# Unidad 2: Gestión Lean-Ágil de Productos de Software

## El proceso de software:

Diagrama

Descripción generada automáticamente

* Conjunto estructurado de actividades para desarrollar un sistema de software.
* Estas actividades varían dependiendo de la organización y el tipo de sistema que debe desarrollarse.
* Debe ser explícitamente modelado si va a ser administrado.

### Definición de un proceso de software:

* Proceso: es la secuencia de pasos ejecutados para un propósito dado.
* Proceso de software: es un conjunto de actividades, métodos, prácticas, y transformaciones que la gente usa para desarrollar o mantener software y sus productos asociados.

Diagrama

Descripción generada automáticamente

Se tienen tres factores determinantes del proceso:

* Procedimientos y métodos: se deben tener los procedimientos y métodos a utilizar definidos y escritos.
* Personas motivadas, capacitadas y con habilidades: las personas deben estar capacitadas y con habilidades para realizar sus tareas asignadas de la manera correcta, y deben estar motivadas para lograr una mayor eficiente.
* Herramientas y equipos: materiales necesarios para llevar a cabo el proceso que nos permiten que las entradas se transformen en salidas.

### Estimaciones de software:

Las estimaciones son difíciles porque derivan de los requerimientos.

Hay un conjunto de cosas que hay que estimar y en un orden dado:

1. Tamaño. ¿Qué tamaño tiene eso que quiero construir?. En el desarrollo tradicional podemos medir el tamaño con líneas de código (fue lo primero que se eligió para estimar tamaño, no sirve), alcances, requerimientos, casos de uso. Lo que se está usando en estos últimos tiempos son los casos de uso por complejidad para el desarrollo tradicional.
2. Esfuerzo. Se mide en horas/personas lineales (asumo cuantas horas necesito para construir eso en base al tamaño teniendo en cuenta que estoy solo). El esfuerzo se deriva del tamaño.
3. Calendario/tiempo. Se tiene en cuenta los días y cantidad de horas que se trabaja semanalmente, la cantidad de personas, si se pueden realizar tareas simultaneas.
4. Costo. Es una variable económica. La valorización se hace al final. Es el costo del esfuerzo, es el costo más representativo del proyecto del software, es el 80% del costo total del software.
5. Recursos críticos.

Son las estimaciones en el contexto del plan de proyecto.

### Riesgo:

El riesgo es probabilidad de la ocurrencia de una pérdida o daño que impacte en el cumplimiento del proyecto. Siempre el riesgo tiene implicada probabilidad, si hay certeza es un problema no un riesgo.

El riesgo se mide por 2 variables o dimensiones:

* Probabilidad.
* Impacto: es la que mide cuanto daño puede ocasionar si ocurre.

Probabilidad x impacto = exposición.

La exposición es la forma que tenemos de cuantificar los riesgos, comparar y decidir de cuales nos vamos a ocupar.

Los proyectos más importantes son:

* Que cambien los requerimientos.
* Que se vaya alguien vital para el proyecto antes que termine el mismo.
* Riesgos que tengan que ver con la tecnología, probabilidad que hoy ante y que mañana no.

#### Gestión de riesgos:

Diagrama

Descripción generada automáticamente

Identificar y gestionar riesgos a través de todas las fases del proyecto.

Gestión de riesgos reactiva: espera a que el riesgo se transforme en problemas, y recién ahí actuar.

Gestión de riesgos proactiva: se anticipa a los riesgos, actúa antes que ocurran realizando un plan de mitigación.

### Recursos:

Los recursos críticos son los que normalmente es escaso, hay más demanda que oferta.

## Métricas de software:

* El dominio de las métricas del software se divide en:
  + Métricas de proceso.
  + Métricas de proyecto.
  + Métricas de producto.
* Las métricas del proyecto se consolidan para crear métricas de proceso que sean públicas para toda la organización del software.

Diagrama

Descripción generada automáticamente

### Métricas básicas para un proyecto de software:

* Tamaño del producto.
* Esfuerzo.
* Tiempo (calendario).
* Defectos.

#### Métricas por rol?

**Desarrollador:**

1. Esfuerzo.
2. Esfuerzo y duración estimada y actual de una tarea.
3. % de cobertura por el unit test.
4. Número y tipo de defectos encontrados en el unit test.
5. Número y tipo de defectos encontrados en revisión por pares.

**Organización:**

1. Tiempo calendario.
2. Performance actual y planificada de esfuerzo.
3. Performance actual y planificada de presupuesto.
4. Precisión de estimaciones en Schedule y esfuerzo.
5. Defectos en release.

**Equipo de desarrollo:**

1. Tamaño del producto.
2. Duración estimada y actual entre los hitos más importantes.
3. Niveles de staffing actual y estimados.
4. Número de tareas planificadas y completadas.
5. Distribución del esfuerzo.
6. Status de requerimientos.
7. Volatibilidad de requerimientos.
8. Número de defectos encontrados en la integración y prueba de sistemas.
9. Número de defectos encontrados en peer reviews.
10. Satus de distribución de defectos.
11. % de test ejecutados.

### Tres factores para el éxito de un proyecto:

* Monitoreo y feedback.
* Tener una misión/objetivo claro.
* Comunicación.

### Causas de fracasos en proyectos:

* Fallas al definir el problema.
* Planificar basado en datos insuficientes.
* La planificación la hizo el grupo de planificaciones.
* No hay seguimiento del plan de proyecto.
* Plan de proyecto pobre en detalles.
* Planificación de recursos inadecuada.
* Las estimaciones se basaron en “supuestos” sin consultar datos históricos.
* Nadie estaba a cargo.

## Manifiesto ágil:

Valores del manifiesto ágil:

1. Valoramos más el software funcionando que documentación exhaustiva.
2. Valoramos más los individuos e interacciones entre los individuos que procesos y herramientas.
3. Valoramos más el cambio (aceptar que los requerimientos pueden cambiar) que el cumplimiento de un plan.
4. Valoramos más la colaboración con el cliente que la negociación contractual.

El agilismo no implica que sea rápida, no implica que no se documente.

El agilismo para funcionar usa procesos empíricos.

### Los 12 principios del manifiesto ágil:

1. Satisfacer al cliente a través de releases frecuentes.
2. Aceptar que los requisitos cambien, aún en etapas finales.
3. Entregar software funcional frecuentemente.
4. Técnicos y no técnicos trabajando juntos todo el proyecto. Los responsables de negocios, diseñadores y desarrolladores deben trabajar juntos día a día durante el proyecto.
5. Desarrollamos proyectos en torno a individuos motivados.
6. El método más eficiente de comunicar información es conversaciones cara a cara.
7. El software funcionando es la principal medida de éxito.
8. Los procesos ágiles promueven el desarrollo sostenible.
9. La atención continua a la excelencia técnica y al buen diseño mejora la agilidad.
10. La simplicidad 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 y de acuerdo a esto ajustan su comportamiento.

### ¿Qué es ágil?

No es una metodología o un proceso. Ágil es una ideología con un conjunto definido de principios que guían el desarrollo del producto. Es una filosofía, es un pensamiento, es algo que apunta a la cultura de las personas que lo adoptan.

Valores de los equipos ágiles:

* Planificación continua, multi-nivel.
* Facultados, auto-organizados, equipos completos.
* Entregas frecuentes, iterativas y priorizadas.
* Prácticas de ingeniería disciplinadas.
* Integración continua.
* Testing concurrente.

Ágil es un balance entre ningún proceso y demasiado proceso. La diferencia inmediate es la exigencia de una menor cantidad de documentación, sin embargo, no es lo más importante.

* Los métodos ágiles son adaptables en lugar de predictivos.
* Los métodos ágiles son orientados a la gente en lugar de orientados al proceso.

#### Algunos frameworks ágiles:

Diagrama

Descripción generada automáticamente

### Requerimientos en agile:

* Usar el “valor” para construir el producto correcto.
* Usar historias y modelos para mostrar que construir.
* Determinar que es “sólo lo suficiente”.

Implica aplicar inspección y adaptación, es decir iterar, por ser un proceso empírico.

### Gestión ágil de requerimientos de software:

Los requisitos cambiantes son una ventaja competitiva si puede actuar sobre ellos.

Imagen que contiene Escala de tiempo

Descripción generada automáticamente

#### Tipos de requerimientos:

* Requerimiento de negocio: disminuir x% de tiempo invertido en procesos manuales relacionados con atención al cliente.
* Requerimiento de usuario: realizar consultas en línea del estado de cuenta de los clientes.
* Requerimiento funcional: generar reporte de saldos de cuenta. Recibir notificaciones por mail.
* Requerimiento no funciona: formato del reporte PDF. Cumplir niveles de seguridad para credenciales de usuarios según la ley bancaria 9999XX.
* Requerimiento de implementación: servidores en la nube.

### Tradicional vs ágil:

* Los cambios son la única constante.
* Stakeholders: no son todos los que están.
* Siempre se cumple eso de que: “el usuario dice lo que quiere cuando recibe lo que pidió”.
* No hay técnicas ni herramientas que sirvan para todos los casos.
* Lo importante no es entregar una salida, un requerimiento, lo importante es entregar, un resultado, una solución de “valor”.

### Principios ágiles relacionados a los requerimientos ágiles:

1. La prioridad es satisfacer al cliente a través de releases tempranos y frecuentes.
2. Recibir cambios de requerimientos, aún en etapas finales.

4. Técnicos y no técnicos trabajando juntos todo el proyecto.

6. El medio de comunicación por excelencia es cara a cara.

11. Las mejores arquitecturas, diseños y requerimientos emergen de equipos autoorganizados.

## Product Backlog:

Es un contenedor de ítems de product backlog, un ítem de producto backlog es una user story. Los ítems que tienen más prioridad son los de arriba, la prioridad no siempre se refiere a que sea más importante, también puede pasar que un ítem de product backlog sea menos importante, pero se requiera antes.

Es una cola o lista priorizada de ítems de product backlog (características de producto).

No puede haber tareas en un product backlog. Puede haber:

* Defectos.
* Deuda técnica.
* Épica.
* Temas.
* Spikes: aparecen cuando hay incertidumbre, cuando es tan grande que no sé porque falta información.

## User Stories:

Son requerimientos de usuario o de negocio, el nivel de abstracción que tienen las user stories es más alto que el de los casos de uso.

Es una descripción CORTA de una funcionalidad esperada del producto que tienen que satisfacer. Describen una necesidad.

Las ideas de las US tienen que venir de product owner.

Se las llama “stories” porque se supone que cuenta usted cuenta una historia. Lo que se escribe en la tarjeta no es importante, lo que usted habla, sí.

Partes que conforman una user story:

* Conversación: es lo que se habla con el cliente, no se guarda en ningún lado, pero a partir de la conversación se obtiene la información para la US.
* Tarjeta: es la parte visible de la US, es lo que efectivamente escribimos y se ingresa en el product backlog. La sintaxis recomendada para la Card responde a: who, what y why (sería el titulo siguiente).
* Confirmación: serían las pruebas de aceptación de usuario.

### Forma de expresar las historias de usuario:

Como <nombre del rol>, yo puedo <actividad> de forma tal que <valor de negocio que recibo>.

* Nombre del rol: representa quién está realizando la acción o quién recibe el valor de la actividad.
* Actividad: representa la acción que realizará el sistema.
* Valor de negocio que recibo: comunica porque es necesaria la actividad.

Ejemplo:

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Lo de arriba se llama Frase Verbal, es un resumen corto de la descripción de la user story.

### ¿Qué es una User Story?

Las user stories son multipropósito.

Las historias son:

* Una necesidad del usuario.
* Una descripción del usuario.
* Un ítem de planificación.
* Token para una conversación.
* Mecanismo para diferir una conversación.

### El Product Owner prioriza las historias en el producto backlog:

Interfaz de usuario gráfica

Descripción generada automáticamente

### Modelado de roles: tarjeta de rol de usuario:

Texto

Descripción generada automáticamente

#### Usuarios representantes (proxies):

Tipos de usuarios representantes:

* Gerentes de usuarios.
* Gerentes de desarrollo.
* Alguien del grupo de marketing.
* Vendedores.
* Expertos del dominio.
* Clientes.
* Capacitadores y personal de soporte.

No son ideales como los usuarios verdaderos, hay que evitarlos.

### Criterios de aceptación de historias de usuario:

* Define límites para usa user story (US).
* Ayuda a que los PO respondan lo que necesitan para que la US provea valor (requerimientos funcionales mínimos).
* Ayudan a que el equipo tenga una visión compartida de la US.
* Ayudan a desarrolladores y testers a derivar las pruebas.
* Ayudan a los desarrolladores a saber cuándo parar de agregar funcionalidad en una US.

Ejemplo:

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

#### ¿Cuáles son los criterios de aceptación buenos?

* Definir una intención, no una solución. Ejemplo: el usuario debe elegir al menos una cuenta para operar.
* Son independientes de la implementación.
* Relativamente de alto nivel, no es necesario que se escriba cada detalle.

#### ¿Los detalles dónde van?

Ejemplo de detalles:

* El encabezado de la columna se nombra “Saldo”.
* El formato del saldo es 999.999.999,99.
* Debería usarse una lista desplegable en lugar de un check box.

Estos detalles que son el resultado de las conversaciones con el PO y el equipo puede capturarlos en dos lugares:

* Documentación interna de los equipos.
* Pruebas de aceptación automatizadas.

### Pruebas de aceptación de historias de usuario:

* Expresan detalles resultantes de la conversación.
* Complementan la user story.
* Se escriben: “Probar… (pasa/no pasa).”
* Es un proceso de dos pasos:

1. Identificarlas al dorso de la US.
2. Diseñar las pruebas completas.

Ejemplos:

Imagen que contiene Texto

Descripción generada automáticamente

Texto, Tabla

Descripción generada automáticamente

### Invest model:

* Independent: calendarizables e implementables en cualquier orden.
* Negotiable: se expresa en términos de “que” no el “cómo”.
* Valuable: debe tener valor para el cliente.
* Estimatable: para ayudar al cliente a armar un ranking basado en costos.
* Small: debe ser “consumidas” en una iteración.
* Testable: demostrar que fueron implementadas.

### Algo más sobre las User Stories:

* No son especificaciones detalladas de requerimientos (como los casos de uso).
* Son expresiones de intención, “es necesario que haga algo como esto…”.
* No están detallados al principio del proyecto, elaborados evitando especificaciones anticipadas, demoras en el desarrollo, inventario de requerimientos y una definición limitada de la solución.
* Necesita poco o nulo mantenimiento y puede descartarse después de la implementación.
* Junto con el código, sirven de entrada a la documentación que desarrolla incrementalmente después.

## Estimaciones:

Una estimación es una predicción con el objetivo de predecir la completitud y administrar los riesgos de un proyecto.

* No ser dogmático sobre nada.
* Ser pragmático es la clave del éxito en casi todo.

**Tips:**

* Si las estimaciones se utilizan como compromisos son muy peligrosas y perjudiciales para cualquier organización.
* Lo más beneficioso en las estimaciones es el “proceso de hacerlas”.
* La estimación podría servir como una gran respuesta temprana sobre si el trabajo planificado es factible o no.
* La estimación puede servir como una gran protección para el equipo.

### Estimaciones agiles:

* Las features/stories son estimadas usando una medida de tamaño relativo conocida como story points (SP).
* Las medidas relativas no son absolutas.
* Story points no es una medida basada en tiempo.

#### Estimación relativa:

* Las personas no saben estimar en términos absolutos.
* Somos buenos comparando cosas.
* Comparar es generalmente más rápido.
* Se obtiene una mejor dinámica grupal y pensamiento de equipo más que individual.
* Se emplea mejor el tiempo de análisis de las storys.

**Tamaño:**

* El tamaño es una medida de la cantidad de trabajo necesaria para producir una feature/story.
* El tamaño indica:
  + Cuán compleja es una feature/story.
  + Cuánto trabajo es requerido para hacer o completar una feature/story.
  + Cuán grande es una feature/story.

### Tamaño vs esfuerzo:

Las estimaciones basadas en tiempo son más propensas a errores debido a varias razones:

* Habilidades.
* Conocimiento.
* Asunciones.
* Experiencia.
* Familiaridad con los dominios de aplicación/negocio.

Tamaño no es esfuerzo.

Para el tamaño se debe elegir una escala. Una vez elegida la escala no se cambia, si se cambia cambiamos el metro patrón.

### Story Point:

* Es una unidad de medida específica (del equipo) de complejidad, riesgo y esfuerzo.
* Story point da idea del “peso” de cada story y decide cuan grande (compleja) es.
* La complejidad de una feature/story tiende a incrementarse exponencialmente.
* Se miden a partir de los datos de la serie de Fibonacci.

### Propuesta de método de estimación:

**Poker estimation:**

* Popular entre los Agile practicioners.
* Combina opinión de experto, analogía y desegregación.
* Participantes en “planning poker” son desarrolladores.
  + Las personas más competentes en resolver una tarea deben ser quienes las estiman.

#### ¿Cómo decodificar las estimaciones?

Imagen en blanco y negro

Descripción generada automáticamente con confianza media

* 0: quizás usted no tenga idea de su producto o funcionalidad en este punto.
* ½, 1: funcionalidad pequeña (usualmente cosmética).
* 2-3: funcionalidad pequeña a mediana. Es lo que queremos.
* 5: funcionalidad media. Es lo que queremos.
* 8: funcionalidad grande, de todas formas, lo podemos hacer, pero hay que preguntarse sino se puede partir o dividir en algo más pequeño. No es lo mejor.
* 13: ¿alguien puede explicar porque no lo podemos dividir?
* 20: ¿cuál es la razón de negocio que justifica semejante story y más fuerte aún, porque no se puede dividir?
* 40: no hay forma de hacer esto en un sprint.
* 100: confirmación de que está algo muy mal. Mejor ni arrancar.

#### Poker planning:

**Prerrequisitos:**

* Lista de features/stories a ser estimadas.
* Cada estimador tiene un mazo de cartas.

**Pasos:**

1. Determinar la base story (la canónica) que será usada para comparar con las otras stories.
   1. La story a ser estimada se lee a todo el equipo.
   2. Los estimadores discuten la story, haciendo preguntas al producto owner (las que se necesiten).
   3. Cada estimador selecciona una carta y pone la carta boca debajo de en la mesa.
   4. Cuando todos pusieron las cartas, las mismas se exponen al mismo tiempo.
   5. Si todos los estimadores selecciona el mismo valor, ese es el valor. Si no, los estimadores discuten sus resultados, poniendo especial atención en los más altos y en los más bajos. Después de discutir ir a 1.c.
2. Se toma la próxima story, se discute con el producto owner.
3. Cada estimador asigna a la story un valor por comparación contra la base story. “Cuán grande/pequeña, compleja, riesgosa es esta story comparada con la base story?”- Volver a 1.c.

### Diferentes niveles de abstracción:

Imagen que contiene Gráfico

Descripción generada automáticamente

#### Spikes:

* Tipo especial de historia, utilizado para quitar riesgo e incertidumbre de una User Story u otra faceta del proyecto.
* Se clasifican en: técnicas y funcionales.
* Pueden utilizarse para:
  + Inversión básica para familiarizar al equipo con una nueva tecnología o dominio.
  + Analizar un comportamiento de una historia compleja y poder así dividirla en piezas manejables.
  + Ganar confianza frente a riesgos tecnológicos, investigando o prototipando para ganar confianza.
  + Frente a riesgos funcionales, donde no está claro como el sistema debe resolver la interacción con el usuario para alcanzar el beneficio esperado.

**Técnicas:**

* Utilizadas para investigar enfoques técnicos en el dominio de la solución.
  + Evaluar performance potencial.
  + Decisión hacer o comprar.
  + Evaluar la implementación de cierta tecnología.
* Cualquier situación en la que el equipo necesite una comprensión más fiable antes de comprometerse a una nueva funcionalidad en un tiempo fijo.

**Funcionales:**

* Utilizadas cuando hay cierta incertidumbre respecto de cómo el usuario interactuará con el sistema.
* Usualmente son mejor evaluadas con prototipos para obtener realimentación de los usuarios o involucrados.

Algunas User Stories requieren de ambos tipos de spikes. Por ejemplo:

* Como un cliente, quiero ver mi uso diario de energía en un histograma, para poder comprender rápidamente mi consumo de energía pasado, presente y proyectado.

En este caso un equipo puede crear dos spikes:

* Spike técnico:
  + Investigar cuanto tiempo requiere actualizar un display de un cliente al uso actual, determinando requerimientos de comunicación, ancho de banda y si los datos se actualizan en formato push o pull.
* Spike funcional:
  + Crear un prototipo de histograma en el portal web y obtener la retroalimentación de algunos usuarios respecto del tamaño, el estilo de la presentación y los atributos gráficos.

##### Lineamientos para spikes:

* Estimables, demostrables, y aceptables.
* La excepción, no la regla:
  + Toda historia tiene incertidumbre y riesgos.
  + El objetivo del equipo es aprender a aceptar y resolver cierta incertidumbre en cada iteración.
  + Los spikes deben dejarse para incógnitas más críticas y grandes.
  + Utilizar spikes como última opción.
* Implementar la spike en una iteración separada de las historias resultantes:
  + Salvo que el spike sea pequeño y sencillo y sea probable encontrar una solución rápida en cuyo caso, spike e historia pueden incluirse en la misma iteración.

### Algunas cosas para dejar en claro:

* Diferir el análisis detallado tan tarde como sea posible, lo que es justo antes de que el trabajo comience.
* Hasta entonces, se capturan requerimientos en la forma de “user stories”.
* Las user stories no son requerimientos de software, no necesitan ser descripciones exhaustivas de la funcionalidad del sistema.

### Tips para que las user stories sean útiles para el equipo:

* Un paso a la vez, evitar la “Y”.
* Usar palabras claras en los criterios de aceptación.
* No olvidarse la parte invisible: la conversación.
* Las user stories se escriben desde la perspectiva del usuario.
* No forzar todo para escribirlo como user stories.

## Scrum:

Scrum es un marco de trabajo liviano para generar valor mediante soluciones adaptativas para problemas complejos. Scrum se basa en el empirismo y el pensamiento Lean, empleando un enfoque iterativo e incremental.

Scrum es un marco de trabajo liviano que ayuda a las personas, equipos y organizaciones a generar valor a través de soluciones adaptativas para problemas complejos.

Scrum requiere un Scrum Master que fomente un entorno donde:

1. Un Product Owner ordena el trabajo en un Product Backlog.
2. El Scrum Team convierte una selección del trabajo en un Increment durante un Sprint.
3. El Scrum Team y los interesados inspeccionan los resultados y se adaptan para el próximo Sprint.
4. Se repite el proceso.

Diagrama

Descripción generada automáticamente

Scrum se fundamenta en la transparencia, inspección y adaptación. La transparencia permite la inspección, que a su vez posibilita la adaptación.

El marco de trabajo Scrum es inmutable y solo existe en su totalidad.

### Valores de Scrum:

El éxito de scrum depende de que las personas se vuelvan más competentes en cuento valores:

* Compromiso: para hacer las cosas en el plazo acordado.
* Foco: en concentrarse en una cosa a la vez. No perder el objetivo.
* Franqueza: la información relevante para construir el producto debe estar disponible para todos.
* Respeto: hacia todos los involucrados en el proyecto.
* Coraje: para aprender a manejar la incertidumbre que tienen los proyectos.

### Scrum Team:

El Scrum Team consta de un Scrum Master, un Product Owner y Developers. Es multifuncional, autogestionado y típicamente de 10 personas o menos.

#### Developers:

Los Developers se comprometen a:

* Crear un plan para el Sprint (Sprint Backlog).
* Inculcar calidad adhiriéndose a la Definición de Terminado.
* Adaptar su plan diariamente hacia el Objetivo del Sprint.
* Responsabilizarse mutuamente como profesionales.

#### Product Owner:

El Product Owner maximiza el valor del producto y gestiona eficazmente el Product Backlog:

* Desarrolla y comunica el Objetivo del Producto.
* Crea y comunica los elementos del Product Backlog.
* Ordena los elementos del Product Backlog.
* Asegura que el Product Backlog sea transparente y entendible.

#### Scrum Master:

El Scrum Master establece Scrum como se define en la Guía de Scrum:

* Guía al equipo en la autogestión y multifuncionalidad.
* Ayuda a enfocarse en crear Increments de alto valor.
* Elimina impedimentos al progreso del equipo.
* Asegura que los eventos de Scrum ocurran y sean productivos.

Es un rol de moredador, vela por que lo definido en Scrum se cumpla.

### Eventos de Scrum:

#### Sprint:

Los Sprints son el corazón de Scrum, donde las ideas se convierten en valor. Tienen una duración fija de un mes o menos. En él se desarrolla una versión o incremento del sistema.

#### Sprint Planning:

La Sprint Planning inicial el Sprint establecimiento el trabajo a realizar. Aborda tres temas:

* Por qué este Sprint es valioso (Objetivo del Sprint).
* Qué se puede hacer en este Sprint.
* Cómo se realizará el trabajo elegido.

#### Daily Scrum:

La Daily Scrum es un evento de 15 minutos para los Developers con el fin de inspeccionar el trabajo hacia el Objetivo del sprint y adaptar el Sprint Backlog.

Se habla sobre que se hizo, que va a hacer y que problemas tuvo.

#### Sprint Review:

El propósito de la Sprint Review es inspeccionar el resultado del Sprint y determinar futuras adaptaciones. El Scrum Team presenta los resultados a los interesados clave y se discute el progreso hacia el Objetivo del Producto.

#### Sprint Retrospective:

La Sprint Retrospective planifica formas de aumentar la calidad y la eficiencia. El Scrum Team inspecciona el último Sprint e identifica mejoras para implementar en el próximo Sprint.

Se deben plantear lo que se hizo bien y lo que se hizo mal como equipo, no de x o y persona. El objetivo es mejorar.

#### Refinamiento:

Se hace cuando sea necesario hacer un refinamiento o ajuste al Sprint Backlog.

El refinamiento es una tarea continua, se hace durante el sprint.

El equipo se reúne con el PO para discutir ítems del backlog.

### Artefactos de Scrum:

#### Product Backlog:

El Product Backlog es una lista emergente y ordenada de lo que se necesita para mejorar el producto. Es la única fuente del trabajo para el Scrum Team.

Compromiso: Objetivo del Producto.

El Objetivo del Producto describe un estado futuro del producto y sirve como meta para que el Scrum Team planifique.

#### Sprint Backlog:

El Sprint Backlog se compone del Objetivo del Sprint, los elementos del Product Backlog seleccionados y un plan para entregar el Increment.

Compromiso: Objetivo del Sprint.

El Objetivo del Sprint es el único propósito del Sprint y proporciona flexibilidad en términos del trabajo exacto necesario para lograrlo.

Para que una user pueda ser incluida en el sprint backlog debe cumplir con el modelo IMVES. Lo que se llama definición de listo.

#### Increment:

Un Increment es un paso concreto hacia el Objetivo del Producto. Cada Increment se suma a todos los Increments anteriores y se verifica minuciosamente.

Compromiso: Definición de Terminado.

La Definición de Terminado es una descripción formal del estado del Increment cuando cumple con las medidas de calidad requeridas para el producto.

### Timebox en Scrum:

Scrum plantea que cada evento debe tener una duración fija.

* Sprint: 1 mes o menos.
* Sprint Planning: 8 horas máximo para un sprint de 1 mes.
* Daily meeting: 15 minutos.
* Sprint Review: 4 horas máximo para un Sprint de 1 mes.
* Sprint Retrospective: 3 horas máximo para un Sprint de 1 mes.
* Refinamiento del PB: 10% del tiempo del Sprint.

### Definición de “Listo” (Ready):

Son una serie de requisitos que debe cumplir una US para poder estar en el sprint backlog, a estos requisitos los establece el equipo.

Ejemplos:

* Valor de negocio claramente expresada.
* Detalles suficientemente comprendidos por el equipo de forma tal que puedan tomar una decisión informada sobre si pueden completar el ítem del producto backlog (PBI).
* Dependencias identificadas y no hay dependencias externas que puedan impedir que el PBI se complete.
* El equipo ha sido asignado adecuadamente para completar el PBI.
* El PBI ha sido debidamente estimado y es lo suficientemente pequeño para ser completado en un Sprint.
* Los criterios de aceptación son claros y testeables.
* Los criterios de performance si hay, son claros y testeables.
* El equipo comprende como mostrar el PBI en la Sprint Review.

### Definición de “Hecho” (Done):

Son los criterios que se establecen como equipo que hacen que una US sea parte del incremento del producto.

Ejemplos:

* Diseño revisado.
* Código completo.
  + Código refactorizado.
  + Código con formato estándar.
  + Código comentado.
  + Código en el repositorio.
  + Código inspeccionado.
* Documentación de usuario actualizada.
* Probado.
  + Prueba de unidad hecha.
  + Prueba de integración hecha.
  + Prueba de sistema hecha.
* Cero defectos conocidos.
* Prueba de aceptación realizada.
* En los servidores de producción.

### Capacidad del equipo en un Sprint:

Texto

Descripción generada automáticamente con confianza baja

Es una estimación de lo que el equipo puede llegar a hacer en un sprint. Lo idea es hacerlo en story points.

### Cálculo de capacidad del equipo en un Sprint:

Tabla

Descripción generada automáticamente

### Múltiples niveles de planificación:

Un conjunto de letras blancas en fondo azul

Descripción generada automáticamente con confianza media

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Nivel | Horizonte | Quién | Foco | Entregable |
| Portfolio | 1 año o más | Stakeholders y Product Owners | Administración de un portfolio de Producto. | Backlog de Portfolio. |
| Producto | Arriba de varios meses o más | Product Owner y Stakeholders | Visión y evolución del producto a través del tiempo | Visión de Producto, Roadmap y características de alto nivel. |
| Release | 3 (o menos) a 9 meses | Equipo Scrum, Stakeholders | Balancear continuamente el valor de cliente y la calidad global con las restricciones de alcance, cronograma y presupuesto. | Plan de release. |
| Iteración | Cada iteración (de 1 semana a 1 mes) | Equipo Scrum | Que aspecto entrega en el siguiente sprint | Objetivo del sprint y sprint backlog |
| Día | Diaria | Equipo Scrum (al menos los que trabajan en IPB) | Cómo completar lo comprometido | Inspección del progreso actual y adaptación a la mejor forma de organizar el siguiente día de trabajo. |

### Cadencia de los sprints:

* Relase luego de múltiples sprints.
* Release luego de cada sprint.
* Release luego de cada feature.

### Planificación de iteración: Sprint Planning:

Diagrama

Descripción generada automáticamente

### Planificación diaria: Daily Scrum:

Diagrama

Descripción generada automáticamente

### Métricas en ambientes ágiles:

Deben medir el software funcionando.

**Regla de oro agile sobre métricas:**

* La medición es una salida, no una actividad.
* Medir lo que sea necesario y nada más.

**Dos principios agiles que guían la elección de las métricas:**

* Nuestra mayor prioridad es satisfacer al cliente por medio de entregas tempranas y continuas de software valioso.
* El software trabajando es la principal medida de progreso.

#### RTF – Running Tested Features:

Gráfico, Gráfico de líneas

Descripción generada automáticamente

Cuantas features fueron testeadas y están corriendo en producción.

#### Capacidad:

* Horas de trabajo disponibles por día (WH) X días Disponibles Iteración (DA) = capacidad.

Gráfico, Gráfico de barras

Descripción generada automáticamente

#### Velocidad (Velocity):

Es una métrica del progreso de un equipo. Se calcula sumando el número de story points (asigados a cada user story) que el equipo completa durante la iteración.

Se cuentan los story points de las Users Stories que están completas, no parcialmente completas.

La velocidad corrige los errores de estimación.

## Gestión de productos:

### ¿Por qué creamos productos?

* Para satisfacer a los clientes.
* Para tener muchos usuarios logueados.
* Para obtener mucho dinero.
* Realizar una gran visión, cambiar el mundo.

### ¿Qué características realmente utilizamos de un producto de software?

Diagrama

Descripción generada automáticamente con confianza media

### Evolución de los productos de software:

Diagrama

Descripción generada automáticamente

### Productos mínimos para la gestión de productos:

* Minimal viable product (producto mínimo viable).
* Minimal viable feature (característica mínima viable).
* Minimal release feature (características mínimas del release).
* Minimal marketable feature (característica mínima comercializable).

#### Relación:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

### MVP:

* Versión de un nuevo producto creado con el menor esfuerzo posible.
* Dirigido a un subconjunto de clientes potenciales.
* Utilizado para obtener aprendizaje validado.
* Más cercano a los prototipos que a una versión real funcionando de un producto.
* Es un concepto de Lean Startup que enfatiza el impacto del aprendizaje en el desarrollo de nuevos productos.
* Una premisa clave detrás de la idea de MVP es que usted produce un producto real que puede ofrecer a los clientes y observar su comportamiento real con el producto o servicio.
* “Versión de un nuevo producto que permite a un equipo que permite a un equipo recopilar la cantidad máxima de aprendizaje validado sobre clientes con el menor esfuerzo”. Este aprendizaje validado viene en forma de si sus clientes realmente comprarán su producto.
* Ver lo que la gente realmente hace con respecto a un producto es mucho más confiable que preguntarle a la gente qué harían.
* Tiene el valor suficiente para que las personas estén dispuestas a usarlo o comprarlo inicialmente.
* Demuestra suficiente beneficio futuro para retener a los primeros usuarios.
* Proporciona un ciclo de retroalimentación para guiar el desarrollo futuro.

### MMF:

* Es la pieza más pequeña de funcionalidad que puede ser liberada.
* Tiene valor tanto para la organización como para los usuarios.
* Es parte de un MMR o MMP.

### MMP:

* Primer release de un MMR dirigido a primeros usuarios (early adopters).
* Focalizado en características clave que satisfarán a este grupo clave.

### MMR:

* Release de un producto que tiene el conjunto de características más pequeño posible.
* El incremento más pequeño que ofrece un valor nuevo a los usuarios y satisface sus necesidades actuales.
* MMP = MMR1.

### MVF:

* Es una mini versión del MVP.
* Es una característica a pequeña escala que se puede construir e implementar rápidamente, utilizando recursos mínimos, para una población objetivo para probar la utilidad y adopción de la característica.
* Un MVF debe proporcionar un valor claro a los usuarios y ser fácil de usar.
* MVF requiere recursos mínimos, los estándares de calidad de la industria y la producción deben guiar el diseño y la confiabilidad.
* El grupo de usuarios para un MVF, son los primeros en adoptar los clientes leales que han compartido conocimientos anteriormente o los miembros de una junta asesora de clientes. Usurarios flexibles y tolerantes.
* Los resultados le ayudarán a tomar decisiones estratégicas sobre productos.

### Errores comunes:

* Confundir a un MVP, que se enfoca en el aprendizaje, con característica comercializable mínima (MMF) o con producto comercializable mínimo (MMP), ambos se enfocan en “ganar”.
* El riesgo de esto es entregar algo sin considerar si es lo correcto que satisface las necesidades del cliente.
* Enfatizar la parte mínima del MVP con exclusión de la parte viable. El producto entregado no es de calidad suficiente para proporcionar una evaluación precisa de si los clientes utilizarán el producto.
* Entregar lo que consideran un MVP, y luego no hacer más cambios a ese producto, independientemente de los comentarios que reciban al respecto.

### Valor vs desperdicio:

* Lean Thinking define la creación de valor como proveer beneficio a los clientes, cualquier otra cosa es desperdicio.
* La productividad de un Startup no puede medirse en términos de cuánto se construye cada día, por el contrario, se debe medir en términos de averiguar la cosa correcta a construir cada día.

#### Build-experiment-learn feedback loop:

Diagrama

Descripción generada automáticamente

#### Hacia la creación de valor:

* El éxito no es entregar un producto, el éxito se trata de entregar un producto (o característica de producto) que el cliente usará.
* La forma de hacerlo es alinear los esfuerzos continuamente hacia las necesidades reales de los clientes.
* The build-experiment-learn feedback loop permite descubrir las necesidades del cliente y alinearlas metodológicamente.

### La fase construir: MVP:

* Ingresar lo más rápido posible con un MVP.
* Un MVP varía en complejidad desde pruebas de humo extremadamente simples hasta prototipos tempranos.
  + Prueba de humo (smoke tests): no hay producto construido aún.
  + Prototipos: un producto con características faltantes o problemas.
* Decidir qué tan complejo no se puede hacer un MPV de manera formulada:
  + Requiere juicio.
  + En caso de duda, simplifique.
  + Evite la construcción excesiva y la promesa excesiva.
  + Cualquier trabajo adicional más allá de lo que necesita para comenzar el ciclo podría ser un desperdicio.
* Un MVP no sólo habla sobre el diseño del producto y las preguntas técnicas, sino que también sirve para probar hipótesis comerciales fundamentales.
* Por lo tanto, sirve para proporcionar una dosis necesaria de realidad.

### Dilema: la audacia de cero:

* A menudo es más fácil recaudar dinero cuando tiene:
  + Cero ingresos.
  + Cero clientes.
  + Tracción cero.

Que cuando tienes una pequeña cantidad de cada uno.

* Cero invita a la imaginación, pero los números pequeños hacen preguntas sobre si los números grandes alguna vez se materializarán.
* Este fenómeno (llamado la audacia de cero) crea un incentivo brutal:
  + Aplazar el lanzamiento de cualquier versión de un producto hasta que esté seguro del éxito.
* Si pospone experimentar con su MVP, surgirán algunos resultados desafortunados como:
  + La cantidad de trabajo desperdiciado puede aumentar.
  + Se perderán los comentarios esenciales.
  + El riesgo de que su startup construya algo que nadie quiere puede aumentar.
* Use un MVP para experimentar (inicialmente, en silencio) con los primeros usuarios en el mercado.
* Verifique su concepto probando TODOS sus elementos, comenzando por los más riesgosos.

### Supuestos de “Saltos de Fe”:

* Los elementos más riesgosos del plan/concepto de una startup (es decir, las partes de las que todo depende) se denominan supuestos de salto de fe.
* La mayoría de las personas no conocen una determinada solución (o incluso un problema); pero una vez que experimentan la solución, no pueden imaginar cómo vivirían sin ella.
* Hipótesis del valor:
  + Prueba si el producto realmente está entregando valor a los clientes después de que comienzan a usarlo.
  + Una métrica de prueba: tasa de retención.
* Hipótesis de crecimiento:
  + Prueba cómo nuevos clientes descubrirán el producto.
  + Una métrica de prueba: tasa de referencia o Net Promoter Score (NPS).

### Preparar un MVP:

1. Encontrar un nicho.
2. Crear un roadmap realista.
3. Investigar la competencia.
4. Pre-vender el MVP.
5. Testear las suposiciones.
6. Asegurarse que el MVP resuelve el problema correcto.
7. Focalizar en las funcionalidades principales.

### Características de un MVP:

* Diseño:
  + Diseño adecuado.
  + Consigue una UX que deleita.
  + Logra satisfacer el aspecto visual y de interacción.
* Usabilidad:
  + Tiene suficiente valor para que la gente esté dispuesta a usarlo/comprarlo.
  + Resulta útil para su público objetivo.
* Confiabilidad:
  + Involucra a los early adopters para que puedan confiar en la solución plenamente.
  + Incluso cuando tenga poco tiempo en el mercado.
* Funcionalidad:
  + Tiene las funciones necesarias para solucionar un problema específico.
  + Satisface las demandas y permite evaluar las funciones a implementar más adelante.

Diagrama

Descripción generada automáticamenteUn conjunto de letras negras en un fondo blanco

Descripción generada automáticamente con confianza media

### Matriz de priorización para el MVP:

Gráfico, Gráfico de rectángulos

Descripción generada automáticamente