



Eusko Jaurlaritzaren  
Informatika Elkarte

Sociedad Informática  
del Gobierno Vasco



# **ARINbide-Adaptativo**

## **Anexo: Conceptos básicos**

Versión 1.0

21 de Enero de 2015



[ARINbide](#) by [EJIE](#) is licensed under a [Creative Commons Reconocimiento-NoComercial-CompartirIgual 3.0 Unported License](#).

Versión	Fecha	Resumen de cambios	Elaborado por:	Aprobado por:
1.0	21/01/2015	Primera versión		

## Contenido

<b>1</b>	<b>Agilismo, Scrum y Extreme Programming (XP)</b>	<b>4</b>
1.1	Los valores y principios del “agilismo”	4
1.2	Los valores y principios de Scrum	4
1.3	Los valores y principios de XP	5
<b>2</b>	<b>Técnicas</b>	<b>10</b>
2.1	Personas y escenarios	10
2.2	Sesiones de trabajo	10
2.3	El modelo de Kano	13
2.4	Poker de Prioridad (Priority Poker)	15
2.5	MoSCoW (Must, Should, Could, Won't)	16
2.6	Poker de planificación o de estimación (Planning Poker)	16
2.7	El tablero de tareas (Task Board)	17
2.8	Estimación de la velocidad del equipo	19
2.9	Desarrollo dirigido por Pruebas (TDD)	20
2.10	Desarrollo dirigido por Pruebas de Aceptación (ATDD)	20
2.11	Integración continua (Continuous Integration)	20
2.12	Spikes	21
<b>3</b>	<b>Bibliografía y referencias</b>	<b>22</b>

# 1 Agilismo, Scrum y Extreme Programming (XP)

## 1.1 Los valores y principios del “agilismo”

El Manifiesto Ágil establece cuatro valores para el desarrollo ágil.

- **Individuos e interacciones** sobre procesos y herramientas
- **Software funcionando** sobre documentación extensiva
- **Colaboración con el cliente** sobre negociación contractual
- **Respuesta ante el cambio** sobre seguir un plan

Esto es, aunque se valoran los elementos de la derecha, se valoran más los de la izquierda.

Y de estos cuatro valores, se establecen doce principios:

1. Nuestra mayor prioridad es satisfacer al cliente mediante la **entrega temprana y continua de software con valor**.
2. **Aceptamos que los requisitos cambien**, incluso en etapas tardías del desarrollo. Los procesos ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.
3. **Entregamos software funcional frecuentemente**, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.
4. Los responsables de negocio y los desarrolladores **trabajamos juntos de forma cotidiana** durante todo el proyecto.
5. Los proyectos se desarrollan en torno a **individuos motivados**. Hay que darles el entorno y el apoyo que necesitan, y **confiarles la ejecución del trabajo**.
6. El método más eficiente y efectivo de **comunicar** información al equipo de desarrollo y entre sus miembros es la **conversación cara a cara**.
7. El **software funcionando** es la medida principal de **progreso**.
8. Los procesos ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios debemos ser capaces de **mantener un ritmo constante** de forma indefinida.
9. La atención continua a la **excelencia técnica** y al **buen diseño** mejora la Agilidad.
10. La **simplicidad**, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.
11. Las mejores arquitecturas, requisitos y diseños emergen de **equipos auto-organizados**.
12. A intervalos regulares el equipo **reflexiona** sobre cómo ser más efectivo para la continuación ajustar y **perfeccionar** su comportamiento en consecuencia.

## 1.2 Los valores y principios de Scrum

Se pueden establecer para Scrum los siguientes valores básicos:

- **Compromiso**. Otorgar a las personas la confianza y autoridad que necesitan para cumplir con sus compromisos. Scrum no es posible sin una actitud activa y comprometida de todos los participantes en el proyecto.
- **Foco**. Es necesaria una dedicación plena de todos los participantes en el proyecto, no caben distracciones.
- **Apertura**. Scrum mantiene todo acerca del proyecto visible a todos. Todos participan.
- **Respeto**. Respetar las diferentes personas del equipo y sus formas de pensar. Sin respeto, no puede existir una comunicación efectiva.
- **Coraje**. Tener el coraje para comprometerse, actuar y ser honesto, con un objetivo común. Requiere coraje comunicar problemas, identificar impedimentos, pedir, recibir y dar ayuda.

Los principios son las pautas básicas que deben seguirse en el proyecto para garantizar la aplicación efectiva del marco Scrum:

- Control de procesos empírico o **empirismo**. Asegura que el conocimiento procede de la experiencia y de la toma de decisiones basadas en lo que se conoce. Se soporta sobre tres pilares:

- *Transparencia.* Los aspectos significativos del proceso deben ser visibles para aquellos que son responsables del resultado. Estos están definidos por un estándar común, y todos los observadores comparten un entendimiento común. Por ejemplo, la definición de “Terminado”.
- *Inspección.* Se deben inspeccionar frecuentemente los artefactos Scrum y el progreso hacia un objetivo para detectar variaciones.
- *Adaptación.* Si se detecta que uno o más aspectos de un proceso se desvían de límites aceptables, y que el producto resultante no será aceptable, el proceso o el material que está siendo procesado deben ser ajustados.
- **Auto-organización.** Un equipo entrega un valor significativamente mayor cuando es auto-organizado. Sus miembros demuestran un gran sentimiento de compromiso y responsabilidad; que a su vez produce un entorno innovador y creativo, más propicio para el crecimiento.
- **Colaboración.** Centrado en las tres dimensiones básicas relacionadas con el trabajo colaborativo: conciencia, articulación y apropiación. También aboga por la gestión de proyectos como un proceso de creación de valor compartido con los equipos de trabajo e interactuar conjuntamente para ofrecer el mayor valor.
- **Priorización basada en el valor.** Se ofrece el máximo valor para el negocio, desde el inicio del proyecto hasta su finalización.
- **Limitar el tiempo (Time-boxing).** El tiempo se considera una restricción limitante, de modo que todas las actividades de planificación y de ejecución del proyecto deben estar restringidos por un tiempo fijo previamente establecidos para cada uno de ellos.
- **Desarrollo iterativo.** Se define el desarrollo en ciclos sucesivos y se enfatiza cómo manejar mejor los cambios y crear productos que satisfagan las necesidades del cliente

### 1.3 Los valores y principios de XP

XP adopta 5 valores para guiar el desarrollo del software en un equipo:

- **Comunicación.** Existe una comunicación directa y continua entre los desarrolladores y entre clientes y desarrolladores. El cliente se integra en el equipo para establecer prioridades y resolver dudas, ve el avance día a día, y es posible ajustar la agenda y las funcionalidades de forma consecuente.

Además de hablar, hay además otras formas de comunicarse, como por ejemplo un código simple y bien escrito.

- **Simpleidad.** Consiste en desarrollar de la forma más simple posible y sólo el sistema que realmente se necesita. Implica resolver en cada momento sólo las necesidades actuales, con lo que disminuye la cantidad de código a escribir y se aumenta su calidad.

Se debe simplificar el diseño, documentar el código de manera simple pero aportando valor y refactorizar para eliminar redundancias y partes inservibles.

- **Retro-información (Feedback).** Conocer a través de los comentarios, reacciones, sugerencias, opiniones, etc. del cliente, cómo de cerca está el producto que se está construyendo de lo que realmente se necesita.

Una metodología basada en el desarrollo incremental iterativo de pequeñas partes, con entregas y pruebas frecuentes y continuas, proporciona un flujo de retro-información valioso para detectar los problemas o desviaciones de manera temprana.

- **Coraje.** Es necesaria la valentía para realizar una comunicación clara y eficiente con el cliente y con los compañeros y para afrontar sin miedo los cambios en el producto.

Implica saber tomar decisiones difíciles, reparar un error cuando se detecta, mejorar el código siempre que tras el feedback y las sucesivas iteraciones se manifieste susceptible de mejora, y tratar rápidamente con el cliente los desajustes de agendas para decidir qué partes y cuándo se van a entregar.

- **Respeto.** Ningún método funciona si no se trabaja basándose en el respeto mutuo, valorando el trabajo de los demás y teniendo presente cómo afecta al trabajo de las personas implicadas en el proyecto.

Y lista además 14 principios para guiar el desarrollo del software

- **Humanidad.** Es importante tener presente que el software lo desarrollan personas y por tanto deben considerarse sus necesidades humanas: sentirse seguro, satisfacción por sus logros, el sentido de pertenencia positiva en un grupo y su aportación en la consecución de sus objetivos, el crecimiento profesional al ampliar sus conocimientos, la capacidad de entender y ser entendido por otros.

Parte del desafío de un equipo de desarrollo de software es balancear correctamente las necesidades del individuo con respecto a las necesidades del equipo.

- **Economía.** Puesto que el desarrollo de software cuesta dinero, hay que asegurarse de que lo que se está construyendo aporta valor al negocio, cumple sus objetivos y sirve para sus necesidades. Resolver primero las necesidades prioritarias del negocio maximiza el valor del proyecto. Un desarrollo incremental con versiones del producto en producción permite obtener beneficios en el menor tiempo posible.

- **Beneficio mutuo.** Toda actividad debería beneficiar a todos los involucrados. El beneficio mutuo es el principio más importante de XP y el más difícil de conseguir. Se trata de buscar prácticas que me beneficien a mí ahora y más adelante, que también beneficien al cliente, pero que en ningún caso perjudique a algún interesado. Como ejemplos se pueden señalar: escribir test automáticos; refactorizar el código; elegir nombres claros y usar metáforas.

- **Auto-semejanza.** Tratar de copiar la estructura de una solución (un patrón de comportamiento) en un nuevo contexto, incluso a diferentes escalas. Por ejemplo, para desarrollar código, escribir un test que falla y después hacer que funcione. Esto puede aplicarse a distintas escalas. Para una Historia de usuario, definir sus criterios de aceptación y hacer que se cumplan.

Esto no quiere decir que siempre tratemos de aplicar los mismos patrones, a veces puede que sea necesario utilizar una solución realmente única y eso no quiere decir que esté mal.

- **Mejora continua.** En el desarrollo de software no se debe esperar al proceso perfecto, al diseño perfecto, a la Historia de usuario perfecta, pero si se debe perfeccionar el proceso, el diseño y las Historias de usuario. Cada día se debe pensar qué podríamos hacer mejor mañana.

- **Diversidad.** Los equipos necesitan personas con diversidad de conocimientos, de actitudes, de perspectivas para ver los problemas y de cómo resolverlos. Los conflictos entre personas que piensen diferente son inevitables, pero si se gestionan correctamente generarán conocimiento y enriquecerán las soluciones que se adopten. Dos ideas sobre un diseño son una oportunidad de aprender y mejorar, y no un problema. Ambas opiniones deberían tenerse en cuenta.

- **Reflexión.** Los buenos equipos no solo hacen su trabajo, también reflexionan cómo trabajan y por qué lo están realizando de esa manera. Analizan por qué tienen éxito o fallan. No ocultan sus errores, los exponen y aprenden de ellos. La programación por pares o la integración continua son momentos de reflexión.

La reflexión se debe traducir en acciones cuyos resultados se analizan después de aplicarse. No se debe caer en reflexiones demasiado profundas y largas, hay que producir, no sólo analizar para buscar la mejor opción.

- **Flujo.** XP propone abordar simultáneamente todas las actividades del desarrollo (análisis, diseño, pruebas y codificación) mejor que ir trabajando de manera secuencial y esperar a cerrar una fase para comenzar con la siguiente. Es más eficaz generar continuamente nuevos incrementos de valor del producto que compilan y también funcionan. Grandes entregas de código poco frecuentes suelen presentar mayores problemas difíciles de manejar, la retroalimentación es lenta y puede conducir a quedar estancado o necesitar cambios aun más grandes.

- **Oportunidad.** Para conseguir la excelencia, los problemas deben entenderse como una oportunidad de aprender y de mejorar, no sólo debe resolverse. XP propone prácticas eficaces

para resolver los problemas típicos del desarrollo del software. Si un programador comete muchos errores, cambie a programación por pares, si no es preciso en su planificación, realice refinamientos.

- **Redundancia.** Los problemas críticos en el desarrollo del software deben ser resueltos de varias maneras. Si una falla las otras alternativas pueden funcionar para evitar el desastre. El coste de la redundancia es menor que el beneficio que se obtiene. Por ejemplo, los defectos en XP se controlan aplicando varias prácticas: programación por pares, integración continua, sentarse juntos, involucración real del cliente y despliegue diario. Pero se deba asegurar que no se estén utilizando prácticas redundantes que puedan eliminarse sin impactar al producto.
- **Fallos.** Si no se sabe cuál de varias alternativas es la solución a un problema, normalmente la manera más rápida de encontrarla es probándolas todas y fallar en el resto, frente a buscar un consenso de solución perfecta. Aunque el fracaso puede parecer un desperdicio, el conocimiento adquirido es también muy valioso.
- **Calidad.** La calidad no es un control variable. Los proyectos no van más rápido aceptando una calidad menor y tampoco se desarrollan más despacio demandando mayor calidad. Normalmente, mayor calidad implica mayor eficiencia y productividad, y por ende entregas más rápidas y eficaces, y menor calidad, entregas impredecibles. La calidad puede ser medida en la cantidad de defectos generados, en la efectividad del diseño, en la efectividad de las pruebas automáticas, etc.
- **Pequeños pasos.** Corregir errores si se han hecho muchos cambios es más costoso que hacerlo sobre pequeñas modificaciones. Los posibles desperdicios generados son mucho menores. Es mejor hacer iteraciones cortas que produzcan un valor reconocible para el sistema. Los pequeños pasos pueden hacernos llegar muy lejos en poco tiempo. XP expresa este principio con prácticas como la programación dirigida por pruebas y la integración continua.
- **Aceptar la responsabilidad.** La responsabilidad no se puede asignar, sólo puede ser aceptada. Las posibilidades de no acertar con el trabajo que se asigna a una persona son altas. Cada miembro del equipo debe desarrollar, responsablemente, la mayor cantidad de trabajo de la mejor manera y tiempo que le sea posible.

XP define también un conjunto de prácticas que los equipos deben aplicar en su trabajo diario. Se clasifican en dos bloques:

- **Prácticas primarias**, que pueden aplicarse de manera independiente y que aportan un valor inmediato, mejorando el desarrollo del software:
  - **Sentarse juntos.** Los equipos de desarrollo deben trabajar en un espacio abierto, capaz de albergar a todo el equipo para maximizar la comunicación.
  - **El equipo completo.** Debe estar formado por personas con todas las habilidades y las perspectivas necesarias para el éxito del proyecto. Deben tener un fuerte sentido de pertenencia, y deben ayudarse unos a otros.
  - **Puesto de trabajo con información.** Desde el puesto de trabajo debe ser posible consultar el estado actualizado del proyecto y las tareas a realizar.
  - **Energía en el trabajo.** El equipo debe concentrarse sólo en su trabajo y ser productivos. Se debe conseguir un ritmo sostenido, evitando la necesidad de invertir horas extra.
  - **Programación en parejas.** El código lo escriben dos personas en la misma máquina. Se favorece el conocimiento compartido y aumenta la calidad del código.
  - **Historias.** Las funcionalidades del sistema se describen mediante historias y en lenguaje de cliente. Son breves descripciones de funcionalidad visibles para el cliente. Deben estar estimadas, es decir, el esfuerzo necesario para su desarrollo.
  - **Ciclos semanales.** El desarrollo de software se realiza en iteraciones de una semana. Al principio de cada semana el cliente decide que historias deben construirse con el fin de alcanzar un objetivo. Las historias se dividen en tareas técnicas que también son estimadas. Al finalizar se revisa el trabajo realizado y se detectan posibles desviaciones.
  - **Ciclos trimestrales.** Trimestralmente se revisa el estado del equipo, del proyecto y los avances. Se analiza si se mantienen los objetivos generales del proyecto y se detectan posibles impedimentos.

- **Holgura.** Al planificar las iteraciones se deben contemplar holguras de tiempo para los imprevistos. Tareas que puedan dejar de hacerse sin ocasionar demasiado impacto en el objetivo de la iteración. No comprometerse a lo que no se pueda cumplir.
  - **Construcción en diez minutos.** El objetivo es que en diez minutos y de manera automática, se pueda compilar el sistema completo y se ejecuten todas las pruebas. Las modificaciones de código que generen errores pueden ser detectadas y corregidas rápidamente sin que esto afecte al ritmo del resto del equipo.
  - **Integración continua.** Cada vez que se desarrolle una nueva funcionalidad, ésta debe integrarse con el resto del sistema, se debe recompilar y probar todo el sistema. Es más rápido y fácil detectar y corregir errores sobre piezas de código más pequeñas.
  - **Pruebas antes de programar.** Escribir las pruebas antes que el desarrollo obliga a centrarse y entender mejor el problema, asegurando además un diseño correcto. Con las pruebas unitarias se declara explícita y objetivamente que es lo que se espera que el programa haga. Construir código que es seguro que funciona genera confianza en el trabajo de los demás. Contar con pruebas automáticas permitirá verificar que cualquier modificación de código no hace que otro trozo de código deje de funcionar, o si es así, se podrá detectar de manera inmediata. Se aplicará el desarrollo dirigido por pruebas (Test Driven Development, TDD)
  - **Diseño incremental.** Se irá diseñando de manera incremental al tiempo que se construye el sistema. El diseño se enriquece y mejora cada día a partir del feedback del cliente y refactorizando el código con la frecuencia que sea necesaria. Las grandes decisiones de diseño se adoptan al principio del proyecto, retrasando las de pequeña escala para más adelante. Los equipos XP trabajan para crear condiciones bajo las cuales el costo de modificar software no sea un problema.
- **Prácticas corolario,** que deben utilizarse después de asegurar que las primarias están ya asentadas en la rutina de trabajo, ya que si no podrían causar graves problemas:
- **Participación real de los clientes.** Todos los afectados por el nuevo sistema deben ser parte del equipo. Clientes con buena visión del sistema pueden formar parte de la planificación trimestral y semanal.
  - **Despliegue incremental.** Para sustituir un producto existente es mejor modificar sólo alguna de sus funcionalidades, y poco a poco modificar el resto. Se debe evitar hacer todos los cambios de una sola vez.
  - **Continuidad de los equipos.** Mantener al equipo aunque se cambie de proyecto, mantener sus relaciones y lo que logran juntos. Ignorar las relaciones y confianza sólo por simplificar un problema de agenda es una falsa economía.
  - **Reducir los equipos.** Cuando un equipo crece en capacidades y productividad hay que mantener su carga de trabajo y gradualmente reducir su tamaño. Con la gente liberada se forman nuevos equipos.
  - **Análisis de las causas.** Siempre que se detecte un defecto o problema, éste debe eliminarse, pero también las causas que lo generaron. El objetivo no sólo es que no se reproduzca, sino que el equipo no vuelva a caer en este tipo de error otra vez.
  - **Propiedad colectiva del código.** El código es de todos, y cualquier miembro del equipo es capaz de modificar cualquier parte de éste aunque no sea quien lo desarrolló. La programación por parejas y las revisiones de código son una buena práctica para conseguirlo.
  - **Código y pruebas.** Son los dos elementos clave del proyecto que no pueden dejar de mantenerse y mejorarse nunca.
  - **Código base único.** Se debe evitar mantener varios programas que compartan parte de su código. Si se fuese necesario trabajar en paralelo al desarrollo principal, esto no deberá prolongarse más de varias horas.
  - **Despliegue diario.** Poner diariamente el código desarrollado en producción evita desfases entre la última versión y las copias locales, y mejora el feedback del usuario. Esto es difícil de conseguir, pues depende de que el número de fallos del sistema sea muy bajo y de que todos los procesos estén automatizados.
  - **Negociar el alcance del contrato.** Normalmente el alcance de los contratos debe ser renegociado durante el desarrollo del sistema. Los riesgos se reducen firmando una



secuencia de contratos cortos en lugar de uno largo. Tendremos mejor margen para negociar alcances.

- **Pagar por funcionalidad.** Habitualmente el cliente paga por cada entrega de producto. Esto genera conflictos entre el proveedor y el cliente, el cual quiere que cada una de ellas contenga el máximo posible de funcionalidades. Si se paga por funcionalidad realizada, se sabrá con exactitud y en el momento preciso, hacia dónde dirigir el desarrollo.

## 2 Técnicas

Se detallan a continuación un conjunto de técnicas que pueden ayudar a la realización de las actividades y tareas propuesta por la metodología, pero que en ningún caso deben considerarse como únicas y de uso obligatorio.

### 2.1 Personas y escenarios

Se trata de una herramienta normalmente utilizada para el diseño de sitios web que se basa en la creación del concepto "persona" y el uso que estas hacen del sistema, los "escenarios".

Las Personas representan patrones de conducta, objetivos y necesidades de grupos de personas, que si bien son ficticios, tienen características comunes basadas en el análisis sobre la audiencia real. Todos los atributos y características de la Persona deben estar basados en información real extraída de los usuarios. Definiendo Personas se consigue entender de manera clara quiénes utilizarán el sistema.

Los Escenarios describen situaciones de uso de la aplicación sobre los que poder contextualizar la interacción persona-sistema, de modo que podamos saber para quién se desarrolla y qué espera encontrar el usuario. El uso de escenarios contextuales va a permitir tener una idea de cómo afectan las motivaciones de los usuarios, la manera en la que estos priorizan las tareas y en consecuencia van a ayudar a responder las siguientes preguntas: ¿Qué debería hacer el sistema? y ¿Cómo debería presentarse y comportarse?

La aplicación de esta técnica para generar la Visión del producto nos permite identificar los usuarios del sistema, y los atributos que debe tener según el orden de criticidad establecido por el usuario.

### 2.2 Sesiones de trabajo

Las sesiones de trabajo pueden ser de varios tipos en función de las personas que participen en ellas, el objetivo que se persiga y el modo de llevarlas a cabo.

Dentro de estas sesiones de trabajo se encuentran algunas técnicas como son el JAD (Joint Application Design), y otras prácticas como las entrevistas y las reuniones.

A continuación se explica brevemente el objetivo principal de cada una de ellas, antes de describir más en detalle la forma de llevarlas a cabo.

- Las entrevistas son un tipo de sesiones de trabajo dirigidas a obtener la información de una forma individual dónde aparecen los perfiles de entrevistado y entrevistador.
- Las reuniones pueden tener el mismo objetivo, pero la información está dispersa entre varias personas y únicamente trabajando en grupo se conseguirá extraer y depurar toda la información de forma global.
- Las sesiones JAD son reuniones en las que se potencia el trabajo en equipo entre el cliente o usuario y el proveedor, con una participación más activa del cliente en los diferentes procesos del ciclo de vida que va a permitir identificar las necesidades planteadas, proponer soluciones, negociar enfoques diferentes y especificar el conjunto preliminar de requisitos que debe cumplir la solución para llegar al objetivo que se propone.

#### 2.2.1 Entrevistas

Las entrevistas constituyen un medio para obtener la información que se necesita sobre un determinado tema, como puede ser, el establecer el alcance de un problema, identificar los requisitos a cubrir por un sistema de información y analizar el funcionamiento de un sistema actual, entre otros, a partir de las personas que tienen conocimiento sobre el mismo.

Se entiende por entrevista el encuentro que se realiza "cara a cara" entre un usuario (o cliente) y la persona responsable de obtener la información.

Para realizar la entrevista solo es necesario designar a las personas que deben participar en ella y determinar el lugar en el que poder llevarla a cabo. Es importante identificar a qué tipo de perfil va dirigida la entrevista, a quiénes se va a entrevistar y cuál es el momento más oportuno, con el fin de evitar situaciones embarazosas y conseguir que la entrevista sea eficaz y productiva.

Como paso previo a la realización de la entrevista se deben tener en cuenta una serie de reglas generales o directrices básicas:

- Desarrollar un plan global de la entrevista.
- Asegurarse de que se cuenta con la aprobación para hablar con los usuarios.
- Preparar la entrevista previamente.
- Realizar la entrevista.
- Consolidar el resultado de la entrevista.

Además, es conveniente planificar las entrevistas estudiando la secuencia en que se van a llevar a cabo, en función de los distintos perfiles implicados y las relaciones existentes entre los entrevistados. Según la información a obtener y dependiendo de las distintas fuentes que pueden proporcionarla, puede ser necesario realizar una entrevista conjunta con varias personas.

Durante la preparación de la entrevista es imprescindible remitir al usuario un guion previo sobre los puntos a tratar, para que pueda estudiarlo con tiempo y solicitar la información que estime conveniente para la entrevista. Se debe pensar bien el tipo de guion, según el perfil y las responsabilidades del entrevistado y su extensión, de forma que se pueda conseguir la suficiente información, sin provocar rechazo en el entrevistado. Si se considera apropiado se pueden utilizar herramientas automatizadas.

Una vez que se dispone de la aprobación para hablar con los usuarios, se hace la convocatoria de la entrevista enviando la información oportuna y fijando los objetivos, el método de trabajo que se va a seguir y el tiempo del que se dispone.

Para realizar la entrevista, es importante hacer un resumen general de los temas a tratar, utilizar un estilo apropiado y crear desde su inicio un clima de confianza entre los asistentes. Es posible que el entrevistado se resista a aportar información, siendo útil en estos casos utilizar técnicas específicas de comunicación.

Antes de finalizar la entrevista es importante que el entrevistador sintetice las conclusiones y compruebe que todos los asistentes están de acuerdo, dejando siempre abierta la posibilidad de volver a contactar para aclarar temas que surjan al estudiar la información recopilada.

Finalmente, el responsable depura y consolida el resultado de las entrevistas, elaborando un informe de conclusiones. En algunos casos puede ser conveniente elaborar un acta que refleje estas conclusiones y remitirla a los entrevistados con el objetivo de asegurar que se han comprendido bien las especificaciones dadas.

## 2.2.2 Reuniones

Las reuniones tienen como objetivo obtener información que se encuentra repartida entre varias personas, tomar decisiones estratégicas, tácticas u operativas, transmitir ideas sobre un determinado tema, analizar nuevas necesidades de información, así como comunicar los resultados obtenidos como consecuencia de un estudio o trabajo realizado.

Para realizar una reunión es necesario designar a las personas que deben participar en ella y determinar el lugar en el que poder llevarla a cabo. Las directrices básicas de una reunión son:

- Preparar y convocar la reunión (orden del día).
- Realizar la reunión.
- Consolidar el resultado de la reunión.
- Elaborar el acta de reunión (si se considera necesario)

Previamente a la convocatoria de la reunión, se definen los objetivos, se planifica el método de trabajo que se va a seguir y el tiempo del que se dispone, se eligen los participantes y se prepara el material necesario.

Después de la preparación, es imprescindible enviar al usuario la convocatoria con el orden del día de la reunión. Este orden incluye la fecha, hora de inicio, hora de finalización prevista, lugar, asistentes y los puntos a tratar, detallando, entre otros, el tiempo que se dedicará a cada tema y la persona responsable de exponerlo. Dicha convocatoria se envía con antelación suficiente para que los asistentes puedan organizar su agenda y prepararse para la reunión con tiempo.

Al inicio de la reunión, es importante hacer un resumen general de los temas a tratar, los objetivos que se persiguen, el método de trabajo y la agenda de la reunión. Si se considera oportuno se puede utilizar la técnica de presentación. Desde su inicio se debe crear un clima de confianza entre los asistentes.

Uno de los primeros puntos recomendables es la revisión del acta anterior y del estado de las tareas señaladas, si existen.

La persona responsable de la reunión ejercita la dinámica de dirección de grupos, estimulando la participación, controlando el ritmo de la sesión y centrando o clarificando el tema cuando sea necesario. Al finalizar, se sintetizan las conclusiones, se comprueba si hay acuerdo o si quedan puntos pendientes de reflexión y se propone fechas para próximas reuniones.

El responsable de tomar las notas en la reunión, levanta el acta (si así se ha acordado) y la remite a los asistentes que deben confirmar su recepción.

Cuando el número de tareas sea muy elevado puede ser conveniente llevar un documento paralelo para el control de las tareas pendientes. Este documento no debe sustituir la incorporación en el acta de las actividades acordadas.

También puede ser conveniente crear una entrada en la bitácora del proyecto (documento de monitorización) con los temas más relevantes y un vínculo al acta, de forma que pueda localizarse y referenciarse el documento correspondiente.

### 2.2.3 JAD (Joint Application Design)

JAD es una técnica utilizada, entre otros fines, para la recopilación de requisitos en el desarrollo de nuevos sistemas de información. El proceso JAD también incluye aproximaciones para aumentar la participación de los usuarios, agilizar el desarrollo, y mejorar la calidad de las especificaciones. Consiste en un taller donde los usuarios y técnicos se reúnen, algunas veces durante varios días, para definir y revisar los requisitos de negocio para el sistema.

- Las características de una sesión de trabajo tipo JAD se pueden resumir en los siguientes puntos:
  - Se establece un equipo de trabajo cuyos componentes y responsabilidades están perfectamente identificados y su fin es conseguir el consenso entre las necesidades de los usuarios y los servicios del sistema en producción.
  - Se llevan a cabo pocas reuniones, de larga duración y muy bien preparadas.
  - Durante la propia sesión se elaboran modelos empleando diagramas fáciles de entender y mantener.
  - Al finalizar la sesión se obtienen un conjunto de modelos que deberán ser aprobados por los participantes.

Es importante definir claramente el perfil y las responsabilidades de los participantes de una sesión JAD. Se pueden distinguir los siguientes perfiles:

- Moderador (líder JAD) con amplios conocimientos de la metodología de trabajo, dinámica de grupos, psicología del comportamiento, así como de los procesos de la organización objeto del estudio.
- Promotor, persona que ha impulsado el desarrollo.
- Jefe de proyecto, responsable de la implantación del proyecto.
- Especialista en modelización, responsable de la elaboración de los modelos en el transcurso de la sesión.
- Desarrolladores, aseguran que los modelos son correctos y responden a los requisitos especificados.
- Usuarios, responsables de definir los requisitos del sistema y validarlos.

La sala en la que se llevarán a cabo este tipo de sesiones juega un papel muy importante debido a que, en las reuniones largas donde se requiere una alta concentración de todos los integrantes del equipo, es necesario prestar especial atención a aspectos relacionados con la temperatura, los medios audiovisuales, la buena visibilidad de los distintos participantes, etc.

Para llevar a cabo una sesión JAD, es necesario realizar una serie de actividades antes de su inicio, durante el desarrollo y después de su finalización. Estas actividades se detallan a continuación:

- Inicio: se define el ámbito y la estructura del proyecto, los productos a obtener, se prepara el material necesario para la sesión, se determina el lugar donde se van a llevar a cabo, se seleccionan los participantes y se sugiere una agenda de trabajo.
- Desarrollo: se identifican las salidas del proyecto y se debe conseguir el consenso entre los participantes de modo que se materialice en los modelos.
- Finalización: se valida la información de la sesión y se generan los productos de la metodología de trabajo propuesta. Si fuera necesario se integran los productos de salida.

En las sesiones de trabajo tipo JAD se distinguen dos tipos de productos:

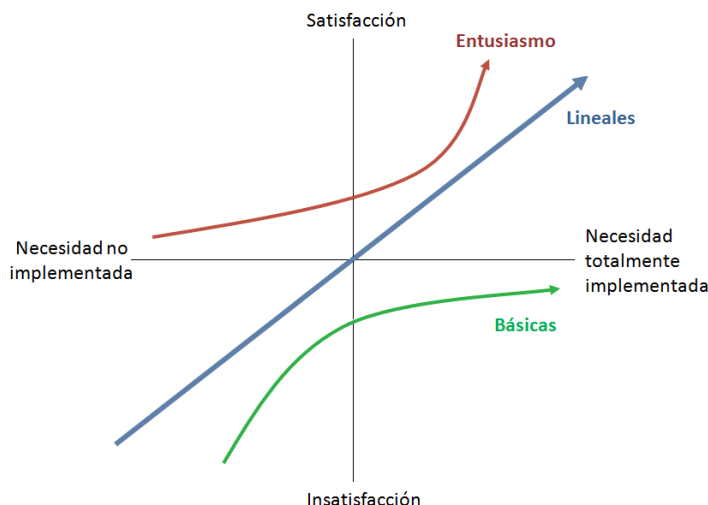
- De preparación donde se incluye, entre otros, la Historia y contexto del proyecto, los objetivos y límites, las actividades del entorno del negocio que pueden afectar al éxito del proyecto y los beneficios.
- De resultado de las sesiones de trabajo, que se establecen con anterioridad al inicio de las reuniones.

## 2.3 El modelo de Kano

El modelo de Kano es una teoría de desarrollo de productos y de satisfacción del cliente elaborada en la década de 1980 por el profesor Noriaki Kano. De las 5 características definidas en el modelo, la satisfacción del cliente se puede alcanzar con 3 de ellas:

1. Las básicas u obligatorias. Son aquellas que si no están presentes, producen un alto nivel de insatisfacción, pero si están, producen un sentimiento neutro. Es lo que se esperaba del producto. Por ejemplo, en un hotel, tener sábanas limpias.
2. Las lineales. Se caracterizan porque la satisfacción que producen es aproximadamente proporcional al número de ellas incluidas en el producto. Por ejemplo, el número de canales de TV que se ofrecen en una habitación de hotel.
3. Las de entusiasmo. Si están, aumentan la satisfacción del cliente, le sorprenden y le agrada, pero si no están, no generan descontento. Por ejemplo, un descuento especial inesperado por su estancia en el hotel.

Para cada de ellas, el modelo establece la relación entre la satisfacción del cliente y el cumplimiento de la necesidad o funcionalidad.



A la hora de construir la Visión del producto, el reto es detectar el tipo de características del sistema que se quieren incluir (atributos críticos), para maximizar los beneficios que el cliente obtendrá con esta combinación de características básicas, lineales y de entusiasmo.

Este modelo también se puede aplicar para priorizar los ítems de la Pila Producto si lo que se desea es orientarlo a mejorar la satisfacción del cliente con el desarrollo que se va a realizar. En este caso se utilizarán las otras dos características del modelo que no mencionadas hasta ahora:

4. Indiferente. La presencia de esa funcionalidad en esa cuantía no afecta en ningún modo al usuario.
5. Reversible. Relativos a la insatisfacción que pueden generar por hecho de que no todos los clientes son iguales. Por ejemplo, algunos clientes prefieren los productos de alta tecnología, mientras que otros prefieren el modelo básico de un producto y no estarán satisfechos si un producto tiene muchas características adicionales.

Para aplicarlo, se deben seguir los siguientes pasos:

1. Categorizar los ítems como básicos, lineales o de entusiasmo. Para ello, lo más práctico es hacer una encuesta a los clientes y usuarios en la que se preguntará por lo temas y épicas desde dos perspectivas opuestas:
  - Pregunta funcional: ¿Cómo te sentirías si esta funcionalidad estuviera presente?
  - Pregunta disfuncional: ¿Cómo te sentirías si esta funcionalidad no estuviera presente?

Ambas deberán responderse con la escala de 5 posibles respuestas:

1. Me gusta
  2. Lo esperaba
  3. Me es indiferente
  4. Podría aceptarlo
  5. No me gusta
2. Cruzar las respuestas para obtener la categorización de cada uno de los ítems según el modelo. Se añade la opción "Cuestionable", que indica un resultado no coherente o contradictorio. Los cruces se realizan en base a una tabla de referencia:

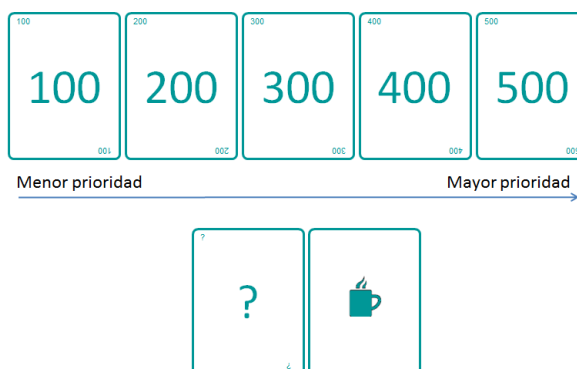
		Pregunta disfuncional				
		Me gusta	Lo esperaba	Me es indiferente	Podría aceptarlo	No me gusta
Pregunta funcional	Me gusta	C	E	E	E	L
	Lo esperaba	R	I	I	I	B
	Me es indiferente	R	I	I	I	B
	Podría aceptarlo	R	I	I	I	B
	No me gusta	R	R	R	R	C

Se priorizarán, en este orden, todos los básicos, casi todos los lineales y alguno de los de entusiasmo.

## 2.4 Poker de Prioridad (Priority Poker)

El Poker de Prioridad es un juego diseñado para establecer prioridades de manera colaborativa entre elementos de cualquier tipo: objetivos, requisitos, metas, etc. El método fomenta ya no solo la asignación de prioridades, sino también la discusión en grupo para establecer dichas prioridades, lo que ayudará a garantizar la fiabilidad de la priorización.

Para aplicarlo es necesario además de, evidentemente, una lista de elementos para priorizar, decidir el número de niveles que se va a utilizar. Estos niveles se trasladarán a las cartas de una baraja, que serán las que utilizará cada uno de los jugadores para realizar sus valoraciones. A esta baraja se deben añadir dos adicionales: “necesito más información” y “necesito un descanso”. En el juego, el valor que se asigne no es significativo, lo que importa es que este sea relativo con respecto a los demás. No hay que olvidar que el resultado final será una lista ordenada de elementos.



El juego consta de varios pasos:

- Se explica el elemento que se va a estimar (máximo, 2 minutos)
- Cada participante reflexiona sobre el valor que va a asignar (máximo, ½ minuto)
- Simultáneamente todos descubren la carta seleccionada
- Si hay consenso, se asigna el valor propuesto al elemento y se pasa al siguiente

- Si no hay consenso, los participantes que hayan propuesto el valor más alto y el valor más bajo explicarán los motivos de su valoración (máximo, 1 minuto para cada uno)
- Se vuelve a dar un plazo para recapacitar y se descubren nuevamente las cartas
- El proceso se repite hasta que se alcance un acuerdo entre todos los participantes

En caso de empate entre varios elementos se estimarán sólo estos, pero teniendo en cuenta que el valor asignado es relativo sólo con respecto a dichos elementos.

El juego no debe durar más de 2 horas, respetando además los tiempos marcados para cada paso. No se deben permitir discusiones de si un elemento debe estar o no en la lista, o si el enfoque de un elemento es o no el correcto, sólo se deben permitir explicaciones que ayuden a su comprensión.

## 2.5 MoSCoW (Must, Should, Could, Won't)

MoSCoW es una técnica de priorización que puede aplicarse a requerimientos, tareas, productos, casos de uso, Historias de usuario, criterios de aceptación y pruebas. Se basa en la segmentación y agrupación de los elementos en cuatro categorías posibles, lo que ayuda a entender cuáles son los criterios que se deben aplicar:

- Debe tener (**Must Have**). Se trata de un elemento clave y esencial, que si no se entrega, los interesados no estarán satisfechos y se considerará un fracaso.
- Debería tener (**Should have**). Es importante, pero no estrictamente necesario. Son tan significativos como los “must”, pero podrían resolverse un poco más tarde, o de otra manera alternativa aunque menos satisfactoria.
- Podría tener (**Could have**). Son menos críticos, pero estaría bien tenerlos. Podrían incrementar la satisfacción de los interesados con un coste de desarrollo bajo.
- No tendrá por ahora (**Won't have this time**). Son los menos críticos, los que menos valor aportan y los que no son necesarios por ahora. Suelen ayudar a aclarar el alcance y se añaden porque posteriormente podría ser descartados definitivamente o bien reconsiderarse para su inclusión.

En otras técnicas de priorización, el uso de valores como alto, medio o bajo, no indica de manera natural su significado. En MoSCoW, las prioridades reflejan el resultado de entregar o no el elemento que se está evaluando, lo que podrá indicar, si se trata por ejemplo de los requisitos del sistema, el éxito o no del producto para los interesados.

Una vez realizada la priorización siguiendo la técnica MoSCoW, se pueden aplicar otras técnicas que, por cada categoría, ordene sus respectivos elementos.

## 2.6 Poker de planificación o de estimación (Planning Poker)

El Poker de estimación es un juego diseñado para realizar estimaciones del tamaño relativo entre elementos de la Pila de Producto. El método fomenta ya no solo la estimación de tamaños, sino también la discusión en grupo, lo que ayudará a garantizar la fiabilidad de la estimación.

Para aplicarlo es necesaria una baraja de cartas para cada jugador, que serán las que utilizará para realizar sus valoraciones. Las cartas están numeradas usando una serie basada en la de Fibonacci: 0, ½, 1, 2, 3, 5, 8, 13, 20, 40 y 100. El fin de utilizar diferencias cada vez mayores sirve para evitar que se trate de dar estimaciones ajustadas cuando se trate de elementos que merecen valores más bien grandes, puesto que en estos casos es fácil incurrir en errores de apreciación. A la baraja se le añaden además otras dos: una con el símbolo de interrogación (?) que significa “no estoy seguro” y otra con una taza de café que quiere decir “necesito un descanso”.



Durante el juego, el valor (el número) que se asigne no es significativo, lo que importa es que este sea relativo con respecto a los demás. Una Historia de usuario con estimación 2 tendrá un tamaño el doble de grande que otra con valor 1.

El juego debe contar con un moderador que no participa en la valoración, que puede ser el Facilitador o el Dueño del Producto. Consta de varios pasos:

- El Dueño del Producto explica el elemento que se va a estimar
- El equipo realizará todas las preguntas que sean necesarias hasta tener claro de qué se trata
- Cada participante reflexiona sobre el valor que va a asignar
- Simultáneamente todos descubren la carta seleccionada
- Si hay consenso, se asigna el valor propuesto al elemento y se pasa al siguiente
- Si no hay consenso, los participantes que hayan propuesto el valor más alto y el valor más bajo explicarán los motivos de su valoración
- Se inicia una ronda de diálogo y discusión tras al cual
- Se vuelve a dar un plazo para recapacitar y descubrir nuevamente las cartas
- El proceso se repite hasta que se alcance un acuerdo entre todo los participantes
- Si no se alcanza un acuerdo, se podrá posponer para más adelante

Para evitar que cada estimación se prolongue demasiado se limitará el tiempo dedicado a cada uno de los pasos del juego.

Esta técnica suele producir estimaciones menos optimistas que otras y que además suelen resultar bastante precisas.

## 2.7 El tablero de tareas (Task Board)

El tablero de tareas muestra de manera visual el progreso del equipo respecto a la Pila de Sprint, de modo que de un solo vistazo todo el equipo es capaz de saber si alguno de sus miembros está trabajando ya en una tarea, si se ha completado una Historia de usuario, el trabajo que queda todavía pendiente, etc. Fomenta por lo tanto la transparencia de la información y permite al equipo organizarse y realizar un seguimiento de los progresos realizados durante el Sprint.

Un tablero de tareas puede representarse en una pared en blanco, en una pizarra, en una hoja de cálculo o en alguna herramienta informática específica.

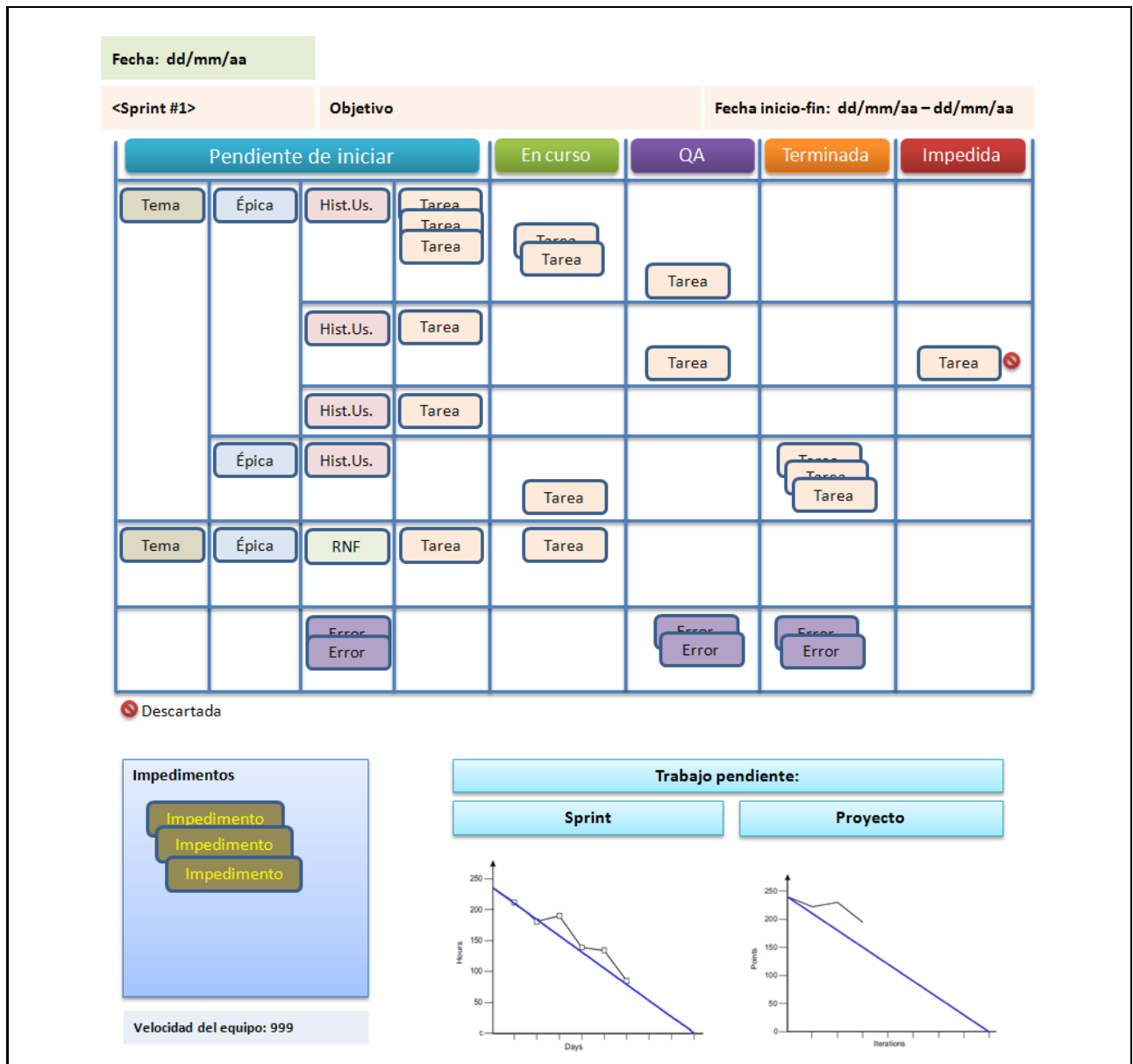
El tablero se organiza en varias columnas, que indican el estado de una tarea. Cada tarea se describe en tarjetas autoadhesivas, las cuales van moviéndose a lo largo del tablero según se vaya completando el trabajo necesario para su resolución. Los temas, épicas, Historias de usuario, requisitos no funcionales y los defectos y errores también pueden representarse en tarjetas que se colocarán sobre el tablero. Suele ser de gran utilidad utilizar tarjetas de colores que ayuden a identificar fácilmente la tipología de cada una de ellas.

Título						
Identificador	Tema		Épica		Código	
Prioridad [1-500]		Tamaño		Peticionario		
Descripción						
Criterios de aceptación						
Sprint			Estado			
Observaciones						

*Tarjeta de Historia de usuario*

Se pueden establecer también filas que agrupen los temas, épicas, Historias de usuario y sus respectivas tareas. Se recomienda además que una de las filas se reserve para recoger los defectos o errores que vayan surgiendo a lo largo del proyecto.

Estos tableros se pueden complementar con información de monitorización o seguimiento: gráfico de trabajo pendiente del equipo, velocidad media, gráfico de trabajo pendiente del Sprint, etc.; o con información sobre los impedimentos.



Tablero de Tareas

## 2.8 Estimación de la velocidad del equipo

La velocidad del Equipo de desarrollo se calcula como el sumatorio del número de Puntos de Historia que este es capaz de resolver en una iteración.

Hay varias formas de estimar la velocidad del equipo:

- Aplazar la estimación hasta que se hayan desarrollado unas cuantas iteraciones. Normalmente esta es la mejor opción, pero es evidente que no se podrá aplicar si se quiere elaborar el plan inicial de entregas de la etapa de Preparación o Sprint 0.
- Utilizar valores históricos medios obtenidos en proyectos anteriores (velocidad media y duración media de las iteraciones). En este caso, además, se deberían considerar otro tipo de factores que

puedan afectar al cálculo: si se han producido cambios significativos en el equipo, la naturaleza del proyecto, la tecnología, etc.

- Desglosar unas cuantas Historias de usuario en tareas y estimar el número de días ideales que podría llevar su desarrollo. De esta forma podemos obtener el número total de días ideales (suma) que se necesitan para resolver los puntos de historia relativos a las Historias de usuario seleccionadas. Y con estos valores, calcular la velocidad del equipo. Pero también en este caso intervienen otros parámetros más: el número de personas asignadas al equipo de desarrollo, y el Factor de dedicación (Focus factor), que indica el porcentaje de tiempo que se dedica de manera efectiva al trabajo. Este varía en función de la persona, su rol, y la organización, pero en condiciones normales se puede considerar que puede estar entre un 50% y un 80%.

Así, la fórmula de cálculo a aplicar podría ser:

$$\text{Velocidad del equipo} = \frac{T \times Ph \times F \times P}{Di \times 100}$$

*T = Tiempo fijado para el sprint (en días)*

*Ph = Suma de puntos de historia asignados a las Historias de usuario seleccionadas*

*F = Factor de dedicación (en valor de porcentaje)*

*P = Número de personas asignadas al equipo de desarrollo*

*Di = Suma de días ideales estimados para resolver las Historias de usuario seleccionadas*

## 2.9 Desarrollo dirigido por Pruebas (TDD)

El desarrollo dirigido por pruebas de software (Test-driven development, TDD) es una técnica de diseño e implementación de software propuesta por XP. Se debe seguir el siguiente proceso:

- En primer lugar, se escribe una prueba y se verifica que estas fallan. Realmente se puede considerar que lo que se va a codificar es un ejemplo o especificación del requisito que se quiere resolver, es decir, los requisitos se traducen a pruebas.
- A continuación, se implementa el código que hace que la prueba pase satisfactoriamente
- Y por último, se refactoriza el código para eliminar duplicidad y hacer mejoras. Se trata de modificar el diseño sin alterar su comportamiento (a ser posible, sin alterar su API pública) y rastrear el código en busca de líneas duplicadas que serán eliminadas. También se asegurará que el código cumpla con ciertos principios de diseño.

## 2.10 Desarrollo dirigido por Pruebas de Aceptación (ATDD)

Esta técnica es como el TDD pero a un nivel de especificación diferente, en este caso, para pruebas de aceptación o de usuario. Dichas pruebas equivalen al criterio escrito de que el software cumple los requisitos de negocio que el cliente demanda.

## 2.11 Integración continua (Continuous Integration)

La integración continua es una sistemática de desarrollo propuesto inicialmente por Martin Fowler que consiste en hacer integraciones automáticas de un proyecto lo más a menudo posible para así poder detectar fallos cuanto antes. Entendemos por integración la compilación y ejecución de pruebas de todo un proyecto.

Este proceso se suele ejecutar cada cierto tiempo (normalmente horas), y consta de los siguientes pasos:

- Descargar el código fuente desde el sistema de control de versiones
- Compilarlo
- Ejecutar pruebas
- Generar informes de resultado

## 2.12 Spikes

Los “Spikes” son una aportación de la Programación Extrema (Extreme Programming, XP) quien lo define como “un programa muy simple para explorar soluciones potenciales”. Se desarrollan como un tipo especial de Historia de usuario, aunque no genere un incremento directo del valor del sistema. Se puede utilizar por diversos motivos:

- Para investigar sobre un nuevo dominio de conocimiento que el equipo no posee, facilitando así el uso de esa nueva tecnología o dominio.
- Analizar las implicaciones de una Historia de usuario demasiado grande que de otro modo es difícil de estimar, o de dividir en otras más pequeñas.
- El equipo quiere investigar o hacer algún prototipo para adquirir el conocimiento básico necesario para resolver una Historia de usuario con un riesgo técnico significativo.
- Para una Historia de usuario con un riesgo funcional importante, un “Spike” permitirá definir claramente cómo debe interactuar el usuario con el sistema para conseguir el beneficio esperado.

Como cualquier otra Historia de usuario los “Spikes” deben ser estimados e incluidos en una Pila de Sprint.

Pueden tomar dos formas:

- “Spike” técnico. Se utilizan para investigar varias aproximaciones técnicas dentro del dominio de la solución. Por ejemplo, evaluar el impacto en el rendimiento o carga de una nueva Historia de usuario; evaluar posibles implementaciones tecnológicas aplicables a una solución técnica, etc.
- “Spike” funcional. Se utilizan cuando hay cierta incertidumbre sobre cómo el usuario puede interactuar con el sistema. Normalmente se utilizará algún nivel de prototipado para realizar la evaluación: “mock-ups”, “wireframes”, “page flows”, etc.

### 3 Bibliografía y referencias

*Métodos ágiles y Scrum* (Anaya Multimedia). Alonso Álvarez García, Rafael de las Heras del Dedo, Carmen Lasa Gómez

*A Guide to the Scrum body of knowledge (SBOK™ guide), 2013 Edition* (VMEdU, Inc.). SCRUMstudy

*Agile Estimating and Planning* (Prentice Hall). Mike Cohn.

*Extreme Programming Explained: Embrace Change, 2nd Edition* (Addison-Wesley). Kent Beck, Cynthia Andres

*Scrum y XP desde las trincheras* (InfoQ). Henrik Kniberg

*Scrum y eXtreme Programming para Programadores*. Eugenia Bahit

<http://agilemanifesto.org/iso/es/>

<https://www.scrum.org/>

<http://www.proyectosagiles.org/>

<http://www.scrummanager.net/>

<http://www.navegapolis.com>

[http://www.scrummanager.net/bok/index.php?title=Criterios para decidir el tipo de gestión de proyecto más adecuado](http://www.scrummanager.net/bok/index.php?title=Criterios_para_decidir_el_tipo_de_gestión_de_proyecto_más_adeecuado)