# ACERCA DEL AUTOR

**ALEJANDRO JUAN CANOSA FERREIRO**

Ingeniero técnico en informática de gestión por la Universidad Nacional de Educación a Distancia. Postgrado en Software Quality Assurance por la Universidad Politécnica de Cataluña. Certificado en Scrum Foundation Professional Certification (SFPC), DevOps Essentials Professional Certification (DEPC), Expert Level Certification en Katalon y a punto de certificarse en ISTQB , Professional Scrum Master e ITIL.

Consultor de QA con más de 10 años de experiencia en el mundo de calidad de software entre España y Colombia y experto en automatización de pruebas de software con herramientas como Selenium, TestComplete, Katalon o Rational Functional Tester de IBM.

# 1

# METODOLOGÍAS DE DESARROLLO ÁGILES

## 1.1 SCRUM. CONCEPTOS

**Scrum** es una metodología de desarrollo ágil que utiliza de manera regular buenas prácticas para trabajar en equipo y conseguir el mejor resultado en un proyecto.

En **Scrum** se realizan entregas parciales y regulares de un producto y que son priorizadas por el cliente.

### 1.1.1 Iteraciones. ¿Qué son?

Es **Scrum,** se hacen entregas de pequeños incrementos funcionales cada dos semanas de manera general pueden ser 3 o 4 semanas, estos ciclos temporales son de duración

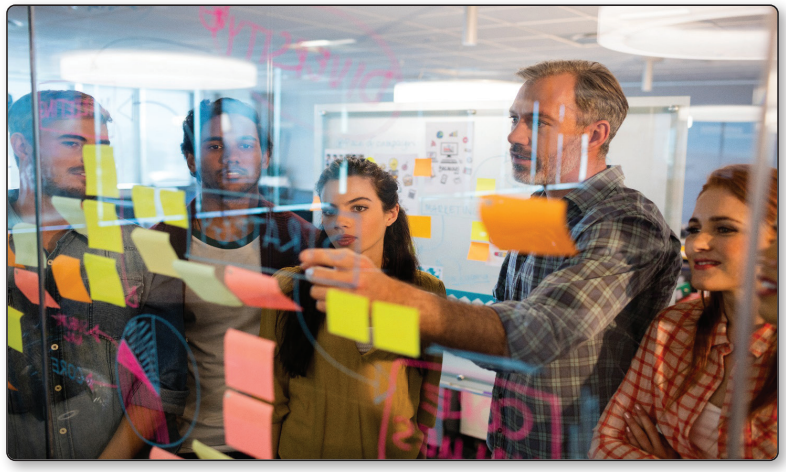
fija, es decir, cuando se empieza un proyecto y se definen estas iteraciones si se llega al acuerdo de 2 semanas suele mantenerse hasta el final ese tiempo, estas iteraciones como supondrás se denomina **Sprints.**

Algo que se debe cumplir siempre es que estas iteraciones o **Sprints** deben proporcionar un resultado completo, es decir, un incremento funcional del producto.

### 1.1.2 Priorización de tareas

El cliente que puede ser el **Producto Owner** (propietario del producto) es el que prioriza todas las tareas que hay en el proyecto comparando su coste y el valor que aporta y prioriza aquellas que tienen mayor valor a menor coste.

### 1.1.3 Reuniones del Sprint



Al comenzar el Sprint se realiza una reunión que se llama **reunión de planificación de Sprint** que se divide en 2 partes:

- Primera parte. Selección de requisitos

  Dura dos horas y el cliente indica cuales son los requisitos más importantes y su prioridad y el equipo de **SCRUM** selecciona aquellos requisitos más importantes y se compromete a entregarlos en ese **Sprint** y además realiza las preguntas oportunas al cliente.

- Segunda parte. Planificación del Sprint

  Dura dos horas y el equipo de **SCRUM** identifica las tareas que se necesitan realizar para completar los requerimientos del **Sprint** y se estima el esfuerzo conjunto y se van autoasignando las tareas del **Sprint** y los problemas complejos se puede resolver en grupo.

Durante el Sprint se realizan reuniones diarias de unos 15 minutos que se llaman **dailys** en donde se responde a 3 preguntas:

1. ¿Qué he hecho desde la última reunión de ayer?
2. ¿Qué voy a hacer hoy?
3. ¿Qué impedimento tengo o voy a tener para realizar mis tareas?

Durante el sprint puede ser que el cliente refine la lista de requerimientos o cambie los objetivos del **Sprint.**

Cuando se termina el Sprint el último día se realiza la **revisión del Sprint** que tiene 2 partes:

- Primera parte. Revisión

  Suele tener una duración de 15 horas y consiste en que el equipo de **SCRUM** presenta al cliente los requerimientos completados del Sprint de una manera que se vea como un incremento del producto y en función del resultado el cliente puede replanificar el proyecto.

- Segunda parte. Retrospectiva

  En esta parte de la reunión el equipo mira los problemas que hubo en el **Sprint** y entre todos buscan maneras de mejorar el proceso de desarrollo y se hacen compromisos que se deben cumplir y se analiza cómo ha ido el sprint en tiempo de tareas con respecto al **Sprint** anterior.

## 1.2 KANBAN

### 1.2.1 ¿Qué es Kanban?

La metodología **Kanban** se basa en la filosofía centrada en la mejora continua, donde las tareas se extraen de una lista de acciones en un flujo de trabajo continuo. La metodología se implementa mediante tableros **Kanban**.

### 1.2.2 Tableros Kanban

Los tableros **Kanban** es una forma de gestión de proyectos que permite a los equipos de trabajo visualizar los flujos de trabajo.

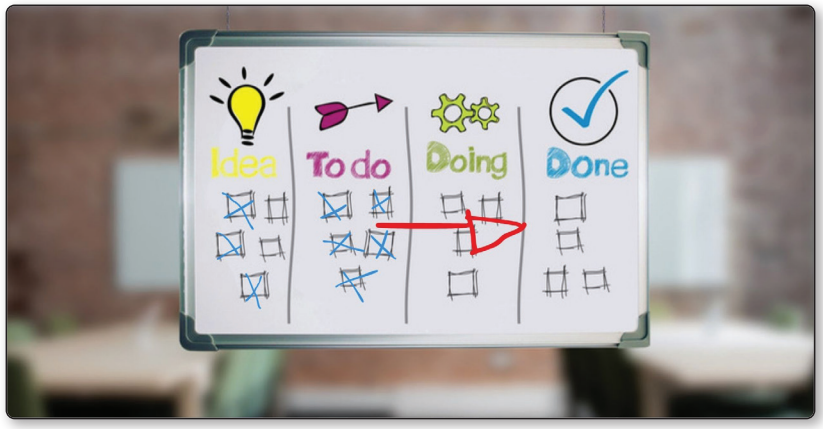
En los tableros **Kanban** el trabajo de un proyecto se ve organizado por columnas donde cada columna representa una etapa del trabajo.


El **tablero Kanban** de trabajo suele tener las columnas:

- *Trabajo pendiente.*
- *En progreso.*
- *Terminado.*

Las tareas están representadas por tarjetas, avanzando a través de las distintas columnas hasta llegar a la columna terminada que es cuando se ha terminado la tarea.

En la imagen de abajo se puede ver cómo en la reunión se sacan una serie de ideas o requerimientos que se implementan mediante tareas y todo se ponen en **Todo** que sería la columna por hacer, después se va pasando a la columna de realizando esa tarea. **Doing** y por último cuando se termina la tarea se pasa a la columna de **Done.**



### 1.2.3 Historia de Kanban

**Kanban** fue creado por **Taichi Ohmo**, ingeniero japonés de Toyota a finales de 1940, lo que hizo esta persona fue crear productos en función de la demanda del consumidor en tiempo real, es lo que se llama **producción justo a tiempo (just in time) .**

Esto soluciono el problema que había, que se fabricaban productos en este caso coches en función de la demanda anticipada, es decir, suponiendo una demanda pero esto hacía que quedaran muchos coches en stock sin vender.

El marco **Kanban** transformó el proceso de fabricación de Toyota, de un proceso de empuje (los productos se introducen en el mercado) a un proceso de extracción (los productos se crean según la demanda del mercado), esto hizo que Toyota tuviera un nivel de inventario más bajo sin que afectara a su competitividad.

Después de ver un poco de historia me gustaría que supierais cual es el significado de **Kanban; Kanban** realmente significa en Japones, ”kan” que significa señal visual y “ban” que significa tablero.

En Toyota cada tarjeta **Kanban** eran tarjetas de papel que representaban que se necesitaba un producto nuevo o una pieza nueva.

La metodología **Kanban** se adoptó para el desarrollo de software a principios de la década del 2000.

Al final podríamos hablar de **Kanban** como un método visual de gestión de proyectos que permite encontrar un equilibrio entre la demanda de trabajo y la disponibilidad de recursos del equipo.

Esta metodología es ágil como lo son **XP** y **SCRUM.**

### 1.2.4 Principios de Kanban

**Kanban** tiene 4 principios que son los siguientes:

1. **Empieza a trabajar con las tareas que existen.**

   **Kanban** es flexible y se puede implementar en cualquier flujo de trabajo.

2. **Compromiso a buscar e implementar cambios progresivos y evolutivos.**

   Si haces cambios muy grandes es posible que tu sistema no funcione correctamente por eso **Kanban** fomenta la mejora continua y el cambio progresivo, es decir, pequeños cambios que son integrados en el sistema general, es decir, no se cambió todo de una vez, si no que se va cambiando poco a poco.

3. **Respetar los procesos, roles y las responsabilidades actuales.**

   **Kaban** no tiene roles integrados al contrario que **XP** y **SCRUM** por lo que puede actuar con los procesos y estructuras que haya en un equipo manteniendo procesos que existan en un equipo y que pueden ser excelentes.

4. **Impulsar el liderazgo a todos los niveles.**

   **Kanban** reconoce que el cambio puede venir de cualquier dirección no solo de la parte directiva, por eso alienta a los miembros del equipo a participar y proponer mejoras.

## 1.2.5 Principales práctica s de Kanban

**Kanban** no es tan estricto como **SCRUM** pero también tiene una serie de práctica s que se deberían seguir y son las siguientes.

### Visualizar el trabajo

En **Kanban** es fácil hacer un seguimiento a las tareas del equipo, ya que empieza en la primera columna la tarea y se va pasando por las siguientes columnas hasta llegar a la columna de finalizando lo que hace que se pueda ver visualmente el trabajo que se está realizando en cada momento.

### Limitar el trabajo en grupo

**Kanban** requiere ser ágil y por lo tanto una tarea no puede estar mucho tiempo en una columna, tampoco pueden estar más de un número determinado de tareas en una columna para ayudar a centrarse en las tareas y no estar con varias tareas y no terminar ninguna, el objetivo es que cada miembro del equipo termine 1 o 2 tareas al día y ser muy productivos.

### Gestionar el flujo de trabajo

Se tiene que limitar el número de tareas que estén en una columna a la vez para ser ágiles y hacer una entrega continua, marcar directrices para saber cuándo cancelar subtareas o ponerlas en espera si ocurre algún tipo de impedimento o problema.

### Implementar políticas de procesos

Se deben implementar políticas de procesos para mejorar la rapidez de las tareas entre las distintas etapas como por ejemplo cuantas tareas pueden estar en una etapa determina, cuando una subtarea se tiene que cancelar o poner en espera o lo mismo parea una tarea.

### Implementar ciclos de comentarios

En **Kanban** se recopilan comentarios de los 2 grupos fundamentales, clientes y equipo.

Los comentarios del cliente son muy importantes porque nos permiten analizar si tubo algún problema el producto o si no están contentos con el rendimiento, cualquier

cosa que no le guste puede comentarlo y tenemos un **feedback** muy rápido para mejorar continuamente no solo el producto si no el proceso de desarrollo o cualquier actividad porque siempre es mejorable.

Es importante los comentarios del equipo para saber si están motivados, cansados, insatisfechos; el rendimiento de un equipo puede bajar si se sienten cansados, explotados laboralmente o incluso mal pagados.

De este tema hablare en la última sección de algo muy importante y son las malas praxis que muchas empresas utilizan actualmente y que hace que cada vez menos jóvenes quieran dedicarse a este mundo.

### Mejorar colaborando y experimentando

**Kanban** es mejora continua por lo que tienes que estar dispuesto a combinar **Kanban** con **SCRUM** o a agregar nuevas etapas al tablero en función de tu proyecto, nuevos estándares, nuevas tecnologías.

Mucho se habla de la **inteligencia artificial** y de **ChatGPT** como si fuera a quitar el trabajo a los desarrolladores pero es más una ayuda que un competidor, puede ayudar a mejorar por ejemplo en la **automatización de pruebas** puede ayudar a ver un **mapa caliente** de un sitio y ver cuáles son las acciones más realizadas por un visitante o usuario y convertir eso en scripts para que se pruebe aquello más importante para el negocio real que sería aquello que más utiliza el usuario de una aplicación web o app.

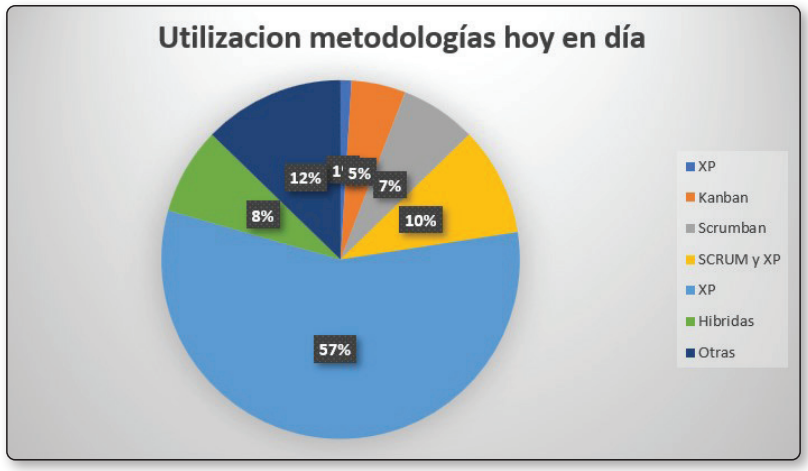
## 1.3 XP

### 1.3.1 Historia de XP

**XP** significa Extreme Programming y es una metodología ágil de desarrollo que fue creada en 1995.

### 1.3.2 Utilización de metodologías hoy en día

Hoy en día muchas empresas utilizan **Scrum** o una combinación de **Scrum** y **XP**, pocas utilizan **Scrum** con **Kanban** o **XP** solamente, según estudios de diferentes páginas web tendríamos una gráfica con la utilización de las distintas metodologías en distintos proyectos como la de abajo, **Scrumban** sería la mezcla de **SCRUM** con **Kanban** e híbrida sería una mezcla de **SCRUM, Kanban y XP,** por ejemplo.




### 1.3.3 ¿Qué es XP?

Es una metodología cuyo objetivo es crear sistemas de alta calidad basados en una estrecha interacción con los clientes, pruebas constantes y ciclos de desarrollo cortos.

### 1.3.4 ¿Cómo funciona XP?

La metodología **XP** es una metodología ágil como **Kanban** o **SCRUM.**

Su objetivo es realizar ciclos de entrega rápidos, continuos e incrementables para conseguir el software que quiere el cliente.

**XP** utiliza una serie de prácticas y fases predefinidas para que el proceso sea efectivo por eso los ciclos increméntales suelen ser semanales, con reuniones constantes entre el equipo de desarrollo y el cliente.

Las prácticas más habituales en XP serían las siguientes:

- **Juego de planificación**

  Esto es una reunión que se realiza el primer día de la semana y que se llama juego de planificación donde el cliente indica las funcionalidades prioritarias y se indica el alcance que tiene que ser flexible y negociable.

- **Cliente siempre disponible**

  En **XP** el cliente siempre tiene que estar disponible para responder a preguntas, priorizar el alcance y realizar cambios si fuera necesario.

- **Entregas cortas**

  Al final de la semana el cliente recibe pequeñas versiones del producto para que el cliente pueda probarlo y sugerir mejoras.

- **Metáfora**

  Es la estrategia utilizada por el ámbito tecnológico para facilitar la comunicación con el cliente, esto mejora las expectativas y se consigue ahorrar tiempo.

- **Código sencillo**

  Hay que crear código sencillo, eficiente y que consiga el resultado esperado por el cliente.

- **Pruebas unitarias**

  Hay que realizar pruebas unitarias para asegurarnos que el código es correcto y devuelve lo que se espera.

- **Ritmo sostenible**

  El ritmo debe ser de unas 40 horas semanales para que sea un ritmo saludable y mantenible.

- **Propiedad colectiva**

  Los proyectos desarrollados deben ser accesibles, transparentes y documentados para todo el equipo.

- **Programación por parejas**

  La programación del código se hace por parejas para una revisión constante y para que el conocimiento de las reglas técnicas y del negocio sea común y así conseguir una igualación del nivel.

- **Estandarización del código**

  Se debe estandarizar el código para que todo el mundo siga las mismas reglas y no que cada persona genere su estructura de código.

- **TDD**

  **TDD** significa Test Driven Development que significa desarrollo dirigido por pruebas lo que significa que no se hace primero el código y luego las pruebas, si

no que se crean primero las pruebas y luego se crea el código para que pasen esas pruebas; se hablara de esto en un tema posterior.

- **Refactorización**

  Es un proceso que permite la mejora continua del proyecto haciendo más refinado el código para que se pueda reutilizar, esto se consigue agrupando el código por funcionalidad para reutilizarlo y para generar menos errores y tener menos duplicidad de código.

  Básicamente agrupamos el código por métodos y luego esos métodos forman parte de una funcionalidad.

- **Integración continua**

  Al crear una nueva funcionalidad se debe integrar rápido a la versión actual del producto para que se pruebe y se encuentren los errores rápidamente por eso se suele automatizar todos los procesos y pruebas que se puedan, el proceso de compilación, el proceso de despliegue en un ambiente, las pruebas unitarias, las pruebas de integración, las pruebas funcionales, las pruebas de rendimiento, de seguridad, etc. Todo lo que se pueda automatizar permitirá construir un flujo de **integración continua** que ayuda y mucho en los tiempos de entrega.

  Se hablará de esto en temas posteriores.

- **Despliegue continuo**

  Se tiene que automatizar las pruebas, pero también como hablamos en la sección anterior el despliegue de los **releases** en los distintos ambientes que existan apara el proyecto.

### 1.3.5 Valores de XP

Existen una serie de valores que son intrínsecos a esta metodología y son los siguientes:

- **Comunicación**

  XP tiene una comunicación constante y continuada con el cliente y sobre todo que sea *cara a cara* para que haya un entendimiento total entre las partes.

- **Simplicidad**

  Hay que mantener el diseño y la funcionalidad lo más fácil de usar que sea posible, priorizar lo que es absolutamente necesario para el proyecto para reducir los costes y el tiempo.

- **Feedback**

  Los comentarios constantes y la retroalimentación son fundamentales para conseguir ajustes precisos y rápidos.

- **Coraje**

  Estar abierto al cambio, superar fracasos, proponer mejoras y saber decir no es fundamental en XP.

- **Respeto**

  El trabajo en equipo es fundamental en **XP** y por eso es necesario que los miembros se respeten, colaboren y tengan una buena relación.

### 1.3.6 Fases en XP

Las fases que hay en la metodología **XP** serian 5 y se pueden ver en la tabla de abajo.

| | |
|---|---|
| 1 | Planeación |
| 2 | Gestión |
| 3 | Diseño |
| 4 | Codificación |
| 5 | Testing |

#### Planeación

En la fase de planeación ocurren las siguientes actividades y tiene las siguientes características:

- Se escriben las **historias de usuario**.
- Se planifica el **release** o entregable.
- El **release** o entregable debe ser pequeño al ser un **sprint** de una semana.
- Se indican los sprints que habrá.
- Planificación del incremento en cada **sprint.**

#### Gestión

- En esta fase de gestión ocurren las siguientes actividades y tiene las siguientes características:
- Se crea espacio de trabajo abierto.
- Ritmo sostenible de las actividades.
- Se realizan reuniones diarias.
- Mediciones diarias de la velocidad de desarrollo.
- Mejoras continuadas del proceso y código.
- Flujo de gente continuada para recibir formación y **feedback.**

## Diseño

El código debe ser simple es decir debe tener unas características que serían:

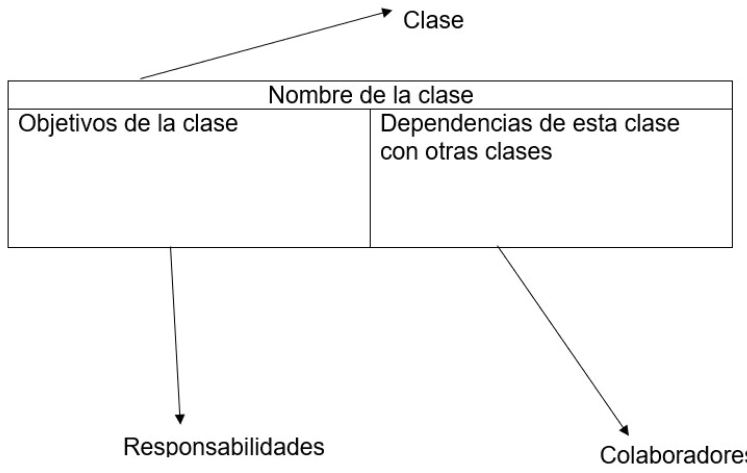
- Comprobable.
- Comprensible.
- Navegable.
- Explicable.

En ingles las siglas son **TUBE**

En el diseño del código se utilizan las tarjetas.

**CRC (clase-responsabilidad-colaboración)**

Abajo podemos ver cómo serían estas **tarjetas CRC.**




En el diseño hay algo importantísimo y es la **refactorización** para poder reutilizar el código y no duplicarlo ni perder tiempo creando código que ya está creado en otra parte.

## Codificando

Se debe codificar en pareja con unos estandartes establecidos de antemano y escribiendo las pruebas primero y luego creando el código e ir refinándolo a través de la refactorización.

Se hablará de esto más en detalle, pero por ahora lo que se debe saber es que la refactorización es mejorar el diseño del código sin cambiar su comportamiento.

### Testing

En esta fase se deben cumplir una serie de directrices que las pongo abajo.

- Siempre se tienen que realizar **pruebas unitarias** al código.
- Las pruebas unitarias son parte del código.
- Deben realizarse pruebas de regresión con cada cambio.
- Todo el código tiene que pasar todas las pruebas unitarias antes de poder ser desplegado en un ambiente pruebas.
- Las **pruebas de aceptación** se realizan en cada nuevo incremento funcional del producto para tener **feedback** del cliente y los comentarios deben ser públicos para que todo el equipo pueda estar al tanto.
- Cuando se encuentra un error se hacen pruebas alrededor de ese error para indagar y ver si hay más errores parecidos.

## 1.3.7 Roles de XP

En XP hay una serie de roles y en la tabla de abajo los voy a comentar, pero lo que hay que saber es que hay 5 roles.

| El cliente | El programador | El entrenador | El rastreador | El probador | El pronosticador |
|---|---|---|---|---|---|
| Es la persona encargada de crear **Requisitos para software** y establecer prioridades | Es la persona que escribe el código del producto | Es la persona que vigila el trabajo del equipo, lo controla y recomienda mejoras técnicas | Es el encargado de monitorear el progreso del desarrollo del software y detectar problemas | Es quien se encarga de probar el producto, hacer todo tipo de pruebas y crear las **historias de usuario** | Es la persona que rastrea los riesgos del proyecto y advierte al equipo |

## 1.4 ¿CUÁNDO UTILIZAR CADA UNA DE LAS METODOLOGÍAS?

Puedes usar las metodologías en cascada si:

- *Si estás trabajando en un proyecto secuencial donde ninguna fase puede comenzar si la anterior no está completa.*
- *Quieres controlar el alcance.*
- *Quieres una planificación clara y eficaz.*
- *Quieres priorizar la planificación.*
- *Quieres conocer todo el ciclo de desarrollo antes de comenzar el proyecto.*
- *Se prioriza la funcionalidad frente a la entrega.*

Puedes usar las metodologías ágiles en cualquiera de las tres, si:

- Quieres un proceso iterativo.
- Deseas obtener resultados rápidamente.
- Tu equipo trabaja rápido.
- Se prioriza en tu proyecto la adaptación a la previsibilidad.
- Tus clientes quieren participar activamente.

Puedes utilizar **Kanban** cuando:

- *Cuando tu proyecto necesita un sistema de gestión visual.*
- *Cuando quieres conocer de manera rápida y visual el estado de un proyecto.*
- *Llevas adelante procesos y proyectos de manera continuada.*
- *La mayor parte de tu trabajo no se realiza en periodos cortos de tiempo.*

Puedes utilizar **SCRUM** cuando:

- *Cuando desarrollas software de manera continuada.*
- *Crees que tu equipo puede ser más productivo con reglas y procesos.*
- *Tienes mucho trabajo por horas o días.*
- *Se tiene que entregar en plazos cortos (1 semana, 2 semanas) de manera rápida y continuada.*
- *Existe un SCRUM máster.*

## 1.5 DIFERENCIAS ENTRE SCRUM Y KANBAN

Las diferencias que hay entre Kanban y Scrum se exponen en la siguiente tabla.

| SCRUM | KANBAN | Explicación |
|---|---|---|
| Tiene una serie de reglas que hay que cumplir | Se utiliza más para visualizar el trabajo | Muchos equipos implementan SCRUM en tableros KANBAN, pero lo que se está implementando es SCRUM realmente |
| Tiene un límite de tiempo | Es flexible, no tiene límite de tiempo | SCRUM suele tener ciclos de entrega de 1 o 2 semanas, en KANBAN no hay fecha de inicio ni fin es una entrega continua de tareas |
| Las columnas de los tableros son de estados | Las columnas pueden mostrar estados u otras características diferentes | En SCRUM el tablero se utiliza para hacer un seguimiento de las tareas y se van pasando entre cada, mientras que en KANBAN las columnas pueden ser de estados o de del trabajo que se ha realizado cada mes o de otras cosas |
| Tiene roles perfectamente definidos | No tiene roles perfectamente definidos | El SCRUM tenemos el Scrum Master, el Product Owner en Kanban no |

## 1.6 CÓMO COMBINAR SCRUM Y KANBAN

Lo ideal es utilizar los roles, reuniones, planificaciones y artefactos de SCRUM y en el tablero de cada Sprint crear un tablero Kanban con estados y columnas que puedan mostrar más información sobre el Sprint y Sprints anteriores para mejorar la organización y la productividad.

## 1.7 CÓMO SER ÁGIL SIN SCRUM

Puedes ser ágil, aunque no utilices ni **Kanban**, ni **Scrum** ni **XP** implementando las siguientes prácticas:

1. Crear proyectos pequeños.
2. Designar un encargado del proyecto.
3. Organizar reuniones periódicas al menos una vez por semana.
4. Programar revisiones periódicas, para revisar el trabajo semanal para evitar amontonar errores.

## 1.8 PROBLEMAS ACTUALES EN LOS EQUIPOS Y EMPRESAS ESPAÑOLAS

Estamos en un mundo globalizado donde la tecnología gobierna el mundo, las redes sociales nos informan y manipulan, ya pocos compran en tiendas la mayoría lo hace a través de tiendas online, casi todos tenemos un móvil al menos y buscamos lo que no sabemos en Google, hasta hacemos amigos y ligamos a través de apps como Tinder o Badoo.

En el tema del desarrollo de software casi todo la innovación viene de China, India, EEUU y Japón, España casi no innova tecnológicamente y las grandes consultoras españolas solo aplican con años de atraso lo que ya es algo habitual en esos 4 países punteros, exigen estudios superiores, postgrados, másteres, certificaciones, hablar inglés y experiencia y que además que acepten ganar 3 veces menos que en otros países como Alemania o Irlanda y se quejan que no hay suficientes ingeniero, ¿quién está dispuesto a estudiar casi 4 años de una ingeniería y luego un master y a tener que certificarse cada 3 años o 2 años de certificaciones por un sueldo ridículo si lo comparamos a otros países con ingenieros tan cualificados como en España?.

Otro problema que veo habitualmente es que no suelen ascender aquellas personas más cualificadas si no los que tiene skills más afines a los jefes, lo que de toda la vida se llamó el amigo del jefe, además no se respeta al ingeniero, en países de Latinoamérica el ingeniero de sistemas es admirado y respetado llamándolo don ingeniero mientras aquí solo es un número más para conseguir proyectos internacionales que cada vez se consiguen menos porque si no mimas al empleado, al empleado no le compensa trabajar

en un sector que quema tanto la mente y la salud de las personas, de hecho he visto ocurrir estos problemas:

- Empleados que al tener el mismo sueldo siempre no están motivados.
- Malas relaciones al haber compañeros que ganan o ascienden simplemente por caer simpático a ciertos jefes.
- Empleados aburridos que ya han llegado al máximo puesto que conseguirán.
- Gente joven que trabaja haciendo horas extra y con sueldos muy bajos.

Las consultoras españolas deben empezar a atraer a los mejores cerebros de España valorando su carrera, fomentando la creación de academias online donde puedan tener ingresos extra vendiendo su conocimiento mediante cursos o e-books, motivándoles a investigar con fondos de la empresa donde las innovaciones puedan ir a medias para verse acompañados, si no se innova y no se valora a los ingenieros que son los que moverán el mundo junto a los ingenieros de medioambiente las consultoras españolas terminaran teniendo que despedir y posiblemente sucumbir ante la innovación tecnológica.

No se puede conseguir proyectos en base a reducir los costes y aumentar las horas porque eso solo lleva a quemar a los ingenieros, pagarle poco, que no salga con calidad el proyecto y que cada vez haya menos proyectos para las consultoras españolas y menos ingenieros porque hay trabajos que se gana lo mismo sin tener que actualizarse cada dos años.

El modelo de fábrica de software como quien hace un edificio no funciona, un obrero se forma mucho menos y en mucho menos tiempo que un ingeniero, dejemos de pensar con la mentalidad de obrero y empecemos a pensar con la mentalidad de un empresario americano del siglo XXI, tenemos los mejores ingenieros de Europa, porque no pensar en España como la fábrica de software de Europa al igual que la India es la fábrica de software de Inglaterra o EEUU, pensemos en grande.