

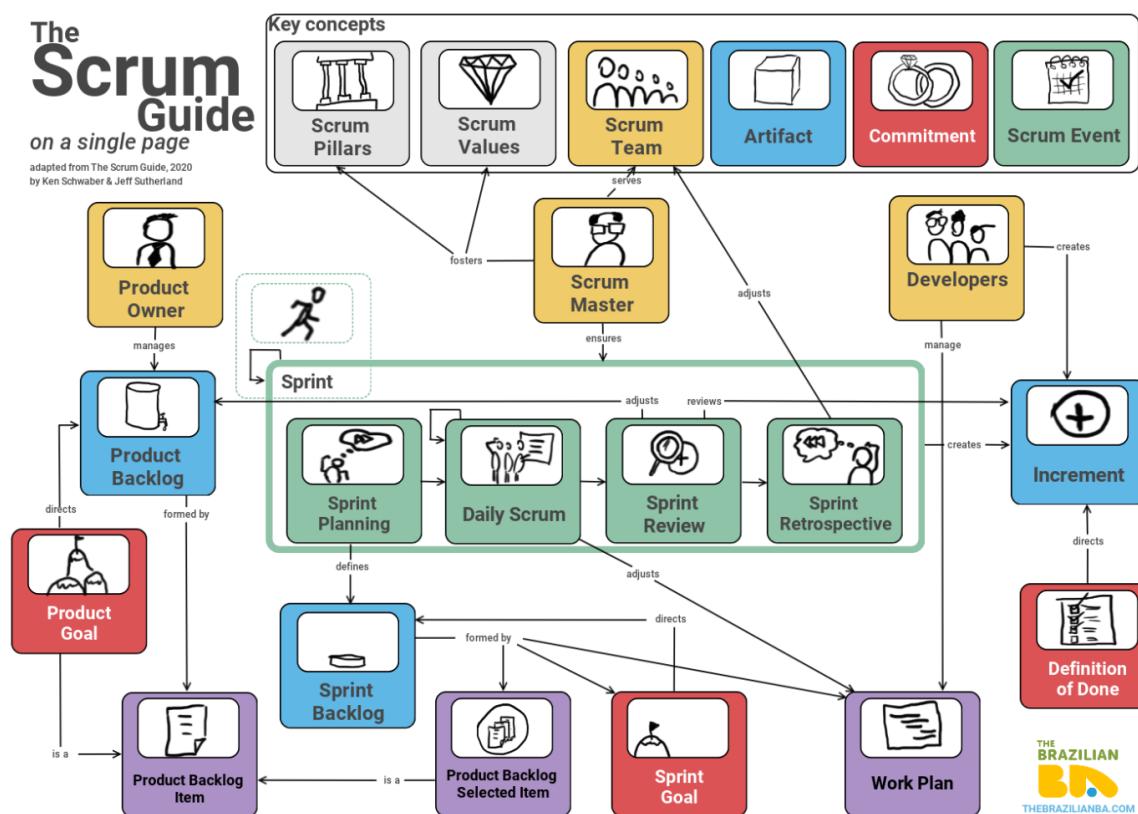
Resumen Parcial 2. ISW

Scrum

Se evalúa la guía Scrum y la información adicional de las profes. Si el parcial es a libro abierto, llevar la guía:

Guía de Scrum

No aclarado en la guía: el refinamiento del product backlog es una **actividad continua**. No se hace durante un momento determinado, si no que puede surgir en cualquier momento.

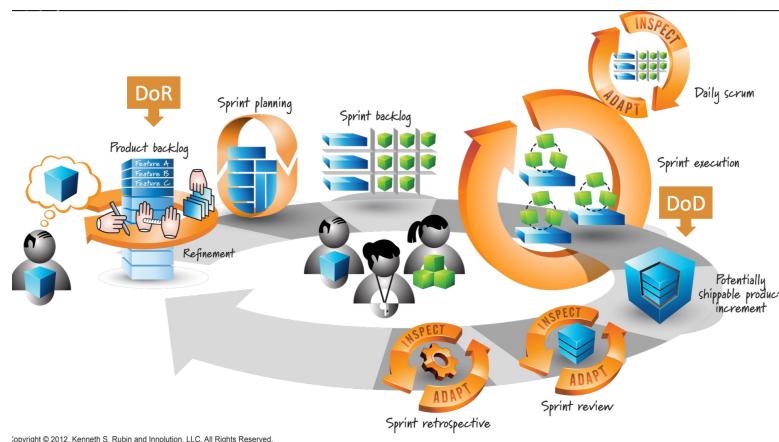


Timebox

- Sprint: es una iteración. 1 mes o menos. La duración también se limita por:
 - No tanto como para que el riesgo sea inaceptable para el Product Owner.

- No tanto como para impedir sincronizar otros eventos del negocio con el equipo de desarrollo.
- Sprint planning: 8 horas máximo para un Sprint de un mes
- Daily Scrum: 15-30 minutos, depende del tamaño del equipo
- Sprint Review: 4 horas máximo para un Sprint de un mes
- Sprint Retrospective: 3 horas máximo para un Sprint de un mes (opcional para el PO)
- Refinamiento del PB: 10% del tiempo del Sprint

Secuencia



Características

- No implementar más de lo acordado
- No aceptar cambios durante la ejecución de una iteración
- Cuidar siempre la excelencia técnica del producto
- Utilizar el proceso que el equipo defina
- **Cimientos de Scrum:** Empirismo, Auto-organización, Colaboración, Priorización, Time Boxing.
- Nunca puede extenderse la duración de un Sprint.
- Daily Scrum: Asisten todos, Se responden las preguntas del scrum master/team
- Product Backlog:
 - Los ítems deberían agregar valor al cliente

- Los ítems están priorizados y ordenados según corresponda
- Los niveles de detalle dependen de la posición que ocupan
- Es un documento vivo
- No contiene ítems de acción o tareas de bajo nivel
- El Equipo SCRUM determina la cantidad de ítems del Product Backlog que podrán cumplimentar durante un Sprint.
- Puede contener errores a solucionar de iteraciones anteriores
- ¿Cuán frecuentes deben ser las reuniones de scrum de status del proyecto? **Diarias**
- El PO no es seleccionado por el Equipo de desarrollo y no asigna tareas.

Definition of Done

El criterio de DoD (Definition of Done) establece cuándo una US se encuentra finalizada. Por lo general, consta de una checklist de actividades a cumplir antes de incorporar la historia en el próximo release de producto de software, pasando previamente por la aceptación del product owner. Por ejemplo:

- Diseño revisado
- Código Completo
 - Código refactorizado
 - Código con formato estándar
 - Código Comentado
 - Código en el repositorio
 - Código Inspeccionado
- Documentación de Usuario actualizada
- Probado
 - Prueba de unidad hecha
 - Prueba de integración hecha
 - Prueba de sistema hecha
- Cero defectos conocidos

Prueba de Aceptación realizada

En los servidores de producción

NO está relacionado al DoR ni al modelo INVEST.

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 (o sea, si entra a la Sprint Planning). Como *mínimo*, se requiere el modelo **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.

Ejemplo:

Valor de negocio claramente expresada.

Detalles suficientemente comprendidos por el Equipo de forma tal que puedan tomar una decisión informada sobre si pueden completar el Product Backlog Item (PBI).

Dependencias identificadas y no hay dependencias externas que puedan impedir que el PBI se complete.

El equipo ha sido asignado adecuadamente para completar el PBI.

El PBI ha sido debidamente estimado y es lo suficientemente pequeño para ser completado en un Sprint.

Los criterios de aceptación son claros y testeables.

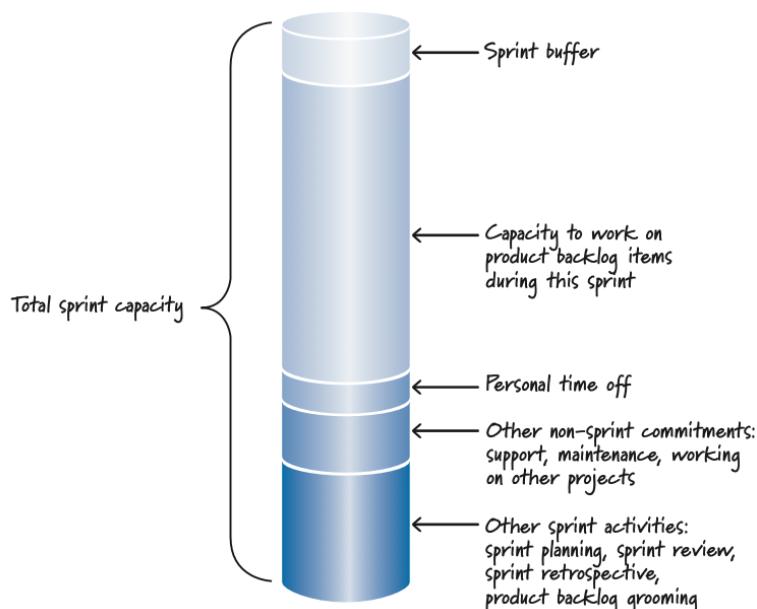
Los criterios de performance si hay, son claros y testeables.

El equipo comprende como mostrar el PBI en la Sprint Review.

Capacidad

La única métrica que se estima, y se usa para ver cuánto compromiso se puede asumir como equipo en un determinado Sprint. Es una de las primeras cosas que se hacen en la Sprint Planning.

- Se determina en la Sprint Planning Meeting
- Se usa para estimar cuánto trabajo puede realizar un equipo durante un sprint
- Se puede medir en story points (puntos de historia)



Capacidad de desarrollo de un equipo en un Sprint

En equipos poco experimentados se mide en rango de horas ideales, en más experimentados en Story Points. Ejemplo:

Persona	Días disponibles (sin tiempo personal)	Días para otras actividades Scrum	Horas por día	Horas de Esfuerzo disponibles
Jorge	10	2	4-7	32-56
Betty	8	2	5-6	30-36
Simón	8	2	4-6	24-36
Pedro	9	2	2-3	14-21
Raúl	10	2	5-6	40-48

Persona	Días disponibles (sin tiempo personal)	Días para otras actividades Scrum	Horas por día	Horas de Esfuerzo disponibles
Total				140-197

Primera variable de definición: la duración del Sprint. Segunda: el cálculo de capacidad, Tercera: el objetivo del Sprint.

Algunas preguntas

- *El Product Owner está en un workshop con el cliente. El cliente explica que el caso de negocio ha cambiado y algunos de los requerimientos ya no serán incluidos. Desafortunadamente, el equipo está trabajando en esas historias en el Sprint actual. ¿Qué debería hacer el Product Owner?*
 - El Product Owner debe llamar al equipo, terminar el Sprint y reiniciarlo con las prioridades cambiadas.
- *Suponiendo que elegimos el framework Scrum para gestionar el proyecto, El Product Owner está en un workshop con el Cliente. El Cliente le explica que el caso de negocio ha cambiado y que algunos de los requerimientos futuros ya no serán incluidos ¿Qué debería hacer el Product Owner?*
 - En el escenario donde el caso de negocio ha cambiado y algunos requisitos futuros ya no serán incluidos, el PO debe actualizar el **Product Backlog** para reflejar las nuevas prioridades y ajustar o eliminar los requisitos según sea necesario. El Product Owner es responsable de gestionar el Product Backlog, asegurándose de que sea transparente y refleje las necesidades y el valor más actual para los interesados. En esta situación, después de comprender el nuevo caso de negocio por parte del cliente, el Product Owner debe:
 - **Repriorizar el Product Backlog:** Eliminar los requerimientos futuros que ya no serán incluidos, y enfocarse en los elementos que ahora proporcionan el mayor valor para el negocio y los usuarios. Esto asegura que el equipo trabaje en los ítems más valiosos y relevantes.
 - **Colaborar con el Scrum Team:** Comunicar estos cambios al Scrum Team para garantizar que todos estén alineados con la nueva dirección y prioridades durante la próxima sesión de **Sprint**

Planning o reunión de **Refinamiento del Backlog**. Esto también ayuda a mantener la transparencia con el equipo de desarrollo y evitar que se trabaje en requisitos obsoletos.

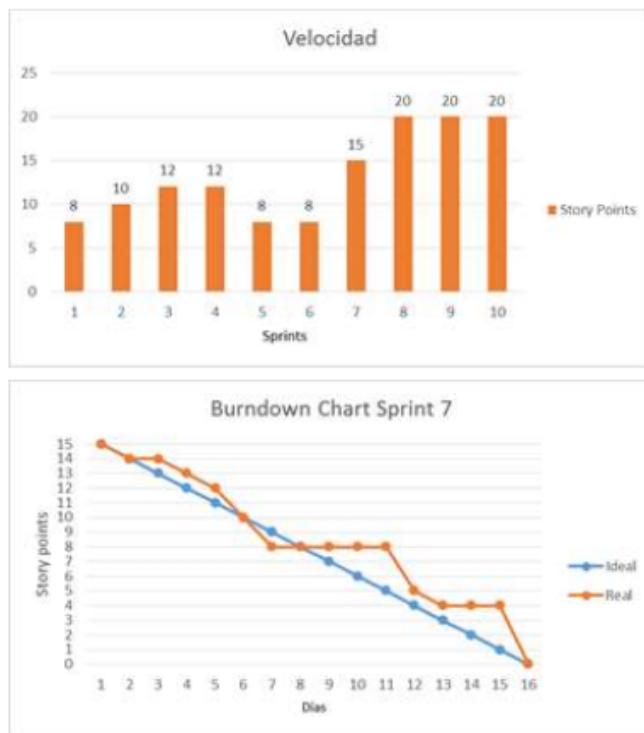
- **Informar a los Stakeholders:** El Product Owner también debe gestionar las expectativas de los stakeholders, asegurándose de que estén al tanto de los cambios y cómo el producto evolucionará según las nuevas prioridades del negocio.
- Otra respuesta, correcta según resolución de parcial:
 - En este caso, si el cliente establece que el caso de negocio ha cambiado, lo que deberá hacer el product owner es trabajar con el Product Backlog. En el Product Backlog tenemos el listado de user stories, épicas o temas a implementar en nuestra aplicación. En el caso de que sea necesario, el product owner debería redefinir el objetivo del producto (ligado al Product Backlog). Esto es solo si los cambios hechos por el cliente requiriera dicha acción. Luego, debería redefinir prioridades, eliminar requerimientos que ya no hagan falta (en user stories) y agregar nuevos si los hubiera. Además, todo este procedimiento debe hacerse con transparencia para poder comunicar de manera abierta al scrum team sobre los cambios nuevos implementados. Esto permite que se siga construyendo el producto con calidad, ya que los valores del negocio están claros para todos.
- *Suponiendo que elegimos el framework Scrum para gestionar el proyecto, ¿Qué debería hacer el Scrum Master en la siguiente situación? Durante una sesión de Sprint Review, mientras se mostraba una historia, aparece un mensaje de error en la aplicación y luego la aplicación se cae.*
 - Controla si el problema está relacionado con una nueva funcionalidad desarrollada en este Sprint, si es así la marca como no terminada. (correcta según multiple choice)
 - Controla si el problema está relacionado con una nueva funcionalidad desarrollada en este Sprint, si no es así, la ingresa en la herramienta de seguimiento de defectos para su resolución. (correcta según multiple choice)
 - Si durante la sprint review visualizamos que se muestra un mensaje de error y se cae el sistema, como Scrum Master debo considerar que la

user story implementada en dicho sprint (que estoy mostrando) no puede ser aprobada y deberá revisarse en el próximo sprint. Esto porque claramente la user story no cumple con los requerimientos de calidad esperados por el product owner y, por ende, no la podemos considerar como implementada y aprobada. Además, la visualización de error y caída del sistema puede funcionar como punto de foco en mi próximo evento: el sprint retrospective. Se podría investigar las causa de falla de dicha user story para que sea lo más transparente posible al resto del equipo. De esa manera, a través de la inspección de dicho error, podríamos readaptar el proceso en caso de ser necesario (si hubiera prácticas o políticas que no estuvieran funcionando hasta el momento). En lo personal, tomaría la user story afectada para correcciones dentro de mi próximo sprint ya que, si fue implementada hasta ese momento, el porque genera valor al negocio y sería prioritario corregirlo y presentarlo en la próxima sprint review de manera correcta y con altos estándares de calidad. (correcta según resolución en un parcial)

- Pedro es el PO de un nuevo proyecto de software. El quiere completar el Product Backlog y crear la lista inicial de user stories y requerimientos. ¿Qué criterio debería considerarse mientras escribe la lista inicial de items?
 - User stories que describen explícitamente las funcionalidades requeridas por el cliente.
 - User stories requeridas para entender el concepto principal del producto.
- *Juan como Scrum Master de un equipo Scrum es invitado a una Scrum Daily Meeting a las 9 am. La reunión siempre excede el límite de tiempo de 15 minutos como todo el mundo habla al mismo tiempo. ¿Que debería hacer Juan?*
 - Asegurarse que el equipo conoce las reglas de la Daily Scrum Meeting; Interrumpir discusiones improductivas de inmediato; y también Introducir una posta (ej. una pequeña pelota). Sólo el miembro que la tenga puede hablar.
- *Juan como Scrum Master de un equipo Scrum es invitado a una Scrum Daily Meeting a las 9 am. Durante la reunion los miembros del equipo actualizan los esfuerzos restantes de sus actividades. Que deberia hacer Juan?*

- Tiene que sumar los valores y actualizar el gráfico de Burndown del Sprint.
- *Al final del Sprint, el Product Owner es invitado a una Sprint Review, el equipo explica que tuvo un problema con un build y que la versión oficial del build no está disponible en ese momento. En lugar de eso ellos prepararon una demo en los equipos de desarrollo. ¿Que debería hacer el Product Owner?*
 - Dar una mirada a las características, pero decirles que no están aceptadas.
 - Pedirles al Scrum Master una reunión adicional cuando el paquete oficial este disponible.
- *Durante la discusión con su cliente el Product Owner notó que se había olvidado de algunas historias importantes. ¿Que debería hacer el Product Owner?*
 - Agregar las nuevas historias en el Product Backlog.
- *En una discusión con uno de los miembros del equipo el Scrum Master advierte que hay aspectos sobre una user story que no están completamente definidos. El PO está muy ocupado y no está disponible para realizar una reunión y responder las preguntas. ¿Qué debería hacer el Scrum Master?*
 - Discutir el tema con el PO y decidir una forma de responder las preguntas.
- *Juan, el Scrum Master del equipo, organiza una Retrospective Meeting. Juan, el equipo y el Product Owner se reúnen en una habitación separada. La reunión empieza y Juan recuerda a cada uno los objetivos y reglas. ¿Cómo debería continuar Juan?*
 - Primero cada uno en la habitación debería decir una breve frase que represente como se sintió en el ultimo Sprint.
- *Las siguientes afirmaciones: (pregunta inicia)*
 - *Los proyectos utilizan estándares y procedimientos*
 - *Se realizan revisiones y auditorías independientes*
 - *Se produce documentación para dar soporte al mantenimiento y la mejora.*

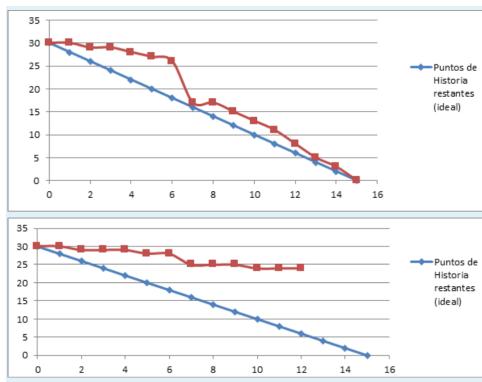
- Las desviaciones a los estándares y procedimientos son expuestas lo más pronto posible.
 - El control de calidad en sí mismo es ejecutado con estándares establecidos
- son aportes vinculados a: (pregunta termina)
- **Incorporar actividades de Aseguramiento de Calidad en la organización. (respuesta)**
 - Análisis de gráficos:



- Ambos graficos resultan positivos, o al menos normales. El grafico de velocidad muestra en general una mejora en el desempeño a lo largo de los sprints, esto es esperable y deseable si el equipo de desarrollo aprende y crece como tal, mejorando su capacidad de generar incrementos. La caída en los Sprints 5 y 6 podría tranquilamente deberse a el faltante de algún miembro del equipo, o un problema inesperado, aun así no es algo grave, dada la naturaleza cambiante del desarrollo de software y los humanos que lo desarrollan es esperable tener este tipo de "caídas" eventualmente.
- El grafico de la quema de puntos muestra también una historia positiva, eventualmente se completo el objetivo del sprint, en el dia 16, si bien

hubo retrasos al principio, y otros dos retrasos grandes casi al final del sprint, el equipo logró llegar al objetivo. Tal vez esto podría indicar que no se fraccionaron correctamente las US, y había alguna con demasiados puntos, que demoró mucho hasta ser completada.

- Especialmente en el día 7 se puede ver que no se queman puntos durante 4 días consecutivos y luego se hace una caída de 3 puntos en un solo día.
- *Durante una Daily Meeting, uno de los miembros del equipo informa que tiene problemas para continuar las tareas porque una de las licencias del software que necesita ha expirado. ¿Qué debería hacer el Scrum Master?*
 - Preguntar al resto del equipo si alguien más está en la misma situación. Luego de la reunión llamar al proveedor y conseguir las licencias.
- *Juan como Scrum Master de un equipo Scrum es invitado a una Scrum Daily Meeting a las 9 am. Durante la reunión Juan nota que las lista de actividades abiertas crece y solo algunas pocas se han cerrado. ¿Qué debería hacer Juan?*
 - Controlar si hay un problema con el equipo o si solo está relacionado a un miembro del equipo. Despues de discutir el problema asegurarse de que el foco es cerrar las actividades iniciadas.
- Al final del sprint, Juan como Scrum Master de un equipo Scrum es invitado para una Sprint Review Meeting. Durante el Sprint el Product Owner estuvo la mayoría del tiempo ocupado y no estuvo disponible para discusiones. Cuando se revisaban las user stories el Product Owner reclamó que la mayoría de las historias no fueron implementadas como el Cliente esperaba. ¿Qué debería hacer Juan?
 - Despues de la reunión discutir la situación con el Product Owner y decidir sobre las medidas para mejorar su disponibilidad durante los siguientes Sprints.
- Suponiendo un desarrollo ágil, analice en los siguientes gráficos el comportamiento del proyecto manifestado en los mismos y explique detalladamente la interpretación que hace luego del análisis:



- En estos graficos podemos observar la comparación entre quema de puntos de historia de manera ideal (en azul) y quema de puntos reales durante el sprint (rojo).
- Análisis primer gráfico: en el primer gráfico observamos que los puntos de historia se mantienen estables es decir, no se queman según lo esperado. En este caso podemos suponer que el equipo enfrentó problemas que dificultaron la quema de puntos. Vemos que a partir del sexto día se produce una quema importante de puntos (aproximadamente 10), probablemente en ese momento encontraron solución al problema y pudieron avanzar de la manera esperada. Otra suposición que podemos hacer es que los puntos fueron quemados de manera adecuada pero los integrantes realizaron la actualización en la herramienta que genera el burndown chart recién el sexto día (mal uso de herramienta)
- Análisis segundo grafico: en el segundo gráfico podemos observar que el equipo solo quemó 5 de los 30 puntos estimados para dicho sprint. Podemos suponer que las historias fueron mal estimadas, que surgieron dudas técnicas, que las user no estaban correctamente refinadas y detalladas, que hubo falta de recursos en el equipo (se enfermaron desarrolladores, no se consideraron licencias). Claramente este es un sprint que no va a cumplir con las expectativas de nuestro PO ya que se han quemado menos del 30% de los puntos estimados en la planning.
- Juan como Scrum Master de un equipo Scrum es invitado a una Scrum Daily Meeting a las 9 am. El equipo está reunido frente al tablero Scrum, pero uno de los miembros siempre llega tarde o ni siquiera aparece. Que debería hacer Juan?
 - Hablar con el miembro del equipo para encontrar una solución

- Juan como Scrum Master de un equipo Scrum es invitado a una Scrum Daily Meeting a las 9 am. Durante la reunión dos de los miembros del equipo discuten respecto de una posible solución para una de las tareas que no han iniciado todavía. ¿Qué debería hacer Juan?
 - Interrumpir la discusión y organizar una reunión de seguimiento inmediatamente después que termine la Daily Meeting.
- Juan como Scrum Master de un equipo Scrum es invitado a una Scrum Daily Meeting a las 9 am. Durante la reunión el equipo completo comienza una discusión profunda y finalmente comienzan a culparse entre ellos. ¿Qué debería hacer Juan?
 - Detener la discusión y continuar con la reunión.
 - Discutir la situación con el equipo y decidir juntos qué hacer para mejorar la situación.
- En una discusión, el arquitecto de software señala que hay tareas adicionales que deben implementarse independientemente de cualquier requerimiento del cliente. ¿Qué debería hacer el Product Owner?
 - Agregar las tareas al PB.
 - Justificación: pueden ir features, cambios de features, defectos, mejoras técnicas, trabajo de adquisición de conocimiento.
- Al final del sprint, Juan como Scrum Master de un equipo Scrum es invitado para una Sprint Review Meeting. La documentación para dos de las user stories no se han terminado conforme se pidió en el criterio de hecho (DoD). ¿Qué debería hacer Juan?
 - Reestimar y terminar esas dos user stories en un Sprint posterior.
- Juan como Scrum Master de un equipo Scrum es invitado a una Scrum Daily Meeting a las 9 am. Uno de los miembros del equipo reporta que necesita soporte para el testing de una de sus tareas. Sin ese soporte requerido la tarea no podrá terminarse hasta el final del Sprint. Un segundo miembro, el que tiene un conocimiento más amplio sobre testing, está actualmente ocupado con otras actividades de mayor prioridad. ¿Qué debería hacer Juan?
 - Controlar si algún otro miembro del equipo puede darle soporte en testing.

- Discutir las prioridades con el Product Owner para entregar el maximo valor para el final del Sprint, como esta situacion puede tener un impacto en la seleccion de los entregables finales del Sprint.
- En una reunion de retrospectiva el equipo reclama que tiene un grupo de user stories para implementar y que todavia no sabe mucho acerca de la estrategia de release detras de esas actividades. Que deberia hacer el Product Owner?
 - Crear un Plan de Release que muestra las prioridades de las user stories y sus fechas de release.
- En un workshop el arquitecto explica que la implementación de cierto conjunto de requerimientos requerirá un gran esfuerzo debido a las limitaciones del framework de desarrollo. ¿Qué debería hacer el Product Owner?
 - Estimar el retorno de inversion (ROI) considerando el valor de negocio, tiempo y costo. Priorizar las caracteristicas de acuerdo con eso.
- El Product Owner es invitado a un workshop para identificar y discutir riesgos de un proyecto. Durante la evaluación de riesgos conocidos, ellos descubrieron que uno de los riesgos previamente identificados ya no es valido. ¿Qué debería hacer el Product Owner?
 - Remover las actividades asociadas a la mitigación del riesgo del Product Backlog

Taskboard



El tablero básico contiene sólo 3 columnas: To Do, Doing y Done. En equipos donde una US es realizada por más de un miembro, tablero con las tareas de las US discriminadas:

Story	To Do	In Process	To Verify	Done
As a user, I... 8 points	Code the... 9 Code the... 2 Test the... 8	Test the... 8 Code the... 8 Test the... 4	Code the... DC 4 Test the... SC 8	Test the... SC 6 Code the... DC 8 Test the... SC 8 Test the... SC 6
As a user, I... 5 points	Code the... 8 Code the... 4	Test the... 8 Code the... 6	Code the... DC 8	Test the... SC 6 Test the... SC 6

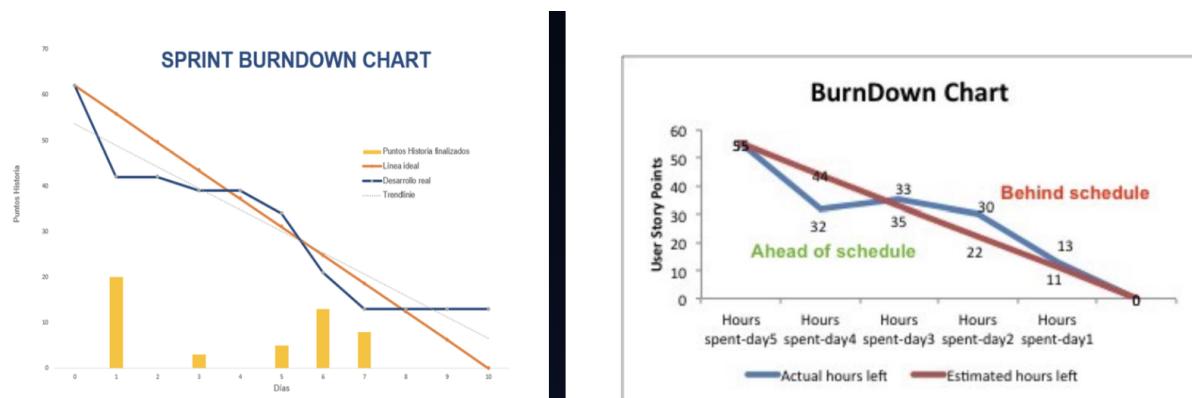
Tablero con tareas. También se suele agregar una columna "done done", que contiene las US cuyas tareas fueron completadas.

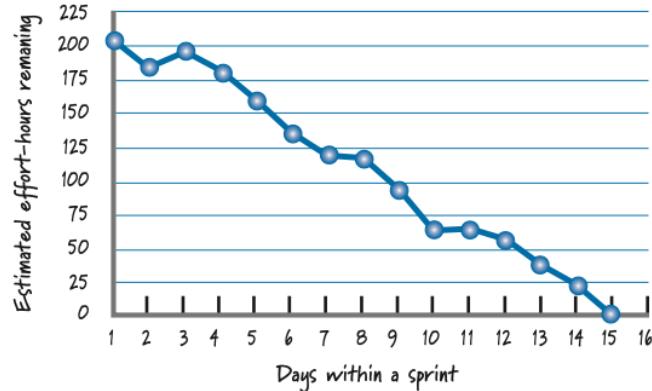
En Agile, las tareas se asignan con sistema **pull**: cada desarrollador elige las tareas con las que se compromete. Deben tener un nivel de granularidad alto.

Sprint Burndown Charts

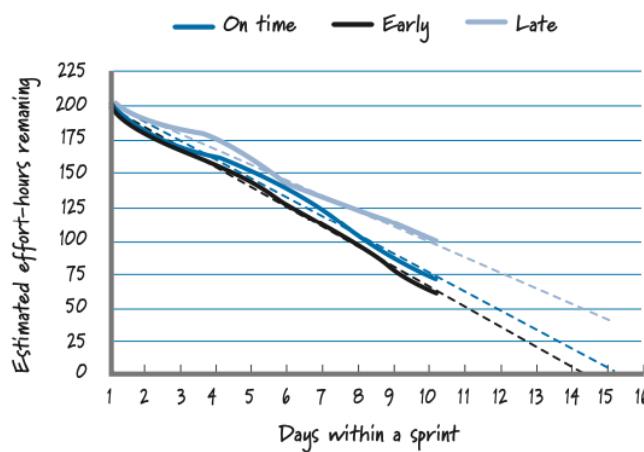
Un Burndown chart representa la tendencia del trabajo remanente a lo largo del sprint, el release o el producto.

Idealmente se actualiza en la Daily, por el Scrum Master. Existe una versión temporal y una permanente. Los **temporales** se limpian a final de cada Sprint, luego de calcular la velocidad del equipo según el gráfico:





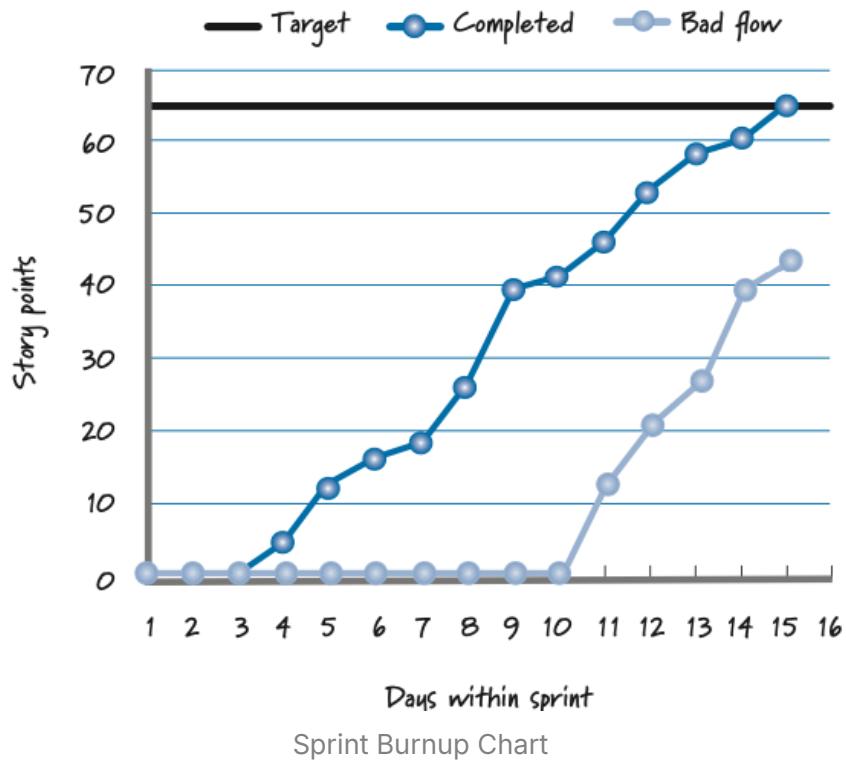
Otro Sprint Burndown chart temporal, que tiene un pico cerca de 200 en el día 3. Eso puede darse por una reestimación. (una US que pasó de 2 a 5 SP)



Sprint Burndown chart con líneas de tendencia

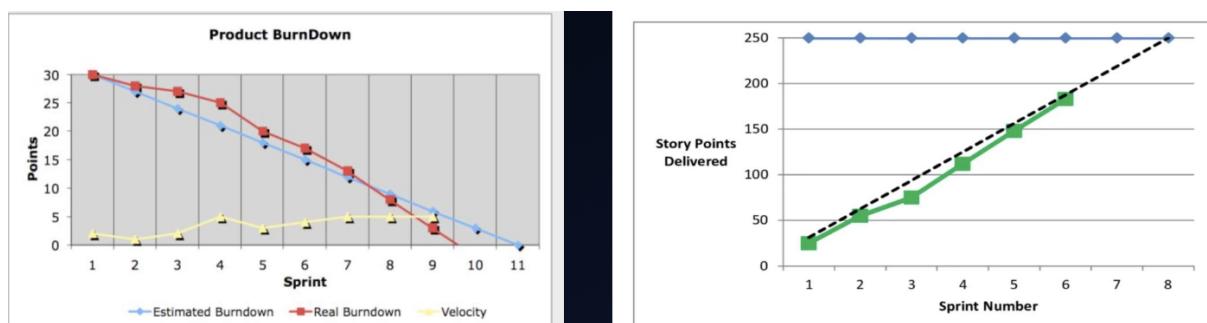
Si el equipo terminó el trabajo comprometido en el Sprint y le sobró tiempo, es recomendable terminar el Sprint en lugar de agregar otra funcionalidad.

La versión **permanente** hace el seguimiento sobre el producto a lo largo del tiempo en base a sprints.



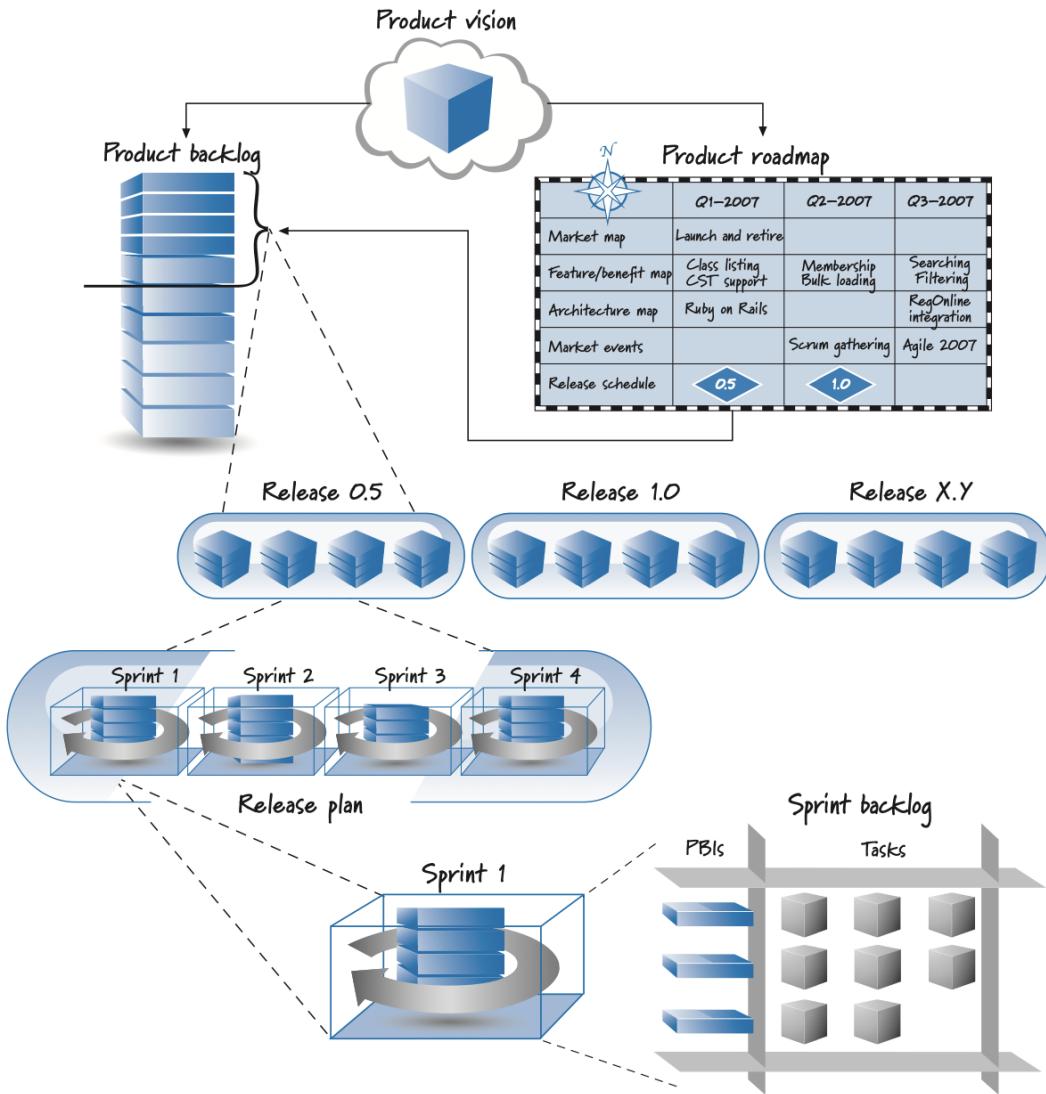
En los Sprint Burnup Chart, el trabajo puede representarse tanto en horas de esfuerzo (como en el gráfico de Sprint Burndown Chart) como en puntos de historia. Muchas personas prefieren usar Story Points en sus gráficos de avance, porque al final del sprint lo único que realmente importa al equipo Scrum es el trabajo de valor para el negocio que se completó durante el sprint, y esto se mide en puntos de historia (o días ideales), no en horas de tareas.

Otros gráficos:



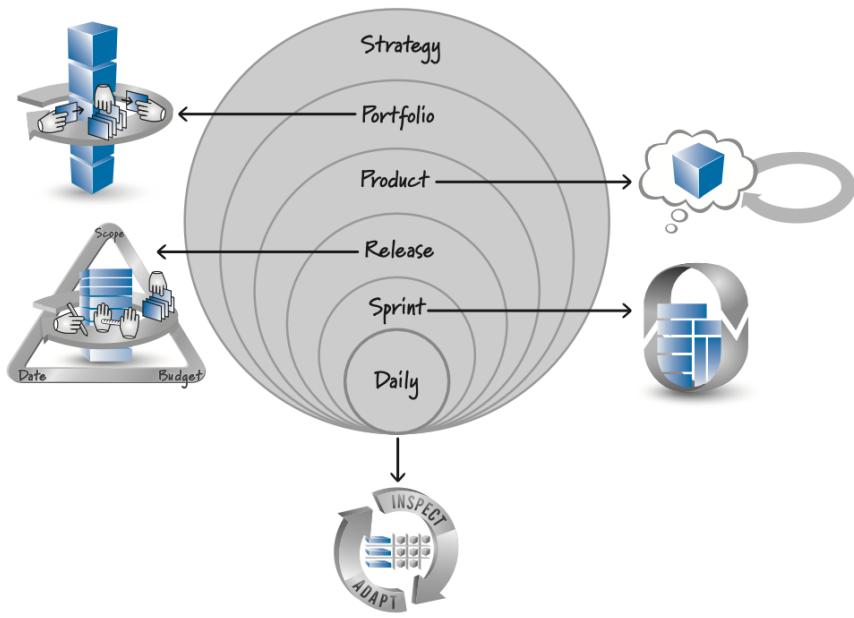
Planificación

Se planifica a nivel jerárquico:



La visión de producto lleva a configurar el PB (no es necesario que esté completo, debe contener al menos funcionalidades que permitan iniciar el trabajo). La sección de Release 0.5, 1.0, X.Y es planificación de producto. La planificación de release es la parte de abajo, con la cantidad de sprints.

Existen distintos niveles de planificación:



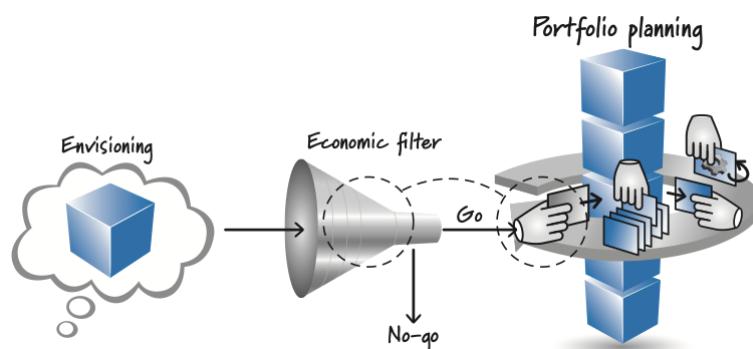
- **Diario:** Daily Scrum
- **Iteración:** Sprint Planning
- **Release:** Release Planning (cuántos Sprints se necesitan para completar las funcionalidades incluidas en el Release) - Triple Restricción
- **Producto:** Sumatoria de Releases a lo largo del tiempo.
- **Portfolio:** Productos de la empresa (Ej. Microsoft tiene Excel, Word, Powerpoint, etc.)
- **Estrategia**

Nivel	Horizonte	Quién	Foco	Entregable
Portfolio	1 año o más	Stakeholders y Product Owners	Administración de un Portfolio de Producto	Backlog de Portfolio
Producto	Varios meses o más	Product Owner y Stakeholders	Visión y evolución del producto a través del tiempo	Visión de Producto, Roadmap y características de alto nivel
Release	3 (o menos) a 9 meses	Equipo Scrum, Stakeholders	Balancear continuamente el valor de cliente y la	Plan de Release

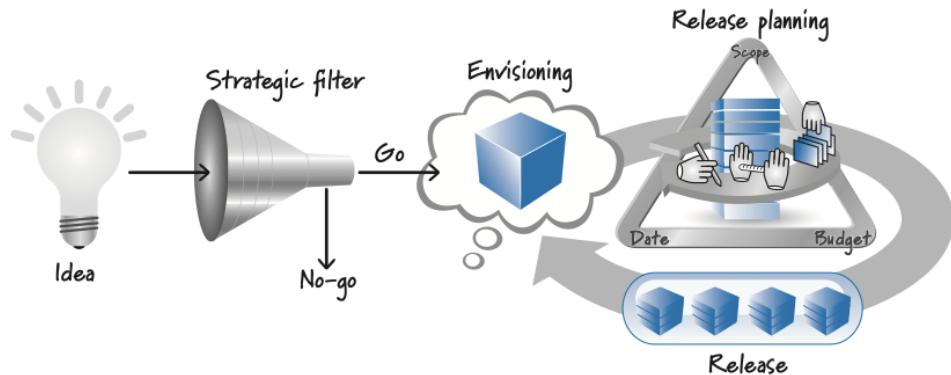
Nivel	Horizonte	Quién	Foco	Entregable
			calidad global con las restricciones de alcance, cronograma y presupuesto	
Iteración	Cada iteración (de 1 Equipo Scrum semana a 1 mes)	Equipo Scrum	Que aspectos entregar en el siguiente sprint	Objetivo del Sprint y Sprint Backlog
Día	Diaria	Equipo Scrum (al menos los que trabajan en IPB)	Cómo completar lo comprometido	Inspección del progreso actual y adaptación a la mejor forma de organizar el siguiente día de trabajo

El **Portfolio backlog** se estima en tamaños de remera (S, M, X, XL). El **Product Backlog** se estima generalmente en SP, y las **tareas del Sprint backlog** se estiman en horas ideales. No todos los equipos dividen las US en tareas.

En empresas también se **planifica el portfolio**:



Planificación de Producto



Idea → Filtro estratégico → Envisioning (Product Planning) → Release Planning

La planificación de producto comienza con una idea para un producto generada por alguien o un equipo (ideación). La idea se evalúa en función de la dirección estratégica de la organización para ver si está alineada y vale la pena explorarla más a fondo. Si pasa esta evaluación, se inicia la planificación inicial para definir cuál debería ser la primera versión mínima, lo que permite una entrega rápida y de bajo costo. Esta primera versión se pone en manos de los usuarios para recopilar retroalimentación, que puede confirmar o desafiar las suposiciones iniciales sobre los clientes objetivo y las características deseadas. Según esta retroalimentación, el equipo puede continuar con la visión actual o pivotar hacia una nueva dirección.

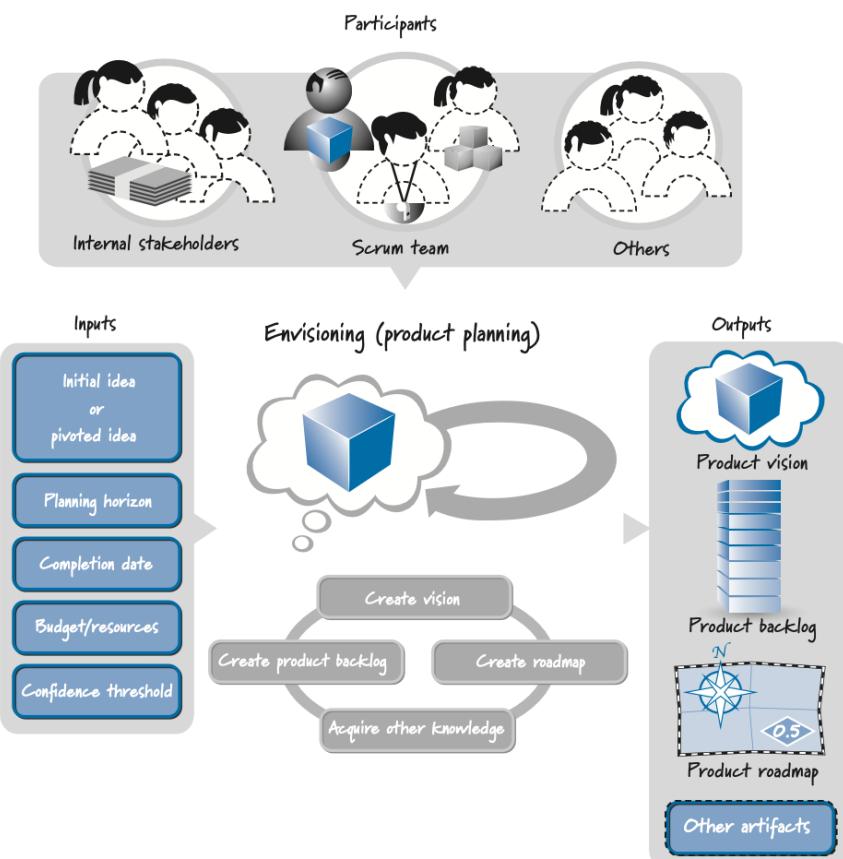
Participantes

El propietario del producto es el único participante requerido durante el proceso de visualización inicial, aunque a veces colabora con partes interesadas internas y especialistas en investigación de mercado, desarrollo de casos de negocio, diseño de experiencia de usuario y arquitectura de sistemas. Idealmente, el ScrumMaster y el equipo de desarrollo participan en este proceso para ofrecer retroalimentación y evitar transferir la visión a otro equipo. Una vez que comienza el desarrollo, el equipo Scrum completo debe ser incluido en cualquier esfuerzo de revisión de la planificación.

Proceso

La entrada principal para la planificación inicial es una idea que ha pasado por el filtro estratégico, mientras que la re-planificación se basa en una idea revisada debido a la retroalimentación, cambios en el financiamiento, acciones de competidores u otros factores significativos. Otros insumos incluyen el horizonte de planificación, la fecha esperada de finalización de la visualización,

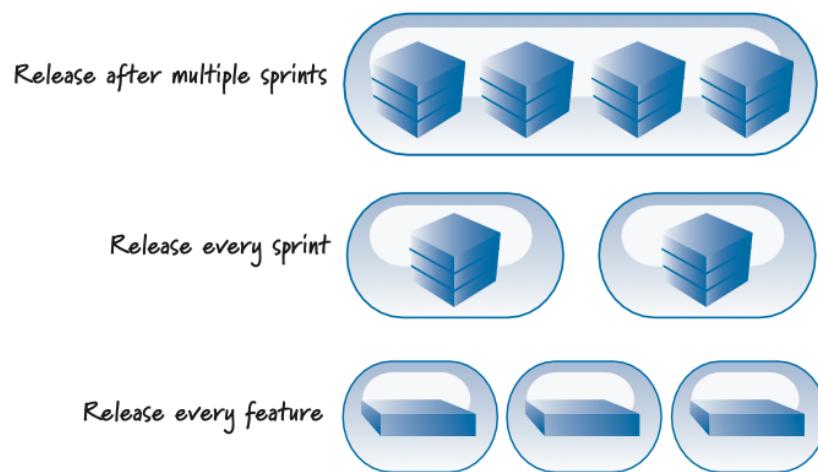
los recursos disponibles y el intervalo de confianza necesario para tomar decisiones de financiamiento.



Actividad de Product Planning. Los participantes opcionales están delimitados por línea discontinua.

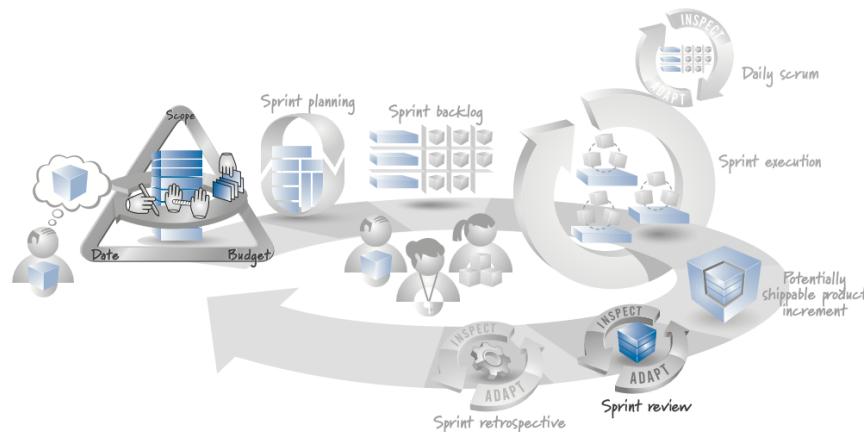
Release Planning

Toda organización debe determinar su cadencia de releases:



Distintas cadencias de Release.

No es un evento único, sino que es un evento frecuente que se puede hacer en cada Sprint. El propósito de la planificación de producto es idear qué debería ser el producto, mientras que el enfoque de la planificación de release es determinar el siguiente paso para alcanzar el objetivo del producto.



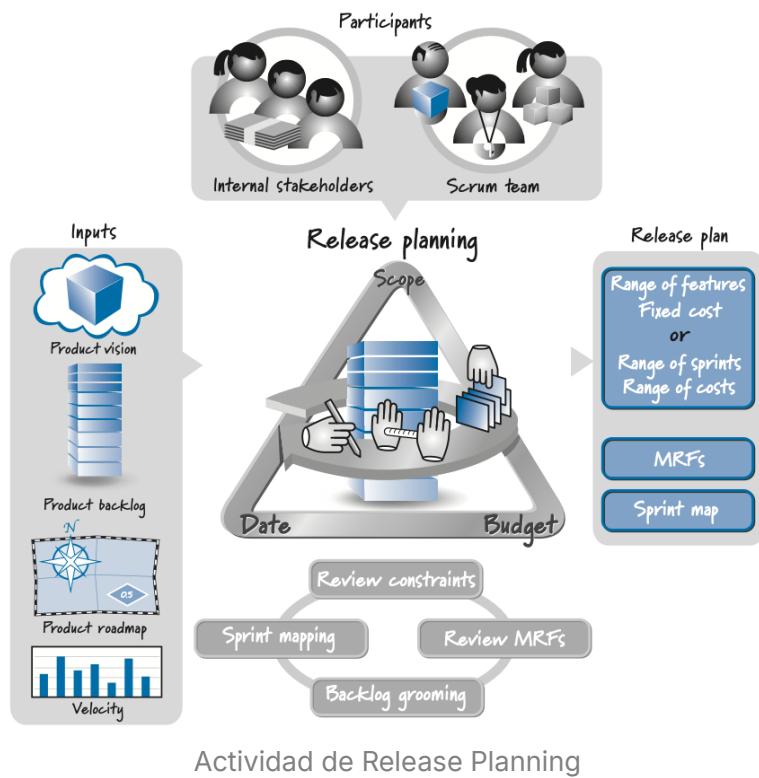
Cuándo ocurre la Release Planning

Participantes

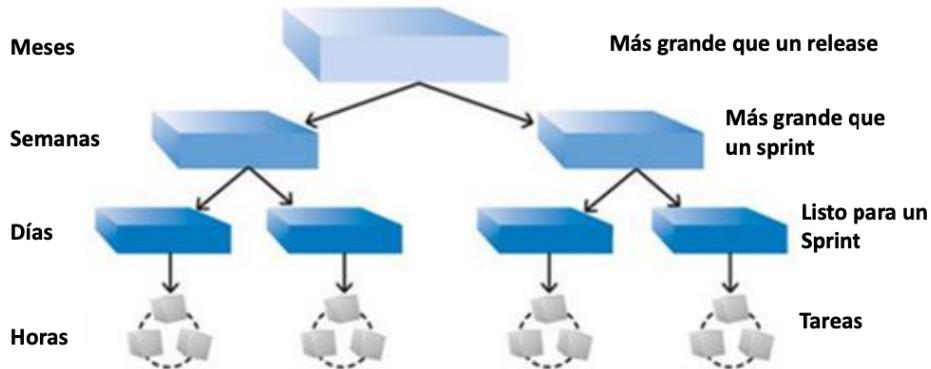
Colaboración entre Stakeholders y Scrum Team.

Proceso

La planificación de lanzamientos toma insumos de la planificación del producto, como la visión del producto, el backlog de alto nivel y la hoja de ruta, además de la velocidad del equipo. Durante la planificación del lanzamiento, se revisan el alcance, la fecha y el presupuesto, y se crean, estiman y priorizan elementos detallados del backlog del producto. También se definen o revisan las Minimally Releasable Features (MRFs) para asegurar que representen el MVP. Puede crearse un mapa de sprints para visualizar los planes de sprints a corto plazo. El resultado de la planificación del lanzamiento es un plan de lanzamiento que comunica cuándo se finalizará, qué características se entregarán y cuáles serán los costos asociados.



Actividad de Release Planning



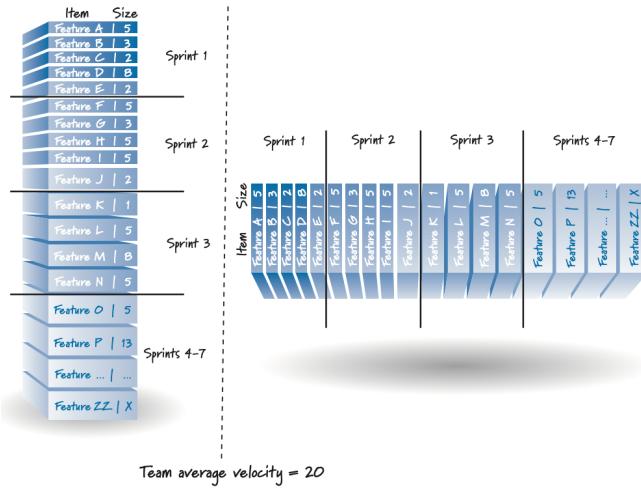
Se trabaja a distintos niveles de granularidad

Sprint Mapping

El mapeo de sprints es una técnica que permite a los equipos planificar provisionalmente qué elementos del backlog del producto se abordarán en sprints futuros *antes* de la planificación formal. Aunque las decisiones finales ocurren durante la planificación del sprint, el mapeo anticipado ayuda a gestionar dependencias entre equipos, especialmente en entornos con múltiples equipos.

Los equipos usan su velocidad para agrupar elementos del backlog en cada sprint, como se muestra en las representaciones vertical y horizontal. La

representación horizontal es útil para alinear equipos Scrum y no Scrum, colocando los mapas de sprints junto a cronogramas de proyectos (como gráficos de Gantt).

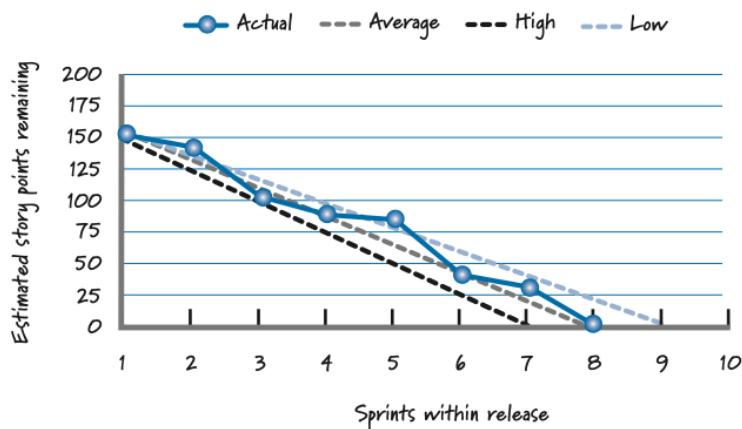


Mapeo de ítems del Product Backlog a Sprints

Release Burndown Chart

El Release Burndown chart para un release de alcance fijo muestra la cantidad total de trabajo sin terminar que queda en cada sprint para alcanzar el objetivo actual del release. En este tipo de gráfico, los números del eje vertical están en las mismas unidades que usamos para dimensionar nuestros elementos del backlog del producto (típicamente puntos de historia o días ideales). El eje horizontal representa los sprints.

Indica cuando es probable terminar el trabajo si nada cambia en el Product Backlog y/o el Equipo de Desarrollo.

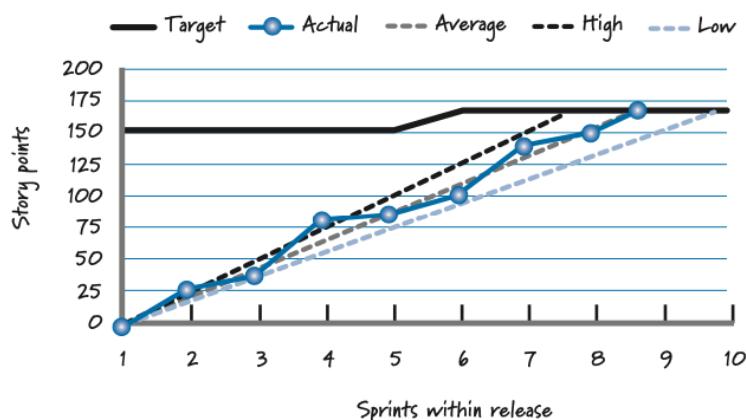
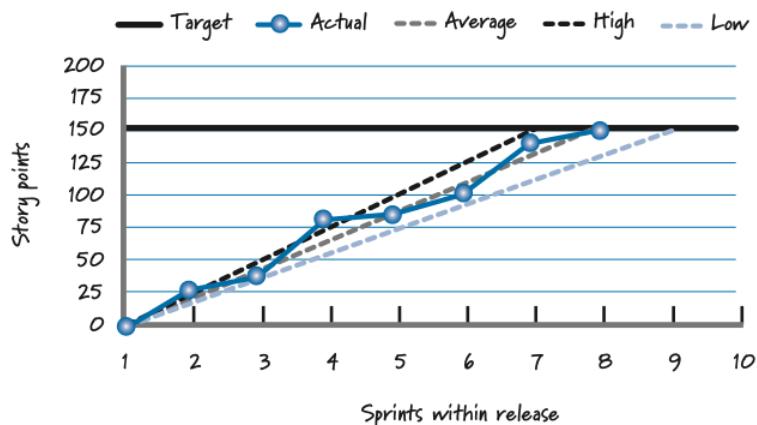


Release Burndown Chart de Alcance fijo

Release Burnup Chart

Un Release Burnup Chart para un release de alcance fijo muestra la cantidad total de trabajo en un lanzamiento como una línea de meta u objetivo y nuestro progreso en cada sprint hacia la consecución de esa meta. Las dimensiones horizontal y vertical del gráfico son idénticas a las del gráfico de trabajo pendiente del lanzamiento.

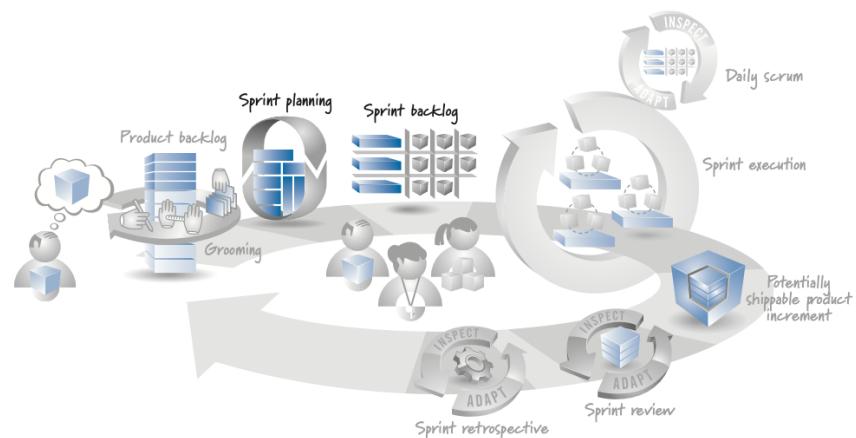
En este gráfico, al final de cada sprint incrementamos los puntos de historia acumulados con el total de puntos de historia completados en ese sprint. El objetivo es avanzar hasta alcanzar el número objetivo de puntos de historia en el lanzamiento. Y, al igual que el gráfico de trabajo pendiente del lanzamiento, este gráfico muestra las mismas tres líneas predictivas que indican el número probable de sprints para alcanzar el objetivo.



Sprint Planning

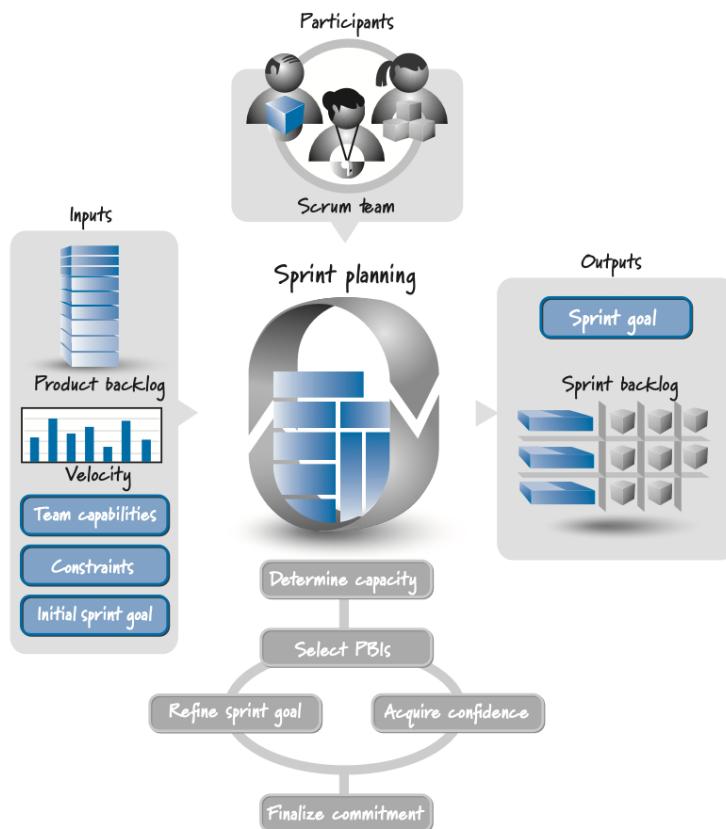
Es una actividad recurrente que se hace just-in-time que toma lugar en el comienzo de cada Sprint. Para un print de 2-4 semanas, la Sprint planning debería tomar no más de 4 a 8 horas a completarse.

Para la primera parte de la reunión, es necesario tener disponible un objetivo para el Sprint, items relevantes del product Backlog, y capacidad del equipo.



Grooming no se hace más, se reemplaza por Refinamiento del PB

Los POs deben ingresar al sprint con una idea de lo que quieren lograr, ya sea algo específico o general. Sin embargo, el equipo de desarrollo y el PO deben colaborar para asegurar un compromiso realista, considerando las capacidades del equipo, su velocidad y cualquier restricción. Por lo general, el equipo luego descompone los elementos del backlog en tareas, con el objetivo de que ninguna tarea lleve más de ocho horas en completarse. El backlog del sprint y el objetivo del sprint son los resultados finales de la planificación, representando el compromiso del equipo para ese sprint.

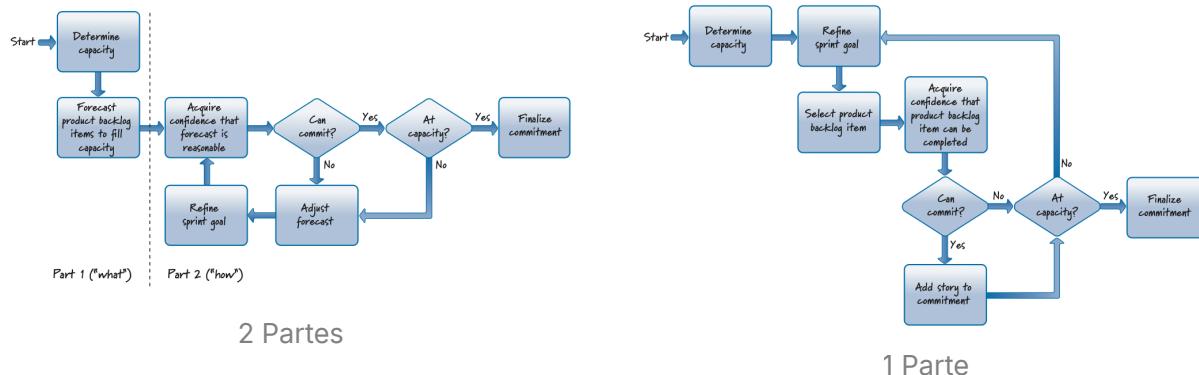


Actividad de Sprint Planning

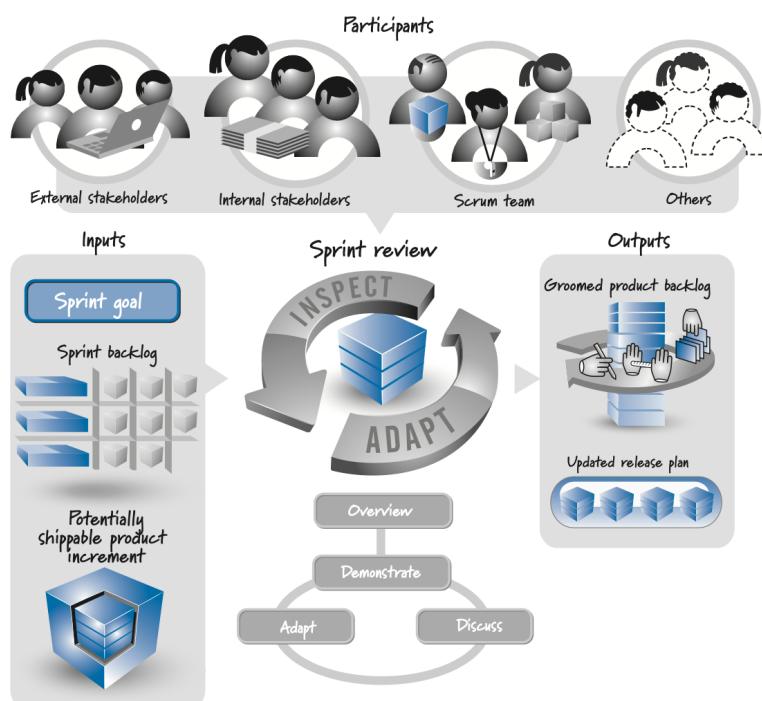
Entradas

- Backlog de producto: Antes de la planificación del sprint, los elementos superiores del backlog del producto fueron refinados hasta un estado *listo*.
- Velocidad del equipo: La velocidad histórica del equipo es un indicador de cuánto trabajo es factible que el equipo complete en un sprint.
- Restricciones: Se identifican las restricciones comerciales o técnicas que podrían afectar lo que el equipo puede entregar.
- Capacidades del equipo: Las capacidades tienen en cuenta qué personas están en el equipo, qué habilidades tiene cada miembro del equipo y qué disponibilidad tendrá cada persona en el próximo sprint.
- Objetivo inicial del sprint: Este es el objetivo de negocio que el PO desea ver cumplido durante el sprint.

Existen dos formas de realizar la Sprint Planning:



Sprint Review



Actividad, participantes, entradas y salidas de la Sprint Review

Métricas en Ambientes Ágiles

La elección de las métricas está guiada por dos principios ágiles:

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

- Running Tested Features. Está obsoleta

- Capacidad: Horas de Trabajo Disponibles por día (WH) X Días Disponibles
Iteración (DA) = Capacidad
- Velocidad: Métrica del progreso de un equipo. Se calcula sumando el número de story points (asignados a cada user story) que el equipo completa durante la iteración.

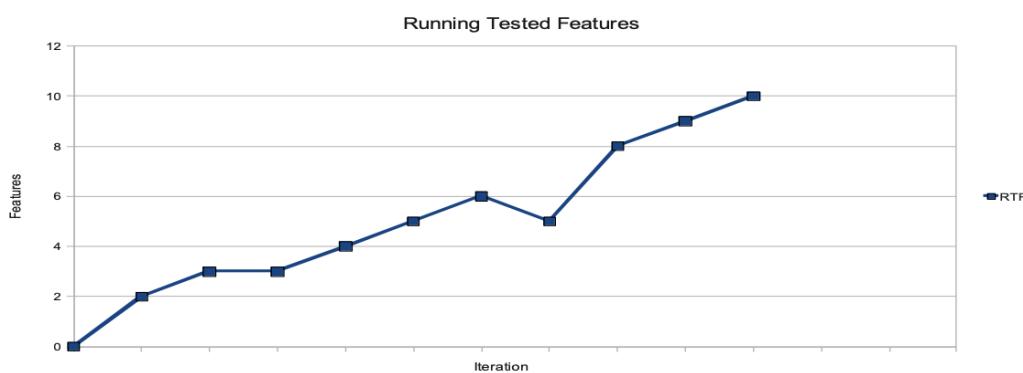
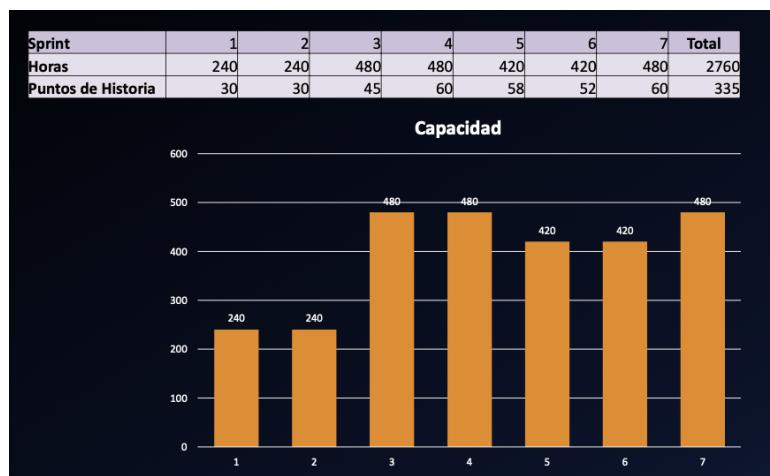


Gráfico de RTF.

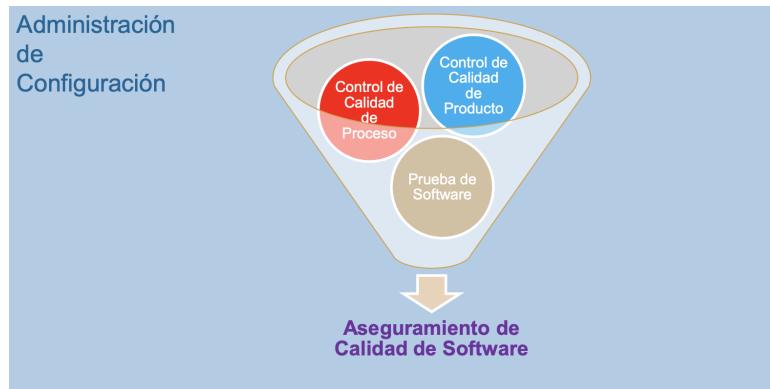
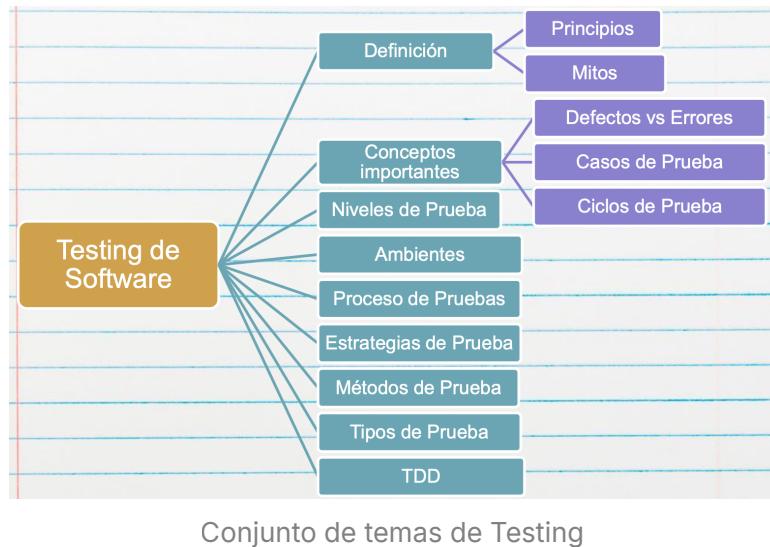


Ejemplo de cálculo de capacidad para un equipo de 8 miembros, en el que 4 miembros están disponibles durante los 2 primeros sprints, 1 miembro se va de vacaciones en sprints 5 y 6 y se trabaja por 6 horas.

Framework para escalar Scrum: Nexus

Marco de trabajo (framework) que consiste de roles, eventos, artefactos y técnicas que vinculan y entrelazan el trabajo de entre tres y nueve equipos Scrum, que trabajan para construir un incremento integrado que cumpla con un objetivo. Debe existir un único DoD. Al implementar Nexus se reemplaza la Sprint Review

Testing de Software



El testing está en el contexto del aseguramiento de calidad (QA). La prueba de software no incluye a control de calidad de proceso ni de calidad de producto.

El testing es un proceso **destructivo** cuyo **objetivo es el de encontrar defectos**. Se asume la presencia de defectos. Mundialmente, este proceso representa el 30-50% del costo de un software confiable.

El costo de corregir un error aumenta exponencialmente en el tiempo.

Es necesario por distintos motivos:

- La existencia de defectos en el software es inevitable.
- Se evitan demandas de clientes

- Se reducen riesgos
- Se construye confianza en el producto
- Las fallas son muy costosas
- Para verificar que el software se ajusta a los requerimientos y validar que las funciones se implementan correctamente.

Asegurar Calidad ≠ Controlar Calidad

Una vez definidos los requerimientos de calidad, se debe tener en cuenta que la calidad no se puede inyectar al final, y que la calidad depende de tareas realizadas durante todo el proceso. Detectar errores ahorra recursos. El testing **no puede asegurar ni calidad en el software ni software de calidad.**

Sigue los siguientes principios:

- El testing muestra presencia de defecto.
- El testing exhaustivo es imposible
- Testing temprano
- Agrupamiento de Defectos
- Paradoja del Pesticida
- El testing es dependiente del contexto
- Falacia de la ausencia de errores (se asume la presencia de errores)
- Un programador debería evitar probar su propio código.
- Una unidad de programación no debería probar sus propios desarrollos.
- Examinar el software para probar que no hace lo que se supone que debería hacer es la mitad de la batalla, la otra mitad es ver que hace lo que no se supone que debería hacer.
- No planificar el esfuerzo de testing sobre la suposición de que no se van a encontrar defectos.

Error ≠ Defecto

Ambos son fallas en un producto de software, pero su clasificación depende del momento en el que se encuentran o producen.

- El **error** se da durante el proceso de desarrollo, en la etapa de implementación. Es conveniente encontrar errores en lugar de defectos,

debido a su bajo costo de corrección.

- El **defecto** es un error que se trasladó a una etapa posterior, como de producción o pruebas finales. Es más caro que un error. El testing busca encontrar **defectos**.

Los defectos se clasifican según su **severidad** y según su **prioridad**.

- Severidad: Bloqueante, crítico, mayor, menor o cosmético.
- Prioridad: Urgencia, alta, media, baja.

Niveles de Testing

Existen 4 niveles principales: (aunque también existe el adHoc Testing)

- Testing unitario o pruebas unitarias
- Pruebas de integración
- Pruebas de sistema
- Pruebas de aceptación

Además, existen 4 ambientes de prueba:

- Desarrollo: testing unitario
- Prueba: testing unitario y de integración
- Pre - Producción: pruebas de sistema y a veces de aceptación
- Producción: pruebas de aceptación

Testing Unitario

- Se prueba cada componente tras su realización/construcción.
- Solo se prueban componentes individuales.
- Se encuentran errores en lugar de defectos.
- Cada componente es probado de forma independiente
- Se produce con acceso al código bajo pruebas y con el apoyo del entorno de desarrollo, tales como un framework de pruebas unitarias o herramientas de depuración.
- Los errores se suelen reparar tan pronto como se encuentran, sin constancia oficial de los incidentes.

- A diferencia de las demás pruebas, lo realizan los desarrolladores. Suelen estar automatizadas.

Ejemplo en Python:

```
import unittest

def sum(a, b):
    return a + b

class TestSum(unittest.TestCase):
    def test_sum(self):
        self.assertEqual(sum(2, 3), 5)
        self.assertEqual(sum(-1, 1), 0)

if __name__ == '__main__':
    unittest.main()
```

Testing de Integración

- Test orientado a verificar que las partes de un sistema que funcionan bien aisladamente (o sea, pasaron por el testing unitario correctamente), también lo hacen en conjunto.
- Cualquier estrategia de prueba de versión o de integración debe ser incremental, para lo que existen dos esquemas principales:
- Integración de arriba hacia abajo (top-down)
- Integración de abajo hacia arriba (bottom-up).
- Lo ideal es una combinación de ambos esquemas.
- Tener en cuenta que los módulos críticos deben ser probados lo más tempranamente posible.
- Los puntos clave del test de integración son simples:
- Conectar de a poco las partes más complejas
- Minimizar la necesidad de programas auxiliares

Testing de Sistema

- Es la prueba realizada cuando una aplicación está funcionando como un todo (Prueba de la construcción Final).
- Trata de determinar si el sistema en su globalidad opera satisfactoriamente (recuperación de fallas, seguridad y protección, stress, performance, etc.)
- El entorno de prueba debe corresponder al entorno de producción tanto como sea posible para reducir al mínimo el riesgo de incidentes debidos al ambiente específicamente y que no se encontraron en las pruebas.
- Deben investigar tanto requerimientos funcionales y no funcionales del sistema.

Testing de Aceptación

- Es la prueba realizada por el usuario para determinar si la aplicación se ajusta a sus necesidades.
- La meta en las pruebas de aceptación es el de establecer confianza en el sistema, las partes del sistema o las características específicas y no funcionales del sistema.
- Encontrar defectos no es el foco principal en las pruebas de aceptación.
- Comprende tanto la prueba realizada por el usuario en ambiente de laboratorio (pruebas alfa), como la prueba en ambientes de trabajo reales (pruebas beta).

Casos de Prueba

Un caso de prueba (CP) Set de condiciones o variables bajo las cuales un tester determinará si el software está funcionando correctamente o no. Incluye las entradas correspondientes y el resultado esperado.

El propósito de diseñar casos de prueba es encontrar el subconjunto de todos los casos que tienen mayor probabilidad de detectar el mayor número posible de defectos.

Buena definición de casos de prueba nos ayuda a **reproducir** defectos. Utiliza datos específicos, como por ejemplo "ingresa el usuario 'juan@gmail.com'" en lugar de "ingresa el email del usuario".

- Objetivo: maximizar errores encontrados
- Criterio: en forma completa

- Restricción: con el mínimo esfuerzo y tiempo

Un caso de prueba es la unidad de la actividad de la prueba. Consta de tres partes:

1. Objetivo: la característica del sistema a comprobar
2. Datos de entrada y de ambiente: datos a introducir al sistema que se encuentra en condiciones preestablecidas.
3. Comportamiento esperado: La salida o la acción esperada en el sistema de acuerdo a los requerimientos del mismo.

Generación de Casos de Prueba

- Ninguna técnica es completa, solucionan distintos problemas
- Lo mejor es combinar varias de estas técnicas para complementar las ventajas de cada una
- Depende del código a testear
- Sin requerimientos todo es mucho más difícil
- Tener en cuenta la conjectura de defectos
- Ser sistemático y documentar las suposiciones sobre el comportamiento o el modelo de fallas

Se pueden derivar casos de prueba desde:

- Documentos del cliente
- Información de relevamiento
- Requerimientos
- Especificaciones de programación
- Código

Condiciones de Prueba

- Esta es la reacción esperada de un sistema frente a un estímulo particular, este estímulo está constituido por las distintas entradas.
- Una condición de prueba debe ser probada por al menos un caso de prueba

- Se definen antes que los casos de prueba

Métodos

Como el tiempo y el presupuesto es limitado, se deben utilizar métodos que prueben la mayor cantidad de funcionalidades con la menor cantidad de pruebas.

- El documento usado para asegurarse que los requerimientos son cubiertos cuando se escriben los test cases es la **matriz de trazabilidad**.

Caja Negra

Testing basado en el análisis de la especificación de una porción del software sin referencia a su estructura interna. Definimos entradas y su resultado esperado, sin conocer el código. Se clasifican en:

- Basado en especificaciones
 - Partición de Equivalencias
 - Análisis de valores límites
 - Etc.
- Basados en la experiencia (no trabajamos esto en el práctico)
 - Adivinanza de Defectos
 - Testing Exploratorio

Caja Blanca

Esta técnica de testing examina la estructura básica de un programa y deriva los datos de testeo desde la lógica del programa, asegurándose que todas las sentencias y condiciones se ejecutan al menos una vez. Se basan en el análisis de la estructura interna del software o un componente del software.

Se puede garantizar el testing coverage

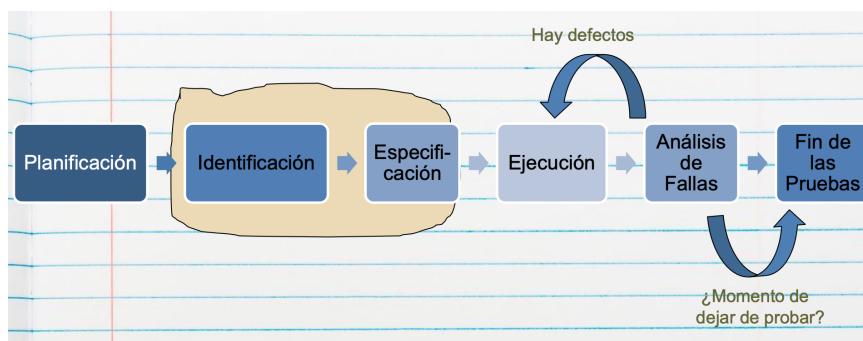
- Cobertura de enunciados o caminos básicos
- Cobertura de sentencias
- Cobertura de decisión
- Cobertura de condición
- Cobertura de decisión/condición

- Cobertura múltiple
- Etc.

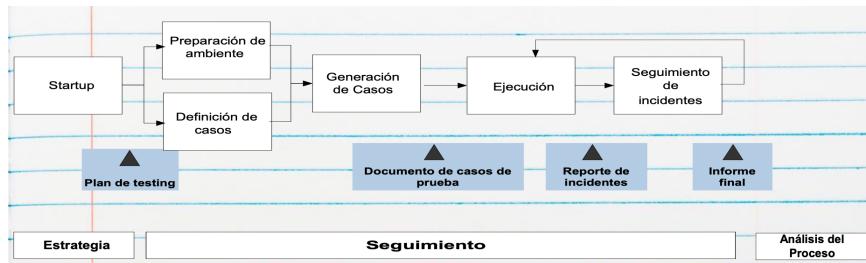
Proceso de Testing

1. Planificación: plan de pruebas (en proyecto tradicional, en ágil podría omitirse).
 - La Planificación de las pruebas es la actividad de verificar que se entienden las metas y los objetivos del cliente, las partes interesadas (stakeholders), el proyecto, y los riesgos de las pruebas que se pretende abordar.
 - Construcción del Test Plan:
 - Riesgos y Objetivos del Testing
 - Estrategia de Testing
 - Recursos
 - Criterio de Aceptación
 - Controlar:
 - Revisar los resultados del testing
 - Test coverage y criterio de aceptación
 - Tomar decisiones
2. Identificación y especificación: escribir los casos de prueba.
 - Revisión de la base de pruebas
 - Verificación de las especificaciones para el software bajo pruebas
 - Evaluar la testeabilidad de los requerimientos y el sistema
 - Identificar los datos necesarios
 - Diseño y priorización de los casos de las pruebas
 - Diseño del entorno de prueba
3. Ejecución: Se ejecutan los CP. Puede automatizada o manual.
 - Desarrollar y dar prioridad a nuestros casos de prueba
 - Crear los datos de prueba

- Automatizar lo que sea necesario
 - Creación de conjuntos de pruebas de los casos de prueba para la ejecución de la prueba eficientemente.
 - Implementar y verificar el ambiente.
 - Ejecutar los casos de prueba
 - Registrar el resultado de la ejecución de pruebas y registrar la identidad y las versiones del software en las herramientas de pruebas.
 - Comparar los resultados reales con los resultados esperados.
4. Análisis de fallas: Si el testing cumplió con su objetivo de detectar defectos, se realiza este análisis de falla.
5. Fin de las pruebas, evaluación y reporte.
- Evaluar los criterios de Aceptación
 - Reporte de los resultados de las pruebas para los stakeholders.
 - Recolección de la información de las actividades de prueba completadas para consolidar.
 - Verificación de los entregables y que los defectos hayan sido corregidos.
 - Evaluación de cómo resultaron las actividades de testing y se analizan las lecciones aprendidas.



Proceso de testing en procesos definidos. En agile es más o menos igual, podría no tener la planificación.



Ídem proceso anterior, pero con entregables.

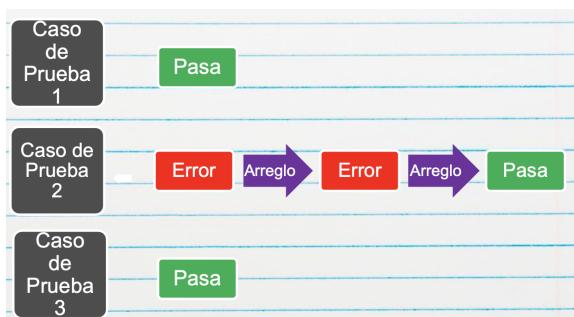
Ciclo de Testing

Un ciclo de pruebas abarca la ejecución de la totalidad de los casos de prueba establecidos aplicados a una misma versión del sistema a probar. Una vez que se corrijen los defectos, se vuelve a realizar otro ciclo. Así hasta que cumplamos con el criterio de aceptación. Hay que definir con el cliente hasta dónde vamos a probar.

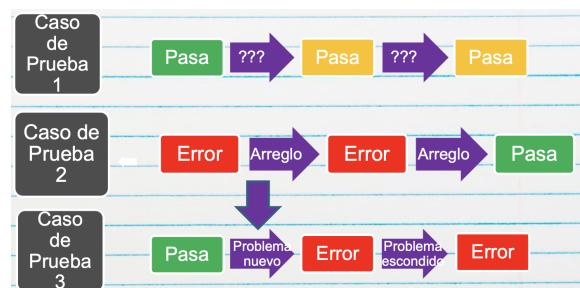
Regresión

Es la actividad de testing en la que se corren todos los casos de prueba en cada ciclo de prueba.

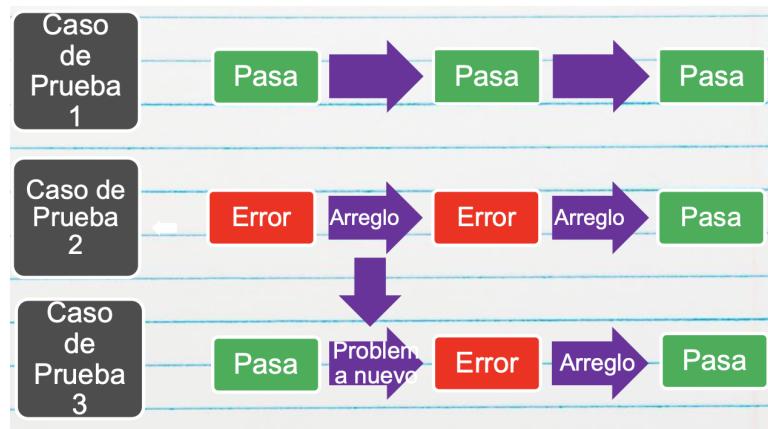
Al concluir un ciclo de pruebas, y reemplazarse la versión del sistema sometido al mismo, debe realizarse una verificación total de la nueva versión, a fin de prevenir la introducción de nuevos defectos al intentar solucionar los detectados.



Sin regresión. Un arreglo no produce errores en otro caso de prueba (CP).



Sin regresión. Un arreglo produjo un problema en otro caso de prueba, y como no se realizan pruebas de regresión, el problema no se encuentra. Tampoco se sabe qué ocurrió con el CP 1.



Con regresión. El arreglo de un CP

Testing en el Ciclo de Vida del Software

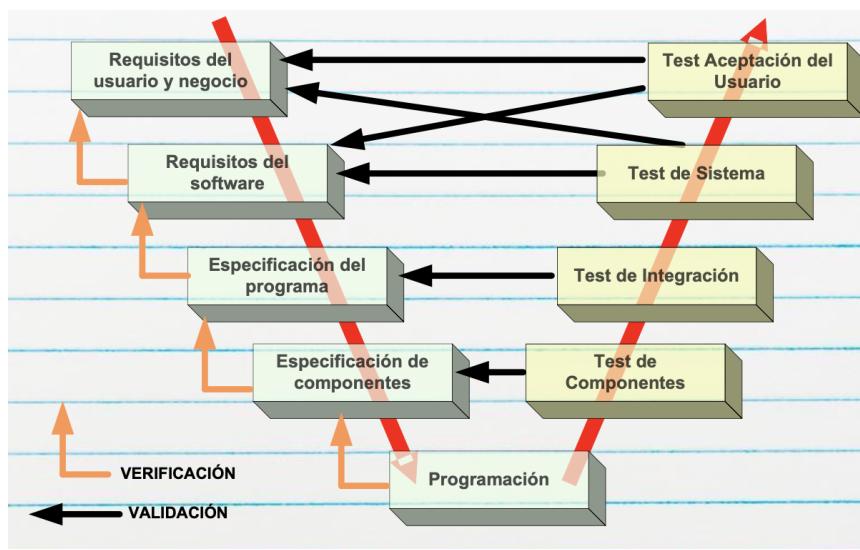
Verificación: estamos construyendo el sistema correctamente?

Validación: estamos construyendo el sistema correcto?

Objetivos de involucrar las actividades de Testing de manera temprana:

- Dar visibilidad de manera temprana al equipo, de cómo se va a probar el producto.
- Disminuir los costos de correcciones de defectos

Si yo tengo los requisitos, ya puedo escribir los casos de prueba. No hace falta escribir código.



Modelo en V

Cuánto Testing es Suficiente

- Cuando se tiene la confianza de que el sistema funciona correctamente
- El testing exhaustivo es imposible.

Decidir cuánto testing es suficiente depende de:

- Evaluación del nivel de riesgo
- Costos asociados al proyecto

Usamos los riesgos para determinar:

- Que probar primero
- A qué dedicarle más esfuerzo de testing
- Que no probar (por ahora)

El Criterio de Aceptación es lo que comúnmente se usa para resolver el problema de determinar cuándo una determinada fase de testing ha sido completada. Puede ser definido en términos de:

- Costos
- % de tests corridos sin fallas
- Fallas predichas aún permanecen en el software
- No hay defectos de una determinada severidad en el software

Tipos de Testing

Smoke Test: Primer corrida de los tests de sistema que provee cierto aseguramiento de que el software que está siendo probado no provoca una falla catastrófica. Es más superficial que el **sanity test**.

- Testing Funcional
 - Las pruebas se basan en funciones y características (descripta en los documentos o entendidas por los testers) y su interoperabilidad con sistemas específicos.
 - Basado en Requerimientos
 - Basado en los procesos de negocio
- Testing No Funcional
 - Es la prueba de “cómo” funciona el sistema. Es más difícil que el testing funcional.

- Los requerimientos no funcionales son tan importantes como los funcionales
 - Performance Testing
 - Pruebas de Carga
 - Pruebas de Stress
 - Pruebas de usabilidad
 - Pruebas de mantenimiento
 - Pruebas de fiabilidad
 - Pruebas de portabilidad
 - Prueba de interfaz de usuario
 - Pruebas de performance (tiempo de respuesta y concurrencia)
 - Pruebas de configuración (verificar como trabaja el sistema)

TDD

Test-Driven Development

- Desarrollo guiado por pruebas de software, o Test-driven development (TDD)
- Es una técnica avanzada que involucra otras dos prácticas: Escribir las pruebas primero (Test First Development) y Refactorización (Refactoring).
- Para escribir las pruebas generalmente se utilizan las pruebas unitarias

Testing Ágil

Casi todo sale del libro

- Manifiesto del testing. En término de valores el testing en entornos ágil.
 - Testing throughout over at the end (a lo largo y no al final del proceso)
 - Es una actividad, no una "fase".
 - Preventing bugs over finding bugs
 - A través de checklists, peer reviews, y automatizaciones, prevenimos errores.

- Testing understanding over testing checking
 - El tester va a ir chequeando como parte de las actividades del sprint, desde la planning identificando posibles CA.
- Building the best system over breaking the system
 - Deja de ser un proceso destructivo, ayuda a construir el mejor sistema
- Team responsibility for quality over tester responsibility
 - La calidad del software depende de todo el equipo, no solo del tester.
- Primer cuadrante: verificación
-

Filosofía Lean y Kanban

Tiene 7 principios:

1. Eliminar desperdicios (dos tipos: el necesario y el puro, y se deben minimizar y evitar respectivamente)

Desperdicios en Manufactura	Desperdicios en servicios basados en conocimiento
Inventario	Trabajo parcialmente terminado
Sobreproducción	Características extra
Proceso extra	Proceso extra
Transporte	Cambio de tareas, Transferencia de conocimiento
Esperas	Esperas/Demoras
Movimiento	Cambio de contextos (Seteo)
Defectos	Defectos
Talento no utilizado	Talento no utilizado

2. Amplificar Aprendizaje (el conocimiento sirve si se comparte, mejora continua)
3. Embeber la Integridad conceptual (calidad, coherencia, consistencia, trabajando en equipo en calidad)

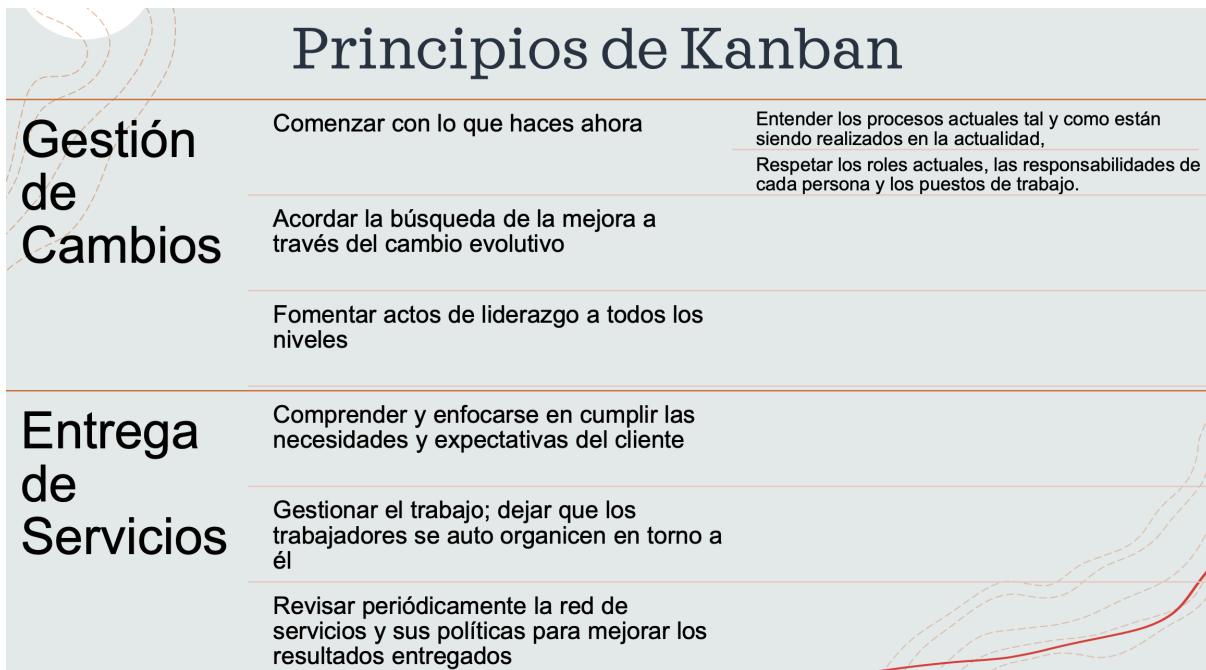
4. Diferir compromisos (hasta el último momento responsable, tomar decisiones informadas,, no tomar compromisos con falta de información)
5. Dar poder al equipo (buscamos equipos motivados, capacitados, autogestionados)
6. Ver el todo (entrega de valor, no solo el código)
7. Entregar lo antes posible (entrega frecuente de productos funcionando)

Kanban

Es un método para definir, gestionar y mejorar servicios que entregan trabajo del conocimiento, tales como servicios profesionales, trabajos o actividades en las que interviene la creatividad y el diseño tanto de productos de software como físicos.

Valores

- Transparencia
- Equilibrio
- Colaboración
- Foco en el cliente
- Flujo
- Liderazgo
- Entendimiento
- Acuerdo
- Respeto



Prácticas de Kanban

- Visualizar
 - Muestra el trabajo y su flujo
 - Visualiza los riesgos
 - Construye un modelo visual que refleje cómo se trabaja
 - Permite absorber y procesar una gran cantidad de información en un corto intervalo de tiempo.
 - Habilita la cooperación, ya que todo el mundo tiene la misma imagen.
 - Permite tomar decisiones de una manera más rápida y colaborativa.
- Limitar el trabajo en curso (Work in Progress WIP)
 - Limitar el trabajo en el sistema a la capacidad disponible
 - Los sistemas eficaces son los que se centran más en el flujo de trabajo y menos en tener ocupados a los trabajadores.
 - Cuando los recursos están ocupados completamente, no hay holgura en el sistema y como resultado el flujo es deficiente, como ocurre en una autopista en las horas pico.
 - Limitamos el WiP para equilibrar la ocupación y asegurar que haya flujo de trabajo.

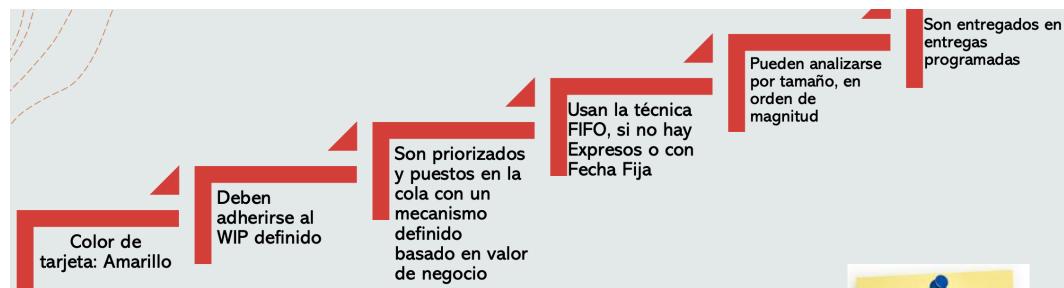
- Asignar límites explícitos de cuántos ítems puede haber en progreso en cada estado del flujo de trabajo
- Gestionar el Flujo de Trabajo
 - El objetivo es poder terminar el trabajo de la forma más fluida y predecible posible, mientras se mantiene un ritmo sostenido.
 - Limitar el trabajo en curso nos ayuda a asegurar un flujo suave y predictivo.
 - El seguimiento y la medición del flujo de trabajo da como resultado información valiosa y útil para gestionar las expectativas de los clientes, hacer predicciones y mejorar.
 - Para gestionar el flujo de trabajo se debe modelar el proceso.
 - Definir tipos de trabajo, asignando capacidad en función de la demanda:
 - Requerimientos: Caso de uso, Historias de Usuario, Características
 - Defectos
 - Desarrollo: Mantenimiento, Refactorización, Actualización de Infraestructura
 - Solicitudes: Solicitud de Cambio, Sugerencias de Mejora
- Hacer las políticas explícitas
 - Tener pocas políticas sencillas acordadas visibles para todos en todo momento: clases de servicio, criterios de pull, límites de WIP
 - Fácilmente modificables por los que prestan el servicio
 - Ejemplo para clase de servicio “expreso”



- Ejemplo para clase de servicio “fecha fija”



- Ejemplo para clase de servicio "estándar"



- Ejemplo para clase de servicio "intangible"



- Establecer ciclos de retroalimentación
- Mejorar colaborativamente, evolucionar experimentalmente
 - Usando el método científico
 - Diseñamos experimentos en entornos donde fallar es seguro de tal forma que si nuestra hipótesis es correcta y el experimento da buenos resultados, mantenemos el cambio.
 - Si el resultado no es positivo, podemos fácilmente volver al estado anterior.

Cómo Aplicar Kanban

- Proceso: modelar nuestro proceso.
- Trabajo: decidir la unidad de trabajo.
- Límites de WIP: limitar el WIP para ayudar al flujo de trabajo.

- Política: definir políticas de calidad.
- Cuellos de Botella y Flujo: mover recursos a los cuellos de botella.
- Clase de Servicio: diferentes trabajos tienen diferentes políticas – definición de hecho (“done”), para cada estado.
- Cadencia: Releases, planificaciones, revisiones

Para modelar el proceso se realiza una tabla como la siguiente:

Cola de Product o	Análisis		Desarrollo		Listo para Build	En Testing		En Producció n
	En progr eso	Hecho	En progreso	Hecho		En Progreso	Listo para Despliegue	
Casos de Uso 60 %								
Mantenimiento 30 %								
Defectos 10 %								

Tabla de modelado de proceso, con capacidad por tipo de trabajo asignada en función de la demanda

Definir el WiP, asignando a cada columna de la tabla la cantidad de trabajo en proceso (idealmente horas, pero puede ser días, semanas, etc.)

Definir políticas, incluyendo:

- Reposición de trabajo en el tablero (cuándo, cuánta cantidad, por quién...)
- Definición de cuándo una actividad está terminada y ese elemento de trabajo puede continuar a través del flujo (criterios de tracción)
- Límites del trabajo en curso (WiP limits) Políticas para gestionar elementos de trabajo de diferentes clases de servicio Cadencia de las reuniones (Horarios y agendas)
- Otros principios y acuerdos de colaboración

Definir cadencia:

Cadencia	Ejemplo de frecuencia	Propósito
Team Kanban Meeting	Diaría	Observar y seguir el estado y flujo del trabajo (no de los trabajadores). ¿Cómo podemos

Cadencia	Ejemplo de frecuencia	Propósito
		entregar los elementos de trabajo más rápido en el sistema? ¿Hay capacidad disponible? ¿Qué debemos tomar a continuación?
Team Retrospective	Quincenal o mensual	Reflexionar sobre cómo el equipo gestiona su trabajo y cómo pueden mejorar
Team Replenishment Meeting	Semanalmente o a demanda	Seleccionar los elementos de la lista de trabajo para realizar a continuación

Métricas Kanban

Cycle Time = Vista Interna

Es la métrica que registra el tiempo que sucede entre el inicio y el final del proceso, para un ítem de trabajo dado. Se suele medir en días de trabajo o esfuerzo.

Medición más mecánica de la capacidad del proceso. Es el "**Ritmo de Terminación**".

Todas las columnas de la tabla menos Cola de Producto

Lead Time = Vista del Cliente

Es la métrica que registra el tiempo que sucede entre el momento en el cual se está pidiendo un ítem de trabajo y el momento de su entrega (el final del proceso). Se suele medir en días de trabajo. Es el "**Ritmo de entrega**"

Si yo no pongo límite de WiP en el Product Backlog, se pueden acumular tareas de forma indefinida y aumentar el Lead Time.

Va desde Cola de producto hasta el final

Touch Time

Es el tiempo en el cual un ítem de trabajo fue realmente trabajado (o "tocado") por el equipo.

Se calcula según cuántos días hábiles pasó este ítem en columnas de "trabajo en curso", en oposición con columnas de cola / buffer y estado.

Touch Time \leq Cycle Time \leq Lead Time

Sumo Análisis "En proceso", Desarrollo "En proceso" y En testing "En proceso"

Eficiencia del Ciclo de Proceso

% Eficiencia ciclo proceso = Touch Time / Lead Time.

Tarea	Fecha de Solicitud	Fecha de Inicio	Fecha de Fin	Días sin trabajo	Lead Time	Cycle Time	Touch Time
23	Agosto 3	Agosto 5	Agosto 12	1	8	7	4

Ejemplo de recopilación de métricas contabilizado en días. Se observa un buen tiempo de terminación, dada la poca diferencia entre cycle y lead time.

Métricas Orientadas a Servicio

- Expectativa de nivel de servicio que los clientes esperan
- Capacidad del nivel de servicio al que el sistema puede entregar.
- Acuerdo de nivel de servicio que es acordado con el cliente.
- Umbral de la adecuación del servicio el nivel por debajo del cual este es inaceptable para el cliente.

Métricas

El dominio de las métricas del software se divide en:

- Métricas de proceso.
- Métricas de proyecto.
- Métricas de producto.

Las métricas del proyecto se consolidan para crear métricas de proceso que sean públicas para toda la organización del software. Las métricas no se pueden atribuir ni a una persona, ni a un producto, ni a un proceso ni a un proyecto en particular.

La cobertura de testing se define como: "La medida de que tan completo fue el testing en relación con una estrategia particular"

<ul style="list-style-type: none"> • Desarrollador 	<ul style="list-style-type: none"> • Equipo de Desarrollo
<ol style="list-style-type: none"> 1. Esfuerzo 2. Esfuerzo y duración estimada y actual de una tarea. 3. % de cobertura por el unit test 4. Número y tipo de defectos encontrados en el unit test. 5. Número y tipo de defectos encontrados en revisión por pares. 	<ol style="list-style-type: none"> 1. Tamaño del producto 2. Duración estimada y actual entre los hitos más importantes. 3. Niveles de staffing actuales y estimados. 4. Nro. de tareas planificadas y completadas. 5. Distribución del esfuerzo 6. Status de requerimientos. 7. Volatilidad de requerimientos. 8. Nro. de defectos encontrados en la integración y prueba de sistemas. 9. Nro. de defectos encontrados en peer reviews. 10. Status de distribución de defectos. 11. % de test ejecutados

Tipos de métrica según personas. El % de cobertura es la cantidad de líneas de código probadas sobre la totalidad de líneas.

Métricas básicas para un proyecto de software:

- Esfuerzo (proyecto)
- Tiempo (proyecto)
- Tamaño (producto)
- Defectos (producto)

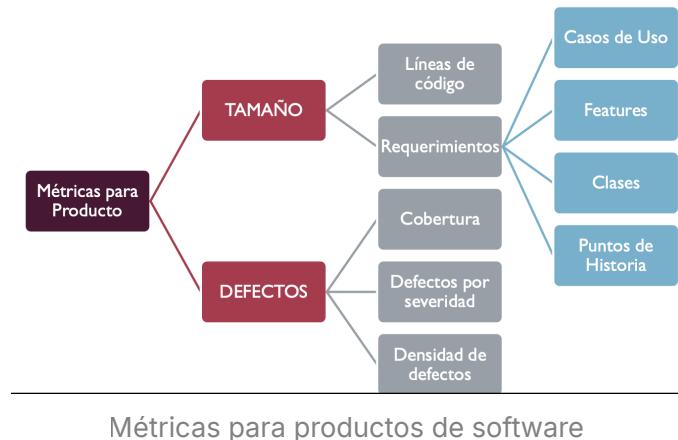
Ejemplo de métrica de proceso: desviación de esfuerzo estimado / real en la organización. Para obtener esa métrica, obtengo las métricas de todos los proyectos, las hago anónimas, y hago un promedio.

Otro ejemplo de métrica de proceso: índice de defectos graves defectos graves / 1.000.000 líneas de código (sin comentarios) a nivel organizacional.

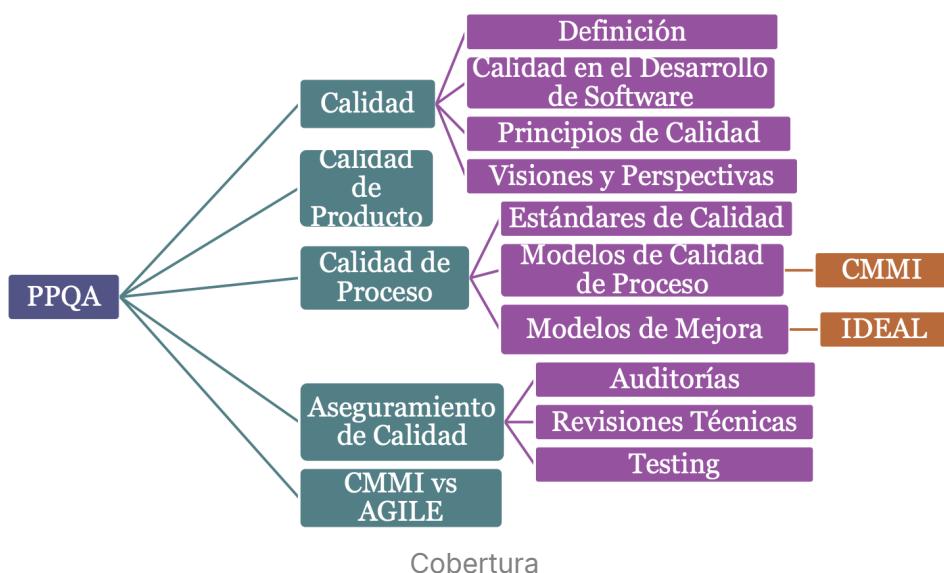
Métricas según cada enfoque

- Tradicionales
 - Esfuerzo
 - Tiempo
 - Costos
 - Defectos
- Ágiles
 - Velocidad
 - Capacidad
 - Running Tested Features

- Kanban
 - Lead Time
 - Cycle Time
 - Touch Time
 - Eficiencia del proceso



Aseguramiento de Calidad



Calidad: Todos los aspectos y características de un producto o servicio que se relacionan con su habilidad de alcanzar las necesidades manifiestas o implícitas. Es *relativa a las personas*.

Para hacer aseguramiento de **calidad de proceso** en un proyecto es necesario que el proyecto tenga un proceso definido de antemano.

La calidad de un producto desarrollado está influenciada por la calidad del proceso de producción utilizado.

Durante los proyectos de desarrollo de software, ocurren distintos problemas: Atrasos en las entregas; Costos Excedidos; Falta cumplimiento de los compromisos; No están claros los requerimientos; El software no hace lo que tiene que hacer; Trabajo fuera de hora; Fenómeno del 90-90 (90% hecho, 90% faltante, casi siempre está casi listo); ¿Dónde está ese componente?.

Un Software de Calidad satisface:

- Las expectativas del Cliente
- Las expectativas del Usuario
- Las necesidades de la gerencia
- Las necesidades del equipo de desarrollo y mantenimiento
- Otros interesados...

Principios

- La calidad no se 'inyecta' ni se compra, debe estar embebida.
- Es un esfuerzo de todos
- Las personas son la clave para lograrlo
- Capacitación
- Se necesita sponsor a nivel gerencial
- Pero se puede empezar por uno
- Se debe liderar con el ejemplo
- No se puede controlar lo que no se mide
- Simplicidad, empezar por lo básico
- El aseguramiento de la calidad debe planificarse
- El aumento de las pruebas no aumenta la calidad
- Debe ser razonable para mi negocio

La calidad tiene distintos puntos de vista:

- Visión del usuario
- Visión de manufactura
- Visión del producto
- Visión basada en el valor
- Visión trascendental

Uno de los grandes desafíos a la hora de realizar software es lograr una coincidencia de estas 3 perspectivas de la calidad. Cuando uno empieza un producto de software resulta que tiene una expectativa de la calidad que ese producto tenga (calidad programada), y obviamente esa calidad programada tiene que estar relacionada lo máximo posible con la calidad necesaria (mínimo que el producto debe tener para satisfacer la necesidad del usuario) y calidad de realizada suele ser la menor de las 3 es lo que realmente hicimos. La intersección de las 3 debe ser lo suficientemente grande como para cubrirlas a las 3, de lo contrario todo lo que hagamos por fuera de este vínculo es desperdicio, generando insatisfacción en los clientes.

Calidad programada: calidad que espero que tenga el producto de software que estamos construyendo como equipo de desarrollo

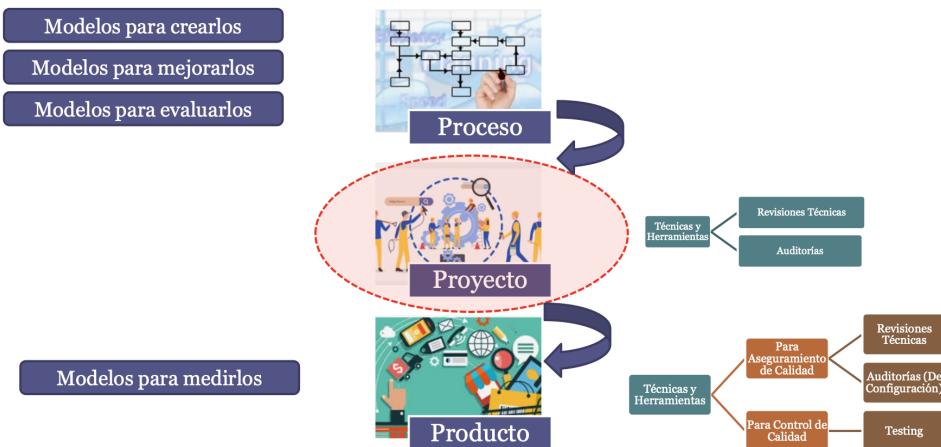
Calidad Necesaria: calidad mínima que debería tener el producto de software

Calidad Realizada: es lo que realmente ocurrió

Si ponemos las tres perspectivas buscando con la Gestión de calidad que la intersección de los 3 ejes sea lo más coincidente posible, es decir lo suficientemente grande para cubrir las 3 perspectivas, de lo contrario todo lo que hagamos por fuera de vinculo de la calidad programada, necesaria y realizada es desperdicio y genera insatisfacción en los clientes. Cuanto más lejos estén una de otras más dificultades y problemas vamos a tener (Respuesta 25/25 en cuestionario)



Todo lo que esté fuera de dicha coincidencia será desperdicio o insatisfacción.



Calidad en el Software. El proyecto está en rojo porque es donde ocurren todas las actividades de aseguramiento de calidad, tanto de proceso como de producto.

Para hacer software, se necesita un proceso (empírico o definido). Para crear el proceso, las organizaciones se basan en modelos. Pueden ser modelos para crear, mejorar o evaluar procesos. Revisiones técnicas son actividades realizadas por pares (internas) para el aseguramiento de calidad de productos y revisión de proyectos. Las auditorías son externas. Los agilistas consideran que las auditorías no son necesarias, porque por ejemplo en Scrum la ceremonia Retrospective sirve para este objetivo.

El proyecto es el que materializa el proceso. En Scrum, la ceremonia que revisa el producto es la Sprint Review.

La auditoría de configuración **funcional, valida**. La auditoría de configuración **física, verifica**. Se hacen a una versión específica de un producto de software en su línea base.

Para controlar la calidad se realiza el testing.

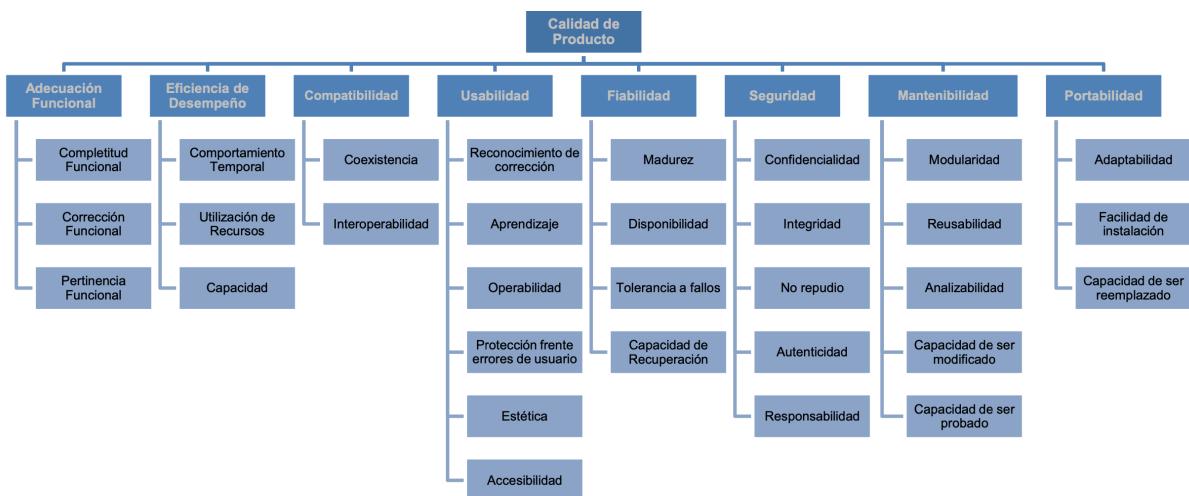
Procesos definidos: el proceso debe tener calidad. Si tiene calidad, y las personas en el contexto del proyecto lo respeta, como consecuencia el producto que se obtenga también tendrá calidad. Esto aplica para cualquier modelo de calidad. La calidad del producto depende de la calidad del proceso. Esto tiene un problema: en la práctica, por más que se tenga un proceso de calidad, no se garantiza que genere un resultado de calidad. Hay un grupo de proceso en las organizaciones que viene impuesto.

Procesos empíricos: la calidad del producto depende del equipo. Si el equipo hace lo que tiene que hacer, el producto va a tener calidad. Los equipos ágiles definen en cada proyecto qué proceso se va a seguir. Si detecta en una Retrospective detecta un cambio necesario en el proyecto, lo cambia.

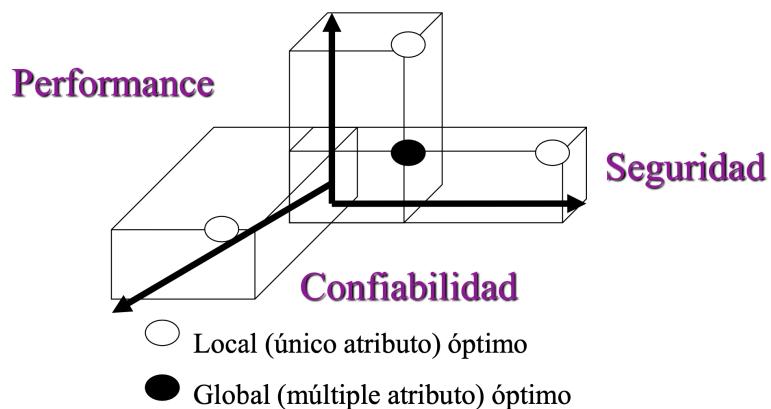
Modelos de Calidad de Producto

ISO/IEC 25000: Modelo

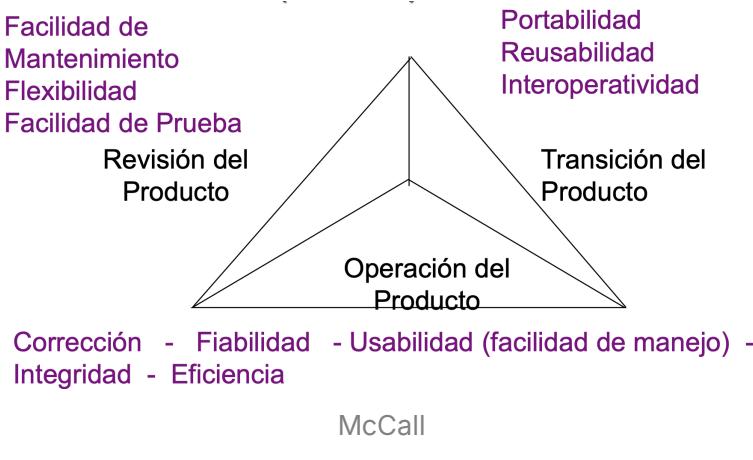
- Calidad en Uso (ISO 25010: 2011)
- Calidad de producto (ISO 25010:2011)
- Calidad de Datos (ISO 25012:2008)



Modelo de Calidad de Producto (En uso) – ISO 25010

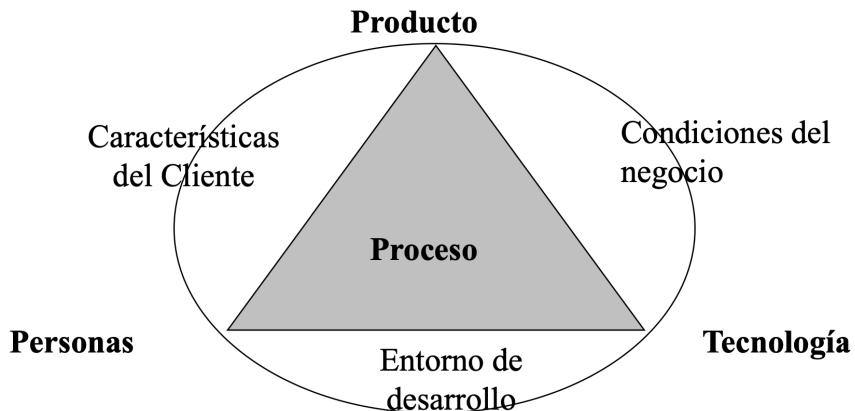


Modelo de Barbacci / SEI

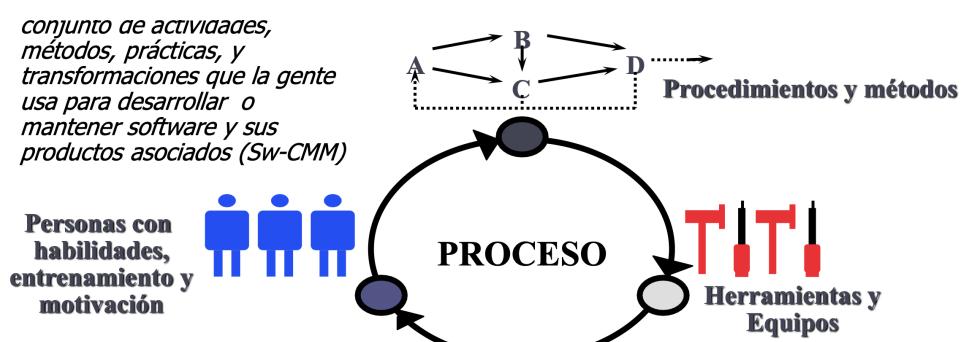


Calidad y Proceso de Desarrollo

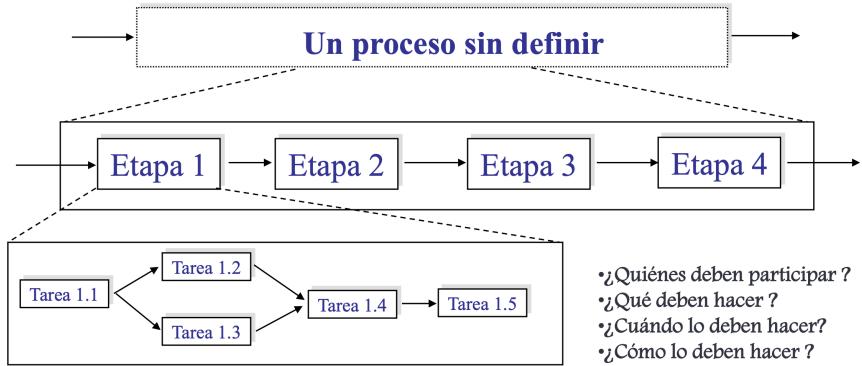
El proceso es el único factor *controlable* al mejorar la calidad del software y su rendimiento como organización.



Proceso de Software: Un conjunto de actividades, métodos, prácticas, y transformaciones que la gente usa para desarrollar o mantener software y sus productos asociados (Sw-CMM)



Un proceso está conformado por personas, procedimientos, y herramientas.



Definir un proceso consiste en “abrir la caja negra” de un proceso sin definir, definir etapas y tareas, quiénes participan, qué se debe hacer, cuándo y cómo.

Proceso para construir software

Los procesos “planificación y seguimiento de proyectos”, “administración de configuraciones” y “aseguramiento de la calidad” son procesos de gestión y soporte estudiados en esta materia. Son transversales a todo el proceso de desarrollo de software



Proceso de construcción de software, incluyendo procesos transversales.

Aseguramiento de Calidad de Software

Se trata de insertar, en cada una de las actividades, acciones tendientes a detectar lo más tempranamente posible oportunidades de mejora sobre el producto y proceso.

- Concerniente con asegurar que se alcancen los niveles requeridos de calidad para el producto de software.
- Implica la definición de estándares y procesos de calidad apropiados y asegurar que los mismos sean respetados.

- Debería ayudar a desarrollar una “cultura de calidad” donde la calidad es vista como una responsabilidad de todos y cada uno.

Reporte del Grupo de Aseguramiento de Calidad (GAC)

- No debería reportar al Gerente de Proyectos.
- No debería haber más de una posición entre la Gerencia de Primer Nivel y el GAC.
- Cuando sea posible, el GAC debería reportar alguien realmente interesado en la calidad del software
- La administración de calidad debe estar separada de la administración de proyectos para asegurar independencia

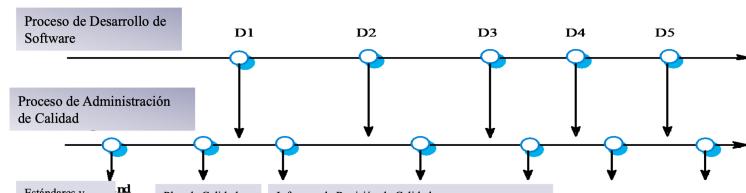
Actividades de la Administración de Calidad de Software

- Aseguramiento de Calidad
 - Establecer estándares y procedimientos organizacionales de calidad.
- Planificación de Calidad
 - Selecciona los procedimientos y estándares aplicables para un proyecto en particular y los modifica si fuera necesario.
- Control de Calidad
 - Asegura que los procedimientos y estándares son respectados por el equipo de desarrollo de software.

Funciones del Aseguramiento de Calidad de Software

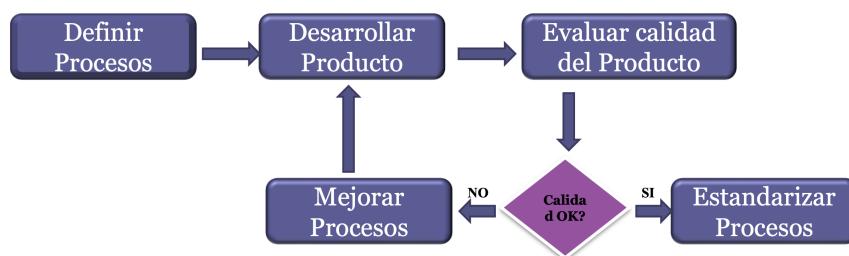
- Prácticas de Aseguramiento de Calidad
 - Desarrollo de herramientas adecuadas, técnicas, métodos y estándares que estén disponibles para realizar las revisiones de Aseguramiento de Calidad.
- Evaluación de la planificación del Proyecto de Software
- Evaluación de Requerimientos
- Evaluación del Proceso de Diseño
- Evaluación de las prácticas de programación

- Evaluación del proceso de integración y prueba de software
- Evaluación de los procesos de planificación y control de proyectos
- Adaptación de los procedimientos de Aseguramiento de calidad para cada proyecto.



La Administración de Calidad y el Desarrollo de Software

Procesos basados en calidad. La estandarización de procesos en general se da más en procesos definidos, porque en procesos empíricos se considera que la experiencia y el proceso le sirve sólo a ese equipo y no es extrapolable.



Calidad de Procesos en la Práctica

- Definir procesos estándares tales como:
 - Cómo deberían conducirse revisiones
 - Cómo debería realizarse la administración de configuración, etc.
- Monitorear el proceso de desarrollo para asegurar que los estándares sean respetados.
- Reportar en el proceso a la Administración de Proyectos y al responsable del software.
- No use prácticas inapropiadas simplemente porque se han establecido los estándares.

Estándares y Aseguramiento de Calidad

- Los estándares son la clave para la administración de calidad efectiva.
- Pueden ser estándares internacionales, nacionales, organizacionales o de proyecto.
- Estándares de Producto definen las características que todos componentes deberían exhibir, ej. estilos de programación común.
- Estándares de Procesos definen cómo deberían ser implementados los procesos de software.

Planificación de la Calidad

- Un plan de calidad define los productos de calidad deseados y como serán evaluados , y define los atributos de calidad más significativos.
- El plan de calidad debería definir el proceso de evaluación de la calidad.
- Define cuales estándares organizacionales deberían ser aplicados , como así también si es necesario utilizar nuevos estándares.

Control de Calidad

- Este implica el control del proceso de desarrollo para asegurar que se siguen los estándares y procedimientos .
- Existen dos enfoques para el control de calidad:
 - Revisiones de Calidad;
 - Evaluaciones de Software Automáticas y mediciones.

Revisiones de Calidad

- Este es el principal método de validación de la calidad de un proceso o un producto.
- Un grupo examina parte de un proceso o producto y su documentación para encontrar potenciales problemas.
- Existen diferentes tipos de revisiones con diferentes objetivos
 - Inspecciones para remoción de defectos (producto);
 - Revisiones para evaluación de progreso (producto y proceso);
 - Revisiones de Calidad (producto y estándares).

Modelos de Mejora

Modelo de mejora IDEAL: Initiating, Diagnosing, Establishing, Acting, Leveraging.

Se necesita un esponsoreo, se empieza por un diagnóstico, trabajamos en la etapa de planes de acciones, los ejecutamos, probamos el proceso en algún proyecto y si el resultado es correcto, lo extrapolamos a toda la organización.

several phases.

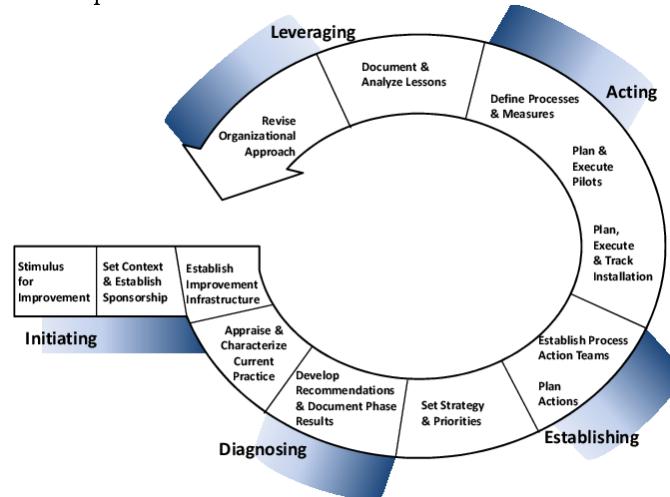


Figure 1. The IDEAL Model, following [5]

Kanban también es un framework de mejora de procesos.

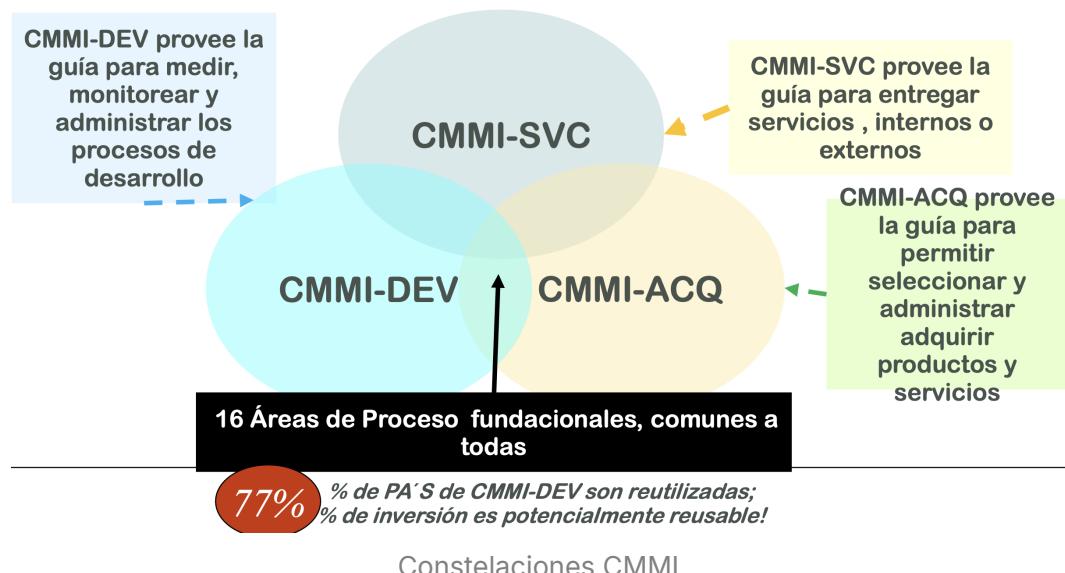
CMMI

Capability Maturity Model Integration (CMMI) es un modelo descriptivo y no prescriptivo (dice qué, no cómo hacer).

Es una guía de mejores prácticas que ayudan a definir procesos de desarrollo de software.

Si se quiere encarar una mejora de proceso, el “cómo” no lo dice el CMMI, lo dice el IDEAL. Qué cosas debería hacer mi proceso para tener cierta calidad, lo dice el CMMI. El modelo de calidad sirve para tener un proceso que mejore continuamente.

Se hace foco en el CMMI-DEV:



Existen dos tipos de representaciones: por etapas y continua.

En la representación **por etapas**, las organizaciones inmaduras tienen nivel 1, del 2 al 5 son maduras.

En la representación **continua**, existen 22 áreas de procesos (AP) que pueden ser mejoradas por separado. Así, se mide la capacidad de un proceso en particular. Pueden existir procesos en nivel 0 (no se ejecuta) y procesos en nivel 2 o cualquier otro. Si casualmente se llama a una evaluación y las AP que se quieren mejorar son las de nivel 2, se dan 2 certificados: **capacidad y madurez**.



Grupo: existencia de roles que cubren ciertas tareas. Los roles se adaptan a la cantidad de empleados, expectativas, etc. No implica un grupo de varias personas, puede ser una persona también. Lo importante es que exista alguien **responsable** de cubrir las *actividades* de cada uno de los roles o grupo.

- Una representación permite a una organización perseguir diferentes objetivos de mejora y presenta los componentes del modelo de manera diferente.
 - La representación por etapas utiliza niveles de madurez para medir la mejora de los procesos.
 - Los niveles de madurez se aplican a la madurez general de una organización.
 - Conjuntos predefinidos de áreas de proceso definen un camino de mejora para la organización.
- La representación continua utiliza niveles de capacidad para medir la mejora de los procesos.
 - Los niveles de capacidad se aplican al logro de la mejora de procesos de una organización para cada área de proceso (AP).
 - Las mejoras se caracterizan en relación a una AP individual.
- El contenido es casi idéntico en ambas representaciones.
- ¿Entonces, por qué ambas?
 - La representación de cada modelo fuente era diferente.
 - Software Capability Maturity Model (SW-CMM) - por etapas.
 - Systems Engineering Capability Maturity Model (SE-CMM) - continuo.
 - Facilidad de adopción por parte de comunidades heredadas.
 - Ambas representaciones ofrecen beneficios inherentes.

Ventajas de cada representación

Representación Continua	Representación por Etapas
Máxima flexibilidad para priorizar mejoras de procesos y alinearlas con los objetivos del negocio (requiere comprensión de las relaciones entre las áreas de proceso)	Camino predefinido y probado con datos de casos de estudio y ROI (reduce la incertidumbre)
Permite una mayor visibilidad de las mejoras dentro de las áreas de proceso Las "victorias rápidas" se pueden definir fácilmente para aumentar la aceptación	Se centra en la mejora organizacional

Representación Continua	Representación por Etapas
Aumenta el enfoque en los riesgos específicos de cada área de proceso	
La mejora de las áreas de proceso puede ocurrir a diferentes ritmos Puede requerirse menos inversión inicial	Los resultados generales se resumen en un nivel de madurez Proporciona una capacidad de evaluación comparativa familiar (normalmente utilizada para calificar a los licitantes)
Fácil actualización desde SE-CMM y SECM	Fácil actualización desde SW-CMM

Ejemplos de uso

Continua

- Perspectiva de una pequeña empresa en el piloto de AMDEC SED en Huntsville, AL
 - A medida que avanzaba el piloto, nuestro énfasis en querer adoptar CMMI cambió de un deseo inicial de "certificarse" a enfocarnos en mejorar en "partes" más pequeñas en áreas identificadas por análisis del negocio.
 - La implementación o áreas de proceso específicas sin el objetivo general de alcanzar un nivel hace que el uso del modelo sea más significativo para nuestra pequeña organización.
 - Ahora nos damos cuenta de que podemos usar CMMI en las áreas que naturalmente agregan valor a nuestra organización y calidad a nuestros productos finales mejorando las actividades donde más lo necesitamos.
- Consultores del piloto patrocinado por AMDEC SED en Huntsville, AL
 - La representación continua permite a las pequeñas empresas centrarse en las mejoras que tienen el mayor retorno para la empresa.
- En general, la representación continua es útil cuando la organización:
 - Entiende cómo CMMI puede abordar los objetivos empresariales.
 - Ha identificado mejoras específicas que necesita.
 - Tiene un presupuesto limitado para la mejora de procesos.
 - Entiende las relaciones entre las áreas de proceso.
 - Puede beneficiarse de "victorias rápidas".

Por etapas

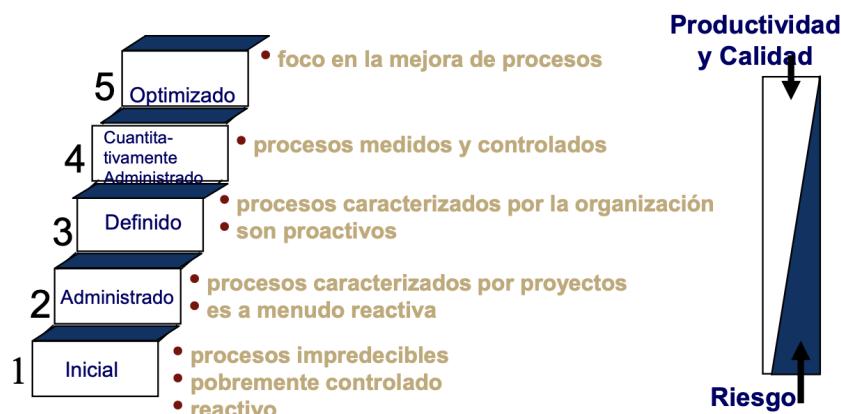
- La representación por etapas proporciona un excelente camino para las organizaciones que:
 - Están en transición del SW-CMM al CMMI.
 - Hacen negocios con organizaciones gubernamentales que requieren un nivel de madurez para sus adquisiciones.
 - No están familiarizadas con las dependencias entre las áreas de proceso.
 - Gran parte de la incertidumbre en la mejora de procesos se reduce gracias al conjunto predefinido de áreas de proceso en cada nivel de madurez.
- La equivalencia por etapas permite la comparación de resultados de ambas representaciones.

Representación Por Etapas

Se hace foco en esta representación. Sigue siendo la más elegida.

Para evaluar un determinado nivel, se deben cumplir todos los objetivos que tienen las áreas de proceso del nivel. Si un área se cae (no cumple al menos un objetivo), se cae el nivel.

Los niveles de madurez (del 2 al 5) son **acumulativos**. No se puede ser nivel 5 sin ser nivel 4, 3, 2 antes. La verificación es una actividad basada en un AP del nivel 3.



Niveles para la representación por etapas



Áreas de proceso por nivel para CMMI v1.3. El nivel 3 "Definido" se encarga de la ingeniería, tiene 11 AP. El nivel 2 es donde se enfoca esta materia (7 AP)

Para CMMI - DEV:

Nivel	Categoría			
	Administración de Proyectos	Soporte	Administración de Procesos	Ingeniería
5 Optimizado		▪ Análisis y Causal y Resolución (CAR)	▪ Administración de Performance Organizacional (OID)	
4 Cuantitativamente Administrado	▪ Administración Cuantitativa del Proyecto (QPM)		▪ Performance del Proceso Organizacional (OPP)	
3 Definido	<ul style="list-style-type: none"> ▪ Administración de Riesgos (RSKM) ▪ Administración Integrada de Proyectos (IPM) 	<ul style="list-style-type: none"> ▪ Análisis y Resolución de Decisión (DAR) 	<ul style="list-style-type: none"> ▪ Definición del Proceso Organizacional (OPD) ▪ Foco en el Proceso Organizacional (OPF) ▪ Capacitación Organizacional (OT) 	<ul style="list-style-type: none"> • Desarrollo de Requerimientos (RD) • Solución Técnica (TD) • Integración de Producto (PI) • Verificación (VER) • Validación (VAL)
2 Administrado	<ul style="list-style-type: none"> ▪ Administración de Requerimientos (REQM) ▪ Planificación de Proyectos (PP) ▪ Monitoreo y Control de Proyectos (PMC) ▪ Administración de Acuerdo con el Proveedor (SAM) 	<ul style="list-style-type: none"> ▪ Aseguramiento de calidad de Proceso y de Producto (PPQA) ▪ Administración de Configuración (CM) ▪ Medición y Análisis (MA) 		
1 Inicial	Procesos sin definir o improvisados			

Una organización Madura según CMMI

CMMI, como modelo de calidad, nos presenta niveles de madurez: del 1 al 5.

En particular, las organizaciones maduras (que van del 2 al 5) son aquellas que aplican políticas o lineamientos de calidad en el proceso de desarrollo.

También consideran la calidad del producto.

Tenemos distintos niveles, comenzando por el 2 (en organizaciones maduras), en donde se aplican lineamientos de calidad básicos, sin considerar procesos de calidad profundos. Sin embargo, en ellas sí se tiene noción de calidad (a

diferencia de las organizaciones inmaduras, donde no se aplica aseguramiento de calidad).

En el caso de las organizaciones de nivel 5, se aplican políticas y lineamientos de calidad, se documenta, se busca la mejora continua constantemente en procesos de calidad de software (y también en producto). Tienen un enfoque del aseguramiento de calidad de proceso y producto bien profundo y definido.

CMMI Ágil

Una organización ágil puede evaluarse contra CMMI, aunque ambas partes deben flexibilizarse un poco.

Referencias:

Sin *: da soporte

*: Neutral

**: desigual

- Nivel 1
 - Identificar el alcance del trabajo
 - Realizar el trabajo
- Nivel 2
 - Política Organizacional para planear y ejecutar*
 - Requerimientos, objetivos o planes
 - Recursos adecuados
 - Asignar responsabilidad y autoridad
 - Capacitar a las personas
 - Administración de Configuración para productos de trabajo elegidos
 - Identificar y participar involucrados*
 - Monitorear y controlar el plan y tomar acciones correctivas si es necesario*
 - Objetivamente monitorear adherencia a lo procesos y QA de productos y/o servicios**
 - Revisar y resolver aspectos con el nivel de administración más alto*

- Nivel 3
 - Mantener un proceso definido*
 - Medir la performance del proceso** (a agile no le interesa medir qué tan bien seguimos un proceso)
- Nivel 4
 - Establecer y mantener objetivos cuantitativos para el proceso**
 - Estabilizar la performance para uno o más subprocessos para determinar su habilidad para alcanzar logros**
- Nivel 5
 - Asegurar mejora continua para dar soporte a los objetivos
 - Identificar y corregir causa raíz de los defectos*

Hipótesis

- Tolerancia de CMMI a Ágil
 - Hay áreas de proceso que:
 - Hay soporte,
 - Otras Neutrales,
 - Otras en conflicto
 - Soporte a evaluación en un ambiente Ágil
- Tolerancia de Agile a CMMI
 - Está la puerta abierta???
 - Es posible???

Diferencias

Valores esenciales

CMMI	MÉTODOS ÁGILES
→ Medir y mejorar el proceso [Mejores Procesos ↓ Mejor Producto]	→ Respuestas a clientes → Mínima sobrecarga → Refinamiento de Requerimientos - Metáforas - Casos de negocio
→ Características de las personas - Disciplinados - Siguen reglas - Aversión al riesgo	→ Características de las personas - Comfortable - Creative - Risk Takers
→ Comunicación - Organizacional - Macro	→ Comunicación - Person to Person - Micro
→ Gestión de Conocimiento - Activos de proceso	→ Gestión de Conocimiento - Personas

Características

CMMI	MÉTODOS ÁGILES
→ Mejora Organizacionalmente -Uniformidad -Nivelación	→ Mejora en el Proyecto - Tradición Oral - Innovación
→ Capacidad/Madurez - Éxito por Predictibilidad	→ Capacidad/Madurez - Éxito por darse cuenta de oportunidades
→ Cuerpo de Conocimiento - Cruzando dimensiones - Estandarizado	→ Cuerpo de Conocimiento - Personal - Evolucionando - Temporal
→ Reglas de Atajo - Desalentadas	→ Reglas de Atajo - Alentadas

CMMI	MÉTODOS ÁGILES
→ Comités	→ Individuos
→ Confianza del Cliente - En la Infraestructura del Proceso	→ Confianza del Cliente - Sw funcionando, Participantes
→ Cargado al frente - Mover a la derecha	→ Conducido por Pruebas - Mover a la izquierda
→ Alcance de la vista [Involucrado, Producto] - Amplio - Inclusivo - Organizacional	→ Alcance de la vista [Involucrado, Producto] - Pequeño - Focalizado
→ Nivel de Discusión - Palabras - Definiciones - Duradero - Exhaustivo	→ Nivel de Discusión - Trabajo en mano

Enfoque

CMMI	MÉTODOS ÁGILES
→ Descriptivo	→ Prescriptivo
→ Cuantitativo	→ Cualitativo
- Número científicos y duros	- Conocimiento tácito
→ Universalidad	→ Situacional
→ Actividades	→ Producto
→ Estratégico	→ Táctico
→ "¿Cómo lo llamaremos?"	→ "Sólo hazlo!"
→ Gestión de Riesgos	→ Gestión de Riesgos
- Proactiva	- Reactiva

FOCO

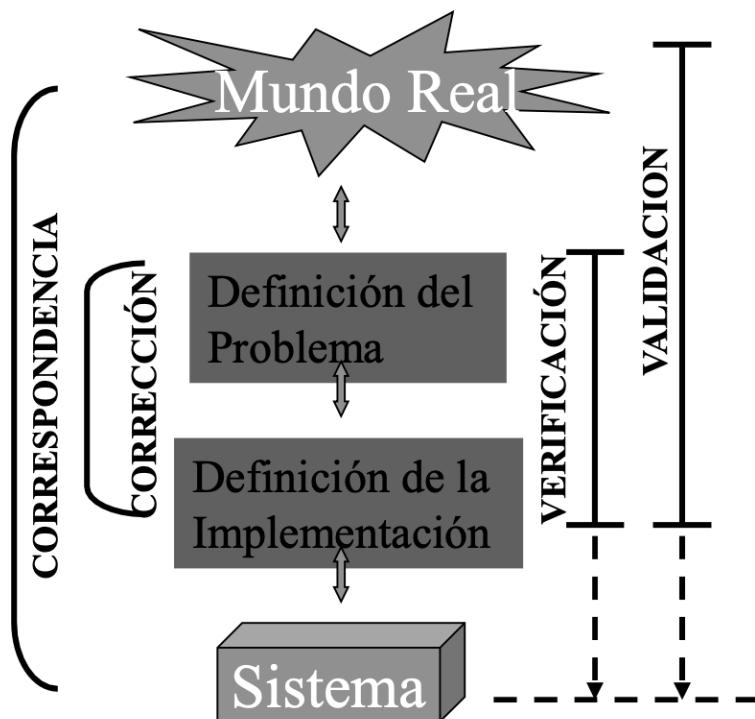
CMMI	MÉTODOS ÁGILES
→ Foco de Negocio	→ Foco de Negocio
- Interna - Reglas	- Externo - Innovación
→ Predictibilidad	→ Performance
→ Estabilidad	→ Velocidad

Similitudes

- Meta: Organizaciones de alto desempeño
- Ambas planean
- Ambas son **CMMs (Consultant Money Makers)**
- Ambas tienen reglas [Reglas = Requerimientos del proceso]
 - La violación tiene serias repercusiones
 - 'SEPG' (Grupo de proceso de ingeniería de software) & 'Política de Proceso'
- Ninguno es completo
- No nuevas ideas
 - Basadas en la experiencia
- Ninguno es aplicable a "cualquier proyecto"

Revisiones técnicas

No sólo se hacen al código.



Verificación y Validación

Es un proceso de ciclo de vida completo.

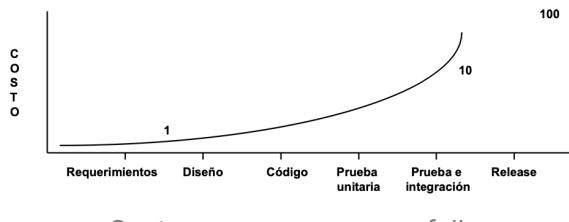
Inicia con las revisiones de los requerimientos y continúa con las revisiones del diseño, inspecciones del código hasta la prueba.

Falla: error en un producto de trabajo

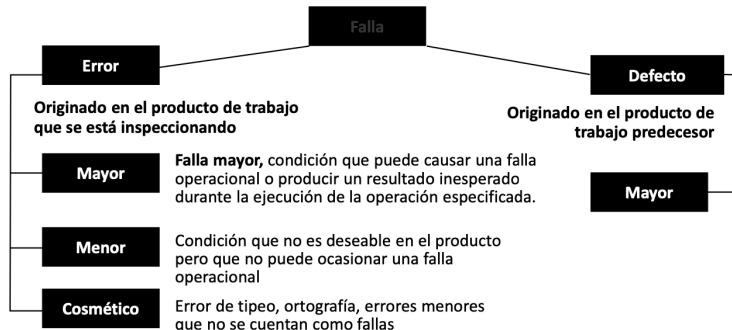
Producto de trabajo: salida de cualquier actividad correspondiente al ciclo de vida de desarrollo

Las fallas aparecen por 2 motivos:

- Ruidos de comunicación
- Limitaciones de memoria:
 - Los límites de la memoria a corto plazo: $7 +|- 2$
 - "Las fallas más persistentes están relacionadas con la complejidad inherente al producto que se desarrolla"*



Costos para reparar una falla



La falla se divide en error y defecto

Algunos ejemplos:

Fallas mayores

- En código: Error lógico, estructural u otro que pueda ocasionar una falla operacional.
- En diseño: Una expresión en el diseño que pudiera ocasionar una falla operacional si se implementara tal cual está especificado.
- En requerimientos: las necesidades del cliente, una expresión ambigua o información faltante que requerirá una investigación posterior.
- En plan de prueba o casos de prueba: Una condición que podría ocasionar que no se detectaran fallas en el programa o que la prueba no pueda llevarse a cabo o repetirse.

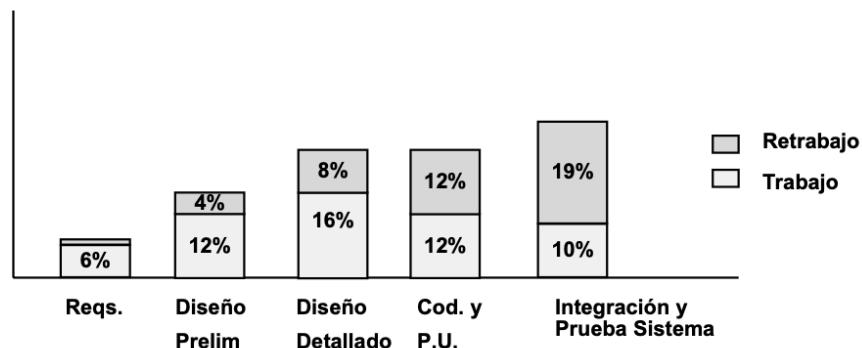
Fallas menores

- En código o diseño: Una violación a los estándares de codificación o de diseño (Ej: comentarios en el código), que no ocasionará una falla operacional pero puede reducir la claridad y causar problemas de mantenimiento.
- En requerimientos: Un requerimiento que no pueda probarse.
- En plan de prueba o casos de prueba: Información que no está clara o que pudiera causar que se requiera esfuerzo de testing innecesario debido a la

redundancia.

Notas cosméticas

- En documentación:
 - Errores de tipo, Errores ortográficos, Errores gramaticales,
 - Se necesita actualizar el documento con una plantilla más nueva (existe una versión más nueva)
 - Se necesita actualizar la historia de revisiones del documento.
- En código:
 - Se necesita actualizar los datos de copyright de un código fuente utilizado
 - Una sugerencia alternativa (Ej. Un algoritmo de búsqueda diferente)



El retrabajo equivale al 40-50% del desarrollo

Principios

- La prevención es mejor que la cura
- Evitar es más efectivo que eliminar
- La retroalimentación enseña efectivamente
- Priorizar lo rentable
- Olvidarse de la perfección, no se puede conseguir

Características de Revisiones técnicas

- Proceso de V & V estático

- Principal objetivo: Detectar defectos y corregirlos en etapas tempranas del desarrollo
- Origen: Proceso de inspecciones de Fagan (1976), basado en experiencia en HW
- Practicado en industrias de SW donde calidad y retrabajo son críticos
- Existen muchas variantes respecto a las inspecciones originales de Fagan
- Se puede inspeccionar cualquier representación legible del SW
- Se aplican en varios momentos del desarrollo
- El trabajo técnico requiere revisión, similar a cómo los lápices necesitan gomas
- Algunos errores son más fáciles de detectar por otros que por el autor original
- Motiva a realizar un mejor trabajo
- No requieren la ejecución del programa

Ventajas de las Revisiones Técnicas

- Permiten descubrir muchos errores
- Posibilidad de inspeccionar versiones incompletas
- Se pueden considerar otros atributos de calidad

Desventajas de las Revisiones Técnicas

- Dificultad para introducir inspecciones formales
- Sobrecarga inicial de costos, con ahorro posterior tras ganar experiencia
- Requieren tiempo para organización y pueden parecer ralentizar el desarrollo

Costos de las Revisiones Técnicas

- Infraestructura:
 - Entrenamiento
 - Desarrollo/ajuste de plantillas e informes
 - Desarrollo/ajuste de guías de lectura

- Implantación de programas de medición
- Herramientas de soporte
- Operacionales:
 - Tiempo individual y grupal
 - Tiempo en completar informes
- Adicionales:
 - Preparación de material, Organización de calendario, Recolección de datos
 - Tiempo dedicado a la mejora de calidad

Método	Objetivos Típicos	Atributos Típicos
Walkthroughs	Mínima Sobrecarga Capacitación de Desarrolladores Rápido retorno	Poca o ninguna preparación Proceso Informal No hay mediciones No FTR!
Inspecciones	Detectar y remover todos los defectos eficiente y efectivamente	Proceso Formal Checklists Mediciones Fase de Verificación

Tipo de documento	Revisores sugeridos
Arquitectura o Diseño de alto nivel	Arquitecto, analista de requerimientos, diseñador, líder de proyecto, testers.
Diseño detallado	Diseñador, arquitecto, programadores, testers
Planes de proyecto	Líder de proyecto, stakeholders, representante de ventas o marketing, líder técnico, representante del área de calidad,
Especificación de requerimientos	Analista de requerimientos, líder de proyecto, arquitecto, diseñador, testers, representante de ventas y/o marketing
Código fuente	Programador, diseñador, testers, analista de requerimientos
Plan de testing	Tester, programador, arquitecto, diseñador, representante del área de calidad, analista de requerimientos

Métricas Sugeridas	Fórmula
Densidad de defectos	Total de defectos encontrados / tamaño actual
Total de defectos encontrados	Defectos.Mayor + Defectos.Menor
Esfuerzo de la inspección	Esfuerzo.Planning + Esfuerzo.Reparación + Esfuerzo.Reunión + Esfuerzo.Retrabajo
Esfuerzo por defecto	Esfuerzo.Inspeccion / Total de def encontrados
Porcentaje de reinspecciones	Cantidad Reinspecciones / Cantidad Inspecciones
Defectos Corregidos sobre Total de Defectos.	Esfuerzo.Inspeccion / tamaño actual

Inspección

- Actividad de garantía de calidad de software
- Objetivos:
 - Descubrir errores
 - Verificar cumplimiento de requisitos
 - Garantizar representación según estándares
 - Lograr desarrollo uniforme
 - Hacer proyectos más manejables
- Se realiza mediante reunión planificada
- Las inspecciones son:
 - Forma económica y eficaz de encontrar fallas
 - Medio para proporcionar métricas al proyecto
 - Manera de fomentar intercambio de conocimientos
 - Método para promover trabajo en equipo
 - Técnica probada para mejorar calidad del producto
- Las inspecciones no son:
 - Para encontrar soluciones a fallas
 - Para obtener aprobación de producto de trabajo
 - Para evaluar desempeño individual del personal

Roles

Autor

- Creador o encargado de mantener el producto que va a ser inspeccionado.
- Inicia el proceso asignando un moderador y designa junto al moderador el resto de los roles.
- Entrega el producto a ser inspeccionado al moderador.
- Reporta el tiempo de retrabajo y el número total de defectos al moderador.

Moderador

- Planifica y lidera la revisión.
- Trabaja junto al autor para seleccionar el resto de los roles.
- Entrega el producto a inspeccionar a los inspectores con tiempo (48 horas) antes de la reunión.
- Coordina la reunión asegurándose que no hay conductas inapropiadas.
- Hace seguimiento de los defectos reportados.

Lector

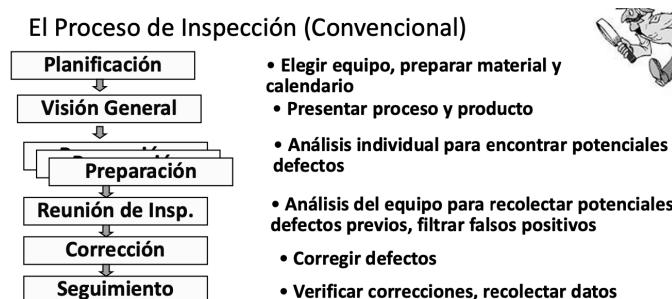
- Lee el producto a ser inspeccionado.

Anotador

- Registra los hallazgos de la revisión.

Inspector

- Examina el producto antes de la reunión para encontrar defectos.
- Registra sus tiempos de preparación.





Duración de una reunión de inspección: 2 horas máximo

Operación	Código	Documentos
Planificación	15 minutos	30 minutos
Vista previa	500 LOC/h	500 líneas de texto/h
Preparación	100 LOC/h	140 líneas de texto/h
Inspección	125 LOC/h	140 líneas de texto/h
Mejora del proceso	30 minutos	45 minutos
Tamaño máximo por inspección	250 LOC	280 líneas de texto

Recorrida / Walkthrough

Técnica de análisis estático en la que un diseñador o programador dirige miembros del equipo de desarrollo y otras partes interesadas a través de un producto de software y los participantes formulan preguntas y realizan comentarios acerca de posibles errores, violación de estándares de desarrollo y otros problemas.

Resumen

Mejorar las pruebas

- No sirve para remover errores en etapas tempranas
- Caro

Recorridas - Walkthroughs

- Buenos resultados, pero se toman pocas métricas
- No hay control del proceso

Inspecciones

- Mejores resultados, proceso controlado

- Métricas útiles a lo largo de todo el ciclo de vida del desarrollo

Auditorías de Software

Objetivos de SQA

- Realizar controles apropiados del software y el proceso de desarrollo.
- Asegurar el cumplimiento de los estándares y procedimientos para el software y el proceso.
- Asegurar que los defectos en el producto, proceso o estándares son informados a la gerencia para que puedan ser solucionados.

¿Por qué auditar?

- Porque se da una opinión objetiva e independiente
- Porque permite identificar áreas de insatisfacción potencial del cliente
- Porque nos permite asegurar al cliente que estamos cumpliendo con nuestras expectativas
- Porque permite identificar oportunidades de mejora.

Auditoría de Calidad de Software

"Evaluación independiente de los productos o procesos de software para asegurar el cumplimiento con estándares, lineamientos, especificaciones y procedimientos, basada en un criterio objetivo incluyendo documentación que especifique:

- La forma o contenido de los productos a ser desarrollados
- El proceso por el cual los productos son desarrollados
- Cómo debería medirse el cumplimiento con estándares o lineamientos."

Referencia: IEEE Std 1028-1988

Beneficios de las auditorías de calidad de software

- Evaluar el cumplimiento del proceso de desarrollo
- Determinar la implementación efectiva de:
 - El proceso de desarrollo organizacional

- El proceso de desarrollo del proyecto
- Las actividades de soporte
- Dar visibilidad a la gerencia sobre los procesos de trabajo

Resultado: Mejores productos conllevan a clientes satisfechos y crecimiento del negocio

Tipos de auditorías de calidad de software

- Auditoría de Proyecto: Valida el cumplimiento del proceso de desarrollo
- Auditoría de Configuración Funcional: Valida que el producto cumpla con sus requerimientos
- Auditoría de Configuración Física: Valida que el ítem de configuración tal como está construido cumpla con la documentación técnica que lo describe

Auditorías de Proyecto

- Se llevan a cabo de acuerdo a lo establecido en el PACS (Plan de Aseguramiento de Calidad de Software)
- El PACS debería indicar la persona responsable de realizar estas auditorías
- Las inspecciones de software y las revisiones de la documentación de diseño y prueba deberían incluirse en esta auditoría

Objetivos de la Auditoría de Proyecto

- Verificar objetivamente la consistencia del producto a lo largo del proceso de desarrollo, determinando que:
 - Las interfaces de hardware y software sean consistentes con los requerimientos de diseño en la ERS
 - Los requerimientos funcionales de la ERS se validan en el Plan De Verificación y Validación de Software
 - El diseño del producto, a medida que DDS evoluciona, satisface los requerimientos funcionales de la ERS
 - El código es consistente con el DDS

Auditoría de Configuración Funcional

- Utilizan la matriz de trazabilidad, requieren una línea base y no pueden detectar requerimientos no desarrollados.
- Compara el software construido con los requerimientos especificados en la ERS
- Asegura que el código implementa sólo y completamente los requerimientos y capacidades funcionales descritos en la ERS
- El responsable de QA debe validar si la matriz de rastreabilidad (o trazabilidad) está actualizada

Auditoría de Configuración Física

- Compara el código con la documentación de soporte
- Asegura que la documentación a entregar es consistente y describe correctamente el código desarrollado
- El PACS debería indicar la persona responsable de realizar la auditoría física
- El software podrá entregarse sólo cuando se hayan arreglado las desviaciones encontradas

Roles

Gerente de SQA:

- Prepara el plan de auditorías
- Calcula el costo de las auditorías
- Asigna los recursos
- Responsable de resolver las no-conformidades

Auditor:

- Acuerda la fecha de la auditoría
- Comunica el alcance de la auditoría
- Recolecta y analiza la evidencia objetiva relevante y suficiente
- Realiza la auditoría
- Prepara el reporte
- Realiza el seguimiento de los planes de acción acordados

- Debe ser independiente, fuera del proyecto que se está auditando.

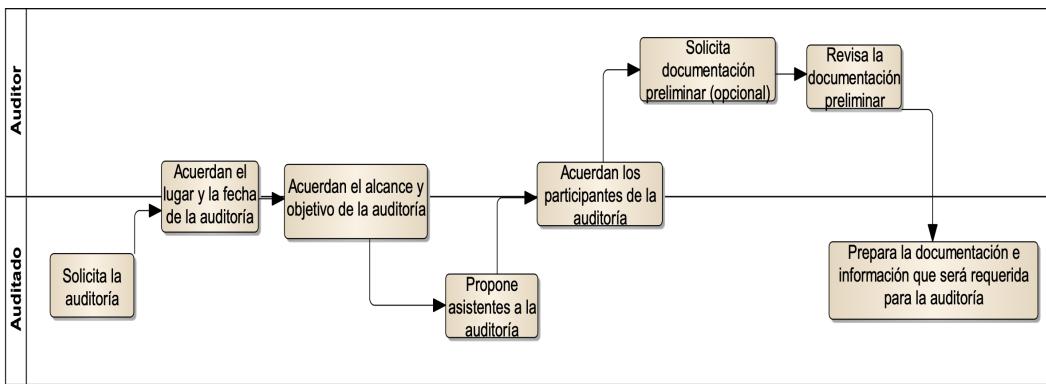
Auditado:

- Acuerda la fecha de la auditoría
- Suele ser el líder de proyecto
- Participa de la auditoría
- Proporciona evidencia al auditor
- Contesta al reporte de auditoría
- Propone el plan de acción para deficiencias citadas
- Comunica el cumplimiento del plan de acción

Proceso de Auditoría

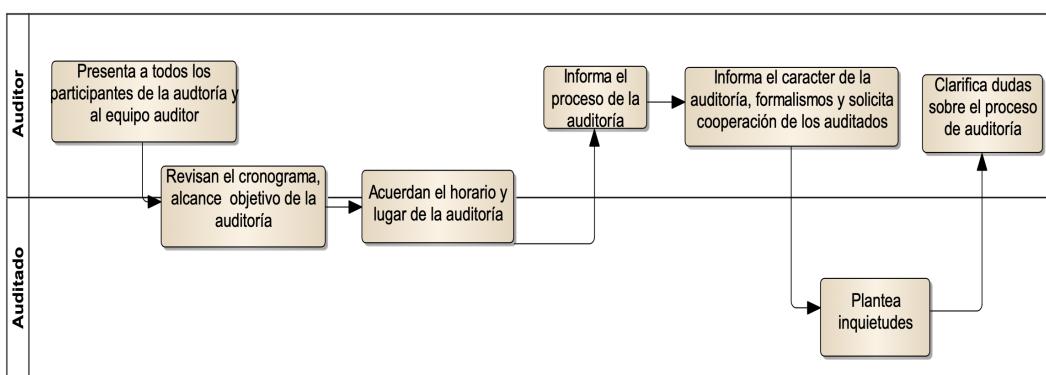


1. Preparación y planificación

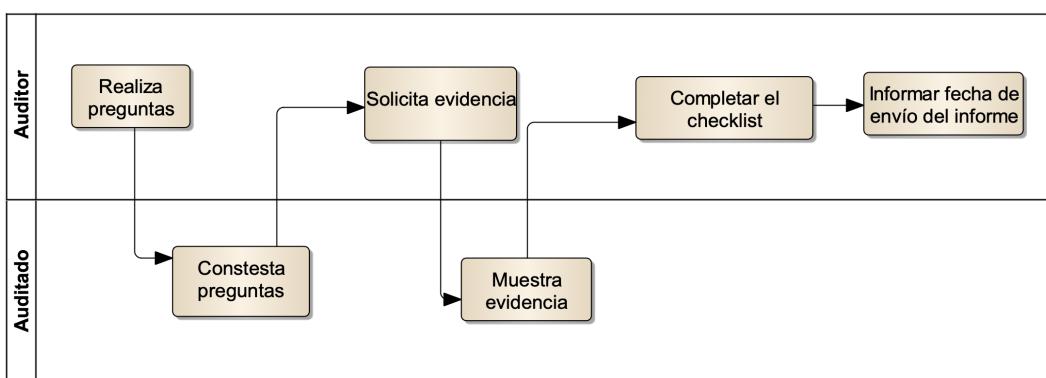


Planificación - Responsabilidades

2. Ejecución



Ejecución - Reunión de Apertura. Responsabilidades

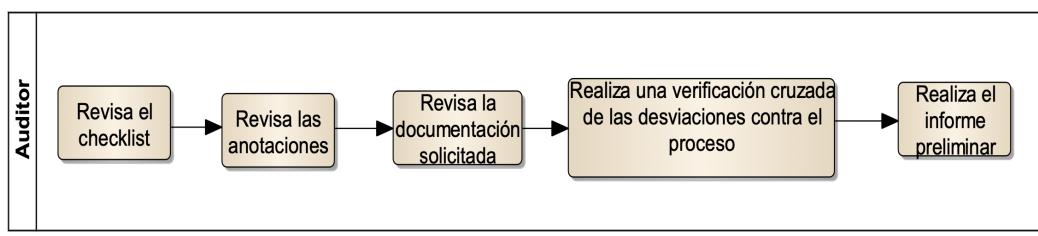


Ejecución propiamente dicha - Responsabilidades

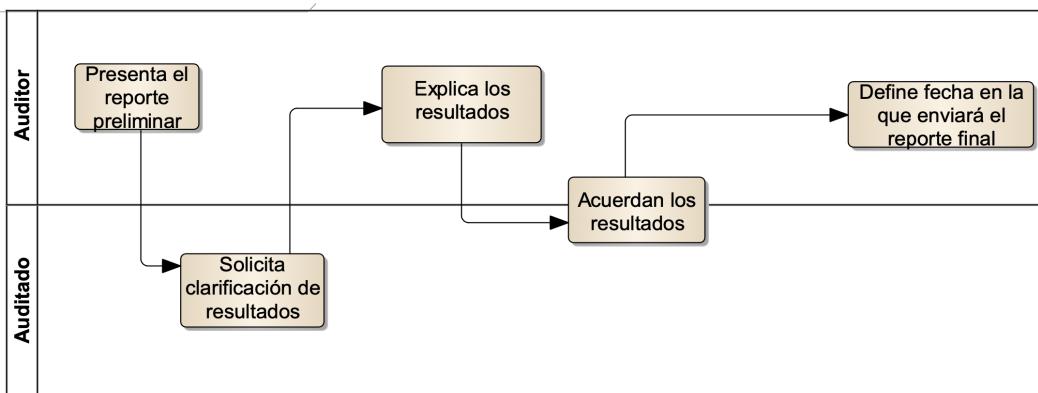
3. Análisis y reporte del resultado

Esta fase está compuesta por las siguientes actividades:

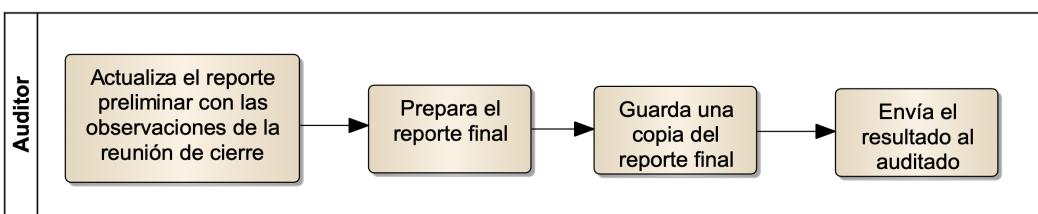
- Evaluación de los resultados
- Reunión de cierre
- Entrega del reporte final



Evaluación de Resultados

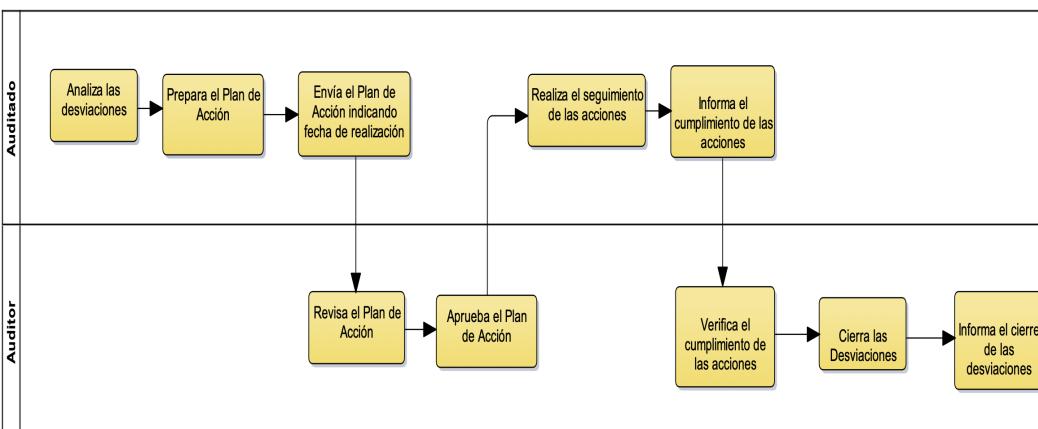


Reunión de Cierre



Entrega del reporte final

4. Seguimiento



Checklist Auditorías

Contenido general del checklist de auditoría

- Fecha de la auditoría
- Lista de auditados (identificando el rol)
- Nombre del auditor
- Nombre del proyecto
- Fase actual del proyecto (si aplica)
- Objetivo y alcance de la auditoría
- Lista de preguntas

Ejemplos de Preguntas - Planificación de Proyectos

- ¿Existe un plan de proyecto?
- ¿Está actualizado el plan de proyecto?
- ¿Existe un responsable para cada actividad?
- ¿Se han asignado recursos para las actividades de soporte?
- ¿Están disponibles los planes para todos los involucrados?

Responsabilidades del auditor

- Asegurarse que los planes estén basados en los requerimientos
- Verificar que las actividades planificadas se hayan llevado a cabo
- Confirmar que todos los involucrados se han comprometido con la última versión de los planes
- Comprobar que los cambios a los planes se hayan aprobado por todos los involucrados
- Verificar que la decisión de esos cambios se haya documentado oportunamente
- Asegurar que se han identificado y comunicado los riesgos del proyecto

Ejemplos de Preguntas - Fase de Requerimientos

- ¿Existe un documento de especificación de requerimientos?
- ¿Se han identificado únicamente los requerimientos?
- ¿Están descriptos cada uno de los requerimientos?

Responsabilidades del auditor

- Verificar que se han revisado y aprobado los requerimientos por parte de todos los involucrados
- Asegurar que los cambios a los requerimientos han seguido el correspondiente proceso de cambios
- Comprobar que se han revisado y actualizado los planes de proyecto en caso de cambios en los requerimientos

Rol del auditor durante la auditoría

- Escuchar y no interrumpir
- Observar el lenguaje corporal del auditado
- Manejar el propio lenguaje corporal
- Tomar notas
- Preguntar
- Repetir lo que ha escuchado para asegurarse que ha comprendido

Técnica de cuestionario

- Comenzar con preguntas de final abierto (quién, cuándo, cómo, qué, dónde)
- Realizar preguntas cortas y puntuales
- Finalizar con preguntas de final cerrado para clarificar conceptos

Reacciones comunes de los auditados

- Querer impresionar al auditor
- Estar ansioso o tensionado
- Sentir como si estuviese siendo examinado
- Utilizar la auditoría para quejarse acerca de la empresa
- Brindar demasiada información, diciendo cosas que el auditor no debería saber
- Estar enojado o nervioso

Técnicas y herramientas

- Checklist
- Muestreo
- Revisión de registros
- Herramientas automatizadas

Análisis y Reporte de Resultados

Contenidos básicos

- Identificación de la auditoría
- Fecha de la auditoría
- Auditor
- Auditados
- Nombre del proyecto auditado
- Fase actual del proyecto
- Lista de resultados
- Comentarios
- Solicitud de planes de acción

Tipos de resultados:

- Buenas prácticas: práctica, procedimiento o instrucción que se ha desarrollado mucho mejor de lo esperado
- Desviaciones: requieren un plan de acción por parte del auditado
- Observaciones: sobre condiciones que deberían mejorarse pero no requieren un plan de acción

Buenas Prácticas

Se deben reservar para cuando el auditado:

- Ha establecido un sistema efectivo
- Ha desarrollado un alto grado de conocimiento y cooperación interna
- Ha adoptado una práctica superior a cualquier otra que se haya visto

Desviaciones

- Cualquier desviación que resulta en la disconformidad de un producto respecto de sus requerimientos
- Falta de control para satisfacer los requerimientos
- Cualquier desviación al proceso definido o a los requerimientos documentados que cause incertidumbre sobre la calidad del producto, las prácticas o las actividades

Observaciones

- Opinión acerca de una condición incumplida
- Práctica que debe mejorarse
- Condición que puede resultar en una futura desviación

Métricas de auditoría

Cada organización deberá establecer las métricas más apropiadas. Algunos ejemplos serían:

- Esfuerzo por auditoría
- Cantidad de desviaciones
- Duración de auditoría
- Cantidad de desviaciones clasificadas por PA de CMMI

Puntos Clave

- Las auditorías al proceso de desarrollo de software son tres:
 - Auditoría de Proyecto.
 - Auditoría de Configuración Física.
 - Auditoría de Configuración Funcional.
- Las auditorías implican esfuerzo y costo para los proyectos, sin embargo sus beneficios son superiores.
- Son un instrumento para el Aseguramiento de Calidad en el Software.