Robert C. Martin

# Desarrollo ágil esencial

## Vuelta a las raíces

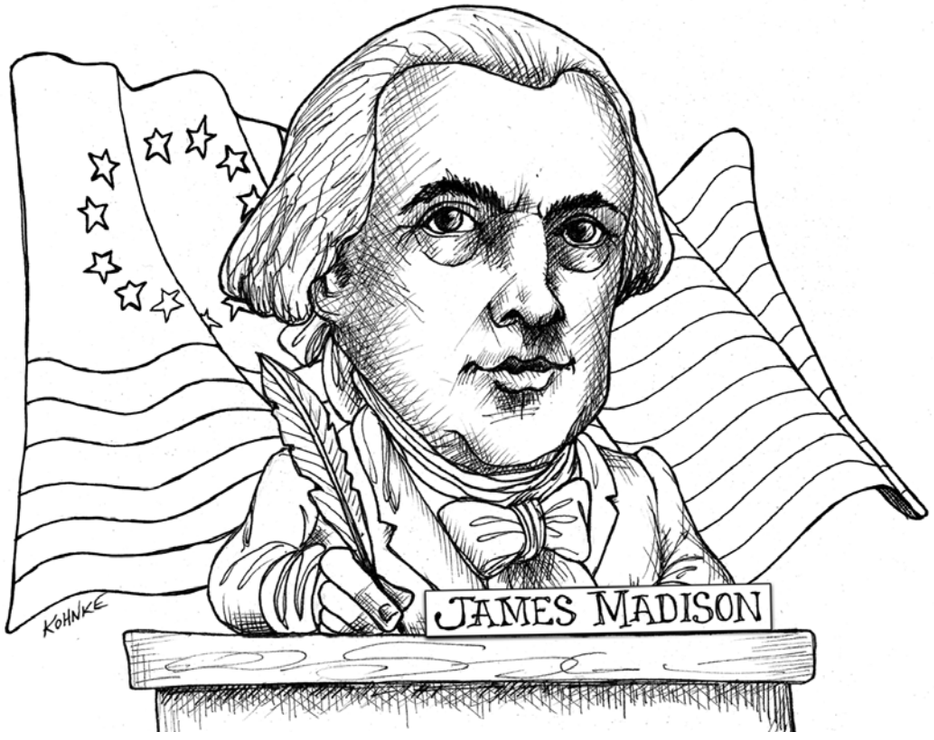
*Con contribuciones de* **Jerry Fitzpatrick, Tim Ottinger, Jeff Langr, Eric Crichlow, Damon Poole** *y* **Sandro Mancuso**

ANAYA MULTIMEDIA

# Índice de contenidos

# 2

# Las razones para usar desarrollo ágil




Antes de adentrarnos en los detalles del desarrollo ágil, quiero explicar lo que está en juego. El desarrollo ágil no solo es importante para el desarrollo de software, sino también para nuestra industria, nuestra sociedad y, básicamente, para nuestra civilización.

A menudo, los desarrolladores y los directores se sienten atraídos por el desarrollo ágil debido a razones efímeras. Quizá lo prueben porque, por algún motivo, les da una buena impresión, o a lo mejor se dejan seducir por las promesas de velocidad y calidad. Estas razones son intangibles, están poco definidas y pueden frustrarse con facilidad. Muchas personas han abandonado el desarrollo ágil solo porque no han experimentado de inmediato el resultado que pensaban que se les había prometido.

Esas razones fugaces no son los motivos por los que el desarrollo ágil es importante. Lo es por razones éticas y filosóficas mucho más profundas, relacionadas con la profesionalidad y las expectativas razonables de nuestros clientes.

## Profesionalidad

Lo que me atrajo del desarrollo ágil en primer lugar fue el elevado compromiso con la disciplina por encima de la ceremonia. Para utilizar bien el desarrollo ágil, había que trabajar por parejas, escribir pruebas primero, refactorizar y ceñirse a diseños simples. Había que trabajar en ciclos cortos, produciendo un resultado ejecutable en cada uno. Había que comunicarse con los negocios con regularidad y continuidad.

Vuelva a fijarse en el círculo de la vida y considere cada una de esas prácticas como una promesa, un compromiso; así verá a qué me refiero. Para mí, el desarrollo ágil es un compromiso de que voy a ponerme las pilas, a ser profesional y a fomentar un comportamiento profesional en toda la industria del desarrollo de software.

Los que formamos parte de esta industria necesitamos aumentar mucho nuestra profesionalidad. Fracasamos con demasiada frecuencia. Enviamos demasiada porquería. Aceptamos demasiados defectos. Ofrecemos soluciones intermedias terribles. Demasiado a menudo, nos comportamos como adolescentes rebeldes con una tarjeta de crédito nueva. En tiempos más sencillos, esos comportamientos eran aceptables porque los riesgos eran relativamente bajos. En los 70 y los 80, incluso en los 90, el coste del fracaso de un software, aunque era elevado, estaba limitado y era controlable.

a la mitad cada seis meses. El director del equipo de aseguramiento de la calidad me enseñaba ese documento preguntándome qué mitad de aquellas pruebas no debería ejecutar.

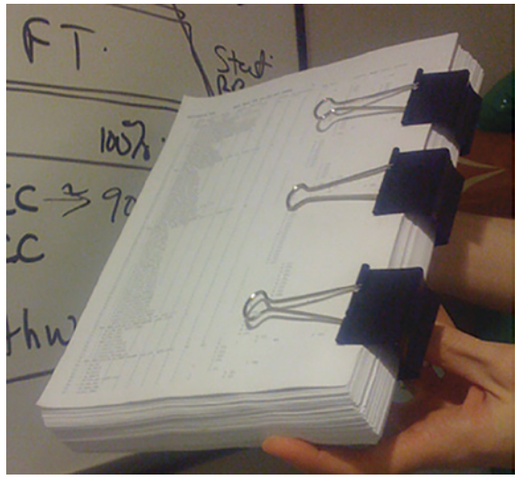


**Figura 2.1.** Índice del plan de pruebas manuales.

Le dije que, al margen de cómo decidiese recortar las pruebas, no sabría si la mitad de su sistema funcionaba.

Este es el resultado inevitable de la realización de pruebas manuales. Las pruebas manuales siempre acaban perdiéndose. Lo que acaba de leer fue el primer mecanismo, y el más obvio, para perder pruebas manuales: estas pruebas son caras y, por tanto, siempre son un objetivo a la hora de recortar.

Sin embargo, existe un mecanismo más insidioso para perder pruebas manuales. Los desarrolladores rara vez entregan su trabajo al equipo de aseguramiento de la calidad a tiempo. Eso significa que el equipo de aseguramiento de la calidad tiene menos tiempo del previsto para ejecutar las pruebas que necesitan realizar. Por tanto, deben elegir qué pruebas les parece más apropiado ejecutar para cumplir el plazo de envío al cliente y, por eso, algunas pruebas no se llevan a cabo. Se pierden.

Además, los humanos no somos máquinas. Pedir a un humano que haga lo que puede hacer una máquina es caro, ineficiente e inmoral. Hay una actividad mucho mejor para la que debería utilizarse al equipo de aseguramiento de la calidad; una actividad que utiliza su creatividad e imaginación humanas, pero hablaremos de ello más adelante.



Los clientes y los usuarios esperan que cada nuevo lanzamiento se pruebe de manera exhaustiva. Nadie espera que el equipo de desarrollo se salte las pruebas solo porque se ha quedado sin tiempo o sin dinero, así que cada prueba que puede automatizarse debe automatizarse. Las pruebas manuales deberían limitarse solo a aquellos aspectos que no pueden validarse de manera automática y a la disciplina creativa de las pruebas exploratorias.[5]

Las prácticas ágiles de desarrollo guiado por pruebas, integración continua y pruebas de validación respaldan esta expectativa.

## Nos cubrimos unos a otros

Como director de tecnología, espero que los equipos de desarrollo se comporten como equipos. ¿Cómo se comportan los equipos? Imagínese un equipo de jugadores moviendo el balón por el campo. Uno de los jugadores tropieza y se cae. ¿Qué hacen los demás jugadores? Tapan el hueco dejado por el miembro del equipo que se ha caído y siguen moviendo el balón por el campo.

A bordo de un barco, todo el mundo tiene una tarea. Todos saben, además, cómo hacer el trabajo de otro, porque a bordo de un barco deben realizarse todas las tareas.

En un equipo de software, si Bob se pone enfermo, Jill se ocupa de acabar las tareas de Bob. Eso significa que más vale que Jill sepa en qué estaba trabajando Bob y dónde guarda los archivos fuente, los *scripts*, etc.

Espero que los miembros de los equipos de software se cubran entre sí. Espero que cada miembro individual de un equipo de software se asegure de que hay alguien que pude cubrirle si él no puede ocuparse de algo. Es responsabilidad suya asegurarse de que uno o más miembros de su equipo pueden cubrirle.

Si Bob es el tío de la base de datos y se pone enfermo, no espero que el progreso del proyecto se pare en seco. Otra persona, aunque no sea "el tío de la base de datos", debería cubrir el puesto. No espero que el equipo mantenga los conocimientos en silos; espero que el conocimiento se comparta. Si tengo que reasignar a la mitad de los miembros del equipo a un proyecto nuevo, no espero que la mitad del conocimiento se vaya del equipo con ellos.

Las prácticas ágiles de programación en pareja, equipo completo y propiedad colectiva apoyan estas expectativas.

---

5. Agile Alliance. "Exploratory testing". Disponible en `https://www.agilealliance.org/glossary/exploratory-testing`.

su trabajo antes de que el sistema pueda implantarse. Los directores y los interesados impacientes presionan al equipo para que acabe y pueda implantarse el sistema.

Cuando el aseguramiento de la calidad se hace al final, todos los retrasos acumulados recaen sobre ese equipo. Si los desarrolladores entregan el trabajo tarde al equipo de aseguramiento de la calidad, ¿cambia la fecha de entrega? A menudo, esa fecha se ha escogido por razones comerciales muy buenas y retrasarla podría tener consecuencias negativas, incluso catastróficas. El equipo de aseguramiento de la calidad tiene que cargar con el muerto.

¿Cómo debería probar el sistema el equipo de aseguramiento de la calidad si no queda tiempo en el calendario para probarlo? ¿Cómo puede ir más deprisa el equipo? Muy sencillo: sin probarlo todo. Pruebe solo las cosas que han cambiado. Haga un análisis del impacto basado en características nuevas y modificadas y pruebe solo las cosas a las que afectan. No pierda el tiempo probando cosas que no se han cambiado.

Pierda esas pruebas. Bajo presión, el equipo de aseguramiento de la calidad ignora todas las pruebas de regresión, sin más. Esperan poder ejecutarlas la próxima vez. A menudo, "la próxima vez" no llega nunca.

### La enfermedad del aseguramiento de la calidad

Sin embargo, ese no es el peor problema que surge cuando el equipo de aseguramiento de la calidad está al final del proceso. Cuando está al final, ¿cómo sabe la organización que está haciendo bien su trabajo? Por el recuento de defectos, por supuesto. Si el equipo de aseguramiento de la calidad está encontrando muchos defectos, está claro que están haciendo bien su trabajo. Los directores de aseguramiento de la calidad pueden utilizar el número de defectos encontrados como una prueba clara de que están desempeñando bien sus funciones.

Así, los defectos se consideran algo bueno.

¿Quién más puede beneficiarse de los defectos? Existe un antiguo dicho entre los programadores más veteranos: "Puedo cumplir el plazo que me pongan, siempre que no haga falta que el software funcione bien". Entonces, ¿quién más se beneficia de los defectos? Los desarrolladores que necesitan cumplir plazos.

No hace falta decir nada. No es necesario acordar nada por escrito. Pero ambas partes entienden que se benefician de los defectos. Surge una economía de mercado negro de defectos. Esta enfermedad se extiende a muchas organizaciones y, aunque no es terminal, está claro que es debilitante.



### Los desarrolladores son los probadores

Todos estos problemas se resuelven con la práctica de las pruebas de validación. El equipo de aseguramiento de la calidad escribe las pruebas de validación para las historias en una iteración, pero no ejecuta esas pruebas. No es tarea del equipo de aseguramiento de la calidad verificar que el sistema pasa las pruebas. ¿De quién es esa tarea? ¡De los programadores, por supuesto!

Es trabajo de los programadores ejecutar las pruebas. Es trabajo de los programadores asegurarse de que su código pasa las pruebas y, por supuesto, son los programadores los que deben ejecutar esas pruebas, que son la única forma que tienen de determinar si sus historias están acabadas.

### Compilación continua

Lo cierto es que los programadores automatizarán ese proceso[7] configurando un servidor de compilación continua. Este servidor simplemente ejecutará todas las pruebas del sistema, incluidas todas las pruebas unitarias y todas las pruebas de validación, cada vez que un programador libera un módulo. Veremos esto con más detalle cuando hablemos sobre la integración continua.

## EQUIPO COMPLETO

La práctica del equipo completo se llamaba al principio "cliente *in situ*" (*On-Site Customer*). La idea era que, cuanto menor fuese la distancia entre los usuarios y los programadores, mejor sería la comunicación y más rápido y exacto sería el desarrollo. El "cliente" era una metáfora para una persona o un grupo que entendía las necesidades de los usuarios y que se ubicaba con el equipo de desarrollo. Lo ideal era que el cliente estuviese en la misma habitación que el equipo.

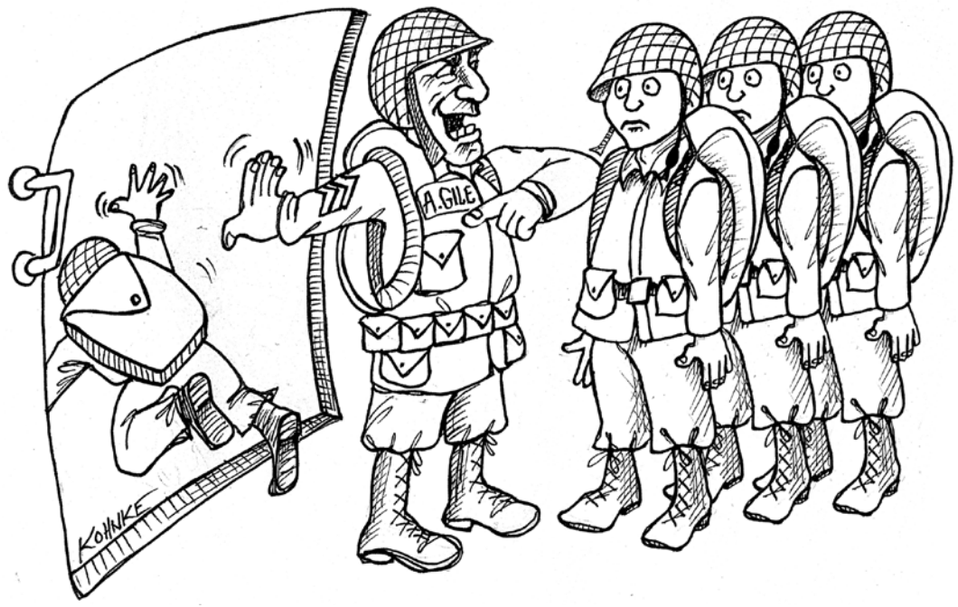
En Scrum, el cliente se llama "propietario del producto" (*Product Owner*). Esta es la persona (o el grupo) que elige las historias, establece las prioridades y proporciona *feedback* inmediato.

Se cambió el nombre de la práctica por "equipo completo" para dejar claro que un equipo de desarrollo no es simplemente una díada cliente/programador, sino que está formado por muchos papeles, incluidos directores, probadores, redactores técnicos, etc. El objetivo de la práctica es minimizar la distancia física entre estos papeles. Lo ideal es que todos los miembros del equipo se sienten juntos en la misma habitación.

7. ¡Porque automatizar cosas es lo que hacen los programadores!

# 6

# Pasarse al desarrollo ágil




Cuando oí hablar de la programación XP por primera vez, pensé: "¿Puede haber algo más fácil que esto? Solo hay que seguir unas pocas disciplinas y prácticas sencillas. Nada del otro mundo".

Sin embargo, a juzgar por la cantidad de organizaciones que intentan, sin conseguirlo, pasarse al desarrollo ágil, parece que es muy, muy difícil adoptar la metodología ágil. Quizá la razón de todo ese fracaso sea que muchas organizaciones creen que el desarrollo ágil es algo que no es en realidad.

## Valores del desarrollo ágil

Kent Beck estableció los cuatro valores del desarrollo ágil hace mucho tiempo. Son el valor, la comunicación, el *feedback* y la simplicidad.

### Valor

El primer valor es el coraje o, dicho de otro modo, un grado razonable de aceptación del riesgo. Los miembros de un equipo de desarrollo ágil no están tan centrados en la seguridad política como para sacrificar la calidad y la oportunidad. Se dan cuenta de que la mejor manera de gestionar un proyecto de software a largo plazo es mostrar cierto grado de agresión.

Hay una diferencia entre valor y temeridad. Hace falta valor para implantar un conjunto de características mínimo. También hace falta valor para mantener código de alta calidad y mantener disciplinas de alta calidad. Sin embargo, es una temeridad implantar un código en el que no tenemos mucha confianza o que tiene un diseño insostenible. Es una temeridad sacrificar la calidad para ceñirse a un calendario.

La creencia de que la calidad y la disciplina incrementan la velocidad es una creencia valiente, porque se verá desafiada todo el tiempo por personas poderosas, pero ingenuas, que tienen prisa.

### Comunicación

Valoramos la comunicación directa y frecuente que cruza canales. Los miembros de equipos de desarrollo ágil quieren hablar unos con otros. Los programadores, los clientes, los probadores y los directores quieren sentarse juntos e interactuar con frecuencia y no solo en el contexto de las reuniones. No solo mediante emails, chats y circulares. Valoran la conversación cara a cara, informal e interpersonal.

No hay ninguna razón para que las herramientas de gestión de ciclos de vida ágiles no puedan ser geniales algún día, pero, si solo necesita gestionar una pared con tarjetas y debe utilizar software, adopte una herramienta para fines generales, como Trello.[8] Es fácil, inmediata, barata, extensible y no le revolverá el estómago.

Nuestra manera de trabajar cambia constantemente. Hemos pasado de SCCS a RCS a CVS a Subversion a Git a lo largo de los años, experimentando un cambio radical respecto a la forma en que gestionábamos el código fuente. Hemos tenido una evolución similar en las herramientas de pruebas, las de implantación y similares (que no recogemos aquí). Es probable que veamos una progresión parecida en las herramientas de gestión de ciclos de vida ágiles automatizadas.

Teniendo en cuenta el estado actual de la mayoría de las herramientas de gestión de ciclos de vida ágiles, puede que sea más seguro e inteligente empezar con herramientas físicas. Más adelante, puede plantearse utilizar una herramienta de gestión de ciclos de vida ágiles. Asegúrese de que sea fácil de aprender, transparente en su uso diario, fácil de adaptar y asequible para adquirirla y ejecutarla. Lo que es más importante: asegúrese de que es compatible con la manera de trabajar de su equipo y resulta rentable respecto a su inversión.

## *Coaching*: una perspectiva alternativa

Por Damon Poole, 14 de mayo, 2019*

*Damon Poole es un amigo que no está de acuerdo conmigo sobre muchas cosas, entre ellas el coaching ágil, así que he pensado que sería buena idea que ofreciese a los lectores un punto de vista diferente.*

*— UB*

### Los múltiples caminos al desarrollo ágil

Hay muchos caminos hacia el desarrollo ágil y, de hecho, muchos de nosotros los hemos recorrido sin querer. Podría decirse que el Manifiesto Ágil fue el resultado de que los autores se diesen cuenta de que todos ellos estaban haciendo un viaje similar y decidiesen describirlo por si había otras personas que quisiesen unirse a ellos. Mi camino hacia el desarrollo ágil empezó cuando entré en una tienda de electrodomésticos en 1977 que resultó que vendía el TRS-80. Cuando era un absoluto novato, ayudé a un programador experimentado a depurar un juego de Star Trek simplemente haciendo preguntas. Hoy en día lo llamamos "programación en pareja". Da la casualidad de que hacer preguntas es una parte muy importante del *coaching*.

8. De nuevo, hablamos de 2019. El paisaje cambia.

* Utilizado con permiso.



Desde entonces hasta algún momento alrededor de 2001, practiqué el desarrollo ágil de forma accidental. Solo creaba código en equipos multifuncionales pequeños, sobre todo con un cliente interno, centrados en lo que ahora se denomina historias de usuarios, y solo producíamos entregas pequeñas y frecuentes. Pero entonces, en AccuRev, nuestras entregas principales empezaron a espaciarse cada vez más, llegando a 18 meses en 2005. Durante 4 años, estuve practicando desarrollo en cascada de manera accidental. Era horrible y no sabía por qué. Además, se me consideraba un "experto del proceso". Detalles aparte, esta historia resultará familiar a mucha gente.

### De experto en procesos a experto en desarrollo ágil

Mi primer contacto con el desarrollo ágil fue doloroso. En 2005, antes de que la popularidad de la Alianza Ágil y otras se disparara, se celebraron las conferencias de la revista *Software Development*. En la bienvenida a los oradores de Software Development East, tras una charla sobre la gestión de prácticas para desarrollo distribuido en la que no se utilizó en absoluto la palabra "ágil", me encontré rodeado de líderes de pensamiento de la industria del software, como Bob Martin, Joshua Kerievsky, Mike Cohn y Scott Ambler. Parecía que los únicos temas por los que sentían pasión implicaban cosas como tarjetas de 3×5, historias de usuario, desarrollo guiado por pruebas y programación en pareja. Me quedé horrorizado por el hecho de que estos líderes de pensamiento se hubiesen tragado lo que a mí me parecía un cuento chino.

Unos meses después, mientras investigaba el desarrollo ágil para desacreditarlo de manera adecuada, tuve una epifanía. Como programador y dueño de un negocio, aquella nueva perspectiva venía del entendimiento del desarrollo ágil como un algoritmo para encontrar las características que producían valores más elevados en el mercado y, después, convertirlas en ingresos más rápido.

Tras aquel momento de inspiración, desarrollé una gran pasión por compartir el desarrollo ágil con todo el mundo. Organicé webinarios, escribí publicaciones en *blogs*, di conferencias, me uní y después dirigí un encuentro Agile New England en la zona de Boston e hice todo lo que pude por correr la voz. Cuando la gente compartía sus dificultades al implementar el desarrollo ágil, estaba deseando ayudar. Me ponía en modo "solucionador de problemas" y explicaba lo que creía que deberían hacer.

# Epílogo

Por Eric Crichlow, 5 de abril, 2019


Recuerdo bien el primer trabajo que tuve en el que decidieron pasarse al desarrollo ágil. Fue en el año 2008, en una empresa que había sido adquirida por una gran corporación. Estaba experimentando cambios significativos en sus políticas, sus procedimientos y su plantilla. También recuerdo otro par de trabajos en los que se ponía énfasis en las prácticas ágiles. Se seguían los rituales religiosamente: planificación de *sprints*, demostración, revisión de *sprints*... En uno de ellos, todos los desarrolladores de la empresa tuvieron que hacer una formación de dos días en desarrollo ágil con un *Scrum Master* certificado. Como desarrollador de aplicaciones móviles, me mandaron escribir una aplicación de móvil para jugar al póker ágil.

Pero, en los 11 años que han pasado desde mi primer contacto con el desarrollo ágil, también he estado en muchas empresas de las que no puedo recordar si utilizaban prácticas ágiles. Quizá sea porque el desarrollo ágil se ha extendido tanto que es fácil darlo por sentado sin ni siquiera pensarlo. O tal vez es porque todavía hay una cantidad importante de empresas que no lo han adoptado.

Cuando oí hablar del desarrollo ágil por primera vez, no me entusiasmó demasiado. El método en cascada tenía sus defectos, pero yo estaba en una empresa que no dedicaba un tiempo significativo a crear documentos de diseño. Mi vida como desarrollador consistía, por lo general, en que me diesen verbalmente un conjunto de características que se esperaban para la siguiente entrega y la fecha en la que debía tener lista esa entrega y que me dejasen vía libre para que hiciese mi magia. Aunque está claro que esto podía conducir a la temida "marcha de la muerte", también me daba la libertad para estructurar mis actividades como yo quisiera. Me evitaba el escrutinio y la rendición de cuentas frecuentes de una reunión de pie diaria en la que tendría que explicar en qué había trabajado el día anterior y en qué iba a estar trabajando ese día. Si decidía pasar una semana reinventando la rueda, era libre de hacerlo sin que nadie juzgase mi decisión porque nadie tenía ni idea de lo que estaba haciendo.

Un antiguo director de desarrollo para el que trabajé solía llamarnos "pistoleros de código". Simplemente, nos encantaba acribillar a nuestros teclados en el salvaje oeste del desarrollo de software. Ese hombre tenía razón y, hasta cierto punto, las prácticas ágiles representaban al nuevo *sheriff* de la ciudad que quería controlar nuestras tendencias inconformistas.

Al desarrolló ágil no le resultó nada fácil conquistarme.

Sería presuntuoso creer que el desarrollo ágil es el *estándar de facto* en las empresas de desarrollo de software o que todos los desarrolladores lo aceptan como algo positivo. A la inversa, sería una ingenuidad negar la relevancia de la metodología ágil en el mundo del desarrollo de software. Pero ¿qué significa eso? ¿Cuál es exactamente su relevancia?

# Desarrollo ágil esencial Vuelta a las raíces

**Principios y valores del desarrollo ágil para una nueva generación**

Casi veinte años después de la presentación del Manifiesto Ágil, el legendario Robert C. Martin "Uncle Bob" vuelve a presentar los principios y valores del desarrollo ágil a una nueva generación, tanto de programadores como de no programadores. Martin, autor de *Código limpio* y de otras guías sobre desarrollo de software de gran influencia, estaba presente cuando se creó el desarrollo ágil. En este libro elimina malos entendidos y distracciones que, a lo largo de los años, han hecho que utilizar el desarrollo ágil sea más difícil de lo que se planeó en principio.

Martin describe qué es el desarrollo ágil sin ambigüedades: una disciplina pequeña que ayuda a equipos pequeños a gestionar proyectos pequeños… con implicaciones enormes, porque todo proyecto grande está formado por muchos proyectos pequeños. Sirviéndose de sus cincuenta años de experiencia con proyectos de todos los tipos imaginables, muestra cómo el desarrollo ágil puede ayudarnos a llevar la verdadera profesionalidad al desarrollo de software.

- Vuelva a las raíces: qué es el desarrollo ágil, qué era y qué debería ser siempre.
- Entienda los orígenes y la práctica adecuada de SCRUM.
- Domine las prácticas ágiles esenciales orientadas a los negocios, desde las entregas pequeñas y las pruebas de validación a la comunicación entre el equipo completo.
- Explore las relaciones de los miembros de un equipo de desarrollo ágil entre sí y con su producto.
- Redescubra prácticas técnicas ágiles indispensables: desarrollo guiado por pruebas, refactorización, diseño simple y programación en pareja.
- Entienda los papeles centrales que juegan los valores y la artesanía en el éxito del equipo de desarrollo ágil.

Si quiere aprovechar los auténticos beneficios del desarrollo ágil, no hay atajos: tiene que practicar el desarrollo ágil bien. *Desarrollo ágil esencial. Vuelta a las raíces* le mostrará cómo, no importa si es desarrollador, probador, director de proyecto o cliente.

**Robert C. Martin** "Uncle Bob" es programador desde 1970. Es cofundador de la empresa de formación en línea *cleancoders.com* y fundador de Uncle Bob Consulting LLC, que ofrece servicios de asesoría en software, formación y desarrollo por todo el mundo. Ha sido "maestro artesano" en 8th Light, Inc.; una consultoría de software con base en Chicago, redactor jefe de la revista C++ Report y primer presidente de la Alianza Ágil. Sus libros incluyen *Código limpio, El limpiador de código* y *Arquitectura limpia*.

ISBN 978-84-415-4227-3
2315154
9 788441 542273

ANAYA MULTIMEDIA
www.anayamultimedia.es

Imagen de cubierta: Peresanz/iStock