

**UNIVERSIDAD POLITÉCNICA DE MADRID**

**Escuela Técnica Superior de Ingeniería de Sistemas Informáticos**

Máster Universitario en Ingeniería Web

Trabajo Fin de Máster

Estado del arte de las metodologías de desarrollo ágiles

**Autor**

**Manuel Puchades Rodríguez**

**Tutor**

**Luis Fernández Muñoz**

**julio de 2019**

## 

## Agradecimientos

Gracias a …

## Resumen

Amparadas bajo el manifiesto ágil, cohabitan diferentes herramientas y prácticas para la gestión y elaboración de proyectos de software. Estas son conocidas como metodologías ágiles. Sin embargo, el término ágil ha sufrido un gran desgaste en los últimos años y varios de los creadores y firmantes del manifiesto sostienen que los principios y valores originales que llevaron a su redacción se han diluido con el tiempo en los detalles de las implementaciones.

En el presente proyecto exploraremos la historia de los preceptos ágiles, estudiaremos las metodologías más extendidas, y cómo ha evolucionado el concepto de agilidad a través de estas hasta nuestros días.

Palabras clave

manifiesto ágil, metodologías ágiles, SCRUM, XP, Lean, Kanban.

## Abstract

*[El resumen anterior en inglés]*

*In 2001, seventeen professionals set up the manifesto for agile software development. They wanted to define values and basic principles for better software development. On top of being brought into focus, the manifesto has been widely adopted by developers, in software-developing organizations and outside the world of IT. Agile principles and their implementation in practice have paved the way for radical new and innovative ways of software and product development. In parallel, the understanding of the manifesto’s underlying principles evolved over time. This, in turn, may affect current and future applications of agile principles. This article presents results from a survey and an interview study in collaboration with the original contributors of the manifesto for agile software development. Furthermore, it comprises the results from a workshop with one of the original authors. This publication focuses on the origins of the manifesto, the contributors’ views from today’s perspective, and their outlook on future directions. We evaluated 11 responses from the survey and 14 interviews to understand the viewpoint of the contributors. They emphasize that agile methods need to be carefully selected and agile should not be seen as a silver bullet. They underline the importance of considering the variety of different practices and methods that had an influence on the manifesto. Furthermore, they mention that people should question their current understanding of “agile” and recommend reconsidering the core ideas of the manifesto.*

Keywords

*[Palabras clave relacionadas con el trabajo]*

## Tabla de Contenidos

[Capítulo 1. Introducción 3](#_Toc12720505)

[1. Antecedentes 3](#_Toc12720506)

[2. Motivación 5](#_Toc12720507)

[3. Objetivos 6](#_Toc12720508)

[4. Contenido 7](#_Toc12720509)

[Capítulo 2. Entendiendo la agilidad 8](#_Toc12720510)

[1. Dynamic Systems Development Method (DSDM) 8](#_Toc12720511)

[2. Adaptive Software Development 9](#_Toc12720512)

[3. Scrum 10](#_Toc12720513)

[4. Crystal 13](#_Toc12720514)

[5. Extreme Programming (XP) 14](#_Toc12720515)

[6. Lean and Kanban 17](#_Toc12720516)

[7. El manifiesto ágil 17](#_Toc12720517)

[Capítulo 3. La agilidad hoy 20](#_Toc12720518)

[1. Cómo Agile se convirtió en sustantivo y perdió sus valores con el paso del tiempo. 20](#_Toc12720519)

[2. Hecho por desarrolladores impuesto por las empresas. 24](#_Toc12720520)

[3. El problema de las certificaciones 26](#_Toc12720521)

[4. La agilidad, ese gran desconocido: prácticas vs. principios 28](#_Toc12720522)

[Capítulo 4. El discurso Agile 32](#_Toc12720523)

[1. Catastrofismo y la falsa dicotomía: Ágil o Cascada 32](#_Toc12720524)

[2. Un conjunto de casos de éxito, pocas estadísticas reales. Confundiendo causalidad y casualidad. 35](#_Toc12720525)

[3. Todo o nada. No estas siendo lo suficientemente ágil 35](#_Toc12720526)

[Conclusiones y Posibles Ampliaciones 36](#_Toc12720527)

[1. Conclusiones 36](#_Toc12720528)

[2. Líneas futuras 37](#_Toc12720529)

[Referencias 39](#_Toc12720530)

## Introducción

### Antecedentes

Probablemente el cambio más notable en el pensamiento del proceso de software en los últimos años ha sido la aparición de la palabra “agile". Con el término agile nos referimos a métodos de software ágiles, de cómo introducir la agilidad en un equipo de desarrollo de software.

Este nuevo movimiento surgió en la década de 1990 cuando los profesionales del software comenzaron a percatarse de que la tasa de cambio en los requisitos iba aumentando mucho más allá de las capacidades de las metodologías de desarrollo clásicas [1] [2] encontrándolas así deficientes.

Esta situación se produce a raíz de la rápida evolución de la industria y tecnologías del software. Los clientes no eran capaces de determinar cuáles serían sus necesidades por adelantado.

Los métodos iterativos, como el enfoques basados en el modelo en espiral [3] [4], los procesos evolutivos descritos en [5] [6] [7] [8] y, más recientemente, los enfoques ágiles [9] cuentan con el cambio como elemento principal, reconociéndolo como la única constante en los proyectos de software.

En su esfuerzo, dichos profesionales buscaron un nuevo enfoque como resultado, las metodologías y prácticas ágiles surgieron como un intentar abarcar de manera más formal mayores tasas de cambio de requisitos. La mayoría de las ideas que formaron este nuevo movimiento no eran nuevas, de hecho, fue una rebautización de unas determinadas formas de trabajar, ya que existía la creencia de que gran parte del software desarrollado hasta entonces de esta forma era exitoso.

Para la creación de este movimiento se determinaron ciertos principios fundamentales que aglutinaron las metodologías ágiles y que supusieron un notable contraste de los supuestos de las metodologías establecidas hasta el momento.

Más en profundidad Larman y Basili [10] describen la historia del desarrollo de software iterativo e incremental (IID). Esta comenzaría mucho antes en los años 70 y llegando a su punto álgido con el manifiesto de 2001. Establecen el comienzo de la relación entre IID y los métodos ágiles de la siguiente manera: "En febrero de 2001, un grupo de 17 expertos en procesos [...] interesados ​​en promover métodos y principios de IID modernos y simples se reunieron en Utah para discutir un terreno común.”

En la publicación estructuran la historia ágil en décadas. Según su investigación la mentalidad ágil comenzó en la década de 1930 con la idea de los ciclos de “planear-hacer-estudiar-actuar”. Mencionan varios proyectos, como el proyecto de la NASA Mercury (el primer programa de vuelo espacial humano de los Estados Unidos) o el desarrollo de software para el sistema de armas de helicóptero a barco de la Armada de los Estados Unidos, en las que fueron aplicadas todas las prácticas del IID. Señalan que los ejercicios de iteraciones cortas y el desarrollo de primera prueba ya se utilizaron en el proyecto Mercury. Estas prácticas permanecen presentes en métodos ágiles como Scrum o XP que se crearon en la década de los 90.

En los años setenta, Royce [11] publicó un artículo considerado como la base para el modelo de cascada. En su artículo, Royce describió sus opiniones sobre la gestión de grandes desarrollos de software y lo que era “necesario para transformar un proceso de desarrollo riesgoso en uno que proporcione el producto deseado”. Su enfoque apunta a alcanzar la condición de que el software de trabajo se entregue a tiempo y dentro de los costos.

Royce propone utilizar las fases del modelo de cascada con una relación iterativa entre fases sucesivas. El proceso se beneficia al reducir el proceso de desarrollo a límites manejables. Además, sugiere utilizar prototipos para obtener una simulación temprana del producto final. En este documento se presentan las primeras reflexiones sobre el desarrollo iterativo, la retroalimentación y la adaptación.

En la década de 1980, se presentaron muchos enfoques de desarrollo de software incremental [12] [13], entre los que destaca el modelo espiral de Boehm [3]. El modelo en espiral es un enfoque orientado al manejo y reducción de riesgo para el desarrollo de software.

Según Boehm, el modelo en espiral resultó de una "variante de gestión de riesgos del modelo de cascada". El proceso de desarrollo se divide en varios círculos que comprenden pasos como el análisis de riesgos, la planificación de círculos y las revisiones. Señala que "el modelo sostiene que cada ciclo implica una progresión a través de la misma secuencia de pasos". Concluyendo en que el enfoque basado en el riesgo es más adaptable que el enfoque que denomina “basado en documentos”.

Inspirado en las ideas de Barry Boehm, James Martin creó durante la década de 1980 en IBM el enfoque de desarrollo rápido de aplicaciones, formalizándolo finalmente al publicar el libro Rapid Application Development en 1991, comúnmente conocido como RAD.

El enfoque RAD incluyó una entrega rápida e iterativa con un pequeño equipo de desarrolladores altamente capacitados. Además, RAD estableció las bases para el Método de Desarrollo de Sistemas Dinámicos (DSDM) que tuvo una amplia acogida en el noroeste de Europa, Reino Unido, Países Bajos, Suecia y Dinamarca.

La creación del término ágil se vio alimentado, además, de una opinión que no las consideraba formas de trabajo rigurosas, por lo que sus creadores decidieron contraponer esta creencia de ideas reprimidas y no tratadas seriedad por personas interesadas en el mismo ámbito de desarrollo.

Tras varias décadas de desarrollo de software y mejoría en las prácticas de enfoques de desarrollo y aplicaciones en la industria, las formas de trabajo ágiles culminaron en Utah donde se combinaron las técnicas más adecuadas dentro del Manifiesto para el desarrollo ágil de software.

El crecimiento de métodos y prácticas ágiles comenzó a recibir una atención significativa a partir de 2001 con la publicación de dicho manifiesto (Manifiesto para el desarrollo de software ágil) a menudo denominado "manifiesto ágil".

En palabras de Dave Thomas, la famosa reunión de Utah se realizó con el propósito de compartir el conocimiento adquirido por una serie de personas que tenían ideas comunes en cuanto al desarrollo de software se refiere [14]. Durante el encuentro trataron de describir aquellas ideas, lo que finalmente resultó en el Manifiesto para el Desarrollo Ágil de Software [15].

Desde entonces se han publicado gran cantidad de documentos que interpretan las declaraciones del manifiesto e introducen prácticas, principios y métodos ágiles. Aludiendo siempre al manifiesto como referencia principal cuando los equipos de desarrolladores pretenden llevar a cabo un desarrollo ágil.

### Motivación

La motivación para la realización del presente proyecto nace de cursar la asignatura de Metodologías de Desarrollo de Software dentro del plan académico del Máster de Ingeniería Web. Durante el desarrollo de la misma se presentan dos metodologías de desarrollo aparentemente opuestas: las metodologías tradicionales o pesadas, y las ligeras o ágiles.

Las primeras centran su atención en mantener una documentación exhaustiva del proyecto, cumpliendo con precisión con el plan previsto y definido en la fase inicial del desarrollo del proyecto. Estas metodologías, también llamadas predictivas, suelen enfatizar la documentación, la planificación y seguimiento riguroso de múltiples actividades llevadas a cabo por diferentes roles dentro del proyecto.

Las metodologías denominadas ágiles, en cambio, son métodos de desarrollo de software en los que las necesidades y soluciones evolucionan a través de una colaboración estrecha entre equipos multidisciplinarios. Y se caracterizan por enfatizar la comunicación frente a la documentación, por el desarrollo evolutivo y por su flexibilidad.

En la actualidad el empleo de prácticas ágiles predomina en el contexto de los proyectos de desarrollo de software. Esta aproximación se ha visto inmensamente extendida y su popularidad va en aumento. Sin embargo, también ha dado lugar a críticas incluso entre los propios creadores e impulsores del movimiento en los últimos años. Entre las más llamativas destaca la publicación “Why Developpers should abandon Agile” de Ron Jeffries, firmante del Manifiesto [1].

Esta situación surge a partir del punto de vista de varios autores del manifiesto: se pueden observar al menos dos mentalidades de acercamiento al desarrollo ágil con resultados potencialmente diferentes. Hay desarrolladores que aplican prácticas ágiles porque creen en los valores y principios del manifiesto, y aquellos que lo hacen en busca de aumentar la eficiencia.

El manifiesto para el desarrollo ágil de software parecía prometer una manera más exitosa de desarrollar software "simplemente" siguiendo los valores y principios originales. Esto a su vez hace que el manifiesto sea especial. Algunos desarrolladores creyeron y siguen creyendo en el manifiesto como el "Santo Grial" para el desarrollo exitoso de software, mientras que otros lo denominan un truco de marketing para vender el comportamiento de desarrollo intuitivo dentro de una nueva fórmula.

Sin embargo, el éxito continuo de las ágiles tuvo varias consecuencias, entre las más comunes: la creación de nuevas tendencias en cómo escalar ágilmente con Scrum of Scrums (SoS) o Scaled Agile Framework (SAFe). Las ideas originales del manifiesto se han ido comercializando cada vez más, muchos desarrolladores y administradores que ahora están adoptando ágil no son conscientes de la diversidad inicial de los métodos ágiles y los principios subyacentes y scrum es a menudo visto como la única práctica ágil.

La llamada transformación ágil de las organizaciones y compañías es un tema que se discute con frecuencia, ya que las formas ágiles de trabajo prometen hacer que las empresas estén preparadas para el futuro. No obstante, los desarrolladores a menudo se declaran "ágiles" cuando solo usan scrum siguiendo sus preceptos de forma estricta. Por estas razones el significado de ágil se interpreta erróneamente y de forma limitada, comercializándose así un concepto incompleto.

Estas malas prácticas son denunciadas públicamente por varios de los autores del manifiesto y constituyen el motivo principal por el que se realiza el presente trabajo

### Objetivos

De lo anteriormente expuesto podemos extraer el propósito perseguido que servirá de guía durante el desarrollo del estudio. El presente Proyecto de Fin de Máster envuelve dos tipos de objetivos: principales y complementarios.

Los objetivos complementarios englobarán: en primer lugar, la realización de un estudio del concepto de agilidad presentando los protagonistas que históricamente participaron en su concepción y cómo el movimiento se materializa en un manifiesto con cuatro valores y 12 principios. Y en segundo lugar, veremos como a partir de la publicación del documento se han expandido las prácticas de la agilidad en el mundo del desarrollo, pasando de ser una comunidad menospreciada por las empresas a ser parte del día a día de los desarrolladores de software en la actualidad.

A partir de aquí descubriremos que, pese a esta gran expansión y reconocimiento de las ágiles, varios de los firmantes del manifiesto han hecho público su descontento con el estado actual de la agilidad y de cómo está afectando a la vida de los desarrolladores. Los objetivos principales serán pues:

Principalmente, analizar cuál es el estado del arte de la agilidad en la actualidad, es decir, estudiar qué ha llevado a aquellos que iniciaron el desarrollo ágil a manifestarse en contra de él, o al menos, en contra del uso que se le da hoy en día. Y contrastar las diferentes opiniones de muchas de las personalidades que en su momento contribuyeron a su creación y las de otros tantos expertos en el mundo del desarrollo y gestión de equipos de software.

Una vez completado el objetivo principal se realizará un análisis de la literatura de la agilidad para comprender cómo se ha transmitido el mensaje y cómo ha podido ser un factor determinante en el estado del desarrollo ágil tal y como lo conocemos actualmente.

### Contenido

El estudio presentado en este proyecto se encuentra estructurado en tres ejes principales con una serie de capítulos dedicados a cada uno de ellos, tal y como se indica a continuación:

En la primera sección se introduce al lector en el objeto de estudio de esta investigación. Comprende los dos primeros capítulos:

* El primer capítulo, que corresponde al actual, conforma una breve introducción al trabajo desarrollado y contiene descripciones del planteamiento general del estudio y de la motivación y objetivo general perseguido con la realización de este.
* En el segundo capítulo se presentan las metodologías de desarrollo ágil, poniendo especial interés en las metodologías que históricamente dieron lugar al concepto de agilidad. Seguidamente se presenta Manifiesto por el desarrollo ágil de software, sus valores y principios.

En la segunda parte se lleva a cabo la investigación que da pie al estudio. Comprende los capítulos tercero y cuarto:

* En el tercer capítulo se presenta el estado actual de la agilidad en términos de popularidad y expansión, para luego estudiar cómo está siendo su adopción por parte de las empresas y equipos de desarrolladores.
* En el cuarto capítulo se lleva a cabo una revisión analítica de la literatura ágil. En este se abordan una serie de afirmaciones realizadas por diversos autores del movimiento y se contrastan con datos y estadísticas reales.

La tercera y última parte de la investigación constituye un único capítulo, en el que se presentan las conclusiones generales del proyecto y las previsibles líneas futuras de desarrollo.

## Entendiendo la agilidad

Los métodos ágiles son un subconjunto de métodos iterativos y evolutivos [7] [8] que se basan en la mejora de forma iterativa [10] y obtienen como consecuencia procesos de desarrollo ventajosos [16].

En todos los procesos iterativos cada iteración es considerada un mini-proyecto con contenido autodefinido, más específicamente, con actividades que abarcan el análisis, diseño, implementación y prueba de requisitos [8]. Cada iteración, además, lleva a una versión de lectura y comprobación, que puede ser solo una versión interna del equipo de desarrollo, que integra todo el software en todo el equipo y es un subconjunto en evolución del sistema final.

Esto se debe a que el cliente especifica y adapta sus requerimientos en consecuencia con la evolución de las iteraciones del proyecto. Establecer fechas de entrega frecuentes para estas iteraciones ayuda a que el desarrollo no varíe tan drásticamente y se puede predecir con más facilidad la dirección que tomará el proyecto. Se puede afirmar que el desarrollo es, por tanto, un proceso empírico y protagonizado por el cambio continuo.

Otra área en común entre todas las metodologías ágiles es la importancia del equipo que participa en la realización de los proyectos y reconocen que, más que cualquier proceso o herramienta, las personas que forman parte dicho equipo son finalmente el factor más influyente en el desarrollo del mismo. Brooks ya advierte dicha tesis en The Mythical Man Month [17], "La calidad de las personas en un proyecto, y su organización y gestión, son factores más importantes para el éxito que las herramientas que usan o los enfoques técnicos que toman”.

Las metodologías que originalmente dieron forma a la Agile Alliance fueron:

* Método de Desarrollo de Sistemas Dinámicos (DSDM)
* Desarrollo de Software Adaptativo (ASD)
* Crystal
* Programación Extrema (XP)
* Desarrollo Dirigido por Características (FDD)
* Scrum

### Dynamic Systems Development Method (DSDM)

DSDM es un método Agile que se enfoca en el ciclo de vida completo del proyecto, DSDM (conocido formalmente como Método de Desarrollo de Sistema Dinámico) fue creado en 1994, después de que los gerentes de proyecto que usaban RAD (Desarrollo Rápido de Aplicaciones) buscaron más gobierno y disciplina para esta nueva forma de trabajo iterativa.

Esta metodología se centra en la filosofía de que “cualquier proyecto debe estar alineado con unos objetivos estratégicos claramente definidos y se enfoca en la entrega temprana de beneficios reales para el negocio". Apoyar esta filosofía con los ocho principios permite a los equipos mantener el enfoque y alcanzar los objetivos del proyecto [18].

Los ocho principios de DSDM:

* Enfocarse en la necesidad del negocio.
* Entregar a tiempo
* Colaborar
* Nunca comprometer la calidad
* Construir incrementalmente desde cimientos firmes.
* Desarrollar iterativamente
* Comunicar de forma continua y clara.
* Demostrar control

DSDM aboga por el uso de varias prácticas probadas, que incluyen:

* Talleres Facilitados
* Modelado y desarrollo iterativo.
* Priorización de MoSCoW
* Time-boxing

DSDM es la columna vertebral del examen AgilePM® (gestión de proyectos ágiles acreditada por APMG).

### Adaptive Software Development

Desarrollo adaptativo de software (ASD) es un proceso de desarrollo de software que proviene de una visión distinta basada en desarrollo RAD y creado por Jim Highsmith y Sam Bayer en los inicios de los 90 [19]. El principio de ese proceso es que el estado normal se base en la continua adaptación del proceso de desarrollo al trabajo real. Es decir, permitir a los equipos adaptarse rápida y eficazmente a los requisitos cambiantes o las necesidades del desarrollo con una planificación ligera y un aprendizaje continuo.

El desarrollo adaptativo de aplicaciones reemplaza al proceso del modelo en cascada con una serie repetitiva de ciclos: especulación, colaboración y aprendizaje. Un ciclo de vida de ASD debe estar enfocado en la consecución de una misión y ser tolerante al cambio. Una vez definida la misión, se procede a la planificación del proceso donde se utiliza información de iniciación del proyecto para definir el conjunto de ciclos de lanzamiento (incrementos de software) que serán requeridos para su ejecución. Esta fase se denomina *especulación*.

La especulación se basa en la creación de falsas suposiciones que serán iteradas en fases cortas y posteriormente se inicia la corrección de los errores que puedan surgir de la especulación. De esta manera se establece el proceso de aprendizaje siendo, por tanto, un proceso colaborativo que se desarrolla según avanza el proyecto. Así, a mayor experiencia, mayor maestría en el dominio del problema.

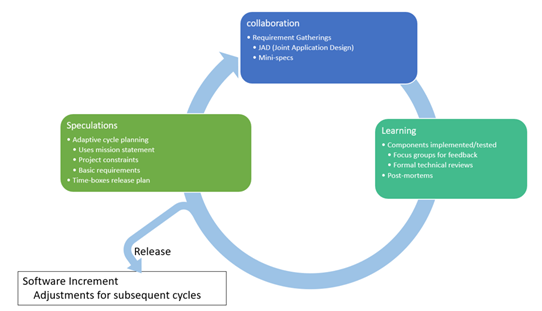


Ilustración 1: Proceso del Desarrollo Adaptativo de Software [20]

Las iteraciones son cortas y se elaboran a través del conocimiento conseguido de falsas suposiciones, corrigiendo los supuestos errores posteriormente. Así, a mayor experiencia, mayor maestría en el dominio del problema.

Las fortalezas de ASD incluyen:

• Enfocado en los usuarios finales, lo que puede llevar a productos mejores y más intuitivos.

• Permite la entrega a tiempo e incluso temprana.

• Fomenta más transparencia entre desarrolladores y clientes.

Las debilidades de ASD incluyen:

• Exige una amplia participación del usuario, pudiendo ser difícil de facilitar.

• Integra las pruebas en cada etapa, lo que puede aumentar los costos de un proyecto.

• El énfasis en la iteración rápida y la retroalimentación continua pueden llevar a un aumento de alcance.

### Scrum

Scrum es un marco de proceso ágil altamente iterativo, creado a principios de la década de 1990 y centrado en el desarrollo orientado a objeto. Sus desarrolladores más conocidos son Ken Schwaber, Jeff Sutherland y Mike Beedle.

El embrión de este marco de desarrollo ágil es el análisis que Nonaka y Takeuchi realiza a principio de los 80 sobre la forma de producción de las principales empresas tecnológicas del momento [21]. En este análisis la nueva forma de trabajo es comparada con el avance en formación de melé (scrum en inglés), una situación de juego de rugby. Este hecho provoca que se acuñe finalmente el término ‘scrum’ para referirse a estas formas de trabajo.

En 1995 Ken Schwaber presenta “Scrum Development Process”, un marco de desarrollo de software basado en los principios de scrum, ya probado tanto por él mismo como por Jeff Sutherland en sus empresas.



Ilustración 2: Roles, Equipos y Eventos en Scrum [22]

#### Artefactos

Hay tres artefactos principales producidos por los equipos de *scrum*, todos estos son abiertamente accesibles e intencionalmente visible para el equipo scrum:

* Backlog del producto. Es una cola priorizada que describe las funcionalidades del producto/proyecto, que evolucionará según las necesidades del cliente.
* Sprint backlog. Es una lista de todas las características empresariales, tecnológicas, mejoras y defectos que se han programado para la iteración actual, llamada *sprint*. El *sprint backlog* también se mantiene en un formato similar a una hoja de cálculo. Los requisitos se desglosan en tareas, cada tarea está descrita en la hoja de cálculo con una breve descripción, quién la originó, quién es el propietario, el estado y el número de horas restantes para completar la tarea. Este se actualiza diariamente.

Las estimaciones pueden variar dependiendo de las estimaciones valoradas por el equipo.

* Tabla de sprint burndown. Es una estimación de las horas restantes para completar las tareas de sprint. Se representan gráficamente y se muestran predominantemente para el equipo. En la siguiente figura se muestra un gráfico de burndown.

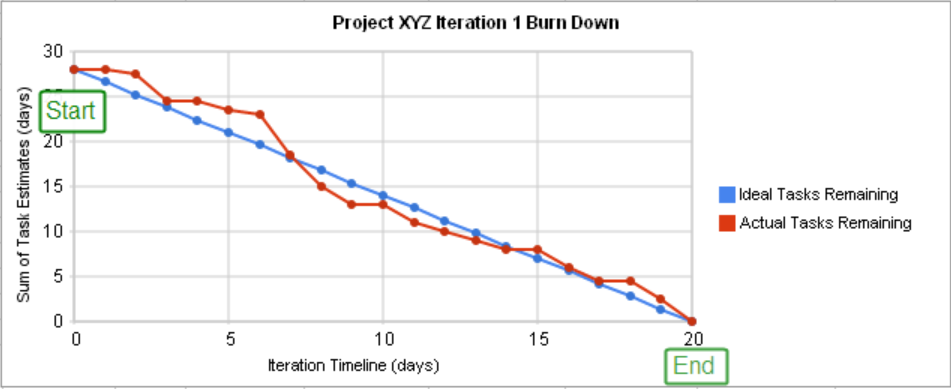


Ilustración 3: Gráfica Burndown [23]

#### Proceso

Scrum fue diseñado para trabajar colaborativamente y su principal particularidad se concentra en ser una estrategia de desarrollo incremental del trabajo a partir de equipos autoorganizados. El equipo se compromete con un objetivo definido para una iteración y se le otorga la autoridad, la autonomía y la responsabilidad para decidir la mejor manera de cumplirlo, que comprometan una alta calidad durante todo recorrido.

Además, la división del desarrollo en *sprints* permite solapar diferentes fases del desarrollo, a diferencia de los métodos en cascada.

Según este marco cada sprint incluye las siguientes cuatro actividades:

• Planificación. Establecer una descripción amplia del proyecto, objetivos generales: aspectos como la funcionalidad, objetivos, riesgos del sprint, plazos de entrega, entre otros.

• Scrum diario. Esta reunión de equipo tiene como objetivo la actualización de status del proyecto: toma de decisiones, mejoras, problemas. Estas reuniones están pensadas para durar como máximo 15 minutos.

• Revisión de sprint.

• Feedback. En esta fase el equipo realiza una retrospectiva del trabajo para analizar y mejorar los procesos realizados.

Al final de cada iteración se realizará una demostración a las partes interesadas.

#### Roles

* Propietario del producto. Es la persona responsable de crear y priorizar la acumulación de productos, elegir qué se incluirá en el próximo sprint y revisar el sistema (con otras partes interesadas) al final de la iteración.
* Scrum master. Conoce y refuerza la iteración, los objetivos del producto, los valores y las prácticas de scrum. Es responsable de llevar a cabo la reunión diaria y la demostración de la revisión escucha el progreso, elimina impedimentos (bloques) y proporciona recursos. El scrum master también es un desarrollador y participa en el desarrollo del proyecto.
* Desarrollador, miembro del equipo scrum. El equipo scrum está comprometido a lograr un objetivo de sprint y tiene plena autoridad para hacer lo que sea necesario para lograr el objetivo. El tamaño de un equipo scrum es aproximadamente 7.

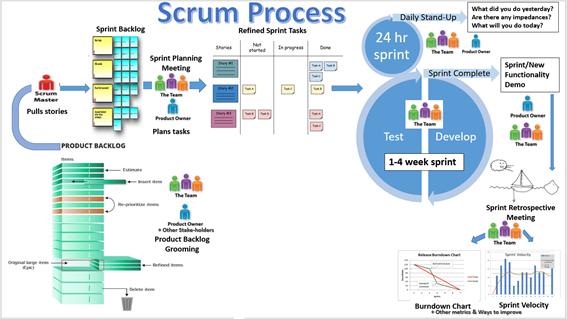


Ilustración 4: El proceso Scrum [24]

Scrum pone mucho menos énfasis en las prácticas de ingeniería, por lo que muchas áreas y sectores combinan su enfoque de gestión de proyectos con las prácticas de ingeniería de programación extrema.

### Crystal

Alistair Cockburn ha sido durante mucho tiempo una de las principales voces de la comunidad ágil. Desarrolló la familia Crystal de métodos de desarrollo de software como un grupo de enfoques adaptados a equipos de diferentes tamaños. Crystal es visto como una familia porque Alistair cree que se requieren diferentes enfoques a medida que los equipos varían en tamaño y cambia la criticidad de los errores [25].

*Crystal* recibe el nombre de familia de métodos porque Cockburn cree que no hay un proceso de desarrollo único e invariante. Se asignan a los diferentes métodos colores en una escala de opacidad ascendente: la versión más ágil es Crystal Clear, seguida de Crystal Yellow, Orange Crystal y Red Crystal.

El gráfico inferior se utiliza para ayudar a elegir un punto de inicio del método. El eje x determina el tamaño del equipo de manera ascendente, lo que supone una gestión de la comunicación cara a cara más y, por lo tanto, mayor es la necesidad de coordinación de documentación, prácticas y herramientas. El eje y aborda el potencial del sistema para causar daños, es decir, la criticidad. El menor impacto del daño es la pérdida de comodidad, la pérdida de dinero discrecional, la pérdida de dinero esencial y, finalmente, la pérdida de vidas. Esta escala se determina basándose en el equipo.

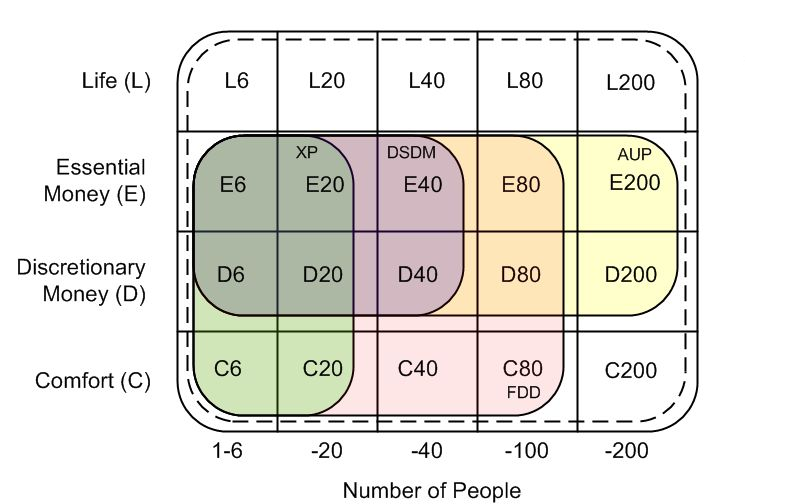


Ilustración 5: Escala Cockburn [25]

Cada metodología tiene un conjunto de prácticas recomendadas, un conjunto básico de roles, productos de trabajo, técnicas y notaciones. A pesar de sus variaciones, todos los enfoques de *Crystal* comparten características comunes y enfatizan la importancia de las personas en el desarrollo de software [2]. Todos los métodos Crystal tienen tres prioridades: seguridad (en el resultado del proyecto), eficiencia, habitabilidad (los desarrolladores pueden convivir con *Crystal*). También comparten propiedades comunes, de las cuales las tres más importantes son: entrega frecuente, mejora reflexiva y comunicación cercana.

Solo hay dos reglas absolutas de la familia de metodologías *Crystal*: en primer lugar, los ciclos incrementales no deben exceder de cuatro meses. En segundo lugar, los talleres de reflexión deben realizarse después de cada entrega para que la metodología se adapte automáticamente.

La prioridad de habitabilidad es una parte importante de la mentalidad de *Crystal*. La búsqueda de Alistair es saber cuál es la menor cantidad de proceso que puede desarrolarse y aún tener éxito con un supuesto subyacente de baja disciplina que es inevitable para los humanos. Como resultado, considera que *Crystal* requiere menos disciplina que la programación extrema, intercambiando menos eficiencia por una mayor habitabilidad y menores posibilidades de fracaso.

### Extreme Programming (XP)

Las raíces de XP se encuentran en la comunidad de Smalltalk, y en particular la estrecha colaboración de Kent Beck y Ward Cunningham a finales de los años 80. Ambos refinaron sus prácticas en numerosos proyectos a principios de los años 90, extendiendo sus ideas de un enfoque de desarrollo de software que fuera tanto adaptativo como orientado a las personas [26].

A lo largo de los 90 Kent continuó desarrollando sus ideas en los contratos de consultoría, destacando en particular el proyecto Chrysler C3 alrededor de 1997. A partir del desarrollo del proyecto C3 es cuando se conoce que nace el concepto de *programación extrema*.

La difusión del concepto de programación extrema, inicialmente a través de descripciones en los grupos de noticias y la wiki de Ward Cunningham, donde Kent y Ron Jeffries pasaron mucho tiempo explicando y debatiendo las diversas ideas.

Finalmente, se publicaron una serie de libros a finales de los años 90 y principios de los 00 que se explicaron en detalle explicando los diversos aspectos del enfoque. La mayoría de estos libros tomaron el libro blanco de Kent Beck como su fundamento. Kent produjo una segunda edición del libro blanco en 2004, que fue una importante re-articulación del enfoque.

#### Valores, principios y prácticas

La metodología XP se basa en cinco valores subyacentes: comunicación, simplicidad, retroalimentación, coraje y respeto.

* La **comunicación**. XP tiene una cultura de comunicación oral y sus prácticas están diseñadas para fomentar la interacción entre el equipo, ya que determina que la mayoría de las dificultades que surgen en un proyecto se deben a malentendidos o faltas de comunicación entre sus miembros. "*Los problemas con los proyectos se pueden rastrear invariablemente a alguien que no habla con alguien lo suficiente sobre algo importante*". [26]
* La **simplicidad**. Es importante diseñar y desarrollar el producto que satisfaga las necesidades del cliente de la forma más simple posible. Centrarse en el valor de diseñar y codificar únicamente lo que se encuentra en los requisitos actuales en lugar de anticipar y planificar los requisitos no declarados.
* La **Retroalimentación**. El equipo de desarrollo recibe comentarios de parte de los clientes al final de cada iteración y lanzamiento externo. Una vez finalizada esta retroalimentación, se produce la siguiente iteración. Además, hay circuitos de retroalimentación muy cortos incorporados a la metodología a través de la programación por pares y el desarrollo basado en pruebas.
* El **Coraje**. Los valores descritos con anterioridad proporcionan las herramientas para que el equipo de desarrollo tenga valor en la toma de decisiones y a la hora de desarrollar sus acciones, incluyendo la realización de tareas en fechas de entrega reales.
* El **respeto**. Los miembros del equipo deben preocuparse por los demás y por el proyecto.

Posteriormente determina catorce principios y veinticuatro prácticas. La idea es que las prácticas son tareas concretas que un equipo puede hacer día a día, mientras que los valores son el conocimiento fundamental y la comprensión que sustenta el enfoque. Los valores sin prácticas son difíciles de aplicar y se pueden aplicar de tantas maneras que es difícil saber por dónde empezar. Las prácticas sin valores son actividades de rutina sin un propósito.

Se necesitan valores y prácticas, pero hay una gran brecha entre ellos: los principios ayudan a cerrar esa brecha. Muchas de las prácticas de XP son técnicas antiguas altamente probadas, pero a menudo olvidadas por muchos, incluyendo la mayoría de los procesos planificados. Además de resucitar estas técnicas, XP las integra en un todo sinérgico donde cada una es reforzada por las otras y tienen un propósito dado por los valores.

Uno de los más llamativos, además de su atractivo inicial, es su fuerte énfasis en la realización de pruebas. Si bien todos los procesos mencionan las pruebas, la mayoría lo hace con un énfasis bastante bajo. Sin embargo, XP establece los tests en la base del desarrollo, ya que cada programador determina pruebas a medida que escriben su código de producción.

Las pruebas se implementan en un proceso continuo de integración y construcción que produce una plataforma altamente estable para el desarrollo futuro. Este enfoque de XP, a menudo descrito bajo el título de Test Driven Development (TDD), ha sido influyente incluso en proyectos y estrategias que no han adoptado mucho más de XP.

#### Herramientas formales

A pesar de que XP basa sus principios en la comunicación oral, hay determinados conocimientos que deberán comunicarse a través de documentación escrita, así como el propio código. Esta formalización de la comunicación a través de documentos también ocurre cuando los equipos de desarrollo son sistemas grandes, son sistemas de alto riesgo o sistemas que requieren capacidad de auditoría por razones legales o de ingeniería de confiabilidad de software, ya que no es un procedimiento recomendado. Esta decisión permite volver a visitar las herramientas periódicamente como parte de un proceso XP más rastreable.

Algunas de estas herramientas son:

* Tarjetas de historias de usuario. Son tarjetas de papel que contienen breves descripciones de requisitos a modo de índice (características, arreglos, no funcionales). Es una forma de determinar de forma clara los requisitos entre el desarrollador y el cliente. Así como la prioridad del cliente y la estimación de recursos del desarrollador.
* Lista de tareas. Una lista de las tareas de uno a tres días de duración que se construyen a partir de las historias de usuario y que se completará en cada iteración.
* Tarjetas CRC [27] (opcionales). Son tarjetas de índice en papel en la que se registran las responsabilidades y los colaboradores de las clases que pueden servir como base para el diseño de software. Estos son identificados durante un diseño sesión de lluvia de ideas / juegos de rol con múltiples desarrolladores. CRC significa Clase-Responsabilidad-Colaboración.
* Pruebas de aceptación del cliente. Son descripciones textuales y casos de prueba automatizados desarrollados por el cliente. El equipo de desarrollo demuestra la finalización de una historia de usuario y la validación de los requisitos del cliente al pasar estos casos de prueba.
* Visible Wall Graphs. Los gráficos de progreso generalmente se publican en el área de trabajo del equipo y se utilizan para fomentar la comunicación y las revisiones. Estos gráficos de progreso a menudo implican cuántas historias se completan y / o cuántos casos de prueba de aceptación se están aprobando*.*

#### Roles

* Manager. Es responsable del equipo y sus problemas. Forma el equipo, obtiene recursos, administra personas y problemas e interactúa con grupos externos.
* Coach. Es la persona encargada de enseñar a los miembros del equipo el funcionamiento del proceso de XP según sea necesario. Interviene en caso de que surjan problemas; supervisa si se está siguiendo el proceso. Su perfil es típicamente de programador y no de gerente.
* Rastreador. Recopila regularmente la historia del usuario y el progreso del caso de prueba de aceptación de los desarrolladores para crear los gráficos del muro visible. El rastreador es un programador.
* Programador. Escribe código, diseña y determina pruebas; identifica y estima tareas e historias.
* Evaluador. Ayuda a los clientes a escribir y desarrollar pruebas. Esta persona también puede ser un programador.
* Cliente. Escribe historias y pruebas de aceptación; recoge las historias para un lanzamiento y para una iteración.

Hay una gran cantidad de publicaciones sobre programación extrema. Un área de confusión, sin embargo, es el cambio entre la primera y la segunda edición del libro blanco. La segunda edición es una "re-articulación" de la programación extrema, en el sentido de que el enfoque sigue siendo el mismo, pero se describe en un estilo diferente. La primera edición, con cuatro valores, doce prácticas y algunos principios importantes, pero en su mayoría ignorados, tuvo una gran influencia en la industria del software y la mayoría de las descripciones de programación extrema se escribieron en base a la descripción de la primera edición.

### Lean and Kanban

#### Lean

Mary Poppendieck (y su esposo Tom) se han convertido en partidarios activos de la comunidad ágil, en particular al observar las superposiciones y las inspiraciones entre la producción lean y el desarrollo de software.

El movimiento de la fabricación en masa fue iniciado por Taiichi Ohno en Toyota y a menudo se lo conoce como el Sistema de Producción de Toyota. Aunque fueron los autores del famoso libro del MIT “La máquina que cambió el mundo” (1990) quienes eligieron el término “lean” para describir cualquier práctica eficiente de gestión que minimizase el desperdicio. Dentro de estas prácticas eficientes se encuentran las técnicas de desarrollo de productos japonesas como las desarrolladas en Toyota. Tal fue el paralelismo entre ambos procesos que el término lean es actualmente sinónimo del método de fabricación de Toyota.

La producción *lean* fue una inspiración para muchos de los primeros agilistas, siendo los Poppendieck los más notables al describir cómo interactúan estas ideas, siendo, además, quienes aplicaron el término lean al desarrollo de software y su asociación a los valores ágiles, y lo popularizaron. [28]

Al igual que la metodología ‘*Just in time’* (JIT) de Toyota, los Poppendieck también hacen hincapié en la eliminación de desperdicios, eliminar la burocracia en el desarrollo de productos, fomentar el aprendizaje por ciclos cortos y frecuentes, iteraciones rápidas, etc. El principal resultado es la obtención de una retroalimentación que ‘tira’ del producto y no se estanca en documentación y requisitos estrictos que lideren el desarrollo general.

En el libro, “*Lean Software Development*” Mary y Tom también destacan los siete principios del desarrollo de software lean, así como un conjunto de 22 instrumentos y herramientas y las comparaciones con otras prácticas ágiles

* Optimizar el todo
* Eliminar desperdicios
* Calidad en la construcción
* Aprender constantemente
* Entregar rápido
* Involucrar a todo el mundo
* Seguir mejorando

El desarrollo *lean* de software no se considera una metodología en el sentido convencional, si no una síntesis de principios y buenas prácticas, una la filosofía de construcción de sistemas de software que puede ayudar a mejorar la calidad**.**

#### Kanban

Kanban es una palabra japonesa que tiene un significado parecido a ‘tarjetas visuales’ y es también el nombre que recibe la técnica creada en Toyota para controlar el avance del trabajo en el contexto de una línea de producción. Su objetivo es gestionar las tareas para obtener una mejora garantizada. [29]

Esta herramienta JIT ha sido trasladada al desarrollo de software junto con la filosofía Lean y sus principales reglas son:

* Visualizar el trabajo y las fases del ciclo de producción/flujo de trabajo.
* Determinar el límite de los trabajos en curso.
* Medir el tiempo necesario para cumplimentar una tarea.

El inicio de esta herramienta comienza con la división del trabajo en partes, hecho que también ocurre en la metodología Scrum, y la generación de una pizarra/tablero donde visualizar de estas tareas. El tablero deberá tener tantas columnas como fases tenga el ciclo de producción, mientras que las tarjetas de tareas deben contener cierta información como descripción y duración de la tarea. Las tarjetas irán ‘pegándose’ y moviéndose a través de estas columnas conforme avanza el proyecto y se completan las fases.

De esta forma la visualización del proyecto ayuda de forma clara a determinar el estado de la producción.

### El manifiesto ágil

Diecisiete críticos de desarrollo de software de procesos se reunieron el 12 de febrero de 2001 en Snowbird, Utah, para la búsqueda de una mejora de las técnicas y procesos en dichos procesos de desarrollo [15]. Este conjunto se considera como la alianza ágil.

Fue en aquel momento dicha alianza documentó su declaración de valor de la siguiente manera:

*‘Estamos descubriendo mejores formas de desarrollar software haciéndolo y ayudando a otros a hacerlo. A través de este trabajo hemos llegado a valorar:*

*Individuos e interacciones sobre procesos y herramientas.*

*Software de trabajo sobre documentación completa.*

*Colaboración del cliente en la negociación de contratos.*

*Responde al cambio sobre el siguiente plan.*

*Es decir, mientras hay valor en los elementos de la derecha, valoramos los elementos de la izquierda más.’*

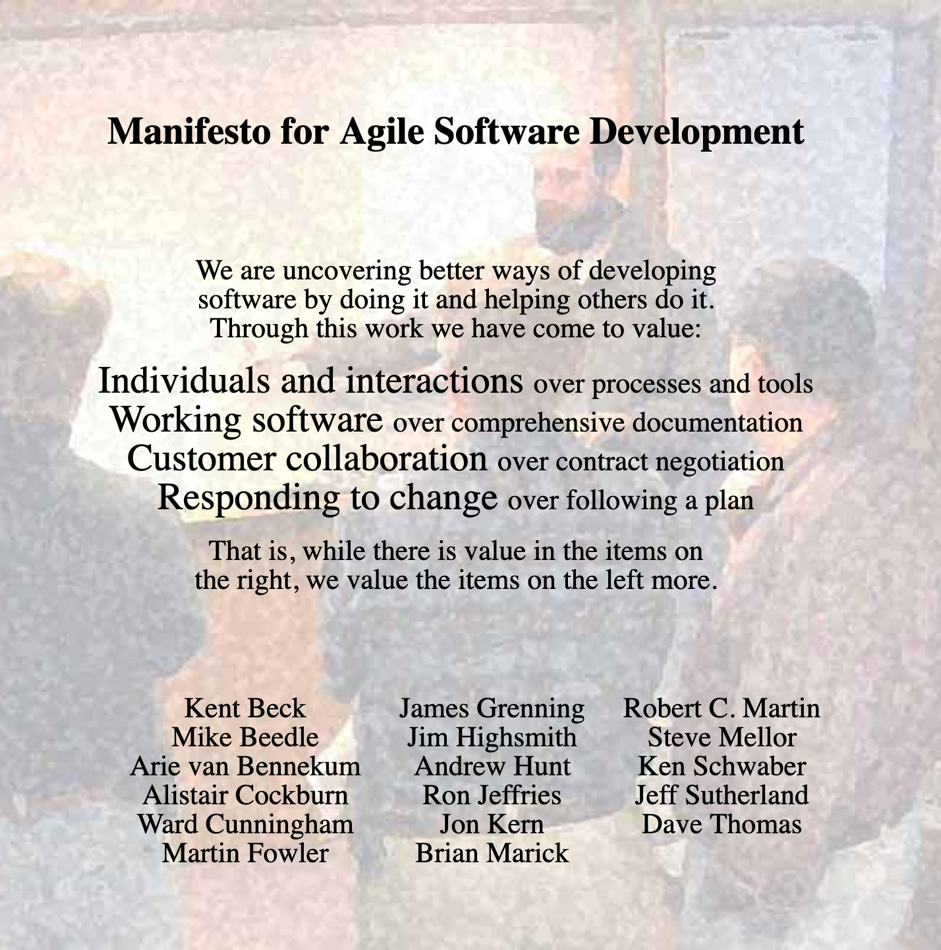


Ilustración 6: Manifiesto por el desarrollo ágil de software [15]

#### Valorar más a los individuos y sus interacciones que a los procesos y las herramientas

Para la filosofía ágil el conocimiento y la aportación que las personas dan al proceso de desarrollo son primordiales. Los procesos, técnicas, herramientas son una guía que mejoran la eficiencia, pero sin la contribución del equipo no se producen resultados.

Las técnicas y metodologías son las que han de adaptarse al equipo humano y sus características, no viceversa.

#### Valorar más el software funcionando que la documentación exhaustiva

Una de las bases son las iteraciones cortas, ya que poder evaluar y comprobar resultados de forma periódica sobre el proyecto, ayuda a desarrollarlo con una perspectiva más realista y completa. Adaptando el proceso al desarrollo. La documentación planeada con anterioridad sobre los requisitos que necesitará un proyecto no permite dicha adaptabilidad.

No se menosprecia la necesidad de documentos, pero se resalta que su importancia es menor a la del producto que funcione. Ya que la comunicación directa entre un equipo de desarrollo producirá soluciones que un documento preestablecido no podrá abarcar.

#### Valorar más la colaboración con el cliente que la negociación contractual

El objetivo de las prácticas ágiles es conseguir que el cliente se involucre con el equipo y colabore, ya que están pensadas para proyectos que son difíciles de definir con detalle desde el principio debido a la velocidad de cambio del entorno.

Esta participación del cliente es importante en el proceso debido a que lo enriquece a través de esta retro-información, teniendo en cuenta que los requisitos de este tipo de proyectos serán muy inestables durante el desarrollo.

#### Valorar más la respuesta ante el cambio que seguir un plan

Los modelos ágiles que están pensados para entornos inestables, con un factor inherente al cambio y una evolución rápida y continua, valoran que el equipo tenga la capacidad de respuesta a dichos cambios y sean capaces de afrontar las situaciones con adaptabilidad y anticipación.

## La agilidad hoy

### Cómo Agile se convirtió en sustantivo y perdió sus valores con el paso del tiempo.

#### Del desprecio a corriente principal

De acuerdo con Martin Fowler [27], la agilidad se ha convertido en algo comúnmente establecido. Ya no es exótico. Tampoco está mal visto ni tiene mala reputación como pudiera ser en los primeros momentos. Así lo demuestran diferentes estudios y encuestas [28] [29] [30] [31].



Ilustración 7: Método de desarrollo primario utilizado en la organización de proyectos 2017. [28]

Los resultados de diferentes investigaciones coinciden en que la gran mayoría de las empresas – por encima del 80% – se apoyan en mayor o menor medida las metodologías agiles dentro de sus procesos y que su adopción va, además, en aumento.



Ilustración 8: Experiencia y adopción de las metodologías Agiles en la empresa 2019. [30]

Más de un 85% de los desarrolladores que respondieron a la encuesta de la comunidad de programadores de StackOverflow [31] señalan que utilizan metodologías agiles en la gestión de sus proyectos, lo que concuerda con los estudios realizados por diferentes agencias privadas.

Curiosamente, a pesar de este alto grado de adopción, las empresas revelan estar aún en proceso de adaptación y consideran a su organización por debajo de un alto nivel de competencia con prácticas ágiles [30].

#### Un éxito a nivel superficial

En los últimos veinte años la expansión y crecimiento en la utilización de métodos ágiles podría verse como un triunfo, sobre todo teniendo en cuenta que los llamados Alianza Agile tenían pocas expectativas tras la reunión [32]. Pensaron que algunas personas leerían el Manifiesto, y tal vez se publicarían algunos artículos, pero que se desvanecería al igual que muchos otros movimientos en el software. La realidad, sin embargo, fue muy diferente y los signatarios vieron sus expectativas ampliamente superadas por la gran acogida que tuvo el movimiento desde un primer momento.

En contraposición, este aparente éxito en la adopción de las metodologías ágiles de desarrollo es en opinión de Martin Fowler [27] un proceso incompleto que se establece en un nivel superficial. Además de Martin un número creciente de “agilistas”, que incluyen a más signatarios destacados del Manifiesto Agile original, han hecho pública su preocupación con el estado de cosas.

Andy Hunt comenta, en la misma línea que Fowler, que tras la publicación del manifiesto el resultado no fue el esperado [33]. En su opinión cuando se redactó el manifiesto en 2001 y se lanzó el movimiento de desarrollo de software ágil, los autores esperaban ver una oleada de nuevos métodos, nuevas prácticas y formas de agilidad. Estas ideas novedosas seguirían idealmente las pautas del manifiesto, pero introducirían nuevos enfoques y ayudarían a avanzar en el desarrollo de software.

Por el contrario, varios firmantes consideran que eso no es lo que finalmente está sucediendo. Se constata una adopción de prácticas individuales que favorecen el desarrollo ágil de software, pero implementadas de forma aislada. Lo que anteriormente fueron consideradas prácticas controvertidas, como el *pair programming* o prácticas de higiene básicas como el control de versiones, que antes no siempre se seguían, se han popularizado enormemente. En general, estas acciones parecen haber tenido un efecto muy positivo en muchas organizaciones de desarrollo, pero Ron Jeffries afirma que es mucho menor de lo que se puede llegar a lograr [34].

#### El cambio de tendencia, Flaccid Agile

Un peligro identificado en los procesos de transformación hacia la agilidad, en los que están sumergidas hoy en día la mayoría de las empresas de la industria, es adoptar ciertas pautas sencillas. Las partes duras que requieren esfuerzo y disciplina constante quedan excluidas en estas adopciones, lo que quiere decir que todo aquello que rodea a la excelencia técnica y a las buenas prácticas de programación está perdiendo protagonismo en el movimiento ágil.

Este fenómeno es denominado “Flaccid Agile” o, en el caso concreto de Scrum, “Flaccid Scrum” por Andy Hunt [35] y Martin Fowler [36]. Este sobrenombre viene a denunciar la tendencia actual de adoptar un marco de trabajo como Scrum, sin prestar el nivel de atención necesario a la calidad del software producido. Traduciéndose en disminución progresiva de la productividad a medida que avanza el proyecto debido a la complejidad de agregar nuevas funcionalidades en un diseño deficiente. En última instancia esto desemboca en una deuda técnica que puede resultar paralizante.

Martin Fowler menciona específicamente Scrum porque desde su punto de vista es bajo este marco de trabajo dónde más a menudo ha observado este fenómeno. Esta situación se ve agravada por el hecho de que Scrum es un proceso que se centra en las técnicas de gestión de proyectos y omite deliberadamente cualquier práctica técnica en contraste con Extreme Programming, por ejemplo.

Recientes estudios [37] confirman que las tendencias en cuanto a metodologías empleadas han cambiado en los últimos veinte años. En sus inicios, la década de 1999-2009, las prácticas de Programación Extrema se encontraban entre las diez prácticas ágiles más utilizadas. En los últimos años 2010-2016 Scrum se encuentra en el centro de las implementaciones de desarrollo de software ágil y sus prácticas son las más extendidas en los proyectos de software.

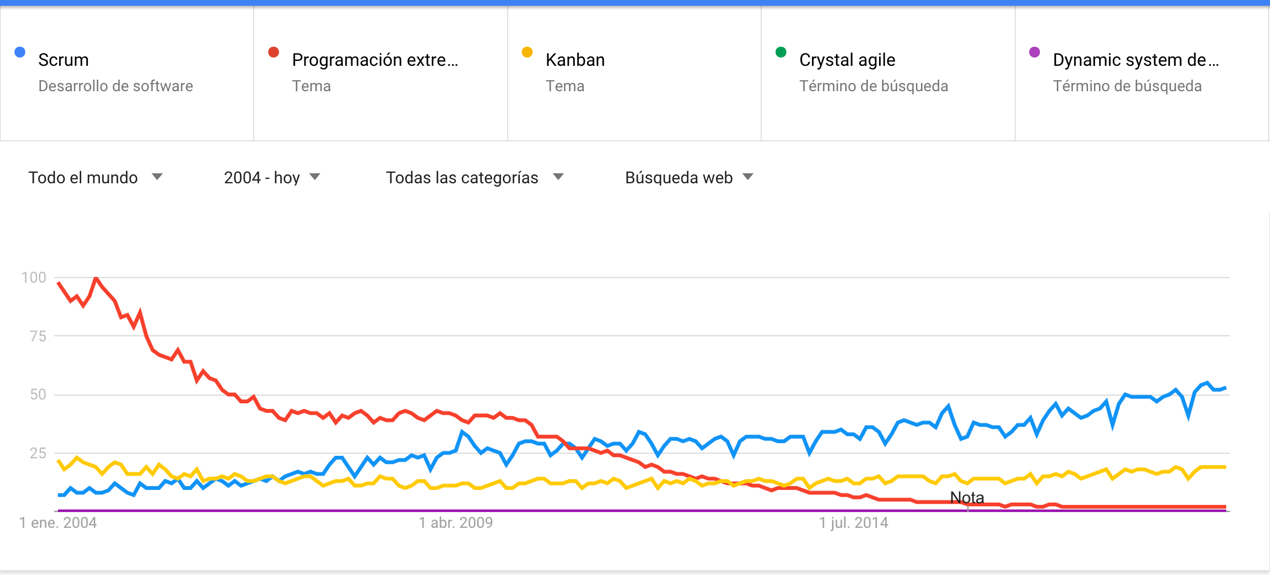


Ilustración 9: Tendencias de búsqueda en los últimos 15 años – Google Trends

En defensa de Scrum, es importante señalar que el hecho de que no incluya actividades técnicas dentro de su alcance no debería llevar a la conclusión de que no se crea que sean importantes. Los autores originales de Scrum siempre han enfatizado la necesidad de tener buenas prácticas técnicas para tener éxito con un proyecto Scrum [38]. No imponen lo que deberían ser esas prácticas técnicas, pero sí que son un requisito indispensable del equipo de desarrollo.

Como respuesta al “Flaccid Agile” surge en los últimos años el movimiento de software Craftmanship o de artesanía del software [39], que ha llegado incluso a tener su propio manifiesto [40]. Esta iniciativa enfatiza en las habilidades técnicas de generar código de calidad por parte de los propios desarrolladores de software, además de promover un código ético para programadores [41].

Para Robert Martin, defensor del movimiento de la artesanía del software, la agilidad tal y como se conoce hoy no es el mismo concepto que se discutió en la famosa reunión en Utah. Ha habido un cambio de enfoque hacia los negocios que se posiciona más lejos de la tecnología, aspecto que se consideró fundamental en la reunión.

#### Transformación de los valores

Robert recuerda que Kent Beck dijo en Snowbird que el principal motivo detrás de Extreme Programming era “curar la brecha entre la programación, la tecnología y el negocio o la empresa”. En su opinión, esa división no se ha curado, al menos no por la corriente más popular y extendida de la agilidad.

Dave Thomas ilustra el deterioro de los principios de la agilidad en el contexto de la gramática [14]. Según sus palabras, La palabra ágil o “Agile” se usa como un nombre cuando es un adjetivo. Dave recuerda como el titulo original del manifiesto es “The Manifesto for Agile Software Development” aunque finalmente se ha popularizado como “The Agile Manifesto”.

En su opinión este giro lingüístico es parte del problema. “Do Agile Right” o “Agile for Dummies” son solo dos de los innumerables ataques contra el idioma que presenta la palabra. Agile no es un sustantivo, es un adjetivo, y debe calificar algo más. "Hacer ágil" es como decir "Hacer naranja".

De esta forma Dave simboliza como, una vez que el Manifiesto se hizo popular, la palabra ágil se convirtió en un imán comercial para cualquier persona con recetas a proponer, horas para facturar o productos para vender. Se convirtió en un término de mercadeo, utilizado para mejorar las ventas de una serie de empresas de consultoría. En este sentido Martin Fowler identifica al “Complejo Industrial Ágil” como el causante de muchos de los problemas a los que se enfrenta a la agilidad hoy. Para los firmantes del manifiesto, el concepto ágil deja de tener significado cuando se convierte en una marca.

#### Mercantilización y Branded Agile

Las múltiples formas del Agile comercial son confusas y pueden dar la impresión de que la agilidad en si misma es confusa. En general, la agilidad tal y como se conoce incluye algunas metodologías genuinas y algunas más comerciales, que se alejan de los principios iniciales y que nacieron para dar respuesta a la demanda por parte de la empresa.

Una forma particularmente desafortunada de "marca ágil" se puede encontrar al tratar de aplicar gestión Ágil a escala. Estos son esquemas destinados a ayudar a las empresas que tienen equipos que implementan prácticas ágiles y desean resolver la tensión entre los equipos ágiles y los sistemas administrativos de la organización como estrategia, planificación, presupuesto, recursos humanos, finanzas, que suelen ser típicamente monolíticos. y burocráticos.

Estos autodefinidos marcos de trabajo generalmente se presentan como "escalamiento ágil". El problema aquí es que, si la empresa está pensando en "escalar ágil", ya está en el camino equivocado. Originalmente la agilidad trata de convertir a los grandes sistemas monolíticos a pequeñas tareas que puedan ser ejecutadas por pequeños equipos de autogestión centrados en el cliente.



Ilustración 10: PortFolio SAFe [43]

Una variante especialmente llamativa es Scaled Agile Framework o SAFe. Este modelo de gestión que se apoya en XP y Scrum como marco de trabajo a nivel del equipo de desarrollo, añade una serie de capas de burocracia, en la cual el cliente está casi totalmente ausente.

Este modelo, que está cada día más extendido en las grandes empresas, ofrece a la administración la potestad de llamarse ágil y seguir haciendo lo que siempre ha hecho. Esencialmente, subordina a los equipos ágiles a la burocracia, en lugar de hacer lo necesario para lograr la agilidad empresarial, es decir, transformar los grandes sistemas monolíticos enfocados internamente en arreglos donde los presupuestos, recursos humanos, finanzas, etc. son flexibles y enfocados externamente en apoyo a los equipos ágiles.

### Hecho por desarrolladores impuesto por las empresas.

Un elemento recurrente en la disconformidad con el estado de la agilidad son las imposiciones de Agile en los equipos, que parece ser una práctica común en la actualidad. Esta opinión está bastante extendida y es respaldada por un número cada vez mayor de líderes de pensamiento ágil [27] [33].

\*\* Developpers should abandon Agile.

Imponer un proceso en un equipo es completamente contrario a los principios del software ágil, y lo ha sido desde su inicio. Existe en la actualidad un conflicto entre los valores de Agile y el enfoque en los procesos de negocios (impuestos) relacionados con Agile.

Una de las características clave de los métodos ágiles es que ponen en primer lugar a las personas que participan en el proyecto. Reconocen que las personas y cómo trabajan juntas es el factor principal en el desarrollo de software, y que los procesos son un factor secundario. Esto se refleja en el primer valor del manifiesto ágil "Individuos e interacciones sobre procesos y herramientas" y se ve reforzado por dos principios del manifiesto:

* Desarrollar proyectos en torno a personas motivadas. Bríndeles el entorno y el apoyo que necesitan, y confíe en ellos para hacer el trabajo.
* Las mejores arquitecturas, requisitos y diseños surgen de equipos autoorganizados.

Una consecuencia importante de estos valores y principios es que un equipo debe elegir su propio proceso, uno que se adapte a las personas y al contexto en el que trabajan. Imponer un proceso ágil desde el exterior despoja al equipo de la autodeterminación que está en el corazón del pensamiento ágil.

Esta noción iría más allá con otro principio del manifiesto: "A intervalos regulares, el equipo reflexiona sobre cómo volverse más efectivo, luego ajusta y ajusta su comportamiento en consecuencia". No solo un equipo debe elegir su propio proceso, el equipo debe controlar cómo evoluciona ese proceso.

Esta noción de un proceso hecho para adaptarse al equipo (y no al revés) es una condición necesaria para los métodos ágiles, pero claramente no es suficiente. Un equipo debe poder elegir libremente su proceso. Pero los métodos ágiles no son los mejores para todas las situaciones, y sería preferible para un equipo trabajar de una manera no ágil que ellos mismos elijan antes que una metodología que le sea impuesta.

*el anti patrón “Methodology Façade” para Ken Schwaber* [44]*.*

*Ron Jeffries* [34]*:*

*Estoy pensando, sin embargo, que cuando llegas a tener una mentalidad ágil, tiene que ser algo que tomes, no es algo que llueva sobre ti. Si alguien nos hubiera dicho sobre el proyecto de Chrisler que teníamos que hacer esas cosas, todos nos habríamos resistido, y nunca lo habríamos conseguido, y ahora no habría agile.*

*Mientras miro a mi alrededor aquí (Agile2016 en Atlanta, GA), aquí hay un montón de cosas que una empresa compra y se impone a los trabajadores. E imponer a los trabajadores no es individuos e interacciones sobre procesos y herramientas*

*Chet Hendrickson* [34]*:*

*El problema con ese enfoque es que no sobrevive.*

### El problema de las certificaciones

Los programas de certificación son hoy en día comunes en la industria del software, aunque esto no evita que haya cierta animadversión hacia ellos. La opinión general es que la certificación tiene poca correlación con las competencias adquiridas por cada persona.

Actualmente, las tres entidades certificadoras más reconocidas a nivel profesional y, por ende, más expandidas son Professional Scrum Master (PSM) de Scrum.org, Certified Scrum Master (CSM) de Scrum Alliance y Agile Certified Professional (PMI-ACP) de PMI.

Las dos primeras tienen su origen en la misma persona, Ken Schwaber. Ken es uno de los creadores de Scrum que, junto con Jeff Sutherland, definió las versiones iniciales de Scrum presentadas por ambos formalmente en la OOPSLA del 95. Juntos crearon también la organización Scrum Alliance en la que comenzaron a legitimar profesionales de Scrum con la certificación CSM.

En 2010 Ken decide dejar la Scrum Alliance y fundar el instituto scrum.org para orientarlo hacia el objetivo de divulgar Scrum. Desde este nuevo instituto (scrum.org) se comenzaron a entregar las certificaciones PSM.

Desde 2012 surge un nuevo competidor en liza, la certificación PMI-ACP que, gracias al prestigio histórico de otra de las certificaciones de la misma entidad: la certificación PMP-PMI, está muy bien considerada en el mundo empresarial.



Ilustración 11: Carreras y tipos de certificaciones Scrum [45]

Como norma general el proceso de formación y evaluación de estos programas de certificación constan de un curso con una duración de dos días y la realización de un examen online de preguntas de opción múltiple. Este método de evaluación es tremendamente débil en opinión de muchos expertos [46] [47] [34].

Kent Beck ha expresado públicamente su malestar con las certificaciones tal y como las conocemos [48], dudando de los conocimientos o aptitudes que se puedan adquirir o demostrar mediante este proceso. Kent alude además a la responsabilidad de la entidad certificadora que, para mantener la credibilidad y reputación de la acreditación, debería asegurarse de que cada candidato es apto y merece dicho certificado [26].

En opinión de Martin Fowler [50], para que una certificación resulte útil, necesita necesariamente una correlación con la competencia en lo que certifica. Martin apunta que en la mayoría de los casos el no ha visto esta relación, y lo que es peor, incluso ha encontrado casos en los que la correlación sería negativa.

La situación actual es que los esquemas de certificación, incluso los útiles, son propensos a la mercantilización. Si puede obtener reconocimiento por un diploma de certificación, existe una buena oportunidad de lucrarse: cursos, evaluaciones, libros, etc. Lamentablemente, no parece haber mucha correlación entre la capacidad de ganar dinero de una certificación y su utilidad. Este es un problema común, no solo en la industria del software, que la certificación se ha convertido en una industria en sí misma. Fomentando la proliferación de exámenes y calificaciones que, cada vez más, están ahí para ayudar a los márgenes de ganancia de las empresas de certificación y pruebas.

Surge entonces el debate ¿Es razonable que una persona competente obtenga una certificación inútil? la realidad es que una certificación a menudo se usa como una puerta de entrada en un proceso de selección, incluso si es deficiente. Como resultado, las personas competentes a menudo necesitan una certificación inútil para obtener una entrevista. Se podría argumentar que esto hace que la certificación sea interesante, al menos desde el punto de vista económico, puesto que puede ofrecer oportunidades laborales, pero esto sería completamente al margen la correlación de competencia.

Los expertos no se muestran especialmente contrarios al hecho de certificarse como tal, lo que encuentran defectuoso es el sistema actual. Un esquema de certificación útil, uno con una correlación de competencia respetable, sería algo positivo en opinión de Martin Fowler [50], especialmente si tuviera un enfoque amplio. Tal esquema facilitaría contratar a alguien para incorporarlo a un equipo.

En este momento, la única manera de determinar si alguien es un buen programador es encontrar otros buenos programadores para evaluar su capacidad. Dicha evaluación es difícil, requiere mucho tiempo y cada organización contratante debe repetirla en sus procesos de selección de personal. Esta tarea se vuelve particularmente complicada para cualquier departamento de recursos humanos, encargados por norma general de la contratación y cuyos miembros no son programadores.

Kent recomienda como modelo de certificación el utilizado por La Leche League [49], similar al propuesto por Dave Snowden en Cynefin [51]. Según este modelo tanto el acreditado como la entidad son responsables de su comportamiento. La acreditación sería una forma de informar de que aquello se sostiene en el currículum es cierto y que la persona en cuestión está alineada con los principios y prácticas de la organización certificadora.

En este caso concreto, para formar parte de la organización deberías ser invitado por un miembro en primer lugar. Tanto este anfitrión como una tercera persona evaluadora deben asegurar que el candidato posee los conocimientos y habilidades necesarias.

El proceso de evaluación incluye:

* Evaluación de los conocimientos técnicos, sociales y organizacionales del candidato para desempeñar su labor.
* Acompañamiento del candidato en su desempeño y critica de sus actuaciones.
* Revisión de un documento
* Interacción social con otros miembros.
* Ser introducido públicamente al resto del grupo en una conferencia regional.
* Acompañamiento y soporte.

Puede ser posible crear un programa de certificación que realmente se relacione con la competencia, pero el ámbito de las certificaciones todavía tiene problemas particulares para los métodos ágiles.

En un proceso impulsado por un plan, es decir, un desarrollo que ha sido previamente planificado, el punto es que la conformidad con el proceso es esencial. Por lo tanto, un esquema de certificación puede probar que un equipo o una organización hace un buen trabajo de conformidad con un proceso definido. Sin embargo, en un mundo ágil, cada proceso sigue la autoadaptación, es decir, se espera que el equipo modifique el proceso para adaptarlo a sus condiciones locales conforme se desarrolla el software. Esto hace que cualquier certificación sea mucho más difícil de diseñar [31].

### La agilidad, ese gran desconocido: prácticas vs. principios

En sus inicios, James Highsmith definió la agilidad como “la capacidad de una organización para reaccionar o responder ante cambios en su entorno más rápido que la tasa de estos cambios” [2]. Esta definición sintetiza el propósito último del desarrollo ágil para una empresa o equipo. Sin embargo, en lugar de exponer este propósito, la agilidad se explica actualmente como una serie sintética de prácticas (véase Scrum, XP, Lean), o bien como un conjunto de propiedades en oposición a otras (véase el manifiesto). Esto ha terminado resultando en la aplicación de pautas sin una visión clara de objetivo a conseguir, aplicar por aplicar.

Lo que denuncian los impulsores y firmantes del manifiesto es que la tendencia actual es adoptar de forma estricta alguna de estas metodologías o un conjunto de ellas para ajustarse perfectamente a aquello que predica o bien el manifiesto o lo que establece una metodología concreta. Esta rigidez conlleva la contradicción de los fundamentos de la agilidad.

La constante iteración y adaptabilidad en los procesos y software ágiles son el fundamento que el colectivo ha olvidado según Hunt. Esto se debe a que las personas que se inician en el desarrollo tienden a seguir fielmente reglas libres de contexto y ciertos métodos relacionados con la agilidad para intentar dominarla. En lugar de seguir la metodología ágil a ciegas, debería analizarse las necesidades de cada uno en particular para ajustar posteriormente el método [54].

Distintos firmantes han bautizado de diferente manera esta inquietud con respecto a la adulteración de la agilidad que se está llevando a cabo hoy día: Martin Fowler lo llama “Dark Agile”, “Faux Agile” según Ron Jeffries [42].

Andy Hunt cita a la Dra. Patricia Benner [34], autora de From Novice to Expert, como la mejor definición que capta el espíritu ágil cuando habla sobre la naturaleza de cómo capacitar a las personas en las diferentes prácticas existentes: *“Las prácticas nunca pueden objetivarse o formalizarse por completo, ya que siempre deben elaborarse de nuevo en relaciones particulares y en situaciones reales.”* [53]*.*

El mensaje que Andy Hunt valora de Benner es el alejamiento de las formalizaciones, ya que, tanto el desarrollo ágil como sus prácticas deben evolucionar para satisfacer necesidades específicas en circunstancias específicas (contexto)*.*

En este sentido Chet Hendrickson reconoce que probablemente la forma en la que se está enseñando la agilidad no sea la más adecuada y que, por ello, se esté fracasando [34]. La pedagogía empleada hasta ahora se basa en aplicar sistemáticamente una serie de técnicas a partir de las cuales, se presupone, los practicantes llegarán a sintetizar y retrotraer las razones por las que se llevan a cabo.

Ken Schwaber y Mike Beedle recomiendan encarecidamente que se sigan las prácticas hasta que entienda por qué y cómo funciona Scrum [51]. Los valores y principios quedarían en un segundo plano en este proceso, cuando paradójicamente son a la vez la raíz y la finalidad del movimiento. Esto iría además en contra del primer principio del Manifiesto por el Desarrollo Ágil de Software: Individuos e interacciones sobre procesos y herramientas [15].

Para ilustrar este fenómeno P. Kruchten utiliza a modo de ejemplo una analogía de definición de una carretera [52]. Según sus palabras podríamos tratar de definir una carretera como algo compuesto de rocas trituradas y alquitrán. O como una superficie que es más negra que blanca, más lisa que ondulada… O bien se podría definir una carretera como un componente de un sistema de transporte que permite a los vehículos circular de un punto a otro. Según sus palabas, las propiedades y los componentes de dicha carretera deberían emanar de esta tercera definición, permitiendo así nuevas y novedosas soluciones en el diseño y ejecución.

James O. Coplien afirma, además, que en su experiencia las herramientas que conducirían a un desarrollo ágil o a una gestión ágil se emplean olvidando de dónde surgieron o a qué necesidad responden [47]. Es decir, no solo se están empleando una serie de herramientas sistemáticamente, sino que las herramientas se emplean esperando un resultado por el mero hecho de utilizarlas.

A modo de ejemplo, en opinión de Coplien, el caso de la práctica “Daily meeting” que tiene como finalidad realizar una replanificación diaria de las tareas en curso y a realizar durante la iteración, habría perdido dicho propósito.

La reunión diaria normalmente se lleva a cabo en un espacio vacío con todos los participantes en pie, la teoría es que nadie se sienta demasiado cómodo y, por lo tanto, se reduzca la tendencia a alargar la duración de la reunión. Para James lo que suele suceder en esta ceremonia es que, tras una breve introducción y algún anuncio por parte del responsable, se realizan una serie de interacciones entre este y algún miembro del equipo, en la que se abordan sistemáticamente tres preguntas: ¿Qué se ha hecho desde la última daily? ¿Qué hay planeado hasta la próxima? ¿Qué impedimentos se han encontrado? El problema es que en estas conversaciones solo estarían involucradas dos personas, el resto del equipo suele dejar de escuchar esperando su turno o, en el peor de los casos, si tienen el ordenador, prestan atención a otros asuntos.

Este ritual en el que los miembros del equipo reportan individualmente al responsable del equipo el estado de las tareas estaría muy alejado del propósito del que pretende satisfacer esta práctica. Lo que sucede además es que los responsables, que conducen la reunión sin saber muy bien por qué la realizan, esperan que esto haga crecer la productividad del equipo. Tom de Marco advierte en su libro Peopleware que este tipo de reuniones cortas, ampliamente expandidas, pueden llegar a ser un obstáculo para la efectividad de la organización si carecen de propósito y enfoque [50].

## El discurso Agile

### Catastrofismo y la falsa dicotomía: Ágil o Cascada

El origen del término de “Ingeniería del Software” se le atribuye a Margaret Hamilton, quien desarrollaba el sistema de guía y navegación para la nave espacial Apollo como jefe de la División de Ingeniería de Software del Laboratorio de Instrumentación MIT.

Hamilton explica por qué eligió llamarlo ingeniería de software [57]: “*luché para legitimar el software, de modo que tanto esta ingeniería como los que la construían recibieran el respeto que merecían, por lo que empecé a usar el término “ingeniería de software” para diferenciarlo del hardware y de otras formas de ingeniería. Cuando empecé a usar estas palabras, se consideraban graciosas. Fue una broma recurrente durante mucho tiempo. Les gustaba bromear con mis ideas radicales. El software acabó ganándose el mismo respeto que cualquier otra disciplina*.”

La expresión se popularizó a mediados de la década de los 60 apareciendo incluso como reclamo comercial [58]. Aunque no fue hasta 1968, en la conferencia NATO de nombre homónimo, cuando se oficializó [59]. En ese evento, Friedrich L. Bauer habló por primera vez de la crisis del software, apuntando a la complejidad del software como principal responsable de la cantidad de proyectos fallidos, retrasados o con sobrecostes.

Curiosamente durante aquel acontecimiento Andy Kinslow ya anunciaba que el proceso de diseño del software es iterativo. Lo que Douglas Ross completaba criticando el proceso de desarrollo basado en el concepto de especificar lo que vas a hacer, y luego hacerlo. Según sus palabras, los proyectos que se denominan exitosos son aquellos que han cumplido con sus especificaciones, aun cuando estas se basaron en la ignorancia de los diseñadores antes de comenzar el trabajo. El proceso que Kinslow y Ross criticaban se conocería más tarde como metodología en cascada.

La icónica representación que ilustra el modelo de cascada apareció por primera vez en un artículo de Winston Royce en 1970 sobre problemas en la gestión de proyectos de TI de grandes sistemas de software [11]. Royce escribe justo después de introducir el patrón: “*Creo en este concepto, pero la implementación descrita anteriormente es arriesgada e invita al fracaso*”.

Royce no define el modelo de cascada y lo recomienda, todo lo contrario: identifica el patrón y expone que, si la fase de prueba se realiza una vez finalizado todo el proceso de desarrollo, los fallos encontradas allí probablemente resultarán en un importante rediseño del software. Royce propone en su artículo mejoras, y desaconseja este proceso de desarrollo.

Además, describe los efectos devastadores de este proceso que 30 años más tarde recordaría la literatura Agile: “*Es probable que los cambios de diseño requeridos sean tan disruptivos que los requisitos de software en los que se basa el diseño y que proporcionan la razón para todo se violan. Se deben modificar los requisitos o se requiere un cambio sustancial en el diseño. En efecto, el proceso de desarrollo ha regresado al origen y se puede esperar hasta un 100 por ciento de exceso en el cronograma y / o los costos*”.



Ilustración 12: Modelo en cascada [9]

Esta ilustración de Winston Royce se hizo famosa como el modelo de cascada, sin embargo, lo usó para mostrar cómo el proceso de desarrollo no está funcionando. Royce nunca utilizó el término “cascada”, ya que no existía un método con dicho nombre en ese momento. El término no aparece hasta el artículo "Requisitos del software: ¿son realmente un problema?" Por T.E. Bell y T.A. Thayer en 1976 [60].

Además, ya en 1970 Royce explicó que no se puede crear un buen software sin iteraciones, realizar pruebas exhaustivas y recopilar comentarios de los usuarios lo antes posible.

Años más tarde, las publicaciones y textos de la literatura en defensa de la agilidad aludían al mal estado de la industria para proponer nuevas ideas y herramientas que solucionasen esta situación. Mencionando de forma explícita la metodología en cascada como el causante de la crisis del software.

Dicha literatura se apoya en el Chaos Report de 1994 [61] de la compañía Standish Group. El informe una visión del éxito o fracaso de los proyectos de software:



Ilustración 13: Chaos Report 1994. Apuntes de MDW – MiW [62]

En la literatura de *scrum* podemos encontrar como se justifica la agilidad como solución a los malos resultados obtenidos con las metodologías empleadas hasta la fecha [65]. Jeff Sutherland y Ken Schwaber insisten en plantear la dicotomía de elegir una metodología ágil enfrentándola a la metodología de cascada [63] [64].

Incluso el último informe de Standish Group sigue haciendo referencia a este enfrentamiento. Ofreciendo unas cifras claramente mejores para los proyecto llevados con metodologías ágiles.



Ilustración 14: Ágil vs. Cascada – Standish Group 2015 [65]

Tras 40 años de evolución de ingeniería de software, dónde desde 1968 ya se anunciaban los problemas de la cascada, las metodologías ágiles dilapidan o se apropian de todo este progreso y confrontan su propia visión con una metodología con conocidas deficiencias.

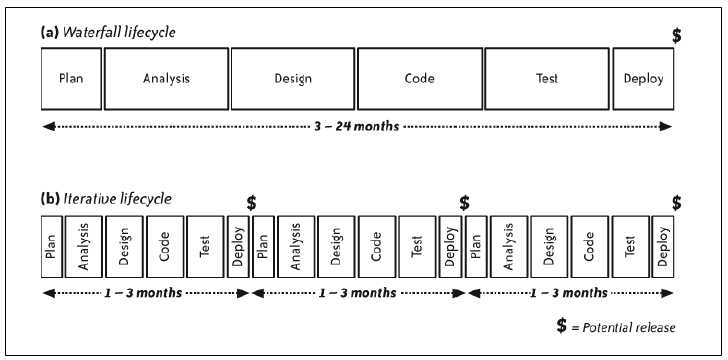


Ilustración 15: Ciclo de vida de procesos en cascada e iterativos [66]

En este ejercicio de confrontación, el movimiento ágil omite otro tipo de metodologías formales como Rational Unified Process (RUP) o también Microsoft Solutions Framework (MSF), Metrica 3… siendo intencionalmente olvidadas o agrupadas junto con cascada.

Curiosamente, el mismo informe del Standish Group de 2015 se expone una tasa de éxito similar tanto a proyectos desarrollados con metodologías modernas, como a aquellos desarrollados con métodos tradicionales. Teniendo los primeros un porcentaje de proyectos fallidos sensiblemente mayor a las difamadas metodologías pesadas.



Ilustración 16: Estadísticas por tipo de proyecto – Standish Group 2015 [65]

Cabe recordar además que estos informes citados en la literatura ágil fueron desacreditados [67] [68]. En resumen, estos resultados son inconsistentes, no están confirmados por otros estudios y se basan en datos de propiedad exclusiva que los investigadores independientes no pueden contrastar. Sin embargo, hasta el día de hoy continúan siendo citados como una justificación para los procesos ágiles.

### Un conjunto de casos de éxito, pocas estadísticas reales. Confundiendo causalidad y casualidad.

### Todo o nada. No estas siendo lo suficientemente ágil

El grado de pertenencia al movimiento ágil es una problemática recurrente que se desprende de los estudios analizados. Es decir, el conjunto de elementos o características que debe cumplir un equipo o son necesarios en un proyecto para que sea considerado ágil son, en el mejor de los casos, imprecisos o inconsistentes.

Los diferentes estudios e investigaciones existentes tienden a medir la agilidad de un proyecto en base a la utilización de las diferentes herramientas y prácticas presentadas en la literatura ágil. Esto ha derivado en la popularización de términos como “prácticas ágiles” o “métodos ágiles” refiriéndose a alguna de las ceremonias o ejercicios característicos de las metodologías de desarrollo ágil.

Parafraseando a Dave Thomas [14], la práctica, por sí sola, no podría aportar agilidad, porque la agilidad es una capacidad o característica propia de una entidad. En el caso del desarrollo de software, un indicador del desempeño de un equipo en un proyecto.

El uso de una o varias prácticas, técnicas o herramientas pueden contribuir o no para desarrollar la "capacidad de cambio" (agilidad) en un proyecto, pero no deberían constituir una medida de la capacidad del equipo, ni podemos considerar a la herramienta ágil per-se.

Tal y como señala Ron Jeffries [37], incorporar estas herramientas podrían contribuir a mejorar el rendimiento de un equipo. Sin embargo, la ganancia es mínima o poco significativa y, además, carecen de sentido si no se persiguen los principios de la agilidad.

Este malentendido no solo se manifiesta en los análisis llevados a cabo por distintas organizaciones. También es observable en la forma -errónea- en la que las empresas adoptan la agilidad. Según Andrew Hunt [38] existe incluso una corriente extrema que critica a quienes no siguen de forma canónica aquello que recomienda o es descrito en alguna metodología en concreto.

Martin Fowler [72] pronosticaba la pérdida de significado de la agilidad en 2006 como parte natural dentro del proceso que denomina difusión semántica. Este fenómeno sucedería cuando una palabra acuñada por un pequeño grupo de personas se propaga a través de la comunidad en general, de manera que se debilita esa definición. Este desgaste podría poner en riesgo su significado por completo, y con ello cualquier utilidad para el término.

El problema se ve además agravado debido al hecho de que no se encuentra en la literatura una definición clara y sintética de lo de que representa el concepto de desarrollo ágil de software. Cada una de las metodologías presenta su propia interpretación.

Ante estos hechos los firmantes del manifiesto aconsejan encarecidamente volver a él como pilar fundamental. Incluso cuando paradójicamente el propio nombre del manifiesto se ha tergiversado: actualmente se le conoce popularmente como el manifiesto ágil.

## Conclusiones y Posibles Ampliaciones

### Conclusiones

En el presente proyecto hemos visto como el movimiento ágil se ha extendido enormemente. Sus prácticas han dejado de ser extravagantes y son ahora comunes, acogidas o al menos adoptadas por la mayoría de los equipos de desarrollo. Una serie de casos de éxito iniciales en las que algunos de los firmantes del manifiesto fueron protagonistas contribuyeron a enardecer el movimiento. Este entusiasmo fue amplificado por el “Complejo Industrial Agile”, que rápidamente convirtió en un negocio lo que se inició como un movimiento que buscaba formas mejores de desarrollar de software.

El cambio hacia la agilidad cambió los caminos de la industria del software, marcando la hoja de ruta de las empresas del sector. Modernizó la forma en que estamos haciendo nuestro trabajo, pero las empresas acogen este cambio más impulsadas por la promesa de la mejora en la eficiencia que por la creencia en los valores o en la mejora de las condiciones de los desarrolladores. Irónicamente, como defienden sus creadores esta eficiencia no puede lograrse en su punto más alto cuando los valores no se tienen en cuenta.

Parece que el enfoque inicial del desarrollo de software ágil se ha olvidado a lo largo del tiempo, y el objetivo ha pasado de centrarse en convertirse en profesionales adaptables, flexibles y ágiles a simplemente seguir un subconjunto de prácticas ágiles prescritas y canónicas. Para muchos de los firmantes la gente ha olvidado por qué surgió este movimiento.

Los problemas a los que se enfrenta hoy en día la agilidad son muy diferentes a los de sus inicios. Destaca una mala comprensión por parte de algunos practicantes, que en ocasiones son forzados a emplear la metodología y al desconocimiento de sus valores.

Los colaboradores originales enfatizan que los métodos ágiles deben ser seleccionados cuidadosamente y que la agilidad y sus prácticas no deben verse como una bala de plata. Subrayan la importancia de considerar la variedad de prácticas y métodos diferentes que influyeron en el desarrollo del manifiesto. Además, mencionan que las personas deberían cuestionar su comprensión actual de ágil y recomiendan reconsiderar las ideas centrales del manifiesto.

*Esa fue siempre la intención con lo que llamamos "ágil". Nunca fue la intención de ser un conjunto fijo y estático de nada. Recuerda las primeras palabras del Manifiesto Ágil, que escribimos hace diez años:*

*"Estamos descubriendo mejores formas de desarrollar software ..."*

*Todavía estamos descubriendo, aun descubriendo. Y tú también deberías estarlo. Lo que me funciona bien no necesariamente funcionará bien para ti; lo que funciona bien para usted ahora no necesariamente funcionará bien la próxima vez* [2]*.*

Piensa en eso por solo un segundo. No puedes hacer "ágil" y hacerte mejor. Solo puedes hacer las cosas mejor y en el proceso volverse más ágil. [55]

* *Agile is something you become (james Coplien)*
* Tal y como recomiendan Andy Hunt y Dave Thomas [56],

No es solo lo que tienes, sino también cómo lo empaquetas. Tener las mejores ideas, el mejor código o el pensamiento más pragmático es, en última instancia, estéril a menos que pueda comunicarse con otras personas. Una buena idea es huérfana sin comunicación efectiva.

Este estudio proporciona impresiones del manifiesto. Revela la incomodidad de algunos de los colaboradores con las tendencias actuales, como la certificación o la comercialización. Además, se identifican direcciones interesantes para una mayor evolución del desarrollo ágil.

Con este trabajo, tratamos de retener este conocimiento para las personas que desean aprender más sobre la opinión de los creadores del manifiesto para el desarrollo ágil de software. La comprensión y el aprendizaje continuo es no solamente una herramienta que aplicar al enfrentarse al desarrollo de software, sino el camino para dominar los conceptos básicos de la agilidad.

### Líneas futuras

El presente trabajo revela varias vías para futuras investigaciones. Entre las cuales destacarían dos direcciones potenciales:

* Por un lado, se podría investigar la influencia de la adopción de la mentalidad ágil en el grado de éxito de los proyectos a nivel internacional. Esta dirección viene motivada del hecho de que las encuestas y estudios encontrados provienen todos empresas de consultoría privadas.

No se ha encontrado ningún análisis realizado por una organización independiente, véase una universidad, que ratifique el grado de adopción de unas metodologías frente a otras y que mida el impacto de estas en el éxito de los proyectos.

Sería necesario investigar, solicitando datos de empresas del sector, el grado de adopción de las metodologías, la satisfacción de los clientes y de todos los actores involucrados en el proceso de desarrollo.

* Por otro lado, asumiendo los beneficios de las prácticas del desarrollo ágil, se propone desarrollar un plan de estudios para acercar la enseñanza actual de ingeniería del software a la filosofía de la agilidad. Este trabajo vendría motivado por la constatación de varios de los firmantes del manifiesto de que la agilidad se está aplicando de forma incorrecta.

Con la idea de subsanar en la medida de lo posible esta situación se pretendería incorporar los preceptos de la agilidad desde el comienzo de la carrera del programador, en este caso en los estudios de ingeniería informática.

Se buscaría que el estudiante pudiera interiorizar durante su periodo formativo conceptos agiles en la forma en la se imparten y evalúan todas las asignaturas y no como una asignatura de metodologías de desarrollo del software.

El plan de estudios debería establecer los mecanismos para que la formación y evaluación recibida fuese iterativa e incremental, revisable y adaptable y donde el trabajo del alumno es realizado mediante la colaboración en equipos autoorganizados, e inmersos en un proceso compartido de toma de decisiones.

# Referecias

|  |  |
| --- | --- |
| [1] | B. Boehm, «Get Ready for Agile Methods with Care,» *IEEE Computer,* vol. 35, nº 1, pp. 64-69, 2002. |
| [2] | J. Highsmith, Agile Software Development Ecosystems, Addison-Wesley, 2002. |
| [3] | B. W. Boehm, «A Spiral Model of Software Development and Enhancement,» *IEEE,* vol. 21, nº 5, pp. 61-72, 1988. |
| [4] | B. Boehm, Software Engineering Economics, Prentice Hall, 1981, pp. 4-21. |
| [5] | V. R. Basili y A. J. Turner, «Iterative Enhancement: A Practical Technique for Software Development,» *IEEE Transactions on Software Engineering,* vol. 1, nº 4, pp. 266-270, 1975. |
| [6] | R. Fairley, Software Engineering Concepts, McGraw-Hill, 1985. |
| [7] | C. Larman, Agile and Iterative Development: A Manager's Guide, Addison Wesley, 2004. |
| [8] | C. Larman y V. Basili, «A History of Iterative and Incremental Development,» *IEEE Computer,* vol. 36, nº 6, pp. 47-56, 2003. |
| [9] | L. Williams y A. Cockburn, «Special Issue on Agile Methods,» *IEEE Computer,* vol. 36, nº 3, 2003. |
| [10] | C. Larman y V. Basili, «Iterative and incremental developments. a brief history,» *IEEE,* vol. 36, nº 6, pp. 47-56, 2003. |
| [11] | W. W. Royce, «Managing the development of large Software Systems,» *IEEE WESCON Proceedings,* pp. 1-9, 1970. |
| [12] | W. Swartout y R. Balzer, «On the inevitable intertwining of specification and implementation,» *Communications of the ACM,* vol. 25, nº 7, pp. 438-440, 1982. |
| [13] | G. Booch, Software Engineering with ADA, Addison Wesley Longman, 1983. |
| [14] | D. Thomas, *Agile is dead, long live agility,* https://pragdave.me/blog/2014/03/04/time-to-kill-agile.html, 2014. |
| [15] | K. Beck, M. Beedle, A. v. Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland y D. Thomas, «Manifesto for Agile Software Development,» 2001. [En línea]. Available: https://agilemanifesto.org/. |
| [16] | B. Curtis, «Three Problems Overcome with Behavioral Models of the Software Development Process,» *International Conference on Software Engineering, Pittsburg,* pp. 398-399, 1989. |
| [17] | F. P. Brooks, The Mithical Man-Month, Addison Wesley, 1995. |
| [18] | J. Stapleton, DSDM: The Method in Practice, Addison-Wesley, 2003. |
| [19] | J. Highsmith, Adaptative Software development, New York: Dorset House, 1999. |
| [20] | S. Mohammed, *Adaptive Software Development (ASD) - An Agile Process,* https://www.linkedin.com/pulse/adaptive-software-development-asd-agile-process-shahab-mohammed/, 2018. |
| [21] | H. Takeuchi y I. Nonaka, «The New New Product Development Game,» *Harvard Business Review,* vol. January/February, pp. 285-305, 1986. |
| [22] | J. P. -. Wikipedia, *Las Reglas de Scrum,* https://es.wikipedia.org/wiki/Scrum\_(desarrollo\_de\_software)#/media/Archivo:Ficha\_scrum.png. |
| [23] | Wikipedia, Project Burndown chart, https://en.wikipedia.org/wiki/Burn\_down\_chart. |
| [24] | S. Mohammed, *Scrum Process Model as part of agile software development methodology,* https://www.linkedin.com/pulse/scrum-process-model-part-agile-software-development-shahab-mohammed/, 2018. |
| [25] | A. Cockburn, Agile Software Development: The Cooperative Game, Addison-Wesley, 2001. |
| [26] | K. Beck y C. Andres, Extreme Programming Explained, Boston: Addison-Wesley, 2005. |
| [27] | D. Bellin y S. S. Simone, The CRC Card Book, Massachusetts: Addison-Wesley, 1997. |
| [28] | M. Fowler, *Agile in 2018,* Agile Australia: https://www.youtube.com/watch?v=G\_y2pNj0zZg, 2018. |
| [29] | H. P. Enterprise, «Agile is the new normal, adopting Agile project management,» Hewlett Packard Enterprise Development LP, 2017. |
| [30] | F. Dynamics y C. technologies, «How Agile and DevOps enable digital readiness and transformation,» Freefrim Dynamics, 2018. |
| [31] | C. VersionOne, «13th annual State of Agile Report,» CollabNet VersionOne, 2019. |
| [32] | S. Overflow, «Developer Survey Results,» Stack Overflow, https://insights.stackoverflow.com/survey/2018/#work-which-methodologies-do-developers-use, 2018. |
| [33] | R. C. Martin, *The Future of Programming,* Expert Talks Mobile: https://www.youtube.com/watch?v=ecIWPzGEbFc, 2016. |
| [34] | A. Hunt, *Uncomfortable with Agile,* CrossTalk, The Journal of Defense Software Engineering, 2016. |
| [35] | C. Hendrickson y R. Jeffries, *Chet Hendrickson & Ron Jeffries: XP Turns 20 and What Have We Learned?,* Agile2016 in Atlanta, GA: https://www.youtube.com/watch?v=unDxq76JcvE, 2016. |
| [36] | A. Hunt, *The Failure of Agile,* Toolshed Technologies: https://toolshed.com/2015/05/the-failure-of-agile.html, 2015. |
| [37] | M. Fowler, *FlaccidScrum,* https://martinfowler.com/bliki/FlaccidScrum.html, 2009. |
| [38] | R. Vallon, B. J. d. S. Estácio, R. Prikladnicki y T. Grechenig, «Systematic literature review on agile practices in global software development,» *Information and Software Technology,* vol. 96, pp. 161-180, 2018. |
| [39] | S. A. Alvares, *An Interview with Agile Manifesto Co-Author and Scrum Co-Founder Jeff Sutherland,* https://www.infoq.com/articles/sutherland-scrum/, 2019. |
| [40] | Wikipedia, *Artesanía de software,* https://es.wikipedia.org/wiki/Artesanía\_de\_software. |
| [41] | *Manifesto for Software Craftmanship,* http://manifesto.softwarecraftsmanship.org/#/en, 2009. |
| [42] | R. Martin, *The Craftsman's Oath,* SCLConf in London: https://www.youtube.com/watch?v=17vTLSkXTOo, 2018. |
| [43] | R. Jeffries, *Developers Should Abandon Agile,* https://ronjeffries.com/articles/018-01ff/abandon-1/, 2018. |
| [44] | D. Leffingwell, *Scaled Agile Framework – SAFe for Lean Enterprises,* https://www.scaledagileframework.com/#. |
| [45] | K. Schwaber, *Methodology Façade Pattern,* Ken Schwaber's Blog: Telling It Like It Is: https://kenschwaber.wordpress.com/2010/10/20/methodology-facade-pattern/, 2010. |
| [46] | S. Alliance, *Certification Types & Tracks,* https://www.scrumalliance.org, 2019. |
| [47] | M. Fowler, Agile Certification, https://martinfowler.com/bliki/AgileCertification.html, 2004. |
| [48] | J. O. Coplien, *The Dehumanisation of Agile and Objects,* GOTO Berlin: https://www.youtube.com/watch?v=ZrBQmIDdls4, 2017. |
| [49] | K. Beck, *Leaving Facebook,* Being Human: https://www.youtube.com/watch?v=fH4gqsIYzyE, 2018. |
| [50] | «La Leche League,» [En línea]. Available: https://www.llli.org. |
| [51] | D. Snowden, *From Agile to agility,* Lean, Agile & Scrum Conference, Zurich: https://www.youtube.com/watch?v=xF6qH8PnxAc, 2018. |
| [52] | T. DeMarco y T. Lister, Peopleware: productive projects and teams, Addison Wesley, 2013. |
| [53] | K. Schwaber y M. Beedle, Agile Software Development with Scrum, Pearson, 2001. |
| [54] | P. Kruchten, «Contextualizing Agile Software Development,» de *EuroSPI 2010 conference*, Grenoble, 2010. |
| [55] | P. E. Benner, From novice to expert: excellence and power clinical nursing practice, Menlo Park, California: Addison-Wesley, 1984. |
| [56] | A. Hunt, *Stop Practicing and Start Growing,* Toolshed Technologies: https://toolshed.com/articles/2016-07-11-stop\_practicing\_and\_start\_growing.html, 2016. |
| [57] | D. Thomas, *Agile @ 10,* https://pragprog.com/magazines/2011-02/agile--. |
| [58] | D. T. Andrew Hunt, The pragmatic programmer from journeyman to master, Addison Wesley, 1999. |
| [59] | J. R. Hancock, «Entrevista a Margaret Hamilton, la pionera de la programación que llevó el Apolo a la Luna,» *Verne, El Pais,* p. https://verne.elpais.com/verne/2014/12/11/articulo/1418314336\_993353.html, 25 December 2014. |
| [60] | A. A. Oettinger, «letter to the ACM membership,» *Communications of the ACM,* vol. 9, nº 8, 1966. |
| [61] | P. Naur y B. Randell, «Software Engineering,» de *NATO Conference on Software Engineering*, Garmisch, 1968. |
| [62] | T. E. Bell y T. A. Thayer, «Software requirements: Are they really a problem?,» *ICSE '76 Proceedings of the 2nd international conference on Software engineering, IEEE,* pp. 61-68, 13-15 October 1976. |
| [63] | *The CHAOS Report,* The Standish Group, 1994. |
| [64] | A. Yagüe y E. G. Pardo, *Apuntes de la asignatura de metodologías de desarrollo web,* Madrid: Universidad Politécnica de Madrid, 2019. |
| [65] | K. Schwaber y J. Suthrland, Software in 30 days: How Agile Managers Beat the Odds, Delight their Customers, And Leave Competitors in Dust, New York: John Wiley & Sons, Inc., 2012. |
| [66] | J. Sutherland, Scrum: The Art of Doing Twice the Work in Half the Time, London: Random House Business, 2014. |
| [67] | *The CHAOS Report,* The Standish Group, 2015. |
| [68] | J. Shore y S. Warden, The Art of Agile Development, O'Reilley, 2008. |
| [69] | R. L. Glass, «The Standish Report: Does it Really Describe a Software Crisis?,» *Communications of the ACM,* vol. 49, nº 49, pp. 15-16, 2006. |
| [70] | J. L. Eveleens y C. Verhoef, «The Rise and Fall of the Chaos Report Figures,» *IEEE Software,* vol. 26, nº 1, pp. 30-36, 2010. |
| [71] | K. Schwaber, *The State of Agile,* https://www.youtube.com/watch?v=8WXT7\_cHsXI: The Central Ohio Agile Association (COHAA), 2014. |
| [72] | A. Hunt, *The End of Agile,* Toolshed Technologies: https://toolshed.com/articles/2011-08-01-TheEndOfAgile.html, 2011. |