Universidad Católica del Uruguay

Modelo de Procesos

Contenido

1 INTRODUCCIÓN 4

2 CICLO DE VIDA DEL SOFTWARE 4

2.1 VENTAJAS DE SELECCIONAR UN CICLO DE VIDA 4

2.2 ACCIONES IMPLÍCITAS EN UN CICLO DE VIDA 4

2.3 DIFERENCIAS ENTRE DISTINTOS MODELOS DE CICLO DE VIDA 4

2.4 MODELO LINEAL 4

2.4.1 Ciclo de vida clásico o en cascada 4

2.4.2 ¿Cuándo aplicarlo? 5

2.5 BRECHA DE EXPECTATIVAS 5

2.5.1 El problema 5

2.5.2 Solución 5

2.6 MODELO ITERATIVO 6

2.6.1 Ciclo de vida prototipo 6

2.6.1.1 Descartable 7

2.6.1.2 Maqueta 7

2.6.1.3 Progresivo 7

2.6.2 ¿Cuándo aplicarlo? 8

2.6.3 Ciclo DRA 8

2.6.4 ¿Cuándo aplicarlo? 9

2.7 MODELO EVOLUTIVO 9

2.7.1 Ciclo de vida incremental 9

2.7.2 ¿Cuándo aplicarlo? 10

2.7.3 Basado en componentes 10

2.7.3.1 Beneficios 10

2.7.4 ¿Cuándo aplicarlo? 11

2.7.5 Ciclo de vida en espiral 11

2.7.6 ¿Cuándo aplicarlo? 12

2.8 METODOLOGÍAS ÁGILES 13

2.8.1 Scrum 13

2.8.1.1 Proceso de Scrum 14

2.8.1.2 Roles de Scrum 15

2.8.1.3 Reuniones de Scrum 17

2.8.2 Artefactos de Scrum 22

2.8.2.1 Product Backlog 22

2.8.2.2 Ítems del Producto Backlog (PBIs) 22

2.8.2.3 Backlog Sprint 22

2.8.2.4 Tareas del Sprint 23

2.8.2.5 Sprint Burndown Chart 23

2.8.2.6 Product / Release burndown chart 24

2.8.3 Escalamiento *24*
2.8.3.1 Equipos y características 25
2.8.3.2 Equipos auto-organizados 25
2.8.4 ¿Cuándo es apropiado Scrum? *26*
3 RESUMEN 27
4 BIBLIOGRAFÍA 28

1 Introducción

Para resolver los problemas reales en la industria, un ingeniero del software debe incorporar una estrategia de desarrollo que acompañe al proceso, métodos y herramientas a utilizar en el proyecto.

Por su naturaleza, un **modelo de procesos del software** es una simplificación o abstracción de un proceso real.

Lo primero que ha de realizarse es la selección del *ciclo de vida* que mejor resuelva las particularidades del proyecto. Éste habrá de abarcar todas las etapas, desde la concepción inicial del producto pasando por su desarrollo, puesta en marcha, mantenimiento, y eventual retiro.

2 Ciclo de vida del software

La ISO 12207 define Ciclo de vida como un *"marco de referencia que contiene los procesos, actividades y tareas involucradas en el desarrollo, explotación y mantenimiento de un producto de Software."*

2.1 Ventajas de seleccionar un ciclo de vida

Seleccionar un ciclo de vida nos permite:

a. Mantener la coherencia entre todos los proyectos de la organización.
b. Definir las actividades/tareas incluidas en cada proceso.
c. Establecer puntos de control para realizar las revisiones o controles de calidad.

Su elección dependerá de las características del proyecto, e influirá en el éxito de éste.

2.2 Acciones implícitas en un ciclo de vida

- Establecer las fases principales de desarrollo de software.
- Indicar el orden en que se ejecutan las fases.
- Determinar los criterios de transición asociados entre las etapas.
- Facilitar la a administrar del progreso del desarrollo.
- Proveer un espacio de trabajo para la definición de un proceso detallado de desarrollo de software.

2.3 Diferencias entre distintos modelos de ciclo de vida

- El alcance del ciclo dependiendo de hasta dónde llegue el proyecto. Un proyecto puede comprender un simple estudio de viabilidad del desarrollo de un producto, o su desarrollo completo (fabricación y modificaciones posteriores hasta su retirada del mercado).
- Las características (contenidos) de las fases en que se divide el ciclo. Esto puede depender del proyecto en sí mismo, o de la organización.
- La estructura y la sucesión de las etapas, si hay realimentación entre ellas, y si tenemos libertad de repetirlas (iterar).

2.4 Modelo lineal

2.4.1 Ciclo de vida clásico o en cascada

Es un enfoque metodológico que ordena rigurosamente las etapas del ciclo de vida del software, de forma que el inicio de cada una debe esperar a la finalización de la inmediatamente anterior.

El modelo en cascada es un proceso de desarrollo secuencial, en el que el desarrollo va fluyendo hacia abajo (como una cascada) sobre las fases que componen el ciclo de vida.

La primera descripción formal de este modelo fue realizada en 1970 por Winston W. Royce, y es el paradigma más antiguo de Ingeniería de software. En el modelo original, existían las siguientes fases:

1. Especificación de requisitos
2. Diseño

3. Construcción (Implementación o codificación)
4. Pruebas
5. Mantenimiento

Para seguir el modelo en cascada, se avanza de una fase a la siguiente en una forma puramente secuencial.

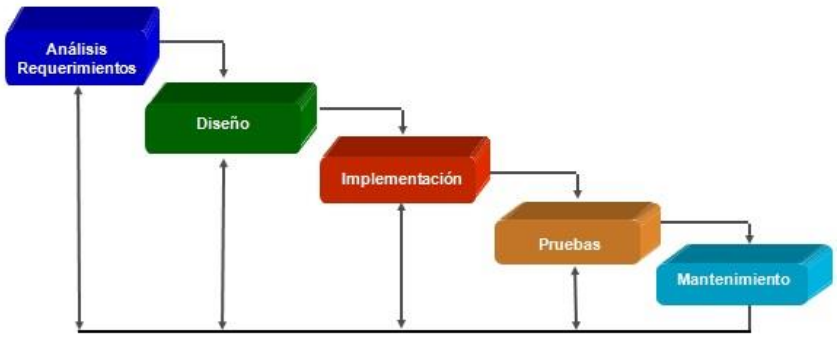


Figura 1 Ciclo de vida en cascada

Si bien ha sido ampliamente criticado desde el ámbito académico y en la industria, sigue siendo el paradigma seguido en algunos proyectos en la actualidad.

2.4.2 ¿Cuándo aplicarlo?

1. Sólo en aquellos casos en que los requerimientos se comprenden bien y sea poco probable que los mismos cambien durante el desarrollo del sistema.
2. En caso que el sistema sea de complejidad baja.
3. Se tenga total disponibilidad de los recursos.

2.5 Brecha de expectativas

2.5.1 El problema

Aplicando el modelo de proceso descrito en el numeral anterior nos encontramos con que el cliente recién verá una versión del sistema cuando el proyecto esté muy avanzado. Debido a ello, es altamente probable que se genere una gran **brecha en las expectativas** entre lo que realmente desea el usuario y lo que los desarrolladores construyen.

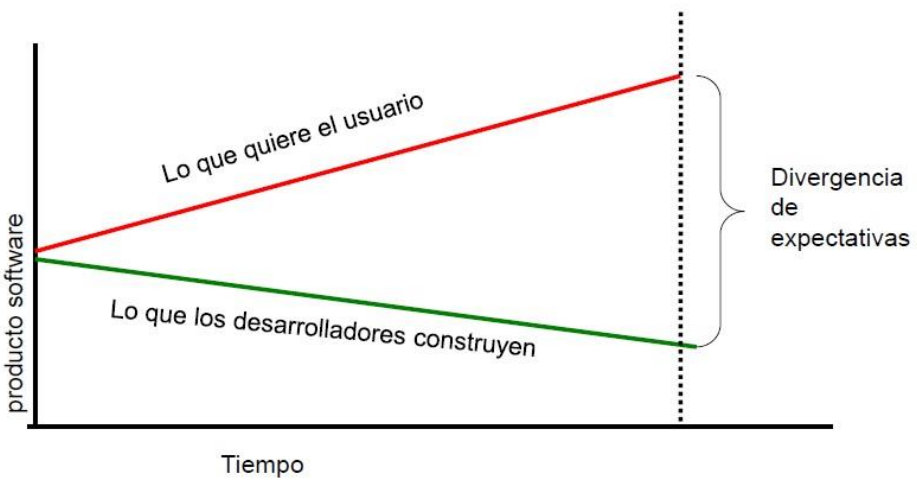


Figura 2 Brechas de expectativas para el modelo lineal

2.5.2 Solución

Para solucionar esto es necesario aplicar otros modelos de procesos, como ser el iterativo y el evolutivo, que consisten en realizar entregas parciales del producto (ver figura 3) para que el usuario las evalúe, pudiendo

aprobar o requerir cambios. Para este último caso, las modificaciones solicitadas se implementan en la próxima entrega (release). Esta estrategia se repite hasta que se llega a la última instancia en el cual, de existir diferencias, serían ser mínimas.

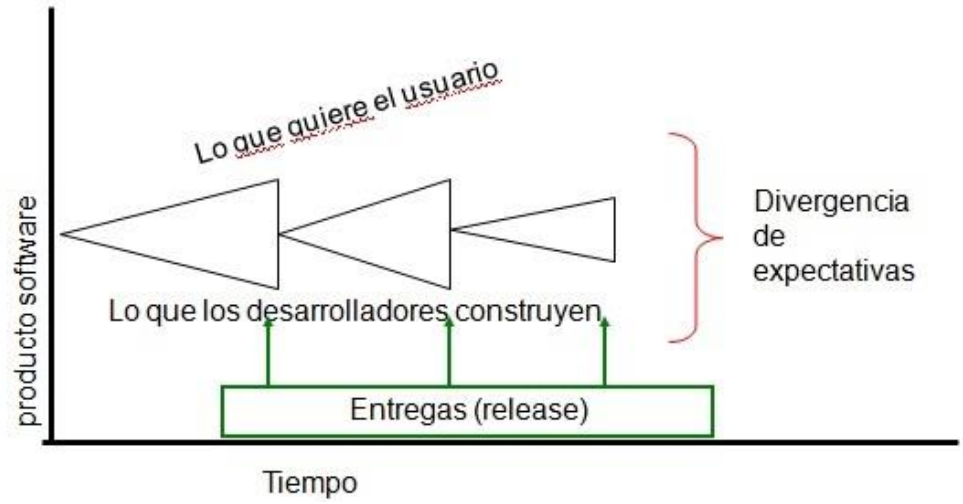


Figura 3 Solución a la brecha de expectativas

2.6 Modelo iterativo

El **modelo iterativo** se *compone de dos ciclos de vida*: el prototipo y el DRA (Desarrollo rápido de aplicaciones)

2.6.1 Ciclo de vida prototipo

Es posible aplicar este ciclo de vida cuando:

1. El cliente define un conjunto de objetivos, pero *no puede identificarlos requisitos detallados de entrada, procesamiento o salida*.
2. No se está seguro de la eficiencia o de la viabilidad del sistema.
3. Se demuestra que el sistema es viable y que la solución satisface los requerimientos del cliente, pero a pesar de ello se decide no solicitar el desarrollo del mismo.

En el paradigma de construcción de prototipos se inicia con la comunicación con el cliente, a partir de la cual el ingeniero construye una iteración del prototipo. El diseño habrá de centrarse en la representación de aquellos aspectos del software que serán visibles para el cliente, por ejemplo, la configuración de la interfaz con el usuario.

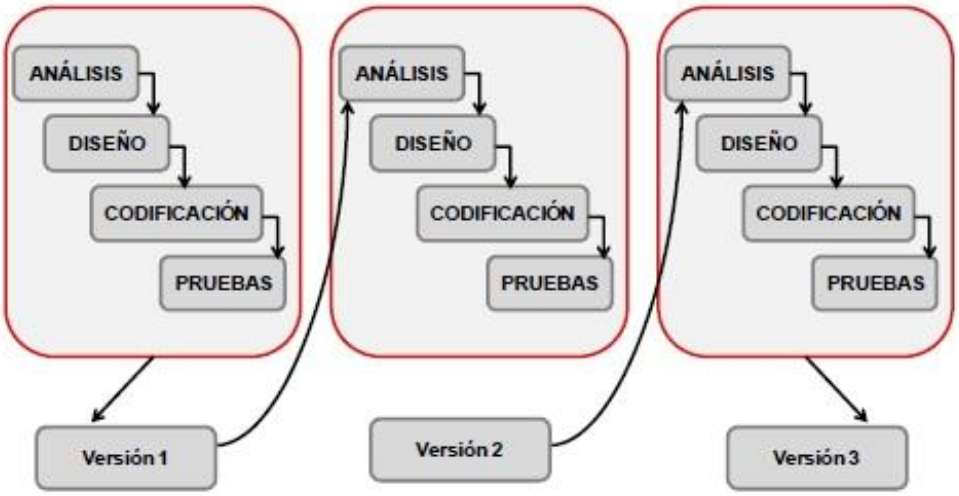


Figura 4. Modelo de ciclo de vida prototipo (genérico)

Este modelo se suele utilizar en proyectos en los que los requisitos no están claros por parte del usuario, por lo que se hace necesaria la creación de distintos prototipos para que el cliente pueda visualizar las funciones y así conseguir su conformidad. Existen 3 tipos de prototipos:

- Descartable
- Maqueta
- Progresivo

2.6.1.1 Descartable

Este prototipo en particular debe implementarse de forma rápida y debe tener un costo de desarrollo bajo debido a que una vez finalizado *debe descartarse*. Lo que se busca es demostrar la viabilidad de un sistema, por lo que se suele implementar de forma incompleta y desprolija, es decir, sin tener en cuenta los métodos formales de construcción que se proponen en Ingeniería de software.

Se *utilizan datos reales,* ya que el usuario va a interactuar con el prototipo.

Si no se descarta éste tipo de prototipo, se podrán generar grandes problemas de mantenimiento.

2.6.1.2 Maqueta

Una maqueta o prototipo de pantallas muestra la interfaz de la aplicación, su cara externa, pero deforma fija, estática sin procesar datos. En este prototipo no se desarrolla su lógica interna, sino que sólo muestra la sucesión de las pantallas, de modo que den una idea de cómo sería el diálogo de la aplicación. Se utilizan *datos estáticos*.

Se debe tener presente que la herramienta con la cual se hayan desarrollado las pantallas deben presentar una interfaz similar a las que se usen para desarrollar el sistema final, en caso que el cliente acepte el proyecto.

2.6.1.3 Progresivo

A diferencia de los dos prototipos anteriores, este va a evolucionar hasta convertirse en el sistema final. Por ello, su construcción debe seguir con todas las reglas que establece la disciplina de Ingeniería de software, evitando así los problemas de mantenimiento. Se utilizan *datos reales.*

En la siguiente figura se detallan todas las etapas que deben ejecutarse cuando se elige el ciclo de vida prototipo progresivo:

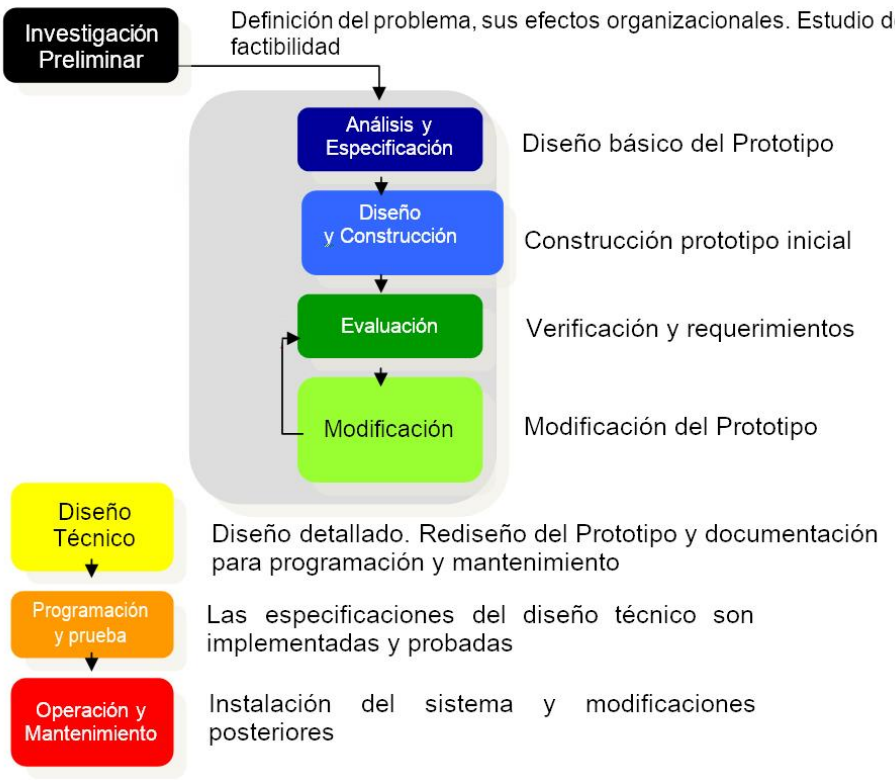


Figura 5 - Ciclo de vida Prototipo progresivo

2.6.2 ¿Cuándo aplicarlo?

1. Cuando el usuario no analizará modelos abstractos.
2. No se está seguro de la viabilidad de la solución (descartable y maqueta).
3. Sólo se puede obtener los requerimientos mediante un proceso de ensayo-error.
4. El sistema se implementará en línea (on-line).
5. Tiene una complejidad media.

2.6.3 Ciclo DRA

El Desarrollo Rápido de Aplicaciones (DRA[1]) es un ciclo de vida de software que requiere un período de desarrollo corto (30 – 90 días). Para poder aplicarlo es necesario contar con un enfoque de construcción basado en componentes y que se entiendan bien los requisitos.

Las funciones deben ser desarrolladas por equipos separados que trabajan en paralelo para finalmente integrarse en un producto.

[1] RAD: Rapid Aplication Development

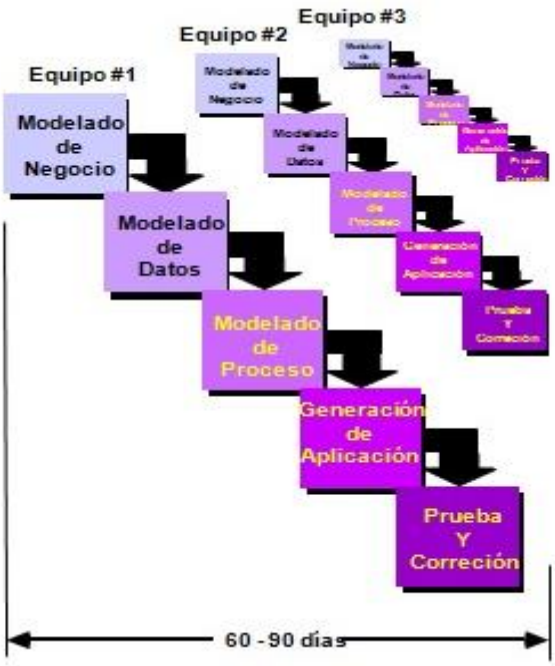


Figura 6 –Ciclo de vida DRA

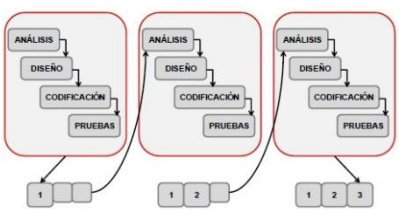
El DRA permite que los equipos de desarrollo creen un "sistema completamente funcional".

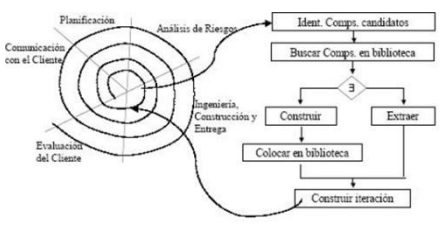
2.6.4 ¿Cuándo aplicarlo?

1. Para proyectos grandes, si se tiene suficientes recursos humanos para crear el correcto número de equipos.
2. Si los desarrolladores y clientes se comprometen con las actividades rápidas necesarias para completar el sistema en un marco de tiempo muy breve.
3. Los riesgos técnicos son bajos, por ejemplo cuando una aplicación no requiere fuertemente del uso de tecnologías nuevas.
4. Si el sistema se puede descomponer de manera tal que pueda ser desarrollado por equipos que trabajen separados en paralelo.

2.7 Modelo evolutivo

El **modelo evolutivo** lo componen *tres ciclos de vida*: el incremental, Basado en componentes y el Espiral.





Incremental *Basado en componentes* *Espiral*

2.7.1 Ciclo de vida incremental

Este modelo, a diferencia de los ciclos de vida prototipo, se centra en la entrega de un *producto operativo* con cada incremento. Los primeros incrementos son versiones incompletas del producto final, pero proporcionan al usuario la funcionalidad que precisa y también una plataforma para la evolución.

Un buen ejemplo de esto son los paquetes de Microsoft, que van incrementando sus funcionalidades dado cierto período de tiempo.

Generalmente, el primer incremento es un producto esencial compuesto por los requisitos básicos.

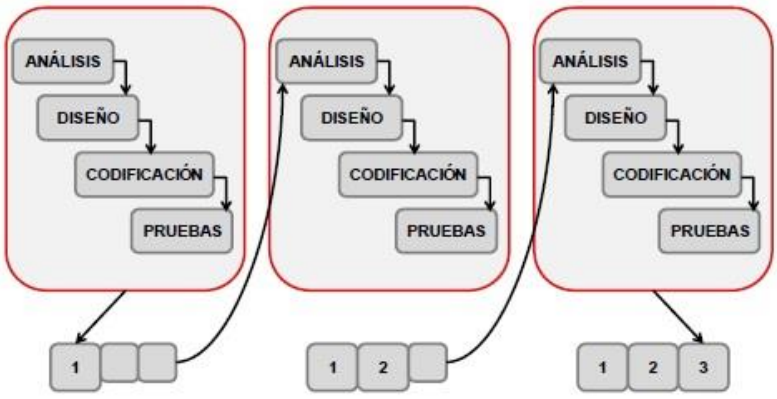


Figura 7 – Modelo de ciclo de vida incremental

2.7.2 ¿Cuándo aplicarlo?

1. El personal que disponemos no es suficiente para una implementación completa.
2. La complejidad del sistema es media
3. Se pueda financiar el proyecto por partes

2.7.3 Basado en componentes

Los sistemas de hoy en día son cada vez más complejos, deben ser construidos en tiempo récord y deben cumplir con los estándares más altos de calidad. Para hacer frente a este escenario, se concibió y perfeccionó lo que hoy conocemos como **Ingeniería de Software Basada en Componentes** (ISBC), que únicamente puede aplicarse si poseemos o podemos desarrollar/adquirir componentes de software[2], y cuyo desarrollo es en espiral.

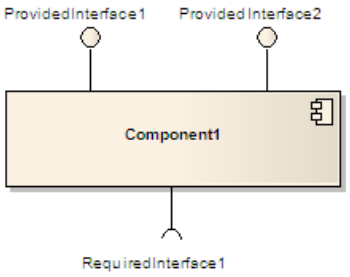


Figura 8 – Ejemplo de componente

Los componentes se juntan y combinan para llevar a cabo determinadas tareas. Posee documentación apropiada y se basa en su re-uso.

2.7.3.1 Beneficios

El uso de este paradigma posee algunas ventajas:

- **Reutilización del software:** nos lleva a alcanzar un mayor nivel de reutilización de software.
- **Simplifica las pruebas:** permite que las pruebas sean ejecutadas probando cada uno de los componentes antes de probar el conjunto completo de componentes ensamblados.

[2] **Componente de software:** pieza de código pre-elaborado que encapsula funcionalidades expuestas a través de interfaces definidas.

- **Simplifica el mantenimiento del sistema:** cuando existe un débil acoplamiento entre componentes, el desarrollador es libre de actualizar y/o agregar componentes según sea necesario, sin afectar otras partes del sistema.
- **Mayor calidad:** dado que un componente puede ser construido y luego mejorado continuamente por un experto u organización, la calidad de una aplicación basada en componentes mejorará con el paso del tiempo.

De la misma manera, el optar por comprar componentes de terceros en lugar de desarrollarlos permite:

- **Ciclos de desarrollo más cortos:** la adición de una pieza dada de funcionalidad tomará días en lugar de meses o años.
- Puede ser más favorable que desarrollar los componentes uno mismo.
- **Funcionalidad mejorada:** para usar un componente que contenga una pieza de funcionalidad, solo se necesita entender su naturaleza, y no sus detalles internos.
- Son piezas ya probadas en otros sistemas en las que podemos confiar. Solo debemos ocuparnos de su correcta integración.

2.7.4 ¿Cuándo aplicarlo?

1. Cuando podemos desarrollar el sistema utilizando un modelo en espiral y tecnología de objetos.
2. Tenemos un número significativo de componentes reutilizables.

2.7.5 Ciclo de vida en espiral

Este modelo fue desarrollado por Barry Boehm en 1985 y es utilizado de forma generalizada en la ingeniería del software. En él, las actividades se conforman en una espiral, ordenadas por grupos en cada bucle. El proceso empieza en la posición central y desde allí se mueve en el sentido de las agujas del reloj.

Figura 9 - Ciclo de vida en espiral simplificado

Las actividades no se encuentran fijadas a priori, sino que las siguientes se eligen en función del análisis de riesgos, comenzando por el bucle anterior. Se dice que uno de los aspectos fundamentales del éxito de este modelo radica en que el equipo que lo gestione tenga la necesaria experiencia y habilidad para detectar y catalogar correctamente riesgos, y realizar el seguimiento de los costos.

Cada iteración interna se crea una versión de prototipo (ver P1, P2 en la figura 9), hasta convertirse en un prototipo operacional.

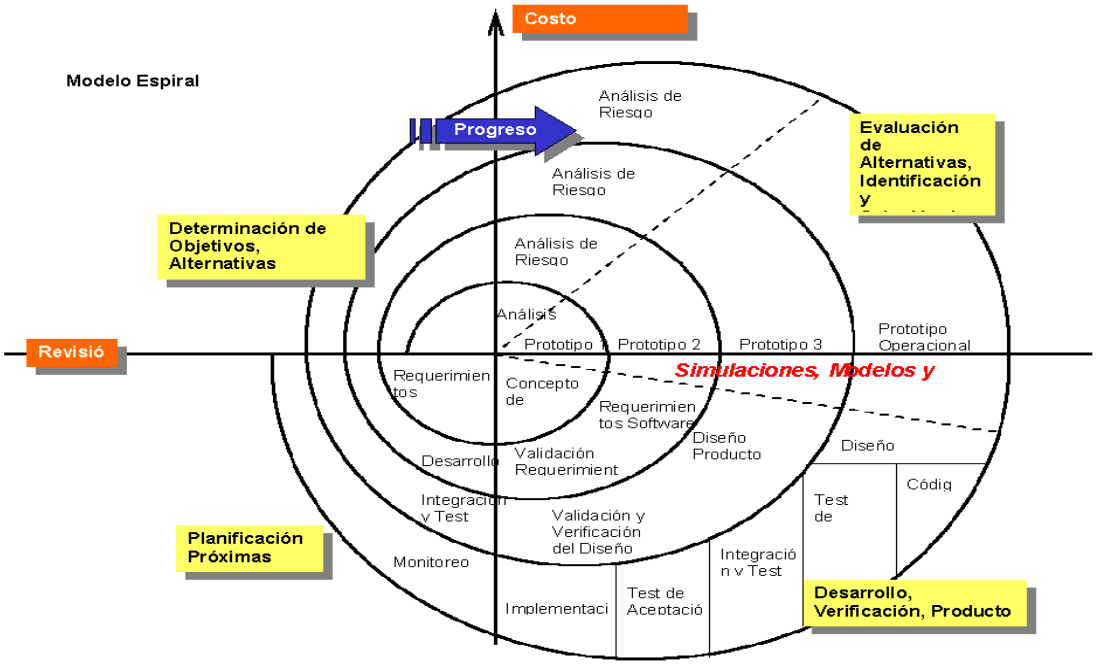


Figura 10 – Ciclo de vida en espiral completo

Para cada ciclo habrá cuatro actividades:

1. **Determinar o fijar objetivos**
    a. Fijar también los productos definidos a obtener: requerimientos, especificación, manual de usuario.
    b. Fijar las restricciones.
    c. Identificar riesgos del proyecto y estrategias alternativas para evitarlos.
    d. Hay una cosa que solo se hace una vez: planificación inicial o previa.
2. **Análisis del riesgo/ Costo**
    a. Estudiar todos los riesgos potenciales y se seleccionan una o varias alternativas propuestas para gestionar el riesgo.
    b. Control de gastos (costos).
3. **Desarrollar, verificar y validar (probar):**
    a. Tarea de la actividad propia y de prueba
    b. Análisis de alternativas e identificación de resolución de riesgo
4. **Planificar**
    a. Revisar todo lo que se ha llevado a cabo, evaluándolo y decidiendo si se continúa con las fases siguientes y planificando la próxima actividad.

2.7.6 ¿Cuándo aplicarlo?

1. Proyectos grandes y complejos.
2. El análisis del riesgo se realiza en forma explícita y clara.
    a. Reduce riesgos del proyecto
    b. Incorpora objetivos de calidad
3. Se controla el aumento del costo del proyecto.

2.8 Metodologías ágiles

2.8.1 Scrum

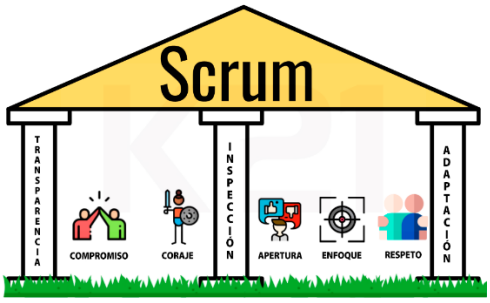
Es un marco de trabajo por el cual las personas pueden abordar problemas complejos y a su vez entregar productos de máximo valor para el cliente.

Scrum es:

- Liviano
- Fácil de entender
- Difícil de llegar a dominar

El mayor beneficio de Scrum se experimenta en el trabajo complejo que implica la creación de conocimiento y colaboración, tal como sucede en el desarrollo de nuevos productos.

Esta metodología se basa en 3 pilares



- Transparencia

Los aspectos significativos del proceso deben ser visibles para aquellos que son responsables del resultado. La transparencia requiere que dichos aspectos sean definidos por un estándar común, de tal modo que los observadores compartan un entendimiento del problema

Por ejemplo:

- Todos los participantes deben compartir un lenguaje común para referirse al proceso; y,
- Aquellos que desempeñan el trabajo y aquellos que aceptan el producto de dicho trabajo

  deben compartir una definición común de “Terminado”.

- Inspección

Los usuarios de Scrum deben inspeccionar frecuentemente los artefactos de Scrum y el progreso hacia un objetivo para detectar variaciones indeseadas. Su inspección no debe ser tan frecuente como para que interfiera en el trabajo. Las inspecciones son más beneficiosas cuando se realizan de forma diligente por inspectores expertos de las correspondientes funcionalidades

- Adaptación

Si un inspector determina que uno o más aspectos de un proceso se desvían de límites aceptables y que el producto resultante será inaceptable, el proceso o el material que se está desarrollando deben ajustarse. Dicho ajuste debe realizarse cuanto antes para minimizar desviaciones mayores.

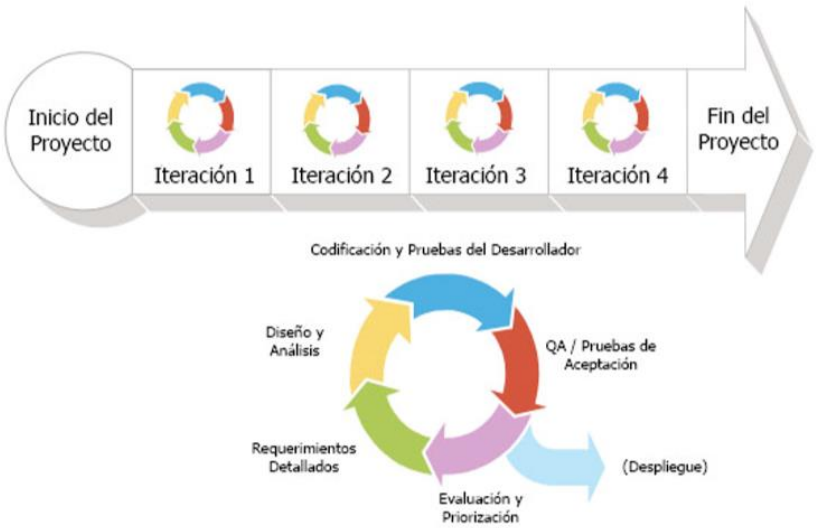
Scrum describe cuatro eventos formales, contenidos dentro del Sprint, para la inspección y adaptación, tal como se describen en la sección Eventos de Scrum del presente documento.

- Planificación del Sprint (Sprint Planning)
- Scrum Diario (Daily Scrum)
- Revisión del Sprint (Sprint Review)

- Retrospectiva del Sprint (Sprint Retrospective)

2.8.1.1 Proceso de Scrum

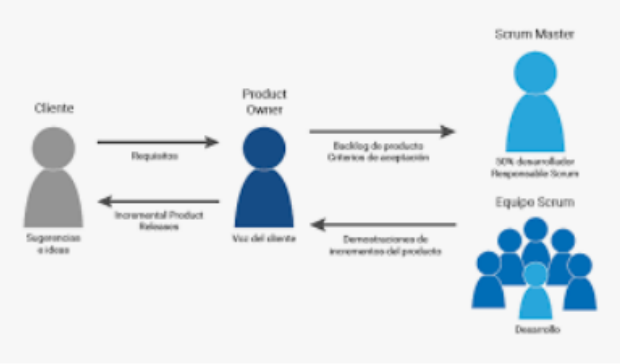
En Scrum un proyecto se ejecuta en ciclos temporales cortos y de duración fija (iteraciones que normalmente son de 2 semanas, aunque en algunos equipos son de 3 y hasta 4 semanas, límite máximo de feedback de producto real y reflexión. Cada iteración tiene que proporcionar un resultado completo, un incremento de producto final que sea susceptible de ser entregado al cliente cuando lo solicite.



El proceso parte de la lista de objetivos/requisitos priorizada del producto, que actúa como plan del proyecto. En esta lista proporcionada por el cliente, el Product Poner la prioriza en función a los objetivos balanceando el valor que le aportan respecto a su coste.

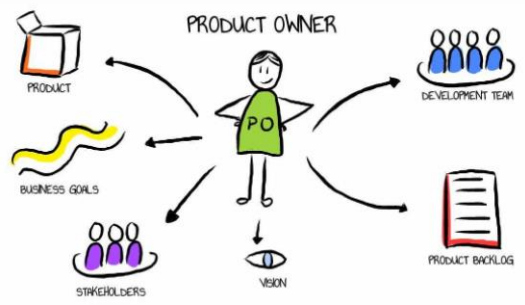
Las actividades que se llevan a cabo en Scrum son las siguientes:

2.8.1.2 Roles de Scrum



Product Owner

El Product Owner es el miembro del equipo Scrum responsable de maximizar el valor del producto entregado por el equipo. El objetivo del Product Owner es lograr que entreguemos el producto "correcto", el producto que quiere el mercado y los stakeholders



Características/tareas

- (única persona) responsable de maximizar el retorno de la inversión (ROI) del esfuerzo de desarrollo
- Responsable de la visión del producto
- Constantemente re-prioriza el Backlog del Producto, ajustando las expectativas a largo plazo, como los planes de liberaciones
- Es el árbitro final de las preguntas sobre requerimiento
- Acepta o rechaza cada incremento del producto
- Decide si se debe liberar
- Decide si se debe continuar con el desarrollo
- Considera los intereses de los stakeholders
- Tiene un papel de liderazgo

Scrum Master

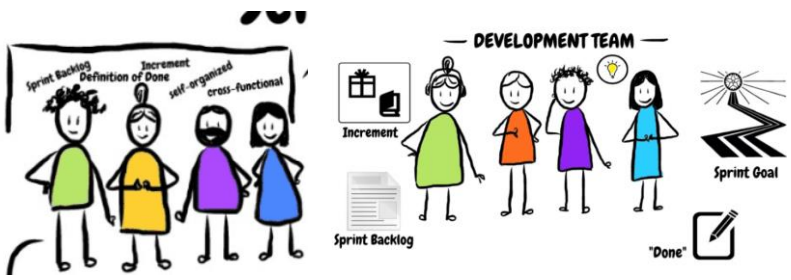
El Scrum Master tiene dos funciones principales dentro del marco de trabajo: gestionar el proceso Scrum. Además, se encarga de las labores de mentoring y formación, coaching y de facilitar reuniones y eventos.

Características/tareas

- Facilita el proceso de Scrum
- Ayuda a resolver los impedimentos
- Crea un ambiente propicio para la auto-organización del equipo
- Captura datos empíricos para ajustar las previsiones
- Protege al equipo de interferencias externas y distracciones para mantener el flujo del equipo (también conocido como la zona)
- Aplica los time boxes
- Mantiene visibles los artefactos Scrum
- Promueve la mejora de las prácticas de ingeniería
- No tiene autoridad en la gestión del equipo
- Tiene un papel de liderazgo

Equipo de Desarrollo de Scrum

Cuando se habla específicamente de «equipo de desarrollo» se refiere al conjunto de personas más «técnicas» que de manera conjunta desarrollan el producto del proyecto. Tienen un objetivo común, comparten la responsabilidad del trabajo que realizan (así como de su calidad) en cada iteración y en el proyecto.



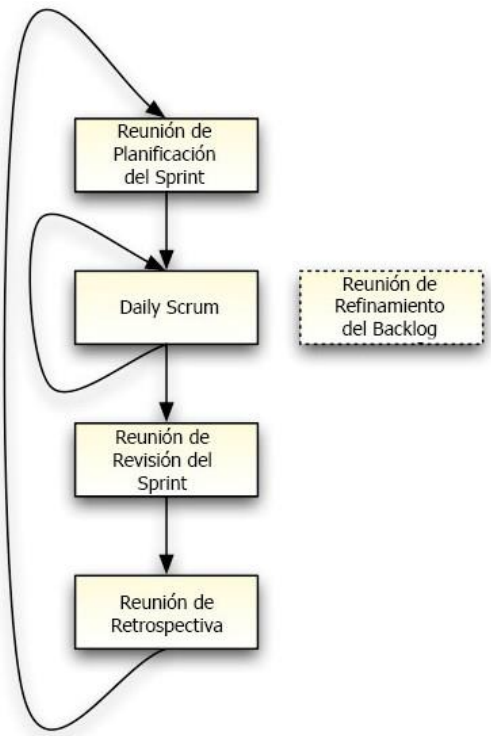
Características/tareas

- Multifuncional (incluye miembros con habilidades de testing y a menudo otros no llamados tradicionalmente desarrolladores: analistas de negocio, expertos de dominio, etc.)
- Auto-organizado/auto-gestionado, sin roles asignados externamente

- Negocia los compromisos con el Product Owner, de un Sprint a la vez
- Tiene autonomía con respecto a la forma de lograr sus compromisos
- Intensamente colaborativo
- Tiene mayor probabilidad de éxito al encontrarse establecido en unmismo lugar, sobre todo para los primeros Sprints
- Tiene más éxito si el involucramiento con el equipo es a largo plazo y full-time. Scrum promueve evitar el traslado de personas o dividirlasentre otros equipos.
- 7 ± 2 miembros
- Tiene un papel de liderazgo

2.8.1.3 Reuniones de Scrum

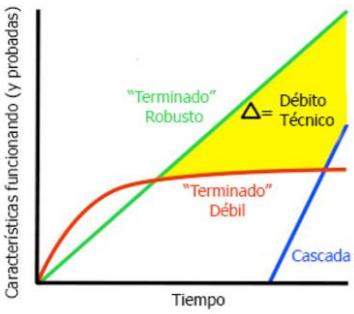
Todas las reuniones de Scrum son facilitadas por el ScrumMaster, quien no tiene autoridad para tomar decisiones en ellas.



2.8.1.3.1 Reunión de Planificación de Sprint

Al comienzo de cada Sprint, el Product Owner y el equipo tienen una Reunión de Planificación del Sprint donde negocian qué ítems del Backlog del Producto intentarán convertir en producto funcionando durante el Sprint.

El Product Owner es el responsable de declarar cuáles son los ítems más importantes para el negocio. El equipo es responsable de seleccionar la cantidad de trabajo que cree que podránrealizar *sin acumular deuda técnica*. El equipo "toma" el trabajo desde el Product Backlog hacia el Sprint Backlog.



Cuando los equipos se encuentran frente a un trabajo complejo con una incertidumbre inherente, deben trabajar juntos para intuitivamente estimar su capacidad para comprometerse con los ítems, mientras aprenden de los sprints anteriores. Planificar su capacidad por hora y comparar sus estimaciones con datos reales hace que el equipo estime con precisión sus compromisos.

Hasta que un equipo haya aprendido a lograr un incremento del producto potencialmente entregable en cada Sprint, se debe reducir lacantidad de funcionalidad comprometida. El fracaso en cambiar hábitos viejos (adquiridas con las metodologías tradicionales) conduce a la deuda técnica y la muerte eventual del diseño.

Si la parte superior Backlog del Producto no ha sido refinada, una parte importante de la reunión de planificación se debe dedicar a esto,tal como se describe en la sección de la Reunión de Refinamiento del Backlog.

Hacia el final de la Reunión de Planificación del Sprint, el equipo desglosa los ítems seleccionados en una lista *inicial* de tareas del Sprint(*Sprint Tasks*) y hace un compromiso final para realizar el trabajo.

El tiempo asignado a un *timebox* es de 2 a 4 semanas.

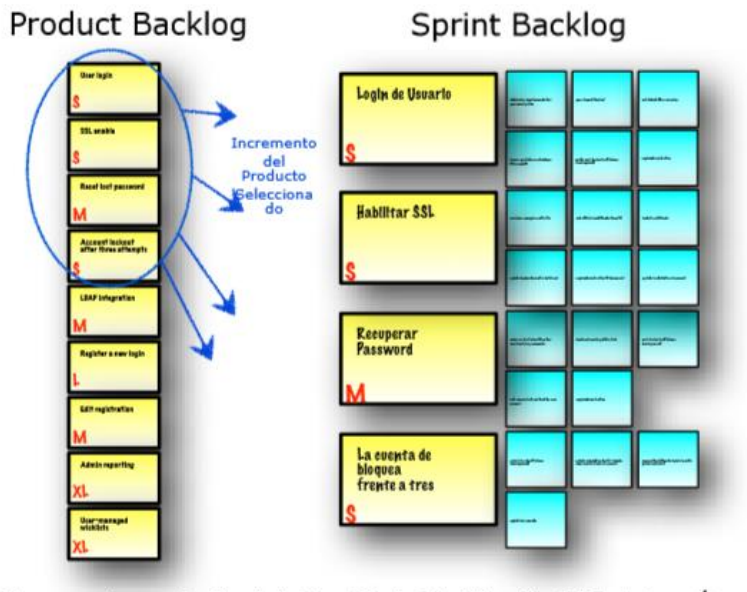


Figura 4: *los resultados de la Reunión de Planificación del Sprint son Ítems del Product Backlog (PBIs) comprometidos y Sprint Tasks subordinadas.*

2.8.1.3.2 Daily Scrum y ejecución del Sprint

Cada día, a la misma hora y en el mismo lugar, los miembros del equipo de desarrollo pasan 15 minutos reportándose entre sí. Cada miembro del equipo resume lo que hizo el día anterior, lo que hará hoy, y qué impedimentos está enfrentando.

Mantenerse de pie en el Daily Scrum ayudará a que la reunión sea breve. Los temas que requieren atención adicional pueden ser discutidos por los interesados después de que cada miembro del equipo lo haya reportado.

Al equipo puede resultarle útil mantener una lista de Tareas del Sprint (*Sprint Task List*), un Sprint Burndown Chart, y un Listado de Impedimentos. Durante la ejecución del Sprint es común descubrir tareas adicionales necesarias para alcanzar las metas del Sprint. Los impedimentos causados por los problemas que escapan al control del equipo se consideran *impedimentos organizacionales.*

Un equipo que necesita experiencia adicional para comprender los requisitos del producto se beneficiará de la participación del Product Owner.

La Daily Scrum tiene la intención de alterar los viejos hábitos del trabajo, el cual se realizaba por separado. Los miembros deben permanecer atentos para detectar signos de la antigua estrategia.

2.8.1.3.3 Reunión de revisión del sprint

Después de la ejecución del Sprint, el equipo mantiene una reunión de revisión del Sprint para mostrar un incremento del producto al Product Owner y a los stakeholders.

La reunión debe ofrecer una demostración en vivo, no un informe.

Después de la demostración, el Product Owner revisa los compromisos contraídos en la Reunión de Planificación del Sprint y declara qué ítems considera terminados.

Por ejemplo, un ítem de software que es meramente "código completado" no se considera terminado, porque el software que no fue probado no es entregable. Los elementos incompletos son devueltos al Backlog del Producto y clasificados de acuerdo con las prioridades del Product Owner como candidatos para futuros Sprints.

El ScrumMaster ayuda al Product Owner y a los stakeholders a convertir su feedback en los nuevos Ítems del Product Backlog o Product Backlog Items (PBIs) para su priorización por el Product Owner.

A menudo, descubrir nuevo alcance supera la tasa del equipo de desarrollo. Si el Product Owner determina que el alcance recién descubierto es mayor que las expectativas originales, el nuevo alcance desplaza el viejo alcance en el Product Backlog.

La reunión de revisión de Sprint es el encuentro apropiado para los stakeholders. Es la oportunidad de revisar y adaptar el producto a medida que emerge, y de forma iterativa refinar la comprensión de cada uno de los requisitos. A veces los productos de software, son difíciles de visualizar al inicio del proyecto. Muchos clientes necesitan ser capaces de reaccionar ante una pieza de software que funcione para descubrir lo que realmente quieren. El desarrollo iterativo, un enfoque dirigido por el valor, permite la creación de productos que no podrían haber sido especificados por adelantado.

2.8.1.3.4 Reunión de Retrospectiva del Sprint

Cada Sprint finaliza con una retrospectiva. En esta reunión, el equipo reflexiona sobre su propio proceso. Inspeccionan su comportamiento y adoptan las medidas para adaptarlo en los futuros Sprints.

Una retrospectiva en profundidad requiere de un ambiente de seguridad psicológica que no se encuentra en la mayoría de las organizaciones. Sin seguridad, la discusión retrospectiva o bien evitará los problemas incómodos, o se deteriorará en la culpa y en hostilidad.

Un impedimento común para la plena transparencia en el equipo es la presencia de personas que realizan evaluaciones de desempeño.

Otro impedimento para una retrospectiva profunda es la tendencia humana a sacar conclusiones y proponer acciones con rapidez, para solucionarlo, autores en estos temas, han propuesto una serie de medidas para ralentizar este proceso: preparar el escenario, recabar datos, generar ideas, decidir qué hacer, cerrar la retrospectiva

Un tercer impedimento a la seguridad psicológica es la distribución geográfica. Equipos dispersos geográficamente por lo general no colaboran como lo requiere esta metodología.

Las retrospectivas a menudo exponen impedimentos organizacionales. Una vez que un equipo ha resuelto los obstáculos dentro de su influencia inmediata, el ScrumMaster debe trabajar para expandir esa influencia.

El ScrumMaster debe utilizar una variedad de técnicas para facilitar retrospectivas, incluyendo la redacción en silencio, líneas de tiempo, e histogramas de satisfacción. En todos los casos, el objetivo es lograr una comprensión común de múltiples perspectivas y desarrollar acciones que llevarán al equipo al siguiente nivel.

2.8.1.3.5 Reunión de Refinamiento del Backlog

La mayoría de los PBIs inicialmente deben refinarse, ya que son grandes y poco comprendidos. Los equipos han encontrado útil tomar un tiempo de la ejecución de cada Sprint para preparar el Backlog del Producto para la próxima Reunión de Planificación del Sprint. En la reunión de refinamiento del Backlog, el equipo estima la cantidad de esfuerzo que se debe invertir para completar los Ítems del Backlog del Producto y proporciona información técnica para ayudar al Product Owner a priorizarlos.

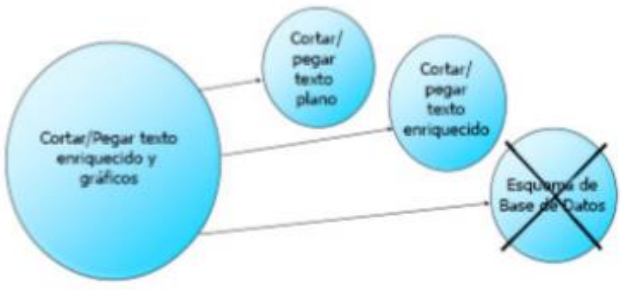
Los grandes ítems se dividen y se clarifican, teniendo en cuenta temas tanto de negocio como técnicos. A veces, un subconjunto del equipo, junto con el Product Owner y los stakeholders, descomponen y dividen los ítems del Backlog del Producto antes de involucrar a todo el equipo en la estimación.

Un ScrumMaster experimentado puede ayudar al equipo a identificar finas rebanadas verticales de trabajo que tienen valor comercial, mientras promueve una definición rigurosa de "terminado" que incluye las pruebas adecuadas y refactorización.

Es común escribir los PBIs en forma de Historias de Usuario. En este enfoque, los PBIs de gran tamaño son llamados Epics (Épica)

El desarrollo tradicional rompe las características en tareas horizontales (que se asemeja a fases en cascada) que no pueden ser priorizadas de manera independiente y que carecen de valor del negocio desde la perspectiva del cliente. Este hábito es difícil de cambiar. La agilidad requiere aprender a dividir las Historias de Usuario que representan características más pequeñas del producto.

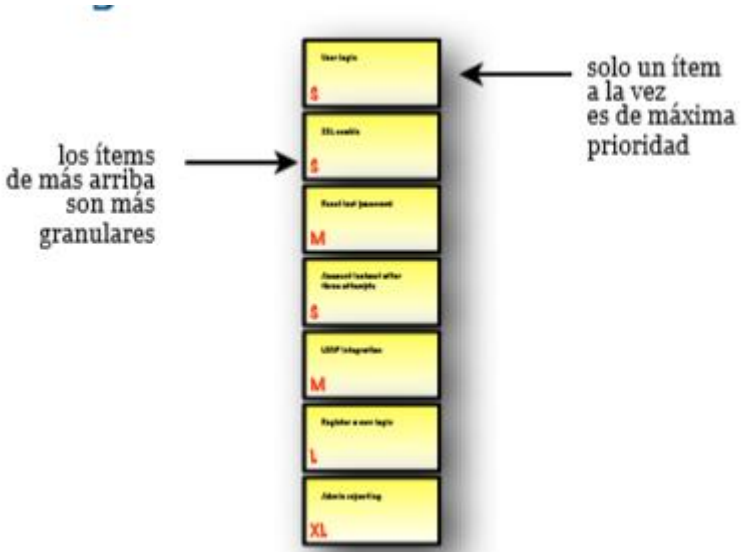
Por ejemplo, en una aplicación para registros médicos la Epic "mostrar todo el contenido de los registros de alergia de un paciente a un médico" produjo la Historia "informar si existen o no registros de alergia". Mientras que los ingenieros prevén importantes desafíos técnicos en analizar los aspectos internos de los registros de alergias, la presencia o ausencia de algún tipo de alergia era la cosa más importante que los médicos necesitaban saber. La colaboración entre los empresarios y técnicos para dividir esta Epic produjo una Historia que representa el 80% del valor de negocio con el 20% del esfuerzo de la Epic original. Como la mayoría de los clientes no utilizan la mayoría de las características de los productos, es conveniente dividir las Epics para entregar las historias más importantes primero. Aunque la entrega de características de menor valor más adelante es probable que incluya algún retrabajo, la repetición es mejor que la ausencia de trabajo. La reunión de refinamiento carece de nombre oficial y también es llamada "Backlog Grooming", "Mantenimiento del Backlog" o "Story Time".



Durante el refinamiento del Backlog, los PBIs grandes (a menudo llamados “Epics”) cerca de la parte superior del Product backlog se dividen en pequeños módulos de funciones verticales (Historias) y no fases de ejecución horizontales

2.8.2 Artefactos de Scrum

2.8.2.1 Product Backlog



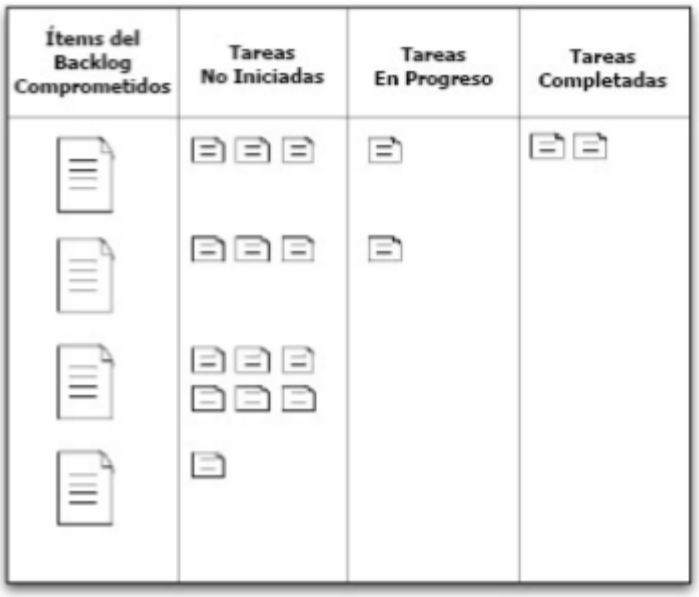
- Lista ordenada de funcionalidad deseada
- Visible para todos los stakeholders
- Cualquier stakeholder (incluido el equipo) puede agregar ítems
- Constantemente re-priorizado por el Product Owner
- Los Ítems superiores son más granulares que los inferiores
- Mantenido durante la reunión de Refinamiento del Backlog

2.8.2.2 Ítems del Producto Backlog (PBIs)

- Especifica el **qué** más que **el cómo** de una característica centrada en el cliente
- A menudo escrita en forma de *Historia de Usuario*
- Tiene una definición de *"terminado"* abarcadora de todo el producto para evitar la deuda técnica
- Puede tener criterios de aceptación específicos del ítem
- El esfuerzo es calculado por el equipo, de preferencia en unidades relativas (por ejemplo, puntos de la historia)
- El esfuerzo es de aproximadamente 2-3 personas 2-3 días, o menos para equipos más avanzados

2.8.2.3 Backlog Sprint

- Consiste en PBIs comprometidos negociados entre el equipo y el Product Owner durante la Reunión de Planificación del Sprint
- El alcance comprometido es fijo durante la ejecución del Sprint
- Las tareas iniciales son identificadas por el equipo durante la reunión de planificación del Sprint
- El equipo descubrirá las tareas adicionales necesarias para cumplir con el compromiso de alcance fijo durante la ejecución de Sprint
- Visible para el equipo
- Referenciado durante la Daily Scrum Meeting



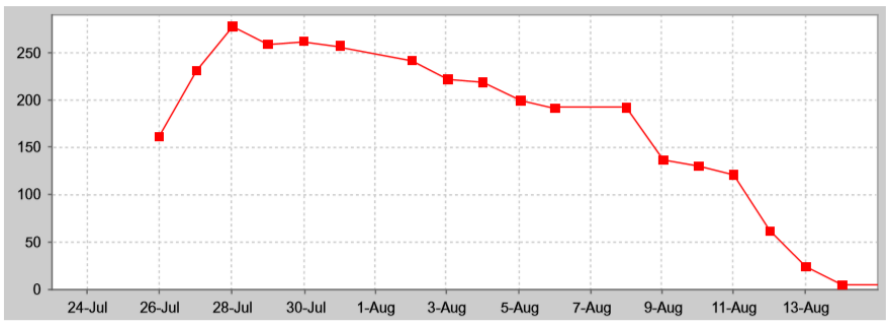
2.8.2.4 *Tareas del Sprint*

- Especifica cómo alcanzar el **qué** del PBI
- Requiere aproximadamente un día de trabajo
- El esfuerzo restante se reestima a diario, por lo general en horas
- Durante la ejecución del Sprint, una persona de contacto puede ofrecerse para ser el principal responsable de una tarea
- Propiedad de todo el equipo, se espera *colaboración*

Las tareas del Sprint necesarias para completar un Item de Backlog se componen de un mix de actividades que ya no se realizan en fases separadas (por ejemplo :captura de requisitos, análisis, diseño , implementación, desarrollo y pruebas)

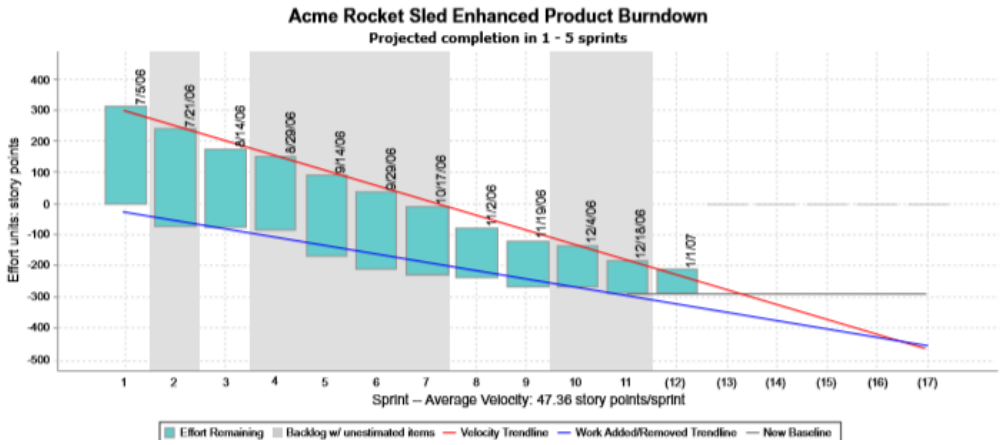
2.8.2.5 *Sprint Burndown Chart*

- Indica el total de horas restantes de las tareas del equipo dentro de un Sprint
- Es revisado a diario, por lo tanto puede subir en vez de bajar
- Diseñado para facilitar la autoorganización del equipo
- Algunas variaciones, como agrupar por persona de contacto o agregando tendencias, tiende a reducir su efectividad para fomentar la colaboración
- En la práctica a menudo ha sido mal utilizada como un informe de gestión, invitando a la intervención. El Scrum Master debe descontinuar el uso de esta tabla si se convierte en un impedimento para la autoorganización del equipo



2.8.2.6 *Product / Release burndown chart*

- Realiza un seguimiento del esfuerzo restante del Product Backlog de un Sprint al próximo
- Puede utilizar unidades relativas como *Puntos de Historia* en el eje Y
- Utiliza tendencias históricas para ajustar las previsiones

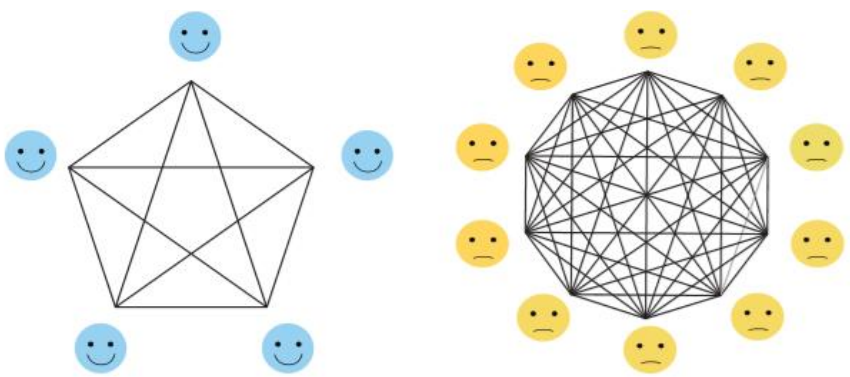


La línea roja sigue la trayectoria de los PBIs completados en el tiempo (velocidad), mientras que la línea azul sigue los nuevos PBIs añadidos (descubiertos en las reuniones). La intersección proyecta la fecha en que estará completa la entrega, basada en tendencias históricas.

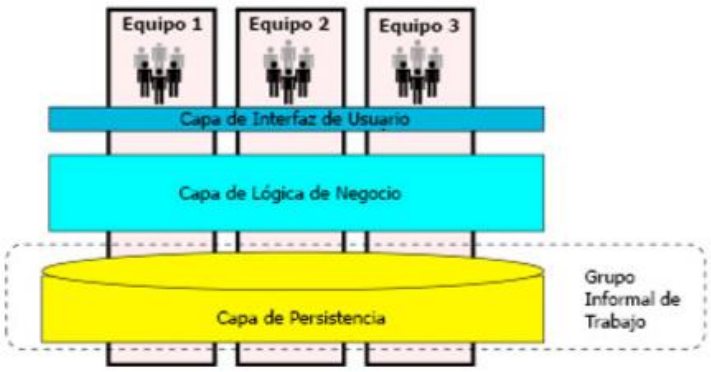
2.8.3 Escalamiento

Scrum aborda los requisitos inciertos y los riesgos técnicos mediante la agrupación de personas de múltiples disciplinas en un solo equipo (idealmente en una única sala) para maximizar el ancho de banda de la comunicación, la visibilidad y la confianza.

Cuando los requisitos son inciertos y los riesgos técnicos son altos, agregar más gente en los equipos empeora la situación. Agrupar personas por especialidad también. Agrupar a las personas por los componentes arquitectónicos (por ejemplo, equipos decomponentes) las empeora ...con el tiempo.



Vías de comunicación aumentarán como se muestra en la figura al agregar personas

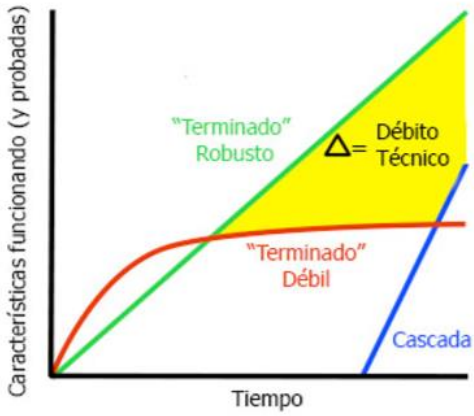


los equipos de características aprenden para abarcar los componentes arquitectónicos

La Agilidad representa un desafío particularmente para las grandes organizaciones. Creyendo hacer Scrum, muchas no lo han logrado. Los Scrum Masters en las organizaciones debían reunirse con regularidad, promoviendo la transformación a través de una lista visible de los impedimentos de organización.

2.8.3.1 Equipos y características

El enfoque más exitoso para abordar este problema ha sido la creación de "equipos de características" totalmente multi-funcionales, capaces de operar en todas las capas de la arquitectura con el fin de ofrecer las características centradas en el cliente. En un gran sistema, esto requiere aprender nuevas habilidades. A medida que los equipos se centran en el aprendizaje, en lugar de micro-eficiencias de corto plazo, pueden ayudar a crear una organización de aprendizaje.



la línea verde recta representa el objetivo general de los métodos ágiles: entrega rápida y sostenible de las funciones valiosas. Implementar Scrum correctamente implica aprender a satisfacer una definición rigurosa de "terminado" para evitar la deuda técnica.

2.8.3.2 Equipos auto-organizados

Durante la ejecución del Sprint, los miembros del equipo desarrollan un interés intrínseco en objetivos compartidos y aprenden a gestionarse para alcanzarlos.

La natural tendencia humana a rendir cuentas a un grupo de compañeros contradice años de hábito de los trabajadores. La observación del ScrumMaster y las habilidades de persuasión aumentan la probabilidad de éxito, a pesar de la incomodidad inicial.

En términos de desempeño, los equipos auto-organizados pueden superar radicalmente a los equipos tradicionales. Los grupos de tamaño familiar se auto-organizan naturalmente cuando se reúnen ciertas condiciones:

• Los miembros se comprometen a metas claras y de corto plazo

• Los miembros pueden evaluar el progreso del equipo

• Los miembros pueden observar la contribución de los otros miembros

• Los miembros se sienten seguros de dar su opinión sin tapujos

La autoorganización óptima lleva su tiempo. El equipo puede desempeñarse peor durante las primeras iteraciones de lo que han actuado como un grupo de trabajo tradicional. En el trabajo complejo, los equipos heterogéneos superan a los equipos homogéneos.

También experimentan mayores conflictos. Los desacuerdos son habituales y saludables en un equipo comprometido, el rendimiento del equipo será determinado por lo bien que el equipo se encargue de estos conflictos.

La teoría de la manzana podrida sugiere que una sola persona negativa ("que requiere esfuerzo de retención del grupo, que expresa afecto negativo, o la violación de importantes normas interpersonales") puede reducir el rendimiento de un grupo entero de manera desproporcionada.

2.8.4 ¿Cuándo es apropiado Scrum?

Scrum se aplica a los tipos de trabajo que las personas han encontrado difícil de manejar con procesos definidos, que incluyen requisitos inciertos combinados con riesgos impredecibles en la tecnología de aplicación. Al decidir si se debe aplicar Scrum, a diferencia de los enfoques impulsados por la planificación, tales como los descritos en la Guía del PMBOK®, se recomienda tener en cuenta los mecanismos subyacentes bien entendidos o si el trabajo depende de la creación de conocimiento y colaboración.

Por ejemplo, Scrum no fue pensado *originalmente para los tipos repetitivos de producción y servicios.* También es aconsejable considerar si existe un compromiso suficiente para fomentar un equipo auto-organizado.

3 Resumen

- **Modelo lineal**
  - El desarrollo es secuencial.
  - Se usa cuando:
    - Se comprenden los requerimientos y son estáticos.
    - El sistema es de baja complejidad.
    - Se tiene total disponibilidad de recursos.
- **Modelos iterativos**
  - RAD
    - Ciclo muy corto (meses).
    - Sistemas modularizables.
    - Basado en re-utilización y trabajo en paralelo.
    - Se usa si:
      - El proyecto es grande.
      - Los desarrolladores y clientes realizan actividades rápidas.
      - Riesgos técnicos bajos.
      - El sistema se puede descomponer para trabajar en paralelo.
  - Prototipos
    - El usuario no analizará modelos abstractos.
    - Requerimientos obtenidos vía ensayo-error.
    - Complejidad media.
    - Se usa cuando:
      - No se identifican requisitos detallados de entrada, procesamiento o salida.
      - No se conoce viabilidad del sistema.
    - Tipos de prototipos:
      - ***Descartable***
      -
        - Ayuda al cliente a identificar requerimientos.
        - Se implementa lo que no se entiende.
        - Costo bajo.
        - ES DESCARTABLE.
        - Se emplean datos reales.
      - ***Maqueta***
        - Requisitos no claros.
        - Alcance no definido.
        - Usuarios no colaboradores.
        - Comunicaciones complejas.
        - Se emplean datos estadísticos.
      - ***Progresivo***
        - Usuarios prueban sistema y añaden requerimientos.
        - El prototipo evoluciona y se convierte en el sistema final.
        - Se emplean datos reales.
- **Modelos evolutivos**
  - Incremental
    - Desarrollo por partes.
    - Cada iteración devuelve un incremento (versión operativa).
  - Basado en componentes
    - Basado en espiral.
    - Alta reutilización.
    - Enfoque en integración y no desarrollo.
  - Espiral

        - Se usa en proyectos complejos.
        - Requiere análisis de riesgos explícito.
        - Control de aumento de costos.
        - Posee 4 sectores:
            - Definición de objetivos.
            - Evaluación y reducción de riesgos.
            - Desarrollo y validación.
            - Planificación.
- **Metodologías ágiles**
    - **Scrum**
        - **Ventajas**
            - **Mejora la satisfacción del producto entregado al cliente**
            - **El cliente forma parte activa en el proceso**
            - **El proceso es transparente para todos**
            - **El equipo se une a un objetivo en común**
            - **Revisión temprana en software**
        - **Desventajas**
            - **Consistencia en la realización de los eventos**
            - **Complejo de implantar, implica un gran cambio cultural**
            - **El equipo puede experimentar un fuerte estrés**
            - **No siempre se cuenta con la disponibilidad del cliente como se requiere**
            - **La deuda técnica puede perjudicar los resultados**
            - **El cambio de un miembro del equipo pone en riesgo el proceso (dado los plazos tan cortos)**

4 Bibliografía

- Pressman, R.S., *Ingeniería del Software. Un enfoque práctico*.
- Sommerville, I., *Ingeniería de Software*.
- Inteco. *INGENIERÍA DEL SOFTWARE: METODOLOGÍAS Y CICLOS DE VIDA*