

Resumen Parcial 1. ISW

 Autor	Tomás Malamud
 Materia	Ingeniería y Calidad de Software
 Profesores	Judith Meles
 Última edición	@September 23, 2024 3:19 PM

RECURSOS

- Para imprimir este resumen →
[https://drive.google.com/file/d/1knYR0ghDVX8w8foMA94i4DVTbQyBCzcQ/view?
usp=sharing](https://drive.google.com/file/d/1knYR0ghDVX8w8foMA94i4DVTbQyBCzcQ/view?usp=sharing)
- En el primer parcial también se evalúa el paper No Silver Bullet. Son sólo 16 páginas y está muy bueno:

[No Silver Bullets - Brooks.pdf](#)

También hice un audio tipo podcast de 10 minutos sobre este documento con NotebookLM:

[https://drive.google.com/file/d/1sNPVOYGA24H8MEElofeia5r4zAyIkij8/view?
usp=share_link](https://drive.google.com/file/d/1sNPVOYGA24H8MEElofeia5r4zAyIkij8/view?usp=share_link)

- Resumen más completo (para mi es más de lo que se necesita saber, pero por las dudas)

Introducción

El software es un conjunto de programas y su documentación. Existen tres tipos:

- Software de Sistemas
- Utilitarios
- Software de Aplicaciones (con el que trabajamos en la materia)

La creación de software es un trabajo de raíz *intelectual*. Depende de la personalidad e intelecto de los miembros del equipo que lo crean.

La **ingeniería de software** es la construcción de múltiples versiones de software realizadas por múltiples personas. Se divide en tres ramas: las disciplinas **técnicas**, las disciplinas **de gestión**, y las disciplinas **de soporte** (SCM, métricas, etc.).

Software ≠ Manufactura

- Software es menos predecible
- No hay producción en masa, casi ningún producto es igual a otro
- No todas las fallas son errores
- No se gasta
- No está gobernado por la física.

Historia

- 1968: Nace el término en una conferencia de la NATO (North Atlantic Treaty Organization)
- 1975: The mythical Man-Month (Frederick Brooks)
- 1978: Tom DeMarco introduce Structured Analysis
- 1980s: Primeros grandes errores de software conocidos
- 1987: No Silver Bullet (Brooks) — Características esenciales del software
- 1989: Managing the Software Projects — Watts Humphrey
- 1990: Internet y OOP (Object-Oriented Programming)
- 1991: CMM 1.0 (Capability Maturity Model)
- 1993: CMM 1.1
- 2000: CMMI 1.0 (Capability Maturity Model Integration) — integró varios modelos CMM
- 2001: Agile Manifesto
- 2003: Lean Software Development

Problemas Comunes en el Desarrollo

- La versión final del producto no satisface las necesidades del cliente

- No es fácil extenderlo o adaptarlo
- Mala documentación
- Mala calidad
- Mayores tiempos y costos de lo presupuestado.

Factores que Influyen en el Éxito de un Producto

- Involucramiento del usuario
- Apoyo de Gerencia
- Enunciado claro de requerimientos
- Planeamiento adecuado
- Expectativas realistas
- Hitos intermedios
- Personas involucradas competentes

Proyectos de Software

El 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. Existen dos tipos: definidos y empíricos.

Procesos Definidos

Asume que se puede repetir el mismo proceso para asegurar un producto de software de calidad. La **administración y control** provienen de la **predictibilidad** del proceso definido. Un proceso conocido de este tipo es el Proceso Unificado de Desarrollo (PUD). El ciclo de vida en cascada trabaja también en línea con los procesos definidos.

Procesos empíricos

Asumen procesos complicados con variables cambiantes. Si el proceso se repite, los resultados obtenidos pueden ser diferentes. Estos procesos se ajustan de mejor forma a procesos creativos y complejos. La **administración y el control** es por medio de **inspecciones y adaptaciones**.

Tienen 3 pilares:

1. **Inspección.** Evaluación y revisión coherente. Puntos de inspección de scrum:
 - a. Planificación de sprint
 - b. Daily Scrum
 - c. Sprint Review
 - d. Sprint Retrospective
2. **Adaptación.** Una vez que el equipo ha inspeccionado el producto y los procesos, adapta sus estrategias en función de los conocimientos adquiridos. Diversas fases en scrum:
 - a. Modificación del Sprint Backlog
 - b. Adaptaciones en las Daily Scrum
 - c. Comentarios durante la Sprint Review
3. **Transparencia.** Permite crecer como equipo. *Mi código no es mi código, es del equipo.*
En scrum se manifiesta en diversas facetas:
 - a. Sprint Backlog
 - b. Product Backlog
 - c. Sprint Review

Patrón de conocimiento de Procesos Empíricos

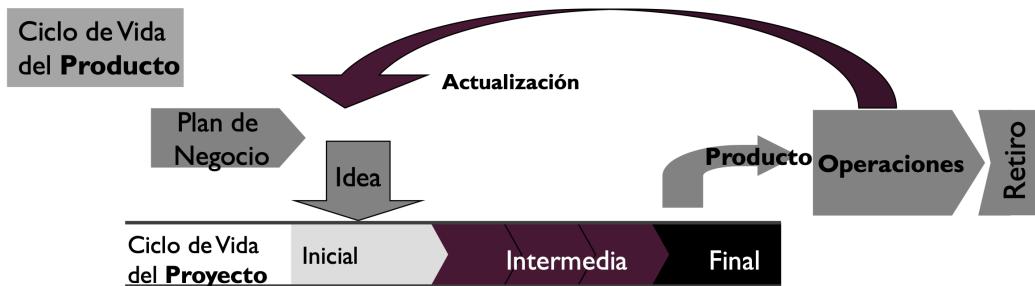
1. Asumir
2. Construir
3. Retroalimentar
4. Revisar
5. Adaptar
6. Volver a punto 1 y repetir

El conocimiento puede ser individual o grupal, y explícito o implícito:

1. Explícito, Individual está relacionado a **hechos**
2. Explícito, Grupal está relacionado a **buenas prácticas**
3. Implícito, Individual está relacionado a **habilidades**
4. Explícito, Grupal está relacionado a **prácticas de trabajo**

Ciclos de Vida

Un ciclo de vida es la serie de pasos a través de los cuales el producto o proyecto progresa. Es decir, el producto como el proyecto tienen ciclos de vida.



Ciclo de vida de producto y de proyecto. Para un ciclo de vida de producto pueden existir múltiples ciclos de vida de proyecto.

Un ciclo de vida de un proyecto de software es una representación de un proceso desde una perspectiva particular.

Los modelos especifican las **fases del proceso** y el **orden** en el que se ejecutan. Por ejemplo, requerimientos → especificación → diseño, etc.

Existen tres tipos principalmente:

1. **Secuencial** (cascada)
2. **Iterativo/Incremental**. PUD sugiere este tipo de ciclo de vida. En los procesos **definidos** suelen ser iteraciones con tiempo **variable** (dependen del alcance de la iteración), mientras que en procesos **empíricos** se suelen utilizar iteraciones con tiempo **fijo** (sprint en Scrum por ejemplo).
3. **Recursivo**

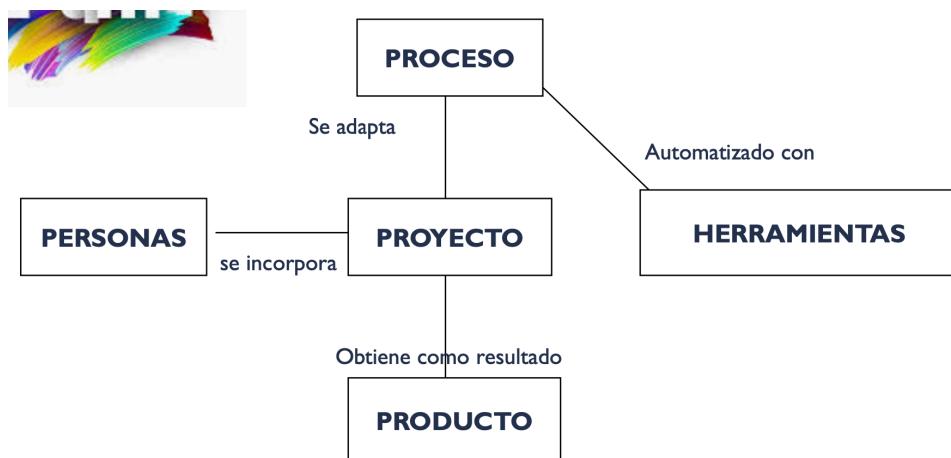
Existen más tipos de ciclos de vida, como se detalla en el libro de McConnell (1996, "Rapid Development: Taming Wild Software Schedules") disponible en la UV:

- Cascada Pura: Una aproximación secuencial donde cada fase debe completarse antes de que comience la siguiente.
- Code-and-fix: Un modelo simple donde los desarrolladores escriben código y arreglan problemas conforme surgen, con mínima planificación.
- Espiral: Un enfoque iterativo que combina elementos de cascada y prototipado, centrándose en la evaluación de riesgos.

- Cascadas Modificadas: Variaciones del modelo de cascada que permiten cierta flexibilidad e iteración entre fases.
- Prototipado Evolutivo: Un modelo donde un prototipo se refina continuamente hasta convertirse en el producto final.
- Entrega por Etapas: Un método donde el producto se desarrolla y entrega en etapas, cada una añadiendo funcionalidad.
- Diseño según Calendario: Un enfoque que prioriza cumplir con una fecha límite específica, ajustando características según sea necesario para ajustarse al cronograma.
- Entrega Evolutiva: Similar al prototipado evolutivo, pero enfocado en entregar versiones funcionales a los usuarios regularmente.
- Diseño según Herramientas: Un modelo que aprovecha herramientas y entornos de desarrollo específicos para guiar el proceso de desarrollo.
- Software off-the-shelf: Uso de soluciones de software pre-existentes que pueden ser personalizadas o integradas en sistemas más grandes.

Las 4 “P”

En un proceso definido, el proyecto es una *instancia* del proceso—el proyecto *se adapta* al proceso. En procesos empíricos, el proyecto *define* su propio proceso.



Modelo de las 4 “P”. Aplica para ambos tipos de procesos, con la salvedad de que en procesos definidos el proyecto *se adapta* al proceso, mientras que en los empíricos el proyecto *define* su propio proceso.

Proyectos

Tienen 4 características:

- **Orientados a Objetivos.** Los proyectos están dirigidos a obtener resultados, lo cual se ve reflejado a través de objetivos claros y alcanzables.
- **Duración Limitada.** Cuando se logra el objetivo, se termina el proyecto.
- **Tareas interrelacionadas basadas en recursos y esfuerzos.** Los problemas tienen una complejidad sistémica inherente. Se debe definir recursos y esfuerzos requeridos.
- **Únicos.** Los proyectos comienzan y terminan, teniendo componentes únicos: personas únicas, recursos únicos, procesos únicos, etc.

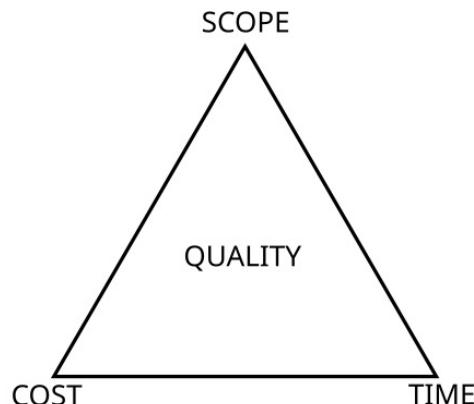
Administración de Proyectos

La administración de proyectos es la **aplicación de conocimientos, habilidades, herramientas y técnicas** a las **actividades del proyecto** para **satisfacer los requerimientos** del proyecto. Esta actividad incluye:

- Identificar requerimientos
- Establecer objetivos
- Adaptar especificaciones, planes, y enfoque a diferentes stakeholders.

Triple Constraint

Está muy bien descrito en [Wikipedia](#).



Triple restricción, Triple constraint, o Triángulo de Hierro.

Establece que:

- La calidad del trabajo está restringida por el **costo, tiempo y alcance** (objetivos).
- Existe un compromiso entre las restricciones, que debe ser manejado por el líder de proyecto.

- Cambios en una restricción necesitan cambios en las otras para compensar, o la calidad podría verse disminuida. Por ejemplo, si disminuyo el presupuesto: o aumento el tiempo disponible, o disminuyo el alcance. Si aumento el alcance: o aumento el presupuesto o aumento el tiempo.

Equipo de Proyecto

Está liderado por un **Líder de Proyecto**, quien a su vez interactúa a nivel interno y externo a la organización:

- Interno: niveles superiores de administración y gerentes funcionales
- Externo: cliente, subcontratistas, entes reguladores.

El equipo de proyecto es un grupo de personas comprometidas en alcanzar un conjunto de objetivos de los cuales se sienten mutuamente responsables.

Plan de Proyecto

El plan de proyecto documenta *qué* se hace, *cuándo*, *cómo*, y *quién* lo va a hacer. Implica las siguientes tareas:

- Definición de alcance
 - Alcance de proyecto y de producto.
 - El alcance del *proyecto* se mide contra el plan de proyecto.
 - El alcance del *producto* se mide contra la especificación de requerimientos (ERS).
- Definición de proceso y ciclo de vida
 - Anteriormente definimos qué hacer, por lo que en esta etapa se define cómo y cuándo de cada tarea hacer. Por ejemplo, el PUD sugiere el ciclo de vida iterativo.
- Estimación
 - Estimamos tamaño, esfuerzo, costo, calendario, y recursos.
- Gestión de Riesgos
 - Es un “problema esperando suceder”, comprometiendo el éxito del proyecto
 - En base a una **base de conocimiento de riesgos**, identificar el riesgo, realizar la especificación de riesgos, y en base a eso ingresar al loop
Analizar → Plan → Seguridad y Control → Aprendizaje para todos los riesgos prioritarios.
- Asignación de Recursos

- Programación de Proyectos
- Definición de Controles
 - Un proyecto se atrasa “de a un día por vez”.
 - Un gran proyecto con una planificación de tiempo ajustada se controla en primer lugar *teniendo un plan*. Debe incluir hitos o *milestones* muy claros y fácilmente verificables.
- Definición de Métricas
 - Existen métricas de **proceso**, de **proyecto**, y de **producto**. **Las de proyecto se consolidan para crear métricas de proceso** que sean públicas para toda la organización.
 - Las métricas básicas de proyecto son: tamaño, esfuerzo, tiempo y defectos.
 - Por ejemplo, el desarrollador podría tener métricas como % de cobertura del unit test, defectos por unit test, esfuerzo y duración estimada vs actual de una tarea
 - El equipo podría tener métricas como distribución del esfuerzo, status de requerimientos, y % de tests ejecutados correctamente
 - La organización podría tener como métricas el rendimiento actual vs planificado tanto del esfuerzo como del costo.
 - Las métricas sirven para balancear costo, tiempo y alcance en base a las restricciones de tecnología, personas y procesos.

Filosofía Ágil

Es un compromiso entre ningún proceso y demasiado proceso. La diferencia inmediata es la necesidad de una menor cantidad de documentación, aunque eso no es lo más importante. Lo más importante es lo siguiente:

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

Valores Ágiles

Son 4 valores. Todo es importante, pero se valora más lo que está a la izquierda:

- Individuos e interacciones > procesos y herramientas
- Software funcionando > documentación extensiva

- Colaborar con el cliente > negociación contractual
- Responder al cambio > Seguir un plan

“>” se lee “por sobre”

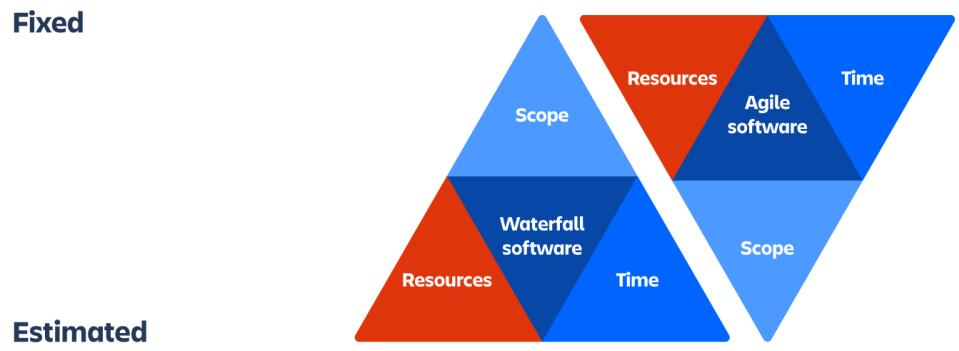
12 Principios del Manifiesto Ágil

1. Nuestra mayor prioridad es **satisfacer al cliente**.
2. **Aceptar** que los requisitos **cambien**.
3. **Entregar** software funcional **frecuentemente**.
4. 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**.

Ágil es una ideología (no una metodología o proceso) con un conjunto definido de principios que guían el desarrollo del producto. Algunos valores de los equipos ágiles son la planificación continua; equipos completos auto-organizados; entregas frecuentes, iterativas y priorizadas; integración continua, etc.

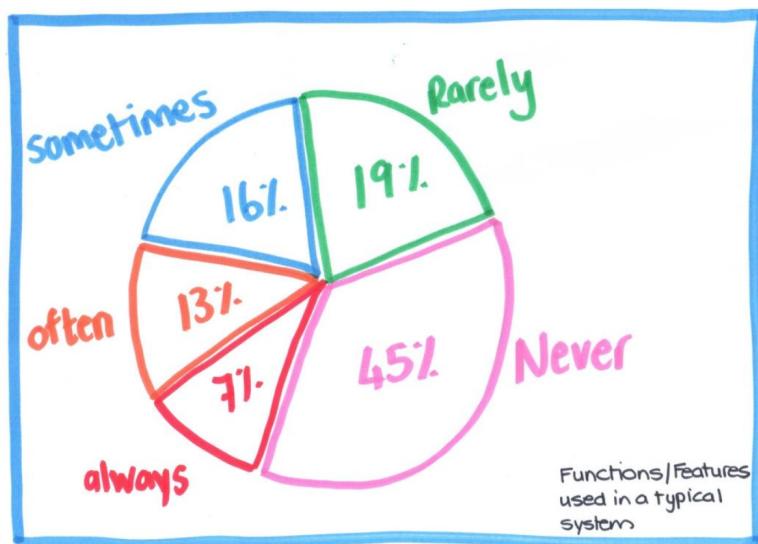
Triángulo Ágil

El modelo en cascada asume que los alcances son fijos, y lo que varía son los recursos y el tiempo. En cambio, el modelo ágil *sabe que los alcances van a cambiar*, por lo que fija los recursos (equipo) y el tiempo (sprints) para definir cuánto se puede construir del producto. Se *acuerda* el alcance para los recursos y tiempo definidos.



Diferencia entre triángulo de hierro o tradicional (izquierda) y triángulo ágil (derecha).

Requerimientos Ágiles



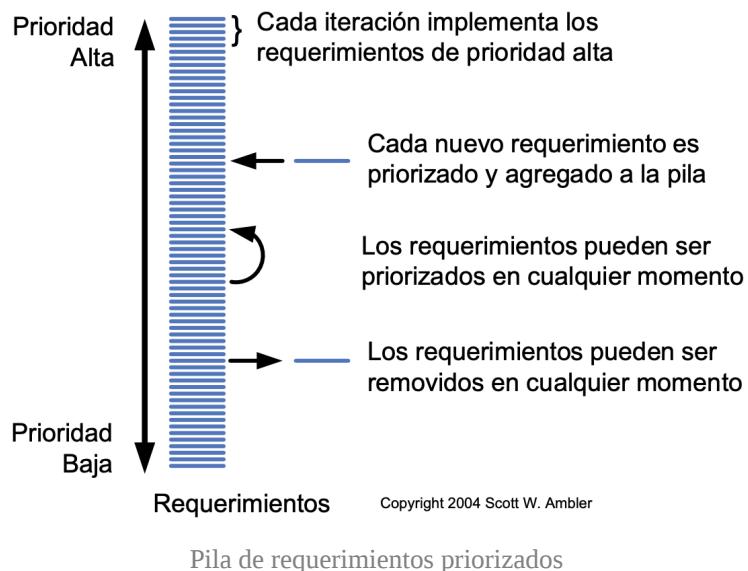
Funciones usadas en un sistema típico. Sólo se usa un 20-36% de las funcionalidades.

Se basan en las siguientes ideas:

- Usar el *valor de negocio* para construir el producto
- Usar historias y modelos para mostrar qué construir
- Determinar qué es solo lo suficiente (no hacer de más, relacionado al gráfico anterior)

Gestión Ágil de los Requerimientos

Los requisitos cambiantes son una ventaja competitiva si se puede actuar sobre ellos. El Product Owner (PO) define los requerimientos con un ordenamiento por prioridad.



También está relacionado al concepto de JIT (Just In Time). Se deben analizar los requerimientos al momento adecuado, ni antes ni después. Otra forma de decirlo es que *se difiere la toma de decisiones hasta el último momento responsable*.

Tipos de Requerimientos

En el dominio del problema:

- **Requerimiento de Negocio:** Disminuir un 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.

En el dominio de la solución (requerimientos de software):

- **Requerimiento Funcional:** Generar reportes de saldo de cuenta. Recibir Notificaciones por email.
- **Requerimiento No Funcional:** Formato del reporte PDF. Cumplir con niveles de seguridad según ley bancaria x.
- **Requerimiento de Implementación:** Servidores en la nube.

Principios Ágiles y Requerimientos Ágiles

1. La prioridad es satisfacer al cliente a través de releases tempranos y frecuentes (2-4 semanas).
2. Recibir cambios de requerimientos, aún en etapas finales.

3. Técnicos y no técnicos (PO) trabajando juntos durante todo el proyecto.
4. El mejor medio de comunicación es cara a cara.
5. Las mejores arquitecturas, diseños y requerimientos surgen de equipos auto-organizados.

User Stories



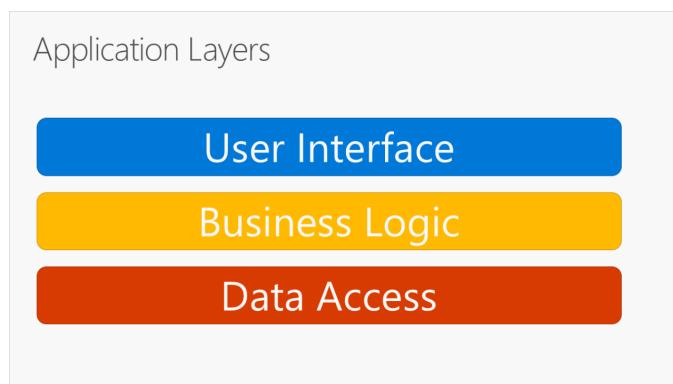
Casi todo lo de la presentación de la profe está sacado de [este paper](#). Es importante leerlo completo (son 16 páginas).

Una User Story (US) es un enunciado breve de intención que describe algo que el sistema debe hacer por el usuario (2009, Leffingwell).

Lo que se escribe en una tarjeta no es importante, pero si las conversaciones entre usuarios y desarrolladores. Se ven reflejadas como:

- Necesidad de Usuario
- Descripción del producto
- Ítem de planificación
- Token para una conversación
- Mecanismo para diferir una conversación

Una US abarca "porciones verticales", incluyendo presentación, lógica de negocio y base de datos. Es preferible crear US pequeñas que cubran todas las capas (por ejemplo, una pantalla específica, sus funcionalidades y la tabla de base de datos correspondiente) en lugar de US separadas para cada capa.

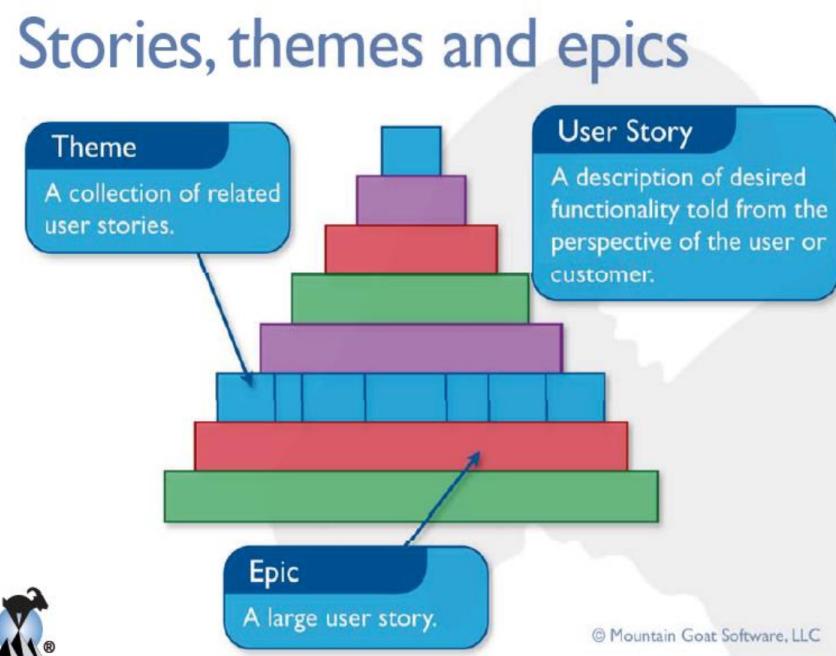


Capas de una aplicación.

User Stories no son Requerimientos

- No son requerimientos detallados, son *expresiones negociables de intención* (debe hacer algo *como esto*)
- Son cortos y fáciles de leer, tanto como para el equipo, stakeholders y usuarios.
- Representan incrementos pequeños de funcionalidad valiosa que pueden ser desarrollados en días o algunas semanas.
- Son relativamente fáciles de estimar, por lo que el esfuerzo para implementar una funcionalidad puede ser fácilmente determinado
- No se guardan en documentos largos, se organizan en listas que pueden ser reorganizadas a medida que se descubre más información
- No son detalladas al inicio del proyecto, sino que son elaboradas JIT. Así, se evita especificidad muy temprana, demoras en el desarrollo, y una solución restringida por demás.
- Necesitan poco o nulo mantenimiento, y pueden ser descartadas luego de la implementación.
- US y el código creado sirven como entrada a la documentación.

Además, presentan distintos niveles de abstracción, siendo una **épica** una US “grande” y un **tema** una colección de US relacionadas.

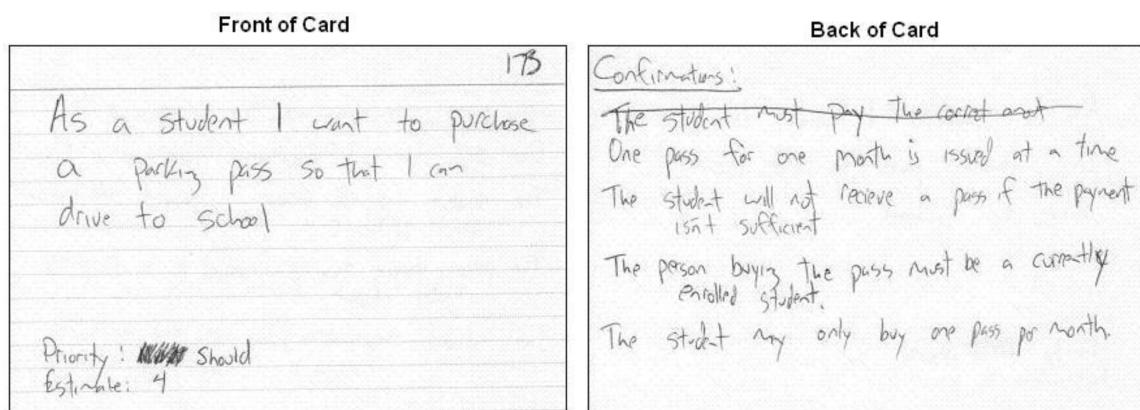


Niveles de abstracción. Jerarquía Tema > Épica > US

Componentes

Son 3 “C”:

- Conversation: la más importante. No queda grabada porque la conversación es cara a cara.
- Card. Formato del frente: “Como **nombre del rol**, yo puedo **actividad** para **valor de negocio**.” El valor de negocio es útil para priorizar la user story en el product backlog.
- Confirmation. Pruebas que expresan y documentan detalles que se pueden utilizar para determinar el grado de avance.



Formato conceptual de la tarjeta. En el dorso se escriben las confirmaciones (criterios de aceptación)

Modelado de Roles

Existen distintas técnicas:

1. Tarjeta de Rol de Usuario

Nos es un experto en computadoras, pero bastante adepto a utilizar la Web. Utilizará el software con poca frecuencia pero muy intensamente. Leerá anuncios de otras compañías para averiguar cuál es la mejor palabra para sus anuncios. La facilidad de uso es importante, pero más importante es que lo que aprenda, lo pueda recordar meses después.

2. Persona (suele incluir fotos de la persona “tipo” también)

Mario trabaja como reclutador en el departamento de Speedy Networks, una fábrica de componentes de red de alta gama. El ha trabajado para Speedy Networks por 6 años. Mario tiene un arreglo de horario flexible y trabaja desde casa cada viernes. Mario es muy fuerte con las computadoras y se considera a sí mismo un usuario avanzado de los productos que usa. La esposa de Mario, Kim, está terminando su Doctorado en Química en la Universidad de Stanford. Dado que Speedy Networks ha estado creciendo consistentemente, Mario siempre está buscando ingenieros.

3. Personajes Extremos

Diseño de un PDA para:

- El Papa
- Una mujer de 20 años con muchos novios
- Un traficante de drogas

Tanto la mujer como el traficante desearán mantener agendas separadas en caso de que la vea la policía o un novio. El Papa probablemente tenga menos necesidad de discreción pero querrá un tamaño de fuente más grande.

Si el Product Owner no está disponible, debe seleccionar un **usuario representante (proxy)**.

Entre ellos, se destacan los siguientes perfiles *de negocio*:

- Gerentes de Usuario
- Vendedores
- Clientes
- Alguien de Marketing
- Capacitadores y personal de soporte

Criterios de Aceptación

Los criterios de aceptación (CA) definen límites para una US, y ayudan a que:

- los PO respondan lo que necesitan para que la US provea valor
- el equipo tenga una visión compartida de la US
- los desarrolladores y testers deriven las pruebas
- los desarrolladores sepan cuando dejar de agregar funcionalidades a la US

Un criterio de aceptación de calidad:

- Define una intención, no una solución.
- Es independiente de la implementación
- Relativamente de alto nivel: no van los detalles

Los detalles (formatos, límites, denominaciones, etc.) van en documentación interna de los equipos y/o en pruebas de aceptación automatizadas.

Pruebas de Aceptación

Expresan detalles resultantes de la conversación y complementan la US.

Es un proceso de dos pasos:

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

Definición de Listo (Definition of Ready)

Es una medida de calidad que constituye el equipo en conjunto para determinar que la US está en condiciones de entrar a una iteración de desarrollo. Como *mínimo*, se requiere el modelo INVEST.

INVEST

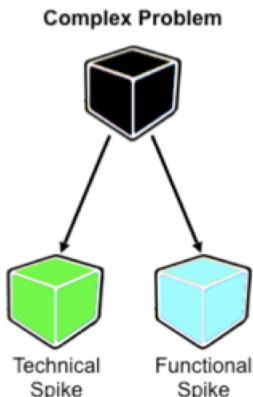
- Independent: no depende de otras US, se implementan en cualquier orden y son calendarizables.
- Negotiable: US escrita en términos de *qué* y no de *cómo*.
- Valuable: debe estar el *para qué*. Debe tener valor de negocio.
- Estimable: asignar un número (Story Points) que permita estimar tamaño relativo.
- Small: depende del equipo, pero deben ser consumidas en una iteración.
- Testable: relacionado a las pruebas de aceptación. Se debe poder demostrar que la US cumple con los criterios de aceptación.

Spikes

Son un tipo especial de story que es usada para disminuir el riesgo e incertidumbre en una US u otra faceta del proyecto. Pueden utilizarse para:

1. Hacer una investigación básica para familiarizar al equipo con una nueva tecnología o dominio.
2. Analizar el comportamiento de la US, con el objetivo de dividirla en piezas estimables.
3. Ganar confianza en un enfoque tecnológico que permita comprometer la US en alguna timebox futura, investigando o haciendo prototipos.
4. La US puede contener riesgo funcional, en el que no queda claro cómo el sistema debe interactuar con el usuario para obtener el beneficio esperado.

Spikes Técnicos y Funcionales



2 tipos de spikes

Spike Técnicos. Utilizados para investigar distintos enfoques tecnológicos en el dominio de la solución. Por ejemplo, un spike técnico puede ser usado para determinar decisiones como “tercerizar una pieza de software o hacerla nosotros”, o “impacto en performance de nueva US”, o por cualquier razón que el equipo necesite desarrollar una comprensión más confiable de un enfoque deseado antes de comprometerse con una nueva funcionalidad.

Spikes Funcionales. Utilizados cuando existe una incertidumbre significativa sobre cómo un usuario podría interactuar con el sistema. Suelen ser evaluados mediante prototipos.

Algunas US pueden requerir los dos tipos de spikes. Veamos la siguiente US:

Como consumidor, quiero ver uso de energía diario en un histograma, para que pueda entender mi pasado, presente, y posiblemente el consumo futuro cercano.

En este caso, el equipo podría crear dos spikes:

- **Spike Técnica:** Investigar cuánto demora actualizar la pantalla del cliente con el consumo actual, determinando requerimientos de comunicación, ancho de banda y los datos se obtienen por *pull* o *push*
- **Spike Funcional:** Hacer un prototipo de un histograma en el portal web y obtener feedback del usuario sobre el tamaño de presentación, estilo y atributos del gráfico.

Lineamientos para Spikes

- Debe ser *estimable, demostrable, y aceptable*.
 - Al igual que las US, las spikes son registradas en el product backlog, se pueden estimar y dimensionar para que encajen en una iteración. Los resultados de una spike

son diferentes a los de US—generalmente producen *información*, y no código funcionando.

- El resultado de una spike es demostrable al equipo. Esto construye responsabilidad compartida sobre las decisiones que se toman.
- Como cualquier otra US, las spikes son aceptadas por el PO.
- Son la *excepción, no la regla*
 - Toda US tiene incertidumbre y riesgo, que son eliminadas mientras el equipo descubre la solución correcta mediante la conversación, la experimentación y la negociación. Sin embargo, las Spikes deberían ser reservadas sólo para los puntos de riesgo e incertidumbre críticos con grandes incógnitas.
- Considerar la *implementación de la spike en un sprint separado de las US resultantes*
 - Básicamente no tener la US y su spike relacionada en el mismo sprint. Se puede hacer sólo si la spike es pequeña o si la solución se encuentra e implementa rápidamente.

Estimaciones Ágiles

Las stories son estimadas utilizando una medida de tamaño *relativo* conocido como Story Points (SP). Las SP son una unidad de medida de **complejidad, incertidumbre y esfuerzo**. La medida de *complejidad* es *independiente de quién resuelva* la story, pero el *esfuerzo* sí *depende* enteramente de eso.

El tamaño es una medida de cantidad de trabajo requerida para producir una story, e indica: **cuán compleja es la story, cuánto trabajo es requerido para completar una story, y cuán grande es una story**. Es independiente de quién lo haga, y a cada persona le puede llevar un *esfuerzo* diferente.

Los temas son más grandes que las épicas, y se miden normalmente en S, M, L o XL. Las US se suelen medir según la serie de Fibonacci (ej., 1, 1, 2, 3, 5, 8, 13, ...). Normalmente a partir de 13 se debe dividir la US.

La estimación en SP separa completamente la estimación de esfuerzo de la estimación de duración.

Velocidad

Es una métrica de progreso del equipo. Se calcula como la suma de story points de las US que el equipo completa durante la iteración. No se estima, sino que sirve para corregir errores de

estimación.

Es muy útil también para conocer la **duración de un proyecto**. La duración de un proyecto se *deriva* en base al número total de SP de sus n US y dividiéndolo por la velocidad:

$$\text{Duración del proyecto} = \frac{\sum_{i=1}^n \text{SP}}{\text{Velocidad}}$$

Las US se convierten entonces en un plan de lanzamiento de la siguiente forma:

User Stories Estimadas → Estimación de tamaño en SP → Derivar duración → **Release Plan**

Así, la velocidad permite definir un *horizonte de planificación*.

Métodos de Estimación

- Basados en experiencia: datos históricos, juicio experto (puro o Delphi), analogía.
- Basados en recursos
- Basados en el mercado
- Basados en los componentes del producto o proceso de desarrollo
- Métodos algorítmicos

Juicio Experto

- **Puro**
 - Un experto estudia las especificaciones y hace su estimación
 - Para estructurarlo se usa la fórmula estimación PERT: $(o + 4h + p)/6$, donde o es optimista, h es habitual y p es pesimista. Las unidades pueden ser de costo o de tiempo. Es básicamente un promedio ponderado, donde lo habitual pesa 4 veces más que el tiempo/costo pesimista u optimista.
- **Wideband Delphi**
 - Un grupo de personas son informadas y tratan de adivinar lo que costará el desarrollo en términos de esfuerzo y duración
 - Especificaciones a un grupo de expertos → Discuten producto y estimación → Remiten sus estimaciones al coordinador → Se comparten anónimamente las estimaciones entre los estimadores → Se reúnen de nuevo → Revisan estimaciones y las envían al coordinador → Se repite.

Poker Planning

Es una propuesta de método de estimación ágil. Combina la *opinión de experto*, la *analogía*, y la *desagregación*. Los participantes son desarrolladores. Es una estrategia de gamificación que vemos principalmente en práctico.

Gestión de Productos

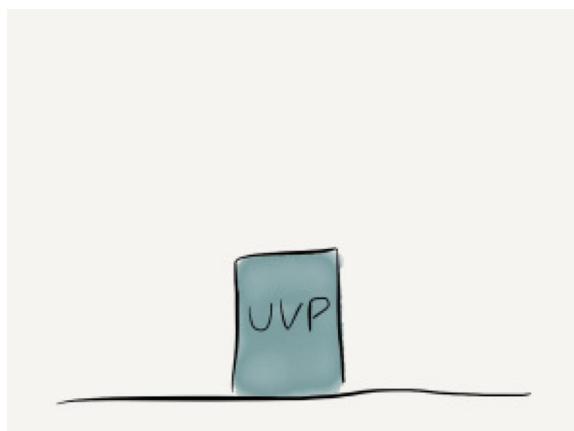


Evolución de los productos de software, desde focalizados en tareas hacia focalizados en experiencias.

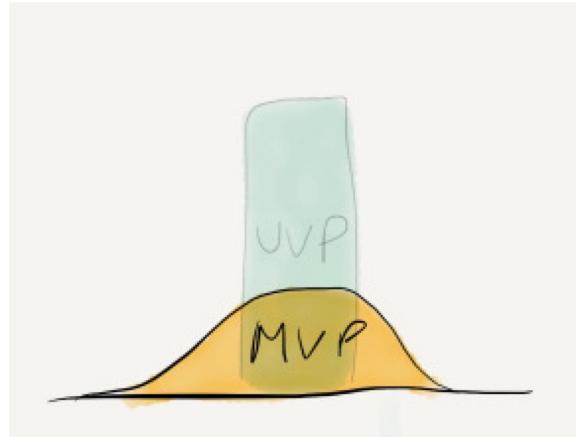
Relación entre UVP, MVP, MVF, MMF

Obtenido de la [referencia original](#).

1. **Identificar la Unique Value Proposition (UVP).** Un producto tiene una hipótesis de valor único.



- 2. Crear el Minimum Viable Product (MVP) para probar su hipótesis.** Está enfocado en la UVP, incluyendo normalmente algunas *features* básicas para asegurar su viabilidad.



- 3. Evaluar la hipótesis del MVP.** El MVP también funciona como hipótesis. Puede ser suficiente para encontrar el Product-Market Fit o no. Si cada cliente potencial dice "Esto es genial, pero necesito X", indica que aún no se ha alcanzado el Product-Market Fit.



- 4. Pivot.** Si por el contrario, se observa que hay muchas solicitudes de una feature que no incluimos en el MVP, entonces vale la pena revisar la hipótesis y comenzar un MVP2



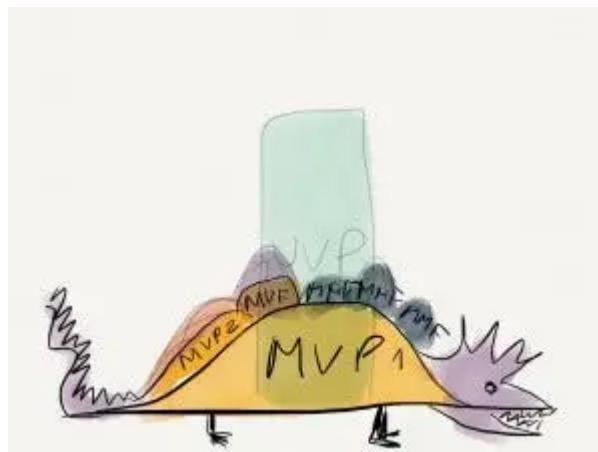
5. **Crecimiento Constante con Minimally Marketable Features (MMF).** Las MMF se enfocan en entregar las unidades de funcionalidad mínimas que brinden valor y generen crecimiento, permitiendo un Time to Market más rápido y la flexibilidad para cambiar de rumbo (pivot, punto anterior) entre distintas áreas de desarrollo de producto.



6. **Experimentación con Minimum Valuable Feature (MVF).** Hipótesis centrada en una característica específica. Se construye un MVF para aprender sobre el uso real de los clientes. Si alcanza el valor requerido, puede convertirse en MMF. Funciona como un mini-MVP, implementándose rápidamente con recursos mínimos para early adopters, probando su utilidad y adopción.



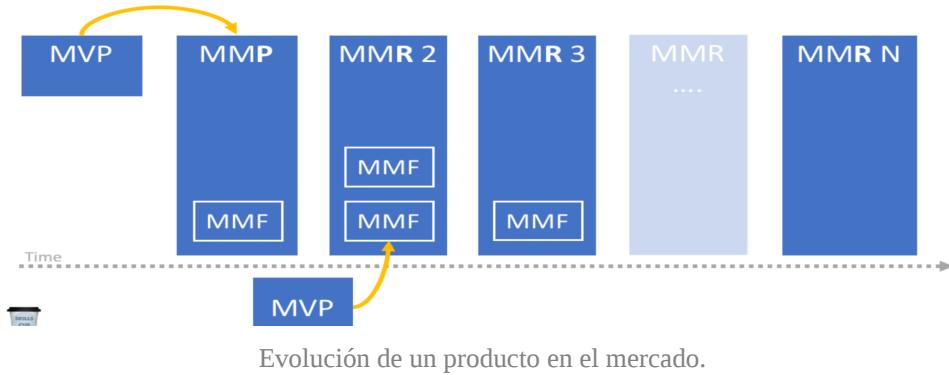
7. Modelo Completo. Un producto crece en mercados desconocidos mediante la experimentación con varios MVPs. Luego, cuando se alcanza Product-Market Fit, se mezclan MMFs y MVFs dependiendo del nivel de la incertidumbre de negocio o requerimiento en las áreas en las que estamos enfocados.



A su vez, se pueden dividir en User Stories para reducir el riesgo de ejecución, las cuales pueden o no tener valor tangible para el negocio.

Relación entre MVP, MMP, MMR

El MVP debe evolucionar hacia un producto comercializable: el Minimally Marketable Product (MMP). El MMP está compuesto por al menos un MMF, aunque generalmente incluye varios MMFs, y está dirigido a los primeros usuarios. Este MMP constituye el primer lanzamiento del producto, conocido como Minimally Marketable Release 1 (MMR1) — es decir, $MMP = MMR1$. Si el producto se desvía de su trayectoria original, se introduce un nuevo MVP. Dependiendo del éxito de este nuevo MVP, podría integrarse en el MMR2 (segundo lanzamiento). Este proceso continúa generando futuros lanzamientos mientras el producto siga existiendo.



MVP

Eric Ries definió al MVP en su libro Lean Startup como:

“versión de un nuevo producto que permite a un equipo **recopilar la cantidad máxima de aprendizaje** validado sobre clientes con el menor esfuerzo.”

Es un producto real que se puede ofrecer a clientes y observar su uso, basándose en la premisa de que analizar qué hace la gente con un producto es más eficiente que preguntarle cómo lo usarían.

Debe tener el valor suficiente como para que las personas estén dispuestas a **usarlo** o pagarla **inicialmente**, demostrando beneficio futuro suficiente para que **retener** usuarios. El MVP proporciona un ciclo de retroalimentación para guiar el desarrollo futuro.

Un MVP no sólo habla sobre el diseño del producto y las preguntas técnicas, sino que también sirve para probar hipótesis comerciales fundamentales.

Además, puede tener distintos niveles de desarrollo: desde un Smoke test (donde no hay producto) hasta un prototipo funcionando.

Características

- Para **preparar un MVP**, se realizan las siguientes actividades:

1. Encontrar un Nicho
2. Crear un Roadmap Realista
3. Investigar Competencia
4. Pre-vender MVP
5. Testear Supuestos

6. Asegurarse que resuelve el problema correcto
 7. Focalizar en funcionalidades Principales
- Composición de un MVP
 1. Funcionalidad: Tiene las funcionalidades necesarias y suficientes para resolver el problema.
 2. Confiabilidad: Involucra a early adopter que confíen en la solución.
 3. Usabilidad: Útil para público objetivo con valor.
 4. Diseño: Buena UX/UI.
 - Matriz de urgencia-importancia para desarrollarlo:
 - Importante, Urgente: Hacer (incluir en MVP)
 - Importante, No Urgente: Planear (desarrollar para un lanzamiento beta)
 - No importante, Urgente: Delegar (integraciones con terceros)
 - No Importante, No Urgente: Eliminar.

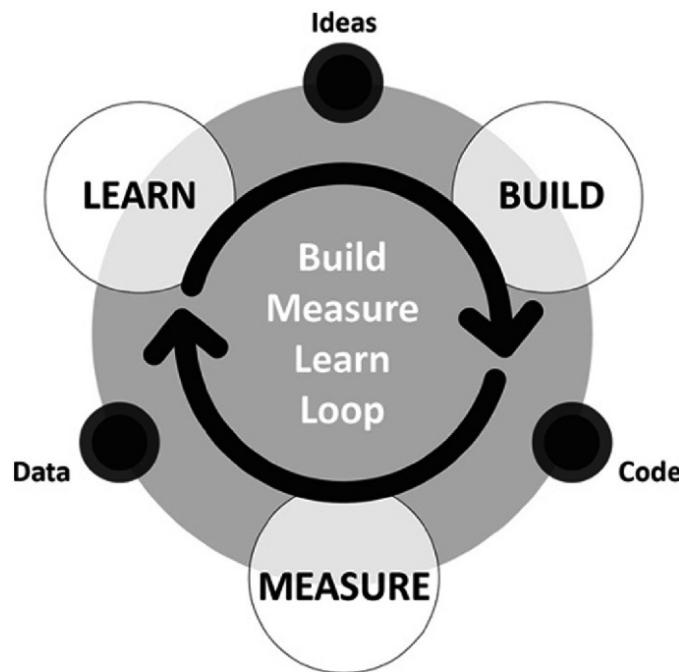
Metodología Lean Startup



Obtenido del libro The Lean Startup:

<https://ia800509.us.archive.org/7/items/TheLeanStartupErickRies/The%20Lean%20Startup%20-%20Erick%20Ries.pdf>

Un componente clave de la metodología Lean Startup es el **build-measure-learn feedback loop** (construir-medir/experimentar-aprender). El primer paso es identificar el problema que debe ser resuelto y luego desarrollar el MVP para comenzar el proceso de *aprendizaje* lo más rápido posible. Una vez que el MVP está establecido, se puede trabajar en “ajustar el motor”. Esto involucra medición y aprendizaje y debe incluir métricas accionables que puedan demostrar una relación causa-efecto.



La actividad fundamental de una startup es convertir ideas en productos, medir cómo responden los clientes y así aprender si hacer un pivot o continuar. *Code* es equivalente a *Product*.

En la etapa de **medición**, puede darse el **dilema de audacia cero**: suele ser más fácil recaudar dinero sin tener ingresos, clientes ni tracción que cuando se tiene una pequeña cantidad de cada uno. Esto es porque el cero *invita a la imaginación*, pero los números pequeños generan dudas sobre si se podrían alcanzar números grandes. *Crea un incentivo negativo*: aplazar el lanzamiento de cualquier versión del producto hasta que su éxito sea seguro. Entonces, si se pospone la experimentación con el MVP pueden surgir resultados desafortunados: mayor cantidad de trabajo desperdiciado construyendo algo que nadie quiere.

Supuestos de Saltos de Fe (leap-of-faith assumptions): Elementos más riesgosos del plan o concepto de una startup, de los que todo depende. Llevan a dos tipos de supuestos:

- **Hipótesis de Valor.** Prueba la entrega de valor por parte del producto siguiendo métricas como tasa de retención.
- **Hipótesis de Crecimiento.** Prueba cómo nuevos clientes descubrirán el producto siguiendo métricas como el Net Promoter Score (NPS)

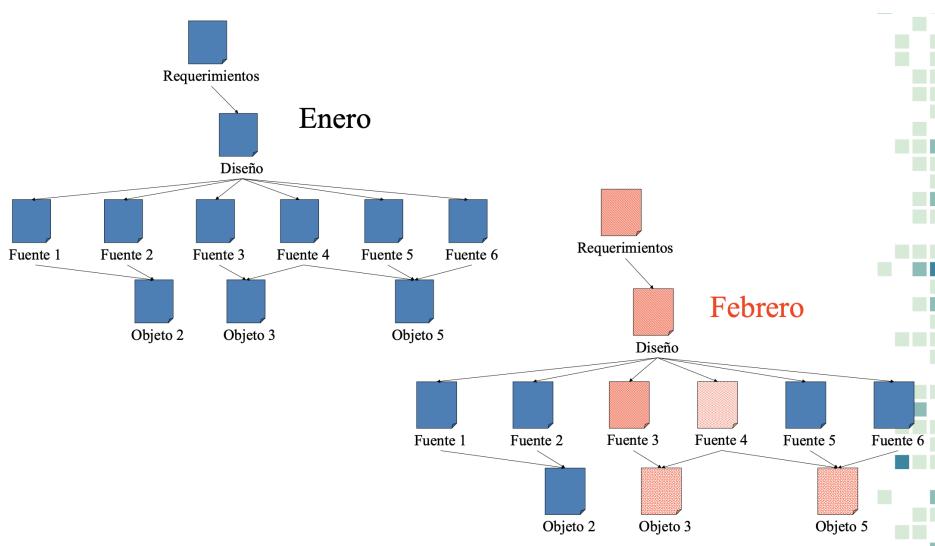
Una vez que se resuelven los supuestos, el primer paso es entrar a la etapa de construcción lo más rápido posible.

SCM

Software de Configuración de Software

Gestión de Configuración de Software (GCS): 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**. ANSI/IEEE 828

El software es información estructurada con propiedades lógicas y funcionales, creada y mantenida en varias formas y representaciones, y confeccionada para ser procesada por una computadora en su estado más desarrollado.



Evolución del Software. Un cambio en los requerimientos afecta potencialmente a, diseño y a distintas piezas de código fuente (Fuentes 3 y 4 en la imagen), y a su vez afectar a determinados objetos.

Los cambios en el software tienen su origen en cambios del negocio y nuevos requerimientos, soporte de cambios de productos asociados, reorganización de las prioridades, cambios en el presupuesto, etc.

El SCM es una actividad paraguas, transversal a todo el proyecto, relevante para el producto en todo su ciclo de vida. Su propósito es establecer y mantener la **integridad** de los productos de software a lo largo de su ciclo de vida. Para que un producto se considere íntegro, debe cumplir con **4 condiciones**:

- Satisface las necesidades del usuario
- Puede ser fácil y rastreable durante su ciclo de vida
- Satisface criterios de performance

- Cumple con expectativas de costo

Involucra para la configuración:

- Identificarla en un momento dado
- Controlar sistemáticamente cambios
- Mantener integridad y origen

Suelen ocurrir distintos problemas en el manejo de los componentes, entre los cuales se destacan:

- Regresión de fallas
- Pérdida de un componente
- Doble mantenimiento
- Superposición de cambios
- Pérdida de cambios

Conceptos Clave

- **Ítem de Configuración IC.** Artefacto que forma parte del producto o proyecto, que puede sufrir cambios o necesita ser compartido entre miembros del equipo y sobre los cuales se necesita saber estado y evolución. Puede ser código fuente, código de prueba, requerimientos en un word, etc.
- **Configuración de Software.** Conjunto de IC con su correspondiente versión en un momento determinado.
- **Versión.** Forma particular de un artefacto en un instante o contexto dato
- **Control de versiones.** Evolución de un único IC. Se puede visualizar en un grafo.
- **Variante.** Versión de un IC que evoluciona por separado, representan configuraciones alternativas.
- **Línea Base.** Configuración que ha sido revisada formalmente y sobre la que se ha llegado a un acuerdo
- **Repositorio.** Contenedor de IC con sus atributos y relaciones. Es usado para hacer evaluaciones de impacto de los cambios propuestos, y puede ser una o varias bases de datos. Existen dos tipos:
 - **Centralizado.** Un servidor contiene todos los archivos con sus versiones, y los administradores tienen un mayor control sobre el repositorio.

- **Descentralizado.** Cada cliente tiene una copia del repositorio, por lo que si un servidor falla sólo es cuestión de copiar y pegar. Además, posibilita workflows no disponibles en un repositorio centralizado.

Líneas Base (Baseline)

Es una configuración que ha sido revisada formalmente y sobre la que se ha llegado a un acuerdo. Sirve como base para desarrollos posteriores y puede cambiarse sólo a través de un procedimiento formal de control de cambios. Permiten ir hacia atrás en el tiempo y reproducir el entorno de desarrollo en un momento dado del proyecto.

Las líneas base pueden ser de **especificación** (requerimientos o diseño) o de **productos** que pasaron un control de calidad definido, también llamado operacional.

Ramas

Existe una rama principal (main, master o trunk) que permiten bifurcar el desarrollo y experimentación. Pueden ser **descartadas o integradas** (merge). La integración lleva los cambios a la rama principal, lo cual puede generar conflictos. Son esencialmente **temporales**, eventualmente deberían integrarse o descartarse todas las ramas.

Actividades Fundamentales de la GCS

1. Identificación de ítems de configuración IC
2. Control de cambios
3. Auditorías de configuración
4. Informes de Estado

Identificación de IC

Consiste en la identificación única de cada ítem de configuración. Determina convenciones y reglas de nombrado, define la estructura del repositorio y la ubicación de los ítems dentro de la estructura del repositorio.



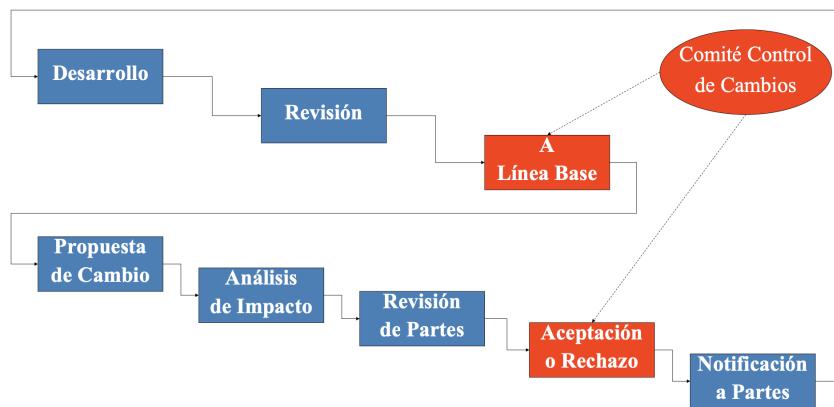
IC para un proyecto de desarrollo de software. Un producto tiene varios proyectos, los cuales tienen varias iteraciones. Por lo tanto, está ordenado por ciclo de vida: del más largo al más corto.

Control de Cambios

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.

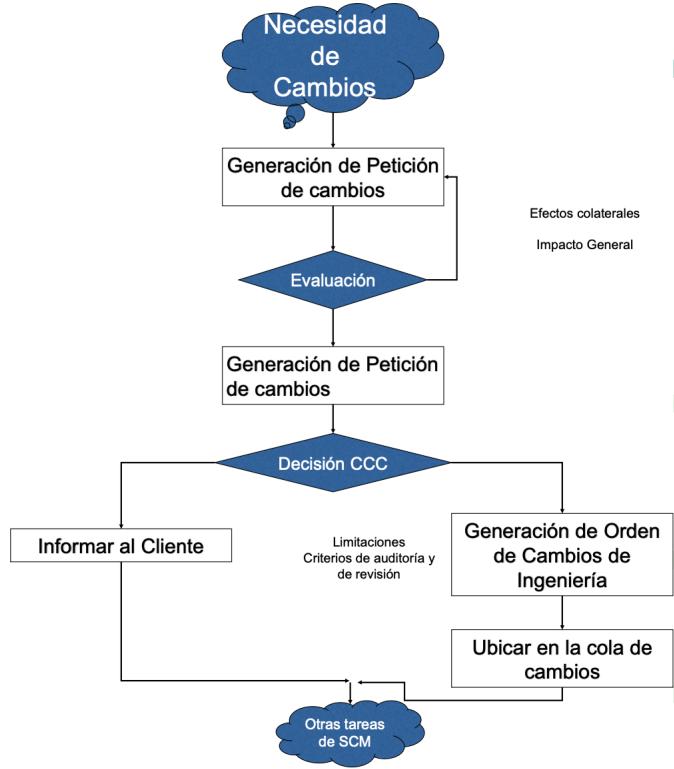
Lo realiza un *comité de control de cambios*, formado por representantes de todas las áreas involucradas en el desarrollo:

- Análisis y diseño
- Implementación
- Testing
- Otros interesados



Proceso de control de cambios en contexto.

Si un IC forma parte de la línea base, se debe realizar un proceso de control de cambios para cambiarlas.



Procedimiento para el control de cambios

Sólo hay control de cambios si existe una línea base, ya que ese estado es el que controla el comité.

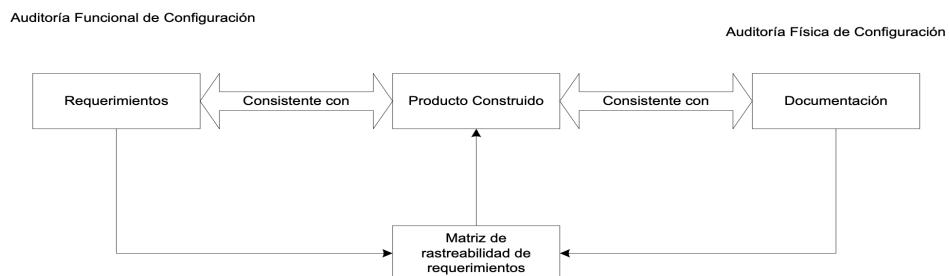
Auditorías de Configuración

El auditor debe ser externo al equipo, porque la auditoría debe ser independiente y objetiva.

Hay dos tipos:

1. **Auditoría física de configuración (PCA).** Asegura que lo que está indicado para cada IC en la línea base o en la actualización se haya alcanzado realmente. Vela por la integridad del repositorio: (que el requerimiento haya sido correctamente). Se hace primero
 - a. Que el repositorio esté donde dijimos que iba a estar
 - b. Que los IC respeten el esquema de nombrado
 - c. Que los IC estén guardados donde se dijo que iban a estar guardados
 - d. Se audita la línea base
2. **Auditoría funcional de configuración (FCA).** Evaluación independiente de los productos de software, controlando que la funcionalidad y performance reales de cada IC

sean consistentes con la especificación de requerimientos (ERS). (que haga lo que tenga que hacer). Si la física no sale bien, la funcional no se hace.



Auditoría de la gestión de la configuración. La FCA verifica que el producto sea consistente con el producto construido, y la PCA verifica que sea consistente con la documentación.

La matriz de trazabilidad de requerimientos vincula los requerimientos con los IC que se generaron después y que generaron a los requerimientos (pre-ers pos-ers).

Las auditorías implican que los proyectos necesitan un plan. Si no hay plan, no hay contra qué controlar. Si no hay línea base, no hay qué controlar. Se obtiene un informe de auditoría con todas las desviaciones.

La auditoría sirve a dos procesos: validación y verificación.

- **Validación.** El problema es resuelto correctamente y el usuario obtiene el producto correcto, asociada con la FCA.
- **Verificación.** El producto cumple con objetivos preestablecidos, definidos en la documentación de líneas base. Todas las funciones son llevadas a cabo con éxito y los test cases tienen status “ok” o bien consten como “problemas reportados” en la nota de release.

Informes de Estado

Se ocupa de mantener el registro de la evolución del sistema. Se suele implementar con procesos automáticos, e incluye reportes de trazabilidad de todos los cambios realizados a las líneas base durante el ciclo de vida. Puede responder preguntas como:

- ¿Cuál es el estado del ítem?
- ¿El requerimiento “x” fue aprobado o rechazado por el comité?
- ¿Diferencia entre una versión y otra?

Plan de Gestión de Configuración

Debe contener información relacionada a las 4 actividades de la GCS:

- Reglas de nombrado de IC
- Herramientas a utilizar
- Roles e integrantes del comité
- Procedimiento formal de cambios
- Plantillas de formularios
- Procesos de auditoría

Gestión de Configuración de Software Ágil

- Sirve al equipo de desarrollo y no viceversa
- Hace seguimiento y coordina el desarrollo en lugar de controlar al equipo de desarrollo
- Responde a los cambios en vez de evitarlos
- Esforzarse por ser transparente y “sin fricción”, automatizando
- Coordinación y automatización frecuente y rápida
- Eliminar desperdicio
- Documentación Lean y trazabilidad
- Feedback continuo sobre calidad, estabilidad e integridad