1. METODOLOGÍAS ÁGILES

Autor: Esp. Damián Barry

1.1 INTRODUCCIÓN

Los negocios actuales, su dinámica y la necesaria adaptación al mercado requieren de enfoques adaptativos para la producción, gestión y comercialización de los productos de una empresa. Por ello la gestión de proyectos ágiles es necesaria en la actualidad para ajustarse y aprovechar los nuevos entornos cambiantes de los mercados.

Las metodologías ágiles parten de las enseñanzas de la metodología de gestión de proyectos LEAN.

La gestión de proyectos LEAN sigue el ciclo de cambio de Deming y se basa en un proceso de tipo iterativo e incremental. En la gestión de proyectos tradicional, el producto se entrega al final y se realizan muchas actividades a la vez. La gestión de proyectos LEAN busca entregar valor a los clientes de forma rápida y continua, gracias a la extracción de productos a entregar y a la adecuación de su procesamiento al límite del trabajo en progreso actual.

El origen de las metodologías de Gestión LEAN está vinculado con la industria del automóvil en Japón, ante la competencia de Estados Unidos en este sector. Concretamente, surge del toyotismo o sistema de producción de Toyota, que optimiza los procesos de producción. Supuso una mejora de la productividad, considerando aspectos como los inventarios, el porcentaje de la fuerza de trabajo en el equipo, el número de clases de trabajos, el adiestramiento de los trabajadores y el absentismo.

Las metodologías ágiles surgen de un grupo de profesionales de la informática que, adoptando y adaptando las enseñanzas LEAN difunden sus valores y principios, potenciando y dotando de un conjunto de metodologías y prácticas que garantizan la visión adaptativa de entrega de valor permanente al negocio.

La diferencia sustancial con el enfoque tradicional de gestión de proyectos se basa en su fundamento de tirar (pull systems) en contraposición a los métodos tradicionales de gestión de tiempo fijo (timeboxing) basados en empujar (push systems).

Hoy en día estos dos enfoques metodológicos de gestión de proyectos son englobados dentro de los que se denominan metodologías ágiles o metodologías incrementales adaptativas.

En la página 46 del PMBok define el “Ciclo de vida adaptativo” cómo: *“Generalmente se opta por métodos adaptativos en entornos que cambian rápidamente, cuando los requisitos y el alcance son difíciles de definir con antelación y cuando es posible definir pequeñas mejoras graduales que aportarán valor a los interesados”*, destinando varías secciones a definir y enmarcar a las metodologías ágiles. Lo mismo ocurre con el Project management Institute (PMI).

1.2 EL AGILISMO A TRAVÉS DE LAS METODOLOGÍAS

1.1.1 Los Principios y valores

1..1.1.1 Los principios LEAN

- Eliminar desperdicios (Eliminating Waste).
- Amplificar el aprendizaje (Amplifying Learning).
- Decidir lo más tarde posible (Deciding as Late as Possible).
- Entrega lo más rápido posible (Delivering as Fast as Possible).
- Capacitar, potenciar, al equipo (Empowering the Team).
- Construir con integridad (Building Integrity In).
- Ver el todo (Seeing the Whole).

La mentalidad LEAN propone un flujo de trabajo basado en la entrega de valor y la reflexión, logrando de esta forma la eliminación de los desperdicios, logrando así la mejora contínua a través de la colaboración de las personas.

Por otro lado…

1.1.1.2 Manifiesto de las metodologías ágiles

En el mismo se propone que… estamos descubriendo formas mejores de producir tanto por nuestra propia experiencia como ayudando a terceros. A través de este trabajo hemos aprendido a valorar:

- 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 valoramos los elementos de la derecha, valoramos más los de la izquierda.

Además, se proponen los siguientes principios:

1. Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de productos y servicios 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 producto funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.
4. Los responsables de negocio y los integrantes del equipo 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 producto o servicio 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 a continuación ajustar y perfeccionar su comportamiento en consecuencia.

1.3 SISTEMAS DE EMPUJAR VERSUS TIRAR (PUSH VS. PULL)

Lo más relevante del planteo realizado por las metodologías ágiles es la forma de planificar y gestionar los proyectos. En este sentido se deben contraponer los dos grandes esquemas de planificación, control y seguimiento de proyectos.

1.3.1 Sistemas de empujar (push system)

Por una parte, encontramos los proyectos de gobernanza tradicional de gestión de proyectos que se basan en un esquema predictivo que implica empujar las tareas y las actividades para producir un bien o servicio donde se presupone la demanda que se debe producir del mismo. Normalmente identificamos este tipo de proyectos mediante el uso del clásico diagrama de GANTT de tareas a tiempo y recursos fijos.

En el sistema Push (empujar), las empresas conciben la producción de los bienes y servicios en función de un pronóstico de la demanda o de un itinerario determinado de trabajo. El principal problema de este sistema radica en que no siempre los pronósticos son correctos y a menudo se cae en una sobreproducción, lo que a la larga genera un gran desperdicio de acumulación innecesaria, tanto del trabajo como de los pasos intermedios de producción.

En definitiva, la metodología Push se refiere cuando el proceso de producción elabora un producto terminado o en proceso sin importar si el siguiente proceso o usuario de consumo lo necesita o tiene la capacidad para su respectivo consumo.

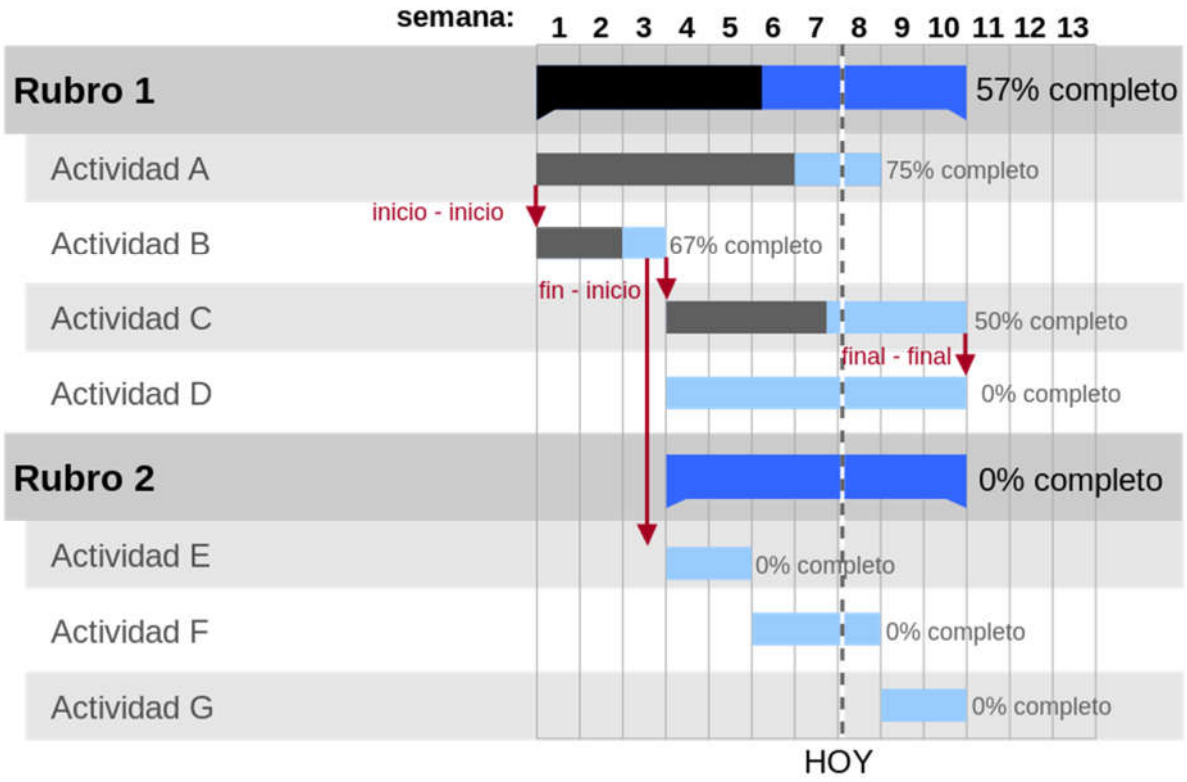

Figura 1.1 - Diagrama de Gantt. Gestión de actividades de metodologías push.

1.3.1.1 Metodología de gestión de proyectos en cascada

¿Cuál es la forma más habitual de planificar un proyecto? Ordenando las tareas que conducen a una entrega final y trabajando en ellas en orden. Nos referimos a la metodología en cascada: el método tradicional para gestionar proyectos y el más fácil de entender. Una tarea debe completarse antes de que comience la siguiente, en una secuencia conectada de elementos que se suman a la entrega general. Es un método ideal para proyectos cuyo resultado son objetos físicos (edificios, ordenadores), y los planes de proyecto pueden replicarse fácilmente para uso futuro.

El punto fuerte de esta metodología de proyecto es que cada paso se planifica previamente y se dispone en la secuencia adecuada. Aunque este puede ser el método más sencillo de implementar inicialmente, cualquier cambio en las necesidades o prioridades de los clientes interrumpirá la secuencia de tareas, haciendo que sea muy difícil de gestionar. Esta metodología sobresale en previsibilidad, pero carece de flexibilidad.

1.3.1.2 Método de la ruta crítica (CPM, por sus siglas en inglés)

El método de la ruta crítica se desarrolló en la década de 1950 y se basa en el concepto que hay algunas tareas que no se pueden comenzar hasta que se haya

completado otra anterior. Cuando se encadenan juntas estas tareas dependientes de principio a fin, se traza su ruta crítica.

Identificar y centrar en esa ruta crítica permite a los gestores de proyectos priorizar y asignar recursos para realizar el trabajo más importante y reprogramar cualquier tarea de menor prioridad que pueda estar obstruyendo la capacidad del equipo. De esta manera, si es necesario realizar cambios en el cronograma del proyecto, se puede optimizar el proceso de trabajo del equipo sin retrasar los resultados finales.

1.3.2 Sistemas de tirar (pull system)

Por otro lado, el sistema Pull (tirar), limita la producción de bienes y servicios en función a una necesidad de la demanda. Cuando un bien o producto es determinado para su consumo o uso, se activan los mecanismos para reemplazarlo. Este sistema permite a las empresas reducir costos en producción e inventarios, eliminando el desperdicio de capas intermedias de producción por si acaso (stock de demanda innecesaria). así como estructurar los procedimientos de fabricación mediante el uso de carteles o tarjetas, las cuales ayudan a dividir el proceso en fases determinadas y ordenadas de forma secuencial. En el sistema Pull, el enfoque principal son los consumidores y sus necesidades.

En entornos dinámicos donde la demanda puede cambiar y donde los tiempos de adaptación de la producción de bienes o servicios son cortos, es claro que un método tradicional de gestión y control de proyecto puede no ser favorable a dichas adaptaciones.

En este sentido las metodologías ágiles promueven mecanismos de dinámica adaptativa, permitiendo ajustar los procesos de la producción de bienes y servicios con mayor eficiencia. la premisa de las metodologías ágiles es contar con un ciclo de vida iterativo e incremental y una gestión de la producción de bienes y servicios basada en la demanda mediante el esquema de gobernanza de tirar (de la demanda) conocido más frecuentemente por su denominación en inglés “*pull system*”.

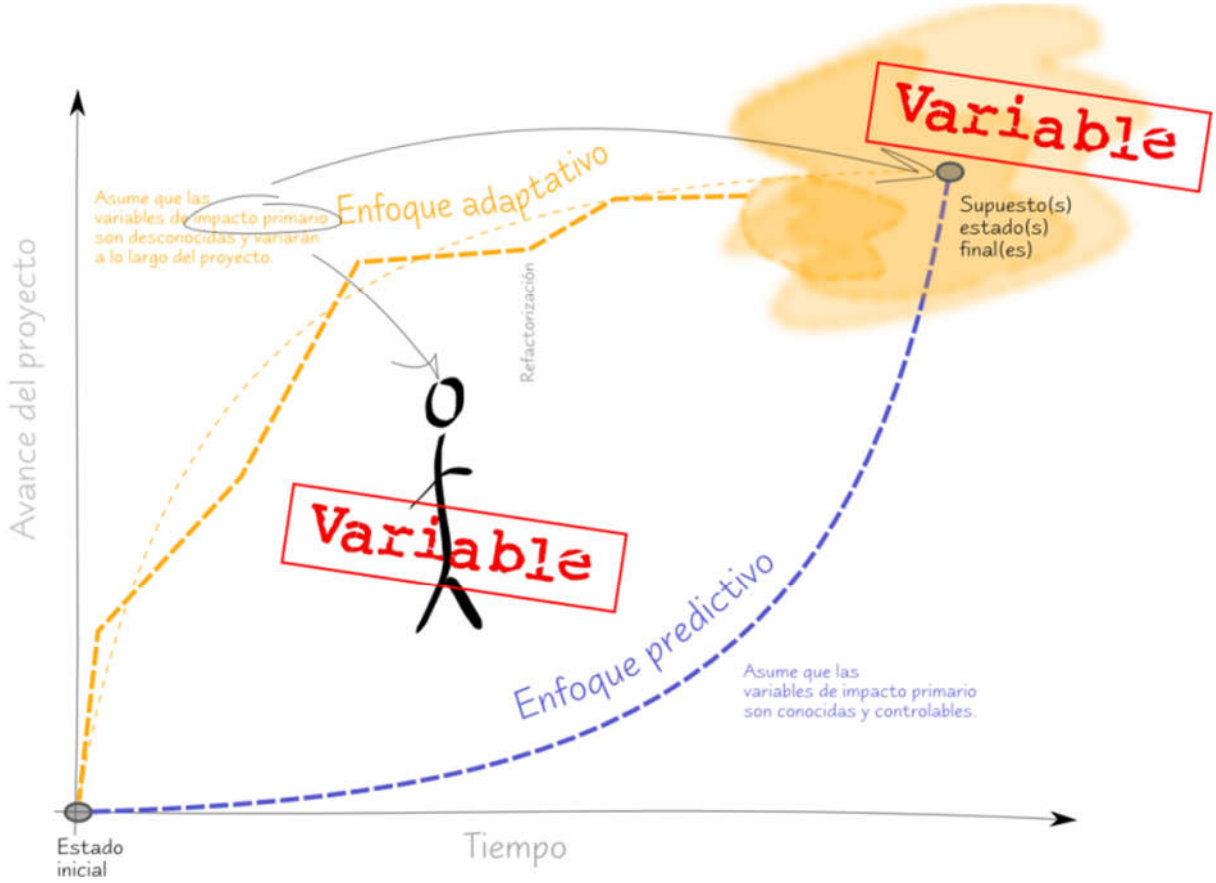

Figura 1.2 - Enfoque predictivo versus adaptativo (ágil).

En este sentido los sistemas adaptativos:

- Pretenden responder a niveles altos de cambio y la participación continua de los interesados.
- Difieren en que las iteraciones son muy rápidas y de duración y costos fijos (normalmente con una duración de 2 a 4 semanas).
- Generalmente ejecutan varios procesos en cada iteración.
- Al final de cada iteración el producto debe estar listo para su revisión por el cliente. (del prototipo al producto).
- Los representantes del patrocinador deben estar continuamente involucrados en el proyecto para proporcionar retroalimentación sobre los entregables.

Las metodologías ágiles en su ciclo de vida iterativo e incremental, basados en pull system aportados por la metodología LEAN, brindan una herramienta de gestión diferente en entornos innovadores altamente cambiantes. En definitiva, nos brinda una mirada a la forma de gestionar los riesgos de un proyecto innovador, donde las características principales del diseño de nuestro producto no existen aún y por lo tanto no son deterministas no predecibles.

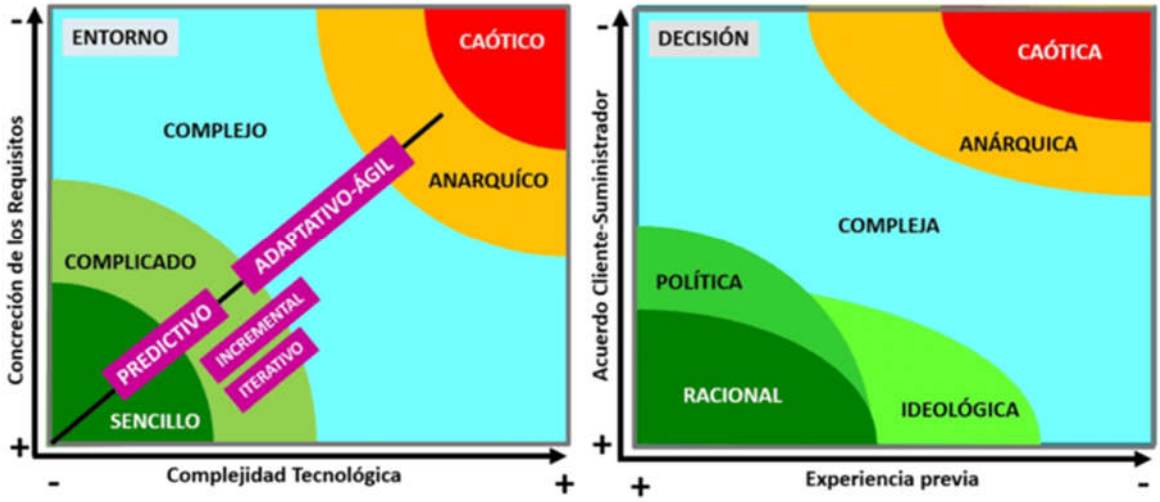

Figura 1.3: Espectro de complejidad del proceso.

1.3.3 Las 4 P's de la Ingeniería

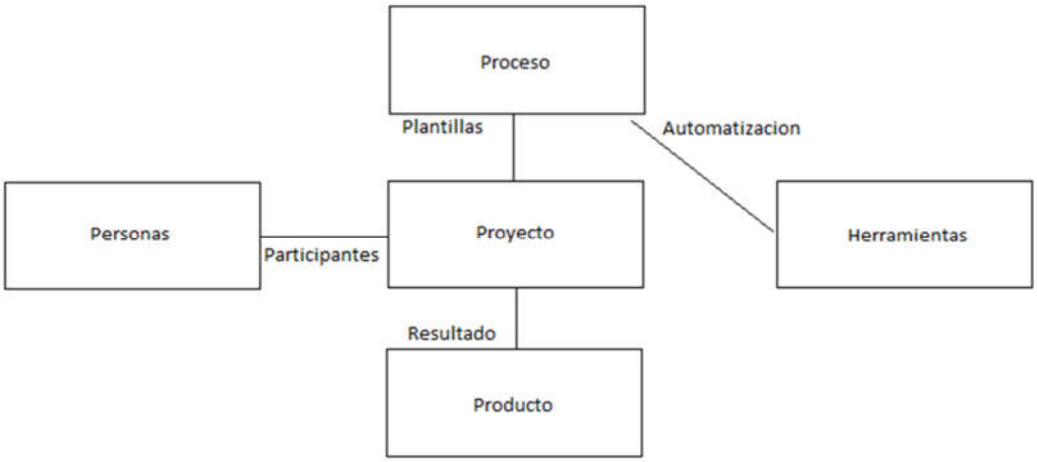

Figura 1.4: Esquema de las cuatro P's de la ingeniería.

- **Personas**: Los principales actores de un proyecto son las personas que lo integran: Ingenieros, arquitectos, constructores, usuarios, cliente, Interesados (stakeholders), etc. Las personas y la integración de las mismas con un mismo objetivo le dan sentido al proyecto.
- **Proyecto**: Elemento organizativo a través del cual se gestiona el desarrollo de un producto. El resultado de un proyecto es la versión de un bien o servicio concreto que es usado por usuarios y clientes.
- **Producto**: Artefactos que se crean durante la vida del proyecto: modelos, diseños, prototipos, documentación, bien, servicio, etc.
- **Proceso**: Conjunto completo de actividades necesarias para transformar los requisitos en un producto. Un proceso es una plantilla para crear proyectos.
- **Herramientas**. Software que se utiliza para automatizar las actividades definidas en el proceso.

1.4 ¿POR QUÉ CAMBIAR?

1.4.1 Leyendas fundacionales

Casi todas las actividades económicas y las disciplinas científicas tienen leyendas fundacionales. Por ejemplo, dicen que la física moderna nació cuando a Newton, que dormía bajo un manzano, le cayó una fruta en la cabeza. Los primeros filósofos, según nos cuenta la tradición, compartían sus conocimientos en el ágora de Atenas, un lugar idílico donde discípulos y maestros podían debatir abiertamente sus preocupaciones.

Y así pasa con todas las ciencias y actividades humanas. Poco importa si son ciertas o no esas leyendas. Probablemente tengan algo de verdad, y otra parte haya sido inventada. Pero lo cierto es que influyen en nuestra percepción actual.

Los nostálgicos aseguran que hubo un tiempo en que los clientes decían que aplicación deseaban, dejaban trabajando al equipo de desarrollo por un largo tiempo, y cuando se entregaba el producto desarrollado, se alegraban porque cumplía plenamente con sus expectativas. Es más, no sólo los clientes no necesitaban ver a los desarrolladores durante la ejecución del proyecto, sino que tampoco los desarrolladores necesitaban comunicarse con los clientes, ya que se contaba con una documentación tan exhaustiva y lograda sobre el producto a construir, que cualquier comunicación podía arruinar nuestro trabajo.

Se ha contado que, en aquellos tiempos, los clientes desconocían que el software era un producto modificable, aunque ya algunos se estaban preguntando qué significaba el prefijo “soft” en la palabra en cuestión. Por otro lado, ese era un problema, pues ¿quién querría modificar algo tan perfecto?

Los profesionales de desarrollo de software, debido a estas condiciones, podían planificar sus proyectos con seguridad, sabiendo que, una vez iniciado el proyecto, no había ningún peligro que el plan cambiase. La tranquilidad que brindaba previsión de todo futuro posible era muy reconfortante.

Además, existía –y existe– el dogma del mayor costo del cambio conforme el proyecto avanza, lo cual favorecía que cualquier modificación fuera vista como una amenaza. No es que el dogma sea absolutamente falso [Boehm 1981], pero más que inducirnos a rechazar pedidos del cliente, debería animarnos a buscar formas más sencillas de implementarlos.

En realidad, ninguno de los profesionales de sistemas informáticos conoció tiempos tan felices. Lo cierto es que, tanto si hubieran existido esos tiempos, como si se tratase de mitos de la historia del desarrollo de muchos productos, especialmente los innovadores, hoy se puede ver que sin duda las cosas no son así. Sin embargo, se trabajaba –y muchos casos todavía se hace– pensando que las cosas funcionaban de esa manera.

1.4.2 Errores de inmadurez

El desarrollo de productos de bienes y servicios innovadores es una disciplina joven. Hace poco más de 60 años que se programan computadoras, hace 40 se empezó a intentar sistematizar un proceso, y hace unos 30 se empezaron a introducir paradigmas que permitieran el manejo de la complejidad. Si se compara con las ingenierías tradicionales, que en general tienen varios siglos de antigüedad, no deberíamos asombrarnos que la nuestra sea considerada una actividad humana inmadura.

Prácticamente no existe ciencia que se haya aplicado en sus inicios sin cometer grandes errores. Los médicos de la antigüedad pretendían curar a sus enfermos con purgas y sangrías, que en ocasiones aceleraban la llegada de la muerte.

Algunos errores no han sido tan graves, sino que solo fueron simplificaciones fácilmente refutables. Pero aun estas simplificaciones, muchas veces han puesto en entredicho la disciplina en sus momentos fundacionales. La economía, dicen los especialistas, nació con el libro de Adam Smith, que explica su funcionamiento en un mundo ideal de competencia perfecta. Cuando se vio que esa simplificación era excesiva, se ha pretendido predicar la invalidez de toda la ciencia económica, por lo que fue necesario investigar en el marco de hipótesis más realistas para devolverle el prestigio que se estaba perdiendo.

Algo parecido pasó con la astronomía, al punto que el sistema de universo definido por Ptolomeo dos siglos antes de Cristo, seguía siendo utilizado en el siglo XV, hasta que Copérnico lo puso en entredicho, provocando gran contrariedad en el mundo académico de la época, que no estaba dispuesto a que se modifican las nociones esenciales de su ciencia.

Con el desarrollo de bienes y servicios también han pasado estas cosas. Se han cometido gruesos errores iniciales que han provocado el fracaso de muchos proyectos y el abandono de iniciativas de incorporación tecnológica. Más tarde, se han establecido algunos fundamentos basados en supuestos simplistas, o simplemente erróneos, que también han hecho perder prestigio a la disciplina.

Además, el ambiente académico es reacio a pensar en términos nuevos, a pesar de la corta historia del diseño innovador de bienes y servicios.

Por ejemplo:

- La idea de la división del trabajo en roles, basada en una visión tayloriana [Aitken 1960], llevó a que se crearan varias especialidades, y que cada una tuviera una labor totalmente diferenciada. Un analista funcional jamás se comparaba con un programador (¡Dios nos libre!), dado que estos últimos eran el equivalente de los obreros industriales en la producción de software, gente a la que no se le pedía pensar, sino solo trabajar. Para pensar estaban los analistas, que sabían lo que querían los clientes, cuáles eran sus necesidades y cómo lograr satisfacerlas

desde lo técnico. En definitiva, cada persona, según su rol, buscaba un objetivo diferente. A nadie se le ocurría pensar que en realidad se necesitaba un equipo multidisciplinario trabajando en pos de entregar valor al cliente.

- También se dio importancia excesiva a los documentos. Partiendo de la idea, tal vez razonable, de dejar registro de las decisiones que se toman en un proyecto mediante documentos, y siguiendo con la noción de que cada perfil se debía ceñir a sus tareas específicas, se llegó al extremo de que la única forma de comunicación en los proyectos eran los documentos. Los analistas especifican requerimientos, que era lo que podían ver los diseñadores, programadores y testers. Al fin y al cabo, ¿qué otras cosas debían conocer los programadores y testers de su proyecto? Los diseñadores especifican también el diseño con documentos, probablemente acompañados de complejos y detallados diagramas. Si esos diagramas luego debían ser mantenidos conforme cambiaba el software, no era un problema que se tuviese en cuenta.
- Los diseñadores se podían lucir con complicados diseños que mostraran su genialidad. Los programadores, toda vez que podían, introducían código que nadie sino ellos podrían entender. Y eso los hacía felices a todos, sin preguntarse si era bueno para el trabajo final. Porque el producto final también había pasado a ser secundario. Lo único importante eran los documentos, que definían qué se iba a desarrollar y cómo se iba a construir la aplicación. El software en sí mismo no era lo que importaba, y hasta se fantaseaba con la generación automática de código para enfatizar esta idea, pensando en un futuro feliz en el que la codificación humana fuera innecesaria. Pero llegó el día en que todo eso ya no se pudo sostener más.

1.4.3 Ya no es tiempo de seguir equivocándose

En los tiempos que corren, los clientes quieren todo lo antes posible. Ya no se puede, como otros períodos, decirle al cliente que vamos a trabajar durante un año y le mostraremos todo cuando esté terminado. Ellos quieren ver el software funcionando, lo antes posible, hacer observaciones sobre el mismo, saber cómo vamos en el proyecto y cuánto falta para el feliz día en que determinada funcionalidad esté lista.

Como además los clientes se han dado cuenta que el software maleable, que admite cambios aún durante su desarrollo, están constantemente buscando mejorar el producto. Si bien en un punto esto puede irritar, es parte ineludible de la naturaleza del software y, por lo tanto, debería serlo también del trabajo.

Precisamente, la necesidad de realizar cambios en los proyectos implica que los planes no puedan ser rígidos. Hay que entender que la planificación debe ser algo vivo, algo que se puede adaptar en tiempos, en alcance, e incluso por cuestiones derivadas de la incertidumbre sobre el diseño de la solución.

Otro inconveniente es el costo del cambio, que no por ser posible es barato. Adicionalmente, ese costo empeora si el diseño es complicado y el código poco legible.

Además, el software que desarrollamos hoy es tremendamente más complejo que el de décadas atrás. Esa complejidad no se puede encarar con procesos de desarrollo que solo funcionan razonablemente bien con pequeños programas. Lo mismo aplica a la calidad: un producto complejo necesita ser construido con la calidad en mente. Esta no puede ser algo que agregamos al producto a posteriori.

Por otro lado, los roles con objetivos contrapuestos pueden hacer perder el objetivo principal, que es entregar valor al cliente. Este valor no puede estar en los documentos, que son artefactos que sirven a los desarrolladores durante la construcción. ¿O alguno de nosotros, cuando quiere que le construyan una casa, está dispuesto a recibir sólo los planos a cambio?

Los clientes exigentes nos obligan a que tengamos criterios de aceptación, sin ambigüedad, que nos indiquen cuando hemos cumplido lo que se requiere y cuando se puede dar por concluido el desarrollo de una funcionalidad. Y esos criterios van a servir también para medir el avance, aun cuando los clientes no sean tan exigentes en ese punto.

Como consecuencia de todo lo dicho, el tiempo en que nuestro optimismo nos hacía pensar que los errores y descuidos podían ser tolerados, o incluso no ser percibidos, ha quedado atrás. Y eso exige pensar en otras maneras de desarrollar.

1.4.4 Buscando encauzar los proyectos de transformación digital

Siguiendo con la analogía de las demás ciencias y técnicas, no siempre es fácil romper con lo establecido para cambiar métodos y procedimientos. Hace falta ver los errores, luego evidenciarlos y finalmente proponer cambios mostrando sus ventajas. Muchos pretenderían mantener sus anteojeras, o a lo sumo realizar pequeños retoques a las visiones del pasado. No les fue fácil a Heisenberg y a Einstein poner en entredicho a la física clásica con proposiciones tan poco intuitivas que parecían descabelladas. Algo parecido debe haberle ocurrido a Lord Keynes cuando se atrevió a proponer cambios en la lógica de la macroeconomía. O a los médicos medievales que, de a poco, introducían cambios aprendidos en la España islámica.

Pero parece obvio que, si hay problemas y se están cometiendo errores, habría que tratar de enfrentarlos y corregirlos. En el desarrollo de software seguimos purgando pacientes hasta matarlos por deshidratación. Esto no es serio en una disciplina que, si bien es joven, ya cumplió 60 años.

Por eso es que a fines del siglo pasado se empezaron a sugerir mejoras que llevaron a los métodos ágiles.

En efecto, resulta totalmente insólito plantear que los proyectos de software se alarguen por la necesidad de generar documentación innecesaria, que cuando un

cliente pida un cambio estemos con la guardia en alto para que no arruine nuestro diseño, que los equipos sean en realidad compartimientos estancos de perfiles que los comunican mediante documentos, que los clientes debían conformarse con ver documentos de requerimientos y de diseño en vez de software funcionando, que los planes sean tallados en piedra.

Y si a fines del siglo pasado no podíamos seguir pretendiendo todo esto, mucho menos una vez que surgieron personas que, como hicieran otros tiempos y desde otras disciplinas Copérnico, Heisenberg, Einstein, Colón o Keynes, han puesto en entredicho unas cuantas verdades instaladas y han demostrado que el desarrollo de software puede ser más eficiente, más cooperativo, más transparente y menos rígido de lo que había sido hasta ese momento. Hoy, un poco más de una década más tarde, es todavía más inverosímil que haya gente que no perciba que los principios en los que nos habíamos basado tenían serios errores de concepción que hacían que los métodos derivados de los mismos fueran inadecuados en determinados contextos.

1.4.5 La naturaleza del software al rescate

Varias de las respuestas a los problemas planteados están en el propio software y en las peculiaridades de los proyectos de desarrollo del mismo.

Expliquemos:

- A menudo se dice que el software es maleable, es decir, que se puede adaptar durante su construcción, y aun una vez terminado. ¿Qué mejor, entonces, que usar esa maleabilidad para poder ofrecer alternativas a nuestros clientes? ¿Por qué resistirse tanto a los cambios que se saben posibles?
- Por otro lado, el software es particionable y, por lo tanto, se puede construir por etapas, de modo tal que cada una implique que a la salida de la misma tengamos un producto, parcial, pero con valor para el cliente. ¿Por qué no usar esta característica para darle visibilidad durante su construcción?
- Otra cuestión a la cual se le presta poca importancia es que el desarrollo de software implica la materialización de conocimiento en programas de computadora. ¿Por qué no usar las reuniones de captura de conocimiento para interactuar más con nuestros clientes y dentro del mismo equipo? ¿Por qué no aprovechar para socializar?
- Contrariamente a lo que sucede en otras disciplinas, el diseño y la construcción del producto no son procesos separados, uno único y el otro repetitivo, sino que se hacen en conjunto, con un único producto para cada diseño. Entonces, ¿por qué nos resistimos de manera tan apasionada a los cambios de diseño durante el desarrollo?
- El software es también susceptible de ser copiado en forma íntegra. ¿Por qué no aprovechamos esta característica para entregar productos de calidad?

- Si decimos que el software es extensible, ¿por qué nos resistimos a hablar de la evolución permanente de cada producto? Es cierto que, otras veces, el propio software conspira para solucionar los problemas, como ocurre con la visibilidad del producto. En efecto: Al ser un producto invisible por definición, es complicado saber, durante el desarrollo, cuánto se ha construido y cuánto queda por construir. Por esto se ha vuelto necesario definir técnicas y métricas específicas para el software. Esta falta de visibilidad del producto también hace difícil que el cliente sepa rápidamente si lo que se construye es lo que se esperaba. Por eso es que conviene definir criterios de aceptación, que también permiten que el propio equipo de desarrollo pueda conocer cuándo puede dar por terminada la construcción de determinada funcionalidad.
- Otro problema del software es su complejidad, muchas veces mencionada, pero poco comprendida cabalmente. En efecto, se habla con ligereza de sistemas medianos de, digamos cincuenta mil clases. ¿Somos conscientes que no estamos hablando del equivalente de una máquina de cincuenta mil piezas, sino de una con cincuenta mil tipos de piezas distintas?

1.4.6 El resto lo cubren las personas

Otro aspecto característico del software es que es un producto construido en su totalidad por personas. Por eso es que la mayor parte de los problemas y de las soluciones se deben ver más desde la sociología que desde la tecnología. Esto puede ser duro para la mayoría de las personas involucradas, e incluso para quienes se desempeñan en roles gerenciales, que han sido formadas más en tecnología que en disciplinas humanísticas. Sin embargo, se debe pensar que las habilidades sociológicas, aunque se carece de nociones formales, están en el ADN y que se vienen practicando desde que se nace. Por ejemplo:

- Durante milenios, se ha evolucionado mediante una comunicación cara a cara para poder entendernos y superar los malos entendidos, los equívocos, y todo lo que las personas podemos expresar mirando al interlocutor. Por eso es importante no olvidar que esa es la manera más natural de comunicación entre humanos.
- Nuestra naturaleza social también hace que seamos propensos a formar equipos exitosos, que se sienten orgullosos de su creatividad y sus éxitos y, sobre todo, si los problemas resueltos fueron muy complejos. Estos equipos tienden a autoorganizarse, sin necesidad de impulsos externos, y a trabajar mejor cuanta más sinergia logren.
- Por último, contrariamente a lo que dicen algunos refranes, se aprende de los errores, y somos los únicos animales capaces de reflexionar sobre los mismos, para no volverlos a cometer. Y como contrapartida, tendemos a analizar y a repetir conductas que hayan llevado a resultados exitosos. Ahora bien,

dificultades también hay. Aunque en este caso las dificultades no suelen venir de las personas que trabajan, sino de prejuicios típicos del mundo corporativo, que atentan contra la naturaleza humana.

- Una de las nociones que más conspira contra la autoestima de las personas, y que afecta la calidad de lo producido por esta misma razón, es el considerar a las personas como “recursos”, pretendiendo que, como ocurre con otros recursos que afectan nuestro proyecto, se trata de piezas intercambiables. Eso lo pueden provocar tanto los gerentes de mentalidad industrial, como las metodologías que ponen el foco en que no importan tanto las personas como el seguimiento de un método supuestamente infalible.
- Asimismo, cuando por cuestiones de costos o cronograma, se impulsa a las personas a construir un producto de baja calidad, también se afecta la autoestima de todo el equipo de trabajo. En realidad, todo cronograma ajustado de manera irreal es percibido como una falta de respeto por los equipos experimentados. Si bien en algunas ocasiones se le puede pedir un compromiso a un equipo ante una necesidad puntual, esto funciona en la medida en que el equipo haga suyo este compromiso, y que realmente sea una excepción definida y acordada, no la regla. Las presiones ejercidas sobre las personas para que produzcan más no funcionan, simplemente porque el hecho de que se pueda obligar a alguien a estar más tiempo sentado en una silla, no implica que de esa manera se logre creatividad o productividad.
- Otro aspecto negativo es un entorno laboral que conspire contra la producción. Sillas y mesas incómodas, falta de luz, infraestructura inadecuada, no contar con el software necesario para desarrollar, restricciones de acceso a Internet que impiden investigar, son todos problemas que acarrean frustración y cuestan bastante más de lo que parece. Por eso, un buen gerente debe enfocarse en permitir que las personas trabajen cómodas y eficientemente más que en “hacer que trabajen”.
- Y, por último, hay que confiar en las personas y en los equipos que las mismas integran. Si un gerente vive obsesionado por desarmar equipos que disfrutan del trabajo en conjunto, por temor a que formen grupos elitistas, terminará con trabajadores poco motivados y menos productivos. Si solo considera que están trabajando cuando los ven escribiendo código, y no cuando piensan, conversan con sus colegas, investigan soluciones en la web o se reúnen espontáneamente, solo va a lograr una actitud defensiva y poco productiva.

1.4.7 Iterativo por naturaleza

En los últimos cuarenta años, no ha habido ningún modelo de proceso de desarrollo de software exitoso que no sea iterativo. Desde el proceso en espiral hasta

el unificado, todos los procesos recomendados tanto desde la academia como la industria asumen una estructura de aproximaciones sucesivas para su desarrollo.

Sin embargo, como se ha visto anteriormente (Sistemas de empujar), son comunes múltiples visiones (el modelo en cascada) y prácticas (la planificación de largo plazo basada en tareas) que asumen un proceso en el que, en mayor o menor medida, se sabe todo lo importante al principio del proyecto.

Muchas veces hemos escuchado enunciados con la clásica estructura de recomendación seguida de claudicación:

“Hay que hacer las pruebas en paralelo con el diseño, pero no podemos enseñarlo así porque es confuso”; o la otra de:

“Claro que el análisis puede ser influenciado por el diseño, pero de todas maneras siempre hay que hacer todo lo que pide el cliente, por eso se llaman requerimientos”.

En estos ejemplos, la vocación parece clara pero falta el coraje o la percepción profunda o la técnica para llevarla a buen término. En cada uno, se deja para después algo que debería hacerse antes y, por lo tanto, se asume en lugar de aprender o discutir.

Parte del éxito de los métodos ágiles está en abrazar esa perspectiva de persona mirando al abismo, y proponer técnicas específicas para lidiar con la complejidad.

1.4.8 El desarrollo de software como proceso iterativo

La principal confusión reinante alrededor del proceso de desarrollo consiste, como vimos en el capítulo anterior, en una mirada basada en actividades disjuntas (análisis, diseño, implementación, pruebas, etc.) que son llevadas a cabo por gente que se comunica poco y mal, principalmente a través de documentos. Esta confusión hace fácil identificar implícitamente el proceso con el modelo en cascada, es decir, con una secuencia de actividades (por ejemplo, primero implementación y después pruebas). Aun cuando no sea explícita, esta preconcepción tiende a dificultar la aplicación de un proceso iterativo.

Ahora bien ¿Qué es un proceso iterativo? Es un proceso de aproximaciones sucesivas al resultado final, que nos permite ir ajustando tanto el producto como el proceso para maximizar el valor del resultado. Un ejemplo es la escritura de estos textos: Se han producido varios borradores hasta llegar al resultado final. En concreto, es un proceso donde las actividades se repiten en cada iteración, permitiendo obtener y analizar los resultados de esas múltiples actividades y utilizarlos para nutrir a las demás (por ejemplo, las pruebas se ejecutan en cada iteración sobre lo construido).

Lo anterior no quiere decir que todas las actividades tengan que realizarse en todas las iteraciones, el foco puede variar a medida que transcurre el proyecto, por ejemplo, estar en un momento en la puesta en producción y en otro en la construcción.

Un Proceso Iterativo nos ayuda a:

- Innovar: se puede usar una o varias iteraciones para explorar alternativas. Si las alternativas son descartadas, el proceso nos permite limitar el esfuerzo dedicado a ellas mediante la asignación de un número acotado de iteraciones a esa exploración. Si las alternativas son seleccionadas, pueden refinarse en futuras iteraciones.
- Minimizar el costo de los errores: como cada iteración aborda solo una parte del problema completo, si se producen equivocaciones, no se está arriesgando todo el proyecto.
- **Maximizar las oportunidades de mejora**: un proceso iterativo provee múltiples oportunidades concretas (por ejemplo, retrospectiva al final de cada iteración) para la reflexión tendiente a la mejora. Inspeccionar y adaptar.
- **Imprimir un ritmo**: con iteraciones de duración fija, los equipos realizan sus actividades a intervalos regulares, facilitando la asimilación de las prácticas (por su repetición disciplinada). El ritmo también fomenta el mantener un nivel de esfuerzo sustentable y evita el agotamiento.
- **Maximizar las oportunidades de control**: al final de cada iteración podemos evaluar métricas y validar el producto para determinar el progreso y la alineación en relación a los objetivos del proyecto.

1.4.9 Un proceso iterativo por incrementos

Para ser efectivo, un proceso iterativo depende de que cada iteración produzca resultados concretos que den sensación de avance. Sin esa sensación será difícil revisar la planificación y garantizar que el proyecto en su conjunto vaya en la dirección correcta (es decir, que vaya a cumplir con sus objetivos).

Para ayudar en esa tarea, es común dividir el producto en incrementos, particiones que pueden desarrollarse en distintas iteraciones y permiten planificar el trabajo para cumplir con el alcance completo mediante un conjunto de iteraciones. En este modelo, cada iteración produce un nuevo incremento o refina uno anterior, de manera tal que al final todos los incrementos han sido construidos mediante sucesivos refinamientos.

1.4.9.1 No estamos solos

Este modelo de proceso iterativo y por incrementos no es una característica exclusiva del desarrollo de software. Como muestran Rob Austin y Lee Devin en su libro Artful Making [Austin 2003], tanto en el teatro como en otras formas de arte, en el diseño de estrategia, y otros trabajos creativos, los procesos exitosos son iterativos por la naturaleza del trabajo. Los autores describen las siguientes condiciones de aplicabilidad de esta forma de trabajo:

- **Necesidad de innovación**: cuando el producto es original o por lo menos lo es en el contexto actual.
- **Repetición confiable**: el proceso de trabajo es repetible, es decir, contamos con la capacidad y disciplina para trabajar iterativamente.
- **Bajo costo de iteración**: el costo de iteración se define como la suma de:
  - **Costo de reconfiguración**: el costo de modificarlo que hemos realizado en iteraciones anteriores, o de cambiar el proceso.
  - **Costo de exploración**: el costo de explorar alternativas que no son incluidas finalmente en el producto final, porque encuentra una alternativa mejor o simplemente porque no forman un todo armónico con el resto.

Estas condiciones son típicas de situaciones en las que el producto que se obtiene es intangible (por lo menos parcialmente), y requiere el trabajo colaborativo de un grupo de personas.

El modelo de Artful Making nos permite caracterizar nuestros procesos de desarrollo para decidir si corresponde o no un proceso iterativo. La filosofía ágil y muchos otros conciben al desarrollo de software como un proceso eminentemente innovador. Por innovador entendemos:

- Que los problemas a los que nos enfrentamos son radicalmente nuevos cada vez. Aunque parezca que muchos proyectos tienen cosas en común, lo que tienen de particular e interesante tiende a ser siempre más de lo esperado.
- Que la forma de trabajo apropiada para lidiar con esos problemas no puede definirse en detalle de antemano. Dicho de otra forma, nuestros procesos y prácticas de trabajo deben adaptarse a la realidad específica de cada proyecto.

Si no hubiera necesidad de innovación, bastaría reusar software existente. Lo que ocurre con el reuso es que normalmente los requerimientos suenan a “quiero algo parecido pero distinto”. Las expectativas sobre el reuso tienden a ser altas, ya que implica no tener que desarrollar software, pero en la mayoría de los casos requiere extensión y adaptación. Además, el reuso en sí implica desafíos análogos o mayores en complejidad a los del desarrollo de software a medida.

En cuanto a la repetición confiable, es uno de los desafíos metodológicos fundamentales de ingeniería software, como tema fundamental de la madurez que discutimos antes. En ese aspecto, los métodos ágiles promueven una práctica y disciplina cotidiana (por ejemplo, reuniones diarias, validación y revisiones periódicas a intervalos regulares, etc.) que soportan tanto la repetición sustentable como la mejora continua. Nuevamente inspeccionar y adaptar.

Finalmente, dada la propia naturaleza intangible del software (como opuesto al hardware, que tiene costos de reconfiguración mayores), y de la mano de ciertas prácticas específicas (por ejemplo, integración continua), en el desarrollo es posible mantener bajo el costo de reconfiguración.

1.4.10 La mejora como un proceso empírico

Dadas las características esenciales del software (complejo, intangible, ajustado al uso y cambiante [Brooks 1975]), y entendiéndolo como información empaquetada, el proceso de desarrollo es un proceso de aprendizaje continuo, tanto sobre los requerimientos y el contexto del sistema, como sobre el diseño y el proceso de desarrollo. Desde esa perspectiva, todo proceso de desarrollo debe implicar la mejora continua para garantizar mínimamente que se logren los resultados esperados, porque si no aprendemos lo suficiente sobre el contexto y el proceso, es poco probable que logremos pasar la prueba final de que el software sirva para lo que se lo construye.

Dicho de otra forma, si no se realiza un esfuerzo por mejorar, es muy poco probable que se acierte desde el principio o que sepamos lo suficiente como para lograr nuestros objetivos.

La filosofía ágil asume esta mejora como un proceso empírico, es decir, basado en la exploración y la experimentación. En un equipo ágil, todos los individuos son solidariamente responsables por experimentar, evaluar y adaptar el proceso y las prácticas de desarrollo en forma iterativa, para lograr los objetivos del proyecto.

Los métodos ágiles promueven la mejora mediante algunas prácticas propias de un proceso iterativo:

- **Retrospectivas**: al final de cada iteración se evalúa el proceso y se establece un compromiso de mejora.
- **Revisiones**: al final de cada iteración se evalúa el producto y se determinan los refinamientos apropiados.
- **Incrementos**: el producto se realiza en partes pequeñas que pueden ser validadas tempranamente, reduciendo el impacto de los errores.
- **Planificación estratégica**: se revisan las prioridades, avance y resultados del proyecto. Basada en hitos como iteraciones completadas y entrega subconjuntos de funcionalidad.
- **Planificación táctica**: se organizan las tareas de la iteración inmediata subsiguiente.

1.5 PLANIFICACIÓN, CONTROL Y SEGUIMIENTO DE PROYECTOS ÁGILES

1.5.1 Introducción. Algo de información de base de procesos

Para comprender la importancia de la gestión de los proyectos en general y de los proyectos ágiles en particular, es necesario comprender algunas cuestiones planteadas por los modelos de certificación y garantía de la calidad de desarrollo de productos de software.

La aproximación y uso del marco teórico en este sentido se propone tomando la mirada propuesta por el Software Engineering Institute (SEI), uno de los organismos de certificación más importantes del mundo y realizando un mapa sobre las técnicas, prácticas y herramientas ágiles para garantizar el desarrollo de los proyectos en el marco de un modelo de empujar (pull system).

1.5.2 Capability Maturity Model Integration (CMMi)

Los modelos CMMI® (Capability Maturity Model® Integration) son colecciones de buenas prácticas que ayudan a las organizaciones a mejorar sus procesos. Estos modelos son desarrollados por equipos de producto con miembros procedentes de la industria, del gobierno y del Software Engineering Institute (SEI).

Este modelo, denominado CMMI para Desarrollo (CMMI-DEV), proporciona un conjunto completo e integrado de guías para desarrollar productos y servicios informáticos.

Un nivel de madurez es una plataforma evolutiva definida para la mejora de procesos de la organización. Cada nivel de madurez desarrolla un subconjunto importante de procesos de la organización, preparándose para pasar al siguiente nivel de madurez. Los niveles de madurez se miden mediante el logro de las metas específicas y genéricas asociadas con cada conjunto predefinido de áreas de procesos.

Figura 1.5: Los 5 niveles de maduración planteados por CMMi.

Los cinco niveles de madurez, cada uno de ellos una base para la mejora de proceso en curso, se denominan por los números del 1 al 5:

1.5.2.1 Nivel De Madurez 1: Inicial

Los procesos son generalmente ad hoc y caóticos. La organización generalmente no proporciona un entorno estable para dar soporte a los procesos. El éxito en estas organizaciones depende de la competencia y la heroicidad del personal de la organización y no del uso de procesos probados. A pesar de este caos, las organizaciones de nivel de madurez 1 a menudo producen productos y servicios que funcionan, pero, sin embargo, exceden con frecuencia el presupuesto y los plazos planificados. Es probable que a mediano o largo plazo también dejen de contar con productos que sigan funcionando.

Las organizaciones de nivel de madurez 1 se caracterizan por una tendencia a comprometerse en exceso, a abandonar sus procesos en momentos de crisis y a no ser capaces de repetir sus éxitos.

1.5.2.2 Nivel De Madurez 2: Gestionado

Se garantiza que, en los proyectos, los procesos se planifican y ejecutan de acuerdo con políticas establecidas; los proyectos emplean personal calificado que dispone de recursos adecuados para producir resultados controlados; se involucra a las partes interesadas relevantes; se monitorean, controlan y revisan; y se evalúan en

cuanto a la adherencia a sus descripciones de proceso. La disciplina de proceso reflejada por el nivel de madurez 2 ayuda a asegurar que las prácticas existentes se mantienen durante periodos bajo presión. Cuando estas prácticas están desplegadas, los proyectos se realizan y gestionan de acuerdo a sus planes y planificaciones.

El estado de lo producido en el trabajo es visible para la dirección en puntos definidos (p. ej., en los hitos principales y al finalizar las tareas principales). Se establecen compromisos entre las partes interesadas relevantes y se modifican, según sea necesario. El trabajo se controla de forma apropiada. Se satisface con las descripciones del proceso, estándares y procedimientos especificados.

1.5.2.3 Nivel De Madurez 3: Definido

Los procesos están bien caracterizados y comprendidos, y se describen en estándares, procedimientos, herramientas y métodos. El conjunto de procesos estándar de la organización, que es la base del nivel de madurez 3, se establece y se mejora a lo largo del tiempo. Estos procesos estándar se utilizan para establecer la integridad en toda la organización. Los proyectos establecen sus procesos definidos adaptando el conjunto de procesos estándar de la organización de acuerdo a las guías de adaptación.

Una diferencia crítica entre los niveles de madurez 2 y 3 es el alcance de los estándares, descripciones de proceso y procedimientos.

Un proceso definido establece claramente el propósito, entradas, criterios de entrada, actividades, roles, medidas, etapas de verificación, salidas y criterios de salida. Los procesos se gestionan proactivamente a través de la comprensión de las interrelaciones de las actividades del proceso, de las medidas detalladas del proceso, de sus productos de trabajo y de sus servicios.

1.5.2.4 Nivel De Madurez 4: Gestionado Cuantitativamente

La organización y los proyectos establecen objetivos cuantitativos para la calidad y el rendimiento del proceso, y se utilizan como criterios en la gestión de los proyectos. Los objetivos cuantitativos se basan en las necesidades del cliente, usuarios finales, organización e implementadores del proceso. La calidad y el rendimiento del proceso se interpretan en términos estadísticos y se gestionan durante la vida de los proyectos.

Una diferencia crítica entre los niveles de madurez 3 y 4 es la **predictibilidad** del rendimiento del proceso. En el nivel 4, el rendimiento de los proyectos y de los subprocesos seleccionados se controla utilizando técnicas estadísticas y otras técnicas cuantitativas, y las predicciones se basan, en parte, en el análisis estadístico de los datos detallados del proceso.

1.5.2.5 Nivel De Madurez 5: En Optimización

Una organización mejora continuamente sus procesos basándose en una comprensión cuantitativa de sus objetivos de negocio y necesidades de rendimiento. La organización utiliza un enfoque cuantitativo para comprender la variación inherente en el proceso y las causas de los resultados del proceso.

Se centra en mejorar continuamente el rendimiento de los procesos mediante mejoras incrementales e innovadoras de proceso y de tecnología. Los objetivos de calidad y de rendimiento del proceso de la organización se establecen, se modifican continuamente para reflejar cambios en los objetivos del negocio y en el rendimiento de la organización, y se utilizan como criterios para gestionar la mejora de procesos. Los efectos de las mejoras de procesos desplegadas se miden utilizando técnicas estadísticas y otras técnicas cuantitativas, y se comparan con los objetivos de calidad y de rendimiento del proceso.

1.5.3 Áreas De Proceso (Claves)

1.5.3.1 Nivel 2 De Madurez

Área de Proceso	***Categoría***	***Madurez***
Gestión de Requisitos (REQM)	Gestión de proyectos	2
Planificación del Proyecto (PP)	Gestión de proyectos	2
Monitorización y Control del Proyecto (PMC)	Gestión de proyectos	2
Gestión de Acuerdos con Proveedores (SAM)	Gestión de proyectos	2
Gestión de Configuración (CM)	Soporte	2
Medición y Análisis (MA)	Soporte	2
Aseguramiento de la Calidad del Proceso y del Producto (PPQA)	Soporte	2

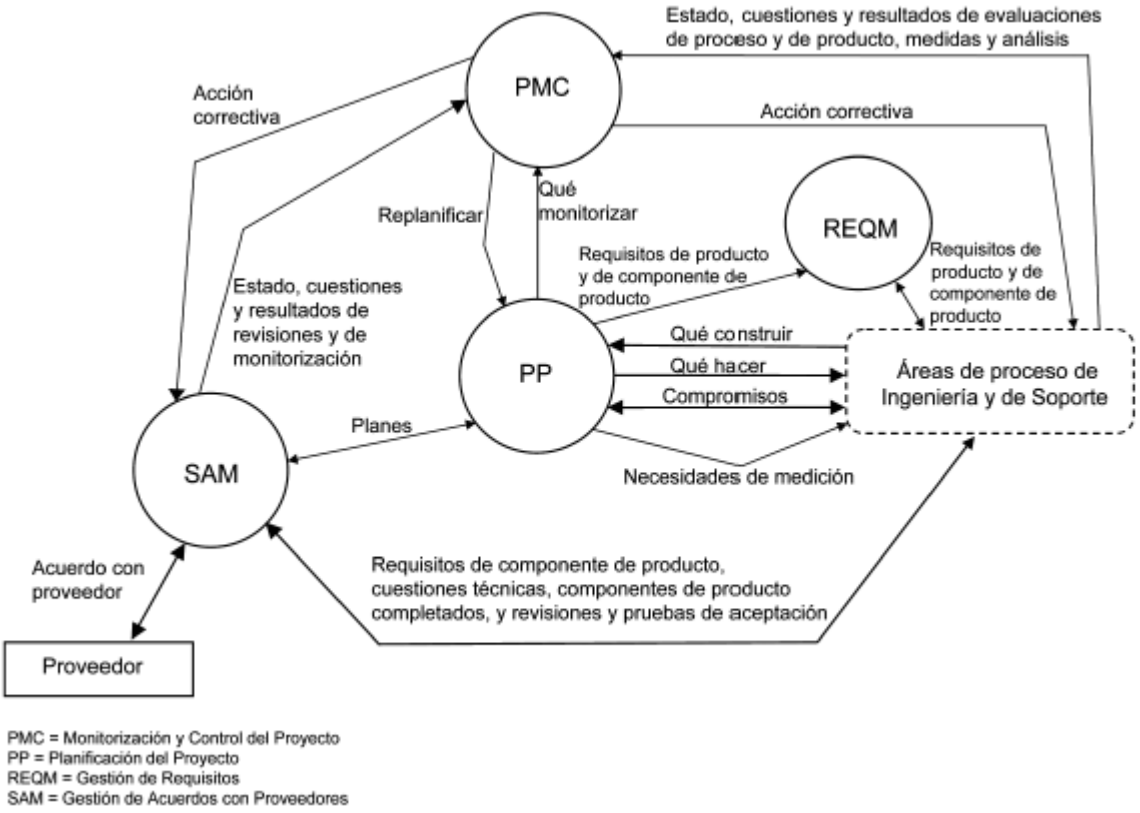

Figura 1.6 - Flujo de proceso de gestión de proyectos planteados por CMMi nivel 2.

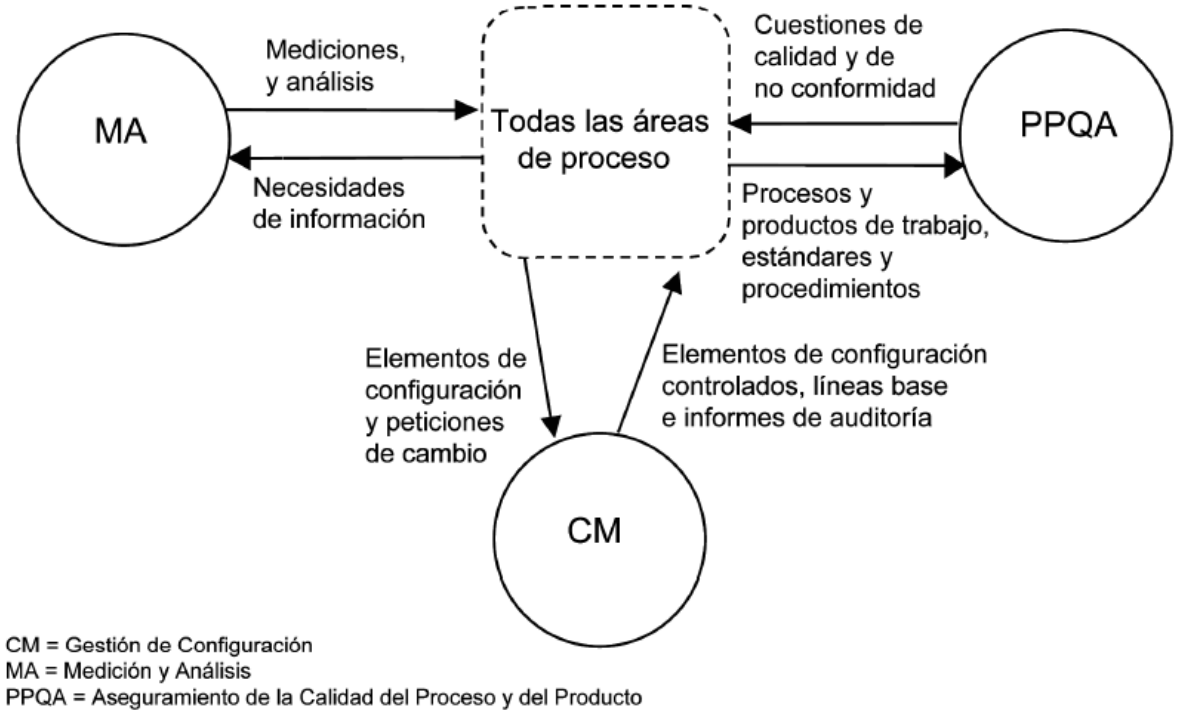

Figura 1.7 - Flujo soporte planteados por CMMi nivel 2.

1.5.3.2 Nivel 3 De Madurez

Área de Proceso	***Categoría***	***Madurez***
Gestión Integrada del Proyecto (IPM)	Gestión de proyectos	3
Gestión de Riesgos (RSKM)	Gestión de proyectos	3
Definición de Procesos de la Organización (OPD)	Gestión de procesos	3
Enfoque en Procesos de la Organización (OPF)	Gestión de procesos	3
Formación en la Organización (OT)	Gestión de procesos	3
Análisis de Decisiones y Resolución (DAR)	Soporte	3
Desarrollo de Requisitos (RD)	Ingeniería	3
Solución Técnica (TS)	Ingeniería	3
Validación (VAL)	Ingeniería	3
Verificación (VER)	Ingeniería	3
Integración del Producto (PI)	Ingeniería	3

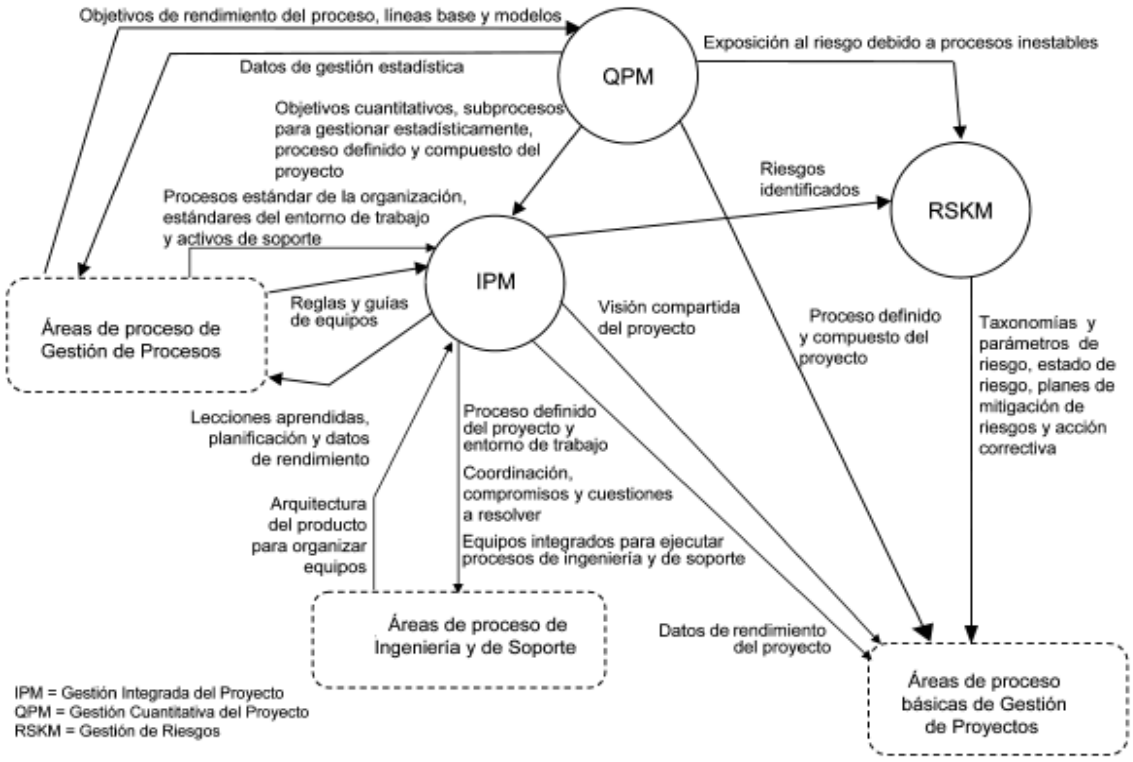

Figura 1.8 - Flujo de gestión de proyectos de nivel 3 planteados por CMMi

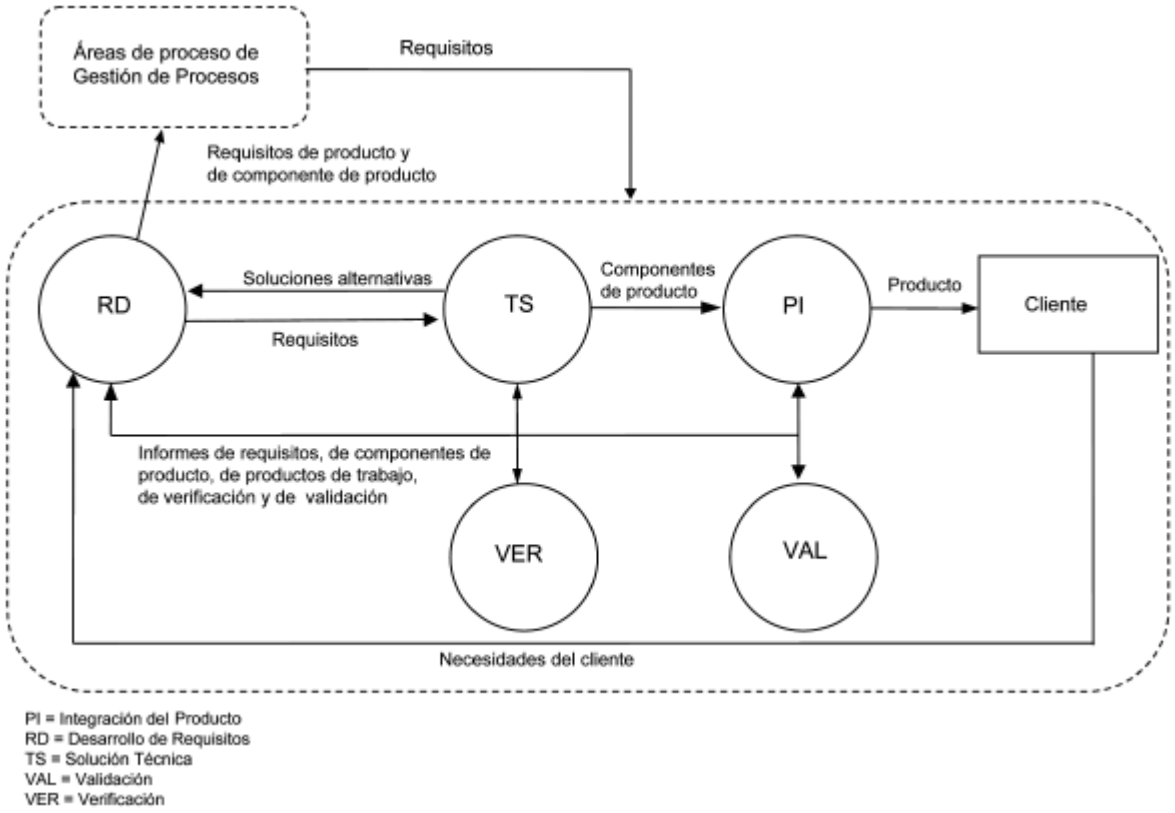

Figura 1.9 - Flujo de actividades de Ingeniería planteados por CMMi para nivel 3.

1.5.4 Puesta en valor del negocio

1.5.4.1 Anteproyecto Ágil

Hay algunas preguntas importantes que deben abordarse antes del inicio de cualquier proyecto, como "¿Por qué estamos haciendo este proyecto?" y "¿Deberíamos hacer este proyecto?" La iniciación del proyecto es la fase en la que se responden estas preguntas, pero por lo general se ha trabajado antes.

El formato de un documento de resumen del proyecto puede ser una visión del proyecto, en parte de una hoja de ruta del producto o una declaración de la misión del proyecto, pero la clave es saber por qué se está gastando este dinero en esta actividad.

Marty Cagan del Grupo de Productos de Silicon Valley sugiere que se proporcionen buenas respuestas a nueve preguntas para saber si se está buscando una buena oportunidad. Las nueve preguntas de la Evaluación de oportunidades son:

- 1) ¿Exactamente qué problema resolverá esto? (propuesta de valor).
- 2) ¿Para quién solucionamos ese problema? (mercado objetivo).
- 3) ¿Cómo vamos a medir el éxito? (métricas comerciales).
- 4) ¿Qué alternativas existen? (panorama competitivo).

- 5) ¿Por qué estamos mejor preparados para perseguir esto? (nuestro diferenciador).
- 6) ¿Por qué ahora? (ventana de mercado).
- 7) ¿Cómo implementaremos esto? (estrategia de despliegue suave).
- 8) ¿Cuál es el costo estimado preliminar? (pequeño mediano Grande).
- 9) ¿Qué factores son críticos para el éxito? (requisitos de la solución).

El grupo de productos de Silicon Valley recomienda un prototipo de alta fidelidad en lugar de utilizar documentos para describir el sistema de destino. Si se aprueba el Resumen del proyecto, entonces es importante invertir algo de tiempo en armar un prototipo que permita contarnos un panorama del sistema.

1.5.4.2 Visión del negocio (Envisioning)

Lo que el negocio plantea como necesidad, donde en base a la misma se define lo que la solución tecnológica aportaría a dicha necesidad a través de una estrategia de producto y un conjunto definido de requerimientos o características de alto nivel que aportan valor al negocio.

Envisioning es la actividad que captura la esencia de un potencial producto y crea un plan aproximado para la creación de dicho producto. La visualización comienza con la creación de una visión, seguida de la creación de una pila de producto de alto nivel (high level product backlog) preferentemente articulado mediante una hoja de ruta del producto (product roadmap).

El documento de visión debe contar con las siguientes secciones:

Descripción del negocio

Contenido (algunos párrafos) que describa el negocio puntual en el que actuará el sistema bajo construcción.

Problemas, síntomas, déficits del negocio

Contenido (algunos párrafos) con los principales problemas del funcionamiento actual que justifican un cambio en la organización para mejorar. Listar las principales motivaciones del cambio.

Visión

La visión del producto proporciona una descripción clara de las áreas en las que las partes interesadas (stakeholders), como los usuarios y los clientes, obtienen valor (del negocio).

Describe cómo el planteo y la estrategia del producto tecnológico a construir aportarán y mejorarán dichos valores.

Pila de productos de alto nivel (high level product backlog)

Lista principal de las historias de usuario a nivel de épicas que describen las principales características planteadas en la visión.

Las épicas son historias de usuarios de alto nivel y se formulan de forma similar (yo como “rol” quiero “un aporte para mi negocio” para “meta, necesidad o justificación de la necesidad”).

Hoja de ruta del producto

También conocida mediante la herramienta “visual story mapping”.

Una hoja de ruta del producto comunica la naturaleza incremental de cómo se construirá y entregará el producto a lo largo del tiempo, junto con los factores importantes que impulsan cada lanzamiento individual (DoD definition of done de cada incremento).

1.5.4.3 Model Canvas (Plan de negocios)

El llamado Modelo Canvas o método canvas fue desarrollado en 2011 por Alexander Osterwalder e Yves Pigneur. Es una herramienta para definir y crear modelos de negocio innovadores que simplifican 4 grandes áreas: clientes, oferta, infraestructura y viabilidad económica en un recuadro con 9 divisiones. Este modelo, se integra dentro de la metodología LEAN-startup, que se basa en encontrar y fomentar nuevas formas de crear, entregar y capturar valor para el cliente mediante el aprendizaje validado.

A continuación, y a modo de guía se expresan las secciones que debe contener el modelo:

Propuesta de valor

Es la pieza clave de todo el modelo de negocio. La propuesta de valor o ventaja competitiva es el motivo por el cual nuestro cliente va a comprar nuestro desarrollo y no otro. Aquí se incluye lo que hace diferente e innovador nuestro producto/servicio.

Se puede innovar en diferentes aspectos como en el modelo de ingresos, alianzas empresariales, procesos productivos, entrega del producto/servicio, marca, innovación tecnológica, etc.

Debemos dar respuesta a:

- ¿Qué valor estamos entregando a nuestros clientes?
- ¿Cuál de los problemas de nuestro cliente vamos a ayudar a resolver?
- ¿Qué paquete de productos o servicios ofrecemos a cada segmento de cliente?
- ¿Cuál es la necesidad que satisfacemos?
- ¿Qué tipo de producto ofrecemos?

Se deben considerar aspectos cómo:

- Novedad.

- Rendimiento.
- Personalización.
- Hacer el trabajo.
- Diseño.
- Marca/estatus.
- Precio.
- Reducción de costos.
- Reducción de riesgo.
- Accesibilidad.
- Conveniencia.
- Usabilidad.

Segmentos de cliente

Detectar las necesidades del mercado, del cliente. El foco siempre es el cliente y se debe orientar el producto a sus necesidades y deseos.

Importante: El cliente no es el stakeholder que nos contrata la realización del producto sino el cliente que va a usar el producto (usuario final). Pueden ser los mismos, pero no son lo mismo en el rol.

Para poder identificar al cliente debemos ponernos en su piel (empatizar) y analizar qué es lo que piensa, siente, ve, escucha, cuáles son sus problemas y los beneficios que le puede aportar nuestro producto/servicio. Esta información surge de lo ya realizado en Visión del negocio (Envisioning).

Debemos dar respuesta a:

- ¿Para quién estamos creando valor?
- ¿Quiénes son nuestros clientes más importantes?

Se deben considerar aspectos cómo:

- mercado de masas.
- mercado de nicho.
- segmentado.
- diversificado.
- plataforma multilateral.

Canales

Una vez definidos los clientes y la propuesta de valor que les ofrecemos, tenemos que llegar a ellos. Si no nos conocen, no nos van a comprar. Aquí vamos a definir los canales de distribución del producto o servicio.

Debemos dar respuesta a:

- ¿Con qué canales podemos llegar a nuestros clientes?
- ¿A través de qué canales quieren ser contactados nuestros segmentos de cliente?
- ¿Cómo les contactamos ahora?

- ¿Cómo están integrados nuestros canales?
- ¿Cuáles funcionan mejor?
- ¿Cuáles son más eficientes en relación a el costo/beneficio? ¿Qué canales funcionan mejor? ¿Cuáles de estos canales son los más rentables?
  Se deben considerar aspectos cómo:

- **Conciencia**: ¿cómo conseguimos conciencia sobre los productos y servicios de nuestra empresa?

- **Evaluación**: ¿cómo ayudamos a nuestros clientes a evaluar la propuesta de valor de nuestra empresa?

- **Compra**: ¿cómo permitimos a nuestros clientes comprar productos y servicios específicos?

- **Entrega**: ¿cómo llevamos la propuesta de valor a nuestros clientes?

- **Post-venta**: ¿cómo proporcionamos soporte postventa?
-

Relación con los clientes

Debemos comunicarnos correctamente con nuestros clientes y estar pendiente de ellos. Ellos son nuestro eje central, por lo que saber definir la relación que vamos a tener con cada segmento de clientes, es fundamental para el éxito de un negocio.

Debemos dar respuesta a:

- ¿Cuál es la relación que tenemos con cada uno de nuestros segmentos de clientes?
- ¿Qué tipo de relación espera que establezcamos y mantengamos en cada uno de nuestros segmentos de cliente?
- ¿Cuáles hemos establecido?
- ¿Cómo están integrados con nuestro modelo de negocio?
- ¿Cuánto cuestan estos relacionamientos?
- ¿Qué tipo de fidelización espera nuestro cliente?
  Se deben considerar aspectos cómo:

- Asistencia personal.
- asistencia personal dedicada.
- autoservicio.
- servicios automatizados.
- comunidades.
- co-creación.
- Servicios colaterales o alternativos: fidelización.

Aliados clave / Socios estratégicos

Para llevar a cabo un negocio, es imprescindible tener aliados. Estos aliados pueden ser:

- Una serie de socios/colaboradores: una buena red de partners nos pueden ayudar a llegar más rápido al cliente e ir avalados y secundados por su reputación y experiencia.
- Los proveedores: aquellos que nos proporcionan los recursos clave para poder ofrecer los servicios/productos finales.

Debemos dar respuesta a:

- ¿Quiénes son nuestros socios clave en el mercado?
- ¿Quiénes son nuestros proveedores clave (cadena de suministros)?
- ¿Qué recursos clave vamos a adquirir de nuestros socios?
- ¿Qué recursos clave vamos a compartir con nuestros socios?
- ¿Qué actividades clave realizan los socios?
- ¿Contamos con red de inversores? ¿Dichos inversores aportan su red al negocio?
- ¿Cómo se comparte el negocio con nuestros aliados?
- ¿Requiere ser creada una nueva empresa o unidad de negocio para llevar adelante nuestro servicio o producto?

Se deben considerar aspectos cómo:

- Optimización y economía.
- Reducción de riesgo e incertidumbre.
- Adquisición de recursos y actividades particulares.

Recursos clave

Conocer con qué recursos contamos y con los que debemos contar para llevar a cabo la actividad de nuestro negocio, es clave a la hora de establecer el plan de negocios. Debemos de ser cautos y prudentes a la hora de definir estos recursos. Siempre debemos pensar en la forma de optimizarlos, es decir, intentar conseguir la máxima productividad posible al mínimo coste.

Debemos dar respuesta a:

- ¿Qué recursos esenciales requiere nuestra propuesta de valor?
- ¿Nuestros canales existentes de distribución?
- ¿Nuestras relaciones con clientes existentes?
- ¿Nuestras fuentes de ingresos?
- ¿Contamos con costos hundidos por la existencia de servicios o productos similares? ¿Se puede aprovechar recursos de productos o servicios existentes?

Se deben considerar aspectos cómo:

- Físico.
- Intelectual (patentes, marca, copyright, datos, etc.).

- Humanos.
- Financieros.

Actividades clave

Para llevar a cabo la propuesta de valor que queremos ofrecer a nuestros clientes, son necesarias ciertas actividades para preparar el producto antes de que llegue al mercado. Es decir, aquí pensamos en el core de nuestro negocio, lo que haremos en nuestro día a día.

Debemos dar respuesta a:

- ¿Qué actividad básica requiere nuestra propuesta de valor?
- ¿Cuáles son nuestros canales?
- ¿Cuáles son nuestras fuentes de ingresos?
- ¿Nuestras relaciones con clientes?

Se deben considerar aspectos cómo:

- Producción.
- Resolución de problemas.
- Plataforma/red.

Flujo de ingresos

Para que un negocio sea rentable y podamos sobrevivir en el mercado, tenemos que pensar ¿Cómo monetizarlo? Es decir ¿De dónde vamos a obtener la facturación? ¿Cómo monetizar nuestro servicio o producto?

Debemos dar respuesta a:

- ¿Cuál es nuestra principal línea de ingresos?
- ¿Cómo pagarán nuestros clientes?
- ¿Por qué están dispuestos a pagar nuestros clientes?
- ¿Qué valor están realmente dispuestos a pagar nuestros clientes?
- ¿Por qué pagan actualmente? ¿cómo están pagando ahora? ¿cómo preferirían pagar?
- ¿cuánto contribuye cada fuente de ingresos a los ingresos totales?

Se deben considerar aspectos cómo:

- **Tipos**: Venta de activos, pago por uso, cuota de suscripción, préstamo/alquiler/leasing, licencias, tasas intermediación, publicidad, servicios de fidelización, etc.
- **Precio fijo**: listas de precios, dependiente de la funcionalidad del producto, dependiente del segmento del cliente, dependiente del volumen.
- **Precio dinámico**: negociación, gestión de rendimientos, mercado en tiempo real.

Estructura de costos

Obviamente, toda esta infraestructura tiene sus costos que debemos pagar y optimizar. Debemos definir cuáles son nuestras prioridades y los gastos fundamentales en el negocio de aquellos que no lo son.

Tener bien clara esta estructura nos ayudará a no desviarnos de los presupuestos y que el negocio fracase por problemas de financiación.

Debemos dar respuesta a:

- ¿Cuáles son los costes más importantes dentro de nuestro modelo de negocio?
- ¿Qué recursos clave son los más costosos?
- ¿Qué actividades clave son las más costosas?
- ¿Conocemos el ABC de nuestros insumos?
    - Categoría A: En torno al 20% de las referencias representan aproximadamente el 80% del valor (regla 80/20).
    - Categoría B: En torno al 30% de las referencias representan aproximadamente el 15% del valor.
    - Categoría C: En torno al 50% de las referencias representan sólo el 5% del valor.

Se deben considerar aspectos cómo:

- El negocio está basado en:
    - **costo**: estructura de coste más escueta, propuesta de valor de precio bajo,automatización, outsourcing extensivo, etc.
    - **valor**: centrado en creación de valor, propuesta de valor premium, etc
- características de ejemplo:
    - **costos fijos**: salarios, rentas, estructuras, etc.
    - **costos variables**: economía de escala, economía de alcance, etc.

1.5.5 Resultados y aportes del sistema al negocio

En esta sección se debe completar lo que se ha resumido a modo de cierre la determinación entre lo proyectado (planificado) en las secciones Visión del negocio (Envisioning) y Model Canvas. Las dos primeras secciones resumen lo que el negocio demandó y cuál fue la estrategia y planificación del mismo y esta última sección debe resumir el grado de complimiento y los aportes reales al negocio en función del cambio.

Métricas: beneficio del aporte

Evaluando detenidamente el punto anterior, se deben indicar los mecanismos, herramientas, datos y análisis (sistémico) que permitirán determinar dichas conclusiones. En definitiva, se debe indicar cómo se respaldan (evidencia) las conclusiones.

1.6 NIVELES DE PLANIFICACIÓN ÁGIL

Figura 1.10 - Los 5 niveles de la planificación ágil

1.6.1 Algunos principios de la planificación ágil

- No obtener el plan completo desde el principio. La planificación desde el principio puede ser útil mientras no sea excesiva.
- Mantener las opciones de planificación abiertas hasta el último momento.
- Enfocarse más en adaptarse y re-planificar que en el cumplimiento conforme a un plan.
- Manejar correctamente el inventario de planificación (product backlog).
- Favorecer las entregas pequeñas y frecuentes.
- planificar un aprendizaje veloz y cambiar cuando sea necesario.

1.7 BACKLOG & VISUAL STORY MAPPING

El Visual Story Mapping es una herramienta que permite generar una representación visual del sistema completo. Ofrece una vista general de todas las funcionalidades que lo componen (la gran pintura) de punta a punta. Permite

identificar requerimientos (User Stories) faltantes en la pila de producto (backlog), planificar entregas (releases), partiendo en rebanadas (Slicing), visualizar cómo se distribuyen las funcionalidades de acuerdo con las diferentes áreas del sistema.

Es una forma de reorganizar la pila de producto (product backlog) en dos dimensiones, una dimensión para el tiempo (medido en releases) y otra dimensión para las funcionalidades.

1.7.1 ¿Cómo se construye?

Para la construcción resulta fundamental pensar todo el proceso desde el punto de vista del usuario y sus objetivos con nuestro sistema; nunca desde el lugar de dueño del producto o de quienes lo construyen y ofrecen el servicio.

1. **Metáforas (Backbone)**: Identificar las grandes áreas (módulos) en las que se divide nuestro sistema. Por ejemplo, en un sistema de venta online, podría ser: llegar al website, búsqueda de producto, compra/pago y entrega.
2. **Actividades**: Identificar todas las actividades que debe o puede realizar el usuario en nuestro sistema, desde que llega al mismo hasta que finaliza su proceso. Ordenarlas de manera secuencial de manera horizontal. Esta secuencia conforma la cadena de valor de nuestro sistema (Value Stream).
3. **Historias de usuario**: Identificar las distintas maneras que nuestro sistema puede ofrecer para que el usuario realice cada una de las actividades identificadas. Por ejemplo, para la actividad «encontrar producto» podría ser: navegar el catálogo, usar el cuadro de búsqueda, sugerencias de productos, etc.
4. **Priorización**: Cada columna debajo de cada Activity funciona como un backlog que debe estar priorizado por relevancia. Para la priorización debemos utilizar el criterio de MoSCoW (Must Have, Should Have, Could Have). De manera que la funcionalidad más elemental, que no puede faltar debe estar primera.

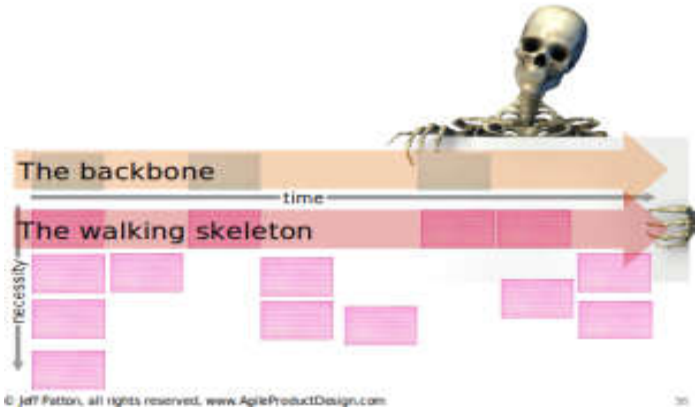

Figura 1.11 - El esqueleto que camina

Una vez armado el mapa, podemos identificar a la fila de grandes funcionalidades (épicas y metáforas) como la columna vertebral de nuestro sistema (The backbone), estas no deben priorizarse y permiten dar contexto a las Historias de Usuario.

Con las activities y la primera fila de User Stories obtenemos lo que Jeff Patton denomina «The Walking Skeleton» (el esqueleto que camina) porque está completo y funciona, aunque sea en su forma más elemental.

Nuestro producto en este estado puede ser identificado como la versión mínima viable MVP (Minimum Viable Product) de su sigla en inglés. O como él lo llama, MMF (Minimum Marketable Feature).

1.7.1.1 Release Planning

El User Story Mapping permite planificar las próximas versiones o releases del producto representando de manera visual en qué partes de nuestro sistema estamos dedicando mayor esfuerzo para generar Incrementos de Producto.

Esto puede ser muy poderoso, como estrategia de negocio si se lo asocia al ciclo de vida del negocio del cliente (Customer Life Cycle) para comprender qué partes de nuestro sistema deben ser mejoradas.

El User Story Map permite a cualquier stakeholder comprender qué funcionalidades ya fueron realizadas, cuáles se están trabajando en este momento y cuáles son las próximas funcionalidades para todo el producto y la organización.

El User Story Map, está centrado en el usuario y representa el flujo del producto de punta a punta, por lo tanto, integra a muchos (sino todos) los equipos de la organización, promoviendo alineamiento de prioridades y una visión más integral.

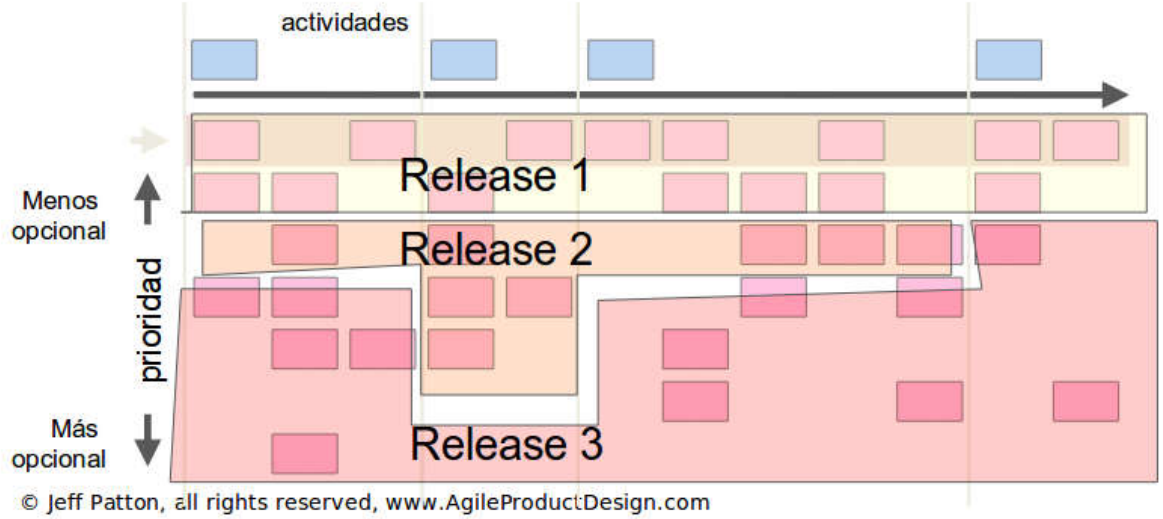

Figura 1.12 - Ejemplo de un Visual Story Mapping

En las metodologías ágiles recomiendan los equipos por funcionalidad del producto (Feature Teams) y la herramienta de User Story Map se adapta perfectamente a esta estructura. Es probable que un equipo se dedique a implementar determinadas actividades y por lo tanto se identifique con determinadas columnas del Story Map.

La herramienta permite visualizar si la carga de trabajo pendiente se corresponde con la capacidad o tamaño de los equipos de desarrollo.

Si la organización estructura los equipos como Component Team (equipos por plataforma) se pierde esta capacidad de visualización. Sin embargo, se podría utilizar código de colores para identificar al equipo.

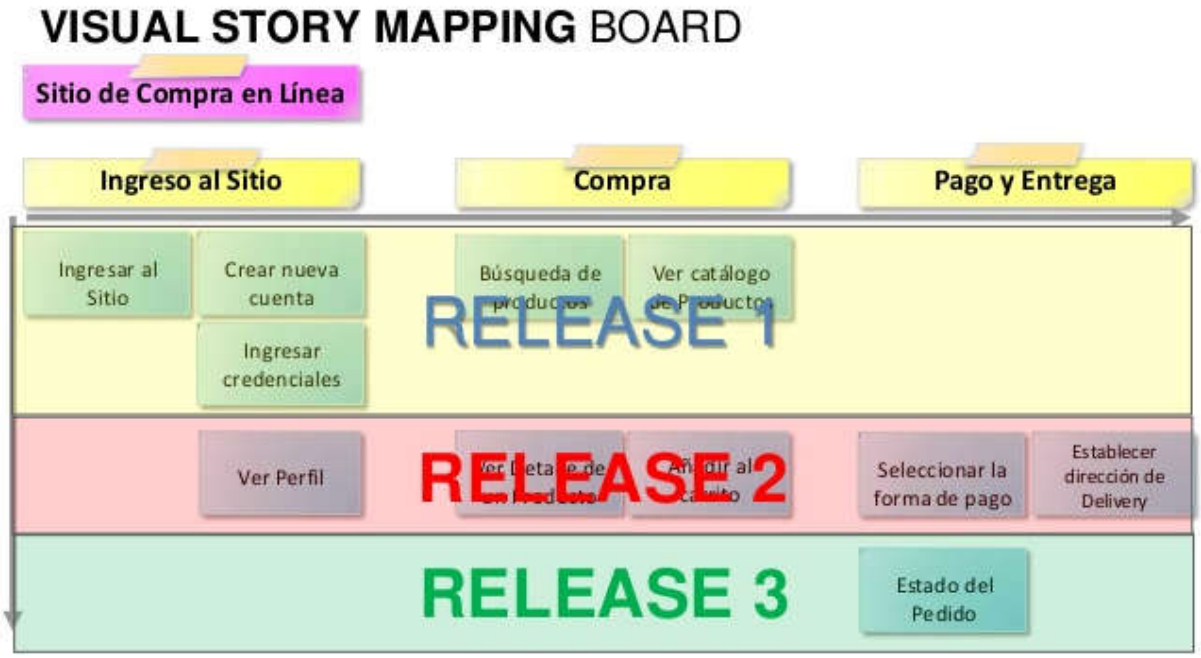

Figura 1.13 - Ejemplo de un Visual Story Mapping

1. 8 LAS 4 DIMENSIONES DE LA PLANIFICACIÓN ÁGIL

Las dimensiones ayudan a poner en valor los distintos aspectos que se deben tener en cuenta en el desarrollo de un producto bajo la metodología y cultura ágil. Cabe destacar que estas dimensiones no sólo se definen en la planificación inicial, sino que deben ser continuadas y mantenidas durante todo el proyecto. Se debe recordar que la cultura ágil promueve la planificación contínua versus la elaboración de planes monolíticos. A Continuación, se mencionan brevemente estas dimensiones:

1.8.1 Dimensión producto (negocio)

En esta dimensión se consideran todos los aspectos de definición necesarios para dar valor a la idea, servicio o producto bajo construcción. Entre algunos aspectos que se deben tener en cuenta describir podemos mencionar:

- Visión del producto.
- Actores y matriz de responsabilidades (Modelo de negocio).
- Metáforas.
- Épicas.
- Pila de Producto.
- Atributos de Calidad.

1.8.2 Dimensión tecnológica

La idea de la dimensión es indicar a nivel de detalle todas las herramientas y sus técnicas asociadas que se utilizarán durante el desarrollo del proyecto.

El objetivo es poner en valor las herramientas tecnológicas que se utilizarán, garantizando el ciclo de desarrollo de forma integral.

A continuación, se enumeran las categorías mínimas de herramientas a presentar:

- Herramientas y sus estándares de desarrollo (uso del stack tecnológico).
- Herramientas y estándares para la gestión, control y seguimiento de las actividades del proyecto.
- Herramientas y estándares para la validación y verificación del software entregado.
- Herramientas y estándares para la gestión de las configuraciones del software.
- Herramientas y estándares para la gestión del despliegue de los componentes en producción.
- Herramientas y estándares para la gestión y control de riesgos del proyecto.

1.8.3 Dimensión gestión

La idea de esta sección es sobre las herramientas planteadas en la sección Dimensión tecnológica realizar un detalle del proceso de cómo se utilizarán e integrarán las mismas de forma sistémica.

Algunos aspectos que se recomienda tener en cuenta para la planificación de esta dimensión son:

- Roles del proyecto (no del negocio)

En esta sección se requiere una lista con los roles que intervinieron en el proyecto, su responsabilidad y quien o quienes han ocupado cada uno de los mismos. No se deben confundir con los roles (actores) del negocio (dimensión producto).

- Artefactos que se utilizarán

A modo de sugerencia se deberían contemplar al menos los siguientes artefactos:

- Visión del producto.
- Pila de producto (product backlog).
- Visual Story Mapping / Product roadmap.
- Pila de releases.
- Pila del ciclo de desarrollo (Sprint backlog).
- Historias de Usuario / Caso de Uso.
- Sprint Burndown.
- Kanban.
- Impedimentos.
- Riesgos.
- Acciones de mejora.
- Incremento.
- Definición de terminado.
- Gráficos:
    - Gráficos del trabajo pendiente.
    - Gráfico de flujo acumulado.
    - Release burnup (down).
    - Product burnup (down).
        - Prácticas de gestión (dinámica)

En función de los artefactos utilizados se requiere una breve explicación de cómo se utilizarán los artefactos mediante las actividades de gestión. A continuación, proponen que se comenten las siguientes actividades:

- Envisioning
- Product planning
- Release planning
- Sprint planning
- Planning poker
- Dinámica de la ejecución del sprint
- Standup daily meeting
- Sprint review
- Retrospectivas

1.8.4 Dimensión calidad y proceso

La última dimensión intenta poner en valor y resumir las acciones correctivas y de mejora tomadas durante el proyecto y si se ha logrado algún tipo de sistematización.

A modo de guía se comentan aspectos a verificar en el proyecto:

Empirismo

El empirismo asegura que el conocimiento procede de la experiencia y de tomar decisiones basándose en lo que se conoce. Las metodologías ágiles emplean un enfoque iterativo e incremental para optimizar la predictibilidad y el control del riesgo.

El empirismo es fundamental en la cultura ágil sin él no podríamos cumplir con las premisas del ciclo de vida iterativo e incremental y realizar gestión adaptativa.

Mejora Continua

La mejora continua del proceso y metodología de desarrollo será un eje rector durante el proyecto. Desde el empirismo es muy importante lograr, no sólo detectar lo que se hace bien o lo que se hace mal. Estas detecciones deben estar fundamentadas en mediciones que permitan establecer si los cambios que introducimos mejoran nuestro rendimiento y calidades de producto esperadas.

El empirismo percé no alcanza como modelo de mejora continua, pues las experiencias realizadas deben fundamentar su evolución en las mediciones que se establezcan.

En este sentido será fundamental al momento de establecer una condición, sucesos o decisiones sobre la relación empírica de oportunidades para el proyecto, cuál será el sistema de medición que nos permita establecer si los cambios que vamos afrontando, mejoran o no el proceso adoptado.

Adaptabilidad

Disposición y buena voluntad para llevar adelante las acciones de cambio sugeridas a partir del empirismo y la mejora continua de nuestro proceso.

Las metodologías ágiles basan la adaptabilidad fundamentalmente en los cambios requeridos para que el proceso de desarrollo mejore.

Colaboración

Fomentar la comunicación tanto entre los integrantes del equipo como así también entre todos los involucrados en el proyecto, entendiendo que las mejores soluciones y diseños creativos surgen holísticamente de todo el grupo de personas interesadas.

Para ello es fundamental fomentar esta cultura mediante prácticas de diseño emergente y convergente. Como por ejemplo desde técnicas de revisión y brainstorming hasta técnicas de convergencia como “Design Thinking”, “Círculo de control de calidad” o “Sí y además”.

- Algunas pistas y características que permiten fomentar la colaboración son:
- Múltiples personas interactuando para resolver el problema. Mejora la resolución del mismo cubriendo todos los aspectos a tener en cuenta.

- Atar cada decisión de diseño a un aspecto a resolver del problema, evitando agregar capacidades innecesarias.
- Despegarnos de nuestros diseños. La humildad intelectual es parte del agilismo. Si contamos con un martillo, todos los problemas son "clavos". La predisposición a escuchar activamente al otro es fundamental.
- Hacer generalizaciones sólo por inducción, cuando hay suficientes ejemplos para justificarla. Esto permite además fomentar el empirismo.

1.9 MAPA DE CALIDAD CMMI-ÁGILES

A modo de resumen del presente capítulo se sintetizan las áreas clave propuestas por CMMi para los niveles de maduración 2 y 3 y se establece la correlación de las herramientas, técnicas y prácticas propuestas por las metodologías ágiles para dar cumplimiento a las mismas.

1.9.1 Gestión De Requisitos (REQM)

REQM	***Práctica CMMi***	***Práctica Ágil***
1.1	Comprender los requisitos	Revisar la pila de producto (*Product Backlog) (características, épicas, metáforas)* con dueño de producto (PO) y equipo.
1.2	Obtener el compromiso sobre los requisitos	*Pila de producto priorizada*, *Visual Story Mapping*, *Release Planning*. Se consensúan con interesados (stakeholders).
1.3	Gestionar los cambios a los requisitos	*Limpieza de la pila de producto (Backlog Grooming)*, *DEMOSTRACIÓN* y *Sprint review*. Planificación constante. Adaptativo.
1.4	Mantener la trazabilidad bidireccional de los requisitos	Herramientas de Scrum: Trello, Jira, Kanbanize, Pivotal Tracker, iceScrum, GitLab. Combinar herramientas con Integración Continua; TDD + BDD → IC.
1.5	Asegurar el alineamiento entre el trabajo del proyecto y los requisitos	*StandUp Daily Meeting (reunión diaria)*, *DEMO*, *Retrospectiva*, *Sprint Planning*, *Backlog Grooming*.

1.9.2 Planificación Del Proyecto (PP)

- Establecer Las Estimaciones (SG1)

PP	***Práctica CMMi***	***Práctica Ágil***
1.1	Estimar el alcance del proyecto.	*Pila de producto (Backlog)* priorizada y estimada, *Visual Story Mapping*, *Release Mapping*.
1.2	Establecer las estimaciones de los atributos de los productos de trabajo y de las tareas.	*Puntos de historias (Story points)*, *Planning Poker*, *pesos relativos*, *fibonacci*.
1.3	Definir las fases del ciclo de vida del proyecto.	Proceso Ágil: Iterativo e incremental: evolutivo.
1.4	Estimar el esfuerzo y el coste.	*Velocidad (Velocity)*, *hora ideal*, *spikes*, *teamwork*.

- Desarrollar Un Plan De Proyecto (SG2)

PP	***Práctica CMMi***	***Práctica Ágil***
2.1	Establecer el presupuesto y el calendario.	*Velocity & spikes*, *visual story mapping*, *sprint release*, *sprint backlog*.
2.2	Identificar los riesgos del proyecto.	Todas las prácticas ágiles identifican riesgos: Daily + Burndown + Retrospectivas + Planning.
2.3	Planificar la gestión de los datos.	No se establece práctica → Ingeniería
2.4	Planificar los recursos del proyecto.	Teamwork + Velocity & spikes
2.5	Planificar el conocimiento y las habilidades necesarias.	*Programación en parejas (pair programming)*, revisión de pares (*peer reviews)*, inspección de código (*code inspection)*, *retrospectivas* y acciones de mejora → "empirismo".
2.6	Planificar el compromiso de las partes interesadas.	Roles Scrum e interacción de los roles.
2.7	Establecer el plan de proyecto.	Proceso ágil, prácticas y herramientas → Software Development Plan → entallamiento de las prácticas, roles y artefactos.

- Obtener El Compromiso Con El Plan (SG3)

PP	***Práctica CMMi***	***Práctica Ágil***
3.1	Revisar los planes que afectan al proyecto.	*Standup daily meeting*, *DEMO*, *retrospectiva*, *sprint planning meeting*, *release planning meeting*.
3.2	Conciliar los niveles de trabajo y de recursos.	*Standup daily meeting y* sprint planning.
3.3	Obtener el compromiso con el plan.	*Daily* con participación de PO, sprint planning

1.9.3 Monitorización Y Control Del Proyecto (PMC)

- Monitorizar El Proyecto Frente Al Plan (SG1)

PMC	***Práctica CMMi***	***Práctica Ágil***
1.1	Monitorizar los parámetros de planificación del proyecto.	Burndown (todos los niveles), velocity vs. capacidad, flujo acumulativo.
1.2	Monitorizar los compromisos.	Burndown (todos los niveles), flujo acumulativo, daily, retrospectiva y planning meeting.
1.3	Monitorizar los riesgos del proyecto.	Risk burndown.
1.4	Monitorizar la gestión de los datos.	No definido. Prácticas de ingeniería.
1.5	Monitorizar la involucración de las partes interesadas.	Sprint review meeting, daily con PO, planning meeting.
1.6	Llevar a cabo las revisiones del progreso.	Daily meeting, Sprint review meeting, retrospectiva.
1.7	Llevar a cabo las revisiones de hitos.	Sprint review meeting, release burndown, kanban board.

- Gestionar Las Acciones Correctivas Hasta Su Cierre. (SG2)

PMC	***Práctica CMMi***	***Práctica Ágil***
2.1	Analizar las cuestiones.	Retrospectiva → acciones de mejora contínua. Risk Burndown.
2.2	Llevar a cabo las acciones correctivas.	Sprint planning, contemplando las acciones de mejora (spikes) y el plan de mitigación de riesgos.
2.3	Gestionar las acciones correctivas.	Mismas herramientas de seguimiento que el resto de la planificación.

1.10 ROLES, ARTEFACTOS Y PRÁCTICAS ÁGILES

En este último capítulo se proporciona una descripción general de las metodologías Scrum y eXtreme Programming, con un enfoque en sus prácticas, roles y artefactos.

1.10.1 Scrum

Scrum es un marco para organizar y administrar el trabajo dentro de un proyecto ágil. Scrum se basa en un conjunto de valores, principios y prácticas que proporcionan la base a la que una organización agrega su implementación única de prácticas de ingeniería relevantes y sus enfoques específicos para realizar el enfoque propio de Scrum. El resultado será una versión de Scrum exclusivamente entallada a las necesidades de cada proyecto.

Para comprender mejor el concepto de la metodología, se puede pensar a Scrum como los cimientos y las paredes de un edificio. Los valores, principios y prácticas de Scrum serían los componentes estructurales clave. No se puede ignorar o cambiar fundamentalmente un valor, principio o práctica sin correr el riesgo de colapsar. Sin embargo, lo que se puede hacer es personalizar dentro de la estructura de Scrum, agregando accesorios y funciones hasta que tenga un proceso que funcione para un proyecto en particular, especialmente si se interpreta a las metodologías ágiles como un marco de desarrollo adaptativo de nuestro trabajo.

El planteo de Scrum es refrescantemente simple y centrado en las personas basado en los valores de honestidad, apertura, coraje, respeto, enfoque, confianza, empoderamiento y colaboración.

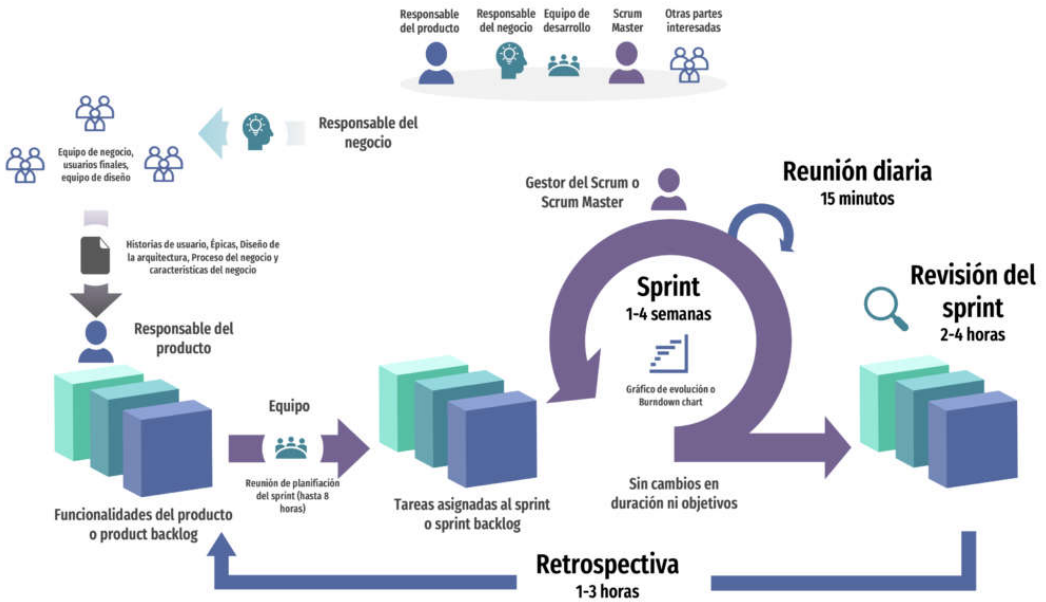

Figura 12-14: - Marco de desarrollo Scrum

1.10.2 Roles de Scrum

Los esfuerzos de desarrollo de Scrum consisten en uno o más equipos de Scrum, cada uno compuesto por tres roles: dueño del producto (Product Owner), Scrum Master y el equipo de desarrollo. Puede haber otros roles, pero para Scrum requiere solo los tres enumerados aquí.

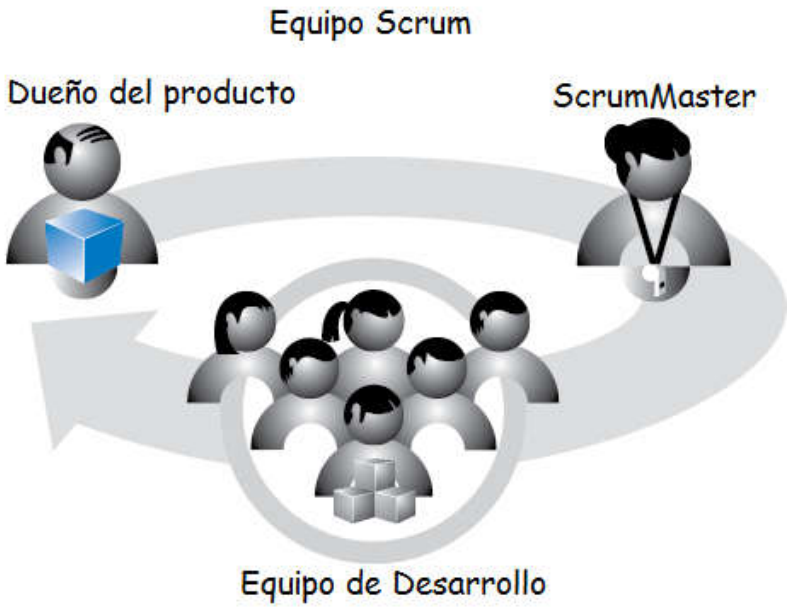

Figura 12.15 - Equipo Scrum

El dueño del producto es responsable de lo que se desarrollará y en qué orden. El Scrum Master es responsable de guiar al equipo en la creación y el seguimiento de su propio proceso basado en el marco más amplio de Scrum. El equipo de desarrollo es responsable de determinar cómo entregar lo que ha pedido el dueño del producto.

Que "gerente" no aparezca como un rol en la figura no nos debe inquietar; los gerentes aún tienen un papel importante en las organizaciones que usan Scrum. La metodología Scrum define sólo los roles que son específicos de Scrum, no todos los roles que pueden y deben existir dentro de una organización que usa Scrum. Es importante comprender que están el resto de los actores del cliente y de la organización, Scrum los divide en los que forman parte del equipo y los que no, pero de ninguna manera son ignorados.

1.10.2.1 Dueño de Producto

El dueño del producto es la persona central facultada con el liderazgo del producto. Es la única autoridad responsable de decidir qué características y funcionalidades construir y el orden para hacerlo. El dueño del producto mantiene y comunica a todos los demás participantes una visión clara de lo que el equipo Scrum

está tratando de lograr. Como tal, el dueño del producto es responsable del éxito general de la solución que se desarrolla y/o mantiene.

No es relevante si el foco está en la implementación de un producto externo desarrollado por terceros o en una aplicación interna; el dueño del producto aún tiene la obligación de asegurarse que siempre se realice el trabajo más valioso posible, que puede incluir trabajo enfocado técnicamente. Para garantizar que el equipo construya rápidamente lo que quiere el dueño del producto, éste colabora activamente con el Scrum Master y el equipo de desarrollo y debe estar disponible para responder preguntas una vez que son planteadas.

En resumen, las responsabilidades del dueño del producto son:

- Decidir qué construir o implementar y que no.
- Relevar, recuperar, sintetizar y tener claros los requisitos del negocio y del producto bajo construcción/implementación.
- Definir buenas “Historias de usuario” o “Casos de uso”. (requerimientos).
- Fijar los criterios de aceptación que garanticen el cumplimiento de los requerimientos.
- Ordenar y priorizar los ítems de la pila del producto.
- Definir el mínimo producto viable (MVP), el visual story mapping y el plan de releases.
- Acordar junto al resto del equipo una definición de terminado. Acordando atributos y principios de calidad del producto bajo construcción.
- Validar las entregas (Sprints Review).

Se recomienda en función de los valores propuestos por la metodología que el dueño del producto no de órdenes al resto del equipo, garantizando una mirada colectiva y colaborativa del producto bajo desarrollo. Así mismo el equipo no debiera trabajar en otros requerimientos distintos a los acordados para cada Sprint.

El dueño del producto debe garantizar o resolver los siguientes aspectos durante el desarrollo del producto bajo construcción:

- Estar disponible y accesible para el equipo, todas las veces que éste lo requiera. Elaborar material de entrada y gestionar los datos del negocio si estos son requeridos.
- Responsable de gestionar los impedimentos durante el sprint y resolverlos dentro de sus posibilidades. En caso extremo también es el único responsable de suspender un sprint en el caso extremo de un impedimento grave.
- Asegurar que todo el entorno del proyecto: equipo, interesados y actores del proyecto entienden el alcance de los requerimientos, estableciendo para ello los criterios de aceptación y los materiales de síntesis como los storyboards o maquetas de la funcionalidad requerida. El equipo colabora con el dueño del producto en estas tareas.

Es importante a su vez que el dueño del producto conozca el ritmo de trabajo del equipo (velocity), garantizando y acompañando el ritmo del flujo de construcción

(paz sostenible), sin presiones, enfocado en aportar el mayor valor posible para resolver el negocio en cuestión.

A su vez el dueño del producto participa de las siguientes actividades establecidas en Scrum:

- Reunión diaria (daily meeting), especialmente enfocado en los impedimentos y la priorización de la pila del sprint (sprint backlog).
- Todas las reuniones de planificación, incluyendo la planificación de cada sprint (sprint planning).
- Revisión del sprint (sprint review).
- Retrospectiva del sprint (sprint retrospective).

1.10.2 Scrum Master

El Scrum Master ayuda a todos los involucrados a comprender y adoptar los valores, principios y prácticas de Scrum. Actúa como entrenador, proporciona liderazgo de procesos y ayuda al equipo Scrum y al resto de la organización a desarrollar su propio enfoque Scrum de alto rendimiento y específico de la organización. Al mismo tiempo, el Scrum Master ayuda a la organización a través del desafiante proceso de gestión de cambios que puede ocurrir durante la adopción de Scrum.

Como facilitador, el Scrum Master ayuda al equipo a resolver problemas y realizar mejoras en el uso de Scrum. También es responsable de proteger al equipo de la interferencia externa y asume un papel de liderazgo en la eliminación de los impedimentos que inhiben la productividad del equipo (cuando los individuos por sí mismos no pueden resolverlos razonablemente). El Scrum Master no tiene autoridad para ejercer control sobre el equipo, por lo que este rol no es el mismo que el rol tradicional de gerente de proyecto o gerente de desarrollo. El Scrum Master funciona como líder, no como gerente y del proceso y sus prácticas y no sobre el alcance del producto.

En resumen, las responsabilidades del Scrum Master son:

- Planifica la implementación de Scrum conjuntamente con la organización.
- Ayuda y colabora con la organización a entender qué interacciones con el equipo aportan valor y cuáles no.
- Ayuda y colabora con el dueño del producto a entender la agilidad.
- Ayuda y colabora con el dueño del producto a maximizar el valor del negocio, respecto de la solución bajo construcción.
- Enseña y transfiere capacidades al dueño del producto para priorizar y gestionar efectivamente la pila del producto, aplicando técnicas de limpieza y mantenimiento de la pila de producto (backlog grooming).

● Ayuda, colabora y transfiere capacidades al equipo de desarrollo para convertirse en un equipo auto-organizado y multifuncional. Un equipo Scrum maduro no necesita Scrum Master.

● Colabora con la solución de impedimentos e imprevistos que suceden durante el proyecto.

● Asegura que existan criterios de aceptación entendibles y garantiza que exista una definición de terminado.

Es muy importante para la metodología no confundir los roles de Scrum Master y dueño del producto, pues ambos tienen funciones y responsabilidades bien distintas dentro del proyecto. El dueño del producto cuenta con una visión más del negocio y de los que se requiere de la solución para garantizar dichos objetivos en cambio el Scrum Master se encarga de que todo el equipo, la organización y los actores involucrados entiendan Scrum y lo apliquen correctamente.

El Scrum Master debe garantizar y colaborar los siguientes aspectos durante el proceso de desarrollo:

● Ayuda, colabora y garantiza que las acciones de mejora detectadas durante las retrospectivas sean llevadas a cabo.

● Mantiene y colabora con la visibilidad del proceso y sus resultados mediante la utilización de las gráficas y las métricas del proceso.

● Asegura y promueve que se cumplan las premisas de las buenas prácticas técnicas de desarrollo y de la programación.

● Realiza y promueve actividades de capacitación y transferencia de capacidades a la organización y al equipo para mejorar los aspectos metodológicos del proceso cada vez que sea necesario.

● Promueve, gestiona y organiza actividades de mejora contínua durante el proyecto: revisión técnica de pares, deuda técnica, atributos de calidad, alineamiento tecnológico de la solución, etc.

● A su vez el Scrum Master participa de las siguientes actividades establecidas en Scrum:

● Reunión diaria (daily meeting), especialmente enfocado en los impedimentos y la priorización de la pila del sprint (sprint backlog).

● Todas las reuniones de planificación, incluyendo la planificación de cada sprint (sprint planning).

● Revisión del sprint (sprint review).

● Retrospectiva del sprint (sprint retrospective).

1.10.3 Equipo de desarrollo

Los enfoques tradicionales de desarrollo de software analizan varios tipos de roles de trabajo, como arquitecto, programador, probador (tester), administrador de base de datos, diseñador, etc. Scrum define el papel de un equipo de desarrollo, que

es simplemente una colección diversa y multifuncional de este tipo de personas que son responsables de diseñar, construir y probar el producto deseado.

El equipo de desarrollo se autoorganiza para determinar la mejor manera de lograr el objetivo establecido por el dueño del producto en función del negocio a resolver. El equipo de desarrollo suele tener entre tres y nueve personas; sus miembros deben tener colectivamente todas las habilidades necesarias para producir software funcional de buena calidad. Por supuesto, Scrum se puede utilizar en esfuerzos de desarrollo que requieren equipos mucho más grandes. Sin embargo, en lugar de tener un equipo Scrum con, digamos, de 35 personas, lo más probable es que haya cuatro o más equipos Scrum, cada uno con un equipo de desarrollo de nueve personas o menos.

En resumen, las responsabilidades del Equipo de Desarrollo son:

- Encargado de producir y entregar incrementos de producto “terminado”.
- Estructurados y empoderados para organizar y gestionar el trabajo, manteniendo el foco en el valor del negocio, proactivos, colaborativos y cooperativos con el dueño del producto.
- Multifuncional e interdisciplinario, con las habilidades necesarias para crear y producir incrementos al producto.
- No existen jefes, ni subgrupos, ni dueños de tareas ni sectorización del trabajo. La colaboración es la constante del equipo. Se mantienen y promueven los valores de “propietarios del código” (code ownership) “todo el equipo” (whole team).
- Si bien se enfocan en el concepto de moverse alrededor del código (move around) pueden existir especialistas en determinados temas que deberán transferir al resto del equipo dichos saberes.

1.10.3 Artefactos y actividades Scrum

La Figura 1.14 ilustra la mayoría de las actividades y artefactos de Scrum y cómo encajan entre sí.

Para interpretar la figura correctamente se realizará un recorrido de izquierda a derecha de la misma.

El dueño del producto tiene una visión de lo que quiere crear (el gran cubo). Debido a que el cubo puede ser grande, a través de una actividad llamada limpieza (grooming), se divide en un conjunto de características que se recopilan en una lista priorizada llamada pila de producto (Product Backlog).

Un ciclo/iteración (sprint) comienza con la planificación del sprint, abarca el trabajo de desarrollo durante el sprint (llamado ejecución del sprint) y finaliza con la revisión y la retrospectiva. El sprint está representado por la gran flecha en bucle que domina el centro de la figura. Es probable que la cantidad de elementos en la pila de producto (Product Backlog) sea más de lo que un equipo de desarrollo puede completar en un sprint de corta duración. Por esa razón, al comienzo de cada sprint,

el equipo de desarrollo debe determinar un subconjunto de los elementos de la pila de producto que el equipo de desarrollo determina que puede completar en un ciclo, una actividad denominada planificación de sprint, que se muestra justo a la derecha del cubo grande de la pila de producto.

Para promover la confianza de que el equipo de desarrollo ha hecho un compromiso razonable para el ciclo, los miembros del equipo crean un segundo trabajo pendiente durante la planificación del sprint, llamado trabajo pendiente del sprint. La pila del sprint (Sprint Backlog) describe, a través de un conjunto de tareas detalladas, cómo el equipo planea diseñar, construir, integrar y probar el subconjunto seleccionado de funciones de la pila del producto durante ese ciclo (sprint) en particular.

Lo siguiente es la ejecución del sprint, donde el equipo de desarrollo realiza las tareas necesarias para concretar y producir las funciones del negocio seleccionadas para ser resueltas. Cada día, durante la ejecución del sprint, los miembros del equipo ayudan a gestionar el flujo de trabajo mediante la realización de una actividad de sincronización, inspección y planificación adaptativa conocida como “reunión diaria” (Daily Scrum). Al final de la ejecución del sprint, el equipo ha producido un incremento de producto potencialmente entregable que representa parte, pero no toda, la visión del dueño del producto.

El equipo Scrum completa el sprint realizando dos actividades de inspección y adaptación. La primera, llamada revisión del sprint, las partes interesadas y el equipo Scrum inspeccionan el producto que se está construyendo. La segunda, llamada retrospectiva del sprint, el equipo Scrum inspecciona el proceso Scrum que se está utilizando para crear el producto. El resultado de estas actividades pueden ser adaptaciones que se abrirán paso en la pila de producto o se incluirán como parte del proceso de desarrollo del equipo.

En este punto, el ciclo de sprint de Scrum se repite, comenzando de nuevo con el equipo de desarrollo determinando el siguiente conjunto más importante de elementos de la pila de producto que puede completar. Después de que se haya completado una cantidad adecuada de sprints, la visión del dueño del producto se realizará y se podrá lanzar la solución. Esto estará alineado con las actividades de visión del producto (solución), incluyendo el Visual Story Mapping y la planificación del plan de versiones (releases) y el mínimo producto viable (MVP).

1.10.3.1 Visión del producto

La actividad de visión del producto ya ha sido descrita en la sección Puesta en valor del negocio.

1.10.3.2 Pila del Producto (Product Backlog)

Cuando se usa Scrum, siempre se hace primero el trabajo de mayor valor para el negocio. El dueño del producto, con el aporte del resto del equipo Scrum y las partes interesadas, es el responsable final de determinar y administrar la secuencia de este trabajo y comunicarlo en forma de una lista priorizada (u ordenada) conocida como Pila de Producto (Product Backlog o simplemente Backlog). En el desarrollo de nuevos productos, los elementos de la pila de producto inicialmente son características necesarias para cumplir con la visión del dueño del producto. Para el desarrollo continuo de productos, la acumulación de ítems también puede contener nuevas funciones, cambios en las funciones existentes, defectos que necesitan reparación, mejoras técnicas, etc.

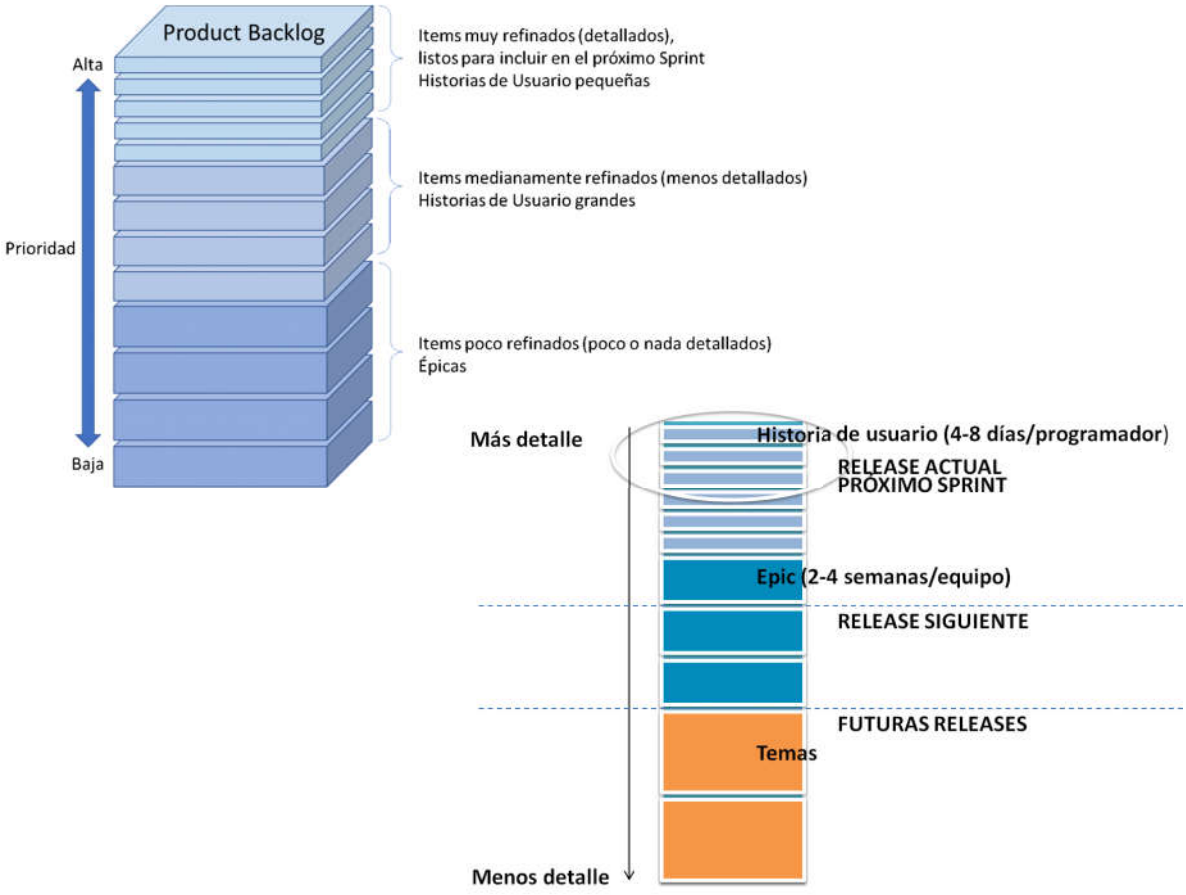

Figura 1.16 - Pila de Producto (Product Backlog)

El dueño del producto colabora con las partes interesadas internas y externas (actores del negocio a resolver) para recopilar y definir los elementos de la pila del producto. Luego se asegura que dichos elementos se coloquen en la secuencia correcta (usando factores como el valor, el costo, el conocimiento y el riesgo) de modo que los elementos de alto valor aparezcan en la parte superior de la pila de producto y los elementos de menor valor aparezcan debajo, en el fondo de la pila. La pila de

producto es un artefacto en constante evolución, completamente dinámico. El dueño del producto puede agregar, eliminar y revisar elementos a medida que cambian las condiciones del negocio, o a medida que crece la comprensión del producto por parte del equipo Scrum (a través de comentarios sobre el software producido durante cada sprint).

La actividad de crear y refinar elementos de la pila de producto, estimarlos y priorizarlos se conoce como limpieza o acicalar la pila de producto (Backlog Grooming).

Antes de finalizar la priorización, limpieza y organización de la pila de producto, es necesario saber el tamaño de cada elemento.

1.10.3.4 Visual Story Mapping

El artefacto de Visual Story Mapping ya ha sido descrito en la sección Backlog & Visual Story Mapping.

1.10.3.5 Sprints

En Scrum, el trabajo se realiza en iteraciones o ciclos de hasta un mes calendario llamados sprints. El trabajo completado en cada sprint debe crear algo de valor tangible para el cliente o usuario.

El trabajo completado en cada sprint debe crear algo. Los sprints tienen un límite de tiempo (timeboxing), por lo que siempre tienen una fecha de inicio y finalización y, en general, se recomienda que todos tengan la misma duración. Un nuevo sprint sigue inmediatamente a la finalización del sprint anterior. Por regla general, no se permiten cambios en el alcance o que el equipo de desarrollo o los actores alteren los objetivos durante un sprint; sin embargo, las necesidades del negocio a veces hacen imposible el cumplimiento de esta regla de valor tangible para el cliente o usuario.

1.10.3.6 Planificación del Sprint

El ítem de la pila de producto puede representar muchas semanas o meses de trabajo, que es mucho más de lo que se puede completar en un solo sprint corto. Para determinar el subconjunto más importante de elementos de la pila de producto para construir un objeto entregable en el próximo sprint, el dueño del producto, el equipo de desarrollo y Scrum Master realizan la actividad de planificación del sprint.

Durante la planificación del sprint, el dueño del producto y el equipo de desarrollo acuerdan un objetivo del sprint que define lo que se supone que se puede lograr para el próximo sprint. Con este objetivo, el equipo de desarrollo revisa la pila de producto y determina los elementos de alta prioridad que el equipo puede lograr

de manera realista en el próximo sprint mientras trabaja a un ritmo sostenible, un ritmo al que el equipo de desarrollo puede trabajar cómodamente durante un período prolongado de tiempo.

Para adquirir confianza en lo que se puede hacer, muchos equipos de desarrollo dividen cada función objetivo en un conjunto de tareas. La recopilación de estas tareas, junto con sus elementos asociados de la pila de producto, forma un segundo conjunto de ítems denominado pila del sprint. Estas tareas requieren ser discutidas y conversadas con el dueño del producto para que no queden dudas del alcance, el negocio y la solución (el qué) debe contener cada ítem seleccionado. Para ello es muy importante que cada ítem cuente, además de su explicación coloquial, con una síntesis del negocio y al alcance esperado como: guión gráfico (storyboard), maquetas (mockups) y fundamentalmente criterios de aceptación con sus datos de base (estado inicial) y resultados esperados (estado esperado).

Luego, el equipo de desarrollo proporciona una estimación (preferentemente determinada en esfuerzo relativo -fibonacci por ejemplo- en relación a la velocidad -capacidad- del equipo en un sprint) del esfuerzo requerido para completar cada tarea. Luego el equipo de desarrollo divide los elementos de la pila del sprint en tareas de menor tamaño lo suficientemente pequeñas como para poder ejecutarlas en una jornada de trabajo de una persona. Esta división sirve al equipo a modo diseño y planificación justo a tiempo (just in time) sobre cómo realizar las funciones.

La mayoría de los equipos de Scrum que realizan sprints de dos semanas a un mes de duración intentan completar la planificación del sprint en aproximadamente cuatro a ocho horas de duración de reunión con intensa participación del dueño de producto. Un sprint de una semana no debería tomar más de un par de horas para planificar (y probablemente menos). Durante este tiempo hay varios enfoques que se pueden utilizar. El enfoque que se utiliza con mayor frecuencia sigue un ciclo simple: seleccionar un elemento de la lista de la pila del producto (siempre que sea posible, el siguiente elemento más importante según lo definido por el dueño del producto), dividir el elemento en tareas técnica que permitan ser verificada su finalización y determinar si el elemento seleccionado logra cumplir con una funcionalidad específica comprobable y de valor para el proyecto y el producto. Planificar el elemento dentro del sprint (en combinación con otros elementos destinados al mismo sprint). Verificar que la capacidad del equipo permite completar el trabajo dentro del sprint, repetir el ciclo hasta que el equipo no tenga capacidad para hacer más trabajo.

1.10.3.7 Ejecución del Sprint

Una vez que el equipo Scrum finaliza la planificación del sprint y acuerda el contenido del mismo, el equipo, guiado por el Scrum Master, realiza todo el trabajo a nivel de tarea necesario para realizar las funciones, donde "termina" significa que hay un alto grado de confianza en que se ha completado todo el trabajo necesario para producir características contando con una buena calidad.

Exactamente qué tareas realiza el equipo depende, por supuesto, de la naturaleza del trabajo (por ejemplo, ¿estamos construyendo software y qué tipo de software, o estamos construyendo hardware, o es trabajo de marketing?).

Nadie le debe indicar al equipo de desarrollo en qué orden o cómo hacer el trabajo a nivel de tarea en la pila del sprint. Los miembros del equipo definen su propio trabajo a nivel de tarea y luego se autoorganizan de la manera que consideren mejor para lograr el objetivo del sprint y cumplir con la definición de terminado del mismo. Recordemos que esta definición de terminado siempre está alineada con el negocio que se debe resolver con la solución en construcción.

1.10.3.8 Reunión diaria (daily scrum)

Durante el sprint todos los días, preferentemente a la misma hora, los miembros del equipo de desarrollo realizan una reunión diaria con un límite de tiempo (15 minutos o menos). Esta actividad de inspección y adaptación está referida a la práctica común de que todos se pongan de pie durante la reunión para ayudar a promover la brevedad de la misma.

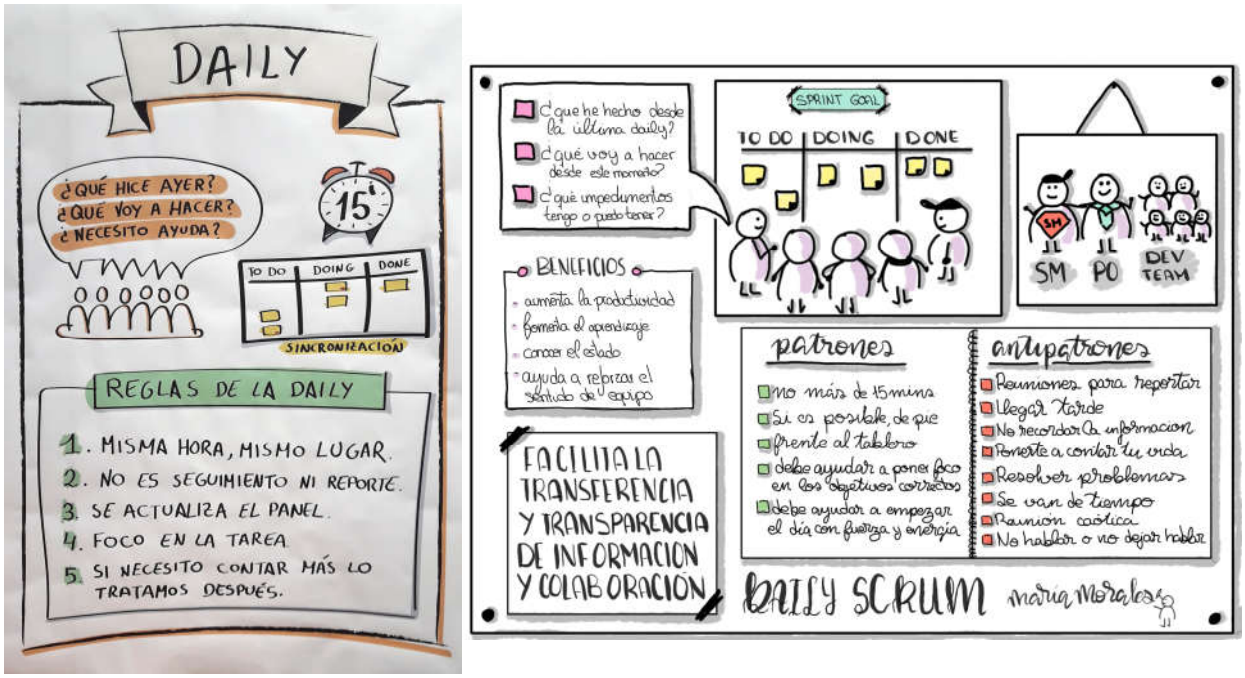

Figura 1.17: Conceptos de la Reunión Diaria

Un enfoque común para realizar la reunión diaria es que el Scrum Master facilite y cada miembro del equipo se turne para responder tres preguntas en beneficio de los demás miembros del equipo:

1. **Progreso:** ¿Qué logré desde la última reunión diaria? También permite mantener la evolución del burndown. (tablero de avance del sprint).
2. **Planificación:** ¿En qué planeo trabajar para la próxima reunión diaria?

3. **Adaptación:** ¿Cuáles son los obstáculos o impedimentos que me impiden progresar?

Al responder estas preguntas, todos comprenden el panorama general de lo que está ocurriendo, cómo están progresando hacia el objetivo del sprint, cualquier modificación que deseen realizar en sus planes para el trabajo del próximo día y qué problemas deben abordarse. La reunión diaria es esencial para ayudar al equipo de desarrollo a administrar el flujo de trabajo de forma rápida y flexible dentro de un sprint.

La reunión diaria NO es una actividad de resolución de problemas. En ocasiones, muchos equipos deciden hablar sobre los problemas después del scrum diario y lo hacen con un pequeño grupo de personas interesadas, normalmente para resolver conversaciones sobre funcionalidad o encarar deuda técnica. La reunión diaria tampoco es una reunión de estado tradicional, especialmente del tipo que históricamente convocan los gerentes de proyecto para que puedan obtener una actualización sobre el estado del proyecto. Sin embargo, puede ser útil para comunicar el estado de los elementos del backlog del sprint entre los miembros del equipo de desarrollo. Principalmente, es una actividad de inspección, sincronización y planificación diaria adaptativa que ayuda a un equipo autoorganizado a hacer mejor su trabajo.

1.10.3.9 Terminado

Scrum se refiere a los resultados del sprint como un incremento de producto potencialmente entregable, lo que significa que todo lo que el equipo de Scrum acordó hacer, realmente se hace de acuerdo con su definición acordada de terminado. Esta definición especifica el grado de confianza sobre el trabajo completado cuenta con la calidad óptima y es potencialmente desplegable para su uso. Por ejemplo, cuando se desarrolla software, una definición mínima de terminado debería generar una porción completa (utilizable por el usuario) de funcionalidad del producto que cuenta con su diseño, construcción, integración, pruebas y debidamente documentada.

Una definición agresiva de terminado permite que la empresa (cliente) decida en cada ciclo si se desea desplegar (implementar o lanzar) lo que se construyó a los clientes internos o externos (usuarios).

Para ser claros, “potencialmente desplegable” no significa que lo que se construyó deba ser entregado al usuario final. La puesta en producción es una decisión del negocio (y comercial), que con frecuencia está influenciada por cuestiones como "¿Tenemos suficientes funciones o suficiente flujo de trabajo del cliente para justificar una implementación?" o "¿Pueden nuestros clientes absorber otro cambio dado que les dimos un comunicado hace dos semanas?".

Potencialmente entregable se considera mejor como un estado de confianza sobre lo construido en el sprint, lo que realmente se hizo, lo que significa que no hay trabajo pendiente sin hacer materialmente importante (como pruebas importantes,

integración, performance, etc.). La calidad final del sprint implica no solo que se entrega un producto potencialmente desplegable sino además que se cumplieron condiciones de calidad que garantizan que el uso del mismo no generará un potencial riesgo al negocio, especialmente debido a fallas sustantivas en el producto.

Como cuestión práctica, con el tiempo algunos equipos pueden variar la definición de terminado. Por ejemplo, en las primeras etapas del desarrollo, tener características que se puedan enviar puede no ser económicamente factible o deseable (dada la naturaleza exploratoria del desarrollo temprano). En estas situaciones, una definición adecuada de terminado podría ser una parte de la funcionalidad del producto que sea lo suficientemente funcional y utilizable para generar comentarios que permitan al equipo decidir qué trabajo se debe hacer a continuación o cómo hacerlo.

1.10.3.10 Revisión del Sprint (Sprint Review)

Al final del sprint hay dos actividades adicionales de inspección y adaptación. Una se llama revisión del sprint.

El objetivo de esta actividad es inspeccionar y adaptar el producto que se está construyendo. Fundamental para esta actividad es la conversación que se lleva a cabo entre sus participantes, que incluyen el equipo Scrum, las partes interesadas, los patrocinadores, los clientes y los miembros interesados de otros equipos. La conversación se centra en revisar las funciones recién completadas en el contexto del esfuerzo de desarrollo general. Todos los asistentes obtienen una visibilidad clara de lo que está ocurriendo y tienen la oportunidad de ayudar a guiar el próximo desarrollo para garantizar que se cree la solución más adecuada para el negocio.

Una revisión exitosa da como resultado un flujo de información bidireccional. Las personas que no están en el equipo de Scrum se sincronizan con el esfuerzo de desarrollo y ayudan a guiar su dirección. Al mismo tiempo, los miembros del equipo de Scrum obtienen una apreciación más profunda del lado comercial y de marketing de su producto al recibir comentarios frecuentes sobre la convergencia del producto hacia clientes o usuarios conformes. Por lo tanto, la revisión del sprint representa una oportunidad programada para inspeccionar y adaptar el producto.

Como cuestión de práctica, las personas ajenas al equipo Scrum pueden realizar revisiones de funciones dentro del sprint y proporcionar comentarios para ayudar al equipo Scrum a alcanzar mejor su objetivo del sprint.

Esta fase incluye además de los aspectos funcionales de valor al negocio la garantía de la calidad del producto a través de los atributos de calidad propuestos en la visión del proyecto. Los atributos de calidad implican todos los aspectos No Funcionales definidos para el proyecto, que como todo desarrollo de software son una oportunidad de mirada innovadora y diferenciadora de otros productos similares. Es importante tener presente que en el desarrollo de software no sólo juegan las

definiciones funcionales del negocio sino cómo realizarlas contando con una mirada sobre la innovación tecnológica al resolver el problema del negocio planteado.

1.10.3.11 Retrospectiva (Sprint Retrospective)

La segunda actividad de inspección y adaptación al final del sprint es la retrospectiva del sprint. Esta actividad ocurre con frecuencia después de la revisión del sprint y antes de la planificación del próximo sprint.

Mientras que la revisión del sprint es un momento para inspeccionar y adaptar el producto, la retrospectiva del sprint es una oportunidad para inspeccionar y adaptar el proceso. Durante la retrospectiva del sprint, el equipo de desarrollo, Scrum Master y el dueño del producto se reúnen para analizar qué funciona y qué no funciona con Scrum y las prácticas y técnicas asociadas. El enfoque está en la mejora continua del proceso necesaria para ayudar a que un buen equipo Scrum se vuelva excelente. Al final de una retrospectiva de sprint, el equipo Scrum debería haber identificado y comprometido con un número práctico de acciones de mejora de procesos que llevará a cabo el equipo Scrum en el próximo sprint.

Es importante para realizar la actividad de retrospectiva que la misma esté debidamente preparada, en general por el Scrum Master que propone distintas técnicas de reuniones para evaluar, diagnosticar, promover y resolver las mejoras deseadas al proceso. En equipos de desarrollo con mayor nivel de madurez, las reuniones pueden ser organizadas por algún integrante del equipo, esto mejora sensiblemente la performance y los resultados de las reuniones debidos a la óptica del compromiso del equipo al realizar y proponer las mejoras. Existe una buena cantidad de técnicas de reuniones grupales orientadas a las retrospectivas que los equipos Scrum deben ir incorporando. En particular Scrum no propone ninguna de ellas en particular.

Una vez que se completa la retrospectiva del sprint, todo el ciclo se repite nuevamente, comenzando con la siguiente sesión de planificación del sprint, que se lleva a cabo para determinar el conjunto de trabajo actual de mayor valor en el que se debe enfocar el equipo.

1.10.4 Extreme Programming

Extreme Programing definida en inglés por su creador Kent Beck generando la sigla y marca registrada XP como se conoce a esta metodología ágil.

En términos generales no existen grandes diferencias con Scrum de hecho se puede mirar a Scrum como un marco de gestión de proyectos ágiles y a XP como un conjunto de prácticas y técnicas orientadas a garantizar la calidad del resultado de los productos elaborados.

En general se recomienda la utilización combinada de ambas metodologías en los proyectos.

En palabras del propio Kent Beck se reconocen y distinguen fundamentalmente los siguientes conceptos cómo destacados:

- Retroalimentación temprana, concreta y continua de ciclos cortos.
- Enfoque de planificación incremental, que genera rápidamente un plan general que se espera evolucione a lo largo de la vida del proyecto.
- Capacidad para programar de manera flexible la implementación de la funcionalidad, respondiendo a las necesidades del negocio aún cuando cambien.
- Generación de confianza mediante la ejecución de pruebas automatizadas escritas por programadores y clientes para monitorear el progreso del desarrollo, permitir que el sistema evolucione y detectar defectos de forma temprana.
- Basado en la comunicación oral, las pruebas y el código fuente para comunicar la estructura y la intención del sistema.
- garantía de calidad a través de un proceso de diseño evolutivo que dura tanto como dura el sistema.
- Estrecha colaboración de programadores con habilidades técnicas acordes a los proyectos.

XP es una disciplina porque hay ciertas actividades y premisas que se deben garantizar y realizar en XP. Por ejemplo, no se puede elegir si se escribirán o no pruebas automatizadas; si no se hacen, no se puede considerar que se está bajo la metodología XP.

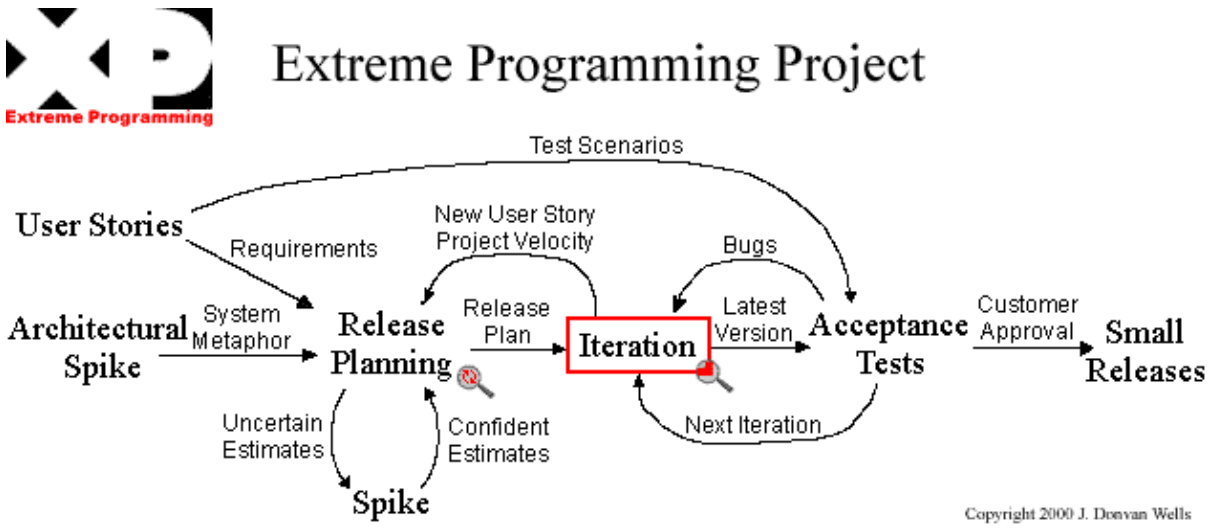

Figura 1.18 - Marco conceptual de la metodología eXtreme Programming

Como se puede observar en la Figura 1.17: Marco conceptual de la metodología eXtreme Programming se puede observar que no existen grandes diferencias con el marco definido por Scrum.

1.10.4.1 Roles

A diferencia de Scrum que plantea que no exista una división de roles dentro del equipo, eXtreme Programing reconoce la división de capacidades de las personas de acuerdo a la capa en la que se desarrolla. eXtreme Programing reconoce los siguientes roles:

Programador:

El programador es el corazón de XP. Para XP el programador debe contar con habilidades que se deben destacar respecto de otros estilos de desarrollo. Algunas de estas habilidades requieren estar orientadas a las capacidades técnicas de los individuos, contando con apego a la calidad de programación, buenas prácticas y al código limpio, comprendiendo los principios de deuda técnica y ser capaz de refactorizar, para ello es importante que el programador cuente con la capacidad de escribir y de realizar pruebas unitarias del código, lo que, al igual que la refactorización, requiere gusto y juicio para que se aplique bien.

El equipo de programación tiene que estar dispuesto a dejar de lado el sentimiento de propiedad individual de una parte del sistema a favor de la propiedad compartida de todo el sistema (code ownership y whole team). Si alguien cambia el código que escribiste, en cualquier parte del sistema, se debe confiar en los cambios y aprender de ellos.

El equipo de programadores debe estar capacitado para reconocer sus temores y debilidades, para aprender de ellos y poder modificarlos. algunos de los miedos y conocidos son:

- Miedo de parecer tonto
- Miedo de ser considerado inútil
- Miedo a volverse obsoleto
- Miedo de no ser lo suficientemente bueno

Por ello es importante destacar los valores de XP entre ellos el coraje, sin el coraje, XP simplemente no funciona. El equipo se pasaría todo el tiempo tratando desesperadamente de no fallar. En cambio, si está dispuesto, con la ayuda necesaria, a reconocer estos medios y sus debilidades. Básicamente las metodologías ágiles se reconocen en el empirismo, el aprendizaje contínuo, la detección temprana de equivocaciones y su pronta corrección.

Cliente

El cliente es la otra mitad de la dualidad esencial de XP. El programador sabe programar. El cliente sabe qué programar (el negocio). Ambos programadores y clientes basados en la relación empírica de la solución a desarrollar aprenden mutuamente y de forma colaborativa el uno del otro.

Al igual que los programadores, los integrantes del cliente requieren habilidades especiales para que funcione XP. Hay habilidades particulares que deben ser aprendidas antes de comenzar el proyecto cómo escribir buenas historias de

usuario y contar con una actitud proactiva que hará exitoso el proyecto. Sin embargo, sobre todo, el cliente debe sentirse cómodo influyendo en el proyecto sin poder controlarlo. Las fuerzas fuera de su control darán forma a lo que realmente se construye tanto como las decisiones que tome. Los cambios en las condiciones del negocio, la tecnología, la composición y la capacidad del equipo, todo esto tiene un gran impacto en el producto resultante (software) que se entrega.

En el marco de XP el cliente deberá tomar decisiones. Esta habilidad resulta muchas veces la más compleja para estructuras tradicionales de desarrollo de sistemas. Muchos de ellos están acostumbrados a que el área de tecnología no entregue la mitad de lo que prometió, y que lo que entregue sea la mitad de malo (pobre en calidad). Esto puede llevar al cliente a nunca ceder ni un centímetro a la tecnología de la información, ya que de todos modos se sentirán decepcionados. XP no funciona con un cliente así. Si es cliente de XP, el equipo necesita que diga con confianza: "Esto es más importante que eso", "Con ésta parte de esta historia es suficiente", "Estas historias juntas son suficientes". Y cuando los tiempos se ponen difíciles, y siempre se ponen difíciles, el equipo necesita que se pueda cambiar de opinión. "Bueno, supongo que no tenemos que tener esto absolutamente hasta el próximo trimestre". Ser capaz de tomar decisiones como estas a veces salvará el proyecto a su equipo y reducirá su estrés lo suficiente como para que se pueda hacer lo mejor para el negocio.

Otra de las habilidades requeridas para los clientes XP es la escritura de pruebas funcionales. Esta habilidad puede ser adquirida durante la ejecución del proyecto con la asistencia de entrenamiento. Se tendrá que trabajar en estrecha colaboración con el equipo para saber qué tipo de cosas son útiles de probar y qué tipo de pruebas son redundantes. Algunos equipos incluso pueden asignarle ayuda técnica para elegir, escribir y ejecutar las pruebas. Su objetivo es escribir pruebas que le permitan decir: "Bueno, si las pruebas se ejecutan y pasan correctamente, entonces se puede confiar en la correctitud del sistema respecto de las definiciones brindadas por el cliente".

Tester

Dado que gran parte de la responsabilidad de las pruebas recae sobre los hombros de los programadores, el papel de tester en un equipo de XP se centra principalmente en el cliente. El equipo es responsable de colaborar con el cliente para elegir y escribir pruebas funcionales. Si las pruebas funcionales no forman parte del paquete de integración, usted es responsable de ejecutar las pruebas funcionales regularmente y publicar los resultados en un lugar accesible por todas las personas del proyecto.

Un tester en XP no es una persona aislada, dedicada a romper el sistema y humillar a los programadores. Sin embargo, alguien tiene que ejecutar todas las pruebas con regularidad (si no puede ejecutar su unidad y las pruebas funcionales

juntas), transmitir los resultados de las pruebas y asegurarse de que las herramientas de prueba funcionen bien.

Rastreador (Tracker)

El rastreador tiene la misión de ser la conciencia del equipo.

Parte de las habilidades requeridas de los rastreadores es colaborar con el equipo para cubrir la retroalimentación que permita contar al proyecto con la información oportuna para la toma de decisiones. Se deben garantizar las estimaciones para luego evaluar cómo la realidad se ajusta a las premisas realizadas. Por lo tanto, el rastreador es de vital importancia para cerrar el ciclo de retroalimentación. La próxima vez que el equipo haga estimaciones, debe poder decir: "Dos tercios de nuestras estimaciones la última vez fueron al menos un 50 % demasiado altas". De forma individual, debe poder decir: "Las estimaciones de su tarea son demasiado altas o demasiado bajas". Las próximas estimaciones que salgan todavía son responsabilidad de las personas que tienen que implementar lo que se está estimando, pero se les ha brindado la retroalimentación para que cuando salgan puedan ser mejores que la última vez. El rastreador es una pieza clave en el proceso de mejora requerido y planteado por las metodologías ágiles.

También es responsable de mantener la vista en el panorama general. A la mitad de una iteración, debería poder decir al equipo si va a lograrlo, si se sigue el curso actual o si necesita cambiar algo. Un par de iteraciones en un cronograma de compromiso debería poder decirle al equipo si van a hacer el próximo lanzamiento sin hacer grandes cambios.

El traqueador es el historiador del equipo. Mantiene un registro de las puntuaciones de las pruebas funcionales. Mantiene un registro de los defectos informados, quién aceptó la responsabilidad de cada uno y qué casos de prueba se agregaron en nombre de cada defecto. Gestiona la trazabilidad del proyecto.

La habilidad que más se necesita cultivar en un rastreador, es la capacidad de recopilar la información que se necesita sin perturbar el proceso de desarrollo más de lo necesario.

Entrenador (Coach)

El entrenador es un rol homónimo al del Scrum Master en la metodología Scrum. El entrenador es responsable del proceso en su totalidad. Evalúa cuando las personas se están desviando del proceso y se lo comunica al equipo. Mantiene la calma cuando todos los demás entran en pánico, recordando que en las próximas dos semanas solo puede hacer el trabajo de dos semanas o menos, y el valor de dos semanas es suficiente o no lo es.

Si bien todos los integrantes de un equipo XP son responsables de comprender su aplicación, el rastreador es responsable de comprenderlo con mayor profundidad: qué prácticas alternativas podrían ayudar al conjunto de problemas actual; cómo otros

equipos están usando XP; cuáles son las ideas detrás de XP; y cómo se relacionan con la situación actual.

Lo mayor complejidad para ser entrenador es trabajar indirectamente sobre los problemas. Si se ve un error de diseño, primero se debe decidir si es lo suficientemente importante como para intervenir. Cada vez que se induce al equipo a actuar de determinada manera se los hace menos autosuficientes. Demasiada dirección provoca perder la capacidad de trabajar sin el entrenador (autosuficiente), lo que resulta en una menor productividad, menor calidad y menor moral.

El papel de entrenador disminuye a medida que el equipo madura. De acuerdo con los principios de control distribuido y responsabilidad aceptada, "el proceso" debe ser responsabilidad de todos.

Consultor

Los proyectos realizados bajo la metodología de XP en general no generan especialistas. Debido a que se promueve a que todos se emparejan con todos los demás, y las parejas flotan tanto, y cualquiera puede aceptar la responsabilidad de una tarea si así lo desea, hay pocas posibilidades de que se desarrollen huecos donde solo una o dos personas entienden exclusivamente de ese negocio o parte del sistema.

Esta es una fortaleza, porque el equipo es extremadamente flexible, pero también es una debilidad, porque de vez en cuando el equipo necesita un conocimiento técnico profundo. El énfasis en la simplicidad del diseño reduce la aparición de la necesidad de un experto, pero esto sucederá de vez en cuando.

Cuando esto sucede, el equipo necesita la participación de un consultor. Lo más probable es que, si el consultor, no está acostumbrado a trabajar dentro de la metodología ágil, es probable que vea lo que hace el equipo con cierto escepticismo. Para ello el equipo debe tener muy claro el problema que debe resolver el consultor.

Es importante en este proceso que el consultor no resuelva el problema a solas, es necesaria la participación del equipo y generar los mecanismos de transferencia de las capacidades técnicas buscadas en el consultor. De esta forma se evitará que el mismo tenga que ser llamado recurrentemente para resolver el mismo problema técnico.

Gran Jefe (Big Boss)

El rol de gran jefe sumado al del cliente es lo que en Scrum denominamos Dueño de Producto.

Al igual que el dueño del producto el gran jefe debe articular entre las necesidades del negocio, las capacidades del equipo y los recursos disponibles.

Al igual que el equipo el gran jefe requiere de coraje para acompañar las decisiones innovadoras propuestas por el equipo. Encarar en proyecto de desarrollo de sistemas informáticos, no pensando en la innovación en los procesos que permite la aplicación de nuevas tecnologías es un error de concepto que se comete muy

frecuentemente. Es importante la confianza y la transparencia entre todos los integrantes del equipo para asumir y provocar correctamente la aplicación de las innovaciones requeridas. Aquí es donde el rol del gran jefe se vuelve necesario para el proyecto. El equipo debe sentir su acompañamiento y comprensión en la aplicación de nuevas tecnologías para resolver su negocio.

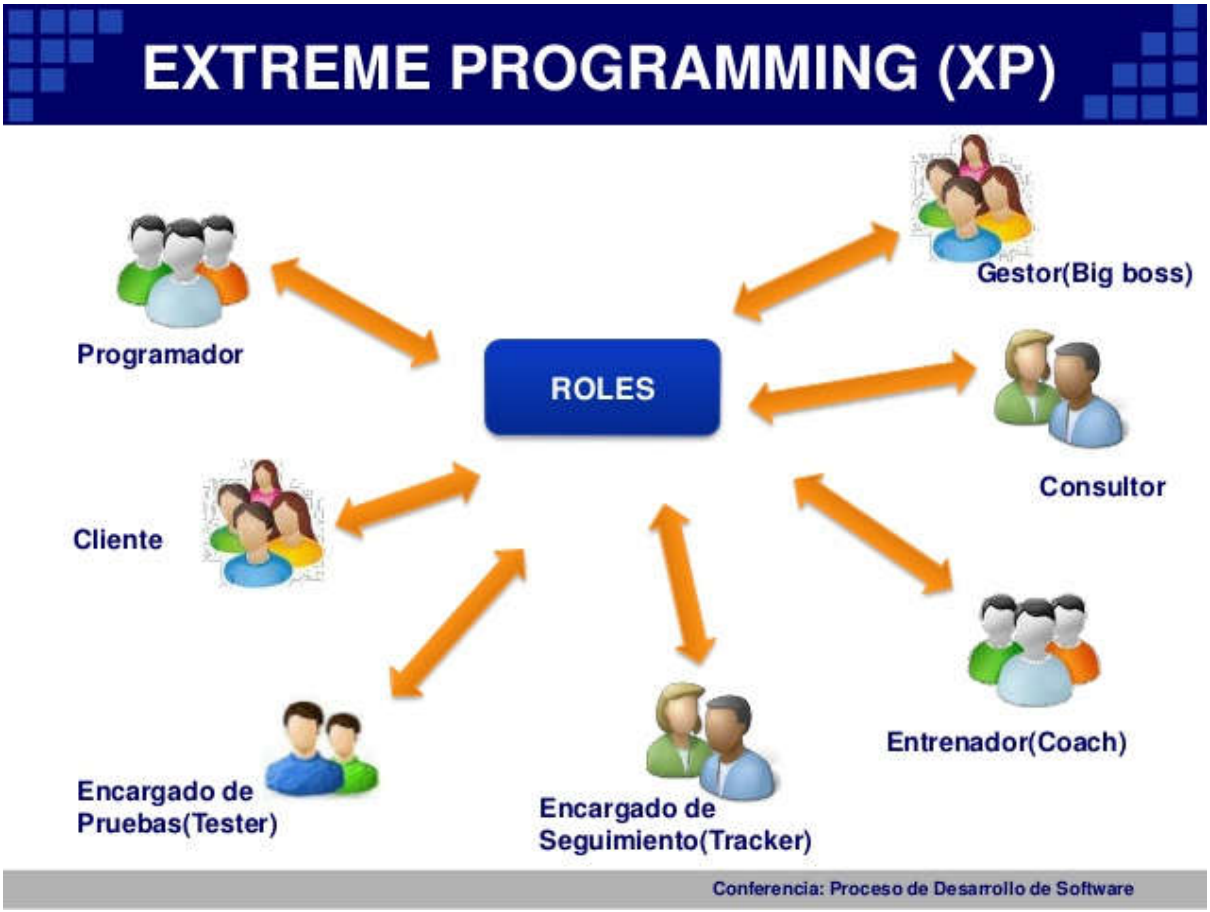

Figura 2.19 - resumen de los roles de eXtreme Programming

1.10.4.2 Los 5 Valores

- Simplicidad
- Comunicación
- Retroalimentación (feedback)
- Respecto
- Coraje

Simplicidad

La simplicidad es la base de la programación extrema. Se simplifica el diseño para agilizar el desarrollo y facilitar el mantenimiento. Un diseño complejo del código junto a sucesivas modificaciones por parte de diferentes desarrolladores hace que la complejidad aumente exponencialmente.

Para mantener la simplicidad es necesaria la **refactorización del código**, ésta es la manera de mantener el código simple a medida que crece.

Comunicación

La comunicación se realiza de diferentes formas. Para los programadores el código comunica mejor cuanto más simple sea. Si el código es complejo hay que esforzarse para hacerlo inteligible. El código autodocumentado es más fiable que los comentarios ya que éstos últimos pronto quedan desfasados con el código a medida que es modificado. Debe comentarse sólo aquello que no va a variar, por ejemplo, el objetivo de una clase o la funcionalidad de un método.

Las pruebas unitarias son otra forma de comunicación ya que describen el diseño de las clases y los métodos al mostrar ejemplos concretos de cómo utilizar su funcionalidad. Los programadores se comunican constantemente gracias a la programación por parejas. La comunicación con el cliente es fluida ya que el cliente forma parte del equipo de desarrollo. El cliente decide qué características tienen prioridad y siempre debe estar disponible para solucionar dudas.

Retroalimentación

Al estar el cliente integrado en el proyecto, su opinión sobre el estado del proyecto se conoce en tiempo real.

Al realizarse ciclos muy cortos tras los cuales se muestran resultados, se minimiza el tener que rehacer partes que no cumplen con los requisitos y ayuda a los programadores a centrarse en lo que es más importante.

Considérense los problemas que derivan de tener ciclos muy largos. Meses de trabajo pueden tirarse por la borda debido a cambios en los criterios del cliente o malentendidos por parte del equipo de desarrollo. El código también es una fuente de retroalimentación gracias a las herramientas de desarrollo. Por ejemplo, las pruebas unitarias informan sobre el estado de salud del código. Ejecutar las pruebas unitarias frecuentemente permite descubrir fallos debidos a cambios recientes en el código.

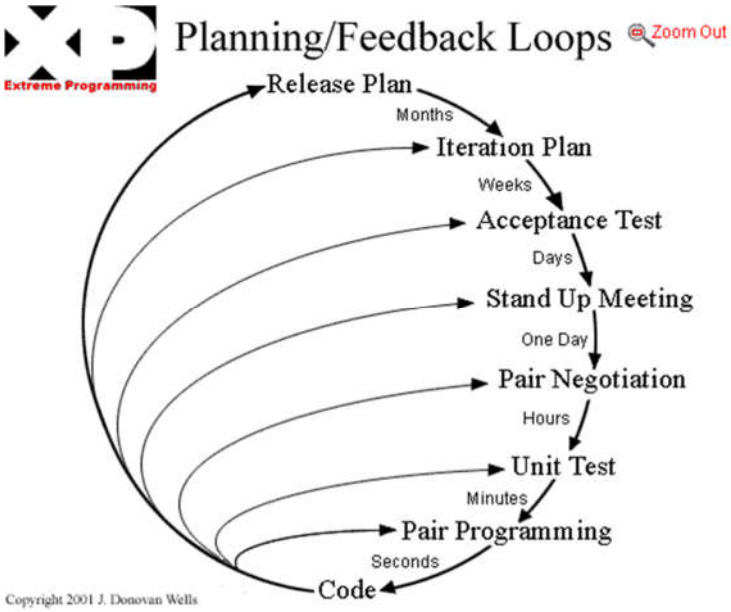

Figura 1.20 - Las distintas capas de retroalimentación

Respeto

El respeto se manifiesta de varias formas. Los miembros del equipo se respetan los unos a los otros, porque los programadores no pueden realizar cambios que hacen que las pruebas existentes fallen o que demore el trabajo de sus compañeros. Los miembros respetan su trabajo porque siempre están luchando por la alta calidad en el producto y buscando el diseño óptimo o más eficiente para la solución a través de la refactorización del código. Los miembros del equipo respetan el trabajo del resto no haciendo menos a otros, una mejor autoestima en el equipo eleva su ritmo de producción.

Coraje

Muchas de las prácticas implican valentía. Una de ellas es siempre diseñar y programar para hoy y no para mañana. Esto es un esfuerzo para evitar empantanarse en el diseño y requerir demasiado tiempo y trabajo para implementar el resto del proyecto. La valentía le permite a los desarrolladores que se sientan cómodos con reconstruir su código cuando sea necesario. Esto significa revisar el sistema existente y modificarlo si con ello los cambios futuros se implementaran más fácilmente. Otro ejemplo de valentía es saber cuándo desechar un código: valentía para quitar código fuente obsoleto, sin importar cuanto esfuerzo y tiempo se invirtió en crear ese código. Además, valentía significa persistencia: un programador puede permanecer sin avanzar en un problema complejo por un día entero, y luego lo resolverá rápidamente al día siguiente, sólo si es persistente.

1.10.4.3 Los Principios

Retroalimentación rápida: la psicología del aprendizaje enseña que el tiempo entre una acción y su retroalimentación es fundamental para el aprendizaje. Los

experimentos con animales muestran que incluso las pequeñas diferencias en el momento de la retroalimentación dan como resultado grandes diferencias en el aprendizaje. Unos segundos de retraso entre el estímulo y la respuesta y el ratón no aprende que el botón rojo significa comida. Entonces, uno de los principios es obtener retroalimentación, interpretarla y volver a poner lo aprendido en el sistema lo más rápido posible. La empresa aprende cómo el sistema puede contribuir mejor y retroalimenta ese aprendizaje en días o semanas en lugar de meses o años. Los programadores aprenden cuál es la mejor manera de diseñar, implementar y probar el sistema, y retroalimentan ese aprendizaje en segundos o minutos en lugar de días, semanas o meses.

Asumir simplicidad: Se debe tratar cada problema como si pudiera ser resuelto con una simplicidad ridícula. El tiempo que se ahorre en el 98% de los problemas para los que esto es cierto brindará recursos para aplicar al otro 2%. En muchos sentidos, este es el principio más difícil de asumir para los programadores. Tradicionalmente se dice que se debe planificar para el futuro, en cambio XP propone que se realice un buen trabajo (pruebas, refactorización, comunicación) para resolver el problema (trabajo) de hoy y se confíe en la capacidad para agregar complejidad en el futuro donde se lo necesite. La economía del software como opciones favorece este enfoque.

Cambio incremental: Si los grandes cambios se realizan todos a la vez simplemente no funcionarán. Cualquier problema se resuelve con una serie de cambios mínimos que marcan la diferencia.

Se encontrarán cambios incrementales aplicados de muchas maneras dentro de la metodología XP. El diseño cambia poco a poco. El plan cambia poco a poco. El equipo cambia poco a poco. Incluso la adopción de XP debe realizarse en pequeños pasos.

Aceptar el cambio: la mejor estrategia es la que conserva la mayor cantidad de opciones y, al mismo tiempo, resuelve el problema más apremiante.

Trabajo de calidad: a nadie le gusta trabajar descuidadamente. A todo el mundo le gusta hacer un buen trabajo. De las cuatro variables de desarrollo del proyecto (alcance, costo, tiempo y calidad), la calidad no es realmente una variable libre. Los únicos valores posibles son "excelente" e "increíblemente excelente", dependiendo de si hay vidas en juego o no. De lo contrario, no se disfruta del trabajo, si no se trabaja bien el proyecto no contará con la estabilidad requerida y menos aún podrá aceptar el cambio.

1.10.4.4 Prácticas

Juego de planificación: determina rápidamente el alcance de la próxima versión combinando las prioridades del negocio y las estimaciones técnicas. A medida que la realidad supere el plan, se debe actualizar el mismo.

Entregas (despliegues) pequeños: poner un sistema simple en producción rápidamente y luego se debe ir lanzando y desplegando nuevas versiones en ciclos muy cortos.

Metáfora: guiar el desarrollo del sistema bajo construcción identificando la modularización del negocio con una simple historia compartida de cómo funciona todo el sistema. Palabras “fuerza” que identifican partes completas del sistema.

Diseño simple: el sistema debe ser diseñado de la manera más simple posible en un momento dado. La complejidad adicional debe ser eliminada tan pronto como se descubre.

Pruebas: los programadores escriben continuamente pruebas unitarias, que deben ejecutarse sin problemas para que el desarrollo continúe. Los clientes escriben pruebas que demuestran que las funciones están terminadas. garantía de correctitud del sistema.

Refactorización: los programadores reestructuran el sistema sin cambiar su comportamiento para eliminar la duplicación, mejorar la comunicación, simplificar o agregar flexibilidad.

Programación en pareja: todo el código de producción se escribe con dos programadores en una máquina.

Propiedad colectiva: cualquiera puede cambiar cualquier código en cualquier lugar del sistema en cualquier momento.

Integración continua: Permite integrar y crear el sistema muchas veces al día, cada vez que se complete una tarea.

40 horas por semana: por regla general, no trabaje más de 40 horas a la semana. Nunca trabaje horas extras por más de una semana consecutiva. El trabajo extra debe ser una medida excepcional. El proyecto debe fluir para lograr la paz sostenible.

Cliente disponible: Se debe incluir un usuario real en vivo conviviendo con el equipo, disponible a tiempo completo para responder preguntas.

Estándares de codificación: los programadores escriben todo el código de acuerdo con reglas que enfatizan la comunicación a través del código. Los programadores deben promover las buenas prácticas impulsadas por cada lenguaje de programación o herramienta de desarrollo. Implica tener superada la curva de aprendizaje tecnológica o disponer de recursos afines para superarla dentro del proyecto.

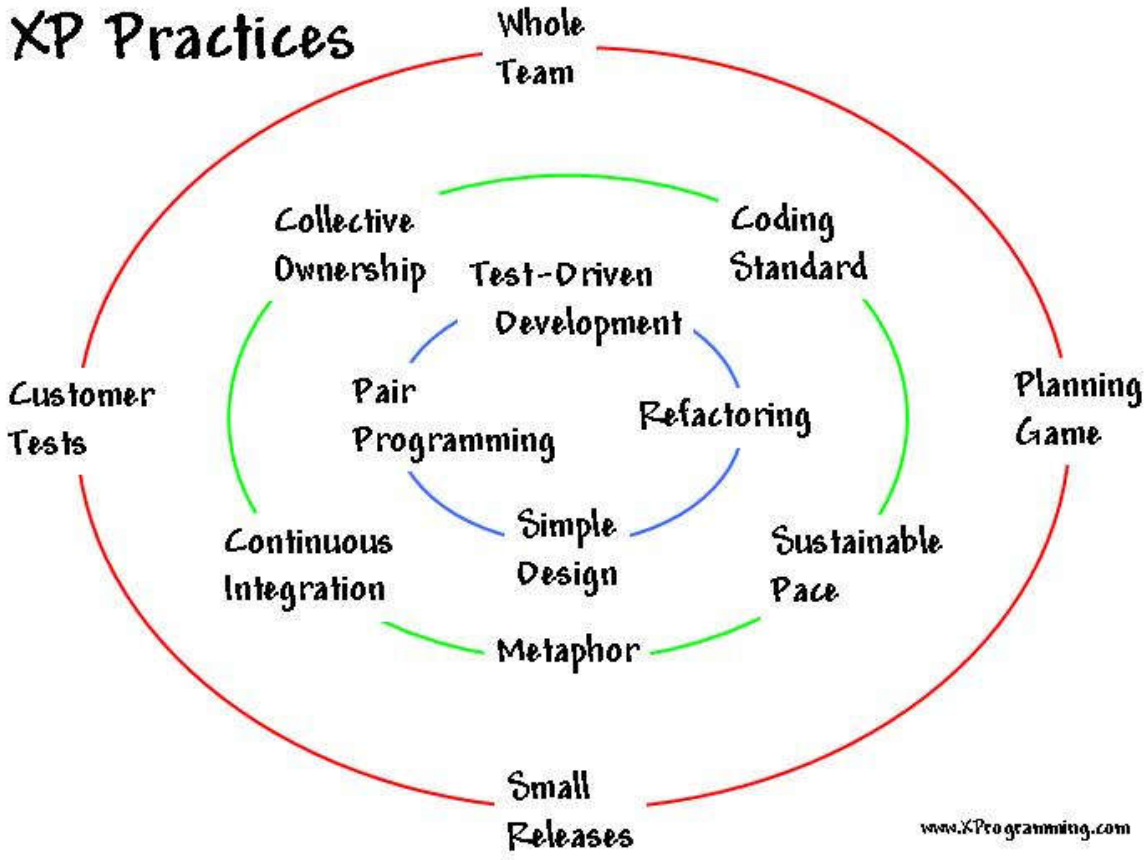

Figura 1.21 - Resumen de prácticas XP

Mover a la gente alrededor del código: Se debe mover a las personas para evitar pérdidas graves de conocimiento y cuellos de botella en la codificación. Si solo una persona en su equipo puede trabajar en un área determinada y esa persona se va o simplemente tiene mucho que hacer, verá que el progreso del proyecto se reduce considerablemente.

Velocidad del proyecto: La velocidad del proyecto (o simplemente la velocidad) es la medida de cuánto trabajo se está realizando en el proyecto para cada iteración. Para medir la velocidad del proyecto, simplemente se deben sumar las estimaciones de las historias de usuario que se terminaron durante la iteración. También se debe sumar las estimaciones de las tareas finalizadas durante la iteración. Ambas medidas se utilizan para la planificación de iteraciones.