RESUMEN TEÓRICO – 2DO PARCIAL

Clase 28/09/23 – Planificación de Release y Sprint

Definición de Scrum
Scrum es un **marco de trabajo** liviano que ayuda a las personas, equipos y organizaciones a generar valor a través de soluciones adaptativas para problemas complejos.

Framework que ayuda a gestionar proyecto de desarrollo de software. No apunta a ninguna actividad de ingeniería. Se quiere popularizar en cualquier ámbito de trabajo, quiere ser aplicable a muchas áreas.

Sirve para escenarios de gran incertidumbre y complejidad.

Características:

1. El marco de Scrum es **deliberadamente incompleto**, solo define las partes necesarias para implementar la teoría de Scrum
2. Scrum se basa en la **inteligencia colectiva** de las personas que lo utilizan. Equipos multifuncionales. Los integrantes pueden agarrar una US, empezarla y terminarla.
3. En lugar de proporcionar a las personas instrucciones detalladas, las reglas de Scrum **guían sus relaciones e interacciones**.
4. No te dicen todo lo que tienes que hacer, solo proporcionan ciertas restricciones y directrices

Teoría de Scrum

Basada en el **empirismo** (el conocimiento proviene de la experiencia y la toma de decisiones basadas en lo que se observa) y el **pensamiento Lean** (reduce los desperdicios y se centra en lo esencial).

Scrum emplea un enfoque iterativo e incremental para optimizar la previsibilidad y controlar el riesgo.

Scrum involucra a grupos de personas que **colectivamente** tienen todas las **habilidades y experiencia** para hacer el trabajo, compartir o adquirir tales habilidades.
Scrum combina **cuatro eventos formales** para la inspección y adaptación dentro de un evento contenedor, el **Sprint**

Roles del equipo de Scrum

Scrum se basa en un equipo de desarrolladores multifuncional y organizado. El equipo se autoorganiza en el sentido de que no existe un líder general que decida qué persona hará qué tarea o cómo se resolverá un problema. Son cuestiones que decide el equipo en su conjunto.
Los equipos de Scrum son multifuncionales, lo que significa que los miembros tienen todas las habilidades necesarias para crear valor en cada Sprint. También son autogestionados, lo que significa que internamente deciden quién hace qué, cuándo y cómo

Valores de Scrum

Compromiso, Enfoque, Apertura, Respeto, Coraje.

Scrum Framework

- Categorías de responsabilidades
  - Scrum Master
  - Product Owner
  - Developers (Desarrolladores). Generalización, pero con la intención de representa a los integrantes del equipo que son capaces de crear cosas.

- Reuniones
  - Sprint Planning
  - Daily Scrum
  - Sprint Review
  - Sprint Retrospective
  - Refinamiento del PB (actividad continua). On demand, cada vez que se necesite.

- Artefactos. Formalización y compromiso.
    - **Product Backlog** es el Objetivo del Producto.
    - **Sprint Backlog** es el Objetivo del Sprint. Obligatorio desde la admisión de cambios que no afecten el objetivo del sprint.
    - **Versión del Producto/ Product Increment** → DOD.

Horizontes de planificación en la Gestión Ágil de Proyectos

Necesitamos distintos horizontes de planificación porque un producto de software no se termina en una iteración. Niveles más altos de planificación
Sprint
El sprint es un horizonte de planificación de mediano plazo (generosamente) Iteración de máximo 4 semanas.
Contenedor de las demás actividades.

Eventos de longitud fija de un mes o menos para crear consistencia. Un nuevo Sprint comienza inmediatamente después de la conclusión del Sprint anterior

Durante el Sprint:

- No se hacen cambios que pongan en peligro el Objetivo Sprint; Si el cambio no afecta el objetivo del sprint, si se podría considerar. Pero únicamente aquellos que estén alineados con el objetivo.
- La calidad no disminuye;
- El trabajo pendiente del producto se refina según sea necesario;
- El alcance se puede clarificar y renegociar con el Propietario del Producto a medida que se aprende más.

Datos para tener en cuenta:

- Permiten la **previsibilidad** al garantizar la inspección y adaptación del progreso hacia un objetivo del Producto, como mínimo, una vez al mes en el calendario
- Cuando el **horizonte de un Sprint es demasiado largo**, el objetivo de sprint puede volverse obsoleto, la complejidad aumenta y el riesgo también.
- Cada Sprint puede considerarse un proyecto corto.
- Los **Sprints más cortos**, generan más ciclos de aprendizaje y limitan el riesgo de coste y esfuerzo a un período de tiempo más pequeño.
- Un Sprint podría ser cancelado si el Objetivo del Sprint se vuelve obsoleto
- Solo el Propietario del Producto tiene la autoridad para cancelar el Sprint

Objetivo del Sprint
El objetivo del Sprint es el Sprint Backlog.

Flexibilidad. Nos permite la variabilidad de características, en sentido de intercambiar algún ítem del product backlog, insertarlo en el sprint backlog.

Aceptación de una realidad: Las situaciones a veces deben modificarse.
"Si no impacta al objetivo del sprint, se podría llegar a aceptar algún cambio"

Ceremonia: Sprint Planning – Planificacion del Sprint

El Sprint planning inicia el Sprint estableciendo el trabajo que se realizará para el mismo.
Definir objetivos del Sprint.
Es creado por el trabajo colaborativo de todo el equipo de Scrum. Todos asisten.
Creación del artefacto → Sprint Backlog
Entrada: Improvements. Ingresan de la retrospectiva. Mejora que haya surgido.
Una planificación de Sprint mal ejecutada puede arruinar por completo todo el Sprint.

Produce concretamente:

- Una meta de Sprint.
- Una lista de miembros (y su nivel de dedicación, si no es del 100%)
- Una Pila de Sprint (lista de historias incluidas en el Sprint)
- Una fecha concreta para la Demo del Sprint.
- Un lugar y momento definidos para el Scrum Diario.

Ceremonia: Daily Scrum – Diaria

El PO no es necesario que asista
Hay inspección y adaptación.
Micro – management. Controlar todos los días para sincronizar al equipo.
Concreto, sin distracciones

Ceremonia: Refinamiento del Product Backlog

Fundamental la presencia del PO.

Ceremonia: Sprint Review

Fundamental la presencia del PO.
El PO puede invitar a más personas interesadas
Sirve para mejorar el **PRODUCTO**.
Momento donde se calcula la **VELOCIDAD (Cantidad de story points que nos acepta el PO)** del equipo. Mide producto, en base a story points. Puntos de historia que el PO aceptó al final de una iteración. Si no acepta la US, vuelve al producto backlog. No se estima, se calcula sumando los story points

Ceremonia: Retrospective

Sirve para mejorar el PROCESO
Todos los integrantes deben asistir.
Solo el equipo, recomendación de que asista un moderador o equipo de coach ágil o intercambio de scrum master.
El PO es importante.
Relacionada con el principio ágil 12. A intervalos regulares, el equipo se reúne a dar una retrospectiva. Cómo ser más efectivos.

Cancelación del Sprint

Lo hace el PO
Tiene costos asociados.
No es ideal.

TimeBoxing

El Timebox (caja de tiempo) es una técnica empleada en Scrum para limitar la duración de todos sus eventos. Una vez superado el tiempo establecido y conocido por todos los asistentes, el evento en curso debe finalizar.

"El tiempo encerrado en una caja" Apunta a que las actividades dentro del sprint son de duración fija. El tiempo es un recurso difícil de administrar. Necesidad de inmediatez.

Obliga a ser respetuosos con nuestro tiempo y con el de los demás. Es una restricción.

Sirve para aprender a negociar, en otros términos. El tiempo no se negocia.

Ayuda a eliminar las características no esenciales y enfocarnos en lo que importa en el tiempo dado.

Cambiar la variable de negociación.

Relación con la Triple Restricción.

Todas las ceremonias tienen TimeBox.

Objetivo es intentar cumplir con los compromisos propuestos, en el tiempo dado, focalizando la atención en lo esencial.

Refinamiento del PB: 10% del tiempo del Spring. Planteada en tiempos porcentuales, en términos relativos. Es el trabajo que se va haciendo

mediante que se ejecuta el Sprint, pero no hay un momento exacto para esta actividad. Puede ser en cualquier momento, cuando haga falta.

Definition of Ready - DOR

Criterio que define cuando una US está correctamente formulada como para ingresar al Sprint Backlog.
De base tiene que cumplir con el INVEST model.

Definition of Done - DOD

Criterio que define cuándo una US está lo suficientemente lista para entrar a la Sprint Review para ver si se lo podés mostrar al cliente.
Después de la aceptación del cliente, la US iría a producción.

Los criterios se relacionan con la calidad esperada del producto y las características esperadas del producto de software. Se necesita compromiso y respeto para que todos los criterios en la checklist del DOD se cumplan, sino se generan retrasos.

Hay cosas que no deberían ser negociables, como, por ejemplo, la seguridad.

Capacidad del Equipo en un Sprint

Es una de las métricas que usa Scrum.

Es la única que se **estima** y se usa para ver cuanto compromiso de trabajo el equipo puede asumir en un determinado sprint. Es una determinación de antemano de la capacidad de trabajo que el equipo puede asumir.

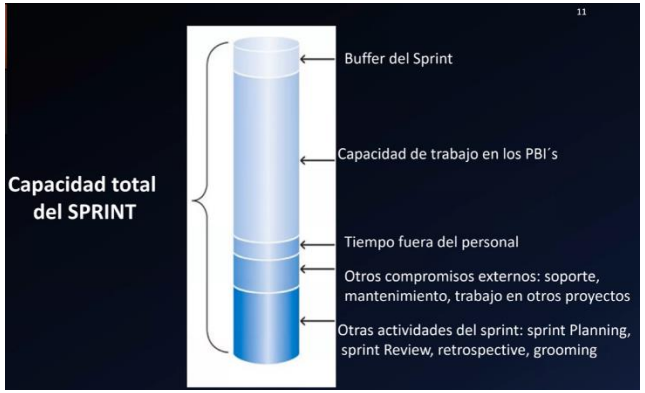
Se realiza dentro del **SPRINT PLANNING**, allí se determina la capacidad del equipo.

Contra esa capacidad, se contrasta cuántas US del product backlog pueden pasar al sprint backlog y hasta dónde nos podemos comprometer para un determinado sprint.

Por cada uno de los integrantes, se debería estimar cuántas horas ideales habría que dedicarle al trabajo en ese sprint.

La capacidad se puede estimar **en horas ideales**, para equipos más principiantes.

Si los equipos son más maduros, están en condiciones de planearlo según **story points.**



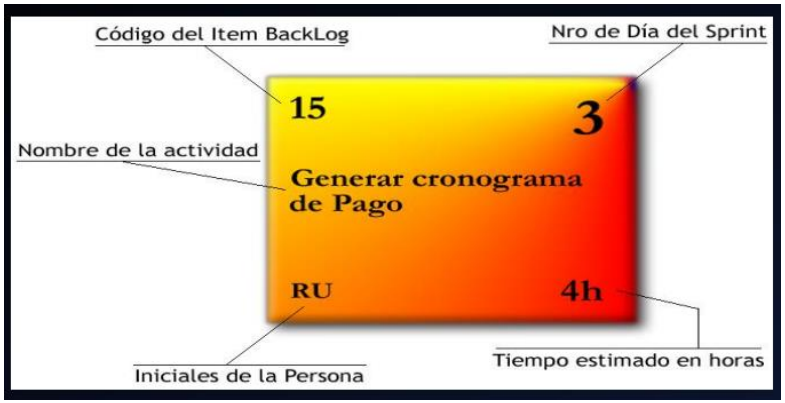
Ambiente de Trabajo

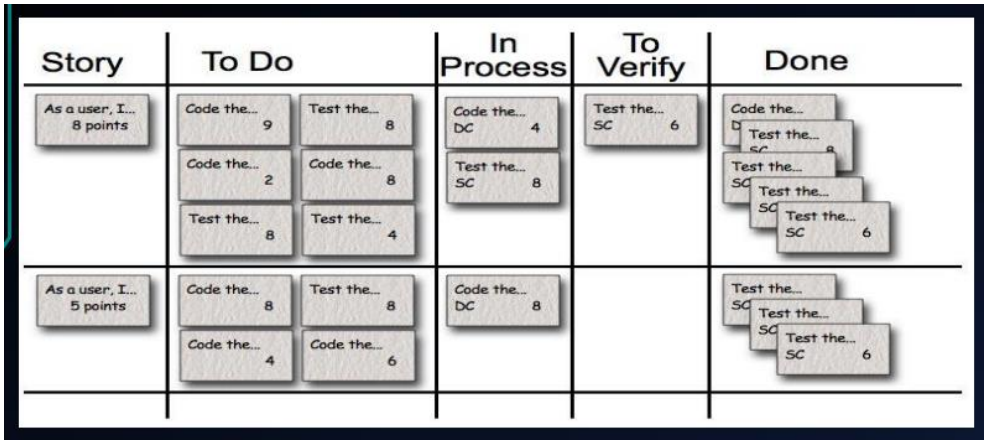
Trabajo VISIBLE, acuerdos VISIBLES

- Abierto
  - El silencio absoluto es un mal signo
- Pizarrones
- El equipo define sus horarios

Herramientas de Scrum

1. Taskboard





El taskboard muestra las US comprometidas en 1 determinado Sprint.

Los tableros duran lo que dura el sprint, se configuran en el sprint planning.

Configuración básica:

- To do: Vienen del PB cuando cumple el DOR.
- Doing: Sistema pull, los integrantes toman un una US y le pone su nombre.
- Done: Cuando cumple el DOD se pone en esta columna.

Configuración avanzada:
Cuando el equipo necesita hacer una desagregación de US en tareas, la configuración del tablero agrega más columnas. Se divide en tareas durante la PLANNING.

Story: User Story que se divide en tareas. Estimada en puntos de historia.

To do: Tareas (estiman en horas ideales)

In Process: Se le agregan las iniciales de la persona que desarrolle esa tarea. Por método pull. Los integrantes del equipo tienen que elegir con qué tarea comprometerse.

To Verify:
Done: Tarea terminada.
Done Done: Cuando todas las tareas están terminadas, está terminada la US.

Consideración importante del tablero y la buena gestión: Granularidad de las tareas. Se debe conseguir una buena granularidad que permita que enfocarnos en una actividad concreta de modo que no estemos todo el sprint haciendo una sola tarea.
Gestion Lean/Ágil - Binaria = tareas de granularidad fina, US chicas.
Fina granularidad del tablero. No se puede hacer un buen monitoreo.

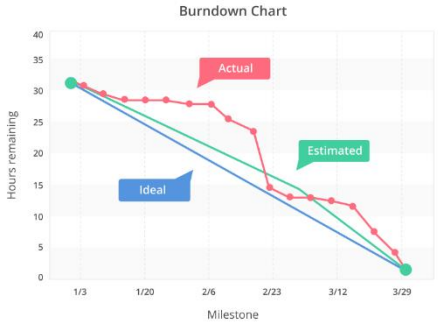
2. Sprint Burndown Charts. Este es el que pide SCRUM.

Deben estar visibles, como toda la información del proyecto.

Gráfico que permite visualizar el trabajo terminado. O el trabajo que te falta para cumplir con el compromiso de tu sprint.

En el eje de las Y, se colocan puntos de historia.
En el eje de las X, se colocan los días.

El día 0 del Sprint, cuando se comienza a trabajar, tengo tantos puntos de historia para terminar.
Idealmente uno actualiza este gráfico en la daily meeting en función del avance que la gente reporta en la daily.
Se limpian al final de cada sprint, y se calcula la velocidad.



3. Gráficos del seguimiento del producto.
Se pueden hacer permanentes. Cuando tengo Sprints en el eje de las x.

- Cuánto hemos hecho en cada sprint. Fijo. Scope: alcance esperado.
- Product Burndown. También permanente.

Sprint Burnup Chart
Es permanente por que hace el seguimiento de trabajo a lo largo del desarrollo de un producto. Sumamos los puntos de historia al finalizar los Sprints.

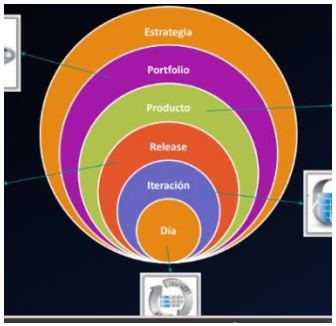
Múltiples niveles de Planificación

Ningún producto de software se puede terminar en un sprint.

Scrum llega hasta Iteración.

El nivel de planificación más chico es la daily meeting. Ahí es donde nosotros sincronizamos y donde se observa la situación de necesidad de hacer cambios, ajustes o adaptaciones en la diaria.

Luego sigue es sprint planning. Que se considera como una iteración. Porque devuelve un incremento.



Planificación del reléase. El reléase es una versión del producto que tiene un conjunto de características incluidas, que se libera y se pone en producción. Implica definir cuantos Sprints voy a necesitar para poder entregar X característica del producto que defini que debía abarcar el reléase.

Producto, comienza con una visión El producto es la sumatoria de los releases que voy a ir generando a lo largo del tiempo. Implica tomar decisiones sobre qué quiero ofrecer funcionalmente o no funcionalmente hablando y qué reléases pienso ir entregando con cada una de estas características. Decidir cuantos releases voy a entregar en un periodo de tiempo.

Portfolio, cuando una organización tiene varios productos. Portfolio backlog. Por ejemplo: Windows tiene un porfolio con Word, Excel, one note, Publisher, etc. Esos serían productos dentro de un protfolio. Toda la línea de productos de Google. Priorización de productos. A qué producto le damos más recursos y a cuál menos. Cual voy a descontinuar.

Backlog Portfolio. Sumatoria de todos los productos que la empresa tiene y la prioridad que tienen de desarrollo la va a dar el orden en el que están en el portfolio backlog.

Product Backlog. Nunca está 100% terminado y no hace falta que este 100% definido para poder llevarlo a cabo. Debe tener la suficiente cantidad de características del producto para poder comenzar a funcionar. NO ES UN CONTENEDOR DE LAS CARACTERÍSTICAS DEL PRODUCTO. No hay tareas en el producto backlog.

Release. Característica de producto que yo quiero liberar al mercado.

Sprint Backlog. User stories con el DOD completo, estimadas en puntos de historia. Luego el equipo decide si esas US las quiere dividir en tareas o no. Si las divide en tareas, las tareas se estiman en horas ideales. Acá si hay tareas.

Cambian los horizontes de planificación. A mayor horizonte, más costo, más tiempo, mirada más amplia. Quienes toman las decisiones también varían.

Nivel	Horizonte	Quién	Foco	Entregable
Portfolio	1 año o más	Stakeholders y Product Owners	Administración de un Portfolio de Producto	Backlog de Portfolio
Producto	Arriba de 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 entero, Stakeholders	Balancear continuamente el valor de cliente y la calidad global con las restricciones de alcance, cronograma y presupuesto	Plan de Release
Iteración	Cada iteración (de 1 semana a 1 mes)	Equipo Scrum entero	Que aspectos entregar en el siguiente sprint	Objetivo del Sprint y Sprint Backlog
Día	Diaria	Scrum Master y Equipo de Desarrollo	Cómo completar lo comprometido	Inspección del progreso actual y adaptación a la mejor forma de organizar el siguiente día de trabajo

Planificación de Portfolio

Niveles de decisión que están por encima del equipo.

Parten de una visión que alguien tiene sobre un producto.

Se hace la pregunta de si conviene o no invertir en un producto. Decidir si incluir o no un producto en el portfolio.

El equipo no tiene incidencia.

Planificación del Producto

Se da en términos de los releases.
Cuantos releases al año se entregan.
Están presentes las restricciones. Cuanto quiero hacer, hasta que fechas.

Planificación del Release

Busco: ¿Cuántos Sprints voy a necesitar para alcanzar este reléase??

Artefacto se llama Release Plan.

Tiene una consideración de cuántos Sprints vamos a ejecutar y un planteo del alcance de cada uno de ellos. También la duración de cada uno para determinar la duración del Release.

Ejemplo: Estimo como capacidad 20 puntos de historia por sprint.

Plan que puede llegar a cambiar según la realidad. Se puede replanificar si nos damos cuenta de que la capacidad del equipo es menor. La formalidad de la planificación es menor.

1er reléase suele considerar al MVP.

Propuesta que se puede cambiar debido a que pueden surgir cambios.

Salida:

- Rango de características que conforman cada sprint
- Cuantos Sprints necesito para terminar el reléase.
- Bases y condiciones determinadas.

Cadencia de los Sprints

Ritmo estándar entre los sprint.

Planificación del Sprint

User stories con DOD completo.

User stories que voy a desarrollar en un sprint determinado.

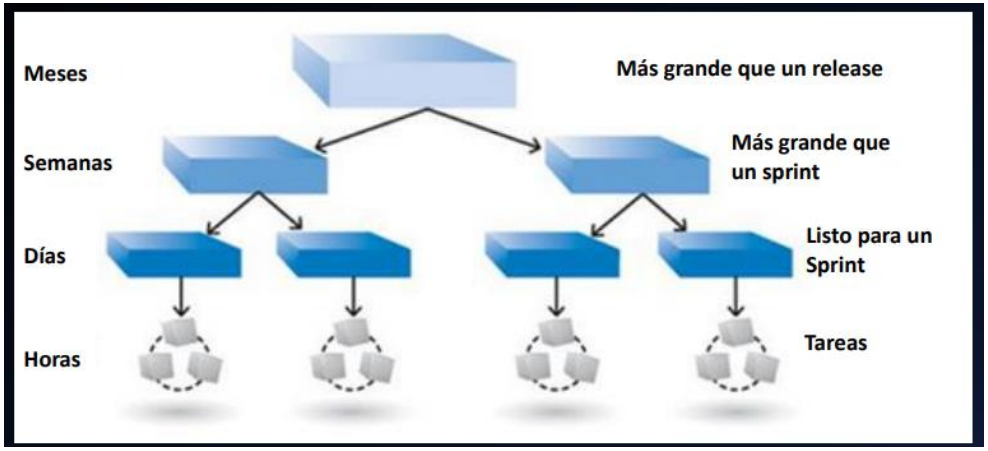
La planificación de iteración se realiza al principio de cada ciclo.

Entrada: Objetivo del sprint. Duración del sprint. Capacidad estimada del equipo.

Se seleccionan características a desarrollar.

- Por qué. Hacia el cumplimiento del objetivo del sprint.
- Que. Decisión de qué características vamos a incluir para cumplir con el objetivo.
- Como. El equipo lo hace desagregando en tareas cada user y estimándolas en horas ideales. Si el equipo decide que necesita hacerlo.

Niveles de Granularidad Distintos



Métricas en ambientes ágiles

- **Running Testes Features (RTF)**. Casi no se usa, cuenta cantidad de características que están funcionando que se desarrollaron en una iteración. Cuantas características tengo en producción por día de la iteración. Reemplazada por la velocidad. Se calcula en puntos de historia si o si. No se estima
- **Capacidad**. Es estimada, y tiene que ver con cuanto se pueden comprometer los integrantes del equipo, durante la planning a hacer durante un sprint. Se puede calcular en horas ideales o puntos de historias. Se estima por que es necesaria en la planificación del sprint.
- **Velocidad**. No se estima, se calcula al final contando los puntos de historia de las US que el PO me aceptó. Es una métrica del progreso de un equipo. Se calcula sumando el número de story points que el equipo completa durante la iteración. La velocidad corrige los errores de estimación.

PRODUCT BACKLOG

Es uno de los ARTEFACTOS DE SCRUM.

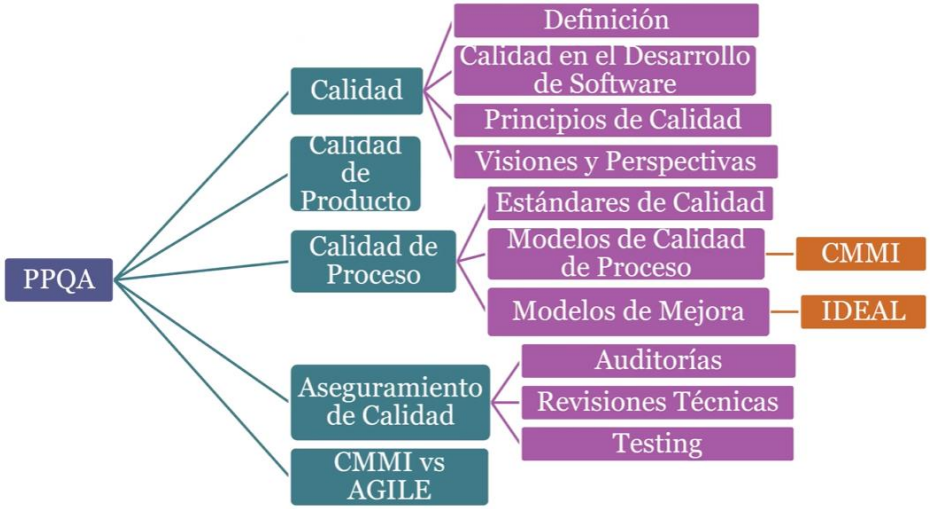
Es una lista ordenada de todo lo conocido que podría ser necesario en el producto y única fuente de requisitos y trabajo para el equipo Scrum.

1. Nunca está completo, es dinámico, emergente. Es un artefacto vivo que evoluciona a medida que el producto y el entorno en el que se usa o la tecnología cambian
2. El Product Owner es el responsable de gestionar el producto backlog. Esto implica aspectos como la creación y comunicación de sus elementos, asegurando que existe un entendimiento compartido por el resto de los miembros del Equipo Scrum, así como su orden. Cuanto más arriba esté un elemento dentro del Product Backlog, más importante o prioritario es.
3. El Product Backlog contiene al Product Goal u Objetivo de Producto, como compromiso asociado.
4. No hace falta que esté 100% definido para poder llevarlo a cabo. Debe tener la suficiente cantidad de características del producto para poder comenzar a funcionar.
5. No es un CONTENEDOR de las CARACTERÍSTICAS DEL PRODUCTO.
6. No hay tareas en el producto backlog.
7. A pesar de su popularidad, las historias de usuario son solo un posible formato para enunciar los elementos del Product Backlog. Algunos de los elementos que pueden ser válidos en el Product Backlog, como historias de usuario, errores o bugs, requerimientos y especificaciones, requerimientos no funcionales, casos de uso, trabajo técnico, spikes, etc.

Clase 05/10/23 – Aseguramiento de calidad del proceso y del producto. Testing de Software

Acá empieza clase PPQA (grabada)

Aseguramiento de calidad de proceso y de producto – PPQA



Aseguramiento de calidad vs Testing

El testing llega tarde porque el producto ya está hecho.

El espíritu del aseguramiento de calidad tiene que ver con hacer cosas antes. Trabajar como prevención para que el producto tenga embebida la mejor calidad que se pueda.

Definición de calidad

Es muy difícil de medir porque es un concepto subjetivo que tiene que ver con las expectativas y con las necesidades de cada una de las personas.

Lo más difícil son las expectativas porque es lo que está implícito.

“Todos los aspectos y características de un producto o servicio que se relacionan con la habilidad de alcanzar las necesidades manifiestas o implícitas.”

Tiene que ver con el producto, con el proyecto, con la gente.

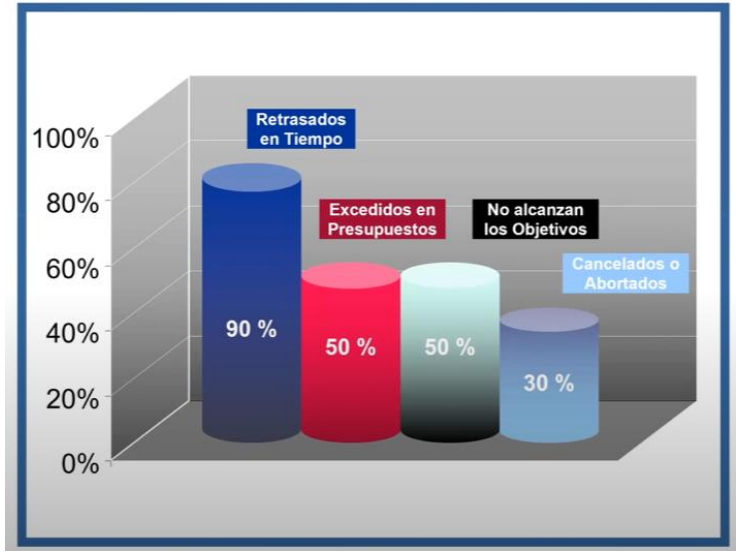
La calidad es relativa a las personas, su edad, las circunstancias de trabajo, el tiempo.

¿Qué cosas ocurren frecuentemente en los proyectos de desarrollo de software?

Cosas que hacen que el producto no tenga calidad. Problemas que hacen que la percepción del cliente o usuarios sea mala y produzca 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
- ¿Dónde está ese componente?

Situación de proyectos de software



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 básicos que sustentan la necesidad del aseguramiento de calidad

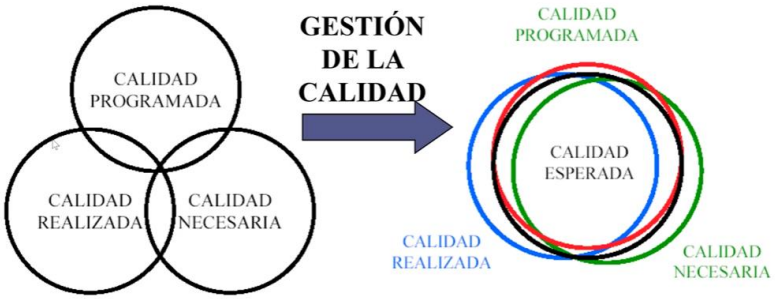
- La calidad no se inyecta ni se compra, debe estar ***embebida***.
- Es un esfuerzo de todos
- Las personas son la clave para lograrlo
    - Capacitación
- Se necesita sponsor a nivel gerencial
    - Pero se puede empezar por uno
- Se debe liderar con el ejemplo
- No se puede controlar lo que no se 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.

¿Calidad para quién?

- Visión del **usuario**. Que sea rápido, que no se clave, que tenga una linda interacción, fácil de usar.
- Visión de manufactura
- Visión del producto
- Visión basada en el valor
- Visión trascendental. Tiene que ver con el hecho de lograr cosas más allá de lo que uno se podría imaginar que tiene que hacer.

Desafío en la calidad

Lograr una coincidencia de estas 3 perspectivas de la calidad.



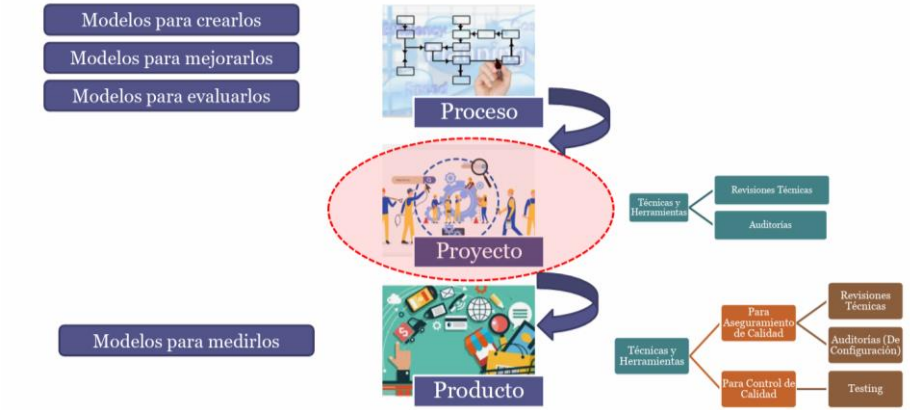
Cuando uno empieza a trabajar en un producto, resulta que tiene una expectativa de la calidad que le gustaría que ese producto tenga.

La calidad programada tiene que estar relacionada lo más que se pueda con la calidad necesaria que es lo mínimo que el producto debe tener para satisfacer los requerimientos del usuario.

La calidad realizada es lo que realmente hicimos por la calidad del producto.

La intersección de las tres debe ser lo suficientemente grande como para cubrirlas a las tres. De lo contrario, todo lo que realicemos por fuera del vinculo entre las tres perspectivas es desperdicio. Genera des-satisfacción en los clientes.

Calidad en el software



Existen muchos modelos para tomar de referencia para definir un proceso para una organización.

Una vez que esos modelos están incorporados y hemos creado nuestro proceso, si nosotros queremos funcionar en un ciclo de mejora continua, que existe independientemente del proceso elegido. Aparecen modelos que sirven para mejorar nuestros procesos.

Nos ayudan a definir proyectos para mejorar los procesos. Dar un marco de referencia a las organizaciones y equipos que quieren mejorar de manera continua sus procesos.

Modelos de evaluación intentan ver el grado de adherencia del proceso al modelo que tomaron de referencia.

Los procesos se materializan e instancian en los proyectos que son la unidad que les da vida a los procesos.

Mientras el proyecto se ejecuta ahí es donde si yo integro la calidad al momento en que se está haciendo el producto, voy a insertar actividades que me permitan a mi ver cómo se están haciendo las cosas en el contexto del proyecto. Para asegurar que estoy haciendo lo que dije que iba a hacer.

Técnicas y herramientas:

- Revisiones técnicas: son revisiones hechas por pares. Un par que tiene formación técnica similar. Las revisiones son a los productos.
- Auditorias: son objetivas, independientes y tiene que ser alguien externo al proyecto que determine si hay desviaciones de lo que el proyecto está haciendo conforme a lo que se comprometió que iba a hacer.

El proyecto es el ámbito en el cual uno trabaja para generar el producto. La versión del producto que yo voy a someter a evaluación se va generando en el contexto de un proyecto. También en el mismo contexto del proyecto, yo tengo que insertar tareas para asegurar la calidad del producto y para controlar la calidad del producto.

Hay muchas técnicas que se repiten. Puedo usar las revisiones técnicas como herramienta para hacer aseguramiento de calidad de producto.

Auditorias (de configuración) funcional y física.

Cuando hacemos aseguramiento de calidad tenemos que saber bajo qué nivel de aseguramiento está nuestro enfoque: si del producto o del proyecto.

DATO:

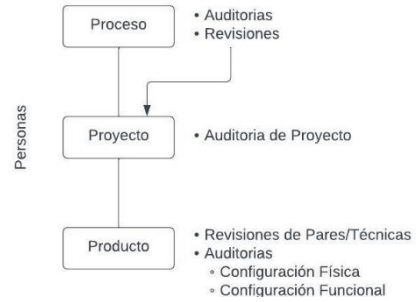
- La gente que apunta a trabajar con procesos definidos como modelos de calidad como el CMMI, trabajan con la base de que el proceso debe tener calidad, y si el proceso tiene calidad y la gente en el contexto del proyecto lo respeta, COMO CONSECUENCIA el producto que se obtenga también va a tener calidad. BASE FILOSÓFICA de cualquier modelo de calidad
- Base filosófica de cualquier modelo de calidad apunta a que la calidad del producto depende de la calidad del proceso que se usa para construirlo. No es LITERAL. El hecho de tener buenas bases definidas en el proceso no alcanza para conseguir la calidad del producto.
- La gente que apunta a trabajar con procesos empíricos, sobre todo los agilistas, sostienen que la calidad del producto depende o es gracias al equipo. Si el equipo hace lo que tiene que hacer, el producto va a tener calidad.

Contexto de Aseguramiento de calidad.

Cuando hablamos de aseguramiento de calidad, lo más importante es el aseguramiento. Prevención es más barata que la corrección. La idea de aseguramiento de calidad tiene que ver con incorporar acciones durante el proceso de construcción del software con un propósito, principalmente económico.

La forma en la que crece el costo de reparar el defecto según la edad del defecto crece exponencialmente. A mas edad, mas caro.

Apunta a un conjunto de acciones que podemos hacer para evitar los errores y evitar que se conviertan en defectos.



El testing es un proceso destructivo cuyo objetivo es encontrar defectos, es decir, poder asumir que una porción de software por mas trivial que sea, puede y VA a tener defectos y nuestro objetivo es encontrarlos para posteriormente poder resolverlos y poder contribuir a la calidad de un procesos de software.

Los defectos existen

Revisiones de Pares. APARECE EN EL FINAL!!!!!!!!!

Foco en el PRODUCTO. Aseguramiento de calidad del producto. La herramienta son las REVISIONES DE PARES. REVISIONES TÉCNICAS.

No las hace el jefe, ni el dueño, ni el cliente. Las hace alguien que técnicamente esta técnicamente igual de preparado que vos y no hay un vínculo jerárquico. Ni es un auditor.

La técnica por excelencia para hacer el trabajo de prevención.

Formales: Inspección de Software, proviene del lado del Hardware. Revisiones de pares formales, con procesos, métricas, roles

Informales: Walk through.

A qué cosas se les puede hacer revisiones técnicas. Código, diseño, documentación. A TODO. Todo es revisable por alguien más.

Las auditorias al agilismo no le gustan, las practicas de pares SI.

TDD. ADD. BDD. Mover el esfuerzo del detalle de los requerimientos hacia el testing. Automatizar.

La ejecución del testing se hace cuando el producto ya está construido.

El testing no es aseguramiento, sino que es control de calidad. Viene a hacer una actividad de verificación y validación.

Verifica que el sistema funcione correctamente y la validación se hace contra los requerimientos.

Detectamos defectos en el testing que ya se han trasladado desde la etapa de implementación.

Aseguramiento de calidad de procesos (Reordenar)

Modelos de calidad. Define las cosas que hay que hacer. Descriptivos. Interpretación del auditor.

Definir un proceso que sea compatible con el modelo de calidad elegido.

Se hace en dos niveles.

Auditorias y Revisiones se hacen a nivel de proceso primero y luego a nivel de proyecto.

“Declaramos que las cosas debían hacerse de cierta forma en el proceso y luego en el proyecto”

En el plan debería estar la declaración del proceso a utilizar.

Lo que se controla del proyecto, es el uso del proceso.

CMMI.

ISO 9001

Si el proceso que yo uso para crear el producto tiene calidad, entonces el producto que yo creo entonces también va a tener calidad

Calidad en el producto

Es importante entender que no se puede sistematizar, no hay modelos en la industria para evaluar el producto.

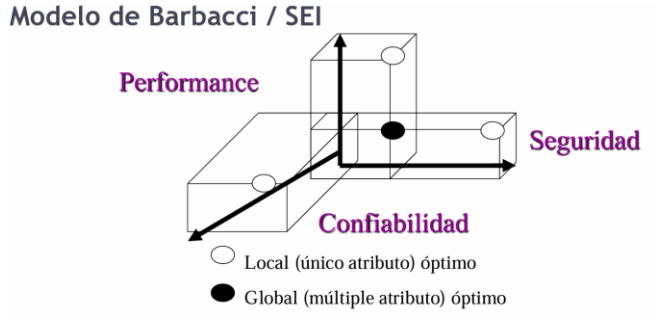
Modelos teóricos.

- ISO/IEC 25000. Modelos
  - Calidad en uso
  - Calidad de producto – ISO 25010. Requerimientos no func.
  - Calidad de datos

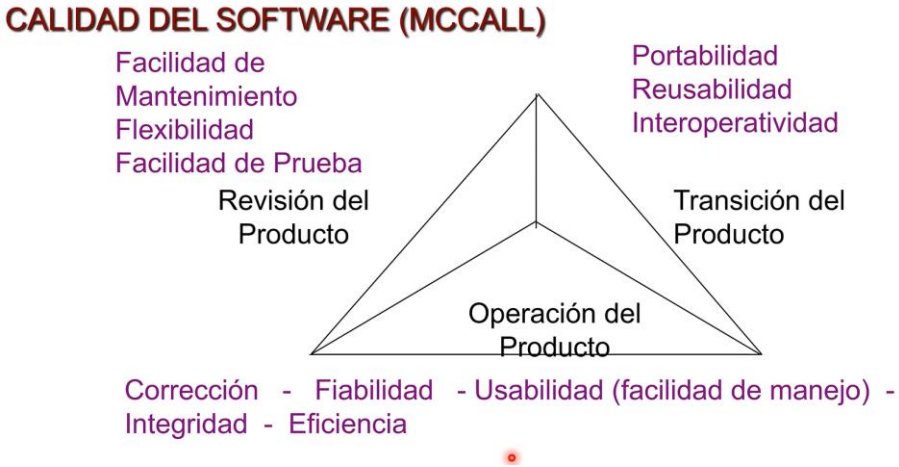
No hay en la industria modelos para acreditar o certificar calidad de producto. Es muy complejo, porque cada producto tiene sus características y cada producto está asociado a necesidades específicas.

Modelos de calidad de producto

- **Modelo de Barbacci/SEI**
  Evalua la calidad del producto con 3 elementos clave. Performance, confiabilidad, seguridad. Trabajar para lograr un equilibrio entre las 3 bases.



- **Modelo del software – McCall**
  La calidad tiene 3 grandes agrupadores que son factores determinantes que tienen que ver con las visiones de calidad
  - Una cosa son las características asociadas a la capacidad de verificación que tengamos del producto. Tienen que ver con el producto internamente.
  - Otras características tienen que ver con la capacidad del producto EN USO.
  - Factores que influyen para que el producto evolucione en el tiempo.



Calidad de producto

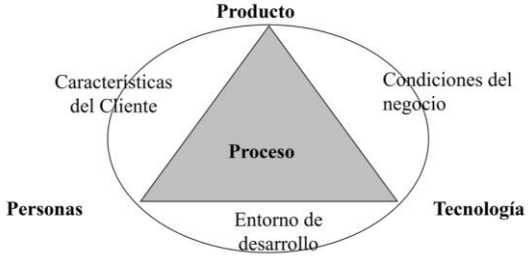
- Si queremos hacer aseguramiento lo hacemos con revisiones técnicas y auditorias. Revisiones técnicas son el elemento fundamental.
- Para controlar la calidad lo hacemos con el testing. Visibiliza que tan cerca o lejos estamos de los requerimientos que se habían planteado.

Calidad en el PROCESO

Problemática: depende del contexto. Lo que les sirve a unos, puede no servirles a otros.

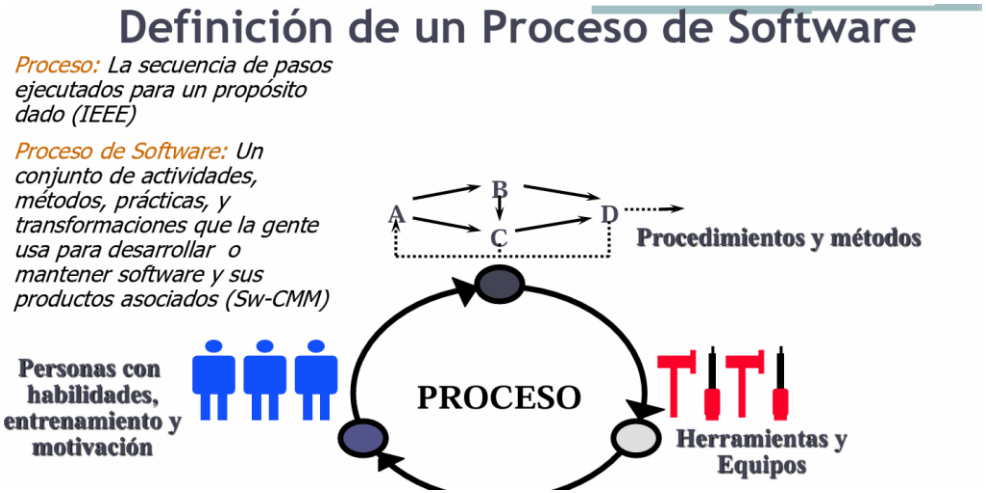
Hay que encontrar un proceso que sea util para un determinado contexto de trabajo en particular.

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



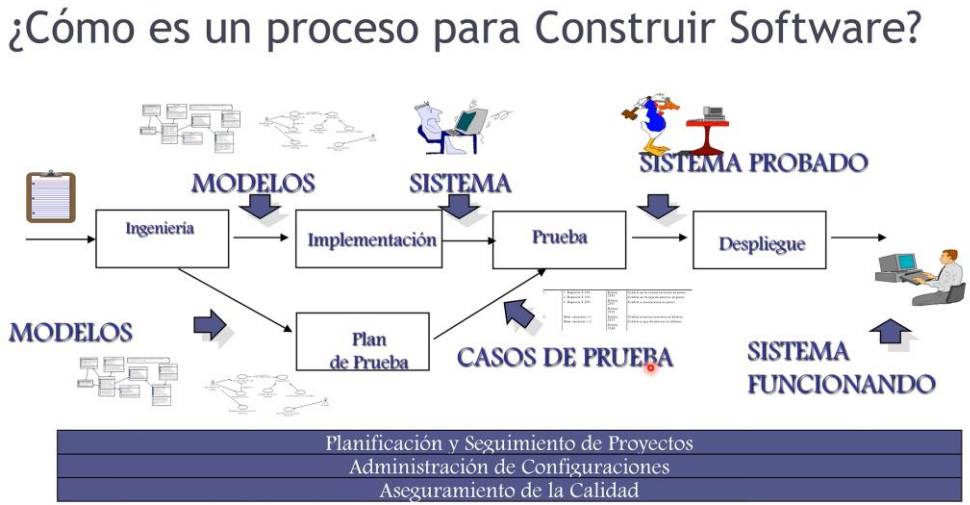
El único factor controlable que tenemos cuando estamos trabajando en un proyecto de desarrollo de software, es el proceso.

El proceso no es solamente las tareas escritas en algún lado, sino que se lo define como una mesa de tres patas donde si es cierto que tiene que haber lineamiento acerca de cómo vamos a trabajar, pero es igual de importante que quede plasmada la gente que va a hacer el trabajo con habilidades, con entrenamiento, con motivación. Y agregando herramientas y equipos.



¿Cómo lo definimos?

- Identificar etapas, workflows, disciplinas
- Etapas se subdividen
- Quienes deben participar
- Que deben hacer
- Cuando lo deben hacer
- Como lo deben hacer



- En el desarrollo de software participan:
  - Disciplinas de ingeniería
  - Disciplinas de gestión
  - Disciplinas de soporte

Disciplina: Aseguramiento de Calidad de Software.

¿Qué hace? ¿Qué implica?

Es insertar en cada una de las actividades acciones tendientes a detectar lo más tempranamente posible, oportunidades de mejora sobre el producto y sobre el proceso. Claramente es algo que tiene que estar en la cabeza y en la voluntad de todos.

Para materializarlo en forma concreta, cuando una organización quiere un grupo de aseguramiento de calidad, y lo quiere incorporar en una empresa deberían tener las siguientes consideraciones:

- No debería reportar al Gerente de Proyectos. Quita independencia y objetividad.
- 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 debería estar separada de la administración de proyectos para asegurar independencia.

Actividades de la administración de calidad de software.

- ***Aseguramiento de calidad***
  - Estándares y procedimientos organizacionales 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.

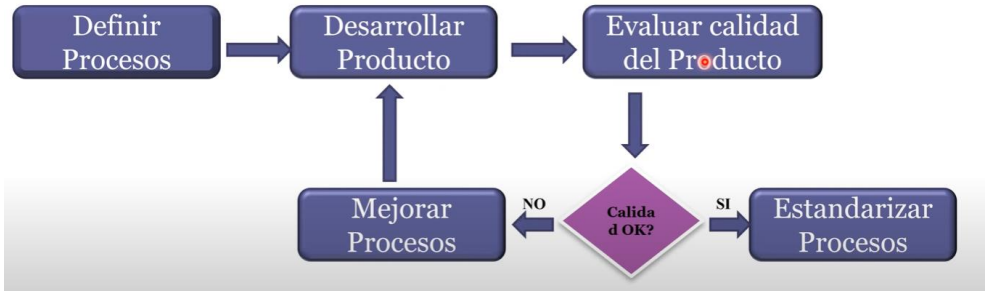
- ***Planificación de calidad***
  - Selecciona los procedimientos y estándares aplicables para un proyecto en particular y los modifica si fuera necesario.
  - Un plan de calidad define los productos de calidad deseados y cómo serán evaluados y define los atributos de calidad más significativos.
  - El plan de calidad debería definir el proceso de evaluación de la calidad
  - Define cuales estándares organizacionales deberían ser aplicados, como así también se es necesario utilizar nuevos estándares.

- ***Control de calidad.***
  - Asegura que los procedimientos y estándares son respetados por el equipo de desarrollo de software.
  - Implica el control del proceso de desarrollo para asegurar que se siguen los estándares y procedimientos
  - Existen 2 enfoques para el control de calidad.
    - ***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)
    - ***Evaluaciones de software automáticas y mediciones***

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.

Procesos basados en calidad



Modelo de mejora de procesos

Cuando hablamos de modelos de mejora nos referimos a esquemas, plantillas o recomendaciones de trabajo para encarar un proyecto de mejora de un proceso.

El propósito de los modelos de mejora es analizar el proceso que tiene la organización y armar un proyecto cuyo resultado, en lugar de ser un producto, que sea un proceso mejorado que se vuelca otra vez a la organización con la idea que mejoro el proceso y la calidad del producto final depende la calidad del proceso que yo uso para construirlo.

La motivación de la mejora de procesos tiene que ver con que si yo tengo un proceso mejor, mi producto resultante también va a ser mejor.

Algunos modelos para la mejora de procesos son:

- SPICE. Software Process Improvement Capability Evaluation
- IDEAL. Initiating, Diagnosing, Establishing, Acting, Learning

Modelo de mejora SPICE.

ISO 1504

Modelo de mejora IDEAL.

Nos sirve para definir en una organización o una unidad organizacional, un proyecto que ayuda a mejorar el proceso que esa empresa tiene.

- I, initiating
- D, diagnosing
- E, establishing
- A, acting
- L, learning

Modelo que plantea como nosotros podemos encarar un proyecto de mejora de procesos en una organización.

- **Inicio**: busca sponsoreo, un apoyo en la organización. Es muy importante ya que este tipo de procesos no son críticos en las empresas y siempre hay cosas más urgentes que estas. Si no tenemos un abal para ejecutar este proyecto en niveles altos de la organización, no solo por el financiamiento sino por garantizar que la gente le dedique tiempo. Este tipo de proyecto no suelena terminar exitosamente. Contexto delimitado
- **Diagnostico**: donde estoy parado en la organización, que cosas hay que mejorar y que cosas están bien. Analizamos a donde queremos ir. Análisis de brecha.
- **Establecer**: definir planes de acción, prioridades y estrategias.
- **Ejecución**: planear y ejecutar pilotos.
- **Aprendemos**: documentación.

Modelo de Calidad

Al momento de hacer el análisis de brecha. A donde estoy y a donde quiero llegar.

Ahí es donde entran los modelos de calidad, que se usan como referencia para bajar lineamientos que nuestro proceso tiene que cumplir para poder llegar a un cierto objetivo. Esos modelos que usamos para crear son los modelos de calidad.

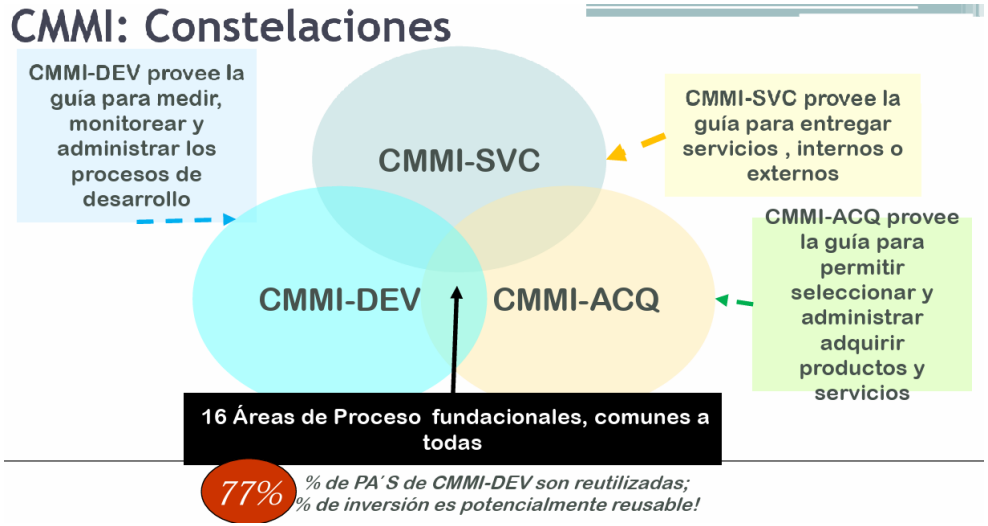
Modelo de Calidad: CMMI.

- Es la evolución del SW_CMM. Con el pedido de departamento de defensa de los estados unidos que estaban hartos de que los proyectos de defensa siempre fracasaran en el software.
- SEI fue el creador.
- El CMMI intenta una variante que sea más parecida a las normas de trabajo ISO como para capturar el publico que elegía ISO.
- Uno de los modelos más implementado en todo el mundo.
- No es una norma, y no se “certifica”, sólo se evalúa a través de profesionales reconocidos por el SEI como Lead Appraisers
- Modelo de referencia. Son buenas prácticas que dicen el QUÉ no el cómo. Da pautas para alcanzar objetivos.
- Me sirve para ver qué prácticas tiene que incluir mi proceso para poder alcanzar los objetivos que allí se definen.

Modelo de Calidad: CMMI - Constelaciones

3 dominios o ámbitos de mejora que les llama constelaciones.

- CMMI DEV. Provee la guía para medir, monitorear y administrar los procesos de desarrollo.

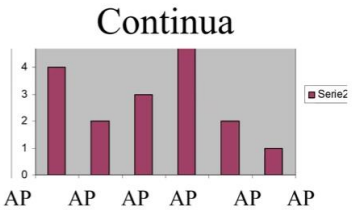


Modelo de Calidad: CMMI – Representaciones

Aparecen 2 formas de mejorar con el proceso.

- Por etapas: identificaba organizaciones y las dividía en 2 tipos:
  - Maduras. Nivel 2 al 5. A mayor madurez, mayor capacidad para cumplir con sus objetivos. Mejoraba la calidad de sus productos, reduciendo riesgos.
  - Inmaduras: organizaciones de nivel 1.
  - Facilita la comparación entre organizaciones
  - Definidos por un conjunto de áreas de proceso
  - Similar al SW-CMM
  - Provee una única clasificación que facilita comparaciones entre organizaciones
  - Provee una secuencia probada de mejoras.

  Ventaja de esta representación es que habla sobre la organización o área de la empresa que quiero evaluar.
- Continua: Elige áreas de proceso dentro de las que el modelo ofrece. (22 áreas de proceso) Yo elijo qué procesos quiero mejorar por separado. En vez de medir la madurez de toda la organización, lo que hago es medir la capacidad de un proceso en particular.
  - 6 niveles de 0 a 5. Permite identificar procesos que directamente no se realizan
  - En mi organización voy a tener procesos de nivel 0, que significa que ese proceso no se ejecuta.
  - Niveles indican la Capacidad de un Área de Proceso.
  - Similar a EIA/IS-731
  - Permite comparaciones sobre la base de cada AP
  - Permite elegir el orden de las mejoras.



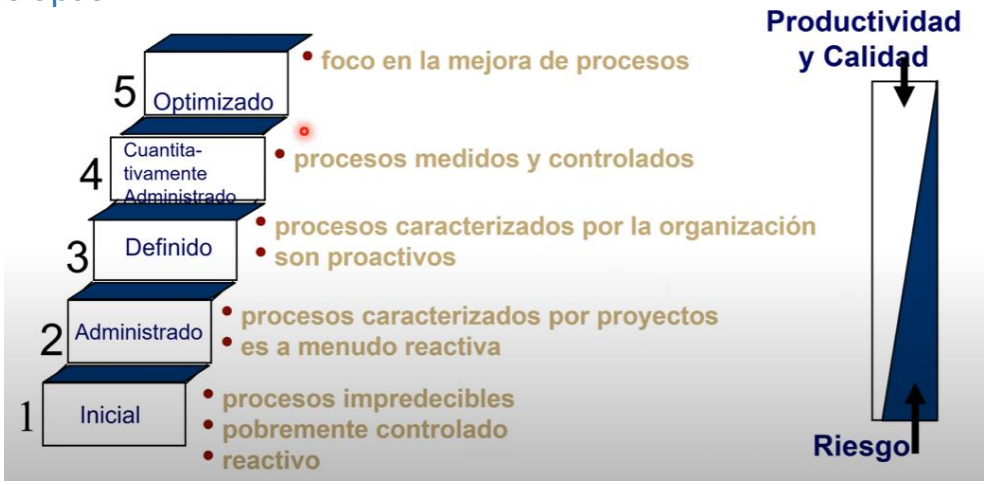
Modelo de Calidad: CMMI – Roles – Grupos

Cuando CMMI se refiere a grupo, hace énfasis en la existencia de roles que cubran ciertas tareas. Esos roles se adaptan al tamaño de la organización y a la cantidad de gente que tienen, expectativas de madurez que la organización quiera alcanzar.

Al nombrar la existencia de un grupo, tiene que ver con que haya gente que asuma el rol o los roles que se necesitan para ese grupo.

Lo importante es que existe alguien responsable de cubrir las actividades de cada uno de los roles o grupos.

Modelo de Calidad: CMMI – Niveles de la Representación por etapas



- Los niveles de madurez son acumulativos. Si sos nivel 5, significa que también sos nivel 4, 3, 2, 1.
- Nivel de inmadurez. No tenemos visibilidad sobre el proceso. No sabemos cuando vamos a terminar, no sabemos cuanto nos va a costar, no sabemos la calidad de lo que vamos a obtener. Gestionan crisis, no proyectos. Actitud reactiva ante las situaciones cuando ya se volvieron un problema.

- Nivel 2. Disciplinas de gestión y de soporte. El foco es tener el proceso administrado. Significa incorporar en las organizaciones la capacidad de gestión y de soporte. Ninguna de las áreas de proceso es de Ingeniería de producto. Los requerimientos los analiza desde el punto de vista de la gestión no del desarrollo.
  En este nivel se apunta a tener los requerimientos identificados, consistentes y controlados a lo largo del ciclo de vida del producto.
  - Única opcional: Administración de acuerdo con el proveedor. Si mi empresa no subcontrata otra empresa para que haga parte del producto, no aplica que yo la defina. Implica definir con mi proveedor como va a planificar los requerimientos, monitoreo, métricas, calidad. Cómo va a a ver el proveedor las cosas.

  Nivel 2: La empresa tiene madurez para administrar sus proyectos. El resultado de sus proyectos va a ser un producto de software en el cual sepamos qué esperar.

2 Administrado	▪Administración de Requerimientos (REQM) ▪Planificación de Proyectos (PP) ▪ Monitoreo y Control de Proyectos (PMC) ▪ Administración de Acuerdo con el Proveedor (SAM)	▪ Aseguramiento de calidad de Proceso y de Producto (PPQA) ▪ Administración de Configuración (CM) ▪ Medición y Análisis (MA)

Calidad en el Software. Comentario sobre CMMI y asignación de nivel

(Comentario que hizo la profe)

Nosotros definimos un proceso tomando como marco de referencia el nivel 2 de CMMI, encaramos un proyecto, obtuve el proceso. Si quiero llamar a una evaluación, necesito que ese proceso se use en los proyectos y generar evidencia.

La evidencia puede generarse de 2 tipos, objetiva y subjetiva. La evidencia subjetiva es lo que la gente dice y se puede evaluar por medio de entrevistas

a la gente del proyecto. Se contrasta con la evidencia física, que también es la evidencia objetiva.

Se evalúa y se tienen que cumplir todos los objetivos que tienen todas las áreas de proceso del nivel que queremos alcanzar.

Puntos clave

- El software puede analizarse desde varias perspectivas: como proceso, como producto. La calidad también
- La calidad del software es difícil de medir
- El software como proceso es el fundamento para mejorar la calidad
- Trabajar con calidad es más barato que hacerlo sin calidad.
- La mejora de proceso exitosa requiere compromiso y cambio organizacionales.
- Existen varios modelos disponibles para dar soporte a los esfuerzos de mejora.
- La mejora de procesos en el software ha demostrado retornos de inversión sustanciales.

CMMI cara a cara con Ágil.

Diferencias en ágil:

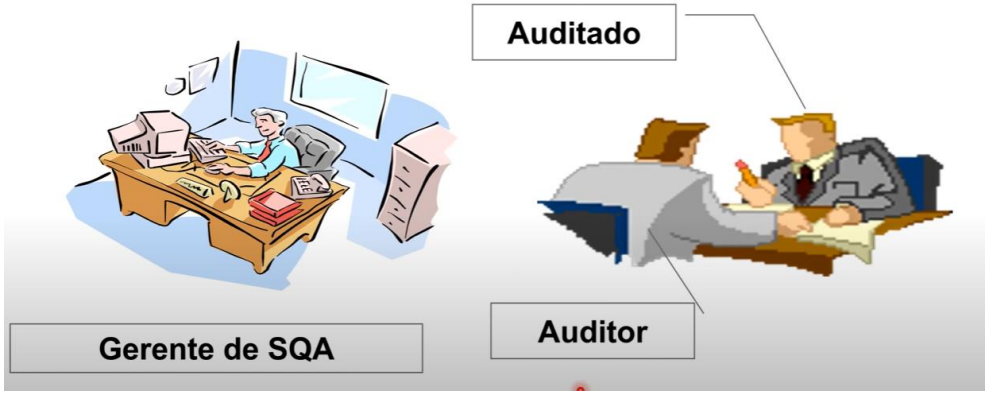
- Nivel 2: Objetivamente monitorear adherencia a los procesos y QA de productos y/o servicios. Apunta a la realización de auditorias que son las revisiones objetivas e independientes. Ágil no está de acuerdo con eso porque ágil resuelve todo con la gente que forma parte del equipo y no incorporamos a nadie del exterior.
- Nivel 3 y 4. Problemas con las métricas y performance de los procesos comparados entre equipos y entre proyectos. Ágil no soporta esto, ágil quiere software funcionando y al cliente contento. En ningún momento de ágil se evalúa si se cumple con el proceso que dijiste que ibas a cumplir. En CMMI si se evalua que se cumpla con lo propuesto en el proceso.

En el marco de desarrollo de software, hay 3 tipos de auditorías:

- Auditoria de proyecto
  - Valida el cumplimiento del proceso de desarrollo. Si el proyecto se ejecuto con el proceso que se dijo que se iba a ejecutar.
- Auditoria de Configuración funcional
  - Valida que el producto cumpla con sus requerimientos
- Auditoria de configuración física
  - Valida que el ítem de configuración tal como está construido cumpla con la documentación técnica que lo describe.

Roles

- Auditor: Una persona o dos que deben ser externos al proyecto.
- Auditado: alguien del equipo, en general suele ser el líder de proyecto. Puede haber alguien más.
- Gerente de SQA:
  - Prepara el plan de auditorias
  - Calcula el costo de las auditorias
  - Asigna los recursos
  - Responsable de resolver las no conformidades.



Proceso de auditoria

Las auditorias no son sorpresa.

- Planificación y preparación. Convocadas por el líder de proyecto. Se preparan y planifican en forma conjunta auditado y auditor.
- Ejecución: el auditor pide documentación y hace preguntar porque necesita evidencias. La objetiva y la subjetiva.
- Análisis y reporte del resultado: se analiza toda la documentación, se prepara un reporte y se le entrega a un auditado. Reporte final con cambios propuestos x el auditado que sería la versión final.
- Seguimiento: en el caso que el auditor siga haciendo un seguimiento hasta que las problemáticas encontradas

Tipos de resultados de auditorías

Existen 3 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. Cualquier cosa que no se hizo o cualquier cosa que no se hizo como el proceso dijo que había que hacerlo. No adecuando, necesita mejorar
- ***Observaciones***: sobre condiciones que deberían mejorarse, pero no requieren un plan de acción. Son practicas riesgosas.

Auditoria de configuración física y funcional de producto

Se hacen a una versión específica de un producto de software señalada en una línea base.

- Auditoria funcional: se encarga de validar.
- Auditoria física: se encarga de verificar.

Auditoria de proyecto

Existencia de un proceso adaptado que se está cumpliendo con lo que se había pactado en el plan.

Es la auditoria que se hace para ver si el proyecto está respetando el proceso que dijo que iba a usar.

Auditorías externas de calidad.

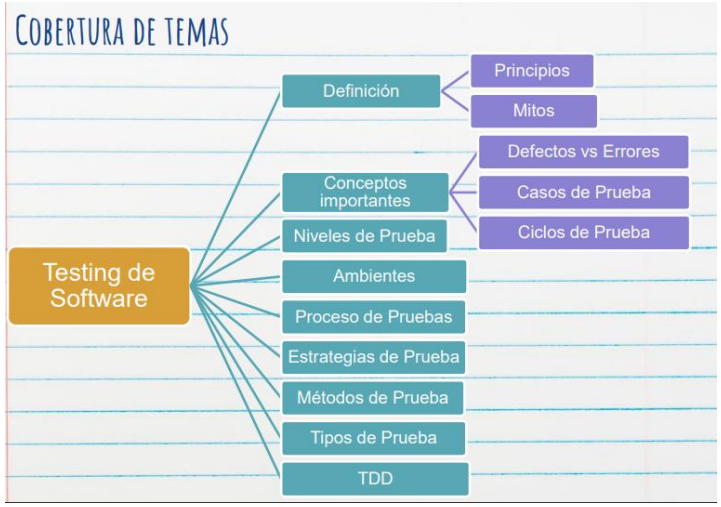
Puede ser de una unidad de negocio distinta no necesariamente otra empresa.

Por fuera del proyecto

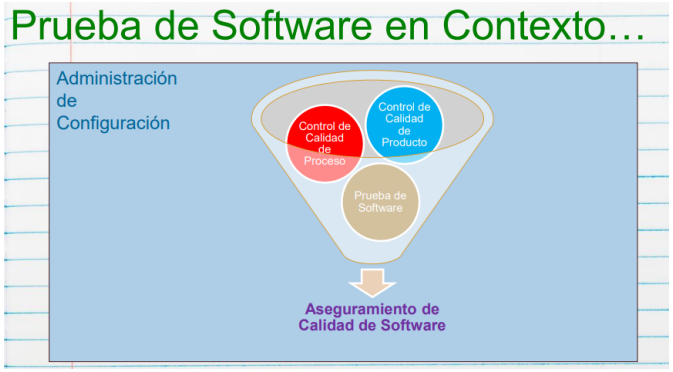
Auditorias de calidad

Clase 05/10/23 – Testing de Software

Temas relacionados al testing.



Prueba de software en contexto



Relación con la gestión de configuración.

Hay otras actividades que ayudan con la calidad, el testing no es la única. El testing está sujeto a un contexto. No es la única actividad que intenta encontrar defectos. Otras actividades que implementan las revisiones técnicas que es una forma de asegurar calidad en el producto y que ese tipo de actividad si se puede hacer a lo largo de todo el ciclo de vida en cualquier momento, pero cuando hablamos de testing, se realiza sobre el código NO a otra cosa.

¿Qué es hacer testing?

Proceso destructivo de tratar de encontrar defectos (cuya presencia se asume) en el código. Identificación de defectos.
Se debe ir con una actitud negativa para demostrar que algo es incorrecto.
Testing exitoso es el que encuentra defectos.
No esta bueno que la realiza la misma persona que lo hizo. No hay objetividad.
EXCEPCIÓN. Testing unitario. Es parte del trabajo de programar.
El testing no certifica nada.
Mientras más se automatice es mejor. Más precisión.
A lo largo del tiempo, es una actividad que ha sido menospreciada, subestimada. En el contexto de las metodologías agiles se le da más importancia.
Proceso destructivo que es exitoso cuando se encuentran defectos. Hacer revisiones.
Encuentra defectos cuya presencia se asume en el software. En la definición misma, estamos aceptando que el software tiene defectos.
El software lo crean las personas, las personas cometemos errores.
Hacer una actividad de testing y no encontrar defectos podría llegar a significar que fue un desarrollo exitoso, pero no sería un testing exitoso.
Si no se encontraron defectos es un indicador de que no se hizo un testing exhaustivo.
El testing visibiliza los defectos.
El éxito de un buen testing está sustentado en qué tan bien estuvieron diseñados los casos de prueba y qué tan bien están conformadas las BD para hacer pruebas.
Nunca se asegura una cobertura del 100%.(Porcentaje de pruebas que se hacen sobre el total de pruebas que deberían hacerse para cubrir el total de las funcionalidades.) Cobertura en sentido de condiciones posibles y la combinatoria al ser tan grande, no puede cubrirse completa. Es inviable en términos de costo y tiempo.

Aseguramiento vs Control de calidad

La intención de las actividades de aseguramiento de calidad es la prevención y la detección temprana, idealmente errores. Buscando que no se conviertan en defectos. Es un producto que llega en mejores condiciones al testing, muy probable que se encuentren menos defectos.

Testing es una actividad de control de calidad, porque revisa algo que fue previamente realizado.

Error vs Defecto

Momento en el que se advierte.
Cuando encuentro un problema en la misma etapa en la que se produjo es un error.
Si se trasladó a la etapa siguiente, ya es un defecto.
Testing encuentra defectos.

Un defecto ocasiona una falla según la magnitud.
Ambos son fallas.
El testing encuentra defectos.

Defectos, Severidad y Prioridad

Severidad: Cuán grave es el defecto que se encontró. Impacto. Viene asociado a un contexto. La define el encargado de pruebas. Margen de tiempo que tiene la empresa para corregirlo. Evalúa que tanto daño le hace el software, ese defecto.

1. Bloqueante. No anda el sistema.
2. Critico
3. Mayor. Mayor problema para distinguirlo.
4. Menor. Mayor problema para distinguirlo
5. Cosmético. 2 semanas. Cuestiones de interfaz, visualización, presentación de ciertos datos, como fechas, sexos.

Prioridad: Decide el PO. Viene del lado del cliente. La define el cliente. Evalúa que tan importante es para el negocio esa resolución de defecto.

1. Urgencia
2. Alta
3. Media
4. Baja

Niveles de Pruebas

El foco en un componente o en armar un build, o en un incremento de producto.

- Pruebas **unitarias**. Las hace el desarrollador. Tendencia hacia TDD. TDD no es testing en sí. Su objetivo no es encontrar errores ni defectos sino de proveer un mecanismo de verificación de ….
  Nivel más bajo y que encuentra errores. Ambiente de desarrollo. No hay un traslado.
- Pruebas de **sistema**. Conocido por los ciclos de vida iterativo o incremental. Testing de una versión del producto. También llamadas pruebas de versión. Testing de los requerimientos de esa porción del producto. Apunta a los principios del testing que dice que un desarrollador no debería probar su propio código. Workflow de implementación.
- Pruebas de **aceptación**. Prueba de aceptación de usuarios. UAT User acceptance test. Usuario. En el workflow de despliegue. En la REVIEW si usamos scrum. Tiene que involucrarse el usuario.
- Pruebas de **integración**. Prueba de interfaces. El foco lo tiene en la integración de componentes. Automatizado es el continuos integración. Disputada. Aparece como una actividad del workflow de testing. Uso de prácticas de integración continua. Obtenemos un build que está en condiciones de ser la entrada al nivel de prueba de sistemas.

Ambientes para la construcción del Software.

Al ambiente de testing, los desarrolladores NO tienen acceso. Tienen que estar separados porque uno tiene que llegar con una versión limpia para probar.

Tiene que ver con evitar que un programa deje de funcionar al trasladarlo de ambiente. Situación frecuente que ocurre que cuando instalamos el software en un dispositivo distinto al que se construyó, haya un error de configuración.

No siempre se suele tener los 4 ambientes.

- Desarrollo
- Prueba. Se crea para realizar las pruebas de sistemas, son preparados con datos de prueba y quien hace las pruebas allí son los de testing.
- Pre-Producción. Pruebas de aceptación de usuario. Costoso. Ámbito en el cual el producto se ejecuta. Deberían hacerse las pruebas de aceptación.
- Producción. Peligroso hacer acá las pruebas de usuario. Ámbito en el cual el producto se ejecuta.

A veces es costoso simular ambientes casi reales.

¿Por qué es conveniente tener un ambiente de desarrollo separado del de pruebas? Es una cuestión de ingerencias. Ambiente de sala limpia en el cual los desarrolladores no puedan intervenir.

Al menos se deberían garantizar 3 ambientes de producción.

Caso de Pruebas.

Artefacto o unidad de trabajo de testing más importante. Se diseñan

Set de condiciones o variables bajo las cuales un testes determinara si el software está funcionando correctamente o no.

La idea de probar utilizando casos de prueba nos permite reproducir los defectos. Un defecto que no se puede reproducir no es un defecto.

Un caso de prueba nos ayuda porque tenemos registrado el paso a paso de acciones que hice.

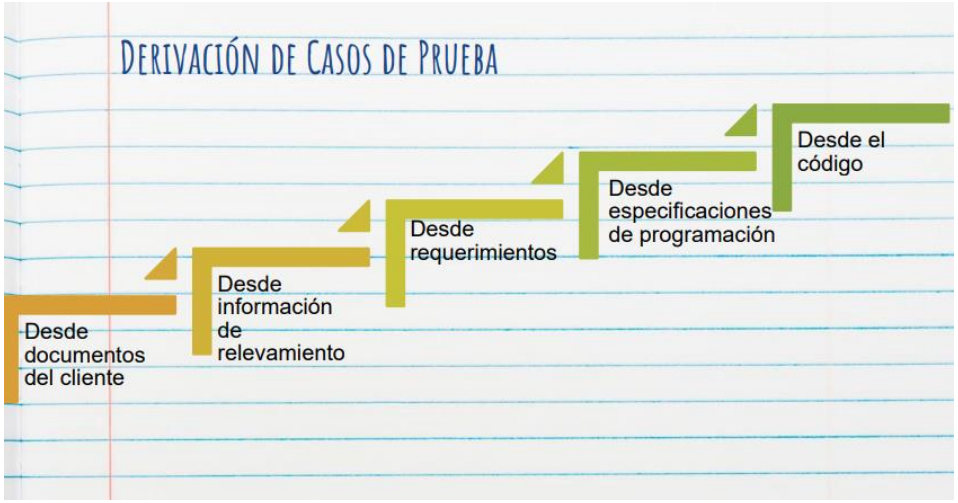
Me permite lograr la posibilidad de reproducir defectos y hace que una prueba sea sistemática.

Un caso de prueba tiene una serie de elementos muy importantes que conforman la construcción del caso de prueba y se toman los criterios de aceptación que tienen las US se toman como base para seber qué condiciones de prueba voy a tomar en cada escenario en particular.

Un caso de prueba es un escenario concreto con datos específicos que intenta probar un conjunto de situaciones o criterios de aceptación o de condiciones de prueba.

Cuando yo hago testing, espero como expectativa, hacer verificación y hacer validación. La validación es si es el producto correcto o no. La verificación es si el producto funciona correctamente.

¿Cuál es el lugar ideal para sacar información para hacer casos de prueba?



Desde los requerimientos. Vamos a ver si el producto se está comportando como dicen los requerimientos o no.

Si se saca del código, no es ideal porque no se puede hacer validación si se diseñan los casos de prueba desde el código. No hay forma de saber si el producto está haciendo lo que debería hacer. Solo se puede hacer verificación.

DEFINIR LOS CASOS DE PRUEBA CONTRA LOS REQUERIMIENTOS. Definen lo que el sistema tiene que hacer.

CONCLUSIONES

- Ninguna técnica es completa
- Las técnicas atacan 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 conjetura de defectos
- Ser sistemático y documentar las suposiciones sobre el
- comportamiento o el modelo de fallas

Condiciones de Pruebas

Definir seteo, contexto.

Información adicional que me hace falta para que la prueba funcione

Base de datos, permisos.

Estrategias de prueba

Métodos que nos ayudan a diseñar casos de prueba con la premisa de que queremos la menor cantidad de casos de prueba que sean lo más eficiente y efectivo posible.

Caja Negra. Si se obtienen las salidas esperadas, TODO OK. Testing dinámico.

- Basados en especificación
  - Partición de equivalencias
  - Análisis de valores limites
  - Etc.
- Basados en la experiencia. Empíricas puras, se basan en la experiencia y el conocimiento de la gente que va a hacer testing.
  - Adivinanza de defectos
  - Testing exploratorio.

Caja Blanca. El código es visible. Se basan en el análisis de la estructura interna del software o un componente de software. Testing estático. No ejecuta el código, mira el código por dentro para ver complejidad, loops infinitos, uso de palabras reservadas, ciclos sin fin, errores de sintaxis.

Se puede garantizar el testing coverage.

Ciclos de Test

Un ciclo de pruebas abarca la ejecución de la totalidad de los casos de prueba aplicados a una misma versión del sistema a probar.

El ciclo de prueba empieza con la corrida del primer caso de prueba y termina cuando se ejecutaron todos los casos de prueba que estaban previstos o cuando existe un error invalidante que impide que ese ciclo de prueba continue y el ciclo se interrumpe en ese momento.

- Primer ciclo de test que se ejecuta. CICLO CERO. Siempre es manual.

Hay casos de prueba positivos y negativos. Cuando corremos los casos de prueba vamos registrando los resultados. Al final del ciclo, se reportan todos los defectos que encontramos con la documentación más clara posible de manera tal que ese error se pueda reproducir. Un error que no se puede reproducir no es un error.

Regresión

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.

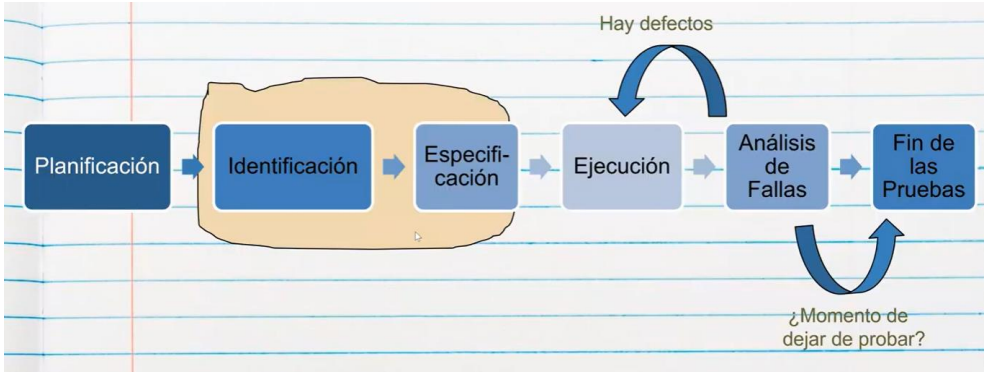
Relación con el concepto de ciclos de prueba

- Sin **regresión**. Solo revisa los defectos que antes habían salido mal. Es la que más se usa. Errores ocultos pueden ocurrir.
- Con **regresión**. Prueba todo de nuevo como si fuese ciclo 0 cada vez. Todos los ciclos se comportan como ciclo 0. Es la mejor. Mas costosa

Proceso de Pruebas

Una prueba sistemática o metodológicas vs. Pruebas ad hoc

Una prueba sistemática o metodológica es una prueba que usa un proceso. Un proceso de prueba implica de las siguientes actividades:



Parte amarilla: también conocida como diseño de las pruebas.

Qué se diseña: casos de prueba con todos los datos necesarios para dejarlo listo para las ejecuciones

Ejecución: tomar la versión del producto que me dan y correr los casos de prueba y obtener como resultado un reporte de defectos.

Análisis de fallas: consiste en el análisis del reporte y se vuelve a la ejecución.

Entregables de Pruebas

- Plan de testing
- Documento de casos de prueba
- Reporte de incidentes
- Informe Final.

Modelo en V

Se prueba de forma inversa a la que se desarrolla. Se desarrolla de lo general a lo particular. Se empieza probando a nivel de prueba de unidad y se va escalando en los niveles.

Uno puede adelantar tareas de testing mientras el producto se está construyendo.

Cuanto testing es suficiente

Una de las cosas más difíciles es definir el criterio de finalización del testing. El testing no puede demostrar de ninguna manera ausencia de defectos. No hay una forma de demostrar que los defectos que encontramos son todos los defectos que el producto tiene.

Existen algunos criterios para definir cuando terminar el proceso de testing.

Algunos son:

- Se termina el tiempo para testear
- Proyección de defectos que cuando se alcanzan la proyección de defectos encontrados ahí se termina.
- Estándares de calidad en cuanto a la severidad. Ejecutamos un conjunto de casos de prueba para mi ciclo y si en ese ciclo no hay ningún requerimiento de testing nuevo no planificado el testing puede finalizar.

Principios del testing

- El testing muestra presencia de defecto
- El testing exhaustivo es imposible
- FALTA COMPLETAR

Testing de Caja Blanca

Métodos en los cuales, si disponemos de los detalles de la implementación, podemos ver el código y en base a eso poder diseñar nuestros casos de prueba.

Permiten diseñar casos de prueba, maximizando la cantidad de defectos encontrados con la menor cantidad de casos de prueba posibles disponiendo de los detalles de la implementación.

- Se basa en el análisis de la estructura interna del software o un componente del software .
- Se puede garantizar el testing coverage.

Coberturas: Estrategia para recorrer los distintos caminos que nuestro código nos provea para desarrollar una funcionalidad.

- ***Cobertura de enunciados o caminos básicos***

  Propuesto por McCabe

  Permite obtener una medida de la complejidad de un diseño procedimental, y utilizar esta medida como guía para la definición de una serie de caminos básicos de ejecución.

  Busca garantizar que a todos los caminos independientes que tenga nuestra funcionalidad los vamos a recorrer al menos una vez. Vamos a pasar por todos los caminos independientes que esta funcionalidad tenga.

  Si nosotros diseñamos un conjunto de casos de prueba que corresponda a esta cobertura de enunciados, podemos decir que garantizamos que nuestro código ha sido ejecutado pasando por todos los caminos independientes.

  Para la prueba del camino básico:

  - Se requiere poder representar la ejecución mediante grafos de flujo
  - Se calcula la **complejidad ciclomatica**: es una métrica de software que provee una medición cuantitativa de la complejidad lógica de un programa. Usada en el contexto del testing, define el numero de caminos independientes en el conjunto básica y entrega un limite inferior para el numero de casos necesarios para ejecutar todas las instrucciones al menos una vez.

Complejidad Ciclomática	Evaluación del Riesgo
1-10	Programa Simple, sin mucho riesgo
11-20	Más complejo, riesgo moderado
21-50	Complejo, Programa de alto riesgo
50	Programa no testeable, Muy alto riesgo

  - Dado un grafo de flujo, se pueden generar casos de prueba.

- ***Coberturas de sentencias***

  Ejemplo:

  ¿Cuántas sentencias hay en esta funcionalidad?

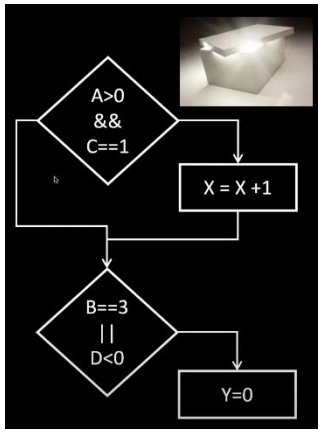
  2 sentencias

  Sentencia es:

  - Instrucción como asignacion de variables o invocacion de métodos
  - Mostrar un mensaje
  - Lanzar una excepcion

  NO son esctructuras de constrol.

```
IF (A>0 && C==1)
        X = X + 1
IF (B==3 || D<0)
        Y=0
END
```



  El objetivo de la cobertura de sentencias es darle cobertura a todas las sentencias. Buscamos cual es la cantidad minima de casos de prueba que me permiten pasar o ejecutar o recorrer todas las sententas.

  Cantidad minima de casos de prueba que me permiten recorrer las sentencias en este ejemplo: 1 solo caso de prueba

- ***Cobertura de decisión***

  Una decisión es una estructura de control completa.

  Atención en: paréntesis en los IFs.

  Se debe cubrir por el verdadero y por el falso.

  Mínima cantidad de casos de prueba para poder probar todas las decisiones que tengo ya sea para forzar la rama del true o forzar la rama del false.

  Independientemente de si por las ramas del false no haya sentencia.

  Se prueba que las decisiones funcionen.

- ***Cobertura de condición***
  Las condiciones se encuentran dentro de las decisiones, separados por operadores lógicos.
  Condiciones son las evaluaciones lógicas que están unidos por operadores lógicos adentro de una decisión.

  Busca encontrar la menor cantidad de casos de prueba que nos permiten valuar cada una de las condiciones tanto en su valor verdadero como en su valor falso, independientemente de por donde salga la decisión.

  No consideramos los IF que son cortocircuitados.

- ***Cobertura de decisión/condición***
  Busca no solamente valuar las decisiones en su valor verdadero y falso, sino también valuar todas las condiciones en su valor verdadero y falso.

- ***Cobertura múltiple***
  Busca valuar el combinatorio de todas las condiciones en todos sus valores de verdad posibles.
  Planteamos el combinatorio de todos los valores de verdad para las condiciones que tenemos disponibles.
  Posibilidad de casos de prueba imposibles

- Etc

Clase 12/10/23 – Filosofía Lean y Kanban

Filosofía Lean

Basada en 7 principios. Tiene un origen en la industria automotriz Toyota. Inicio en Japón en los años 40. Nada se opone a Ágil. Lean surgió primero.

Nace en industrias tangibles a diferencia del agilismo que ya inició en industrias intangibles.

Ilustración 1 – Principios Lean Software Development (LSD)

Principio: Eliminar Desperdicios

- **Eliminar desperdicios**. Evitar que las cosas se pongan viejas antes de terminarlas o evitar retrabajo. (Tiene que ver con el principio ágil de Software Funcionando y el de simplicidad, arte de maximizar lo que no hacemos).
  Es uno de los principios que mas **fuerza** tiene en esta corriente y apunta a mejorar la productividad, y por consiguiente la calidad también en lo que se esté haciendo.
  Desperdicio: todo lo que no genera valor.
  Ej.: tomarnos el tiempo de especificar mucho los requerimientos es uno de los mayores desperdicios porque suelen cambiar.

  Reducir el tiempo removiendo lo que NO agrega valor.
  - Desperdicio es cualquier cosa que interfiere con darle al cliente lo que el valora en tiempo y lugar donde le provea más valor.
  - En manufactura: el inventario. Logística y stock. El gasto de mantener un inventario es un gasto muy grande. La idea es no tener mucho inventario.
  - En software: es el trabajo parcialmente hecho y las características extra.
  - El 20% del software que entregamos contiene el 80% del valor. Hacer características que nadie usa, es un desperdicio.

Los siete desperdicios Lean en Software.

- **Características extra**. Requerimientos que nadie solicito. Inventario
- **Trabajo a medias**. Cosas parcialmente terminadas. Procesos empíricos apuntan a gestiones binarias. Procesos mas chicos, que puedan terminarse o no. No cuenta y genera desperdicios.
- **Proceso extra**. Pasos innecesarios extras

- **Movimiento**. Trabajo desagregado en distintas partes del mundo. Complicaciones de acceso, tardanza en comunicación. Se da en equipos distribuidos que están locados en lugares físicos distintos.
- **Defectos**. Ocasiona retrabajo. Factores de costo más alto
- **Esperas**. Información que la necesitamos de otro equipo o un proveedor. Dependencias con otros equipos. Esperas de aprobación.
- **Cambio de Tareas**. Multitasking, switchear entre una cosa y otra. Resulta muy improductivo. Problema en nuestra industria.
- **Talento no utilizado o subutilizado.** Mala asignación de roles de trabajo. No aprovechar los conocimientos de cada miembro del equipo.

Principio: Amplificar el aprendizaje

Relacionado con crear y mantener una cultura de mejoramiento continuo y solución de problemas. Conocimiento continuo y compartido con el resto del equipo***. Hacer explicito un conocimiento implícito***. El conocimiento sirve si yo lo comparto. Trabajo en equipo, transparencia.

- Un proceso focalizado en crear conocimiento esperará que el diseño evolucione durante la codificación y no perderá tiempo definiéndolo en forma completa, prematuramente
- Se debe generar nuevo conocimiento y codificarlo de manera tal que sea accesible a toda la organización.
- Muchas veces los procesos estándares hacen difícil introducir en ellos mejoras.

Principio: Embeber la integridad conceptual

Relacionado con la calidad, trabajamos para entregar un producto y la calidad es algo no negociable. Debe estar implícito en el trabajo que tenemos que hacer.

Relacionado con el principio: excelencia técnica.

Encastrar todas las partes del producto o servicio, que tenga coherencia y consistencia (tiene que ver con los requerimientos No funcionales). La integración entre las personas hace el producto más integro.

Se necesita más disciplina, no menos.

- Integridad percibida: El producto total tiene un balance entre función, uso, confiabilidad y economía que le gusta a la gente.
- Integridad Conceptual: todos los componentes del sistema trabajan en forma coherente en conjunto.
- El objetivo es construir con calidad desde el principio, no probar después.
- Dos clases de inspecciones:
    - Inspecciones luego de que los defectos ocurren
    - Inspecciones para prevenir defectos
- Si se quiere calidad no inspeccionar después de los hechos
- Si no es posible, inspeccione luego de pequeños pasos.

Principio: Diferir compromisos

El ultimo momento responsable para tomar decisiones ***(en el cual todavía estamos a tiempo)***. Si nos anticipamos, tenemos información parcial.

- Se relaciona con el **principio ágil**, decidir lo más tarde posible, pero responsablemente. No hacer trabajo que no va a ser utilizado. Enlaza con anterior de aprendizaje continuo, mientras mas tarde decidimos más conocimiento tenemos.
- No significa que todas las decisiones deben diferirse
- Se deben tratar de tomar decisiones reversibles, de forma tal que pueda ser fácilmente modificable
- Vencer la parálisis del análisis (por la perfección) para obtener algo concreto terminado.
- Las mejores estrategias de diseño de software están basadas en dejar abiertas opciones de forma tal que las decisiones irreversibles se tomen lo más tarde posible.
- Postergar asumir un compromiso hasta tener más información.
- Ejemplo: la elicitacion de requerimientos. Asignación de trabajo.

Principio: Dar poder al equipo

Equipos que puedan tomar decisiones, formados, capacitados, motivados que se autoorganicen y se gestionen.

- Respetar a la gente
    - Entrenar lideres
    - Fomentar buena ética laboral
    - Delegar decisiones y responsabilidades del producto en desarrollo al nivel mas bajo posible.
    - Libertad de acción
    - Ágil: el propio equipo puede estimar el trabajo. Equipos auto gestionados.

Principio: Ver el todo

Relacionado con la entrega de valor, es decir, no solo ver las líneas de código ni la pantalla de la computadora sino ver el valor que aporta el software como herramienta.

Tener una visión holística, de conjunto (el producto, el valor agregado que hay detrás, el servicio que tiene los productos como complemento.)

Visión completa del proceso y producto.

Practicas que permitan conocer el flujo de trabajo, donde hay visibilidad.

Principio: Entregar rápido

Estabilizar ambientes de trabajo a su capacidad más eficiente y acotar los ciclos de desarrollo.

Entregar rápidamente esto hace que se vayan transformando n veces en cada iteración. Incrementos pequeños de valor. Llegar al producto mínimo que sea valioso. Salir pronto al mercado.

Relacionado con el principio ágil de entrega frecuente.

Dar **retroalimentación** al cliente de forma que entregue valor.

KANBAN en el Desarrollo de Software

- Framework que sirve para la mejora continua de procesos basada en el pensamiento lean.

- 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, el diseño tanto de productos de software como físicos.

- El método fue formulado por David Anderson.

- Es un enfoque para gestión de cambio

- ***No es un proceso de desarrollo de software o una metodología de administración de proyecto. Menos descriptivo que Scrum, no enseña a hacer software.***

- Kanban es un método para introducir cambios en un proceso de desarrollo de software o una metodología de administración de proyectos. Mejoras evolutivas. Pueden ser productos o servicios.

- **Busca**: la visualización de flujos de trabajo.

- kan-ban es una señalización de algo, que nos sirve para saber en qué estado está el proceso de producción de la pieza que le queremos entregar al cliente.

- El sistema de trabajo Kanban está basado en la teoría de colas.

- A través de la administración de las colas, plantean la visualización de todo el proceso e ir viendo el avance.

- Kanban fomenta la evolución gradual de los proceso existentes. Empezar con lo que se tiene, no hacer grandes cambios.

- Kanban no pide una revolución, sino que fomenta el cambio gradual

- Kanban está basado en una idea muy simple, limitar el trabajo en proceso

- Sistemas de arrastre: no hay nadie que te busque en tu columna, sino que vos buscas el trabajo que vos consideras que podés hacer. Auto asignación.

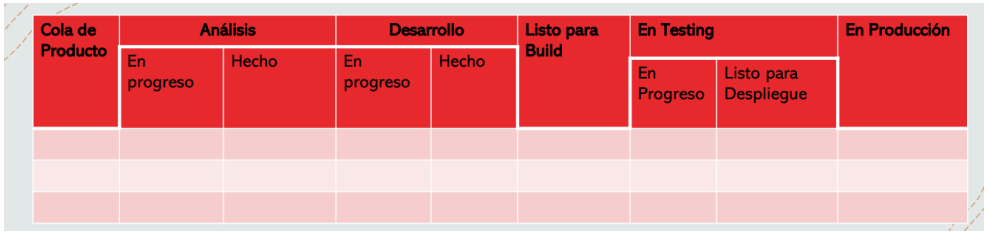
- PULL, NO PUSH.

- El Kanban implica que una señal visual se produce para indicar que el nuevo trabajo se puede tirar (pull) porque el trabajo actual no es igual al limite acordado.

TABLERO DE KANBAN.

Ejemplo de tablero:

kan-ban: atraviesan el proceso.



¿Cómo aplicar Kanban?

1. Empezar con lo que se tiene ahora
2. Entender el proceso actual
3. Acordar los límites de WIP para cada etapa del proceso
4. A continuación, comienza a fluir el trabajo a través del sistema tirando de él, en presencia de señales Kanban.

Proceso: Modelar nuestro proceso en un tablero

Trabajo: decidir la unidad de trabajo.

- Requerimientos
- Defectos
- Desarrollo
- Solicitudes

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. DOD, para cada estado

Cadencia: releases, planificaciones, revisiones.

Principios de Kanban

- ***Gestión de cambios***
  - Comenzar con lo que se tiene ahora: Entender los procesos actuales tal y como están siendo realizados en la actualidad. Respetar los roles actuales, las responsabilidades de cada persona y los puestos de trabajo.
  - Acordar la búsqueda de la mejora a través del cambio evolutivo
  - Fomentar actos de liderazgo a todos los niveles
- ***Entrega de servicios***
  - Comprender y enfocarse en cumplir las necesidades y expectativas del cliente
  - Gestionar el trabajo, dejar que los trabajadores se auto organicen.
  - Revisar periódicamente la red de servicios y sus políticas para mejorar los resultados entregados

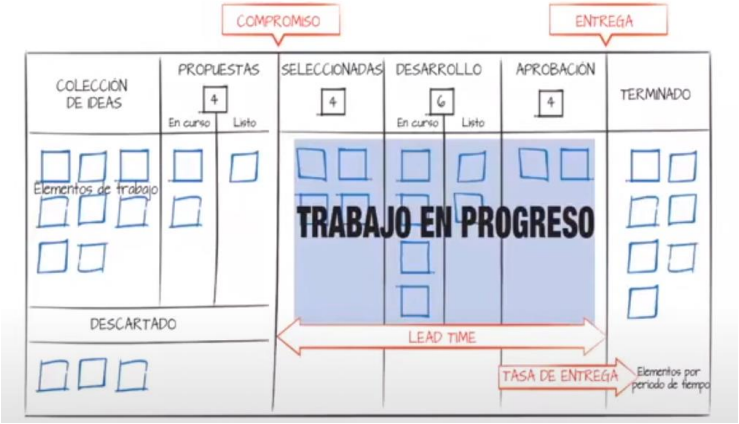
Prácticas generales de Kanban

Definen el QUÉ. Apuntan a cambios culturales.

1. ***Visualizar el flujo. Hacer el trabajo visible***
   Muestra el trabajo y su flujo
   Visualiza los riesgos
   Construye un modelo visual que refleje como se trabaja
   Dividir el trabajo en piezas, las US son buenas para eso.

Distribuir el trabajo en las columnas: el trabajo fluirá de izquierda a derecha en las columnas.
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.
Bajar a tierra la información que tenemos.

Columnas de trabajo y acumulación.



2. ***Limitar el trabajo en progreso. WIP***
   Limita el trabajo en el sistema a la capacidad disponible, basándote en los datos.
   Asignar limites explícitos de cuantos ítems puede haber en progreso en cada estado del flujo de trabajo.
   Limitar el trabajo en estado: EN PROGRESO.
   Capacidad máxima de trabajo en progreso en un determinado tiempo.
   Asignar límites explícitos de cuántos ítems puede haber en progreso en cada estado del flujo de trabajo.

3. ***Administrar el flujo, ayudar a que el trabajo fluya***
   Al 100% de la capacidad, se tiene un rendimiento mínimo.
   Evitar atorarse de tareas.
   - Definir tipos de trabajo: Asignando capacidad en función de la demanda.
     - Requerimientos
     - Defectos
     - Desarrollo
     - Solicitudes
   - Administración de colas. Sistema de arraste, PULL.

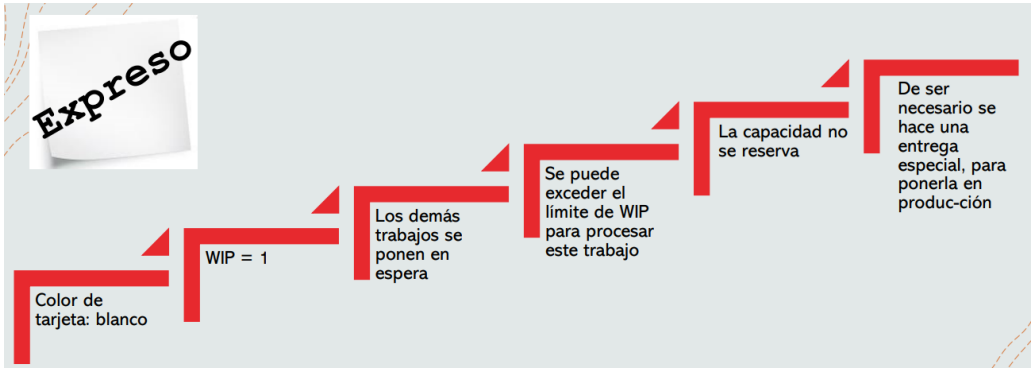
4. ***Hacer explicitas las políticas***
   Todas las políticas deben ser acordadas entre todas las partes involucradas, incluyendo a los clientes, interesados y trabajadores responsables del trabajo que está en el tablero.
   Políticas explicitas en el tablero.

   Las políticas deben estar expuestas en un lugar destacado

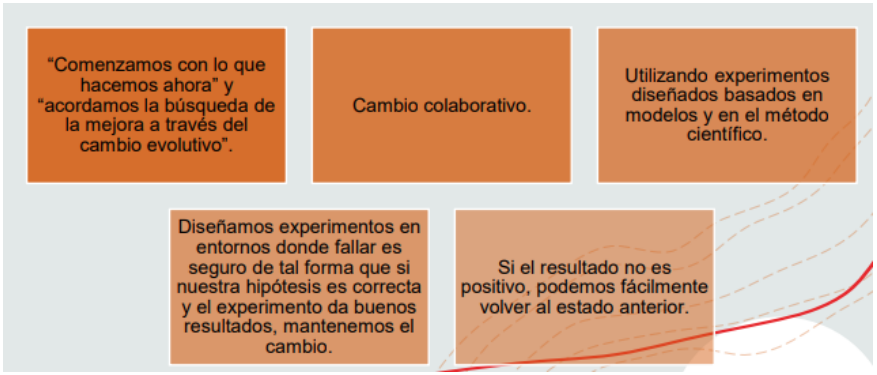
   A nivel de equipo, los acuerdos son una buena forma de introducir las políticas

   Deben ser:
   - Pocas
   - Sencillas
   - Bien definidas
   - Visibles
   - Aplicables en todo momento
   - Fácilmente modificables por los que prestan el servicio

Políticas para la clase de servicio **expreso/bala de plata/urgencias**:



5. Mejorar colaborativamente.



Relacionado con la restrospective de Scrum.

https://itnove.com/blog/kanban/equipos/cuales-son-las-practicas-de-kanban/

https://mamaqueesscrum.com/2021/01/17/metodo-kanban-en-6-practicas/

MÉTRICAS CLAVE: Kanban mide proceso. Mide cuanto tiempo nos demoramos en entregar cada pieza de trabajo que va pasando por el tablero.

- **LEAD TIME**. Vista del cliente, la mira el cliente. Desde que el cliente me lo pidió hasta que yo se lo entregue.
  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.
  Ritmo de entrega.

- **CYCLE TIME**. Vista interna. Para el equipo. No cuenta el tiempo que están los ítems en el producto backlog. Es la métrica que registra el tiempo que sucede entre el inicio y el final del proceso, para un item de trabajo dado. Se suele medir en días de trabajo o esfuerzo.
  Medicion más mecánica de la capacidad del proceso
  Ritmo de terminación

- **TOUCH TIME.** El tiempo en el cual un ítem de trabajo fue realmente trabajado (o "tocado") por el equipo.
  Cuántos días hábiles pasó este ítem en columnas de "trabajo en curso", en oposición con columnas de cola / buffer y estado

Relación con los conceptos de Lean

- Definiendo el valor desde la perspectiva del cliente
- Limitando el trabajo en progreso
- Identificando y eliminando el desperdicio
- Identificando y removiendo las barreras en el flujo
- Cultura de mejora continua

Valores de Kanban

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

Clase 19/10/23 – Métricas de software en los diferentes enfoques de gestión

Una métrica debe tener un conjunto de características para que realmente justifique y valga la pena invertir. Invertir en métricas tiene un costo.

El beneficio tiene que ser mayor al costo de invertir en obtenerlas.

Una métrica es un número, **valor cuantitativo** que determina la presencia de algo que quiero medir en el contexto determinado.

Atención al final !!!!!!!!!!!!!

Resultado de una medición. Facilita la construcción de indicadores para la toma de decisiones

Una métrica es la medida o la presencia o el grado de valor de un atributo sobre un aspecto que quiero medir.

EJEMPLO DE MÉTRICA **MEDIBLE** Y QUE SE REPRESENTE NUMÉRICAMENTE PARA QUE SEA **OBJETIVA**.

Se hacen métricas para obtener **visibilidad** sobre ciertas situaciones y aumentar el nivel de conocimiento sobre algún aspecto y las métricas nos ayudan cuando construimos indicadores desde las métricas ayudándonos a tomar decisiones informadas.

Las métricas deben tener una necesidad, ser útiles para alguien que las necesite.

Automatizar lo más posible sería ideal. Las métricas que definamos tengamos una línea clara sobre para qué se van a usar.

Definir la formula para calcularla.

El ámbito generador de las métricas es dentro del proyecto.

Métricas en el enfoque tradicional

Basado en procesos definidos. ↓

Métricas de Software

En el software, lo que elegimos medir o el dominio se divide en 3 partes. A la gente no se la mide.

El dominio de las métricas del software se divide en:

- Métricas de **proceso**. Son públicas. Puedo armarlas basándome tanto en las métricas de proyecto como de producto. Necesito hacer un trabajo de consolidación y despersonalización de los datos. Para que una métrica sea publica, no debe tener atribución a ningún producto ni ningún proyecto en particular de la organización. Es una métrica despojada de vinculación. Tomar datos de muchos proyectos, recopilo información y público. Son de la organización.
    - Índice de defectos por líneas de código. Comportamiento organizacional
    - MEJORAN EL PROCESO. Información objetiva
    - Eficiencia en la remoción de procesos/defectos.
    - PASOS: Se obtienen de los proyectos, se hace un promedio, despersonalizo, publico.

- Métricas de **proyecto**. Relacionadas con la triple restricción. Medimos cosas relacionadas a los recursos, tiempo y alcance. Ejemplo: esfuerzo, esfuerzo total, esfuerzo por etapas, esfuerzo de personas, diferencias entre es esfuerzo estimado con el esfuerzo real. Calendario: cuánto dijimos que iba a durar vs. Cuanto realmente duró. Tiempo que llevó cada iteración.
    - No se publican

- Métricas de **producto**. Ejemplo: líneas de código, complejidad ciclomatica, casos de usos, medición de defectos. Defectos por nivel, defectos por severidad, densidad de defectos, cobertura del testing(porcentaje de lo que testié, sobre la totalidad de cosas que había que testear). Son privadas
    - Miden al software específicamente. Más técnicas.
    - Líneas de código sin comentarios, usada pero no sirve
    - Satisfacción, cumplimiento de calidad del cliente.

- Epifenómeno.
- Porcentaje de requerimientos cumplidos
- Cantidad de casos de uso por complejidad

Las métricas del proyecto se consolidan para crear métricas de proceso que sean publicas para toda la organización del software.

Cuando uno quiere armar la política de medición de una organización, tiene que ver con la organización las métricas que se van a definir. Sino, cada proyecto en su plan hay que planificar qué métricas tomar. Si ya lo tengo a nivel organización, puedo utilizar estas métricas. Sino las debería ver el líder de proyecto en ese proyecto en particular.

En la gestión tradicional, se plantean un conjunto de métricas, denominadas métricas básicas por una razón de establecer una base y aportar una visibilidad mínima.

Métricas básicas para un proyecto de software

1. ***Tamaño del producto (PRODUCTO):***
   Producto y trabajo no es sinónimo.
   Una cosa es el tamaño del producto y otra cosa es el esfuerzo que necesito para construir el producto de ese tamaño.
   A medida que el producto es mas grande o complejo, más horas vas a necesitar para completar el trabajo.

   Se mide en alcances, requerimientos, casos de uso por complejidad.
   La idea es homogeneizar lo más posible el valor.
   Responde al qué
   En base a requerimientos, alcances, requerimientos por complejidad.

2. ***Esfuerzo (PROYECTO)***
   Se mide en **horas personas lineales**.
   La idea es asumir que es una persona la que hace el trabajo haciendo una cosa por vez.
   No se tiene en cuenta en el esfuerzo, el trabajo paralelo, el índice de solapamiento, la cantidad de gente.
   Es una “bolsa de horas” donde no tenemos en cuenta si podemos hacer mas de una cosa a la vez o la cantidad de personas que tenemos.
   No contamos tiempo muerto, ni de almuerzo. Trabaja una sola persona haciendo una cosa a la vez.
   Responde al como
   Esfuerzo por iteración, esfuerzo por caso de uso. Nivel de granularidad es una decisión del líder de proyecto.

3. ***Tiempo (calendario) (PROYECTO)***
   Se toma como base el esfuerzo.
   En esta métrica si es relevante la cantidad de trabajo por día, cuantos días se trabajan por semana, índice de solapamiento para ver si se pueden hacer cosas en paralelo, cuanta gente va a trabajar y en función de eso se arma un calendario.
   Se arma un calendario que indica la fecha para la entrega del producto.
   El tiempo responde al cuándo.
   De acuerdo con el tamaño del proyecto. No es ideal que se mida en horas. Confunde con el esfuerzo.

4. ***Defectos (PRODUCTO)***
   Cantidad de defectos por severidad.
   Densidad de defectos. Mide cuantos defectos tenemos por bloque de código, como un caso de uso, modulo, etc.

“MANTÉNGALO SIMPLE”

No definir métricas de costo de atención altísimo.

La precisión es cara. Las estimaciones agiles se fueron hacia las certezas.

Preguntas:

- ¿Nos da más información que la que tenemos ahora?
- ¿Es información de beneficio practico?

- ¿Nos dice lo que queremos saber?

EJEMPLOS DE MÉTRICAS:

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

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.

Foco en satisfacer las necesidades de los interesados, que suelen ser distintas. BALANCE. Para que le sirva a todo.

Política de métricas. Triple ambientes restricción.

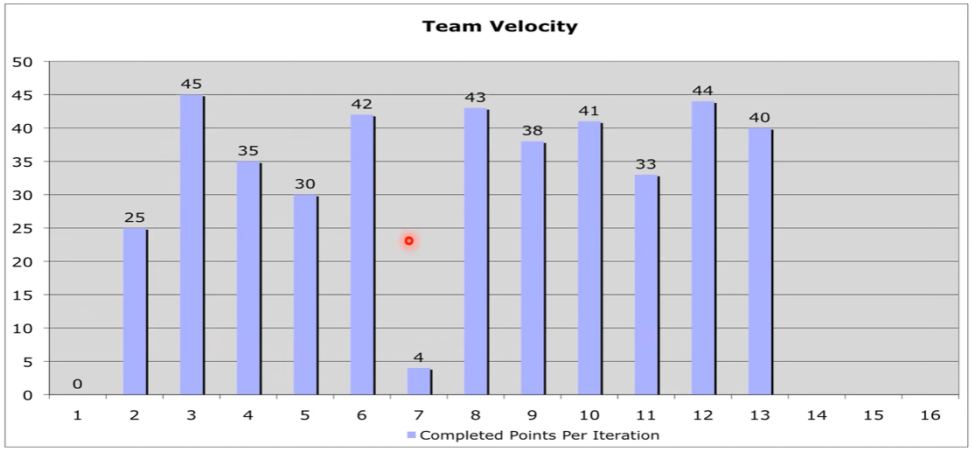
Métricas en agiles

La visión que hay sobre las métricas en ágil es distinta.
La medición es una salida, no una actividad. Esté integrado, que sea una consecuencia.
Filosofía minimalista: medir lo que sea necesario y nada más.
Foco en el producto.
El proceso no se mide. La experiencia no es extrapolable a otros equipos.

DOS PRINCIPIOS QUE GUÍAN LA ELECCIÓN DE LAS MÉTRICAS:

- Nuestra mayor prioridad es satisfacer al cliente por medio de entregas tempranas y continuas de software valioso, funcionando.
- El software funcionando es la principal medida de progreso.

PRINCIPAL MÉTRICA DE ÁGIL: **VELOCIDAD**



El equipo no logró el principio de desarrollo sostenible.

La velocidad es una métrica que mide cuanto producto nosotros pudimos entregarle al PO al final de la iteración y el nos lo aceptó.

Cantidad de puntos de historia aceptados por el PO en un sprint.

La velocidad se mide en puntos de historia, la velocidad se mide en producto.

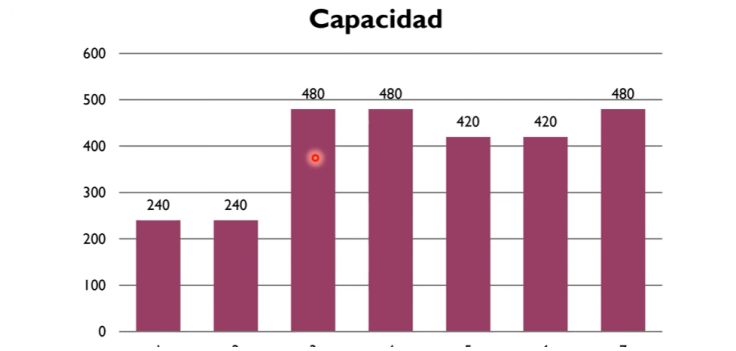
La velocidad no se estima, se calcula al final del sprint.

CAPACIDAD.

Si se estima. La usamos como estimación para determinar cuanto nos podemos comprometer como equipo para un determinado sprint.

Se puede medir en horas ideales o en puntos de historia.

Sprint	1	2	3	4	5	6	7	Total
Horas	240	240	480	480	420	420	480	2760
Puntos de Historia	30	30	45	60	58	52	60	335



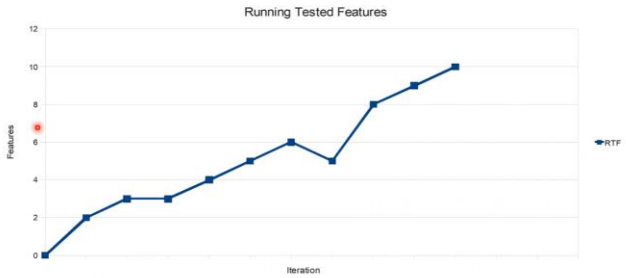
RUNNING TESTED FEATURES (RTF)

No se utiliza mucho. Se encarga de contar la cantidad de características de software que están en producción, que han sido testeadas y funcionan.

Cuantas características entregué por cada sprint.

Features podrían ser US.

No es muy útil pero no indica mucho debido a que las características pueden tener complejidades muy distintas.



Métricas en KANBAN

Para procesos empíricos con enfoque lean. Se miden para cada pieza de trabajo que pasa por el tablero. El foco en lean es en el proceso.

- Lead time. Vista del cliente
  Le sirve al cliente.
  Mide el ritmo de entrega

- Cycle time. Vista interna
  Le sirve al equipo de desarrollo porque para ellos su reloj empieza a contar a partir de que empiezan a trabajar realmente en algo.
  Mide el ritmo de terminación

- Touch time
  Toma todas las columnas que tenía el tablero Kanban y suma las columnas de trabajo y no las de acumulación.
  Nos permite conocer la eficiencia del proceso.

Touch Time (Tiempo de Tocado)

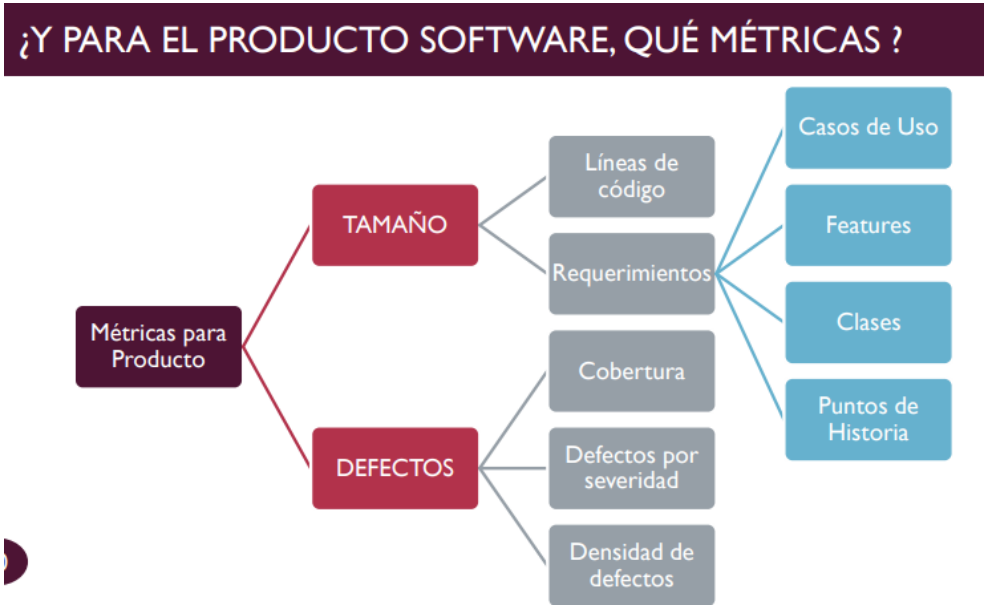
- El tiempo en el cual un ítem de trabajo fue realmente trabajado (o "tocado") por el equipo.
- Cuántos días hábiles pasó este ítem en columnas de "trabajo en curso", en oposición con columnas de cola / buffer y estado bloqueado o sin trabajo del equipo sobre el mismo.

Touch Time ≤ Cycle Time ≤ Lead Time

Eficiencia del Ciclo de Proceso

% Eficiencia ciclo proceso = Touch Time / Elapsed Time.

Independientemente del enfoque de gestión que se elija, al producto hay que medirlo.



Cobertura: porcentaje. Del 100% de los CU, se comprobaron. Porcentaje de cosas probadas sobre el total de cosas del producto.

Métricas orientadas al 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. Nivel de tolerancia del cliente.

Revisiones Técnicas

Comparación de las revisiones técnicas con la otra forma de hacer verificación de software, que es PROBANDO. El problema con las pruebas es que son MAS caras. Nos sale más caro que una falla llegue hasta la etapa de pruebas.

Para detectar una falla en la etapa de requerimientos, no hago testing, sino que hago verificación estática. Que son las REVISIONES TÉCNICAS. Reviso la ERS, por ejemplo, reviso como se están escribiendo los requerimientos.

Retrabajo: es costoso.

2 TIPOS DE REVISIONES:

- Walkthroughs. Más informal. Mínima sobrecarga, capacitación de desarrolladores. Rápido retorno.
    - Poca o ninguna preparación
    - No hay mediciones
    - No FTR
- Inspecciones. Mas formal. Detectar y remover todos los defectos eficiente y efectivamente.
    - Proceso formal
    - Checklists
    - Mediciones
    - Fase de verificación

Ventajas

- Pueden descubrirse muchos errores
- Pueden inspeccionarse versiones incompletas
- Pueden considerarse otro atributos de calidad

Desventajas

- Es difícil introducir las inspecciones formales
- Sobrecargan al inicio los costos y conducen a un ahorro sólo después de que los equipos adquieran experiencia en su uso.
- Requieren tiempo para organizarse y parecen ralentizar el proceso de desarrollo

- Tareas. Bd, investigación,
- Tiempo de cada tarea. Tiempo. Según cada US.
- Criterio de Done. Armar nuestro criterio.
- Acá aclarar días hábiles. Especificar feriados, vacaciones.
- 80hs para dedicar a una sprint. Incluyen horas de ceremonias.

DESCOMPONER LAS US EN TAREAS

PRÁCTICO 8 – SCRUM - Planificación de Release y de Sprint

Release Planning.

- Cuanto tiempo
- Cuantos Sprints. Cuánto dura cada sprint.
- Que voy a trabajar en cada sprint
- Qué US son parte del reléase
- Nombre de US y relación con el sprint

SPRINT N°	US	SP

Sprint Planning

- Quienes trabajan. Contexto del equipo
- Considerar feriados
- Capacidad de cada uno y del equipo

PRÁCTICO 10 – Testing de caja blanca

A continuación se presenta una porción de pseudocódigo que resuelve la US "**Ver mapa de taxis**":

```
If (BusquedaNumeroChapa = True)
       If (Se encontró número de chapa)
              [Mostrar Datos de número de chapa]
              Switch (Estado) {
              Case ("Libre"): [Mostrar taxi en el mapa resaltado en Verde]
              Case ("Solicitado"): [Mostrar taxi en el mapa resaltado en Amarillo]
              Case ("Ocupado"): [Mostrar taxi en el mapa resaltado en Rojo]
              Case ("Fuera de Servicio"): [Mostrar taxi en el mapa resaltado en Negro]
              }
              If (Estado= "Solicitado" OR Estado = "Ocupado")
                     [Mostrar Datos de Pasajero]
                     If (Estado = "Ocupado")
                     [Mostrar Datos de Viaje]
                     End if
              End if
       Else
              [Mostrar mensaje de error "Taxi no conectado"]
End if
```

COBERTURA: De sentencia

Caso de prueba N°1:

- BusquedaNumeroChapa = true
- Se encontró número de chapa = true
- Switch case Solicitado
- Estado = Solicitado (Mostrar datos de pasajero)

Caso de prueba N°2:

- BusquedaNumeroChapa = true
- Se encontró número de chapa = true
- Switch case Ocupado
- Estado = Ocupado (Mostrar datos de viaje)

Caso de prueba N°3:

- BusquedaNumeroChapa true
- Se encontró numero de chapa true
- Switch case Libre

Caso de prueba N°4:

- BusquedaNumeroChapa true
- Se encontró numero de chapa true
- Switch case Fuera de servicio

Caso de Prueba N°5:

- BusquedaNumeroChapa = true
- Se encontró numero de chapa = false

COBERTURA: De decisión

Caso de prueba 1:

- BusquedaNumeroChapa = true
- Se encontró numero de chapa = true
- Case Estado = Solicitado
- (Estado = Ocupado) = false

Caso de Prueba 2:

- BusuedaNumeroChaoa = true
- Se encontró numero de chapa = true
- Case Estado = Ocupado
- (Estado = Ocupado) = true

Caso de prueba 3:

- BusuedaNumeroChaoa = true
- Se encontró número de chapa = true
- Case Estado = Libre

Caso de prueba 4:

- BusuedaNumeroChapa = true
- Se encontró número de chapa = true
- Case Estado = Fuera de servicio

Caso de prueba 5:

- BusquedaNumeroChapa = true
- Se encontró numero de chapa = false

Caso de prueba 6:

- BúsquedaNumeroChapa = false

COBERTURA: Condición (queda igual que de decisión)

Caso de prueba 1:

- BusquedaNumeroChapa = true
- Se encontró número de chapa = true
- Case Estado = Solicitado
- (Estado = Ocupado) = false

Caso de Prueba 2:

- BusuedaNumeroChaoa = true
- Se encontró número de chapa = true
- Case Estado = Ocupado
- (Estado = Ocupado) = true

Caso de prueba 3:

- BusuedaNumeroChaoa = true
- Se encontró número de chapa = true
- Case Estado = Libre

Caso de prueba 4:

- BusuedaNumeroChapa = true
- Se encontró número de chapa = true
- Case Estado = Fuera de servicio

Caso de prueba 5:

- BusquedaNumeroChapa = true
- Se encontró numero de chapa = false

Caso de prueba 6:

- BúsquedaNumeroChapa = false

COBERTURA: Multiple (queda igual que de condición)