RESUMEN INGENIERIA Y CALIDAD DEL SOFTWARE

# Componentes de proyecto de desarrollo de software

## Proceso de software

Se define un proceso como la colección de actividades de trabajo, acciones y tareas que se realizan cuando va a crearse algún producto terminado. Cada una de las actividades, acciones y tareas se encuentra dentro de una estructura o modelo que define su relación tanto con el proceso como entre sí, que define la organización de acuerdo con lo que le resulte útil a la misma.

Mientras que un **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.

Se pueden diferenciar dos tipos de procesos:

* **Definidos**: en este tipo se asume que se puede repetir una y otra vez el mismo proceso para obtener siempre el mismo resultado. Están inspirados en las líneas de producción, por lo que la administración y el control provienen de la predictibilidad del proceso.
* **Empíricos**: ideales para ambientes de incertidumbre (como lo suele ser el caso del desarrollo de software) donde las variables cambian frecuentemente. Por ello, aunque se aplique el mismo proceso varias veces, se pueden obtener diferentes resultados obligando a que la administración y control se realice a través de inspecciones y adaptaciones. A la hora de aplicar este tipo de procesos se debe utilizar un patrón de conocimiento cíclico que funciona de la siguiente manera: Asumir 🡪 Construir 🡪 Retroalimentar 🡪 Revisar 🡪Adaptar

## Ciclos de vida o modelo de proceso

El ciclo de vida define como se ejecutarán las actividades definidas en un proceso de desarrollo. Representan la serie de pasos a través de los cuales el producto y/o proyecto progresa. Ej.: secuencial, iterativo incremental, espiral, etc.

Un ciclo de vida de un proyecto de SW es una representación de un proceso. Grafica una descripción del proceso desde una perspectiva particular, para ello, especifican las fases del proceso (requerimientos, especificación, diseño, etc.) y el orden en el cual se llevarán a cabo.

Podemos clasificar a los ciclos de vida en tres tipos básicos: secuencial, iterativo/incremental y recursivo.

## Proyecto

Proyecto es el conjunto de las actividades que desarrolla una persona o una entidad para alcanzar un determinado objetivo. Estas actividades se encuentran interrelacionadas y se desarrollan de manera coordinada.

Diagrama

Descripción generada automáticamente

Las características de un proyecto son:

* Están orientados a objetivos:
  + Están dirigidos a obtener resultados y ello se refleja a través de objetivos
  + Los objetivos no deben ser ambiguos y deben guiar al proyecto
  + El objetivo debe ser alcanzable
* Poseen una duración limitada: son temporarios, una vez alcanzado el o los objetivos el proyecto termina
* Están compuestos por tareas interrelacionadas basadas en esfuerzos y recursos. Es decir, esta compuesto por tareas ejecutadas por personas que pueden utilizar o no otros recursos y en donde cada actividad requiere un esfuerzo.
* Son únicos: por más similares que sean entre sí, todos los proyectos tienen características que los vuelven únicos.

## Administración de proyectos

La administración de proyectos es la aplicación de conocimientos, habilidades, herramientas y técnicas a las actividades del proyecto para satisfacer los requerimientos del proyecto. Es decir que, administrar un proyecto abarca:

* Identificar requerimientos
* Establecer objetivos claros y alcanzables
* Adaptar especificaciones, planes y el enfoque a los intereses de los stakeholders.

A la hora de desarrollar proyectos, es crucial que se tenga un plan. Dicho plan documenta:

* Alcance: ¿Qué es lo que hacemos?
* Tiempo: 🡪 ¿Cuándo lo hacemos?
* Actividades, recursos: ¿Cómo lo hacemos?
* Personas: ¿Quién lo hace?

### La restricción triple

Dentro de la administración de proyectos nace el concepto de la restricción triple. Plantea que un proyecto de alta calidad entrega el **producto requerido** (alcance), en el **tiempo** estipulado y bajo el **presupuesto** planificado.

Para ello, el líder de proyecto es el encargado de balancear estos tres ejes de manera que se obtenga el mejor resultado. Sin embargo, cabe destacar los siguientes puntos:

* La calidad no es negociable
* Al modificar uno de los ejes es muy probable que se modifique uno de los otros dos. Es decir, si aumentamos el presupuesto quizás podamos disminuir el tiempo de realización del proyecto.

### Roles en la administración de proyectos

#### Líder de proyecto

El líder de proyecto asigna tareas al equipo. Se relaciona con clientes, subcontratistas y organizaciones regulatorias, es decir: stakeholders.

Debe concentrarse en comprender el problema que se va a resolver, en administrar el flujo de ideas y, al mismo tiempo, en dejar que todos en el equipo sepan (por medio de palabras y, mucho más importante, con acciones) que la calidad cuenta y que no se comprometerá.

#### Equipo de proyecto

Grupo de personas comprometidas en alcanzar un conjunto de objetivos de los cuales se sienten mutuamente responsables. Las características de estos son:

* Diversos conocimientos y habilidades
* Posibilidad de trabajar juntos efectivamente (apuntan a la sinergia)
* Usualmente son grupos pequeños
* Tienen sentido de responsabilidad como una unidad.

### Planificación de proyectos de software

Se le llama planificación de proyectos de software, a dar respuesta a los siguientes puntos:

* Definir alcance del proyecto
* Definir el proceso y ciclo de vida
* Realizar estimaciones
* Gestionar los riesgos
* Asignar recursos
* Programar el proyecto
* Definir los controles necesarios
* Definir las métricas útiles

Causas comunes de fallos de 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

#### Definición del alcance

Se debe distinguir entre dos conceptos similares:

* **Alcance de producto:** Todas las características a incluirse en un producto o servicio. Es decir, se basa en los requerimientos o en otras palabras, se mide el alcance del producto contra la especificación de requerimientos.
* **Alcance del proyecto:** Es todo el trabajo, y solo el trabajo, que debe hacerse para entregar el producto/servicio con todas las características y funcionalidades especificadas. Se mide contra el plan de proyecto.

#### Definición de un ciclo de vida

Debemos definir aquellas actividades que se realizarán, el orden de las mismas y los alcances que tendrá. Ejemplo:

Diagrama

Descripción generada automáticamente

#### Estimaciones

A la hora de realizar estimaciones, se deben tener en cuenta los siguientes puntos:

* Tamaño
* Esfuerzo
* Calendario
* Costo
* Recursos críticos

#### Gestión de riesgos

Un riesgo se puede pensar como un problema que esta esperando a suceder y que podría comprometer el éxito del proyecto. Para gestionar los riesgos es necesario tener en claro el concepto que tanta exposición tiene un riesgo, es decir, que tan probable es que pase y que tan grave sería si ocurriese.

Los pasos básicos para identificar y gestionar riesgos son:

1. Identificarlos y especificarlos
2. Analizar su exposición
3. Planear como se gestionarán
4. Realizar un seguimiento y control
5. Aprender de los mismos y almacenarlos en una base de datos

#### Métricas

Como su nombre lo indica, las métricas serán útiles para medir diversos aspectos del desarrollo del SW y su dominio se divide en tres:

* Métricas de proceso
* Métricas del proyecto
* Métricas de producto

Las métricas deben ser útiles para lo que necesitamos saber: “No tiene sentido medir en milímetros si la distancia es de kilómetros.”

# Desarrollo ágil del software

La ingeniería de software ágil surge como una filosofía con un conjunto de lineamientos de desarrollo. Ágil no es una metodología o un proceso, es una ideología con un conjunto de principios que guían el desarrollo del producto. Básicamente, se busca un balance entre ningún proceso y demasiado proceso (disminuir la documentación), se adaptan en lugar de ser predictivos y están orientados a los individuos.

Los métodos agiles dan mejores resultados cuando los problemas son complejos (como el desarrollo de un nuevo software).

El paradigma de desarrollo tradicional de software plantea que los requisitos se deben obtener al mayor detalle posible al inicio del desarrollo, y estos se mantendrán fijos durante la vida del proyecto por lo que la persona a cargo debe ser capaz de establecer los recursos y tiempo de acuerdo con la utilidad o beneficio que se le quiera otorgar al usuario, es decir el valor. Por otro lado, las metodologías ágiles entienden que los requisitos cambian, se estima el uso de recursos y el tiempo de desarrollo (lo que servirá para armar un plan) pero el alcance final del producto dependerá del valor que se le otorgue al cliente.

Los precursores de este cambio de paradigma plasmaron los valores del mismo en el manifiesto ágil:

* Se valoran los individuos y sus interacciones sobre los procesos y las herramientas
* Es más importante un software funcionando que una documentación exhaustiva
* Se busca colaboración con el cliente de manera constante sobre un contrato estructurado
* Es más importante responder al cambio sobre seguir un plan

Dichos valores, se ven reflejados en los 12 principios del manifiesto ágil:

1. La prioridad más alta es satisfacer al cliente a través de la entrega pronta y continua de software valioso.
2. Son bienvenidos los requerimientos cambiantes, aun en una etapa avanzada del desarrollo. Los procesos ágiles dominan el cambio para provecho de la ventaja competitiva del cliente.
3. Entregar con frecuencia software que funcione, de dos semanas a un par de meses, de preferencia lo más pronto que se pueda.
4. Las personas de negocios y los desarrolladores deben trabajar juntos, a diario y durante todo el proyecto.
5. Hay que desarrollar los proyectos con individuos motivados. Debe darse a éstos el ambiente y el apoyo que necesiten, y confiar en que harán el trabajo.
6. El método más eficiente y eficaz para transmitir información a los integrantes de un equipo de desarrollo, y entre éstos, es la conversación cara a cara.
7. La medida principal de avance es el software que funciona.
8. Los procesos ágiles promueven el desarrollo sostenible. Los patrocinadores, desarrolladores y usuarios deben poder mantener un ritmo constante en forma indefinida.
9. La atención continua a la excelencia técnica y el buen diseño mejora la agilidad.
10. Es esencial la simplicidad: el arte de maximizar la cantidad de trabajo no realizado.
11. Las mejores arquitecturas, requerimientos y diseños surgen de los equipos con organización propia.
12. El equipo reflexiona a intervalos regulares sobre cómo ser más eficaz, para después afinar y ajustar su comportamiento en consecuencia.

## Requerimientos ágiles

Los principios ágiles relacionados con los requerimientos son los siguientes:

1. La prioridad es satisfacer al cliente a través de entregas tempranas y frecuentes
2. Se debe poder recibir cambios de requerimientos, aun en etapas finales
3. Todos deben trabajar en el proyecto, tanto personal técnico como no técnico.
4. El medio de comunicación por excelencia es cara a cara
5. Las mejores arquitecturas, diseños y requerimientos emergen de equipos auto-organizados.

## User Stories – Historias de usuarios

Las historias de usuario son una explicación de una función de software, escrita desde la perspectiva del usuario final. Esto ayuda a los equipos ágiles a comprender lo que quieren los usuarios y así poder ofrecerles las mejores funciones. Un conjunto de historias de usuario relacionadas se denomina “tema”.

En forma general, las historias de usuario impactan “verticalmente” a todo el proyecto, es decir que afectan a todo el equipo de desarrollo (GUI, Business Logic, Database, etc.).

Cabe destacar que las historias de usuario NO son requerimientos, si bien sirven para hacer el parte del trabajo que se hace en la especificación de requerimientos, difiere de lo que formalmente llamamos requerimientos.

Por lo general, una historia de usuario seguirá el siguiente formato: “Como [nombre de rol], quiero [actividad], para lograr [valor de negocio que recibo]”.:

* Rol: representa quien realiza la acción o bien quien recibe el valor de la actividad
* Actividad: representa la acción a ser realizada por el sistema
* Valor: es el valor que le aporta al negocio. Es el “para” de la actividad, implica la utilidad que le dará la actividad al sistema.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Como se muestra en el ejemplo, las historias de usuarios permiten encontrar roles (tipos de usuarios), idealmente, estos deben ser específicos para evitar caer en roles muy abarcativos (clientes, gerentes, vendedores, etc.).

Están compuestas por tres partes comúnmente llamadas “Las 3C” por sus siglas en inglés:

1. **Tarjeta (card):** es una descripción escrita de la historia de usuario utilizada para la planificación del sprint.
2. **Conversación (conversation):** es una discusión entre clientes, usuarios y desarrolladores sobre la prioridad y las posibles soluciones para la historia del usuario. La conversación se documenta en los criterios de aceptación o bien en la herramienta que utilice el equipo de desarrollo.
3. **Confirmación (confirmation):** es un acuerdo entre las partes interesadas de que se han alcanzado los objetivos y las soluciones de la historia de usuario. Las pruebas de usuarios serán claves para representar los testeos necesarios para saber que la US se resolvió correctamente.

A su vez, las historias de usuarios son multipropósito, ya que representan:

* Una necesidad del usuario
* Una descripción del producto
* Un ítem de planificación
* Token para una conversación
* Un mecanismo para diferir una conversación

Es trabajo del Product Owner priorizar las historias en el Product Backlog con el fin de que cada iteración entregue software que implemente los ítems de trabajo de mayor prioridad.

### Criterios de aceptación de historias de usuarios

Los criterios de aceptación en las User Stories definen límites para las mismas, ayudan al PO a responder cuales son los requerimientos funcionales mínimos que la US provee para agregar valor al sistema, permite que el equipo tenga una visión compartida de la US, facilita el desarrollo y el testing.

Texto

Descripción generada automáticamente

Un buen criterio de aceptación define una intención, no una solución a la US. Además, deben ser independientes de la implementación y son de “alto” nivel.

Los detalles de implementación o de requerimientos no funcionales que afecten a la US son el resultado de la conversación entre el PO y el equipo, por lo que se pueden capturar en documentación interna del equipo o en pruebas de aceptación automatizadas. Estas últimas, entonces, expresan detalles de la conversación y son complementarias a la US. El proceso genérico para hacerlas involucra identificarlas al dorso de la US y luego diseñar la prueba completa.

Texto

Descripción generada automáticamente

### Criterio de DONE & READY

El criterio de *ready* se utiliza para saber cuando una US la especifico el PO y está lista para agregarla a una iteración. Esta estrechamente relacionada con los atributos INVEST.

El criterio de done sirve para determinar cuándo una US esta finalizada, es decir, se deben definir criterios para corroborar que la US ha sido terminada y puede ser parte del entregable al final de la iteración.

### INVEST en buenas historias de usuarios

Los equipos de desarrollo ágil utilizan una gran cantidad de tiempo descubriendo, elaborando y entiendo las US; y luego desarrollando las pruebas de aceptación debido a que necesitan un perfecto entendimiento de los objetivos que representa la US para el desarrollo del producto. Es por ello que idealmente se deben tener historias de usuarios con buenos atributos que valgan el esfuerzo que realizan los equipos, para ello existen los atributos INVEST:

* Independientes: la US debe poder ser desarrollada, testeada e incluso desplegada por si misma. Es decir, debe aportar valor de manera independiente a otras.
* Negociables: deben definir el “qué” no el “como”, es decir que deben aclarar el objetivo no la funcionalidad que resolverá el problema.
* Valuable: Deben aportar valor al cliente. El valor tiene que ser en ese momento del tiempo, para el negocio tratado y para el usuario que lo utilizará.
* Estimable: el equipo debe poder dar una estimación sobre la complejidad y el trabajo que requiere completarla.
* Small (Pequeña): No deben ser mayores a una iteración.
* Testeable: Se debe poder demostrar que fue implementada.

### Spikes

Son tipos especiales de US que se usan para quitar el riesgo y la incertidumbre propia de una US u otra faceta del proyecto. Pueden ser utilizadas por varias razones:

* El equipo puede no tener conocimiento sobre algún nuevo dominio y la spike puede servir como una inversión básica para familiarizar al equipo con el mismo.
* La US puede ser muy grande para ser correctamente estimada, por lo que el equipo puede utilizar una spike para analizar un comportamiento de la misma y dividirla en partes manejables.
* Si una historia contiene un riesgo técnico significante y el equipo debe realizar algún tipo de investigación o prototipo para ganar confianza en alguna solución técnica.
* Si la historia contiene algún riesgo funcional, donde no esta claro como el sistema debe resolverla para alcanzar el beneficio esperado por el usuario.

Se clasifican en:

* **Técnicas:** utilizadas para investigar diversos enfoques en el dominio de la solución. Abarcan cualquier situación en la que el equipo necesite una comprensión más fiable antes de comprometerse a una nueva funcionalidad en un tiempo fijo. Como puede ser: evaluar rendimiento potencial, decisiones de hacer o comprar, evaluar la implementación de una tecnología, etc.
* **Funcionales:** utilizadas cuando existe cierta incertidumbre respecto a cómo el usuario interactuará con el sistema. Usualmente son mejor evaluadas con prototipos para obtener retroalimentación de los usuarios o involucrados.

Puede suceder que algunas historias de usuarios requieran dos spikes, una de cada tipo. Ejemplo:

“*Como cliente, quiero ver mi gasto energético en un histograma, de manera que pueda entender rápidamente mi pasado, presente y futuro (cercano) gasto de energía”*

*En este caso deberíamos dividirlo en:*

* *Spike técnica: investigar cuando tardaría actualizar la pantalla al uso actual, determinando requerimientos de comunicación, banda ancha o si la información debe ser tipo push o pull.*
* *Spike funcional: realizar un prototipo de histograma en algún portal web y obtener un feedback en términos de presentación, tamaño, estilos y características del gráfico.*

#### Lineamientos para spikes

Las spikes deben ser estimables, demostrables y aceptables. Por lo general los resultados de las spikes difieren de los de las US, ya que no son códigos ejecutables, sino información y/o prototipos; en cualquier caso el resultado de la spike debe ser suficiente para resolver la incertidumbre que posee la historia de usuario a la que “reemplaza”.

Las spikes deben ser la excepción, no la regla. Todas las US tienen cierta incertidumbre/riesgo debido a la naturaleza ágil de las mismas, pero el equipo debe ser capaz de encontrar la solución adecuada discutiendo, colaborando, experimentando y negociando. Las spikes únicamente deben dejarse para las incógnitas críticas y grandes (son la última opción).

Usualmente las spikes deben ser implementadas en una iteración separada de las historias resultantes. Esto puede obviarse si la spike es muy pequeña o de solución sencilla.

AGREGAR ESTIMACION

## Errores de estimación

Los errores en los sistemas de información vienen de cuatro fuentes genéricas:

* Información imprecisa sobre el proyecto a estimar
* Información imprecisa acerca de las capacidades de la organización que realizará el proyecto
* Demasiado caos en el proyecto: tratar de estimar un objetivo constantemente cambiante
* Imprecisiones referidas al proceso de estimación en sí mismo

## Técnicas de estimación

### Contar

Los proyectos de software tienen una gran cantidad de cosas que se pueden contar: requerimientos, características, casos de uso, US, entre otros. Se puede decidir que contar de acuerdo con una pequeña cantidad de objetivos:

* Se debe encontrar algo para contar que este estrechamente relacionado con el tamaño del software que se estima. Es decir, si las características del software son fijas y se está tratando de estimar el costo y el cronograma, el tamaño del software será una medida adecuada que influenciará esos factores. Básicamente debemos encontrar algo para contar que sea un indicador importante sobre lo que estamos tratando de estimar.
* Encontrar algo para contar que este disponible en las primeras fases del desarrollo. Idealmente, entre más temprano encontremos algo para contar, más rápido seremos capaces de proveer una estimación a largo plazo.
* Encontrar algo para contar que produzca una estadística viable. Es decir, debemos encontrar algo para contar que provee una información que se repite o que representa una media; si solo tomamos algo para contar que se produzca una única vez la información provista no provee valor.
* Entender que se está contando. Para contar algo mediante lo que estimar es necesario entender que es, como nos aporta valor y de donde provee lo que estamos contando.
* Encontrar algo que se pueda contar con el menor esfuerzo posible.

## Métodos para estimar: Juicio Experto

Un método basado en experiencia ampliamente utilizado en la industria de desarrollo de software es el de juicio experto. Básicamente, este método tiene dos variantes:

* Puro: un individuo, considerado el experto, realiza las estimaciones de acuerdo con las especificaciones que logre distinguir y su experiencia con proyectos similares. Idealmente, el experto debe seguir un método y evitar hacer estimaciones intuitivas:
  + Granularidad: dividir las tareas grandes en tareas de menor tamaño. La idea es minimizar el error, si las tareas son enormes, el error de estimación tiene mayor probabilidad de ser grande.
  + Uso de rangos: es un método que consiste en tomar estimaciones optimistas, pesimistas y habituales para luego utilizar una fórmula que permita realizar una estimación ajustada.
  + Realizar checklists: verificar que se estén teniendo en cuenta todos los factores en el desarrollo, es decir, que no solo se considere la escritura de código, sino el tiempo de testing, la documentación, la rotación de personal, etc.

Sin embargo, este método es altamente riesgoso debido a varios factores tales como que el experto que estima no es necesariamente parte del desarrollo, las experiencias que posea no son necesariamente trasladables al nuevo equipo y siempre se posee un grado de subjetividad. Si el experto desaparece, el experto deja de estimar.

* Delphi o grupal: considera que las estimaciones deben ser realizadas por un grupo de expertos, donde deban converger en estimaciones que consideren adecuadas. Se le dan especificaciones a un grupo de expertos, cada uno de ellos realiza las estimaciones correspondientes y se las entregan a un coordinador. Luego se reúnen para discutir las distintas estimaciones, se las vuelva a revisar de forma individual y se repite el paso inicial hasta que se converja a nivel grupal.

### Métodos para estimar: Datos históricos

Como su nombre lo indica, las estimaciones surgen de antiguos proyectos o resultados anteriores. Es decir, se toma una referencia de un hecho pasado y se traslada al nuevo desarrollo.

Se arma un dominio de parámetros (tecnología, dominio del problema, personal) para generar una base de datos para luego extrapolarlo a un nuevo proyecto y realizar las estimaciones correspondientes.

### Métodos para estimar: Analogía

Es una corrección del método anterior en el sentido que dentro de todos los datos históricos solamente se tienen en cuenta aquellos que sean similares a la situación del nuevo proyecto, es decir que se busca una comparación entre casos similares.

## Estimaciones ágiles

En las estimaciones ágiles, las features son estimadas utilizando una medida de tamaño **relativo** (buscan evitar utilizar absolutismos o linealidades a la hora de estimar) denominadas story points. Estas últimas son utilizadas como una medida de esfuerzo, incertidumbre y la complejidad (tamaño) de la historia de usuario.

Se utilizan, entonces, estimaciones relativas. Las personas no estiman en términos absolutos (lo cual es complejo e impreciso) sino que somos buenos comparando cosas. Para ello, se comparan US entre ellas para determinar los esfuerzos necesarios, obteniendo una mejor dinámica grupal y favoreciendo el pensamiento de equipo.

Cabe aclarar que para estimar una US el tamaño (la medida de la cantidad de trabajo necesario para producir esa feature) debe ser pequeño. El tamaño es una medida que nos indica la complejidad, el esfuerzo y cuan grande es una story.

Además, la duración de un proyecto no se estima, se deriva tomando el número total de story points de sus US y dividiéndolo por la velocidad del equipo. Este último concepto (velocidad) es una medida del progreso de un equipo que se calcula sumando el número de story points asignados a cada US que el equipo completa en una iteración.

Para estimar nos basamos en 3 variables que determinaran el peso de la US:

* Complejidad: Dificultad de adaptación con el entorno de programación, cálculos lógicos, diseños de interfaz, entre otros.
* Esfuerzo: Mano de obra que se le debe dedicar a la US para lograr aplicar esa funcionalidad al producto
* Incertidumbre: Desconocimiento sobre el negocio y/o las técnicas necesarias para resolver la US

### Método propuesto para la estimación ágil: Poker planning

El planning poker (o póquer de planificación) es un método de estimación que ayuda a tu equipo ágil a calcular la cantidad de esfuerzo que se necesita para completar una historia de usuario en un backlog del producto. Se utiliza a menudo en metodologías ágiles de gestión de proyectos, y a veces también se lo conoce como “Scrum poker” o “pointing poker”. El término “póker” del nombre hace referencia a las cartas que cada miembro del equipo usa durante el proceso.

Cada carta tendrá uno de estos valores: 1, 2, 3, 5, 8, 13, 21, 44 y 100. Estos números pueden parecer aleatorios, pero en realidad son variaciones redondeadas de la secuencia de Fibonacci. Por lo general, estos valores representan puntos de historia, a mayor número, mayor tamaño, incertidumbre y esfuerzo requiere esa US.

El primer paso es definir una User Story canónica, una US que sirva de comparación para asignarle story points a las demás US. La US a elegir debe ser simple (tanto en esfuerzo como en complejidad) y NO debe tener ningún nivel de incertidumbre, para que la comparación con las demás sea lo más acertada posible. Por lo general, las user stories canónicas son aquellas que poseen una calificación baja.

El proceso de funcionamiento es el siguiente:

1. El propietario del producto (product owner) o el Scrum master actuará como moderador y leerá una historia de usuario específica del backlog del sprint o del producto. Luego, los miembros del equipo tendrán la oportunidad de hacer preguntas y aclarar lo que sea necesario para que todo el equipo pueda tener una idea exacta de las tareas que deben completarse para ese elemento específico del backlog.
2. Una vez que el equipo termina de analizar una historia de usuario, cada estimador elige una carta que corresponde a la cantidad de esfuerzo o puntos de historia que considera que se correlaciona con ese elemento del backlog. Luego, todos revelan la carta al mismo tiempo. Si todos eligieron la misma estimación, esa se convierte en la estimación oficial para ese elemento del backlog.
3. Si los miembros del equipo tienen opiniones diferentes sobre sus estimaciones iniciales, el miembro del equipo con la mayor estimación y el miembro del equipo con la menor estimación deben tomarse un tiempo para debatir por qué eligieron ese número específico. Una vez que se termina el debate, todos vuelven a seleccionar sus cartas. Este proceso se repite hasta que el equipo llegue a un consenso.
4. Dado que todo el equipo ha llegado a un consenso con respecto a cuánto tiempo demandará realizar cada tarea, es mucho más probable que se pueda destinar la cantidad adecuada de trabajo en los sprints.

## Gestión de productos – Lean start-up

La gestión de productos hace referencia a todas aquellas actividades necesarias para crear/lanzar al mercado un producto de software de manera exitosa.

Si analizamos un producto de software, se observará que existe una gran cantidad de funcionalidades que no son utilizadas por los usuarios indicando que se invirtió tiempo, esfuerzo y dinero en características que prácticamente no aportan valor al software final.

En este punto, el objetivo es reducir ese costo excesivo en desarrollar características que no son de interés para los usuarios finales y solo enfocarse en aquellas partes del sistema que podríamos considerar indispensables. Entonces, al pensar en un nuevo desarrollo tenemos una hipótesis que debemos comprender en su totalidad. En este contexto, aparecen algunos conceptos interesantes:

### MVP (Mínimo producto viable)

Es una “parte” del producto final con las mínimas características necesarias para que el usuario pueda evaluarlo y saber si le interesa el producto o no. La premisa clave es se produzca un producto real que se pueda ofrecer a los clientes y observar su comportamiento frente a este. Sirve para controlar la hipótesis del producto a desarrollar, es decir, no tiene como finalidad entregar un producto que sea utilizado, sino para verificar que la hipótesis es viable. Se centra en el valor único de la hipótesis, pero por lo general también posee algunas funciones de table stakes para asegurar la viabilidad del producto.

Un buen MVP posee las siguientes características:

* Tiene valor suficiente para que las personas estén dispuestas a usarlo o comprarlo.
* Demuestra tener un beneficio futuro para retener a los usuarios
* Proporciona un ciclo de retroalimentación para guiar el desarrollo.

A través de la presentación del MVP se inicia un circuito de retroalimentación y aprendizaje, donde se pone a prueba la hipótesis, recepto la evaluación de los usuarios, considero aquellas sugerencias, quejas o incluso críticas al producto y realizo los cambios necesarios. Entonces, pueden suceder los siguientes casos:

* El producto es totalmente aceptado tal cual fue presentado lo que es un indicador positivo de que se entendió la necesidad del mercado y se debe continuar por ese camino.
* Los usuarios difieren totalmente en los requerimientos que solicitan del producto para que les sea útil, lo que implica que quizás no se haya entendido bien la necesidad del mercado o incluso no se este siendo lo suficientemente especifico.
* Muchos usuarios tienen una tendencia a solicitar el mismo requisito, indicando que quizás tenga sentido revisar la hipótesis para construir un MVP2 basado en el aprendizaje del primero.

### MMF (Características mínimas comercializables)

Al obtener una retroalimentación positiva del MVP se debe empezar a pensar en aquellas características necesarias que debe tener el producto para que sea rentable (comercializable). Es decir, encontrar aquellas MMFs que atraigan crecimiento al producto y aporten valor.

El objetivo del MMF es aportar valor, trabajar en aquellas características que permitan disminuir el tiempo de lanzamiento al mercado, que sean atractivas para los usuarios y permiten profundizar en aquellas áreas que sean de utilidad para los usuarios.

### MVF (Mínima característica viable)

Existe un grado de incertidumbre al tratar de encontrar aquellas características que aporten valor, ya no solamente tenemos incertidumbre a nivel producto (que intentamos resolver con el MVP) sino también a nivel funcionalidad. La forma de afrontar esto, es crear una función “pionera”: Mínima característica viable. Es una funcionalidad que es útil para el uso real y para aprender sobre la respuesta de los usuarios finales.

Si una MVF resulta exitosa, se pueden desarrollar MMFs alrededor de esa área para tomar ventaja. Sino, quizás sea más viable enfocarse en otras características alternativas.

### MMP (Mínimo producto comercializable)

Es el primer MMR dirigido a los primeros usuarios focalizado en las características clave para estos interesados.

### MMR (Mínimo reléase comercializable)

Es el lanzamiento del producto que tiene el conjunto de características para ofrecer valor a nuevos usuarios y satisfacer las necesidades actuales.

## Método Lean start-up: Ciclo de retroalimentación

La productividad de un StartUp no puede medirse en términos de cuanto se construye por día, sino que se debe medir en términos de cuantas cosas correctas (funcionalidades que les interesen a los clientes) se construyeron cada día.

Es decir, que el éxito no es entregar un producto, el éxito es entregar un producto que el cliente usará. La forma de hacerlo es alinear los esfuerzos continuamente hacia las necesidades de los usuarios. Para ello se propone el ciclo de retroalimentación Construir-Experimentar-Aprender que permite descubrir las necesidades del cliente y alinear el trabajo de desarrollo con ellas.

Diagrama

Descripción generada automáticamente

# SCM (Software Configuration Management)

## Definición y concepto

La configuración del software (CM) es una disciplina que aplica dirección y monitoreo administrativo y técnico a: identificar y documentar las características funcionales y técnicas de los ítems de configuración, controlar los cambios de esas características, registrar y reportar los cambios y su estado de implementación y verificar correspondencia con los requerimientos.

El propósito de gestionar la configuración del software es el de establecer y mantener la integridad de los productos de software a lo largo de su ciclo de vida. Para ello se requiere:

* Identificar la configuración del software en puntos discretos en el tiempo (en un momento dado)
* Controlar sistemáticamente los cambios (controlando el producto)
* Mantener la integridad y el origen de la configuración

La SCM también se debe planificar, es decir, que el plan debería incluir:

* Reglas de nombrado de los IC
* Herramientas a utilizar para la gestión
* Cuales son los roles y quienes integran el comité
* Procedimiento formal de cambios a seguir
* Plantillas de formularios
* Procesos de auditoría

### Conceptos clave dentro de la SCM

* **Item de configuración:** Se llama ítem de configuración (IC) a todos y cada uno de los artefactos que forman parte del producto o del proyecto, que pueden sufrir cambios o necesitan ser compartidos entre los miembros del equipo y sobre los cuales necesitamos conocer su estado y evolución. Ejemplos: Plan de CM, propuestas de cambio, requerimientos, código fuente, etc.
* **Versión:** la forma particular (estado) de un artefacto en un instante o contexto dado. El control de versiones se refiere a la evolución de un único ítem de configuración (IC), o de cada IC por separado.
* **Variante:** versión de un ítem de configuración (o de la configuración) que evoluciona por separado. Las variantes representan configuraciones alternativas. Un producto de software puede adoptar distintas formas (configuraciones) dependiendo del lugar donde se instale.
* **Configuración del software:** es un conjunto de ítems de configuración con su correspondiente versión en un momento determinado
* **Repositorio:** “lugar” donde se contiene la información de los ítems de configuración como así también su historia con sus atributos y relaciones. Se utiliza para hacer evaluaciones de impacto de los cambios propuestos. Puede encontrarse en una o varias bases de datos. Pueden ser centralizados o descentralizados.
* **Línea de base:** configuración que ha sido revisada formalmente y sobre la cual se ha llegado a un acuerdo. Sirve como base para desarrollos posteriores que solo pueden cambiarse a través de un procedimiento formal de control de cambios. Permiten ir atrás en el tiempo y reproducir el entorno de desarrollo en un momento dado del proyecto. No hay que confundir este concepto con la versión de producto.
* **Ramas:** son “líneas de trabajo” del software. Existe una rama principal pero las ramas se utilizan para bifurcar el desarrollo, permiten experimentar, descartar las inútiles o integrarlas. La integración comúnmente se denomina como merge e implica llevar los cambios de las demás ramas a la principal.

### Actividades fundamentales de la SCM

Existen 4 actividades principales representadas en la siguiente figura:

Diagrama, Escala de tiempo

Descripción generada automáticamente

#### Identificación de ítems de configuración

Es la habilidad de identificar qué información ha sido aprobada para uso concurrente en el proyecto, quien hace uso de la misma y como fue aprobada para el control de CM junto con sus últimos cambios aprobados.

Implica la identificación univoca de cada IC, el establecimiento de convenciones y reglas de nombrado, la definición de la estructura del repositorio y la ubicación de cada uno de los IC en esta última.

Diagrama

Descripción generada automáticamente

#### Control de cambios

El proceso de control mediante el cual cada elemento o trabajo del producto debe pasar. Debe identificar las personas o grupos con la autoridad para autorizar cambios y hacerlos en cada nivel, y también los pasos necesarios para obtener dicha autorización, el proceso para solicitar cambios, el seguimiento de cambios, la distribución de estos y el mantenimiento de versiones pasadas.

Diagrama

Descripción generada automáticamente

Tiene su origen en un requerimiento de cambio a uno o varios ítems de configuración que se encuentran en una línea base. Es un procedimiento formal que involucra diferentes actores y una evaluación del impacto del cambio.

Para la gestión de los cambios existe un comité de control de cambios que esta formado por representantes de todas las áreas involucradas en el desarrollo (análisis/diseño, implementación, testing, otros). Entonces, la propuesta de cambio se puede hacer mediante una ECP (Engineering Change Proposal) y será revisada por el CCB (Control Change Board).

La incorporación del cambio no es una tarea de SCM, pero si lo es la monitorización del cambio.

#### Auditorias de configuración de SW

Es la frecuente evaluación del contenido, integridad de la baseline y de la release de todos los productos controlados para asegurarse que cumplen con su documentación de configuración asociada. La Auditoria de Configuración provee un mecanismo para determinar el grado en que el estado actual del sistema refleja el sistema establecido en la baseline y en los requerimientos de la documentación.

Sirve a dos procesos básicos:

* **Validación**: el problema es resuelto de manera apropiada que el usuario obtenga el producto correcto.
* **Verificación**: asegura que un producto cumple con los objetivos preestablecidos, definidos en la documentación de líneas base (línea base). Todas las funciones son llevadas a cabo con éxito y los test cases tengan status “ok” o bien consten como “problemas reportados” en la nota de reléase.

La intención de las auditorias es aumentar la visibilidad del SW y establecer trazabilidad durante todo el ciclo de vida del producto de SW. Entonces, las auditorias hacen más visible a la gerencia el estado actual del SW auditado. También revela si los requerimientos del proyecto están siendo satisfechos o si lo que se pretende de la próxima baseline se ha cumplido. Con todo esto el PM puede evaluar la integridad del producto de SW desarrollado, resolver conflictos que pueden haber aparecido en la auditoria y corregir defectos. Esta visibilidad también provee una base para establecer una nueva baseline.

Por último, la auditoria provee trazabilidad entre el ciclo de vida del Producto y los requerimientos. Cada requerimiento es tracekado sucesivamente entre baselines.

Las auditorías pueden ser de dos tipos:

* **Auditoria física de configuración (PCA):** aseguran que lo que está indicado para cada ICS en la línea base, o actualización, se ha alcanzado realmente. Además, que el SW y la documentación son internamente consistentes para entregarlos al cliente.
* **Auditoría funcional de la configuración (FCA):** Evaluación independiente de los productos de software, controlando que la funcionalidad y performance reales de cada ítem de configuración sean consistentes con la especificación de requerimientos.

Diagrama

Descripción generada automáticamente

#### Informes de estado (Status accounting)

Es un reporte formal de los documentos de configuración establecidos, el estado de las propuestas de cambios y el estado de la implementación de los cambios aprobados.

Se ocupa de mantener los registros de la evolución del sistema. Por ello, contienen mucha información y salidas por lo que suele implementar dentro de procesos automáticos. Además, incluyen reportes de trazabilidad con todos los cambios realizados a las líneas base durante el ciclo de vida.

El alcance de lo que debe reportar el Status Accounting es:

* El estado de un IC
* Descripción y estado de un cambio
* Objetivos de cada versión de cada IC para saber que requerimiento cumple
* Diferencias entre versiones

### SCM en ambientes ágiles

Se puede aplicar la gestión de la configuración del software siguiendo algunos principios ágiles:

* Individuos e interacciones sobre los procesos y herramientas: los procesos y herramientas utilizados en SCM deben ser soportados por el equipo de trabajo, no se deben imponer procesos rígidos que sean difíciles de aplicar.
* Documentación Lean y trazabilidad: si bien SCM soporta la producción de SW funcionando, se debe enfocar el trabajo en automatizar y asegurar que los procesos adecuados están siendo usados sin requerir una gran cantidad de documentación de proceso.
* SCM se debe adaptar para un seguimiento y coordinación del desarrollo en lugar de controlar a los desarrolladores
* Adaptación al cambio en lugar de evitar cambios o seguir un plan