

**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](#_Toc11705910)

[1. Antecedentes 3](#_Toc11705911)

[2. Motivación 3](#_Toc11705912)

[3. Objetivos 3](#_Toc11705913)

[4. Contenido 4](#_Toc11705914)

[Capítulo 2. Entendiendo la agilidad 5](#_Toc11705915)

[1. Dynamic Systems Development Method (DSDM) 5](#_Toc11705916)

[2. Adaptive Software Development 5](#_Toc11705917)

[3. Scrum 5](#_Toc11705918)

[4. Crystal 6](#_Toc11705919)

[5. Extreme Programming (XP) 6](#_Toc11705920)

[6. Pragmatic programming 7](#_Toc11705921)

[7. Lean and Kanban 7](#_Toc11705922)

[8. El manifiesto ágil 7](#_Toc11705923)

[Capítulo 3. La agilidad hoy 10](#_Toc11705924)

[1. Cómo Agile se convirtió en sustantivo y perdió sus valores con el paso del tiempo. 10](#_Toc11705925)

[2. Hecho por desarrolladores impuesto por las empresas. 15](#_Toc11705926)

[3. El problema de las certificaciones 17](#_Toc11705927)

[4. La agilidad, ese gran desconocido: prácticas vs. principios 19](#_Toc11705928)

[Capítulo 4. El discurso Agile 22](#_Toc11705929)

[1. Catastrofismo y la falsa dicotomía: Ágil o Cascada 22](#_Toc11705930)

[2. Un conjunto de casos de éxito, pocas estadísticas reales. Confundiendo causalidad y casualidad. 25](#_Toc11705931)

[3. Todo o nada. No estas siendo lo suficientemente ágil 25](#_Toc11705932)

[Conclusiones y Posibles Ampliaciones 26](#_Toc11705933)

[1. Conclusiones 26](#_Toc11705934)

[2. Líneas futuras 27](#_Toc11705935)

[Referencias 29](#_Toc11705936)

## Introducción

### Antecedentes

Probablemente el cambio más notable en el pensamiento del proceso del software en los últimos años ha sido la aparición de la palabra "Agile". Hablamos de métodos de software ágiles, de cómo introducir la agilidad en un equipo de desarrollo o de cómo resistir la tormenta inminente de agilistas decididos a cambiar las prácticas bien establecidas.

Este nuevo movimiento surgió de los esfuerzos de varias personas que lidiaron con el proceso de software en la década de 1990, los encontraron deficientes y buscaron un nuevo enfoque para el proceso de software. La mayoría de las ideas no eran nuevas, de hecho, muchas personas creían que se había desarrollado mucho software exitoso de esa manera durante mucho tiempo. Sin embargo, hubo una opinión de que estas ideas se habían reprimido y no se las había tratado con tanta seriedad, particularmente por personas interesadas en el proceso de software. Hubo algunos principios fundamentales que unieron estas metodologías, y estos principios fueron un contraste notable de los supuestos de las metodologías establecidas.

Larman y Basili [1] describen la historia del desarrollo de software iterativo e incremental (IID), comenzando en los años 70 anteriores y terminando con el manifiesto en 2001. Describen 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 Larman y Basili estructuraron la historia ágil en décadas. Según la 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, que se aplicaron todas las prácticas del IID. Señalan que las prácticas como las iteraciones cortas y el desarrollo de primera prueba ya se utilizaron en el proyecto Mercury. Estas prácticas permanecieron presentes en los métodos ágiles como Scrum o XP.

En los años setenta, Royce [2] publicó un artículo que se considera como la base para el modelo de cascada. En su artículo, Royce describe sus opiniones sobre la gestión de grandes desarrollos de software y lo que es “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 [3] [4], destaca el modelo espiral de Boehm [5]. 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. Como señala, "el modelo sostiene que cada ciclo implica una progresión a través de la misma secuencia de pasos". Concluye que el enfoque basado en el riesgo es más adaptable que el enfoque basado en documentos (cascada) y el basado en códigos (desarrollo evolutivo).

Inspirado en las ideas de Barry Boehm, James Martin desarrolló el enfoque de desarrollo rápido de aplicaciones durante la década de 1980 en IBM y finalmente lo formalizó 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, como el Reino Unido, los Países Bajos, Suecia y Dinamarca.

Varias décadas de desarrollo de software y mejores prácticas de enfoques de desarrollo y aplicaciones en la industria terminaron 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 del manifiesto para el desarrollo de software ágil, a menudo denominado "manifiesto ágil".

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 cuando 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 y cumplir con el plan previsto y definido con precisión 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. Pero también ha dado lugar a críticas, en los últimos años incluso entre los propios creadores e impulsores. Entre las más llamativas destaca la publicación “Why Developpers should abandon Agile” de Ron Jeffries, firmante del Manifiesto [1].

Desde el punto de vista de varios autores del manifiesto se podrían observar al menos dos mentalidades de acercarse 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 librea.

Sin embargo, el éxito continuo de ágil tuvo varias implicaciones. Entre otras, las nuevas tendencias como escalar ágilmente con Scrum of Scrums (SoS) o Scaled Agile Framework (SAFe) son temas comunes. 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. Scrum es a menudo visto como la única práctica ágil.

La llamada transformación ágil de las organizaciones es un tema que se discute con frecuencia porque las formas ágiles de trabajo prometen hacer que las empresas estén preparadas para el futuro. Sin embargo, los desarrolladores a menudo se declaran "ágiles" cuando solo usan Scrum por el libro. En este caso, el significado de ágil se interpreta erróneamente, no se entiende y se comercializa.

Estas denuncias realizadas públicamente por varios de los autores del manifiesto es el motivo principal que motiva la realización del 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 varios objetivos: en primer lugar, se realizará un estudio del concepto de agilidad, presentando los protagonistas que históricamente participaron en su concepción, y como el movimiento se materializa en un manifiesto con cuatro valores y 12 principios.

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, 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 analizar cuál es el estado del arte de la agilidad en la actualidad, es decir, estudiar que 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 como esto 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 estudio objeto de esta investigación. Comprende los dos primeros capítulos:

* El primer capítulo, que corresponde al actual, conforma una breve introducción al trabajo desarrollado y contiene una descripción por un lado del planteamiento general del estudio y, por otro, 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

### 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.

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

### Scrum

Scrum también se desarrolló en los años 80 y 90 principalmente con los círculos de desarrollo OO como una metodología de desarrollo altamente iterativa. Los desarrolladores más conocidos son Ken Schwaber, Jeff Sutherland y Mike Beedle.

Scrum se concentra en los aspectos de gestión del desarrollo de software, dividiendo el desarrollo en iteraciones de treinta días (denominadas "sprints") y aplicando un monitoreo y control más estrechos en las reuniones diarias de scrum. Pone mucho menos énfasis en las prácticas de ingeniería y muchas personas combinan su enfoque de gestión de proyectos con las prácticas de ingeniería de programación extrema. (Las prácticas de gestión de XP no son realmente muy diferentes).

### 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.

A pesar de sus variaciones, todos los enfoques de cristal comparten características comunes. Todos los métodos de cristal tienen tres prioridades: seguridad (en el resultado del proyecto), eficiencia, habitabilidad (los desarrolladores pueden vivir con el cristal). También comparten propiedades comunes, de las cuales las tres más importantes son: entrega frecuente, mejora reflexiva y comunicación cercana.

La prioridad de habitabilidad es una parte importante de la mentalidad del cristal. La búsqueda de Alistair (como lo veo) es buscar cuál es la menor cantidad de proceso que puedes hacer y aún tener éxito con un supuesto subyacente de baja disciplina que es inevitable para los humanos. Como resultado, Alistair 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.

Kent continuó desarrollando sus ideas durante los contratos de consultoría, en particular el proyecto Chrysler C3, que desde entonces se conoce como el proyecto de creación de programación extrema. Comenzó a usar el término 'programación extrema' alrededor de 1997. (C3 también marcó mi contacto inicial con la Programación Extrema y el comienzo de mi amistad con Kent).

A fines de la década de 1990, se difundió la palabra de programación extrema, inicialmente a través de descripciones en los grupos de noticias y la wiki de Ward Cunningham, donde Kent y Ron Jeffries (un colega de C3) 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.

XP comienza con cinco valores (Comunicación, Retroalimentación, Simplicidad, Valor y Respeto). Luego los elabora en catorce principios y nuevamente en veinticuatro prácticas. La idea es que las prácticas son cosas 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, probadas y 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 tiene un propósito dado por los valores.

Uno de los más llamativos, además de su atractivo inicial, es su fuerte énfasis en las pruebas. Si bien todos los procesos mencionan las pruebas, la mayoría lo hace con un énfasis bastante bajo. Sin embargo, XP pone las pruebas en la base del desarrollo, ya que cada programador escribe pruebas a medida que escriben su código de producción. Las pruebas se integran en un proceso continuo de integración y construcción que produce una plataforma altamente estable para el desarrollo futuro. El enfoque de XP aquí, a menudo descrito bajo el título de Test Driven Development (TDD), ha sido influyente incluso en lugares que no han adoptado mucho más de XP.

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. He dicho anteriormente que 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.

### Pragmatic programming

### Lean and Kanban

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 magro en la fabricación fue iniciado por Taiichi Ohno en Toyota y a menudo se lo conoce como el Sistema de Producción de Toyota. La producción Lean fue una inspiración para muchos de los primeros agilistas: los Poppendieck son más notables al describir cómo interactúan estas ideas. En general desconfío de este tipo de razonamiento por analogía, de hecho, la separación de ingeniería entre diseño y construcción nos metió en este lío en primer lugar. Sin embargo, las analogías pueden llevar a buenas ideas y creo que las ideas magras han introducido muchas ideas y herramientas útiles en el movimiento ágil.

### El manifiesto ágil

## La agilidad hoy

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

De acuerdo con Martin Fowler [6], la agilidad se ha convertido en algo comúnmente establecido. Ya no es exótico, ni está mal visto como pudiera ser en los primeros momentos. Efectivamente así lo demuestran diferentes estudios y encuestas [7] [8] [9] [10].



Ilustración 1: Método de desarrollo primario utilizado en la organización de proyectos 2017. [7]

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 2: Experiencia y adopción de las metodologías Agiles en la empresa 2019. [9]

Más de un 85% de los desarrolladores que respondieron a la encuesta de la comunidad de programadores de StackOverflow [10] señalan que utilizan metodologías agiles en la gestión de sus proyectos, lo que concuerda con los estudios realizados por diferentes agencias privadas.

Curiosamente, 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 [9].

Esta expansión en los últimos veinte años podría verse como un triunfo, al menos a ojos de aquellos que iniciaron el movimiento. Los asistentes tenían pocas expectativas tras la reunión. 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. Pero la realidad fue muy diferente y los signatarios se vieron sus expectativas ampliamente superadas por la gran acogida que tuvo el movimiento desde un primer momento [11].

Sin embargo, este aparente éxito en la adopción de las metodologías ágiles de desarrollo permanecería a nivel superficial en opinión de Martin Fowler [6]. Para Andy Hunt [12] tras la publicación del manifiesto, el resultado no fue el esperado. El documento fue creado con la idea de cambiar la cara del desarrollo de software para mejor. En 2001, cuando se redactó el manifiesto 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 nuevas ideas seguirían las pautas del manifiesto, pero introducirían nuevas ideas y enfoques y ayudarían a avanzar en el desarrollo de software.

Sin embargo, varios firmantes consideran que eso no es lo que finalmente está sucediendo. En cambio, hemos visto una adopción a mayor escala de prácticas individuales que favorecen el desarrollo ágil de software, incluso anteriormente prácticas controvertidas, como la programación en pares, e incluso prácticas de higiene básicas, como el control de versiones, que antes no siempre se seguían. En general, esto parece haber tenido un efecto muy positivo en muchas organizaciones de desarrollo, pero seguramente es menos de lo que muchos de ellos habían esperado.

Un peligro identificado de los procesos Agile es hacer las partes fáciles y no las partes duras. He visto a varios equipos decir que están adoptando Scrum, pero cuando te fijas bien, están haciendo las partes triviales como las reuniones diarias y llamando a alguien un Scrum Master y se están perdiendo ingredientes difíciles pero importantes como hacer que su equipo sea multidisciplinar o reflexionar sobre sus prácticas regularmente. Del mismo modo, sería fácil tomar las partes triviales de XP no hacer el diseño inicial, por ejemplo, e ignorar algunos elementos importantes de la disciplina como un cliente disponible permanentemente y refactorización constante.

Sin embargo, recientes estudios [13] revelan que las tendencias en cuanto a metodologías empleadas han cambiado. 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.

Un número creciente de “agilistas”, incluidos los signatarios destacados del Manifiesto Agile original, han hecho pública su preocupación con el estado de cosas.

Distintos firmantes han bautizado de diferente manera esta inquietud con respecto a la implementación de la agilidad que se está llevando a cabo hoy día: Martin Fowler lo llama “Dark Agile” [14], “Faux Agile” según Ron Jeffries [15], “Flacid agile” en palabras de Andy Hunt [16], o el anti patrón “Methodology Façade” para Ken Schwaber [17]. Aunque cada uno de los autores proporcione diferentes matizaciones al respecto, estas denominaciones vienen a ilustrar que la agilidad, tal y como se práctica en muchos casos, no sigue aquello que inicialmente se concibió.

Dicho esto, el ágil de hoy no es el mismo ágil que se discutió en la reunión. Ha habido un cambio hacia los negocios, y lejos de la tecnología, que se consideró fundamental en la reunión. Bob recuerda que Kent Beck dijo en Snowbird que su motivo detrás de la programación extrema, o XP, “era curar la brecha entre la programación, la tecnología y el negocio. Y esa división no se ha curado, al menos no por la ágil corriente principal ”. Los administradores del proyecto y los maestros del scrum se han hecho cargo de la comunidad ágil, lo que significa que una gran parte del movimiento ágil ha abandonado el lado tecnológico de la ecuación de Beck. Bob encuentra esto profundamente perturbador. Los tecnólogos han creado el movimiento de la artesanía del software para corregir el equilibrio, por lo que conceptos como la programación de pares, el desarrollo basado en pruebas y las pruebas funcionales están creciendo en popularidad entre los equipos de desarrollo, aunque todavía están rezagados con respecto al énfasis en la gestión de proyectos que parece haber abrumado a todos.

Si bien la comunidad de desarrollo de software ha visto un enfoque en los últimos años en agile a gran escala, Bob dijo que la reunión no abordó la escala ágil explícitamente. Sin embargo, es escéptico sobre el concepto. “Dudo que exista la adopción de ágile a gran escala. Puede ser que la única forma de adoptar ágile sea a pequeña escala, e incluso en una organización grande terminas con un grupo de equipos ágiles muy pequeños ".

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 [18].

Durante el encuentro trataron de describir aquellas ideas, lo que finalmente resultó en el Manifiesto para el Desarrollo Ágil de Software [19].

Dave Thomas ilustra el deterioro de los principios de la agilidad en el contexto de la gramática. 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”. como Eso es simplemente un error. “Do Agile Right” o “Agile for Dummies” son solo dos de los innumerables ataques contra el idioma inglés que presenta la palabra. No tienen sentido. Agile no es un sustantivo, es un adjetivo, y debe calificar algo más. "Hacer ágil" es como decir "Hacer naranja".

Pero, más allá del problema de la gramática, hay un problema mayor. Una vez que el Manifiesto se hizo popular, la palabra ágil se convirtió en un imán para cualquier persona con puntos a proponer, horas para facturar o productos para vender. Se convirtió en un término de mercadeo, cooptado para mejorar las ventas de la misma manera que palabras como eco y natural. Una palabra que se abusa de esta manera se vuelve inútil: deja de tener significado cuando se convierte en una marca.

Primero, hay una tendencia creciente a tratar la palabra "Ágil" como un sustantivo. La gente dice "estamos haciendo ágil". Pero amigos, ágil no es un sustantivo. Es un adjetivo, que significa poder moverse con rapidez y facilidad. Cuando acuñamos el término por primera vez en nuestra reunión de Snowbird como una forma de describir lo que hicimos, sentimos que la palabra "ágil" también tenía connotaciones de gracia.

Entonces, al menos para mí, realmente no se puede usar "ágil" sin tener que calificar algún tipo de sustantivo: es el proceso, el pensamiento o las interacciones que son ágiles. Y eso es algo en lo que nos gustaría que nos volviéramos a enfocar a medida que reflexionamos y planeamos hacia adelante Ser un desarrollador de software ágil significa hacer las cosas de una manera ágil, no solo nombrar las cosas que haces "ágil". Y eso es lo que hace que todo esto sea tan difícil, pero también tan efectivo. No se puede simplemente comprar un kit de herramientas ágil, ni pegar carteles ágiles, ni pagar por un asesor ágil. En última instancia, lo que hace que las cosas que haces sean ágiles es la forma en que las haces. Y, según mi experiencia, si continúa trabajando para hacer las cosas más rápidas y fáciles (a largo plazo), se volverá más ágil.

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. [20]

* *Agile is something you become (james Coplien)*

Esto duele a todos, pero soy particularmente sensible al daño que causa a los desarrolladores. No es fácil escribir código, y los desarrolladores, naturalmente, están atentos a las cosas que les ayudarán a entregar valor de manera más efectiva. Todavía creo firmemente que adherirse a los valores y prácticas del manifiesto les ayudará en este esfuerzo.

Pero una vez que la palabra ágil deja de tener sentido, los desarrolladores ya no pueden usarla como una guía de lo que es útil en su práctica.

*Creo que la mejor definición que he visto que capta este espíritu ágil proviene de la Dra. Patricia Benner, autora de From Novice to Expert. Hablando sobre la naturaleza de la experiencia y cómo capacitar a las personas en las prácticas del mundo real (enfermería clínica en este caso), dijo: “Las prácticas nunca pueden objetivarse o formalizarse por completo, ya que siempre deben elaborarse de nuevo en relaciones particulares y en situaciones reales. tiempo”* [21]*. Es decir, nunca se puede definir completamente ágil, o sus prácticas, porque están en constante evolución para satisfacer necesidades específicas en circunstancias específicas. Agile debe ser siempre cambiante, siempre cambiante, siempre respondiendo al cambio en el contexto. Como profesional de desarrollo de software ágil, debe seguir pensando y ajustándose. Sugiero humildemente que esta idea fundamental de desarrollo ágil es lo que (colectivamente) hemos olvidado.*

*Esto se debe a que los principiantes de una nueva habilidad tienden a seguir reglas libres de contexto y es probable que se conviertan en fanáticos ágiles que ponen demasiado énfasis en algunos métodos formales vinculados a ágil. Haciendo cosas en las que ya se sienten cómodos, en lugar de pensar por sí mismos. En lugar de seguir la metodología ágil a ciegas, debe ajustarse a las necesidades de cada uno* [22]*.*

*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* [23]*.*

*Según él, los valores ágiles se han corrompido principalmente por tres razones: primero, un complejo agilindustrial en constante crecimiento⁴. Este complejo ágil-industrial simplemente capacita a las personas brevemente, proporciona certificados brillantes de pocos días y los empuja a ocupar puestos de consultoría o gestión.*

*En segundo lugar, un enfoque en la metodología. Esto ha llevado a un énfasis excesivo en las reglas formales a costa de la falta de reconocimiento de la excelencia técnica por parte de ingenieros capaces y experimentados. Y en tercer lugar, centrarse en proyectos en lugar de productos. En lugar de conectar a los desarrolladores con los clientes y centrarse en la calidad, se valora el plazo y la finalización del proyecto*.

*Robert C. Martin señaló recientemente, el enfoque Agile en sí mismo fue una reacción a una crisis en el desarrollo de software* [11]*.*

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 gran 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 3: PortFolio SAFe [24]

Una variante especialmente preocupante es el Scaled Agile Framework o SAFe. Esencialmente esto es una burocracia codificada, en la cual el cliente está casi totalmente ausente. Ahora está muy extendido en las grandes empresas porque le da a la administración el mandato 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 en operaciones. El papel insignificante del cliente en el cuadro anterior es indicativo del problema.

Otra forma preocupante de Agile que ha comenzado a aparecer es algo que se llama “Agile-lite”. Esto apareció en un artículo de Harvard Business Review el año pasado en un artículo que explicaba cómo algunos servicios de recursos humanos intentaban encontrar formas de volverse ágiles. El artículo ofrecía el titular, "HR va ágil". Pero el texto sugería que "HR se está volviendo 'ágil' '. Aprendimos que" HR está aplicando los principios generales de Agile sin adoptar todas las herramientas y protocolos del mundo tecnológico. . ”A juzgar por los ejemplos, parece que“ Agile lite ”significa la adopción de herramientas y prácticas de Agile sin necesariamente implementarlas con una mentalidad Agile. Sin una mentalidad ágil, Agile sigue siendo un conjunto inerte de ceremonias sin vida.

### Hecho por desarrolladores impuesto por las empresas.

*Un síntoma sorprendente de la crisis de Agile son las imposiciones de Agile en los equipos, que parece ser una práctica común en la actualidad. Si Agile es tan grande y realmente otorga más poder y autonomía a los desarrolladores, ¿por qué es comúnmente impuesto por la alta gerencia? En mi experiencia, esto es bastante común y esta impresión está respaldada por un número cada vez mayor de líderes de pensamiento ágil.*

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

*Las verdades del manifiesto ágil parecen sostenerse a sí mismas. Pero las deficiencias en el análisis de por qué exactamente no funcionan según lo planeado son desconcertantes. ¿Somos todos demasiado suaves en ágil? ¿Está el emperador desnudo y nadie se atreve a decir la verdad abiertamente?*

*Un gran error, al bloquear nuestra comprensión de la naturaleza de Agile, es la idea de que Agile está superando el taylorismo como lo sugirió Fowler. Al igual que el mito de la cascada, esto no es cierto, si se examina más detenidamente:*

*El software de escritura e ingeniería es (quizás pronto: ¿solía ser?) Un trabajo de conocimiento altamente calificado. Es intrínsecamente diferente de los bienes de fabricación y no se puede racionalizar fácilmente. Pero trabajar en los procesos ágiles comunes como lo sugieren Scrum o Kanban es lo más parecido a trabajar en una línea de ensamblaje, ya que el software de escritura puede ser:*

*El trabajo se divide en los pasos más pequeños y más fáciles posibles.*

*El ritmo del trabajo es controlado, medido y gestionado.*

*El objetivo final es hacer que el trabajador de software sea desechable por el proceso e incluso la brecha entre los ingenieros altamente experimentados y los miembros menos capacitados del equipo. Esto se hace en beneficio de la productividad y la calidad predecible del producto resultante, de una manera que pretende ser lo más reproducible posible. Sin embargo, al hacer esto, Pioneros y Genios en un extremo y Spaghetti-Script-Cowboys en el otro extremo del espectro ya no son indispensables y están claramente desactualizados.*

*Cortar el trabajo y eliminar la habilidad requerida tanto como sea posible aprovechando procesos predefinidos es el corazón del taylorismo. Pero la industria del software no ensambla en masa el mismo modelo de automóvil una y otra vez como lo hizo el Fordismo del siglo 20 y no puede simplemente copiar la línea de ensamblaje. E incluso la manufactura se alejó de este enfoque. Si observamos algunos otros aspectos de los procesos de producción ágil, vemos más aspectos que también surgieron en la producción en masa moderna, tales como:*

*Controlar y mantener una alta calidad está en el centro de atención.*

*Control total de la productividad, con el objetivo de eliminar las fallas y maximizar el uso de recursos.*

*Integración del personal en producción por un énfasis en el trabajo en equipo.*

*Más autonomía en la toma de decisiones en el sitio de producción real.*

*Jerarquías planas y reducción de énfasis de los símbolos de estado.*

*Todos estos también son componentes elementales de los métodos de gestión modernos, desarrollados en Japón y, a menudo, etiquetados como "Toyotismus". Algunos sociólogos identifican el toyotismo como el posfordismo. Algunos como ni pre- y post-fordismo²⁰. Al igual que con el taylorismo, el toyotismo también es un proceso altamente racionalizado y tienen mucho en común.*

*La cercanía de los procesos ágiles al toyotismo también se puede ejemplificar examinando Kanban, que es la segunda metodología ágil más popular hoy en día. "Kanban" es la palabra japonesa para Signboard, un elemento central en el estilo japonés de producción justo a tiempo y el Kanban aplicado en la producción de Software es un descendiente directo del Kanban que surgió en el Toyotismo. Se aplicó por primera vez al software en Seattle 2007²¹. Pero también Scrum muestra un alto grado de primo con este estilo de gestión.*

*Esos enfoques modernos y magros en la gestión son las claves para el auge de la industria moderna japonesa y su alta competitividad en la industria electrónica y automotriz. El enfoque ágil \* no \* superó el taylorismo, simplemente implementó algo parecido a su forma moderna: manufactura esbelta: el modo Toyota.*

*Ron Jeffries* [25]*:*

*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* [25]*:*

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

### El problema de las certificaciones

Actualmente, las tres más importantes son Professional Scrum Master (PSM) de Scrum.org, Certified Scrum Master (CSM) de Scrum Alliance y Agile Certified Professional (PMI-ACP) de PMI (los mismos de la certificación PMP).

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 que presentaron juntos formalmente en la OOPSLA del 95.

Juntos crearon también la organización Scrum Alliance en la que comenzaron a certificar profesionales de Scrum con la certificación CSM.

En 2010, Ken decide dejar la Scrum Alliance y fundar el instituto scrum.org para intentar orientarlo más hacia el objetivo de divulgar Scrum. Desde este nuevo instituto (scrum.org) se comenzaron a entregar las certificaciones PSM. Desde 2012 hay también un nuevo competidor en liza, la certificación PMI-ACP que está sonando bastante fuerte desde entonces.



Ilustración 4: Carreras y tipos de certificaciones Scrum [26]

*Los programas de certificación son comunes en la industria del software, pero los desarrolladores más capaces que conozco tienen poco respeto por ellos. La opinión general es que la certificación tiene poca correlación con la competencia. Esto se agrava en la comunidad ágil con la asociación de certificación con el CMM, que históricamente es todo menos ágil.*

*Parte del problema con la certificación en general es que los programas de certificación han sido muy débiles. Muchos esquemas de certificación orientados al desarrollador son poco más que preguntas de opción múltiple que son fáciles de configurar y marcar, pero no hacen un buen trabajo de sondear más que la memorización de memoria. También es un problema común, no solo en la industria del software. esa certificación se convierte en una industria en sí misma, lo que fomenta la proliferación de pruebas y actualizaciones que cada vez más están ahí para ayudar a los márgenes de ganancia de las empresas de certificación y pruebas.*

*Ninguno de estos problemas es necesariamente fallas con la certificación en general. Puede ser posible crear un programa de certificación que realmente se relacione con la competencia. Pero la certificación todavía tiene problemas particulares para los métodos ágiles.*

*En un proceso impulsado por un plan, 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. Esto hace que cualquier certificación sea mucho más difícil de diseñar.* [27]

*Recuerdo una conversación sobre cerveza después de XP Universe 2002. Nos preguntamos qué se necesitaría para una certificación de XP. Consideramos que implicaría varias semanas de observación, ver a las personas lidiar con las distintas etapas de un proyecto de software, ver cómo utilizan diversas habilidades, incluida la optimización del proceso. Tal prueba sería costosa de realizar. ¿Estaría la gente dispuesta a pagar por este tipo de programa?*

*A pesar de todos estos recelos, me gustaría que la industria del software encontrara la manera de idear un esquema de certificación significativo. Ayudaría a separar a las personas más capaces y permitiría que las personas más competentes sean mejor recompensadas por su habilidad. No creo que la industria esté en condiciones de idear un solo esquema, de ahí mi escepticismo sobre Swebok. Pero puede ser posible que una escuela particular de desarrollo de software, como XP, encuentre algo. Pero ese aspecto será muy diferente al tipo de programas de certificación que vemos actualmente.*

Curiosamente Kent Beck también ha expresado su malestar con las certificaciones tal y como las conocemos [28], dudando de los conocimientos o aptitudes que se pueden llegar a adquirir en un par de días y haciendo una prueba online. 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 [29].

Kent propone además como modelo de certificación el utilizado por La Leche League [30]. Según este modelo, tanto el acreditado como la entidad son responsables de su comportamiento. La acreditación sería una forma de informar que aquello que dice en el curriculum es cierto, y que la persona en cuestión está alineada con los principios y prácticas de la organización. Para formar parte de la organización debes en primer lugar ser invitado por un miembro.

En este caso concreto, tanto el 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.
* Dave Snowden Cynefin

### 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” [31]. Esta definición sintetiza el propósito último del desarrollo ágil para una empresa. En su lugar, la agilidad se explica como una serie sintética de prácticas (véase Scrum, XP, Lean), o bien como un conjunto de propiedades en oposición a otras (véase el manifiesto).

Lo que denuncian los impulsores y firmantes del manifiesto es que la tendencia actual es adoptar alguna de estas metodologías, o un conjunto de ellas para ajustarse perfectamente a aquello que predica el manifiesto o lo que establece una metodología concreta, sin por ello ser ágil.

James O. Coplien afirma que desde su experiencia las herramientas que conducirían a un desarrollo ágil, o a una gestión ágil se emplean olvidando de donde surgieron, o a qué necesidad responden [32]. Concretamente, tomando como ejemplo el caso del “Daily meeting”, esta práctica tiene como finalidad realizar una replanificación diaria de las tareas en curso y a realizar durante la iteración. En opinión de James, este propósito se ha perdido.

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. Tom de Marco advierte de que este tipo de reuniones cortas pueden llegar a ser un obstáculo para la efectividad de la organización si carecen de propósito y enfoque [33].

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, prestando 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.

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 está fracasando [25]. La pedagogía empleada hasta ahora se centra en una serie de prácticas a partir de las cuales, a base de ser empleadas sistemáticamente, los practicantes sean capaces de 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 [34]. 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 [19].

Para ilustrar este fenómeno P. Kruchten utiliza a modo de ejemplo una analogía de definición de una carretera [35]. 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 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.

* No explanation of how scrum works just recipes
* Volver a los principios

El software está siendo escrito por una gran cantidad de personas que no están de acuerdo con un estándar ético. Entonces vemos cosas como la debacle de Volkswagen, que es profundamente aterradora. Si ese tipo de cosas continúa, es muy probable que nuestra sociedad exija algún tipo de regulación. Y si la sociedad nos regula antes de que nos regulemos a nosotros mismos, será un desastre. Por lo tanto, estaría prestando mucha atención a las personas que se enfocan en el tema de nuestra responsabilidad con la sociedad. ¿Cuáles son nuestras éticas? ¿Cuál es nuestra profesión? ¿Quiénes somos como programadores? ¿Qué reglas tenemos y cómo hacemos cumplir esas reglas? [11]

* Tal y como recomiendan Andy Hunt y Dave Thomas [36],

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 un huérfano sin comunicación efectiva.

## 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 [37]: “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 [38]. Aunque no fue hasta 1968, en la conferencia NATO de nombre homónimo, cuando se oficializó [39]. 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 [2]. 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”.

Sin embargo, Royce nunca usó la palabra cascada ya que no existía un método de cascada en ese momento. El término cascada hasta el artículo "Requisitos del software: ¿son realmente un problema?" Por T.E. Bell y T.A. Thayer en 1976 [40].



Ilustración 5: Modelo en cascada [9]

Esta ilustración de Winston Royce se hizo famosa como el modelo de cascada. Sin embargo, no usó el término cascada en su artículo y lo usó para mostrar cómo el proceso de desarrollo no está funcionando.

No solo eso, sino que 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 [41] de la compañía Standish Group. El informe una visión del éxito o fracaso de los proyectos de software:



Ilustración 6: Chaos Report 1994. Apuntes de MDW – MiW [42]

Podemos encontrar como se justifica la agilidad como solución a los malos resultados obtenidos con las metodologías empleadas hasta la fecha. La literatura ágil insiste en plantear la dicotomía de elegir una metodología ágil enfrentándola a la metodología de cascada [43] [44].

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 7: Ágil vs. Cascada [45]

Tras 40 años de evolución de ingeniería del software, dónde desde 1968 se anuncian los problemas de la cascada, las metodologías agiles dilapidan o se apropian de todo este progreso y confrontan una metodología con conocidas deficiencias al nuevo modelo.

En este ejercicio de confrontación, las metodologías formales como Rational Unified Process (RUP) o también Microsoft Solutions Framework (MSF), Metrica 3… son olvidadas o agrupadas junto con cascada.

* RUP?

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



Ilustración 8: Estadísticas por tipo de proyecto [45]

Cabe recordar además que estos informes citados en la literatura ágil fueron desacreditados [46] [47]. 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

## 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. Pero suponer que Agile es el final de la historia en el desarrollo de software es increíblemente miope. Asumir que la insatisfacción con Agile solo surge de "no hacer las cosas bien" nos impide preguntar por sus defectos inherentes.

Ir post-ágil no es solo buscar un nuevo nombre para la comercialización y reinventarlo o una Difusión semántica como argumentó Fowler. El posmodernismo no es una difusión semántica del modernismo. Es algo que siguió y evolucionó desde el modernismo, al comprender mejor la naturaleza del modernismo y sus limitaciones inherentes.

El movimiento ágil comenzó como una especie de crisis de gestión (proyecto) y la gestión todavía parece ser un problema difícil en TI. Tenemos que profundizar en la crisis de gestión y la crisis de software y volver a visitarla. Por el momento, al parecer, los ideales de la comunidad ágil son difíciles de defender en la dura realidad del mundo de los negocios. Llamar a la responsabilidad individual de cada profesional de TI no es suficiente. Martin demostró que la industria se está expandiendo rápidamente y que los desarrolladores tienen cada vez menos experiencia o en muchos otros campos. En muchos casos pueden carecer de la madurez para hacerlo.

Los colaboradores originales enfatizan que los métodos ágiles deben ser seleccionados cuidadosamente y que los ágiles 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.

Esta publicación 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 y más allá.

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.

# Referencias

|  |  |
| --- | --- |
| [1] | C. Larman y V. Basili, «Iterative and incremental developments. a brief history,» *IEEE,* vol. 36, nº 6, pp. 47-56, 2003. |
| [2] | W. W. Royce, «Managing the development of large Software Systems,» *IEEE WESCON Proceedings,* pp. 1-9, 1970. |
| [3] | 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. |
| [4] | G. Booch, Software Engineering with ADA, Addison Wesley Longman, 1983. |
| [5] | B. W. Boehm, «A Spiral Model of Software Development and Enhancement,» *IEEE,* vol. 21, nº 5, pp. 61-72, 1988. |
| [6] | M. Fowler, *Agile in 2018,* Agile Australia: https://www.youtube.com/watch?v=G\_y2pNj0zZg, 2018. |
| [7] | H. P. Enterprise, «Agile is the new normal, adopting Agile project management,» Hewlett Packard Enterprise Development LP, 2017. |
| [8] | F. Dynamics y C. technologies, «How Agile and DevOps enable digital readiness and transformation,» Freefrim Dynamics, 2018. |
| [9] | C. VersionOne, «13th annual State of Agile Report,» CollabNet VersionOne, 2019. |
| [10] | S. Overflow, «Developer Survey Results,» Stack Overflow, https://insights.stackoverflow.com/survey/2018/#work-which-methodologies-do-developers-use, 2018. |
| [11] | R. C. Martin, *The Future of Programming,* Expert Talks Mobile: https://www.youtube.com/watch?v=ecIWPzGEbFc, 2016. |
| [12] | A. Hunt, *Uncomfortable with Agile,* CrossTalk, The Journal of Defense Software Engineering, 2016. |
| [13] | 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. |
| [14] | M. Fowler, *The State of Agile Software in 2018,* https://martinfowler.com/articles/agile-aus-2018.html, 2018. |
| [15] | R. Jeffries, *Developers Should Abandon Agile,* https://ronjeffries.com/articles/018-01ff/abandon-1/, 2018. |
| [16] | A. Hunt, *The Failure of Agile,* Toolshed Technologies: https://toolshed.com/2015/05/the-failure-of-agile.html, 2015. |
| [17] | 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. |
| [18] | D. Thomas, «The Coding Gnome,» 4 March 2014. [En línea]. Available: https://pragdave.me/blog/2014/03/04/time-to-kill-agile.html. |
| [19] | 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/. |
| [20] | D. Thomas, *Agile @ 10,* https://pragprog.com/magazines/2011-02/agile--. |
| [21] | P. E. Benner, From novice to expert: excellence and power clinical nursing practice, Menlo Park, California: Addison-Wesley, 1984. |
| [22] | A. Hunt, *Stop Practicing and Start Growing,* Toolshed Technologies: https://toolshed.com/articles/2016-07-11-stop\_practicing\_and\_start\_growing.html, 2016. |
| [23] | A. Hunt, *The End of Agile,* Toolshed Technologies: https://toolshed.com/articles/2011-08-01-TheEndOfAgile.html, 2011. |
| [24] | D. Leffingwell, *Scaled Agile Framework – SAFe for Lean Enterprises,* https://www.scaledagileframework.com/#. |
| [25] | 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. |
| [26] | S. Alliance, *Certification Types & Tracks,* https://www.scrumalliance.org, 2019. |
| [27] | M. Fowler, *Should there be a certification program for agile methods?,* https://martinfowler.com/bliki/AgileCertification.html, 2004. |
| [28] | K. Beck, *Leaving Facebook,* Being Human: https://www.youtube.com/watch?v=fH4gqsIYzyE, 2018. |
| [29] | K. Beck y C. Andres, Extreme Programming Explained, Boston: Addison-Wesley, 2005. |
| [30] | «La Leche League,» [En línea]. Available: https://www.llli.org. |
| [31] | J. A. Highsmith, *Agile software development ecosystems,* Boston: Addison-Wesley, 2002. |
| [32] | J. O. Coplien, *The Dehumanisation of Agile and Objects,* GOTO Berlin: https://www.youtube.com/watch?v=ZrBQmIDdls4, 2017. |
| [33] | T. DeMarco y T. Lister, Peopleware: productive projects and teams, Addison Wesley, 2013. |
| [34] | K. Schwaber y M. Beedle, Agile Software Development with Scrum, Pearson, 2001. |
| [35] | P. Kruchten, «Contextualizing Agile Software Development,» de *EuroSPI 2010 conference*, Grenoble, 2010. |
| [36] | D. T. Andrew Hunt, The pragmatic programmer from journeyman to master, Addison Wesley, 1999. |
| [37] | 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. |
| [38] | A. A. Oettinger, «letter to the ACM membership,» *Communications of the ACM,* vol. 9, nº 8, 1966. |
| [39] | P. Naur y B. Randell, «Software Engineering,» de *NATO Conference on Software Engineering*, Garmisch, 1968. |
| [40] | 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. |
| [41] | *The CHAOS Report,* The Standish Group, 1994. |
| [42] | A. Yagüe y E. G. Pardo, *Apuntes de la asignatura de metodologías de desarrollo web,* Madrid: Universidad Politécnica de Madrid, 2019. |
| [43] | 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. |
| [44] | J. Sutherland, Scrum: The Art of Doing Twice the Work in Half the Time, London: Random House Business, 2014. |
| [45] | *The CHAOS Report,* The Standish Group, 2015. |
| [46] | 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. |
| [47] | 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. |
| [48] | K. Schwaber, *The State of Agile,* https://www.youtube.com/watch?v=8WXT7\_cHsXI: The Central Ohio Agile Association (COHAA), 2014. |