a tal punto de convertirlos en los nuevos defensores de la metodología ágil, bastaría con que accedan a conocer los principios de la metodología, sus bases y la forma en que se realiza el trabajo. Es importante que estos grupos conozcan como el uso de la metodología en el departamento de desarrollo afectará a sus actividades. Anticiparse a estas dificultades y comunicárselas a los jefes intermedios facilitaría en gran medida a que la transferencia sea realice eficazmente. Por ejemplo un departamento de Finanzas debe asumir que un equipo de desarrollo no pueda ser capaz de realizar una estimación exacta del coste total de un proyecto ya que no tiene una lista de completa de los requisitos que se tienen que desarrollar. Según la información que aporte el Dueño del Producto, se realiza una estimación de los requisitos conocidos, pero se asume que esta cambiará durante el proyecto.

### 4.2.3 Buenas prácticas para adoptar una metodología ágil

En los procesos de adopción de Scrum se ha demostrado que existen patrones que han sido claves para el éxito de su implantación. Mike Cohn (Cohn, 2010, pp. 43-50) menciona alguno de ellos y explicados a continuación.

#### 4.2.3.1 Empezar con poco (Start small) o Ir con todo (Go all-in).

Los expertos en las metodologías ágiles generalmente sugieren que el proceso de adopción se inicie con un proyecto piloto y con uno a tres equipos. Esto permite aprender con la experiencia y posteriormente escalarlo a toda la organización. El éxito de un primer grupo ayuda a promocionar los beneficios obtenidos a través de la metodología. Tras iniciar la implantación en el primer grupo se puede optar por esperar a que éste termine el proyecto para continuar con los siguientes grupos. Otra opción es esperar a que el primer grupo termine dos o tres iteraciones del proyecto para iniciar la implantación en los otros grupos. Esto dependerá de que tan rápido necesite la organización que la metodología sea adoptada.

Como alternativa a empezar poco a poco se puede enfocar la implantación general en el que se incluyan a todos los equipos al mismo tiempo.

**Ventajas de iniciar poco a poco:**

- Requiere menos inversión económica.
- Éxito a corto plazo casi asegurado.
- Disminuye el riesgo de cometer errores que lleven al fracaso de la adopción.
- Es un proceso menos estresante.
- No es necesaria una reorganización drástica en los departamentos.

**Ventajas de iniciar de ir con todo:**

- El compromiso general reduce la resistencia al cambio.
- Evita los desafíos provocados al trabajar con metodologías ágiles y tradicionales de forma simultánea.
- La adopción finalizará en menor tiempo.

**¿Qué opción elegir?**

Si existe un riesgo alto o el coste de fracaso es elevado es más recomendable elegir "poco a poco".

En caso de que la organización requiera de una implantación urgente, en el que factor tiempo sea decisivo se sugiere "ir con todo". Dado el riesgo de esta opción es imprescindible que la organización cuente con personal experto en la metodología.

#### 4.2.3.2 Implantación pública o sigilosa

Cuando una PYME inicia el proceso de implantación debe decidir si hacerlo público en toda la organización o restringir la información exclusivamente al grupo involucrado.

**Ventajas de una implantación pública**

- Aumenta la probabilidad de colaboración.
- Permite una visión del objetivo y fomenta el intercambio de opiniones.
- Permite exponer argumentos y propuestas a escépticos y aliados
- Una exhibición pública aporta transparencia al proceso de transición
- Declara seguridad, compromiso y determinación.
- No limita la capacidad de solicitar ayuda.

**Ventajas de una implantación sigilosa**

- Permite progresar antes de la oposición de los escépticos y evitar detener el proceso.
- Reduce la presión.
- Permite corregir y mejorar antes de publicarlo.

**¿Qué opción elegir?**

La implantación pública se recomienda si existe seguridad y compromiso de la transición, o un interés en superar la resistencia al cambio.

En contraposición, el enfoque sigiloso es sugerido si el proceso de implantación es experimental o no existe un convencimiento total. También es sugerido en circunstancias en las que el apoyo no está garantizado.

### 4.2.4 Proyecto Piloto

El proyecto piloto es una pieza clave en el éxito o fracaso de la implantación, por ello es de vital importancia seleccionar el proyecto adecuado. Se debe considerar que este proyecto será una fuente de información y experiencia para proyectos posteriores.

#### 4.2.4.1 Atributos de un proyecto piloto

Los atributos a tener en cuenta en la selección del proyecto piloto son: el tamaño, duración, compromiso del patrocinador ejecutivo e importancia. Para esto se debe tomar en cuenta lo siguiente:

**Duración:** que el proyecto tenga una duración cercana a la mitad de la duración de los proyectos más comunes en la organización. Por ejemplo, si generalmente los proyectos duran nueve meses, el proyecto ideal debe ser de tres o cuatro meses.

**Tamaño:** proyectos de un solo equipo, incluso si tiene un crecimiento previsto que supondrá incluir a más equipos. La complejidad de la gestión es directamente proporcional al número de equipos, por ello el número de equipos no debe sobrepasar más de lo habitual en la organización.

**Importancia:** el nivel de importancia del proyecto debe ser lo suficiente para demostrar de forma contundente los beneficios, y que los resultados sean tenidos en cuenta por la organización. También se debe considerar que elegir un proyecto de importancia elevada supone un riesgo significativo y añadirá presión a los involucrados.

**Apoyo y compromiso del patrocinador ejecutivo:** el apoyo de un ejecutivo de la organización proporciona seguridad y aumenta la sensación de respaldo durante el desarrollo del proyecto piloto. Es de gran ayuda para superar los posibles obstáculos burocráticos y a la vez facilita el proceso de cambio organizacional.

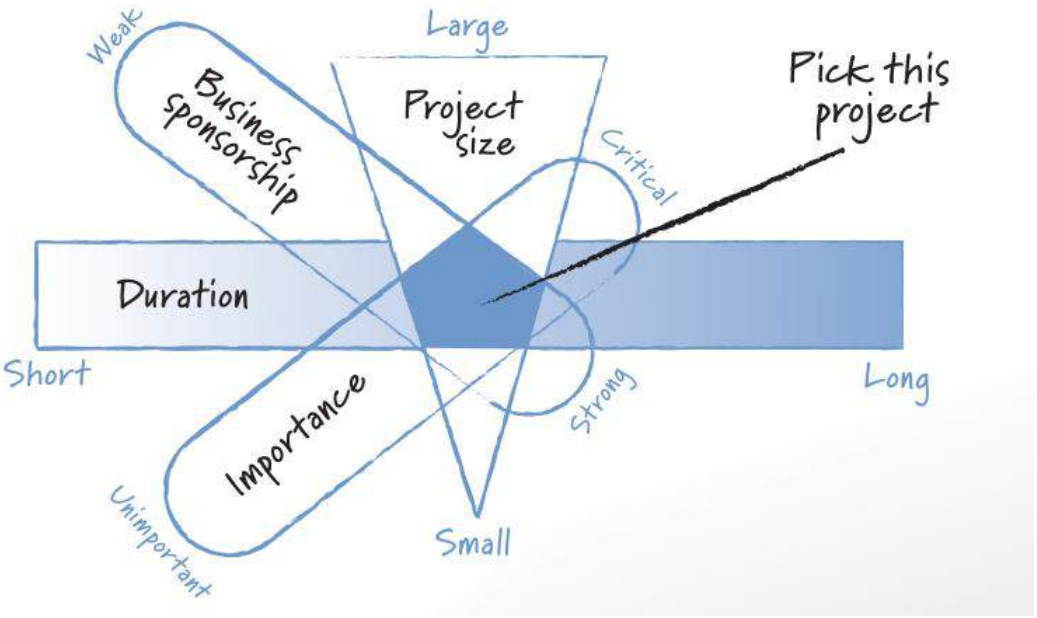


Figura 4-3 Los cuatro atributos para un proyecto piloto ideal (Cohn, 2010, p. 82).

#### 4.2.4.2 Escoger el momento adecuado para iniciar

Se puede adoptar la metodología en un proyecto desde su inicio, esto supone esperar a que se presente el proyecto adecuado y por lo tanto posponer de forma innecesaria las ventajas de la metodología.

También es posible adoptar la metodología en un proyecto que ya está en ejecución, este caso implica asumir un desperdicio de tiempo, pero a su vez puede aumentar la atención del equipo. Proyectos de este tipo pueden ser aquellos que se encuentren en una situación crítica, es decir, que requieren un cambio para evitar el fracaso inminente.

#### 4.2.4.3 Elegir el equipo adecuado.

Un equipo ideal es aquel cuyos miembros están motivados a adoptar la metodología. Según Cohn (Cohn, 2010, p. 88) es preferible un equipo que combine personas de los siguientes grupos:

- **Aliados:** personas que han apoyado la adopción de la metodología.
- **Dispuestos:** comprenden la necesidad del cambio y tienen la voluntad para colaborar.
- **Escépticos:** no están convencidos de la adopción. Añade un esfuerzo adicional durante la ejecución, pero presenta la oportunidad para que cambien de opinión tras la experiencia.

Otra característica ideal es formar grupos con personas que han trabajo juntas obteniendo buenos resultados en proyectos anteriores.

#### 4.2.4.4 Establecer y gestionar las expectativas

La gestión de expectativas es importante porque según la percepción de las personas interesadas

puede suponer que el proyecto ha sido un éxito o un fracaso independientemente de la realidad.

Las expectativas de un proyecto software juegan un papel decisivo en la percepción del resultado, independientemente del valor aportado. Es decir, si un proyecto cumple con los requisitos funcionales encaja en el presupuesto y el tiempo de entrega ha sido satisfactorio

Puede ser considerado un proyecto exitoso sin embargo si las expectativas no eran realistas y sobrepasaban el alcance realista será percibido como un fracaso

Las expectativas en un proyecto software juegan un papel decisivo en la percepción del resultado, independientemente del valor aportado. Un proyecto que cumple con los requisitos funcionales, que no exceden el presupuesto y se ha entregado en plazo, puede ser considerado un fracaso si las expectativas no eran realistas.

Cohn (Cohn, 2010, pp. 88-92)considera que existen cuatro expectativas que se deben establecer y gestionar.

**Progreso:** Uno de los beneficios que aporta Scrum es la entrega continua de valor en periodos cortos, esto puede ser confundido con un incremento en la velocidad de producción de líneas de código y aumento de rendimiento sin esfuerzo. Esta expectativa se puede controlar informando al equipo que ser ágil es un proceso que requiere dedicación, tiempo y capacitación.

Un error común en esta expectativa es generado por una sobreestimación de trabajo en el primer sprint, debido a que no se tiene suficiente experiencia con la metodología. Otro error común es no considerar factores externos que pueden afectar el avance del proyecto.

**Previsibilidad:** En cada iteración de un proyecto se realiza una estimación inicial del trabajo. Se debe informar a los interesados que estas estimaciones no son exactas y tienen un margen de error que depende de diversos factores (experiencia, nuevas tecnologías, rotación del personal). Especialmente en los primeros sprints en los que no se dispone de datos históricos con los que comparar. A medida que avanza el proyecto aumentara la precisión de las estimaciones.

**Actitudes:** La implantación de una metodología ágil requiere de la colaboración de todas las partes interesadas, pero como en todo proceso de cambio, es posible que al inicio de la adopción surja la tentación de regresar a la forma de trabajo previa y conocida, especialmente si no se percibe un beneficio inmediato. Se debe informar al equipo que ante esta situación el objetivo es continuar con Scrum a pesar de las inconformidades y que estas dificultades son previstas y forman parte del proceso de aprendizaje.

**Implicación:** Scrum requiere de una participación activa y constante del equipo. Es importante asegurarse desde el principio que cada parte interesada comprende su nivel de compromiso y participación en el proceso.

## 4.3 Equipo Ágil

Un equipo ágil está formado por un grupo de personas que trabajan juntos por un objetivo en común y están orientados a conseguir el máximo valor del resultado final. El líder del proyecto es el responsable de realizar una adecuada selección de las personas que formarán el equipo. Sus miembros deben poseer habilidades técnicas y sociales necesarias para llevarlo a cabo el proyecto (Highsmith, 2010, p. 52).

La Guía de Práctica Ágil elaborada por el PMI y "Agile Alliance" sugiere que los miembros de un equipo ágil deben cumplir con los siguientes atributos:

| Atributo | Meta |
|---|---|
| Personas dedicadas | • Mayor concentración y productividad.<br>• Equipo pequeño, con menos de diez personas. |
| Miembros de equipo multidisciplinario | • Desarrollar y entregar a menudo.<br>• Aportar valor terminado como un equipo independiente.<br>• Integrar todas las actividades para entregar trabajo terminado.<br>• Proporcionar retroalimentación desde dentro y fuera del equipo. |
| Ubicación en el mismo lugar o que tengan la capacidad de gestionar cualquier desafío de localización | • Mejor comunicación.<br>• Mejor dinámica del equipo.<br>• Intercambio de conocimientos.<br>• Reducción en el costo de aprendizaje.<br>• Poder comprometerse a trabajar con los demás. |
| Equipo mixto de generalistas y especialistas | • Los especialistas proporcionan conocimiento específico y los generalistas proporcionan flexibilidad acerca de quién hace qué.<br>• El equipo aporta sus capacidades especializadas y a menudo se convierten en especialistas generalizados, con una especialidad principal y amplia experiencia a través de múltiples habilidades |
| Entorno de trabajo estable | • Depender unos de otros para entregar.<br>• Llegar a un acuerdo respecto al enfoque del trabajo.<br>• Simmlificar los cálculos de costes del equipo(tarifa de |

| | |
|---|---|
| | ejecución - run rate)[22].<br>• Conservación y expansión del capital intelectual. |

Tabla 4-1 Atributos de los Equipos Ágiles Exitosos (PMI, 2017, p. 40)

### 4.3.1 Roles

#### 4.3.1.1 Scrum Master

El Scrum Master está presente durante todo el proyecto para orientar y ayudar al Equipo de Desarrollo y al Dueño del Producto en sus responsabilidades. Como líder del equipo, el Scrum Master debe tener los conocimientos y habilidades necesarios para conducir al equipo hacia el éxito. Entre sus atributos más importantes destacan (Cohn, 2010, pp. 119-120):

- **Responsable:** es una persona con total disposición y sentido de responsabilidad. Ayuda al equipo a dar lo mejor de sí, y los motiva a adoptar y utilizar la metodología ágil.
- **Humilde:** no busca el protagonismo y mucho menos que reconozcan su intervención en el logro de los objetivos. Es una persona que está dedicada al equipo, busca su excelencia y reconoce el valor que todos sus miembros aportan al proyecto.
- **Colaborativo:** fomenta un ambiente de colaboración y comunicación dentro del equipo. Anima a los miembros del equipo a que sean participativos y que respeten las opiniones de los demás. Los guía a buscar las mejores soluciones que beneficien a todos.
- **Comprometido:** el Scrum Master debe estar disponible desde el inicio hasta el final de proyecto. El equipo debe saber que cuentan con su apoyo y que pueden acudir a él en todo momento. Está comprometido con el resto del equipo a cumplir con los objetivos del proyecto.
- **Influyente**: es una persona con capacidad de persuasión que influye de manera positiva en el equipo de desarrollo, el dueño del producto u otra persona interesada en el proyecto. Su objetivo es convencerles para trabajar de forma colaborativa, basándose en los principios ágiles e implementando las mejores prácticas.
- **Experto**: el Scrum Master debe tener experiencia en liderar equipos ágiles y tener conocimientos en diferentes áreas como gestión de proyectos y programación.

[22] "La velocidad se puede denominar como la tasa de ejecución del equipo. Los equipos ágiles usan esta métrica para medir cuánto trabajo se realizó en un sprint, de modo que sepamos cuánto más queda para explicar la razón" (Agile Practitioners Guide, s.f.).

**Problemas comunes**

Una organización puede enfrentarse a algunos problemas en el proceso de selección de la persona adecuada para este rol o también con el desempeño del Scrum Master (Cohn, 2010, pp. 122-124):

**Necesidad de contratación externa.** Si al iniciar con el proyecto, la organización no cuenta con una persona para cumplir con este rol dentro de su plantilla, será necesario contratar a alguien externo para liderar al equipo. La única desventaja que presenta esta decisión es una posible dificultad para crear vínculos de confianza y colaboración con el equipo, el Dueño del Producto u otras partes interesadas, ya que es una persona ajena a la organización.

**Persona inapropiada para el rol.** Cuando se elige a una persona para este rol y no cumple con los objetivos deseados, la persona responsable del proyecto debe tomar una decisión rápida y oportuna. Existen dos opciones: reunirse directamente con el Scrum Master e indicar aquellos aspectos de su trabajo que es necesario que modifique y ofrecer sugerencias o, definitivamente sustituirlo por otra persona más adecuada.

**Cuando la persona cumple otro rol en el equipo (desarrollador, tester, o líder de otro equipo):** en ciertas organizaciones y según el tipo de proyecto, el Scrum Master puede cumplir otro rol dentro del equipo de desarrollo. Esto puede traer algunos problemas ya que su tiempo se ve limitado para ejercer de forma adecuada su rol como líder del equipo. En este caso se debe intentar que sus responsabilidades no estén relacionadas con las tareas que forman parte de la ruta crítica del proyecto, porque podrían verse afectadas (Cohn, 2010, p. 124). Otro riesgo es que no tenga una visión completa e imparcial del trabajo del equipo y afecte su criterio para la toma de decisiones. Es importante que el líder del equipo sea capaz de separar ambos roles en cada momento. Se puede presentar una situación similar si el Scrum Master pertenece a múltiples proyectos. Adicionalmente puede suponer un aumento en el desperdicio generado por la multitarea.

**Cuando el Scrum Master tome decisiones por el equipo**: esto puede pasar cuando el equipo de desarrollo no tiene la suficiente experiencia de autoorganización y necesite que otra persona sea quien tome las decisiones. El Scrum Master debe recordar que su rol es de facilitador, ofrece orientación si existe algún problema, pero debe motivar a los miembros del equipo a que tomen las decisiones por ellos mismos.

#### 4.3.1.2 Dueño del Producto

El Dueño del Producto es el responsable de dirigir al Equipo de Desarrollo y asegurar que tenga la información necesaria para que entregue un producto que cumpla con las expectativas del cliente. Debe hacer énfasis en dos actividades muy importantes: proporcionar Visión y Límites (Cohn, 2010, pp. 125-127).

**Visión:** es fundamental que el Dueño del Producto defina y comparta correctamente la visión del producto con el resto del equipo por ejemplo, las necesidades de los usuarios finales o la diferenciación del producto con respecto a otros en el mercado. Esto conduce a un mejor entendimiento y comunicación para ambas partes.

La visión del producto tiene que verse reflejada a través de la Pila del Producto con historias de usuario bien definidas, detalladas y priorizadas. El Dueño del Producto es responsable de su gestión y actualización.

**Establecer límites:** el Dueño del Producto debe establecer y comunicar las restricciones del producto. Por ejemplo, una fecha límite y especificaciones técnicas como "funcionalidad mínima requerida, rendimiento dramáticamente más rápido, consumo reducido de recursos" (Cohn, 2010, p. 128). El equipo enfocará sus propuestas y organizará su trabajo basándose en estos límites.

En cuanto a fechas de entrega, es importante aclarar que el Equipo de Desarrollo será siempre el responsable de estimar y establecer la fecha en que pueden entregar los incrementos. Sin embargo, el Dueño del Producto debe informar cuál es la fecha máxima de entrega del Proyecto para aquellas funcionalidades que son más urgentes de desarrollar. Aquí se inicia un diálogo o más bien una negociación en el que ambas partes deben consensuar sobre la cantidad de trabajo y el tiempo disponible. Cuando el Dueño del producto establezca límites de fechas de entrega tiene que establecer un tiempo suficiente para no desmotivar al Equipo a realizar un trabajo que será imposible terminarlo o requerirá de muchas horas extras.

Entre sus atributos más importantes destacan (Cohn, 2010, pp. 130-131):

- **Disponible:** el Dueño del producto debe ser una persona que trabaja activamente con el Equipo de Desarrollo. Los equipos ágiles están comprometidos a la entrega de valor temprana, por lo que su flujo de trabajo tiene que ser constante y sin ningún contratiempo. Si el equipo necesita aclarar dudas, el Dueño del Producto debe estar disponible para dar la información necesaria.
- **Comunicativo:** es una persona con excelentes habilidades para comunicarse con las diferentes partes interesadas en el proyecto. El Dueño del Producto interactúa de manera constante con el equipo del proyecto, los directivos de su organización, jefes de distintos departamento, usuarios finales, cliente, etc. Debe tener capacidad de escuchar y comprender las necesidades, preocupaciones, sugerencias de todas las partes interesadas y transmitirlas eficazmente.
- **Decisivo:** el Dueño del Producto debe resolver con prontitud y seguridad todas las dudas de Equipo de Desarrollo.
- **Empoderado:** debe ser una persona con suficiente autoridad en la organización para tomar decisiones sobre el proyecto.

**Errores comunes**

Entre los problemas más comunes que se puede presentar el Dueño del Producto están:

**Delega la toma de decisiones pero luego anula al tomador de decisiones:** para decisiones sobre temas muy específicos, el Dueño del Producto puede delegar que esas decisiones sean tomadas por una persona experta con conocimientos sobre esa parte del software. El problema se da cuando el Dueño del Producto cambia la decisión que ha delegado, generando confusiones y errores en el trabajo del equipo de desarrollo. El dueño del producto debe asumir que es responsable de las decisiones delegadas, por ello debe asegurarse que el delegado dispone de toda la información.

**Presión excesiva al equipo de desarrollo:** en ocasiones el Dueño del Producto exige las entregas del Software en ciclos demasiados cortos y sobrecarga el trabajo al equipo de desarrollo. El Scrum Master debe trabar junto al Dueño del Producto en una mejor gestión de la Pila del Producto y que durante la planificación del Sprint se establezcan objetivos alcanzables.

**Disminuir la calidad del software:** esta situación se da cuando se quiere entregar cierta cantidad de funcionalidades en una fecha cercana y el tiempo disponible no es el suficiente. Esto se traduce en disminuir los controles y pruebas que aseguran la calidad del software. Las consecuencias se ven en etapas avanzadas, cuando es necesario detener el proceso para corregir defectos del software. Ken Schwaber recomienda que esta decisión sea tomada únicamente por los altos ejecutivos ya que considera que sacrificar la calidad de software es una decisión crítica que puede generar graves consecuencias para el proyecto (Cohn, 2010, p. 133). Para evitar que se llegue a este punto, el Scrum Master debe intervenir y comunicar con tiempo suficiente, la inviabilidad de disminuir la calidad del software. Su responsabilidad es asegurar que el cliente reciba el máximo valor.

**El Dueño del Producto se encuentra en una ciudad diferente a la del equipo de desarrollo:** para que esto no sea un impedimento el Dueño del Producto debe cumplir con lo siguiente:

- Mantener una participación activa en el proyecto.
- Tener una relación directa con el equipo de desarrollo.
- Cumplir con todas sus responsabilidades.
- Estar siempre disponible para el equipo, dedicando una parte de su jornada para contactar con él. De no estarlo, debe responder en la brevedad posible para no dilatar el tiempo de espera.

#### 4.3.1.3 Equipo de Desarrollo

El Equipo de Desarrollo es multidisciplinar, es decir, sus miembros tienen los conocimientos y habilidades requeridas para el desarrollo del software. La composición de este equipo dependerá de las necesidades del proyecto, en su caso habrán: analistas, arquitectos, desarrolladores, adminitradores de base de datos, testers, etc. Aunque que cada miembro tiene mayor experiencia en un área específica, todos tienen conocimientos generales de cada área. Es por esto que no se definen roles específicos dentro del equipo y todos son llamados desarrolladores (Goldstein, 2014, p. 26).

La característica fundamental del equipo de desarrollo es que es auot-organizado y esto representa "el núcleo de la Gestión Ágil de Proyectos" (Highsmith, 2010, p. 51). Al equipo de desarrollo se le otorga autoridad para decidir sobre el trabajo. Esa participación directa genera un fuerte compromiso con el proyecto ya que se sienten responsables de su éxito o fracaso (Cooke, 2014, p. 164). La responsabilidad compartida y el trabajo en equipo unifican fuerzas para entregar el máximo valor en cada iteración.

Crear un equipo auto-organizado no es tan fácil, según Highsmith (Highsmith, 2010, pp. 52,54) se requiere:

- Conseguir las personas adecuadas.
- Articular la visión del producto, los límites y los roles del equipo.
- Fomentar la colaboración.
- Insistir en la responsabilidad.
- Fomentar la autodisciplina a través de :
    - Aceptar la responsabilidad de los resultados.
    - Confrontar la realidad a través del pensamiento riguroso.
    - Participar en una interacción y debates intensos.
    - Trabajar voluntariamente dentro de un marco auto-organizado.
    - Respetar a los compañeros
- Dirección en lugar de control.

### 4.3.2 Espacio de Trabajo

Los equipos ágiles necesitan un espacio adecuado para trabajar de manera colaborativa, que les permita tener una comunicación "cara a cara" y visualizar la información del proyecto. Las metodologías ágiles promueven que los equipos de desarrollo trabajen en un mismo espacio y que dispongan de una zona específica para realizar sus reuniones diarias frente a los tableros del proyecto.

En la actualidad, las organizaciones están diseñando espacios en el que se intenta equilibrar las jornadas en las que es necesario que los miembros del equipo estén trabajando juntos en la misma zona y aquellos momentos en los que los desarrolladores necesitan trabajar concentrados sin interrupciones (PMI, 2017, p. 46) . De igual forma, cuando los equipos se encuentran geográficamente distribuidos, es importante que establezcan su zona de trabajo que es privada, y la zona en la que tendrán las reuniones virtuales.

Los espacios de trabajo de los equipos ágiles deben cumplir con los siguientes requisitos (Scaled Agile, 2019):

- Proporcionar un espacio individual dentro de la zona común que permita hacer las tareas asignadas y a la vez mantenerse en el flujo de trabajo con los demás miembros del equipo.
- Proporcionar espacios para el trabajo continuo y colaborativo.
- Proporcionar espacios para el trabajo individual de alta concentración.
- Ofrecer una zona para las reuniones diarias del equipo y otra para la ubicación de los artefactos utilizados para mostrar la información del proyecto como los tableros, pizarras, monitores, etc.

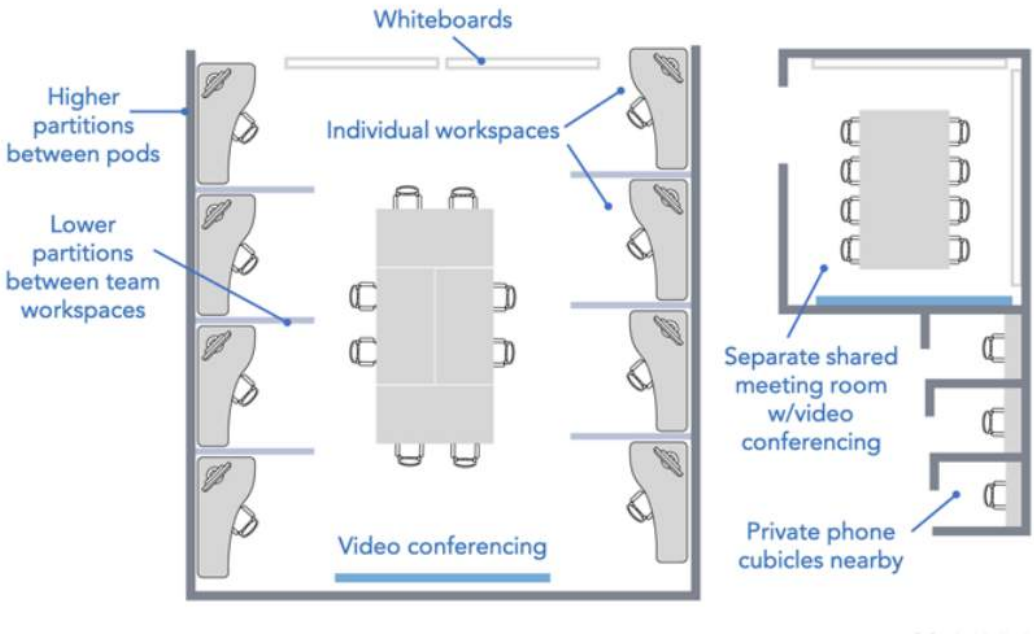


Figura 4-4 Espacio de trabajo ágil (Scaled Agile, 2019).

## 4.4 Contratos Ágiles

Los contratos de proyectos ágiles deben constituir un marco de trabajo que facilite la colaboración entre los involucrados, y sobre todo que fomente una relación basada en la transparencia y confianza (Arbogast, et al., 2012).

Existen diferentes modalidades de contratos ágiles, su organización y cláusulas dependerá si el alcance, el coste y el tiempo son restricciones fijas o variables. Para esta sección se toma de referencia al autor (Albaladejo, 2011) y se mencionan algunos ejemplos de tipo de contratos.

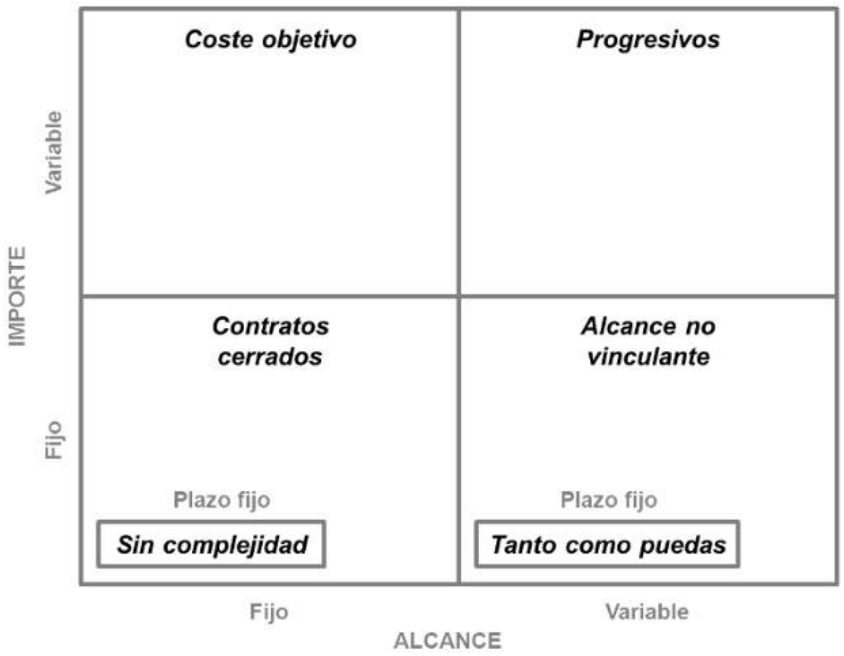


Figura 4-5 Modalidades de contratos en función de las variables que se fijen (Albaladejo, 2011).

### 4.4.1 Contrato con Alcance Fijo e Importe Fijo

Este tipo de contrato restringe la posibilidad de cambios por lo que es adecuado únicamente para proyectos cuya complejidad e incertidumbre son bajas y además sus especificaciones son completas y bien definidas. No se recomienda en los proyectos software debido a que afectan directamente su calidad.

### 4.4.2 Contrato con Alcance Variable e Importe Fijo

En este contrato el cliente puede cambiar los requisitos según sus necesidades. Se reemplazan requisitos que ya se han definido en la Pila del Producto por otros de mayor importancia. Estos cambios deben ir de acuerdo al presupuesto que se ha fijado.

Una situación particular es cuando el importe y plazo de entrega son fijos, en este caso el alcance es variable de acuerdo a la capacidad de entrega del equipo.

Alguna de las cláusulas que se pueden establecer son:

- **Cambio gratis (Change for free):** el cliente puede solicitar cambios siempre que el alcance total especificado en el contrato no sea modificado. En cada iteración el cliente puede intercambiar, sin coste adicional, nuevos requisitos por otros que requieran el mismo esfuerzo (Sutherland, 2008).
- **Finalización anticipada**: cuando el cliente recibe suficiente valor con las funcionalidades

entregadas hasta ese momento o en el caso contrario, que el cliente decida cancelar el contrato debido a que los resultados no eran los que esperaba.

- **Dinero a cambio de nada (Money for nothing):** el cliente puede finalizar el contrato en cualquier momento, pero debe pagar un importe por la cancelación. Es una manera de proteger al equipo del proyecto retribuyéndole por el desarrollo realizado y compensando cualquier daño o perjuicio que pudiera causarle esa decisión (Albaladejo, 2011). Por ejemplo, Sutherland recomienda que se puede establecer un acuerdo para que el cliente pague al proveedor del servicio, el 20% del valor pendiente del contrato (Sutherland, 2008).
- **Pago por tareas adicionales:** corresponde al pago adicional de aquellas tareas que no se especificaron al inicio del contrato y requieren de un esfuerzo extra.

### 4.4.3 Contrato con Alcance Variable y Importe Variable

Este tipo de contrato establece que el pago se realiza según las funcionalidades desarrolladas en cada iteración, existen dos formas de hacerlo:

- **Pago fijo por fase o iteración:** el proyecto se divide en etapas y se establece una cantidad de trabajo para cada una. El pago es fijo en cada iteración.
- **Pago por coste de la fase o iteración (Time & Materials):** el cliente junto al equipo de desarrollo establecen objetivos que se tienen que cumplir en cada iteración. El cliente paga de acuerdo al esfuerzo y los recursos invertidos por el equipo de desarrollo.

### 4.4.4 Contrato con Alcance Fijo e Importe Variable

Este tipo de contrato se establece un coste objetivo por el que el cliente y el proveedor trabajan juntos para conseguirlo. Las ganancias y pérdidas son asumidas por ambas partes. Es un contrato que necesita de una alta colaboración para disminuir el riesgo, aumentar el rendimiento del proyecto y mejorar la calidad del software.

Para establecer el coste objetivo los autores (Arbogast, et al., 2012) recomiendan que:

El Cliente y el Proveedor:

1. Identifiquen, analicen y estimen todos los posibles requisitos del proyecto.
2. Estimen el coste del cambio o aumento del alcance durante el proyecto. Es importante que se tenga en cuenta de manera realista el esfuerzo y los costes.

El Proveedor:

3. A partir de la información anterior, debe establecer el coste objetivo.
4. Calcular el beneficio objetivo, basado en el coste objetivo (por ejemplo, el 15% del costo objetivo).

5. Compartir todos los detalles y resultados con el cliente.

Se toma el ejemplo de (Arbogast, et al., 2012) en el que se acuerda que el 60% de cualquier diferencia en los costes son asumidos por cliente y el 40% le correspondería al proveedor.

| Coste Objetivo | Beneficio Objetivo | Pago Objetivo del Cliente | Coste Actual del Proveedor | Ajuste | Pago Actual del Cliente | Beneficios Actual del Proveedor |
|---|---|---|---|---|---|---|
| 1.000.000 | 150.000 | 1.150.000 | 1.100.000 | +60.000 | 1.210.000 | 110.000 |
| 1.000.000 | 150.000 | 1.150.000 | 900.000 | -60.000 | 1.090.000 | 190.000 |

Tabla 4-2 Ejemplo de un Contrato de Coste Objetivo (Arbogast, et al., 2012).

Tal y como se ve en la Tabla 4-2, en la primera fila se observa un aumento en los coste con respecto al coste objetivo, en este caso ambas partes comparten las perdidas. En la segunda línea, se ve reflejado un ahorro en el coste del proyecto, por lo que tanto el cliente y el proveedor salen beneficiados.

## 4.5 Medición del éxito Ágil

En Scrum se utilizan varias herramientas para medir el trabajo realizado, la velocidad del equipo y valor entregado al cliente. La información obtenida sirve para ayuda a monitorear y tener bajo control el proyecto y, tomar decisiones sobre la dirección del mismo.

Entre las métricas más utilizadas se encuentran:

- Gráfico de Avance (Burn down).
- Gráfico de Producto (Burn up).
- Análisis de Valor Ganado.
- Enfoque Correctivo.

### 4.5.1 Gráfico de Avance (Burn down)

El Gráfico de Avance es un gráfico que muestra el esfuerzo pendiente con respecto al tiempo. Indica la velocidad actual del equipo en un sprint y la compara con la velocidad estimada en la Planificación del Sprint. Cada día, los miembros del equipo actualizan la Pila del Sprint con la información del esfuerzo pendiente de cada tarea reflejando el avance en el gráfico.

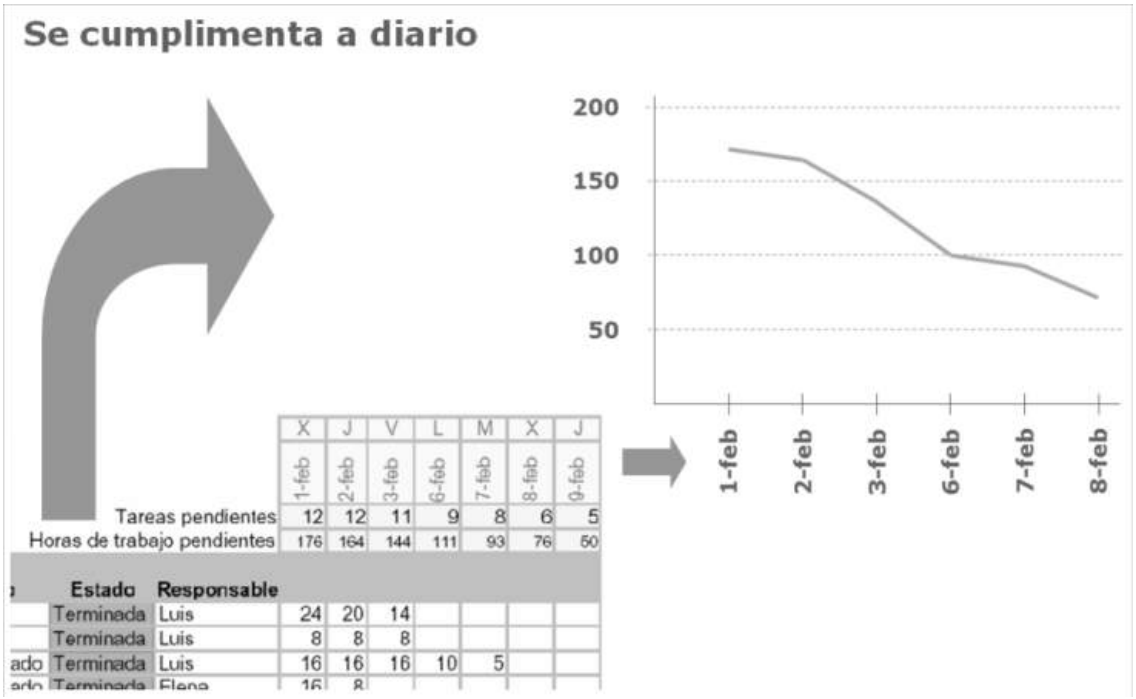


Figura 4-6 De la Pila del Sprint al Gráfico de Avance (Menzinsky, et al., 2016, p. 46).

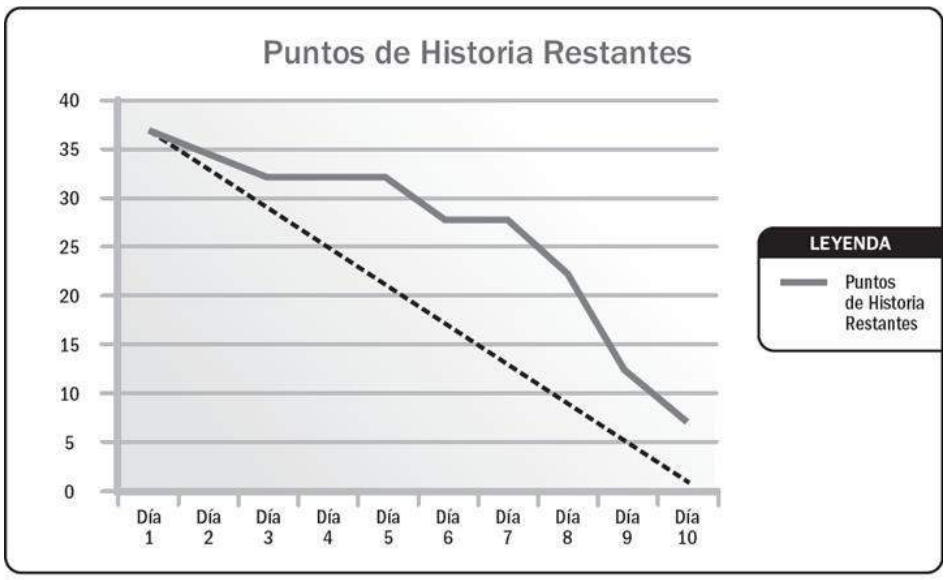


Figura 4-7 Gráfico de Trabajo Pendiente (PMI, 2017, p. 62)

### 4.5.2 Gráfico de Producto (Burn up)

El Gráfico de Producto muestra la evolución del producto con respecto al tiempo. El equipo actualiza los puntos de historia completados que son reflejados en el gráfico como puntos acumulados, es decir, la cantidad de trabajo entregado (Palacio & Ruata, 2016, p. 41).


Figura 4-8 Gráfico de Trabajo Realizado (PMI, 2017, p. 63)

### 4.5.3 Análisis del Valor Ganado

El Análisis del Valor Ganado es una métrica que mide el desempeño durante el proyecto en términos de coste, tiempo y esfuerzo; mide el valor generado a través del avance del proyecto. Se utilizan dos medidas el SPI llamado "Desempeño del Cronograma" y el CPI que es el "Índice de desempeño del Costo".

A continuación se explica un ejemplo de los cálculos de SPI y CPI de un proyecto ágil. Este ejemplo ha sido tomado de (PMI, 2017, p. 69) y tiene los siguientes datos:

- Funcionalidades completas: 25 puntos.
- Funcionalidades planificadas 30 puntos.
- Valor Ganado: $ 2,2 millones millones.
- Costo Real: $ 2,8 millones.

$$SPI = \frac{Funcionalidades\ completas}{Funcionalidades\ planificadas}$$

$$CPI = \frac{Valor\ Ganado}{Costos\ Actuales}$$

- El SPI del proyecto es:

$$SPI = \frac{25\ puntos}{30\ puntos} = 0{,}83$$

El equipo tiene un desempeño actual del 83% según la tasa prevista.

- El CPI del proyecto es:

$$CPI = \frac{\$\ 2{,}2\ \text{millones}}{\$\ 2{,}8\ \text{millones}} = 0{,}79$$

Este resultado muestra que por cada invertido, 79 céntimos de dólar se convierten en valor para el producto.
El Análisis del Valor Ganado proporciona información valiosa para los interesados del proyecto, ya que mide de forma constante la tasa de entrega de valor.

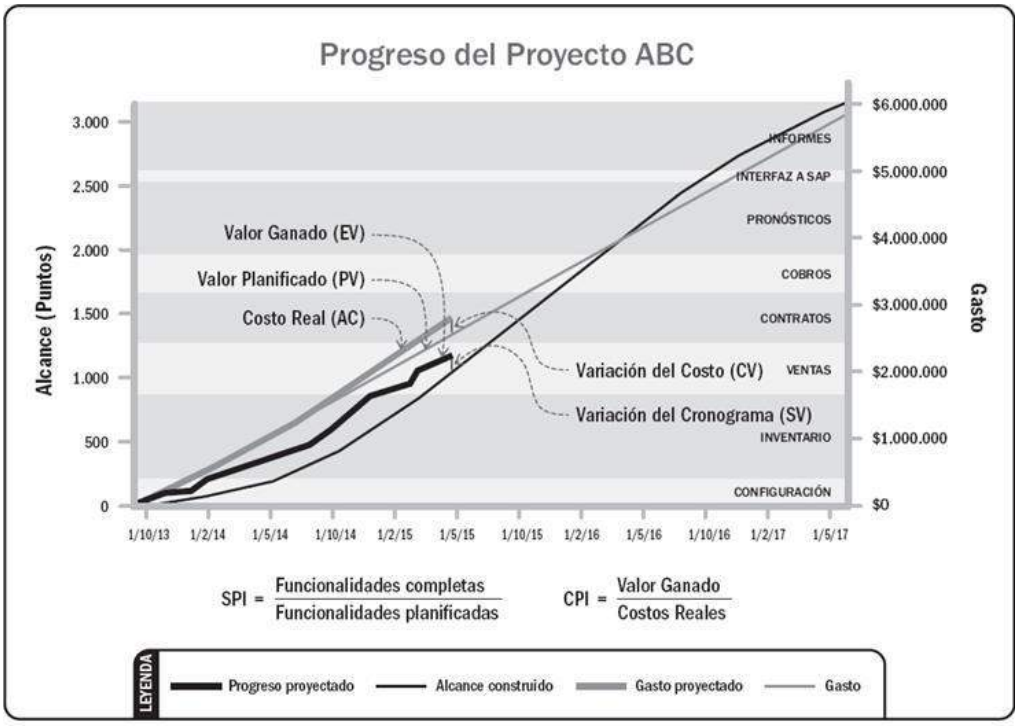


Figura 4-9 Gráfico de Valor Ganado en un Proyecto Ágil (PMI, 2017, p. 69)

### 4.5.4 Enfoque Correctivo

El Enfoque Correctivo es una métrica de calidad que mide el esfuerzo invertido por el equipo en la corrección de errores (Goldstein, 2014, p. 105). Este esfuerzo es considerado un desperdicio por lo que el objetivo es reducirlo al mínimo.

Esta medición se realiza al final de cada sprint. El equipo debe calcular la velocidad obtenida sumando los puntos de las funcionalidades desarrolladas y los errores corregidos para comparar los esfuerzos.

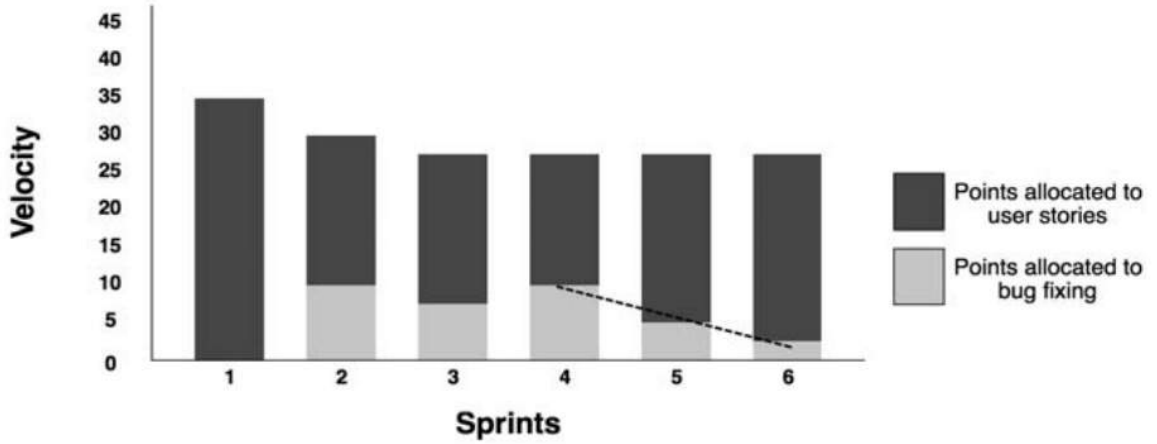


Figura 4-10 Gráfico de enfoque correctivo. La calidad mejora y los errores disminuyen (Goldstein, 2014, p. 106).

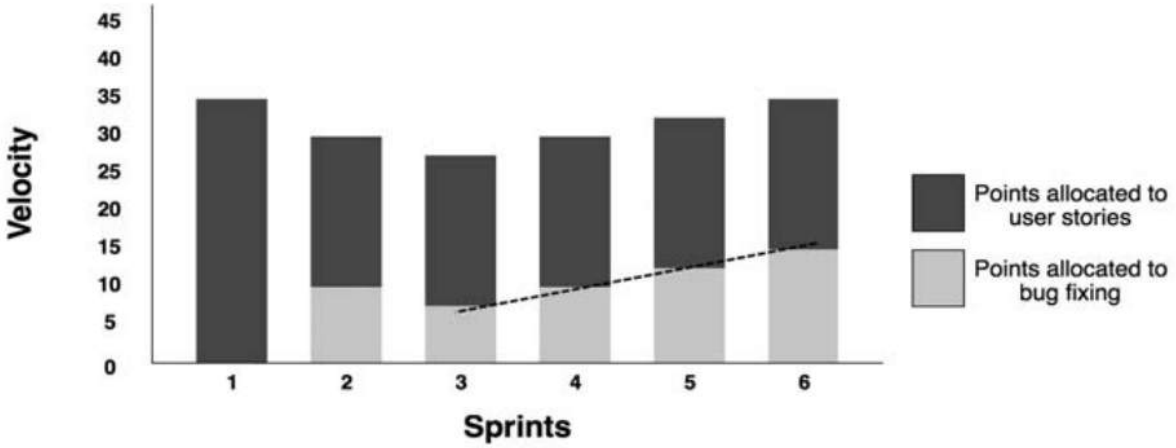


Figura 4-11 Gráfico de enfoque correctivo. La velocidad del equipo aumenta pero la calidad disminuye (Goldstein, 2014, p. 106).

# 5 CONCLUSIONES

Una vez finalizado este TFM, podemos decir que la efectiva gestión de los proyectos de desarrollo software necesita de una comprensión de cómo son los procesos involucrados. Precisamente hemos realizado una investigación sobre los aspectos teóricos y la naturaleza del software, desde los modelos tradicionales a los modelos que en las últimas décadas han surgido como respuesta a los resultados desfavorables obtenidos por el uso de los modelos predictivos. Definitivamente, se constata que los modelos prescriptivos no se adaptan a las particularidades y complejidad del desarrollo software de nuestros días. En este trabajo precisamente, hemos estudiado en profundidad la Metodología Ágil, como respuesta a esta necesidad.

A lo largo del documento se ha justificado que el enfoque ágil es una propuesta que se adapta mejor a la naturaleza de los proyectos software ya que se enfoca en aquellos aspectos más complejos en el actual desarrollo de software que son: la gestión de cambios y la adaptación rápida y continua. Cada metodología ágil ofrece diferentes prácticas y herramientas que aplicadas al proceso de desarrollo generan mejores resultados. Entre las ventajas que cabe esperar de la efectiva aplicación de una Metodología Ágil en el desarrollo de Software (Cohn, 2010, pp. 11-17):

**Mayor productividad y menores costos:** Los factores productividad y costos repercuten de forma directa en los resultados de la ejecución del proyecto y en los beneficios. En enfoque ágil propone diversas prácticas orientadas a mejorar los índices de estos factores. El cálculo de estas variaciones se dificulta debido a que la productividad en el desarrollo software no tiene una medida estándar por consenso, los indicadores varían según la empresa, equipo o proyecto, y no siempre son elegidos de forma acertada. Sin embargo, existen estudios y encuestas que revelan una considerable mejoría:

- QSMA: Los proyectos ágiles eran 16% más productivos que los proyectos tradicionales.
- Sobre el uso de los métodos ágiles en los proyectos, las encuestas de DDJ mostraron que el 82% de los encuestados consideraron que había aumentado la productividad. VersionOne publicó que el 50% de las personas aseguraron que la productividad había mejorado y un 23% consideró que el aumento de la productividad había sido significativo. Las encuestas estudiadas por David Rico reflejaron que la media en la mejora de la productividad fue de un 88% y que el ahorro de costes fue de un 26%.

Estas cifras no son comunes en todas las empresas que implementan metodologías ágiles debido a las diferencias organizacionales e implantaciones, pero reflejan una evidencia de mejoría general. Los costos también se ven reducidos debido a una tasa de defectos inferior. Esa variación aumentaría al considerar el tiempo ahorrado en el desarrollo de funcionalidades no necesarias.

**Mayor compromiso de los empleados y satisfacción laboral:** un equipo de desarrollo se siente motivado porque tiene un mayor control sobre el trabajo que realiza. El ritmo de trabajo es sostenible durante el proyecto, lo que favorece a tener ciclos de trabajo más eficientes. Esto se

traduce en realizar las entregas en las fechas establecidas y evitar que el equipo trabaje horas extras para cumplir con los objetivos.
Los resultados de un estudio realizado por Chris Mann y Frank Maurer de la Universidad de Calgary (Mann & Maurer, 2005)referente a las horas extras trabajadas, revelaron una reducción cercana a dos tercios tras la implantación de la metodología:

- 19% Promedio de horas extra antes de la implantación
- 7% Promedio de horas extra después de la implantación.

De los empleados encuestados se reportaron los siguientes resultados:

- Salesforce: 89% confirmó un aumento de satisfacción en el trabajo.
- VersionOne: 44% de las personas admitieron un aumento en la satisfacción en el trabajo y el 34% consideró que la mejoría fue significativa.

**Tiempo de comercialización más rápido:** en cada incremento, un equipo ágil hace entrega de un software que es funcional, por lo que el cliente no tiene necesidad de esperar hasta el final del proyecto para lanzar su producto al mercado.

Según los encuestados por VersionOne sobre el tiempo de comercialización un 41% aseguró que sus proyectos habían mejorado y otro 23% consideró que habían mejorado significativamente
QSMA mostró que los equipos ágiles tienen un tiempo de comercialización un 37% más rápido, comparando 26 proyectos ágiles con 7.500 proyectos tradicionales *(Véase Figura 5-1).*
Salesforce.com realizó un comparación de las funcionalidades entregadas del antes (2006) y después (2007) de la adopción de Scrum. Otros resultados de la encuesta de SalesForce fueron los siguientes:

- 94% de los clientes recomendarían a Salesforce.com.
- 61% mejora en tiempo medio de entrega
- 568% incremento en valor acumulado (funcionalidades) en entregas principales *(Véase Figura 5-2).*
- 94% Incremento en la entrega de funcionalidades solicitadas
- 38% Incremento en la entrega de funcionalidades solicitadas por desarrollador.

**Mayor calidad**: Los ciclos iterativos de las metodologías ágiles promueven prácticas que ayudan a mejorar la calidad del software. Las entregas continuas y retroalimentación permiten descubrir errores durante la ejecución del proyecto.

La investigación de David Rico referente a 51 estudios sobre proyectos ágiles, muestran que un 10% obtuvo una mejora mínima de calidad y un 63% una mejora media.
Mike Cohn (Cohn, 2010, p. 15) encontró que la empresa ePlanServices redujo un 70% la tasa de defectos por mil líneas de código.
VersionOne reportó que el 44% de los participantes consideró que la calidad había mejorado desde la adopción de la metodología ágil y un 24% consideró que había mejorado significativamente.

Además, el 84% de los encuestados consideró que habían reducido el número de defectos de software en un 10% o más. Otro 30% de los encuestados consideró que habían reducido en un 25% o más.

La encuesta del DDJ informó resultados similares, un 48% dijo que la calidad era algo más alta y el 29% dijo que era mucho mejor.

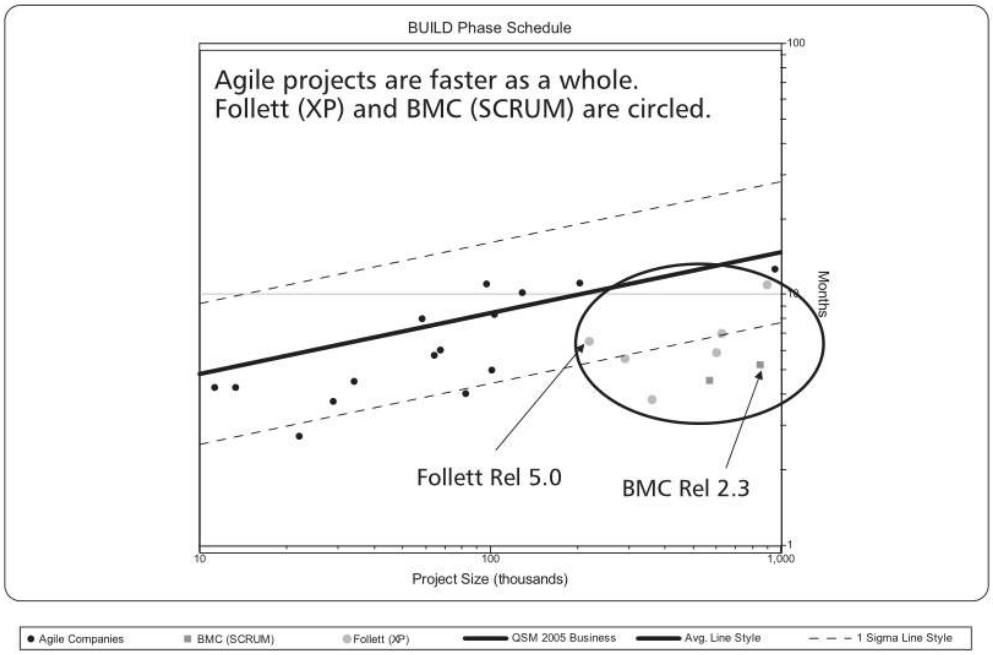


Figura 5-1 Tendencia de Cronograma: proyectos ágiles frente a proyectos tradicionales (Mah, 2008).

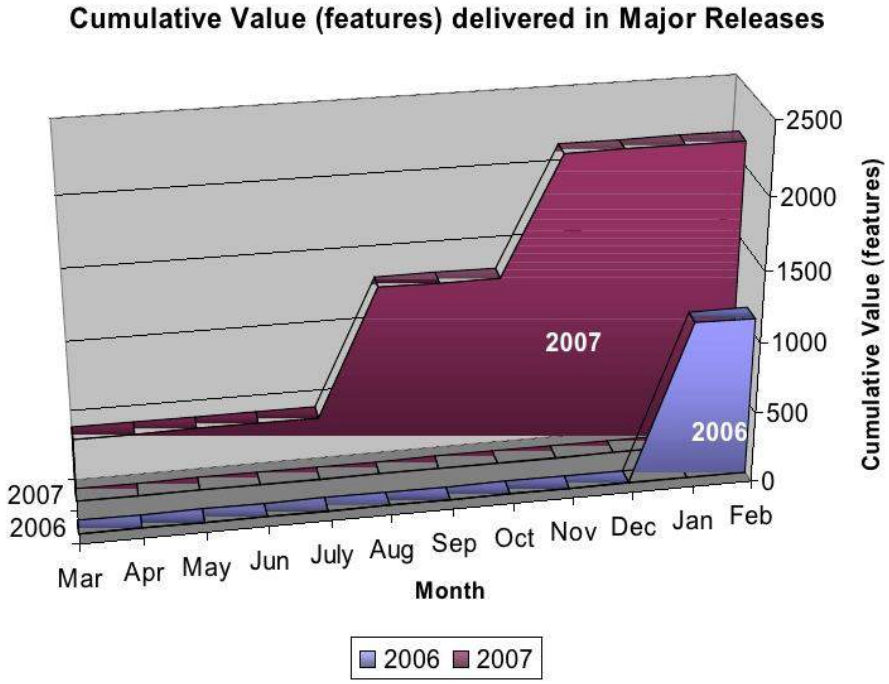


Figura 5-2 Valor acumulado (funcionalidades) entregado en Versiones Principales (Greene, 2008)

**Mayor satisfacción de las partes interesadas:** el enfoque flexible de las metodologías ágiles posibilita a realizar cambios de acuerdo a las necesidades de negocio, reduce la tasa de riesgo y permite al cliente evaluar el producto de forma temprana y continua. Además, promueve la transparencia de la información y la participación activa del cliente que facilita la toma de decisiones que determinen la correcta dirección del mismo.
DDJ publicó los siguientes resultados de una encuesta sobre la satisfacción de los interesados:

- 47% de los encuestados cree que aumentó.
- 31% de los encuestados cree que aumentó significativamente

La encuesta sobre capacidad de gestión de cambios en las prioridades realizada por VersionOne (Greene, 2008) reveló que el 92% de los participantes consideraron que mejoraba.

A propósito de la variedad de metodologías ágiles existentes, partiendo del Manifiesto Ágil y sus principios, se han revisado y examinado las metodologías ágiles más utilizadas, para finalmente explicitar de forma sintética una comparación entre ellas.

Finalmente, para completar la aproximación práctica a la Incorporación del Método Agile al desarrollo software, se ha detallado en este TFM una propuesta de implantación de la metodología ágil Scrum. Tal y como se ha visto en el Capítulo 4, se plantean los diferentes desafíos que afrontan las organizaciones durante el proceso de adopción y gestión del cambio, así como algunas propuestas y herramientas para afrontarlos. Se presentan diferentes prácticas que facilitan la toma de decisiones en cuestiones críticas como la selección del proyecto y del equipo. También se destaca la importancia de los contratos y mediciones en la gestión de proyectos ágiles. Tras este capítulo, se puede concluir que la implantación de una metodología ágil es un proceso complejo que requiere de:

- Un compromiso por parte de los diferentes niveles de la organización. Para ello es indispensable que la organización utilice un modelo de cambio que facilite este proceso de adopción y que defina un camino, como el propuesto en este trabajo (ADAPT).
- La alineación de la cultura organizacional hacia los principios y valores ágiles para eliminar aquellas barreras estructurales o burocráticas que ralenticen o impidan una adopción exitosa.
- Una comunicación y gestión del cambio eficaces. La organización debe transmitir los objetivos que se quieren lograr con la adopción de la metodología y la forma en la que ésta se llevara a cabo, ya que las personas que son afectadas necesitan un tiempo para analizar, comprender y aceptar el cambio. De igual forma, la organización debe atender aquellas preocupaciones y miedos expresadas por sus empleados para ayudarles durante la transición hacia la agilidad. Esto ayudará en gran manera a disminuir la resistencia al cambio.
- Los empleados de la organización necesitan recibir formación de personas con una amplia experiencia y dominio de Scrum. La capacitación y acompañamiento en el proceso de

transformación es fundamental para aumentar las probabilidades de una implantación exitosa.

La conclusión de todo este trabajo es clara en las necesidades organizacionales y de metodología que deben tenerse en cuenta para la implementación efectiva de Metodologías Ágiles. Este esfuerzo queda recompensado en una mayor flexibilidad en la incorporación de valor al producto software y en la gestión de los cambios, para una mejor eficiencia en el desarrollo software.

# 6 ANEXOS

## 6.1 ¿Por qué la gestión de proyectos de software es un desafío?

La extensión de Software de la Guía del PMBOK menciona algunos factores que dificultan la gestión de proyectos software (PMI, 2013b):

- Los proyectos de software son desafiantes porque el software es un producto intangible y maleable; el código fuente del software es texto escrito. En la mayoría de los casos, los equipos de desarrolladores de software generan y revisan documentos compartidos (por ejemplo, requisitos, especificaciones de diseño, código y planes de prueba). El desarrollo de software a menudo se caracteriza como un proceso de aprendizaje en el que se adquiere conocimiento y se genera información durante el proyecto.
- Atributos claves que hacen a los proyectos software sean difíciles son la complejidad del proyecto y del producto, la escala no lineal de los recursos, la medición del proyecto y del producto, la incertidumbre inicial del proyecto, el alcance del producto, y el conocimiento ganado a medida que el proyecto evoluciona.
- Los requerimientos a menudo cambian durante la ejecución del proyecto software a medida que se adquiere más conocimiento y aumenta la información sobre el alcance del proyecto y del producto.
- Los requisitos para crear o modificar software a menudo influyen, y están influenciados por, los procesos de negocios de la organización y el flujo de procesos de los empleados.
- El capital intelectual del personal es el principal capital activo para los proyectos software y las organizaciones de desarrollo software porque el software es un producto directo de los procesos cognitivos.
- La comunicación y la coordinación dentro de los equipos de software y con los interesados del proyecto a menudo carece de claridad. Algunas de las herramientas y prácticas utilizadas en la Ingeniería de Software están destinadas a mejorar la comunicación y coordinación.
- La creación de software requiere de soluciones innovadoras a los problemas para crear soluciones únicas. Los proyectos de software son más parecidos a proyectos de investigación y desarrollo que a proyectos de construcción o fabricación.
- Los proyectos software involucran riesgo e incertidumbre porque requieren innovación, el producto es intangible, y los interesados no pueden articular efectivamente o acordar las necesidades para estar satisfechos con el producto software.
- La planificación y estimación inicial de los proyectos software es un reto porque estas actividades dependen de requisitos que a menudo son imprecisos o parten de un histórico de datos que a menudo está olvidado o es inaplicable. Una estimación precisa es también difícil

porque la eficiencia y eficacia de los desarrolladores de software es ampliamente variable.

- La complejidad del producto dificulta el desarrollo y la modificación de software, debido al enorme número de interacciones entre los módulos del programa combinado con los datos que producen y las combinaciones de los detalles de las interfaces entre los módulos del programa.
- Las pruebas exhaustivas de software son inviables debido al tiempo que se requeriría para probar todas las interacciones, interfaces y combinaciones de datos y estímulos de entrada.
- El desarrollo de software a menudo implica incluir productos de diferentes proveedores y el desarrollo de interfaces con otros software; esto puede resultar en problemas de integración y rendimiento.
- Debido a que la mayoría del software está interconectado, las técnicas de seguridad de la información son necesarias. La seguridad del software es un reto grande y creciente.
- La cuantificación y medición objetiva de la calidad del software es difícil por la naturaleza intangible del software.
- Los desarrolladores del software utilizan procesos, métodos, y herramientas que están evolucionando constantemente y son actualizados frecuentemente.
- El software es elemento de un sistema que a menudo se modifica cuando se van a cambiar los atributos de funcionalidad, comportamiento o calidad.
- Se puede requerir que un producto software opere en una variedad de plataformas hardware e infraestructuras software.
- El software no es un producto independiente. Este es ejecutado en un hardware y a menudo es un elemento de un sistema compuesto por un hardware diverso, otro software y procedimientos manuales.
- Las plataformas tecnológicas, infraestructura software y software suministrado por proveedores son actualizados o modificados con frecuencia lo que puede requerir cambios en el software que se está desarrollando.

## 6.2 Herramientas de Lean Software Development

Los principios de LSD se soportan en 22 herramientas. En la siguiente tabla, el autor (Measey, 2015, pp. 155-156) hace una descripción de cada una de estas herramientas.

| Herramienta | Descripción |
|---|---|
| Identificar desperdicios | Algunos desperdicios son obvios, es el caso de los defectos. Otros desperdicios son menos evidentes: historias de uso extra, esperas, cambios de tarea. |
| Mapeo del flujo de valor | Mapear el flujo de un requerimiento desde la idea a la implementación. Esto ayuda a identificar colas, esperas, traspasos excesivos, etapas o fases que no aportan valor entre otros. |
| Retroalimentación | Incrementar la velocidad y la calidad de la retroalimentación en varios niveles para mejorar la calidad de los procesos. |
| Iteraciones | Entrega temprana y continua para obtener retroalimentación del producto y su valor. |
| Sincronización | Integrar software con frecuencia para evitar sorpresas y trabajo adicional. |
| Set-based development | Trabaje en múltiples opciones o deje que surja la solución. |
| Options thinking | Permitir múltiples opciones abiertas aporta valor. Retrasar las decisiones en lo posible hasta disponer de mayor información. |
| Último momento responsable | Es el momento en que cual una opción importante será eliminada si se retrasa una decisión. No se puede continuar retrasando la decisión |
| Toma de decisiones | Delegar en lo posible las decisiones al equipo y tomar las decisiones basándose en reglas simples (reglas de aceptación con una clara definición de método de prioridades para las historias del backlog entre los interesados). |
| Pull systems | Permitir al equipo añadir tareas cuando tengan capacidad disponible. Retirar tareas al equipo aumenta el trabajo pendiente sin mejorar el rendimiento. |
| Teoría de colas | Identificar, medir y reducir colas de trabajo parcialmente terminado en el proceso de desarrollo.<br>Existen múltiples formas para realizar este proceso, sin embargo, medir el coste del retraso es un buen punto de partida. |

| | |
|---|---|
| Coste del retraso | Diseñar un modelo económico que valore la velocidad de entrega de forma que se pueda compensar con otros factores como el riesgo, coste entre otros |
| Autodeterminación | Permitir a los equipos decidir como organizan su propio trabajo. |
| Motivación | La motivación del equipo proviene de un propósito alcanzable, el acceso al cliente y la libertad de asumir sus propios compromisos en un entorno en el que los errores son oportunidades para aprender. |
| Liderazgo | La mayoría de las iniciativas exitosas tienen un miembro campeón y un desarrollador maestro que se preocupan apasionadamente por el cliente y la integridad del producto, respectivamente. |
| Experiencia | Comparta la experiencia y haga cumplir los estándares por parte del líder ágil lo que facilita que el equipo se auto-organice y trabaje de manera colaborativa. |
| Integridad percibida | Alinee todas las actividades al valor del cliente para que quede claro por qué están sucediendo las cosas. |
| Integridad conceptual | Asegúrese de que los componentes / interfaces del producto funcionen juntos como un todo cohesivo y uniforme. |
| Refactorización | Esencial para mantener la salud del producto, ya que no todo se puede prever por adelantado en el momento del diseño. |
| Testing | Automatice las pruebas para acelerar el ciclo de retroalimentación. |
| Mediciones | Haga visibles los problemas y el progreso. |
| Contratos | Evite fijar el alcance por adelantado en un contrato. |

Tabla 6-1 Herramientas de Lean Software Development (Measey, 2015, pp. 155-156)

## 6.3 Variante: sucesión de Fibonacci

En el libro de Scrum Manager, los autores (Palacio & Ruata, 2011, p. 104) realizan una breve explicación acerca del uso de la sucesión de Fibonacci en la práctica del planning poker, y lo describen de la siguiente forma:

"Basado en el hecho de que al aumentar el tamaño de las tareas, aumenta también el margen de error, se ha desarrollado una variante que consiste en emplear sólo números de la sucesión de Fibonacci para realizar las estimaciones, de forma que:

- El juego de cartas está compuesto por números de la sucesión de Fibonacci.
- La estimación no se realiza levantando varias cartas para componer la cifra exacta, sino poniendo boca arriba la carta con la cifra más aproximada a la estimación.
- Para estimar tareas puede ser válido un juego de cartas como éste:

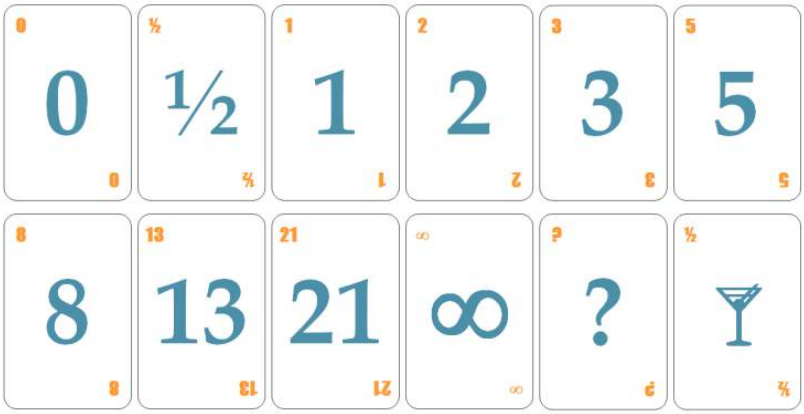


Figura 6-1 Ejemplo de cartas para estimación de póquer con secuencia Fibonacci (Palacio & Ruata, 2011, p. 104)

Si se quiere emplear la planificación de póquer para estimar requisitos a nivel de producto o de versión (funcionalidades, temas…) además de usarlo al nivel de tareas de sprint, se pueden añadir cartas al juego para permitir estimaciones de mayor tamaño (34, 55, 89, 144…).

Es frecuente emplear una carta con un símbolo de duda o interrogación para indicar que, por las razones que sean, no se puede precisar una estimación. También es posible incluir otra carta con alguna imagen alusiva, para indicar que se necesita un descanso".

# 7 BIBLIOGRAFÍA


AENOR, 2013. *UNE ISO 21500:2012. Directrices para la dirección y gestión de proyectos..* Madrid: AENOR.

Agile Practitioners Guide, s.f. *Agile Practitioners Guide.* [En línea]
Available at: https://agilefaq.wordpress.com/2013/01/14/what-is-velocity/
[Último acceso: 28 Noviembre 2020].

Agile PrepCast, 2013. *Comparison of Agile Methods by The Agile PrepCast.* [En línea]
Available at:
https://hwcdn.libsyn.com/p/7/3/7/73773eb1e8ac86ca/The_Agile_Methods_Comparison_by_The_Agile_PrepCast.pdf?c_id=5993202&cs_id=5993202&expiration=1605251739&hwt=45fae9a55f268faa22f28f59530c0451
[Último acceso: 1 Noviembre 2020].

Ahmed, A., 2011. *Software Project Management : A Process-Driven Approach.* FL: CRC Press.

Alahyari, H., Gorschek, T. & Berntsson Svensson, R., 2019. An exploratory study of waste in software development organizations using agile or lean approaches: A multiple case study at 14 organizations. *Information and Software Technology,* Volumen 105, pp. 78-94.

Albaladejo, X., 2011. *Proyectos Ágiles.* [En línea]
Available at: https://proyectosagiles.org/2011/12/13/como-cocinar-tu-contrato-agil/
[Último acceso: 25 Noviembre 2020].

Al-Zewairi, M., Biltawi, M., l Etaiwi, W. & Shaout, A., 2017. Agile Software Development Methodologies: Survey of Surveys. *Journal of Computer and Communications,* 5(5), pp. 74-97.

Ambler, S., 2002. *Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process.* New York: John Wiley & Sons, Inc..

Ambler, S., 2005. *Agile Modeling.* [En línea]
Available at: http://agilemodeling.com/practices.htm
[Último acceso: 28 Octubre 2020].

Ambler, S., 2008. *Ambysoft.* [En línea]
Available at: http://www.ambysoft.com/surveys/agileFebruary2008.html
[Último acceso: 2020 Noviembre 2020].

Anderson, D. J., 2010. *Kanban: Successful Evolutionary Change for Your Technology Business.* Sequim: Blue Hole Press.

Anderson, D. J. & Carmichael, A., 2017. *Essential Kanban Condensed.* s.l.:Lean Kanban University.

Arbogast, T., Larman, C. & Vodde, B., 2012. *Agile Contracts.* [En línea]
Available at: https://www.agilecontracts.org/agile_contracts_primer.pdf
[Último acceso: 22 Noviembre 2020].

Bauer, F. L., Bolliet, L. & Helms, H. J., 1969. *Ingeniería del Software: Reporte de una conferencia patrocinada por el comité científico de la OTAN.* Garmisch, Alemania, s.n.

Beck, K., 1999. *Extreme Programming Explained. Embrace Change..* Primer Edición ed. Boston: Addison-Wesley.

Beck, K., 2005. *Extreme Porgramming Explained: Embrace Change.* Segunda Edición ed. s.l.:Addison - Wesley.

Beck, K. y otros, 2001. *Manifesto for Agile Software Development.* [En línea]
Available at: http://agilemanifesto.org/
[Último acceso: 20 Febrero 2020].

Bhavsar, K., Gopalan, S. & Shah, V., 2020. Scrumban: An Agile Integration of Scrum and Kanban in Software Engineering. *International Journal of Innovative Technology and Exploring Engineering ,* 9(4), pp. 1626-1634.

Boehm, B., 2008. Making a Difference in the Software Century. *Computer,* 31 Marzo, 41(3), pp. 78-84.

Boehm, B. & Turrner, R., 2005. Management Challenges to Implementing Agile Process in Traditiona Development Organizations. *IEEE Software,* 22(5), pp. 30-39.

Boehm, B. W., 1981. *Software Engineering Economics.* s.l.:Englewood Cliffs, N.J. Prentice-Hall.

Boehm, B. W., 1988. A spiral model of software development and enhancement. *Computer,* 21(5), p. 61–72.

Boehm, B. W., 2001. The Spiral Model as a Tool for Evolutionary Software Acquisition. *CrossTalk,* Volumen 801, p. 4.

Bradac, M., Perry, D. & Votta, L., 1994. Prototyping a process monitoring experiment. *EEE Transactions on Software Engineering,* 20(10), pp. 774-784.

Brezočnik, L. & Majer, Č., 2016. *Comparison of agile methods: Scrum, Kanban, and Scrumban.* s.l., s.n.

Brooks, F. P., 1987. No Silver Bullet. Essence and Accidents of Software Engineering. *Computer,* 20(4), pp. 10-19.

Broy, M., 2018. The Leading Role of Software and Systems Architecture in the Age of Digitization. En: V. Gruhn & R. Striemer, edits. *The Essence of Software Engineering.* Primera edición ed. s.l.:Springer International Publishing, pp. 1-23.

Caamaño, J. E., 2011. *Project Management Práctico. Técncias, Heramientas y Documentos.* Primera edición ed. s.l.:Editorial Círculo Rojo.

Charette, R. N., 2005. Why Software Fails. *IEEE Spectrum,* 1 Septiembre, 42(9), pp. 42 - 49.

Cobb, C. G., 2015. *The Project Manager's Guide to Mastering Agile : Principles and Practices for an Adaptive Approach.* New York: John Wiley & Sons, Incorporated.

Cockburn, A. & Williams, L., 2001. The Costs and Benefits of Pair Programming. En: *Extreme programming examined.* s.l.:Addison-Wesley , p. 223–243.

Coding Sans, 2020. *State of Software Development,* s.l.: Coding Sans.

Cohn, M., 2010. *Succeding with Agile. Software development using Scrum.* s.l.:Addison-Wesley.

Combe, M., 2014. Building Change Agility: The Strategic Process for Agility Improvement. *PMI White Papers.*

Cooke, J. L., 2014. *Agile Productivity Unleashed : Proven Approaches for Achieving Productivity Gains in Any Organisation.* Segunda edición ed. s.l.:IT Governance Ltd.

Cooke, J. L., 2016. *Agile : an executive guide : real results from it budgets.* Segunda edición ed. Cambridgeshire: IT Governance Publishing.

Crawford, B. & Leon de la Barra, C., 2008. *Proceedings of the International MultiConference of.* s.l., Newswood Limited, pp. 1026-1032.

Crystalloids, s.f. *Crystalloids.* [En línea]
Available at: https://www.crystalloids.com/hs-fs/hubfs/Scrum%20Process.jpg?width=1800&name=Scrum%20Process.jpg
[Último acceso: 8 Noviembre 2020].

Deemer, P., Benefield, G., Larman, C. & Vodde, B., 2012. *The Scrum Primer.* [Online]
Available at: http://scrumprimer.org/primers/es_scrumprimer20.pdf
[Accessed 10 Febrero 2020].

DeMarco, T. & Lister, T., 2013. *Peopleware Productive Projects and Teams.* Tercera edición ed. s.l.:Addison - Wesley.

Doran, T. G., 1981. There's a S.M.A.R.T. way to write management's goals and objectives. *Management Review,* 70(11), pp. 35-36.

Fairley, R. E. (., 2009. En: *Managing and Leading Software Projects.* New Jersey: John Wiley & Sons. Inc..

Fuller, M. A., Valacich, J. S. & George, J. F., 2008. *Information Systems Project Management.* New Jersey: Pearson Prentice Hall.

Goldstein, I., 2014. *Scrum Shortcuts without cutting corners.* s.l.:Addison-Wesley.

Greene, S. & Fry, C., 2008. [En línea]
Available at: https://www.slideshare.net/sgreene/scrum-gathering-2008-stockholm-salesforcecom-presentation
[Último acceso: 19 Noviembre 2020].

Griffiths, M., 2011. *Leading Answers.* [En línea]
Available at: https://leadinganswers.typepad.com/leading_answers/2011/09/pmi-acp-value-stream-mapping.html
[Último acceso: 30 Octubre 2020].

Hanssen, H., Bjørnson, F. O. & Westerheim, H., 2005. Using Rational Unified Process in an SME. En: I. Richardson, P. Abrahamsson & R. Messnarz, edits. *Software Process Improvement.* Budapest: Springer-Verlag Berlin Heidelberg, pp. 142-150.

Hanssen, H., Bjørnson, F. O. & Westerheim, H., 2007. Tailoring and Introduction of the Rational Unified Process. En: P. Abrahamsson, N. Baddoo, T. Margaria & R. Messnarz, edits. *Software Process Improvement 14th European Conference, EuroSPI 2007, Potsdam, Germany, September 26-28, 2007, Proceedings ..* Primera edición ed. Berlin: Springer Berlin Heidelberg, pp. 7-18.

Highsmith, J., 2010. *Agile Project Management.* Segunda ed. Boston: Addison-Wesley.

Holcombe, M., 2008. Running an Agile Software Development Project. En: New Jersey: John Wiley & Sons, Incorporated.

Hughes, B. & Cotterell, M., 2009. *Software Project Management.* Quinta ed. Londres: McGraw-Hill Education.

IBM, 2003. *IBM.* [En línea]
Available at:
https://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf
[Último acceso: 2 Agosto 2020].

IEEE, 1990. *IEEE Standard Glossary of Software Engineering Terminology.* New York: IEEE.

IEEE, 1990. *IEEE Standard Glossary of Software Engineering Terminology. Std 610.12-199.* New York: Institute of Electrical and Electronics Engineers.

IPMA, 2009. *NCB: Bases para la Competencia de Dirección de Proyectos, versión 3.1.* Valencia: UPV.

ISO, 2015. *Sistemas de gestión de la calidad. Fundamentos y vocabulario. (ISO 9000:2015).* s.l.:AENOR.

Jacobson, I., Booch, G. & Rumbaugh, J., 2000. *El Proceso Unificado de Desarrollo Software.* Reimp ed. Madrid: Addison Wesley.

Kniberg, H. & Skarin, M., 2010. *Kanban y Scrum – Obteniendo lo mejor de ambos.* s.l.:C4Media Inc..

Kruchten, P., 2000. *The Rational Unified Process An Introduction.* Segunda edición ed. s.l.:Addison Wesley.

Letelier, P. & Penadés, M. C., 2006. Métodologías ágiles para el desarrollo de software: eXtreme Programming (XP). *Dialnet,* 5(26).

Lindvall, M. y otros, 2002. *Empirical Findings in Agile Methods.* s.l., Springer, Berlin, Heidelberg, pp. 197 - 207.

Mah, M., 2008. *How Agile Projects Measure Up, and What This Means to you?,* s.l.: Cutter Consortium .

Maley, C. H., 2012. *Project Management Concepts, Methods, and Techniques..* s.l.:Taylor & Francis Group, LLC.

Mann, C. & Maurer, F., 2005. *A case study on the impact of Scrum on overtime.* IEE, s.n., pp. 70-79.

Measey, P., 2015. *Agile Foundations : Principles, practices and frameworks.* Swindon: BCS Learning & Development Limited.

Menzinsky, A., López, G. & Palacio, J., 2016a. *Scrum Manager.* Versión 2.6 ed. s.l.:Info 4 Media SL.

Menzinsky, A., López, G. & Palacio, J., 2016. *Scrum Manager.* Versión 2.6 ed. s.l.:Info 4 Media SL.

Mohapatra, P. K. J., 2010. *Software engineering. (A Lifecycle Approach).* Primera ed. Nueva Delhi: New Age International.

Mohapatra, P. K. J., 2010. *Software engineering. (A Lifecycle Approach).* Primera ed. Nueva Delhi: New Age International.

Murray, A. P., 2016. *The Complete Software Project Manager : Mastering Technology from Planning to Launch and Beyond.* New Jersey: John Wiley & Sons.

O'Regan, G., 2011. *Introduction to Software Process Improvement.* Primera ed. Londres: Springer-Verlag London.

Palacio, J. & Ruata, C., 2011. *Scrum Manager - Gestión de Proyectos.* 1.4.0 ed. s.l.:Safe Creative.

Palacio, J. & Ruata, C., 2016. *Scrum Manager.* Versión 2.6 ed. s.l.:Info 4 Media SL.

Palacio, M., 2020. *Scrum Master.* Versión 3.04 ed. s.l.:Iubaris Info 4 Media SL.

Patel, V. N., 2008. *Project Management.* Jaipur: Oxford Book Co..

PMI, 2013a. *Guía de los Fundamentos para la Dirección de Proyectos (Guía del PMBOK).* Quinta ed. Pensilvania: Project Management Institute.

PMI, 2013b. *Software Extension to the PMBOK. Guide Fifth Edition.* Pennsylvania: Project Management Institute, Inc.

PMI, 2017. *Guía de los fundamentos para la dirección de proyectos : (Guía del PMBOK).* Sexta ed. Pennsylvania: Project Management Institute.

PMI, 2017. *Guía Práctica de ÁGIL.* Pennsylvania: Project Management Institute, Inc.

Poppendieck, M. & Cusumano, M. A., 2012. Lean Software Development: A tutorial. *IEEE SoftwarE,* 29(5), pp. 26-32.

Poppendieck, M. & Poppendieck, T., 2003. *Lean Software Development: An Agile Toolkit.* s.l.:Addison- Wesley.

Pressman, R. S., 2010. Ingeniería del Software. En: *Ingeniería del Software, un enfoque práctico..* Séptima ed. D.F.: McGraw-Hill Interamericana Editores, S.A. de C.V..

Rao, N., 2017. *Scrumban Software Maintenance: 5 Steps to Stop Starting and Start Finishing.* s.l.:CreateSpace Independent Publishing Platform.

Rico, D., Sayani, H., Sone, S. & V, J., 2009. *The Business Value of Agile Software Methods : Maximizing ROI with Just-in-Time Processes and Documentation.* s.l.:J. Ross Publishing.

Roche, J., 2020. *Deloitte.* [En línea]
Available at: https://www2.deloitte.com/es/es/pages/technology/articles/ceremonias-scrum.html
[Último acceso: 29 Septiembre 2020].

Rojas Pino, L. A., 2017. *INTEGRACIÓN DE LA ARQUITECTURA DE LA INFORMACIÓN DENTRO DE UN PROCESO ÁGIL DE DESARROLLO CENTRADO EN ELUSUARIO,* s.l.: s.n.

Royce, W. W., 1970. *Managing The Development of Large Software Systems.* s.l., s.n., pp. 328-338.

Ruhe, G. & Wohlin, C., 2014. *Software Project Management in a Changing World.* Nueva York: Springer.

Scaled Agile, I., 2019. *Scaled Agile.* [En línea]
Available at: https://www.scaledagileframework.com/agile-workspaces/
[Último acceso: 25 Octubre 2020].

Schwaber, K. & Sutherland, J., 2012. *30 Days: How Agile Managers Beat the Odds, Delight Their Customers, And Leave Competitors In the Dust.* s.l.:Wiley.

Schwaber, K. & Sutherland, J., 2017. *La Guía de Scrum.* [En línea]
Available at: http://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-Spanish-SouthAmerican.pdf#zoom=100
[Último acceso: 3 Septiembre 2020].

SCRUMstudy, 2017. *Una Guía para el CUERPO DE CONOCIMIENTO DE SCRUM (GUÍA SBOK).* Tercera Edición ed. s.l.:SCRUMstudy™, una marca de VMEdu, Inc..

Shaydulin, R. & Sybrandt, J., 2017. *To Agile, or not to Agile: A Comparison of Software Development Methodologies.* [En línea]
Available at: https://arxiv.org/abs/1704.07469
[Último acceso: 3 Noviembre 2020].

Shenhar, A., Dvir, D., Lechler, T. & Poli, M., 2002. *One size does not fit all--true for projects, true for frameworks.* Seattle, PMI.

Siegelaub, J. M., 2007. *Project Management Institute.* [En línea]
Available at: http://www.pmi.org/learning/six-constraints-enhanced-model-project-control-7294
[Último acceso: 4 Agosto 2020].

Sommerville, I., 2011. Ingeniería del Software. En: L. M. Cruz Castillo, ed. Novena ed. s.l.:Pearson Education, Inc..

Standish Group, 2015. *CHAOS Report.* [En línea]
Available at: https://www.standishgroup.com/sample_research_files/CHAOSReport2015-Final.pdf
[Último acceso: 25 Julio 2020].

Sutherland, J., 2008. *Scruminc.* [En línea]
Available at: https://www.scruminc.com/agile-contracts-money-for-nothing-and/
[Último acceso: 23 Noviembre 2020].

Sutherland, J., 2016. *Scrum. El arte de hacer el doble de trabajo en la mitad de tiempo..* s.l.:Océano.

Takeuchi, H. & Nonaka, I., 1986. The New New Product Development Game. *Harvard Business Review.*

Teamhood, s.f. *Teamhood.* [En línea]
Available at: https://teamhood.com/agile/scrumban-a-hybrid-agile-framework-with-long-term-planning/
[Último acceso: 5 Noviembre 2020].

Tukey, J. W., 1958. The Teaching of Concrete Mathematics. *The American Mathematical Monthly,* 65(1), pp. 1-9.

Vanderjack, B., 2015. *The Agile Edge: Managing Projects Effectively Using Scrum.* Primera edición ed. New York: Business Expert Presss.

VersionOne, 2008. [En línea]
Available at: http://www.versionone.com/pdf/3rdAnnualStateOfAgile_FullDataReport.pdf

VersionOne, 2020. *14th Annual State of Agile Report,* s.l.: Digital.ai.

Villafiorita, A., 2014. *Introduction to Software Project Management.* s.l.:CRC Press, Taylor & Francis Group.

Wagner, M., 2015. *Project Management.* [En línea]
Available at: https://www.projectmanagement.com/articles/301100/Effective-Steering-Committee-Meetings
[Último acceso: 25 07 2020].

Wisocky, R. K., 2014. *Effective Project Management. Traditional, Agile, Extreme..* Séptima ed. Indianapolis: John Wiley & Sons, Inc..

Womack, J. P., Jones, D. T. & Roos, D., 1990. *The machine that changed the world.* s.l.:s.n.