- **Calidad del Producto: Planificación de pruebas para el software- Niveles y tipos de pruebas para el software.**

En todo proyecto de software hay un conflicto inherente de intereses que ocurre conforme comienzan las pruebas. Esto es debido a que existe un intento por "romper" el código que se construyó, de modo que si las planifica quien lo construyó, va a diseñar y ejecutar pruebas que demuestren que el software funciona, en lugar de descubrir errores, que es justamente lo que se quiere.

En base a esto, se deben planificar para que puedan conocerse defectos porque la existencia de defectos en el software es inevitable. Por otro lado, el testing exhaustivo es imposible por lo que se debe dejar sentado en algún lado hasta dónde probar, además es costoso y consume tiempo. Es por esto que se deben identificar y detallar las pruebas más importantes, que tengan una mayor cobertura y sean efectivas.

La planificación de las pruebas es la actividad de verificar que se entienden las metas y objetivos del cliente, las partes interesadas, el proyecto y los riesgos de las pruebas que se pretende abordar. Es por esto que se puede empezar a planear sin necesidad de tener líneas de código, porque las principales pruebas se respaldan en los requerimientos. Además, sirve para dar visibilidad de manera temprana al equipo de cómo se va a probar el producto y disminuir los costos de correcciones de defectos.

Esto incluye decidir sobre los recursos, el tiempo y el presupuesto para las pruebas, así como establecer un calendario de pruebas. El plan de pruebas de aceptación debe incluir también la cobertura requerida de los requerimientos y el orden en que se prueban las características del sistema. Tiene que definir riesgos al proceso de prueba, como caídas del sistema y rendimiento inadecuado, y resolver cómo mitigar dichos riesgos.

Para validar cada objetivo del negocio se pueden utilizar distintos niveles y tipos de pruebas para el software.

NIVELES DE PRUEBA

- **Pruebas unitarias**: es el nivel más pequeño que encuentra errores, por ejemplo, métodos o clases de objetos. Se hacen en el ambiente de desarrollo por los developers.
- **Pruebas de integración**: juntos los componentes si funcionan bien, por eso se le dice pruebas de interfaces porque hay que ver cómo unirlos en términos de componentes. En PUD: actividad del workflow de testing/prueba. Con el uso de las prácticas de integración continua se automatizan. Se enfocan en mostrar que la interfaz de componente se comporta según su especificación, o bien se debe corregir. Deben tenerse en cuenta condiciones inusuales.
- **Pruebas de sistema**: o de versión, porque si estamos hablando de un ciclo de vida iterativo e incremental, en cada incremento no se trata del producto final. Lo que se testea es una versión del producto que no está 100% terminada, a menos que se trate de un ciclo de vida en cascada. Demuestran que los componentes son compatibles, interactúan correctamente y transfieren los datos correctos en el momento adecuado a través de sus interfaces.
- **Pruebas de aceptación de usuario**: lo hace el usuario, tiene que estar presente. En el workflow de despliegue se ubican las pruebas de aceptación. En SCRUM se hace en la review.
  Son esenciales aún cuando se hayan realizado las otras → la influencia del entorno de trabajo del usuario tiene gran efecto sobre la fiabilidad, rendimiento, uso y robustez de un sistema. Implican que el cliente decida si acepta o no el sistema. Meta: decidir si el software es suficientemente adecuado para desplegarse y utilizarse, si les gusta a los clientes y si hace lo que necesitan. Objetivo no es que el cliente encuentre errores, sino que vea la funcionalidad que se hizo y la valide contra la necesidad.

En cuanto a los tipos de prueba:

Prueba de humo/Smoke Test → primera corrida de los tests de sistema que provee cierto aseguramiento de que el software que está siendo probado no provoca una falla catastrófica. Es decir, se verifica que las funcionalidades básicas de una aplicación o sistema funcionen correctamente antes de realizar pruebas más exhaustivas. Puede considerarse como una estrategia de integración constante, dado que el software se reconstruye con los nuevos componentes y se prueba cada día. Se enfoca más en comprobar pequeños cambios y funciones en lugar de la estabilidad general del software.

Prueba de sanidad/Sanity Test → se realiza después de una ronda de pruebas más exhaustivas para verificar si se han corregido los errores o problemas críticos identificados previamente. El objetivo principal es asegurarse de que las correcciones o características realizadas no hayan introducido nuevos problemas y que las funcionalidades clave del software sigan funcionando correctamente después de las correcciones.

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. Responden a ¿Qué hace el sistema? Controla que el software se comporte de la misma manera que lo especificado en la documentación, cumpliendo con las funcionalidades y características definidas. Se basa en los requerimientos funcionales y el proceso de negocio:

1. Basado en requerimientos: cuando se prueban requisitos específicos, apunta a probar una funcionalidad sola (utilizan a los requisitos definidos en una ERS o los acuerdos que contienen las pruebas de usuario y los criterios de aceptación de una US para realizar las pruebas.)

2. Basado en los procesos de negocio: cuando se prueba un proceso de negocio completo, es decir, se prueba todo el proceso. Por ejemplo, en una venta se prueba la búsqueda del artículo, la selección y facturación del mismo.

Testing No Funcional → se basa en cómo trabaja el sistema haciendo foco en los requerimientos no funcionales. Responden al ¿**cómo** funciona el sistema? Las pruebas se basan en aspectos no relacionados directamente con las funciones específicas de un software. Acá se ve reflejada directamente la necesidad de que los ambientes de prueba sean lo más parecidos posible al entorno de producción. Incluye varios tipos de prueba según el aspecto que trate:

→ Performance: Se ve el tiempo de respuesta (escenario esperado respecto a los tiempos de respuesta), la concurrencia, cuellos de botella y puntos de fallo. Deben pasar esta prueba sí o sí.

→ Carga: no solo mira performance, mira el comportamiento de los dispositivos de hardware (procesadores, discos, etc.) y de las comunicaciones. Importante para la escalabilidad.

→ Estrés: queremos forzar al sistema para que falle, se lo somete a condiciones más allá de las normales. Se ve el tiempo de recuperación y la robustez del sistema. Buscar identificar el límite en el que el software deja de funcionar correctamente y qué ocurre.

→ Mantenimiento: para ver si el producto está en condiciones de evolucionar, se controla que haya documentación, manual de configuración, etc. Se observa la facilidad que existe para corregir un defecto.

→ Usabilidad: que sea cómodo para el usuario.

→ Fiabilidad: probamos que podemos depender del sistema. Resultados que se obtienen, seguridad física del software. Buscan vulnerabilidades.

→ Portabilidad: comprueban la flexibilidad y facilidad con que se puede transferir el software desde su entorno actual.

- **Técnicas y herramientas para probar software.**

Por el hecho de que el testing exhaustivo es imposible, y que el tiempo y presupuesto es limitado, surge la necesidad de pasar por la mayor cantidad de funcionalidades con la menor cantidad de pruebas. Por eso surgen formas de diseñar casos de prueba con el menor esfuerzo para identificarlas.

CAJA NEGRA

No se tiene el código para verlo, no se dispone de la estructura interna de la implementación, sino que se hace directamente una entrada versus salida. Es decir, quien las define especifica cuáles son las entradas y el resultado esperado según las mismas. Justamente lo que lo caracteriza es que no se sabe específicamente cómo el sistema llega a esa salida.

Proceso consiste en comparar los resultados obtenidos con los resultados esperados y en base a eso encontrar defectos.

Se clasifican en:

- **Basados en especificaciones**
    - Partición de equivalencias: analiza las diferentes condiciones externas que van a estar involucradas en el desarrollo de la funcionalidad: entradas y salidas. Ejemplo: variables, campo de texto, combo de selección, coordenadas, medición de un sensor de temperatura. Salida: listado, luz, emisión de un mensaje. Dividir subconjuntos que producen resultado equivalente → particiones de equivalencias.
      Condición externa: todo tipo de entradas que podemos hacer en un software.

    PARTICIÓN DE EQUIVALENCIA: subconjunto de valores que puede tomar una condición externa para el cual, si tomo cualquier miembro, el resultado va a ser equivalente en cuanto a lo que se quiere lograr.

    - Análisis de valores límites: variante de la partición de equivalencias, en vez de seleccionar cualquier elemento como representativo de una clase de equivalencia, se seleccionan los bordes de una clase.

    Idea → mayor cantidad de los defectos se encuentran en los bordes de los intervalos.

    En vez de tomar cualquier valor, seleccionar los valores del límite de la clase de equivalencia.

- **Basados en la experiencia (y el conocimiento)**
    - Adivinanza de defectos: basado en la intuición y experiencia para identificar pruebas que probablemente expongan defectos del software que se está probando. Se elabora una lista de situaciones propensas a error y se realizan las pruebas a partir de la misma.
    - Testing exploratorio: ver de qué se trata el producto y ver qué me puedo encontrar. Mientras se va probando el software, se va aprendiendo a manejar el sistema y junto con su experiencia genera nuevas pruebas a ejecutar.

CAJA BLANCA

Se dispone de los detalles de implementación, se dispone del código o bien un pseudocódigo o diagrama de flujo, y en base a eso se diseñan los casos de prueba. Se basan en el análisis de la estructura interna del software o un componente del mismo.

Tiene dos cuestiones a tener en cuenta: el número de caminos lógicos únicos puede ser sumamente grande tomando muchísimo tiempo de probarlos y además no detecta caminos faltantes ni determina errores de datos sensibles.

Se puede garantizar el testing coverage dado que hay diferentes coberturas, es decir, formas de poder recorrer distintos caminos que el código provee para desarrollar una funcionalidad. Entonces en base a esto, el objetivo es encontrar la cantidad mínima de casos de prueba para garantizar cierta cobertura, aunque no implica que no se puedan desarrollar más.

- COBERTURA DE ENUNCIADOS O CAMINOS BÁSICOS: garantiza que se va a pasar al menos una vez por todos los caminos independientes de la funcionalidad. Se da uso de la complejidad ciclomática y en base a esta se evalúan los riesgos y se determinan los casos de prueba.
- COBERTURA DE SENTENCIAS: garantiza que se va a pasar al menos una vez por cada sentencia de la funcionalidad.
- COBERTURA DE DECISIÓN: permite cubrir los dos caminos, tanto para true como para false, de todas las decisiones (dadas por un if o un switch case) de una funcionalidad.
- COBERTURA DE CONDICIÓN: garantiza que se cubra al menos una vez el resultado V y F de cada una de las condiciones de la funcionalidad, es decir, ver la decisión en detalle.
- COBERTURA DE DECISIÓN/CONDICIÓN: se tienen en cuenta las dos anteriores.
- COBERTURA MÚLTIPLE: es la más exhaustiva y de donde pueden generarse la mayor cantidad de casos de prueba debido a que busca valuar el combinatorio de todas las condiciones en todos sus valores posibles.

- **Diferentes tipos de Auditorias: Auditorías de Proyecto y Auditorías al Grupo de Calidad.**

Definición: actividad incluida dentro de las disciplinas de soporte e instrumento para el aseguramiento de calidad en el software que incluye evaluaciones independientes 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:

  - Forma o contenido de los productos a ser desarrollados
  - Proceso por el cual los productos van a ser desarrollados
  - Cómo debería medirse el cumplimiento con estándares o lineamientos

Marco de desarrollo de software → 3 tipos de auditorías:

- **AUDITORÍA DE PROYECTO**: se realizan para validar si el proyecto se ejecutó con el proceso que se dijo que se iba a ejecutar. Apunta a ver el nivel de cumplimiento del proceso que se comprometió a utilizar.

Se llevan a cabo de acuerdo con lo establecido en el *Planeamiento de Aseguramiento de Calidad de Software*, donde se debería indicar además la **persona responsable de realizar la auditoría.** Por ejemplo, se podrían incluir inspecciones de software y revisiones de documentación o bien la realización de pruebas.

- **AUDITORÍA DE CONFIGURACIÓN FUNCIONAL**: valida que el producto cumpla con los requerimientos.

La auditoría funcional compara el software que se ha construido (incluyendo sus formas ejecutables y su documentación disponible) con los requerimientos de software especificados en la ERS. El propósito de la auditoría funcional es asegurar que el código implementa sólo y completamente los requerimientos especificados en la ERS.

- **AUDITORÍA DE CONFIGURACIÓN FÍSICA**: valida que los ítems de configuración tal como está construido cumpla con la documentación técnica que los describen.

  Asegura la trazabilidad y satisfacción de los requerimientos, además de que la documentación que está y que describe el producto es consistente con el código desarrollado. Es muy importante en casos donde el ciclo de vida del producto trasciende el del proyecto dado que se debe dejar sentado una documentación correcta del mismo.

- **Proceso de Auditorías: Responsabilidades. Preparación y ejecución. Reporte y seguimiento.**

Roles

- **Auditado**: alguien del equipo, en general el líder de proyecto y quien propone la fecha de la auditoría. Comunica el cumplimiento del plan de acción, proporcionando evidencia y contestando las dudas al auditor.
- **Auditor**: 1 persona o 2. Tiene que ser de fuera del proyecto que se está auditando para que sea una revisión objetiva e independiente. Recolecta y analiza la evidencia objetiva relevante y suficiente para tomar conclusiones acerca del proyecto que esté auditando.
- **Gerente de SQA**: responsable de manejar a las personas que tiene a su cargo que hacen auditorías. Prepara los planes de auditoría, calcula el costo, asigna los recursos, resuelve las no conformidades.

Preparación y planificación: Se planifica y prepara la auditoria en forma conjunta entre el auditado y el auditor. Generalmente es el líder de proyecto quien la convoca. Estas no son sorpresas.

Ejecución: Durante la ejecución, el auditor pide documentación y hace preguntas. Busca evidencia objetiva (lo que está documentado), y subjetiva (lo que el equipo expresa que hace).

Análisis y reporte de resultado: Se analiza la documentación, se prepara un reporte y se lo entrega al auditado.

Seguimiento: Dependiendo de cómo funcione el acuerdo entre el auditado y el auditor, el auditor puede hacer un seguimiento de las desviaciones que encontró hasta que considere que han sido resueltas.

Checklist → para no perder el foco, guía de mínimos para responder para garantizar que independientemente quien es el que las hace, se controle lo mínimo.

Tipos de resultados:

- Buenas prácticas: auditor se encuentra con algo superador de lo que se esperaba.
- Desviaciones: cualquier cosa que no se hizo como el proceso dijo que había que hacer.
- Observaciones: cosas que advierte el auditor que no llegan a ser desviaciones, pero son riesgosas entonces se deja por sentado para tener en cuenta.

- **Técnicas y Herramientas para la realización de revisiones técnicas del software.**

Actividades de aseguramiento de calidad que comprueban la calidad de los entregables del proyecto. Incluye: examinar el software, su documentación y los registros del proceso para descubrir errores y omisiones, así como observar que se siguieron los estándares de calidad. También pueden revisarse los modelos de proceso, planes de prueba, procedimientos de gestión de configuración, entre otras cosas.

- Objetivo principal: encontrar errores durante el proceso a fin de que no se conviertan en defectos después de liberar el software, detectar los errores antes de que pasen a otra actividad de la ingeniería de software o de que se entreguen al usuario final.
- Beneficio: descubrimiento temprano de los errores, de modo que no se propaguen a la siguiente etapa del proceso de software, reduce el costo de las actividades posteriores en el proceso de software.

El propósito de las revisiones e inspecciones es mejorar la calidad del software, no de valorar el rendimiento de los miembros del equipo de desarrollo. Se debe desarrollar una cultura de trabajo que brinde apoyo y no culpar cuando se descubran errores.

Los procesos ágiles pocas veces usan procesos de inspección formal o revisión de pares. En vez de ello, se apoyan en los miembros del equipo que cooperan para comprobar mutuamente el código y en lineamientos informales.

A diferencia del testing, la revisión no requiere de la ejecución del software para realizar dicho análisis por lo que puede aplicarse sobre cualquier artefacto.

Existen distintos tipos de revisiones que se pueden clasificar según el objetivo que persigan o el grado de formalidad con las que se lleven a cabo.

Método	Objetivos Típicos	Atributos Típicos
Walktroughs	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

Dos aproximaciones:

- **PEER REVIEW – Revisiones de pares**

Miembros del equipo colaboran para encontrar bugs en el programa.

Permiten identificar problemas con las pruebas, y así, mejorar la efectividad de las mismas en la detección de bugs del programa.

- **WALKTHROUGH**

Técnica de análisis estático donde el desarrollador guía a los miembros de un equipo de desarrollo a través del artefacto a revisar

Se diferencia de otras técnicas en que el autor del artefacto es el que toma el rol dominante, porque justamente guía a los demás.

Objetivo se centra en satisfacer las propias necesidades del autor. Capacitación de los desarrolladores.

No sigue ningún procedimiento definido, no requiere aportes de gerencia y no genera métricas.

Registros de las presentaciones son raramente tomados y guardados.

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.

Se toman pocas métricas

No hay control de proceso

- **INSPECCIÓN**

Principal actividad FORMAL de garantía de calidad de software.

Al ser **formal** es un proceso controlado que cuenta con etapas se encuentran definidas, tiene roles y responsabilidades establecidas y una agenda específica.

Proporciona métricas útiles a lo largo de todo el ciclo de vida del desarrollo.

Miembros de equipo realizan una revisión línea por línea del código fuente del programa, buscan defectos y problemas, y los informan → examen visual de un producto de software, documentos relacionados y desvíos con respecto a estándares o especificaciones.

Se suele utilizar una lista de verificación de errores comunes de programación para enfocar la búsqueda de bugs. Cada organización debería desarrollar su propia lista con base en estándares y prácticas locales.

Puede aplicarse a cualquier artefacto generado. Es uno de los métodos más efectivos de aseguramiento de la calidad.

Una RTF sólo tendrá éxito si se planifica, controla y atiende apropiadamente.

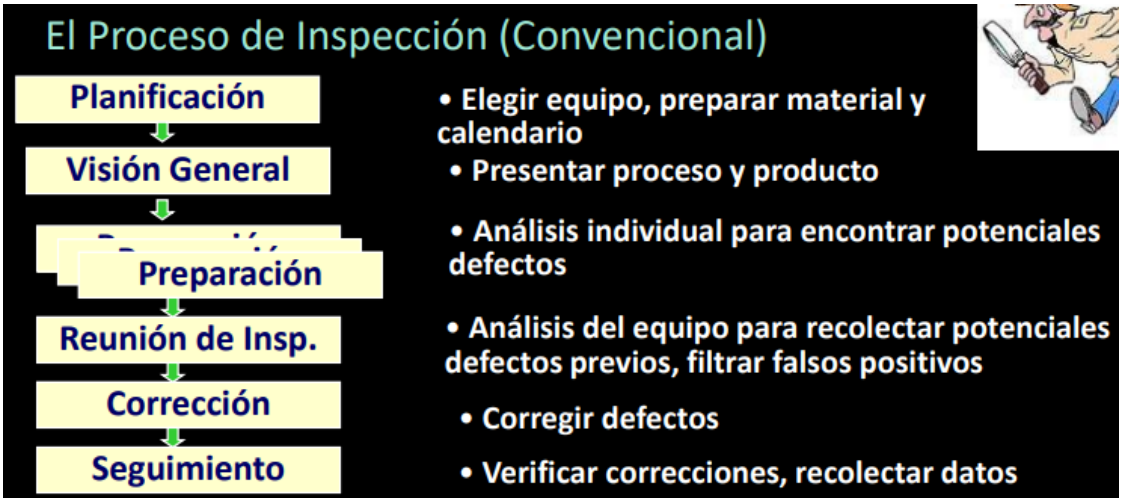
Su objetivo principal es descubrir los errores durante el proceso, de modo que no se conviertan en defectos después de liberar el software.

Objetivos:

- Descubrir errores.
- Verificar que el software alcanza sus requisitos.
- Garantizar que el software ha sido representado de acuerdo con ciertos estándares.
- Conseguir un software desarrollado de manera uniforme.
- Hacer que los proyectos sean más manejables.

SON	NO SON
La forma más barata y efectiva de encontrar fallas	Utilizadas para encontrar soluciones a las fallas
Una forma de proveer métricas al proyecto	Usadas para obtener la aprobación de un producto de trabajo
Una buena forma de proveer conocimiento cruzado	Usadas para evaluar el desempeño de las personas
Una buena forma de promover el trabajo en grupo	
Un método probado para mejorar la calidad del producto	

Rol	Responsabilidad
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 nro. 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 (48hs) antes de la reunión. •Coordina la reunión asegurándose que no hay conductas inapropiadas • Hacer 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.



- **Conceptos generales sobre calidad.**
- **Importancia de trabajar para y con Calidad. Ventajas y Desventajas.**

- Calidad: “Todos los aspectos y características de un producto o servicio que permiten alcanzar las necesidades, tanto las manifiestas o dichas, como las implícitas (expectativas).”

- Muy difícil de medir → concepto muy subjetivo/relativo relacionado con las necesidades y expectativas de las personas.
- Directamente relacionada con la persona, su circunstancia, su contexto, su edad, circunstancias de trabajo, el paso del tiempo.
- Los requerimientos del software son la base de las medidas de la calidad.

Hablando específicamente de software → circunstancias que hacen que no se tenga calidad

- 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 listo nunca se transforma en listo.
- ¿Dónde está ese componente?

Software de calidad satisface necesidades/expectativas de muchas personas que son diferentes

- Expectativas del cliente
- Expectativas del usuario
- Necesidades de la gerencia
- Necesidades del equipo de desarrollo y mantenimiento
- Otros

PRINCIPIOS que sustentan la necesidad del aseguramiento de calidad.

- **Calidad no se "inyecta" ni se compra, debe estar embebida**

Se concibe desde el momento 0, no se agrega al final, ni el testing agrega calidad. No se va a conseguir que el producto tenga calidad con testing. Es algo que se debe insertar MIENTRAS se hace el software.

- **Es un esfuerzo de todos**

Todos están involucrados, como cultura. No es responsabilidad de un departamento o individuo, sino que todos aportan una parte para mantener la integridad del producto de software. SCM da las bases, el cimiento para que se pueda trabajar con calidad.

- **Personas son la clave para lograrlo**

Capacitación: factor clave. Contexto de la industria: actividad humano-intensiva. Software lo hacen las personas. Clave de éxito = personas → capacitación.

Contar con empleados comprometidos y motivados → clave para ejecutar sus objetivos de calidad y crear valor.

- **Se necesita sponsor a nivel gerencial** → se puede empezar por uno.

Debe construir y mantener valores mientras asegura que otros líderes dentro de su negocio cumplan con un modelo ético. Contar con el respaldo de que la gerencia considera el desarrollo de un producto de calidad como fundamental y proporcione recursos y apoyo.

- **Se debe liderar con el ejemplo**

Grandes líderes ayudarán a su equipo a trabajar hacia los mismos objetivos de calidad, mejorando el nivel general de calidad en su organización y demostrando prácticas que lo hagan.

- **No se puede controlar lo que no se mide**

Es importante llevar un seguimiento y tener en claro los defectos y errores del software que se está construyendo, sino no se van a poder tomar decisiones que logren corregirlo o bien encontrar los puntos de fallo donde quizás se estén generando. Las pruebas son una herramienta fundamental para medir y controlar la calidad del software. Sin pruebas adecuadas, no se puede tener un control efectivo sobre la calidad.

- **Simplicidad, empezar con lo básico**

En lugar de complicar los procesos, se debe buscar la simplicidad y comenzar por los fundamentos básicos teniendo en cuenta la posición de la organización.

- **El aseguramiento de la calidad debe planificarse**

Es muy importante y necesario contar con un plan de aseguramiento de calidad que establezca los objetivos, los estándares a seguir, las actividades y recursos necesarios para garantizar la calidad del producto o servicio. Si no se planifica, no se tiene dimensión de los riesgos que puede conllevar el no lograr la calidad esperada del producto. Se deben especificar y tener visibilidad de los pasos a realizar para lograrla.

- **El aumento de las pruebas no aumenta la calidad**

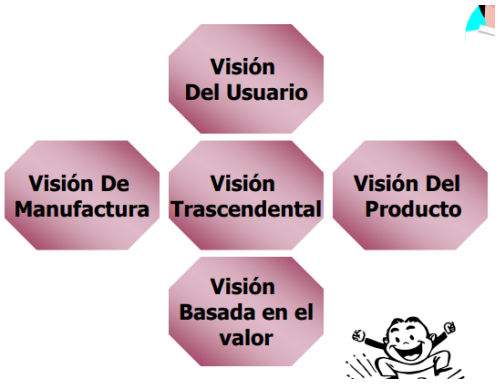
No garantiza una mayor calidad. Se deben tener en cuenta las diferentes técnicas y enfoques adecuados para lograr resultados efectivos, analizando los distintos niveles y tipos de prueba y decidir analíticamente las que se vayan a realizar que logren la mayor cobertura del producto.

- **Debe ser razonable para mi negocio**

Relacionado a la simplicidad, el aseguramiento de la calidad al cual se quiere llegar debe adaptarse a las necesidades y características específicas de cada organización, siendo realistas en cuanto a tiempo y recursos para lograr objetivos realmente alcanzables.

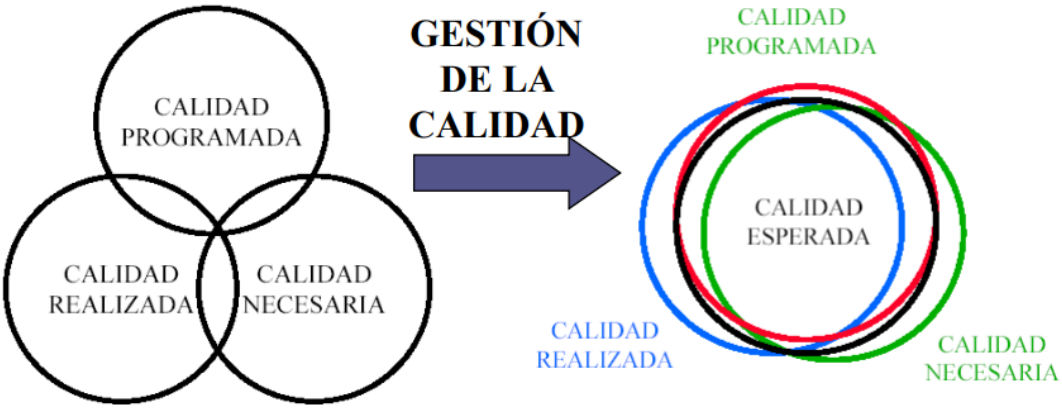
¿CALIDAD DESDE QUÉ PERSPECTIVA?

Cuando se habla de calidad, es importante definirla desde una perspectiva. Características de calidad el producto que se esperan, que son distintas con respecto a otro producto, expectativas de calidad el usuario, del equipo de desarrollo, etc.



- **VISIÓN TRASCENDENTAL** → es algo que se reconoce de inmediato, pero que no es posible definir explícitamente. Lograr cosas más allá de lo que uno se puede imaginar que puede hacer. Motor que ayuda a seguir, avanzar.
- **VISIÓN DEL USUARIO** → metas específicas del usuario final, si un producto las satisface, tiene calidad.
- **VISIÓN DEL FABRICANTE/MANUFACTURA** → especificaciones originales del producto, si un producto las cumple, tiene calidad.
- **VISIÓN DEL PRODUCTO** → tiene que ver con las características inherentes de un producto (funciones y características).
- **VISIÓN BASADA EN EL VALOR** → la mide de acuerdo con lo que un cliente está dispuesto a pagar por un producto.

Desafío → lograr una coincidencia de las tres perspectivas de calidad.



- **Calidad programada**: expectativa de la calidad que ese producto tenga.
- **Calidad realizada**: lo que realmente se hizo por la calidad del producto. Esta puede varias debido a limitaciones de tiempo, recursos, cambios en los requisitos o capacidades del equipo.
- **Calidad necesaria**: mínimo que el producto tiene que tener para cumplir con los requerimientos del usuario, sin los cuales el software no cumpliría su propósito principal.

Buscar → intersección de las 3 sea lo suficientemente grande como para cubrir todas. Porque todo lo que esté fuera de dicha coincidencia va a ser DESPERDICIO o INSATISFACCIÓN.

VENTAJAS

- Ayuda a detectar defectos, errores y problemas con la experiencia de usuario, garantizando que el software cumpla con los objetivos de la empresa y las necesidades del usuario.

Mejora el tiempo de llegada al mercado.

Satisfacción del Cliente: Cuando los productos o servicios cumplen con los estándares de calidad, los clientes están satisfechos, lo que conduce a una mayor lealtad y la posibilidad de que recomienden la empresa a otros.

Además, ayuda a satisfacer las necesidades y expectativas no sólo del cliente sino también de otros interesados, equipo de desarrollo y mantenimiento, usuarios finales, entre otros.

Eficiencia Operativa: La calidad conlleva procesos más eficientes. Se reducen los errores y se minimizan los retrabajos, lo que disminuye los costos operativos y aumenta la productividad.

Reputación Mejorada: Las organizaciones que entregan productos de alta calidad ganan una mejor reputación en el mercado. Esto puede atraer nuevos clientes y mantener a los existentes.

Ventaja Competitiva: La calidad puede ser una ventaja competitiva significativa. Las empresas que ofrecen productos o servicios superiores a menudo pueden cobrar precios más altos o competir de manera más efectiva en el mercado.

Reducción de Errores y Costos: Trabajar para la calidad ayuda a prevenir errores desde el principio, lo que reduce los costos asociados con correcciones y devoluciones.

Puede reducir el costo de desarrollo de software. Cuando los errores llegan al producto final, solucionar el problema puede ser muy costoso, lo cual a su vez puede causar retrasos.

Cultura de Mejora Continua: Fomenta una cultura de mejora continua, lo que significa que la organización siempre busca formas de optimizar sus procesos y productos.

Mejora continua de los procesos: el SQA tiene como objetivo mejorar constantemente los procesos de desarrollo de software. Esto se logra evaluando los procesos existentes, identificando áreas de mejora e implementándolas.

DESVENTAJAS

- Gente no lo mantiene.

- Es caro. Tiene un alto costo porque conlleva más recursos.

- Planificación extra.

- Los beneficios no son a corto plazo y se debe esperar para poder visualizarlos.

Insatisfacción del Cliente: La falta de calidad puede llevar a la insatisfacción del cliente, lo que puede resultar en pérdida de negocios y mala publicidad.

Costos Adicionales: La corrección de errores y defectos puede ser costosa, ya que puede implicar volver a hacer el trabajo, reemplazar productos o enfrentar reclamaciones de garantía.

Pérdida de Reputación: La falta de calidad puede dañar la reputación de la empresa, lo que a largo plazo puede ser difícil de restaurar.

Competencia Difícil: Las organizaciones que no trabajan para la calidad pueden tener dificultades para competir en un mercado donde la calidad es un factor importante para los consumidores.

Sanciones Regulatorias: En ciertas industrias, la falta de calidad puede resultar en sanciones regulatorias, multas y demandas legales.

Pérdida de Clientes: La insatisfacción del cliente puede llevar a la pérdida de clientes y una disminución de los ingresos.

- **Actividades relacionadas con el Aseguramiento de la Calidad del Software.**

Patrón planeado y sistemático de acciones que se requieren para garantizar alta calidad en el software y que “se hacen las cosas correctas en el momento correcto y de la forma correcta”.

Actividad sombrilla que incluye un conjunto de procesos y prácticas que las empresas utilizan para garantizar la calidad de sus productos.

Se ve materializado en un grupo de ACS que funciona como el representante del cliente en el interior de la organización y tiene como objetivo asegurar que las actividades de apoyo del software se lleven a cabo, además de planear, supervisar, registrar, analizar y hacer reportes acerca de la calidad. De esta manera “ayuda” al equipo del software a lograr un producto final de alta calidad.

El Aseguramiento de la Calidad del Software es una actividad sombrilla, que incluye:

1) Un proceso de aseguramiento de calidad

2) Tareas específicas de aseguramiento y control de calidad (incluye revisiones técnicas y estrategia de pruebas relacionadas entre sí)

3) Prácticas eficaces de ingeniería de software (métodos y herramientas)

4) Control de todos los productos del trabajo de software y de los cambios que sufren

5) Procedimiento para garantizar el cumplimiento de los estándares del desarrollo de software

6) Mecanismos de medición y reporte

Actividades:

1) **Definición de un *Plan de Aseguramiento de Calidad de Software* para un proyecto**:

Identificar productos de calidad deseados, proceso de evaluación que se va a realizar, auditorías y revisiones, seguimiento de errores, actividades de prueba, la definición de criterios de aceptación y la estimación de recursos necesarios.

Establecimiento de estándares ya sea nacionales, internacionales, organizacionales o del proyecto, son clave para la administración de calidad efectiva. Pueden ser de:

- Producto: definir características que todos los componentes deberían tener.
- Proceso: cómo deberían ser implementados los procesos de software. Cómo deberían conducirse revisiones, cómo debería realizarse la administración de configuración.

Importante hacerla → se debe tener CONTRA QUÉ COMPARAR.

2) **Participar en el desarrollo de la descripción del software del proyecto**:

El equipo de software selecciona un proceso para el trabajo que se va a realizar y el grupo de ACS revisa la descripción del mismo a fin de cumplir con las políticas, estándares establecidos y otras partes del plan de proyecto.

3) **Supervisar, controlar y monitorear el cumplimiento de las actividades según el proceso definido**:

Identifica, documenta y da seguimiento a las desviaciones del proceso, verificando que se realicen las correcciones pertinentes en caso de ser necesarias. Que se realicen las pruebas planificadas, que las revisiones se realicen.

4) **Auditar y supervisar los productos designados para verificar que se cumpla con aquellos definidos**:

Revisa productos de trabajo seleccionados; verifica que se realicen las correcciones necesarias y garantiza que se sigan los lineamientos de calidad.

5) Registrar toda falta de cumplimiento y reporta a la alta dirección

6) Gestión de defectos: reúne y analiza errores y datos acerca de los defectos para entender mejor cómo se cometen y qué actividades son más apropiadas para eliminarlos.

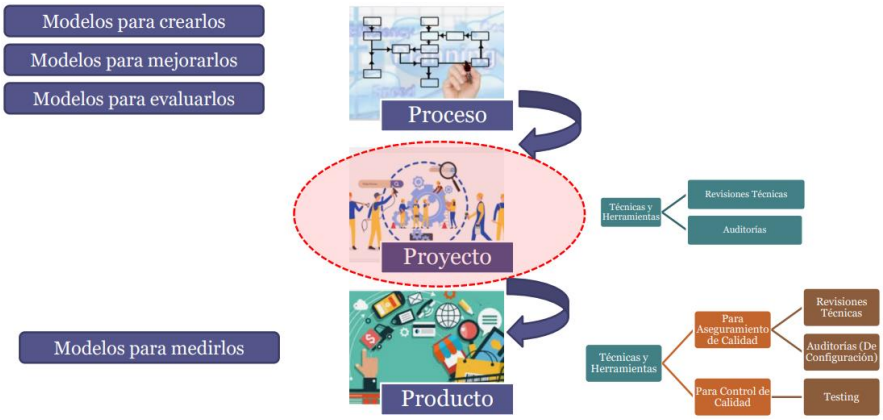
7) Documentación y métricas: lleva una documentación detallada de las pruebas, resultado y procesos para rastrear las actividades realizadas, además de analizar los indicadores de la existencia de calidad.

- **Principales Modelos de Calidad existentes (CMMI – SPICE – ISO) y sus métodos de evaluación.**

CONTEXTO

Para hacer software se necesita una guía o estructura para saber lo que hay que hacer para lograr un objetivo → PROCESO.

En un inicio, al crear un proceso en una organización, lo mejor es basarse en modelos. **Modelos para crearlos** proporcionan lineamientos y mejores prácticas para la creación y gestión de procesos, así como para garantizar la calidad del software.



Una vez que se ha establecido el proceso y se quiere funcionar en un ciclo de mejora continua del mismo existen → **modelos para mejorarlos**: sirven para dar marco de referencia a las organizaciones que quieren mejorar sus procesos.

Después posible formalizar o certificar el cumplimiento de un proceso → **modelos de evaluación**: sirven para medir y evaluar el grado de adherencia del proceso al modelo que se tomó como referencia. Lo usan los evaluadores para que no sea tan distinto lo que evalúan.

Ese proceso seleccionado se va a ver instanciado en el **PROYECTO** y es el ámbito en el cual uno trabaja para generar el producto.

Entonces, mientras el proyecto se está ejecutando, si se quiere integrar calidad, se insertan actividades que permitan ver cómo se están haciendo las cosas para ver que estoy haciendo, asegurando que se sigue con lo que se dijo que se iba a hacer: Revisiones Técnicas, Auditorías (del proyecto).

En este contexto a su vez → insertan tareas para asegurar y controlar la calidad del producto: Revisiones Técnicas, Auditorías funcionales y físicas, Testing.

Técnicas se repiten → se puede usar Revisiones Técnicas para revisar la calidad del producto.

TODO OCURRE EN EL PROYECTO: independientemente de las 3 dimensiones, todo ocurre en el proyecto, se integra la gente, la gente adapta el proceso que va a usar y que genera el producto. Cuando se hace actividades de aseguramiento de calidad, tanto de proceso como de producto se hace en el contexto de un proyecto específico.

Problema de visión entre los procesos definidos y empíricos:

- Definidos: Calidad del producto depende de la calidad del proceso que se utiliza para construirlo. Pero no es tan literal ni directo.
- Empíricos: Calidad del producto depende de la calidad del equipo, de la gente.

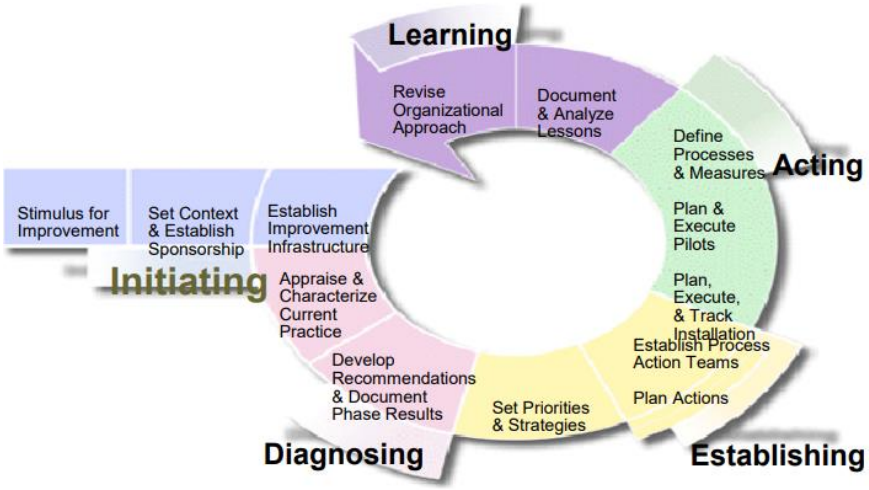
CALIDAD DEL PRODUCTO

No se puede sistematizar como el proceso → no hay modelos en la industria para evaluar la calidad de un producto que se pueda aplicar como una plantilla a todos los productos por igual.

Cada producto tiene sus características y la calidad depende de quienes van a usar el producto.

MODELOS DE MEJORA DE PROCESOS

- Esquemas plantillas o recomendaciones de trabajo para encarar un proyecto de mejora de un proceso.
- Propósito: analizar un proceso de la organización para organizar un proyecto cuyo objetivo va a ser un proceso.
- Motivación: calidad del producto final depende de la calidad del proceso, entonces, si mejoro la calidad del proceso voy a mejorar la calidad del producto.
- SPICE: Software Process Improvment Capability Evaluation.
- **IDEAL: Initiating, Diagnosing, Establishing, Acting, Leveraging.**



Sirve para ayudar a mejorar un proyecto **cuyo resultado va a ser un proceso definido**, no importa con qué prácticas ni con qué proceso. DICEN EL QUÉ NO EL CÓMO. PARA ACREDITAR O EVALUAR SI O SI TIENE QUE SER UN PROCESO DEFINIDO.

Al principio se busca un sponsoreo, apoyo en la organización, alguien que tenga poder en la organización y ayude a empujar a que se haga → muy importante porque este tipo de proyectos nunca son críticos en las empresas, siempre hay algo más crítico. Entonces, así se asegura que la gente le ponga "ganas" a realizarlo.

- Se inicia con un diagnóstico → dónde se está parado, qué se hace bien, qué se hace mal, a dónde se quiere ir → análisis de brecha, gap análisis.
- Se desarrollan planes de acción. Proceso como CMMI nivel 2. Qué modelo de calidad voy a usar como referencia
- Definir las actividades, roles, prácticas que quiero que la gente haga, framework de gestión.
- Se prueba el proceso definido en algún proyecto, dos o tres, para tener una supervisión más fina. Recomendación de trabajar con **proyectos piloto**: esos 2 o 3 proyectos donde se prueba, el proceso se adapta y recién después hacer la implementación en el resto de la organización.
- Después vienen personas a evaluar, auditar. Se eligen 2 3 4 proyectos más representativos de la empresa. Ver si el proceso cumple con el modelo de calidad que se eligió seguir.
- Si está bueno extrapolarlo a la organización en aquellos proyectos que van a utilizar el proceso definido.
- Identificar oportunidades de mejora y adaptación → se transforma en un ciclo: se obtiene información para obtener aprendizaje.

Cuando digo dónde estoy con mi proceso y a dónde quiero llegar → entran → **MODELOS DE CALIDAD**: se usan como referencia para bajar lineamientos que nuestro proceso tiene que cumplir para alcanzar cierto objetivo. Idea implícita: mejora continua.

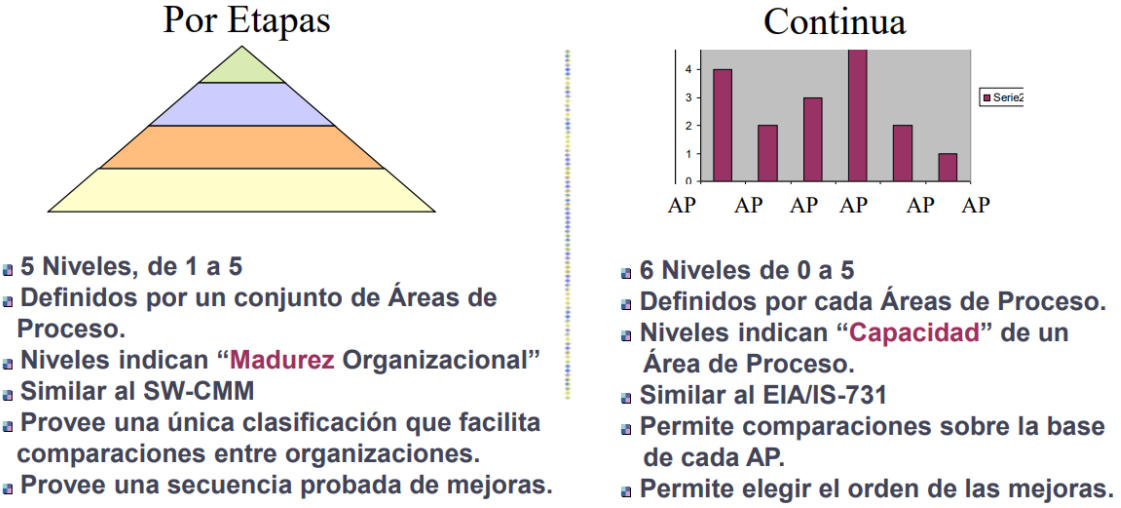
Si se quiere definir un proceso que la organización tiene que respetar y usar → miles de modelos. Tienen origen militar. TODOS EVALÚAN PROCESO.

Resultado de lo que se hace debe estar escrito en algún lado.

CMMI → pensado para empresas que hacen software.

- Empieza → departamento de defensa de EEUU
- **Integración de los modelos de madurez y capacidad**
- **Proporciona un conjunto de mejores prácticas y criterios que las organizaciones pueden seguir para mejorar sus procesos y que tengan cierta calidad.**
- Intenta variante parecida a las normas ISO para capturar el público.
- No es una norma, no se certifica, sólo se evalúa a través de profesionales reconocidos por el SEI. Se tiene que llamar a alguien que lo certifique y evalúe.
- Método formal para evaluar para que una empresa sepa que adquirió un determinado nivel → por ejemplo SCAMPI, es un modelo de evaluación de CMMI.
- Suma 3 dominios: constelaciones, es como una especialización.

    - DEV → **Desarrollo**: provee guía para mediar, monitorear y administrar los procesos de desarrollo.
    - SVC → Servicios: provee guía para entregar servicios internos o externos.
    - ACQ → Adquisición: provee guía para permitir seleccionar y administrar adquirir productos y servicios.
- Diferencia con CMM → dos formas de mejora.



Por Etapas

SW-CMM identificaba organizaciones y las dividía en dos tipos:

- Maduras: de nivel 2 al 5. Mientras más madura, más capacidad tiene para cumplir con sus objetivos. Mejoraba la calidad de sus productos y bajaba sus riesgos, de manera gradual.
- Inmaduras: de nivel 1.
- Niveles definen qué AP se tienen que cumplir de manera obligatoria.
- Habla a nivel organizacional. Ventaja: facilita la comparación entre organizaciones al proveer de una única clasificación.
- La representación por etapas proporciona una estructura clara y lineal para mejorar la madurez de la organización, donde cada nivel construye sobre los logros del nivel anterior.
- HABLA DE **MADUREZ** DE LA ORGANIZACIÓN: determina la capacidad que tiene para hacer las cosas que tiene que hacer, transparencia en los artefactos.

Continua

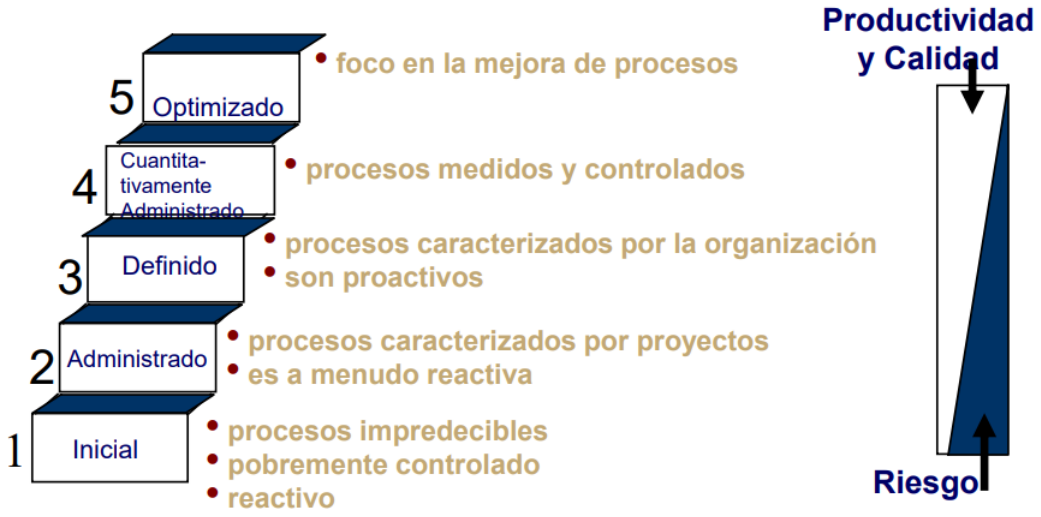
- Elegir áreas de proceso dentro de las que el modelo ofrece → 22. (AP son las mismas en las dos representaciones).
- Se elige qué proceso se quiere mejorar por separado. En vez de medir la madurez de toda la organización, se mide la capacidad de un proceso en particular.
- La empresa elige las AP y apunta a mejorar esas en particular.
- Mide la **CAPACIDAD** de las distintas áreas de proceso de manera individual para poder cumplir los objetivos.

GRUPO: existencia de roles que cumplan ciertas tareas.

- Esos roles se adaptan al tamaño de la organización, su nivel de madurez.
- Ej: grupo de SCM. Alguien tiene que asumir el rol de gestor de configuración de software.
- Importante → que exista alguien responsable de cubrir las actividades de cada uno de los roles o grupos.

POR ETAPAS

Niveles de madurez son acumulativos. Ej: si una organización es nivel 5, también sos nivel 4, 3 y 2.



Nivel 1: nivel de inmadurez.

- No se tiene ninguna visibilidad sobre el proceso.
- No se sabe cuándo se va a terminar, calidad de lo que se va a obtener. Algo entra y en algún momento algo va a salir, pero no se sabe cuándo.
- Procesos sin definir o improvisados.
- Éxito se consigue, pero a qué costo. A veces hace el esfuerzo 1 o 2 personas.
- Actitud frente a los riesgos: REACTIVA → ataca las cosas cuando ya se convirtieron en problemas.

Foco → nivel 2: disciplinas de gestión y soporte. No hay ningún área de ingeniería de producto. Organización con madurez para ADMINISTRAR SUS PROYECTOS y dar soporte, se sabe qué esperar.

- Administración de requerimientos
- Planeamiento de proyectos
- Monitoreo y control de proyectos
- Administración de acuerdo con el proveedor → única que es opcional, si se contrata otra empresa para el desarrollo de software, tercerización.
- Medición y análisis
- Aseguramiento de calidad del proceso y del producto
- Administración de configuración.

- **Métricas en los 3 enfoques (Tradicional/Lean/Agile)**

- Tema difícil, debe tener un conjunto de características para que realmente sirvan, porque tiene un costo.
- Beneficio debe ser mayor que no tenerla.
- MÉTRICA: valor cuantitativo que determina la presencia de algo que se quiere medir en un contexto. Sirve para construir indicadores para tomar decisiones.
- Medida o presencia o grado de valor sobre un atributo que se quiere medir.
- **Es un NÚMERO**. Es objetiva.
- ¿Para qué se mide? → para tener visibilidad sobre una cierta situación, aumentar el nivel de conocimiento de algún aspecto. Finalmente ayuda a tomar decisiones de manera INFORMADA.
- Sirven a alguien para un propósito → necesidades varían según quien las necesite.
- Automatizar lo mayor posible.
- Tener línea clara de para qué se van a utilizar.

Dominio de las métricas del software TRADICIONAL, basada en procesos definidos:

- **Métricas de proceso**: no tienen que tener atribución a ningún producto ni proyecto particular de la organización.
  - Son públicas: hablan de un comportamiento organizacional.
  - Se utilizan fundamentalmente para mejorar el proceso, cómo lo mejoro sobre la base de una gestión objetiva.

    - Se sacan de las métricas del proyecto → despersonalización de las de producto o las de proyecto
- **Métricas de proyecto**: se consolidan para crear métricas de proceso que son públicas para toda la organización. Directamente relacionadas con la triple restricción, generalmente se mide cosas relacionadas a los recursos, tiempo y alcance.
    - Son privadas porque sino es muy difícil despersonalizar
    - Esfuerzo
- **Métricas de producto:** miden el software, hablan directamente del software. Muy difícil de medir directamente → epifenómeno, elegir algo alrededor para medir. Por ejemplo, la calidad, ejemplo: porcentaje de requerimientos satisfechos por el producto, índice de retención de usuarios.
    - Líneas de código
    - Funcionalidades
    - Defectos

Política de medición de una organización → depende. Si se tiene una definición organizacional, fijarse esa. Si no, tener en cuenta en cada proyecto y definirlas. De dónde sacar los datos, cálculos y cómo hacerlos para que todos lo hagan igual.

- **A LA GENTE NO SE LA MIDE, por eso no están las personas. No se hace de ninguna manera, no hay que buscar la culpa.**

ENFOQUE TRADICIONAL plantea MÉTRICAS BÁSICAS

No definir muchísimas métricas, después no se usan → DESPERDICIO.

Idea → se puede usar la experiencia de otros proyectos en proyectos que vendrán → repetitividad.

Foco → medir todo.

Proyecto → ámbito donde se hacen las métricas.

TAMAÑO DEL PRODUCTO

- Para medir: líneas de código sin comentarios → no sirve. Se obtiene al final del producto, es difícil.
- Ahora → funcionalidades, alcances, casos de uso, defectos.
- Se mide en: lo que el proyecto crea conveniente, casos de uso por complejidad. Busca homogeneizar el valor de diferencia.
- Es el QUÉ.
- Métrica de producto

ESFUERZO

- Esfuerzo que conlleva para realizar el producto de cierto tamaño. Ver nivel de desagregación.
- Horas de la implementación, testing, diseño. Según cada workflow.
- Se mide en: **horas personas lineales**. Asumir que es una persona la que hace el trabajo haciendo una tarea por vez, no tiene en cuenta si se puede hacer más de una cosa a la vez, ni la cantidad de personas, el solapamiento de tareas.
- Es el CÓMO.
- Métrica de proyecto.

TIEMPO (CALENDARIO)

- Esfuerzo se toma como base para determinar el calendario.
- Entran en juego otras variables → horas que se trabaja por día, días que se trabaja de la semana, índice de solapamiento para saber si se pueden hacer tareas en paralelo o bien son dependientes, cuánta gente va a trabajar.
- Se mide en: días, semanas, meses. Horas no está bueno porque es para el esfuerzo.
- Es el CUÁNDO.
- Métrica de proyecto.

DEFECTOS

- Se mide sobre el producto
- Cosas que se detectaron que no son coincidentes con lo que se espera.
- **Tener en cuenta defectos y su severidad**, porque sino no me sirve de nada tener la cantidad de defectos solo y únicamente.
- **Densidad de defectos**, cuántos defectos se tienen por bloque de código: user, caso de uso, módulo.
- Métrica de producto.

Desarrollador

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

Organización

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

Equipo de Desarrollo

1. Tamaño del producto
2. Duración estimada y actual entre los hitos más importantes.
3. Niveles de staffing 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

Volatilidad: cuánto cambian los requerimientos. Cambio de requerimientos por unidad de tiempo. Sirve para hacer análisis del impacto de los cambios, qué margen se tiene de estabilidad en los planes que se hacen.

Status de requerimientos. Cuánto tiempo tarda un requerimiento para moverse de un estado a otro. Capacidad de respuesta que se tiene con la capacidad de respuesta del cliente.

ENFOQUE ÁGIL

- Filosóficamente → visión de las métricas es ≠. Medir lo que sea necesario y nada más.
- "Medición es una salida, no una actividad" → No actividades específicas respecto de cuándo tomar las métricas, sino que sea algo integrado, que salga como una salida más como producto de las actividades.
- Principio: **LA MEJOR MÉTRICA DE PROGRESO ES EL SOFTWARE FUNCIONANDO**. "Nuestra mayor prioridad es satisfacer al cliente con entregas de software de producto de valor".
- Ágil plantea que la experiencia no es extrapolable y se construye en el propio equipo.
- Métricas que asume el agilismo para medir su framework → no significa que no se deban agregar las otras métricas de producto.
- Foco en ÁGIL → medir producto. Plantea que no tiene mucho sentido medir proceso, por eso de que la **experiencia no es extrapolable**.
- Se quiere satisfacer necesidades y expectativas, que son distintas → hacer equilibrio, balancear y en función de eso hacer esfuerzo porque se mantenga SIMPLE y que se vayan a usar realmente. Similar con la triple restricción → mantener balanceado para no poner en juego la calidad. Defectos: lo más básico relacionado a la calidad del producto.

En el contexto de un sprint:

VELOCIDAD

- No se puede asumir que un equipo va a tener la misma velocidad en un sprint y en otro → oscilaciones en la velocidad es normal.
- Mide cuántos PUNTOS DE HISTORIA **aceptó el PO** al final de un sprint → **mide producto**.
- No se estima, se calcula al final del sprint.
- Es la que mide software funcionando. Es la métrica de ágil, fundamentalmente es para el cliente.

- Métrica de producto.

CAPACIDAD

- Sí se puede estimar → se utiliza para saber cuánto se puede comprometer el equipo al inicio de un sprint. Ver cuántas horas puede trabajar cada miembro del equipo y cuántos días.
- Se usa en el sprint planning.
- Se puede medir: **horas ideales** (equipos menos maduros → necesitan desagregación de US en tareas) y **puntos de historia** (para equipos más maduros).
- Métrica de proyecto.

RUNNING TESTED FEATURES (RTF)

- Cantidad de características de software que están funcionando.
- Features = US a veces. Mide cantidades absolutas, pero no tiene en cuenta los puntos de historia.
- Mismo problema que cuando se contaban los casos de uso solos y no los casos de uso por complejidad.
- Si no se tiene en cuenta el tamaño o complejidad de cada característica no dice mucho, no se sabe por dónde pasa el valor.

ENFOQUE LEAN → KANBAN

- Lo que mide lo hace para cada pieza de trabajo que pasa por el tablero → unidad básica de medición.
- Foco en LEAN → medir proceso.

LEAD TIME

- Vista del cliente = lo que él ve y lo que le importa.
- La que el cliente mira, desde que el cliente me pidió algo hasta que se lo entregué.
- Registra el tiempo que sucede entre el momento en el cual se está pidiendo un ítem de trabajo (ingresa un kanban) y el momento de su entrega (el final del proceso).
- Tiempo que le cuesta pasar a un elemento de trabajo a través del sistema, desde el principio hasta finalizar.
- Medición = días de trabajo.
- Ritmo de entrega

Definir WIP en el backlog → IMPACTO → Lead Time.

CYCLE TIME

- Vista interna = para el equipo, los que hacen el trabajo.
- NO IMPORTA el tiempo que está algo en el backlog hasta que se toma, porque se cuenta desde que se empezó a hacer el trabajo hasta que se termina.
- Registra el tiempo que sucede entre el inicio y el final del proceso, para un ítem de trabajo dado.
- Medición más mecánica de la capacidad del proceso porque no se puede tomar muy en cuenta si hay una disparidad con la anterior. Porque finalmente la intención es satisfacer al cliente, y pasa a "no importar" el tiempo en que se trabaja.
- Medición = días de trabajo o esfuerzo.
- Ritmo de terminación

TOUCH TIME

- Tiempo en el cual un ítem de trabajo fue realmente trabajado (o tocado) por el equipo, horas reales de trabajo.
- Cuántos días hábiles pasó este ítem en **columnas de "trabajo en curso"** (no columnas de acumulación), en oposición con columnas de cola/buffer y estado.
- Cantidad total de elementos que hay en un sistema en un momento dado.
- Medición = días de trabajo.

EFICIENCIA DEL CICLO DE PROCESO

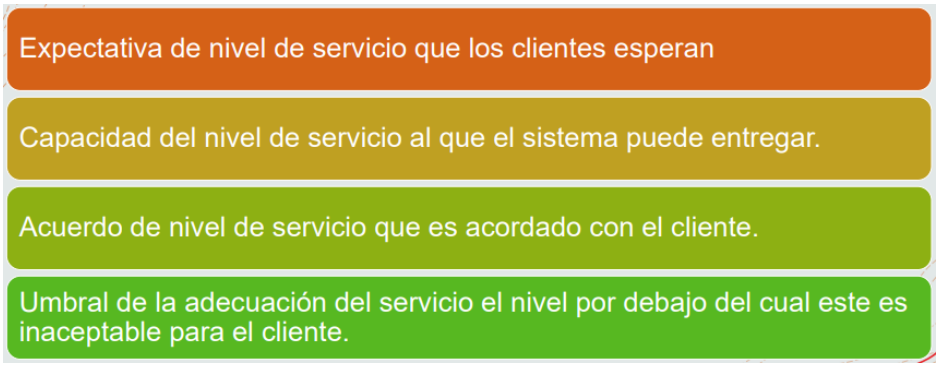
- % Eficiencia ciclo proceso = Touch Time / Lead Time → mientras más se acerca a 1, más eficiente porque se tiene menos desperdicio, menos tiempo que pasó entre que se pidió y se trabajó.

$$TT <= CT <= LT$$

Dos tipos de desperdicios:

- Evitables
- Inevitables: por ejemplo, las horas que se utilizan para las reuniones.

MÉTRICAS ORIENTADAS A SERVICIO



Expectativa de nivel de servicio que los clientes esperan.

Acuerdo de nivel de servicio → service level agreement: se suele hacer con los defectos. Tiempo máximo que se le da al equipo para resolverlo.

Nivel de tolerancia de los clientes. Qué tan malo se puede ser para no perderlos.

Independientemente del enfoque de gestión que se elija, el producto/defectos hay que medirlo.