SCRUM

Filosofía de SCRUM

Un principio que tiene que ver con scrum

“En un SRUM, hay que actuar como una unidad, no como 8 individuos. Todos tienen un rol. Nunca debemos olvidar que cuando trabajamos juntos como una unidad, el todo es más que la suma de las partes.”

EL principio que tiene que ver con scrum es un poco el que conocemos en otro contexto como el de sinergia, en donde el todo es más que la suma de las partes y eso es lo que trata SCRUM de potenciar en estos grupos de trabajo de 5 a 7 u 8 personas.

¿CÓMO SE BUSCA ESO?

¿Cómo se busca potenciar estos grupos de trabajo? Se busca a través de este framework.

Que nos ayuda a crear o que crea un proceso. En particular scrum es un proceso que está en el contexto de los procesos empíricos para que en el desarrollo de software se puedan crear nuevos productos.

Scrum es un proceso empírico, pero trabaja bajo cierta definiciones concretas, es decir, se definen lineamientos que se deben seguir, no es que el equipo va decidiendo qué hacer ni eligiendo como trabajar. Son lineamientos muy claros que si o si deben cumplirse.

- Scrum es un framework que permite crear su propio proceso para crear nuevos productos.
- Scrum es simple, puede ser implementado en pocos días, pero puede tomar mucho tiempo perfeccionarlo.
- “Scrum no es una metodología – es un camino” KEN SCHWABER

No es una metodología basada en procesos definidos. Que son estas metodologías basadas en líneas de ensamblado. Y en relación al control tiene que ver, está relacionado con las inspecciones y muchas veces con el control de pares (lo que tiene que ver con auditorias “controles”

- Las metodologías rigurosas se basan en métodos definidos, con la idea de línea de ensamble.
- El control de scrum se alcanza con inspecciones frecuentes y correspondientes ajustes.

LOS VALORES DE SCRUM

- CORAJE
- FOCO
- COMPROMISO
- RESPETO
- APERTURA

Estos valores podemos ver que están muy vinculados a los valores de manifiesto ágil. Osea que va en línea con esas definiciones. Y van en línea con lo que se promueve a partir de las metodologías agiles. SCRUM específicamente además, refuerza que estos son los valores con los que se trabaja.

EL RITMO DE SCRUM

(Se ve con las ceremonias) tiene que ver con planificar, ejecutar, analizar cómo nos fue en esa ejecución, realizar las acciones correctivas que consideremos necesarias y volver a planificar. Y así iterar la cantidad de veces que haga falta para crear el producto que nosotros queremos.

- PLAN
- EJECUTE
- REFLEXIONE

¿CUALES SON LOS CIMIENTOS DE SCRUM?

- EMPIRISMO – Ya que Scrum es un proceso empírico, pero trabaja bajo ciertas definiciones concretas, es decir, se definen lineamientos que se deben seguir, no es que el equipo va decidiendo qué hacer ni eligiendo como trabajar. Es decir, se trata de un framework que trabaja bajo las características de un proceso empírico, no de procesos definidos ni de metodologías tradicionales. Son lineamientos muy claros que si o si deben cumplirse. No es una metodología basada en procesos definidos. Que son estas metodologías basadas en líneas de ensamblado.
- AUTO ORGANIZACIÓN – Hablamos de autoorganización entre los miembros del equipo, y por eso se habla de realizar un control, que también tiene que ver con un equipo que tenga la característica de poder autoorganizarse, es decir, un equipo conformado por personas autónomas que tengan la capacidad de cumplir los objetivos que se plantean.
- COLABORACIÓN – Es entre los medios del equipo, entre el PO, entre todas las personas que participan de la construcción o que tienen algún rol, dentro de este framework.
- PRIORIZACIÓN – Es muy importante, ya que hace referencia en que nos vamos a concentrar en aquello que definamos con mayor prioridad y sobre eso vamos a hacer foco y sobre eso vamos a prestar especial atención. Y lo que quede con menor prioridad no es que no lo vamos a ver sino que lo vamos a ver cuando llegue su momento, cuando esa prioridad esté dentro del tope o este dentro del momento que necesitemos atenderla
- TIME BOXING – Concepto importante y que tiene que ver con que el planteo de las iteraciones y el trabajo que se hace se hace planteando limites de tiempo CONCRETOS. En aclaración con respecto a otros ciclos de vida iterativos e incrementales, por ejemplo, lo que se hace en el proceso unificado de desarrollo es partir el alcance y cada una de las iteraciones es un alcance concreto que se ejecuta. Por ejemplo, en el PUD que hacemos? Lo que está fijo es el alcance y lo que modificamos es el tiempo que nos va a llevar cubrir ese alcance, en cambio, En el caso de scrum lo que se plantea es: dejar fijo EL TIEMPO. Y lo que vamos a definir y lo que vamos a mover es el alcance para ver cuanto podemos construir en ese tiempo que está fijo.
  PARA QUÉ NOS SIRVE ESTE CIMIENTO DE TIME BOXING? Para darle una cadencia de entrega a nuestro cliente y para asegurarnos que cada cierto tiempo se cumplimenta con esa entrega. POR QUÉ ES IMPORTANTE? Porque si bien el alcance lo vamos definiendo sprint a sprint el cliente sabe que cada ese cierto tiempo que nosotros definimos como time boxing el va a recibir algo. Y eso nos permite manejar las expectativas del cliente.
  Por supuesto que este time boxing se extiende más allá de la duración del sprint. Se extiende también a la duración de las ceremonias, se extiende también a como trabajamos en términos de equipo cuando tenemos algo para ejecutar. Este concepto de time boxing se ve reflejado durante todos los lineamientos del framewok de scrum.

  CÓMO TRABAJA SCRUM

Scrum trabaja con equipos pequeños, menos que diez personas, lo ideal sería 7 +-2, la mayoría de las veces suele conformarse de esa forma. Sin embargo, hay casos en donde se necesita construir un producto mas grande donde con un equipo de 7+- personas no es suficiente y necesitamos escalar a un equipo más grande. Pero en un principio cuando hablamos de scrum, siempre se habla de un equipo pequeño.

LOS SPRINTS
Es la duración fija que le vamos a poner a cada una de las entregamos que vamos a hacer. La cual va a durar 30 días, puede durar 3 semanas, esa duración la define el equipo.

Lo más importante es que a partir de cada uno de esos sprint obtenemos como resultado incrementos visibles y usables, son incrementos que sirven potencialmente para ser desplegados en producción independientemente de la decisión que nosotros tomemos después.

Y trabaja con los conceptos ya vistos de time boxing, colaboración, priorización, autoorganización.

¿CÓMO SE PLANTEAN CADA UNO DE ESTOS TEAMS PLANTEADOS?

¿POR QUÉ SCRUM PLANTEA UNA FORMA DE TRBAJAO DIFERENTE?
Tener en cuenta que algunas formas y principios que plantea el scrum están muy relacionadas con cosas de manifiesto ágil.

- En principio, menos tiempo planeado y definiendo tareas,
- Menos tiempo creando y leyendo reportes
- Se pasa más tiempo con el equipo investigando la situación.
- Scrum supone dominios impredecibles
- Scrum no supone un proceso repetible.

Las métricas que se utilizan son las mínimas que nos hacen falta, no hay un exceso o un conjunto de métricas a elegir como si pudiera pasar con proyectos que trabajan bajo procesos definidos.

Scrum es un framework muy útil para trabajar, por un lado, EL DOMINIO ES IMPREDECIBLE. Esto es en referencia al negocio, no tengo claridad todavía del producto que yo voy a construir en TERMINOS DE REQUERIMIENTOS.

¿POR QUÉ ES UTIL EN DOMINIOS IMPREDECIBLES?

Porque yo no tengo que definir el alcance completo desde el principio, sino que vamos definiendo y priorizando en conjunto con el PO a medida que vamos avanzando. Esto nos permite que toda aquella incertidumbre que tenemos de ese dominio la vamos trabajando y la vamos atacando poquito a poco. Sin ocuparnos mucho al principio de tratar de lograr tener mayor certidumbre o mayor predicción sobre todo el dominio completo. En conclusión, esto nos va a permitir retroalimentarnos y poder seguir trabajando mas adelante.

Scrum por otro lado se supone que no es un proceso repetible por eso en parte es que estamos planteando una cuestión un poco disruptiva con respecto a los procesos definidos y a los proyectos gestionados de manera tradicional.

ROLES DE SCRUM

- SCRUM MASTER Es el **MODERADOR**, no es lo mismo que un líder. ¡Tiene que quedar claro que es UNO MÁS DEL EQUIPO, es quien más conoce el framework! Ayuda al equipo a salvar los impedimentos, a establecer cómo tienen que trabajar. ¡Pero es un concepto distinto de líder! No tiene un rol de control como un líder. Es una línea muy fina. No es lo mismo que sirva como un FACILITADOR. A que CONTROLADOR.

- PRODUCT OWNER Es el experto en dominio y se encarga específicamente del producto backlog, de priorizar los ítems. ¿En función de qué? En función del valor que le aporta al negocio. Es el referente del dominio para nosotros y debe tener la capacidad de tomar decisiones y para definir la prioridad, para donde vamos, eso no significa que él no pueda consultar a otros expertos del dominio.

- EQUIPO DE TRABAJO – SCRUM TEAM Tiene que estar conformado por integrantes motivados. ¿CÓMO SE HACE CON LOS ROLES DEL EQUIPO? No hay roles dentro del equipo. TODOS HACEN TODO. Tiene que ser autoorganizado, trabajar de manera colaborativa y tener en cuenta que todos hacen todo. Y eso también es importante porque podemos tener algunos que sean mejores que otros en algunas cosas, cada uno tiene mas skills para algunas cosas que para otras. ¿Pero en un equipo de scrum todos tienen que saber programar? SI porque todos hacen todo. ¡Es importante!

ROLES	RESPONSABILIDADES
Product Owner	• Asistir a la Sprint Retrospective • Asistir a la Sprint Review • Asistir a la Daily Meeting • Asistir Refinamiento del Backlog (Grooming) • Crear Ítems del Product Backlog • Hacer seguimiento de progreso del Sprint (DAILY) • Priorizar el Product Backlog • Hacer el Product Backlog visible para todos • Hacer seguimiento de progreso del Release • Probar (por las pruebas de usuario) • Asegurar calidad (por las pruebas de usuario)
The Team	• Asistir a la Sprint Retrospective • Asistir a la Sprint Review • Asistir a la Sprint Planning • Asistir a la Daily Meeting • Asistir Refinamiento del Backlog (Grooming) • Hacer seguimiento de progreso del Sprint • Construir • Integrar el Software • Probar • Resolver Impedimentos técnicos • Diseñar

	• Desplegar • Asegurar Calidad
ScrumMaster	• Asistir a la Sprint Retrospective • Asistir a la Sprint Review • Asistir a la Sprint Planning • Asistir Refinamiento del Backlog (Grooming) • Asistir a la Daily Meeting • Hacer seguimiento de progreso del Sprint • Dar coaching al equipo • Asegurar que se siguen las reglas • Facilitar eventos (liderar significa, conducirlo) • Mejorar el proceso • Resolver Impedimentos organizacionales

Todos están en las dailys por lo que todos hacen el seguimiento del sprint

ENTREGABLES

Se resumen en 3

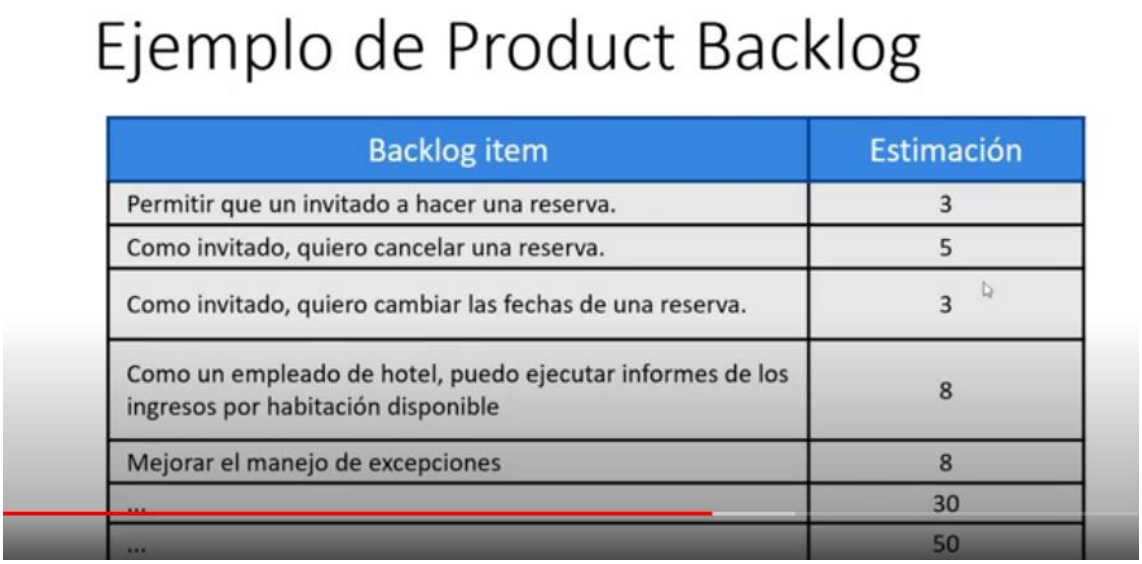
- PRODUCT BACKLOG, es una lista, es una cola priorizada de funcionalidades técnicas y de negocio, que necesitan ser desarrolladas. Son técnicas y de negocio porque van las cuestiones de negocio pero que necesitan ser desarrolladas (por eso serían técnicas). Normalmente en esa lista tenemos aquellos ítems que tienen mayor prioridad en función del valor de negocio y la prioridad que se le asignó y normalmente lo mas importante es que ese PB no necesariamente tiene que estar definido de manera completa. Lo que si tiene que estar definido de manera completa es lo que está en la pila, en la parte de arriba de la pila y que es lo que vamos a incluir en el próximo sprint. Lo demás puede estar en términos de épicas, en termino de temas. Y se van a ir refinando conforme van subiendo de prioridad en la lista. Esto es lo que nos ayuda a trabajar sobre esta cuestión del cambio ante un dominio impredecible. ¿POR QUE? Porque al hacer foco sobre lo que está arriba de la fila, lo de abajo puede ser un Recordatorio de que es algo que vamos a tener que ver si o si y en algún momento nos tenemos que poner a trabajarlo.

  En la medida que esto no llegue al comienzo de la pila porque no se le da prioridad lo podemos ir dejando expresado en términos generales sin entrar en mayor detalle.
  Sin embargo, lo que está en el principio de la pila tiene que tener un nivel de detalle suficiente como para que nosotros podamos sentarnos a trabajar en eso.
  - ✓ Trabajar con un backlog de producto es como jugar a asteroides.
  - ✓ GRANDES ROCAS (EPICAS) se rompen en pedazos de rocas mas pequeñas (STORIES) hasta que son lo suficientemente pequeñas para ser eliminadas (DESARROLLADAS Y ENTREGADAS)

RELACIONADO CON LO QUE VIMOS EN REQUERIMIENTOS AGILES, Podemos decir que lo las users que están en la parte de arriba del BACKLOG que son las que vamos a trabajar en el próximo sprint que ya vimos nosotros que tienen que cumplir con el modelo INVEST.

¿CÓMO ES EL PRODUCT BACKLOG? ¿CUAL ES LA VISTA DEL PRODUCT BACKLOG?
Nosotros elegimos una partecita, son las historias, los requisitos que están arriba de toda la lista de todos los trabajos deseados en el proyecto (PB) que tienen la mayor prioridad (por eso están arriba) que son las que tienen mayor valor para el cliente. Quien define esa prioridad es el PO. Y esta lista del producto backlog puede ser repriorizada al inicio de cada sprint, es decir que las prioridades no son inamovibles.

EJEMPLO DE PRODUCT BACKLOG(PB)



Una vez que yo definí cuales de esos requisitos o cuales de esos requerimientos voy a hacer en el siguiente sprint, logro el siguiente de los entregables que es el SPRINT BACKLOG, QUE ES LA PORCIÓN DEL PRODUCT BACKLOG QUE NOSOTROS VAMOS A CONSTRUIR EN EL SIGUIENTE SPRINT.

- SPRINT BACKLOG

Es una porción del PB que nosotros decidimos como equipo hacer en el próximo sprint, en donde en conjunto con el PO decidimos cuales son las users que vamos a incluir en este nuevo sprint.

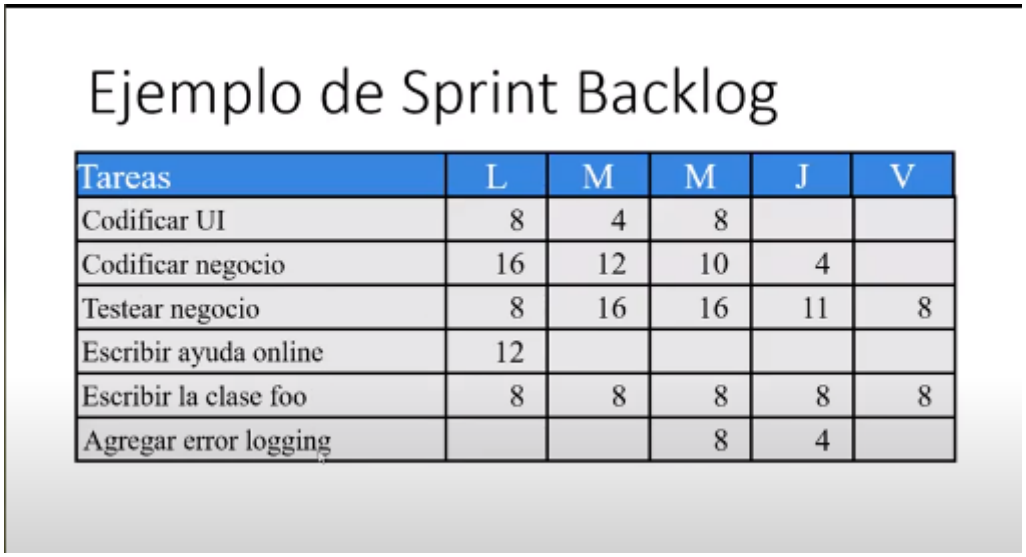
Para poder construir el sprint backlog vamos a necesitar además de la estimación que ya hicimos en términos de story point, hacer una lista de tareas para completar cada uno de esos requisitos/users. Teniendo en cuenta que esta lista de tareas que debemos hacer para cada una no deberían demorar más de 16hs ni menos de 4hs para poder completarse (CADA TAREA).

EN EL SPRINT BACKLOG APARECE EL CONCEPTO DE TAREAS para que cada una de las personas que conforman el equipo puedan realizarlas.

- El equipo determina que debe hacerse para cumplir el objetivo del sprint
- El dueño del producto suele asistir
- Se realiza una lista de tareas que tomen de 4 a 16 horas para completarse
- El backlog no se modifica durante el desarrollo

- Si el equipo descubre que el backlog no puede realizarse, el scrum master y el dueño del producto (PO) determinan si:
    - ✓ Algún item puede ser removido del backlog
    - ✓ Alguna funcionalidad puede eliminarse.

EJEMPLO DE SPRINT BACKLOG



El tercero de los entregables es:

- PRODUCTO DE SW POTENCIALEMNTE LISTO PARA PRODUCCIÓN

Parte del producto desarrollado en un sprint en condiciones de ser usado en el ambiente correspondiente.

Cuando termine el sprint si o si como resultado debo tener un entregable de ese producto que yo puedo desplegarlo en producción.

Por qué se plantea “POTENCIALMENTE LISTO PARA PRODUCCIÓN” porque no necesariamente yo voy a definir que ese incremento vaya a desplegarse en producción pero está construido de tal manera que si se pidiera, se puede.

ESTIMACIONES DEL BACKLOG

Una de las características del modelo INVEST es que las User tenia que se estimable y pequeña. En ambos casos, para saber si es estimable y si es pequeña necesitamos hacer una ESTIMACIÓN.

Las estimaciones son un esfuerzo colaborativo entre las partes

- ✓ Las estimaciones son iterativas
- ✓ Si un item no puede ser debidamente estimado, se debe dividir en el backlog.
- ✓ Los estimados sirven de base para la funcionalidad en el sprint.

¿Cuál es la técnica que usamos para estimar? EL POKER PLANNING

¿Cuál era la unidad con la que estimábamos? LAS STORY POINT

¿Cómo sabíamos si la cantidad de story point eran razonables? ¿Cuáles eran los valores que nos ayudaban a saber que las story point eran razonables? Mayor a 8

MAYOR A 8, hay que dividirlo. No podríamos tener en la parte de arriba del backlog para ser incluido en el próximo sprint un ítem que tenga un valor de 13

Estos estimados, esta cantidad de story point que le vamos poniendo a cada una de las users son las que nos sirven para poder definir, es uno de los elementos que nos va a hacer falta para definir cuáles son los ítems que nosotros vamos a poder incluir en el próximo sprint.

Sin embargo, es solo uno de los ítems que nos van a ayudar a definir, pero faltan otros. Porque esto solo me dice cuanto esfuerzo, tamaño e incertidumbre en esto que yo estoy estimando. Pero por otro lado también me falta poder determinar cuánto es lo que yo puedo hacer.

4 REUNIONES – MEETINGS

SPRINT PLANNING (obigatoria) Parte de lo que hacemos tiene que ver con tomar el producto backlog, priorizar (priorizar puede implicar incluso modificar alguna prioridad o valor de negocio) y elegir hacer cuales son los requisitos o requerimientos o user de ese producto backlog que nosotros vamos a incluir en el próximo sprint. Eso es básicamente lo que hacemos en la planning. Entonces mas allá de definir las users, se define el objetivo del sprint, tenemos que decir cómo se va a alcanzar ese objetivo, eligiendo las users que van a estar incluidas, y a partir de ellas, se crea el sprint backlog con esas tareas que definimos para las users incluidas,

Tomamos las users que vamos a incluir en el prox sprint backlog, las desglosamos creando todas las tareas que necesitamos para ejecutar o construir nuestro entregable potencial y lograr una estimación del sprint backlog en horas.

Todos los sprints tienen que tener un objetivo porque es lo que nos permite saber si lo alcanzamos o no cuando terminamos.

Las users o los features que vamos a incluir de PB en ese sprint, la lista de tareas para poder construir las features y las horas de esas tareas. Todo esto hacemos en la planning.

CAPACIDAD DEL EQUIPO.

¿COMO HAGO YO PARA SABER SI EN MI SPRINT BACKLOG TOMO 1 2 3 5 USERS? ¿COMO SE CUALES VOY A INCLUIR EN MI SPRINT?

Eso va a depender de la capacidad de mi equipo. ¿CÓMO CALCULAMOS LA CAPACIDAD DEL EQUIPO? Lo vemos mas tarde.

DAILY (obigatoria) Son cortas, de 15-30 min. Se analiza el estado. EL FACILITADOR ES EL SCRUM MASTER. Asiste todo el equipo. No son para solucionar problemas

Las 3 preguntas son que hace el scrum son:

- ✓ ¿QUE SE COMPLETÓ DESDE LA ULTIMA REUNION? (traducción de Mili: ¿qué hiciste ayer?)
- ✓ ¿QUÉ OBSTACULOS SE PRESENTARON? (traducción de Mili: ¿tuviste algún impedimento?)
- ✓ ¿QUE VAS A HACER HASTA LA PROXIMA REUNION? (traducción de Mili: ¿qué vas a hacer hoy?)
- ✓ No es dar un status report al scrum master. ¡¡NO TIENE QUE VER CON DAR UN SEGUIMIENTO DE PARTE DEL SCRUM MASTER COMO SI FUERA UN LIDER DE PROYECTO!! SINO ESTO DEL EQUIPO AUTOORGANIZADO QUE SE COMPROMETE DELANTE DE SUS PARES. Si bien el scrum es facilitador, no es un rol de control. Es un rol que hace que el resto del equipo pueda ser ayudado fácilmente a cumplir con las pautas que necesita.
- ✓ Se trata de compromisos delante de pares.

SPRINT DEMO/REVIEW (obigatoria)

Dura mas o menos unas 4 horas, **tiene un time boxing igual que todas las ceremonias**.

- ✓ Reunión informativa de unas 4 hs.
- ✓ El equipo presenta el incremento desarrollado a gerentes, clientes, usuarios y el dueño del producto. El equipo presenta el incremento del producto, este producto potencial para ponerse en producción.
- ✓ Informal:
  - Regla de 2hs de preparación
  - No usar diapositivas
- ✓ Todo el equipo participa. Se discute acerca de lo que el producto incluye. SE ACEPTA O NO LA ENTREGA DEL PRODUCTO. Se da el OK. De todas las user implementadas por el equipo, las user ACEPTADAS son las que cuentan, las demás volverán al producto backlog para corregirse lo que sea necesario. Los criterios de aceptación nos ayudan a refinar o decidir o definir, cuales son las cosas que el PO va a mirar para darnos el OK o no del USER que

nosotros estamos implementando. SCRUM no solo se hace con user, pero si lo usaramos, los criteros de aceptación toman vital importancia en este proceso.

- ✓ Se invita a todo el mundo
- ✓ Se reportan los problemas encontrados
- ✓ Cualquier item puede ser agregado, eliminado, repriorizado
- ✓ Se estima el nuevo sprint y se asignan tareas.

RETROSPECTIVA (obigatoria)

¿Vieron que SCRUM tiene que ver con planificar, ejecutar y reflexionar? ¡Bueno, la retro tiene que ver con la reflexión! Todo el equipo participa (incluso el PO) después del sprint, qué cosas funcionaron y qué cosas no funcionaron. Las que funcionaron hay que repetirlas y las que no, para generar acciones correctivas.

No es un espacio para la catarsis, ni quedarse callados y que uno solo hable y cumpla. Que se encuentre la manera de que todos en el equipo puedan participar, que todos puedan decir cual es su parecer. Por eso, en este tipo de ceremonia se usan distintas dinámicas, para que sean anónimas, para que todos participen. Y que a partir de esto realmente se pueda mejorar en el próximo sprint.

Las acciones que se generan tienen que ser concretas, darle un marco (no hacer 10) elegir una o 2, plantearlas para trabajarlas en el próximo sprint. Para efectivamente hacerlas y ver como resulta.

- Periódicamente, se echa un vistazo a lo que funciona y lo que no.
- Normalmente 1h a 4hs
- Se realiza luego de cada sprint
- Todo el equipo participa
  SCRUM
  PRODUCT OWNER
  EQUIPO
  POSIBLEMENTE CLIENTES Y OTROS.

STORY TIME/PRODUCT BACKLOG REFINEMENT (también llamada GROOMING) (Opcional)

Tiene que ver con poder discutir algunas cuestiones de los ítems del backlog en términos de alta prioridad, justamente para hacer foco en los criterios de aceptación. Es una reunión interesante para que surja antes y no después de la review, ya que, si se discute antes los criterios de aceptación, ya vamos a saber bien claro qué espera el cliente y no encontrarnos con sorpresas después.

Es opcional, el equipo se reúne con el PO, para discutir ítems del backlog de alta prioridad, determinar su criterio de aceptación y asignar la priorización a cada nuevo item.

Esto ocurre inicialmente antes del desarrollo y después iterativamente en cada sprint.

SPRINT

- Periodo fijo de tiempo, sugerido 30 días. Idealmente es de 3 y 5 semanas, lo define el equipo a ese tiempo. La idea que nosotros empezamos a ejecutar el sprint no se puede hacer nada de lo dicho acá abajo… No se puede cambiar nada de lo que haya adentro del sprint.
- Dentro del sprint:
  - ✓ No se puede cambiar el alcance.

- ✓ No se puede agregar funcionalidad
- ✓ No se pueden modificar las reglas del equipo

- Todo lo que ocurre luego de terminado el sprint nos va a permitir hacer las modificaciones en el próximo sprint backlog en función de lo que hayamos aprendido en este sprint.
- Todos los sprint tienen un objetivo, que se define en la planning. Es una declaración, un objetivo claro a alcanzar. Es donde decimos cual es la declaración de lo que vamos a hacer en el sprint.
- REGLAS DE UN SPRINT:
  - ✓ Foco en la tarea
  - ✓ SIN interrupciones
  - ✓ SIN cambios
  - ✓ El mismo equipo puede descubrir mas trabajo a ser hecho

La premisa de scrum es amigable para los entornos cambiantes y tiene flexibilidad para atender a los cambios siempre y cuando no ocurran durante la ejecución de un sprint. Esa es la regla.

- ✓ ELEMENTOS DEL SPRINT
  Tienen que ver con las reuniones que se ejecutan, con el incremento de producto que se ejecuta y se obtiene y que tiene que tener un incremento con respecto al producto anterior existente y con la colaboración y compromiso de los miembros del equipo.
  - Reuniones de scrum
  - Se produce u incremento usable y visible
  - Los incrementos se basan en un producto anterior
  - Compromiso de los miembros a la asignación
- ✓ MECANICA DE LOS SPRINT
  Hacemos foco en el timebox, en que se congelan algunas variables del proyecto durante ese tiempo del sprint, el ambiente se congela, se establecen ciertas reglas con las que vamos a trabajar y mientras se ejecuta el sprint eso no cambia.

  - En un sprint se congelan 3 de las 4 variables de un proyecto:
    Tiempo: 30 dias
    Costo: salarios + ambiente
    Calidad: generalmente un estándar organizacional

  - El equipo puede cambiar funcionalidad siempre y cuando cumpla con el objetivo del sprint → estos pueden ser unos cambios de funcionalidad mínima que se hayan acordado con el equipo. “sacamos o agregamos alguna funcionalidad”. Pero si algo de esto cambia el objetivo del sprint, no se puede. Es decir, que si hay que eliminar por ejemplo una funcionalidad del sprint que no permite alcanzar el objetivo, entonces lo más probable es que Se cancela el sprint ya que el sentido del sprint es que se cumpla el objetivo.

    CASO HIPOTÉTICO: ¿Si tenemos un ambiente de desarrollo y otro productivo y se reportan errores en el productivo los equipos deben ser diferentes? Si, el equipo si, salvo que vos dentro de tu sprint backlog de alguna manera decidas incluir la corrección de un bug. Lo que no podes hacer es atender esos errores en producción mientras estas generando un nuevo incremento del producto y ya no lo habías

incluido en el sprint backlog. Ahora la corrección de algo que haya sido incluido en el sprint, en principio puede ser válido

HERRAMIENTAS DE SCRUM

TASKBOARD

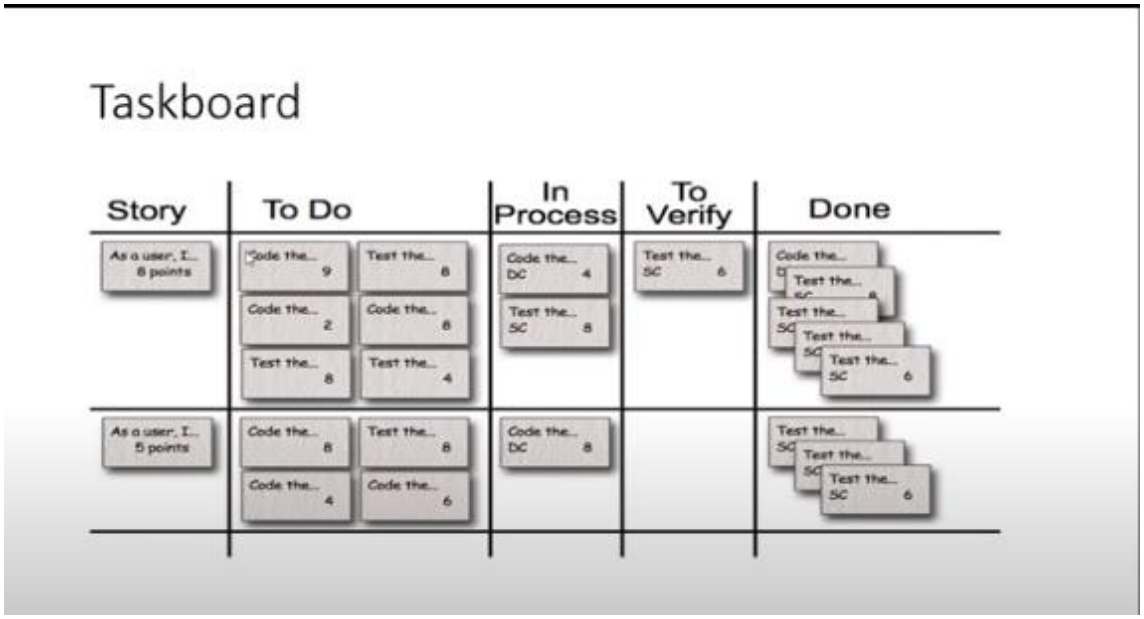
Tiene distintos componentes. Tienen que ver con la actividad, con la estimación, con quién se va a hacer cargo.

Estas tarjetas que vamos haciendo tiene distintos componentes.

Tiene el nombre de la actividad, el código del ítem que está en el sprint backlog, tiene la persona que se va a hacer cargo y el tiempo estimado en horas.

Y con este taskboard obtenido en la reunión de la planninng nosotros lo podemos dividir en estos términos.

El TO DO que es lo que hay para hacer, lo que está en proceso, lo que tenemos para verificar y lo que ya está hecho. Teniendo en cuenta que la idea es que tienen que tener suficiente granularidad como para que nosotros podamos incluir muchas tareas dentro de lo que tiene que ver con la resolución de cada uno de los ítems que nosotros tenemos que trabajar. Por eso se habla de descomponerlo en una granularidad entre 4 y 16 horas. Para que el taskboar tenga muchas tareas, no una sola.



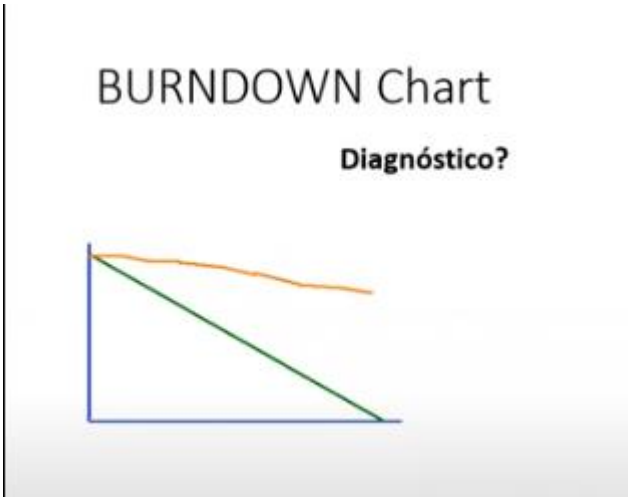
OTRA HERRAMIENTA SON LOS GRAFICOS DE BACKLOG

¿PARA QUÉ NOS SIRVEN?

Para poder medir el progreso o cuanto nos queda por hacer, alguna de las 2 cosas depende del grafico que nosotros usemos y la mirada que nosotros querramos darle.

- La gerencia necesita datos acerca de: PROGRESO DEL SPRINT, PROGRESO DEL RELEASE, PROGRESO DEL PRODUCTO.
- El backlog de trabajo es la cantidad de trabajo que queda por ser realizado
- Tendencia del backlog: trabajo que queda vs tiempo.

EJEMPLO: se llama burndown (se queman puntos) chart puede tener por ejemplo esta forma. ¿Qué significa? Que terminó el sprint, y que no alcanzamos el objetivo, nos quedó trabajo por hacer.



Uno de los problemas que pudo haber habido es que se haya estimado mal, o mas bien que no hubo nadie atento a plantear las correcciones necesarias o subsanar los impedimentos para que se pudieran corregir esas dificultades y que esta línea que ya venia complicada y finalmente se acomode.

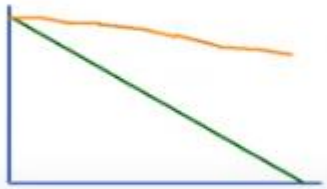
CAUSAS: pobres técnicas de estimación, mal manejo de riesgos, rotación del equipo o renuncias.

Este grafico muestra que el scrum master y el PO no reaccionaron para evitar esta situación.

Si vas viendo que hasta mas de la mitad de camino venir con esa línea, mas vale cancelarlo. No tiene sentido llegar al final de esta manera.

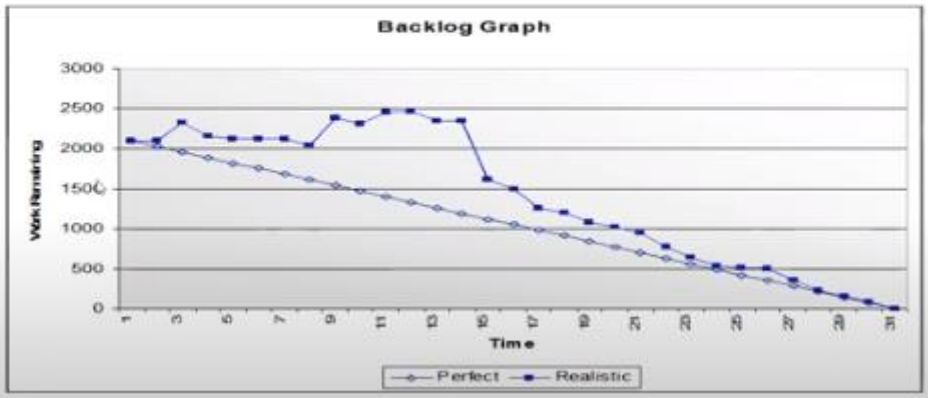
BURNDOWN Chart

Diagnóstico?



- No todo lo incluido en el sprint será entregado en la fecha propuesta. El backlog no fue modificado para alcanzar la fecha (remover requerimientos o modificar la fecha)
- Posibles causas:
  - Pobres técnicas de estimación
  - Mal manejo de riesgos
  - Rotación del equipo o renuncias
- Este gráfico muestra que el Scrum Master (SM) y el Product Owner (PO) no reaccionaron para evitar esta situación.

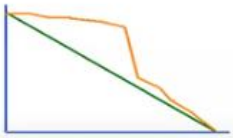
Ejemplo



En uno de los ejes tenemos el trabajo que tenemos que realizar y en el otro de los ejes tenemos el tiempo. Se supone que los 2000 puntos deberíamos consumirlos o quemarlos durante los 30 días. Se supone que el ultimo día tendríamos que terminar de hacer todo el trabajo que teníamos planificado.

Burndown charts

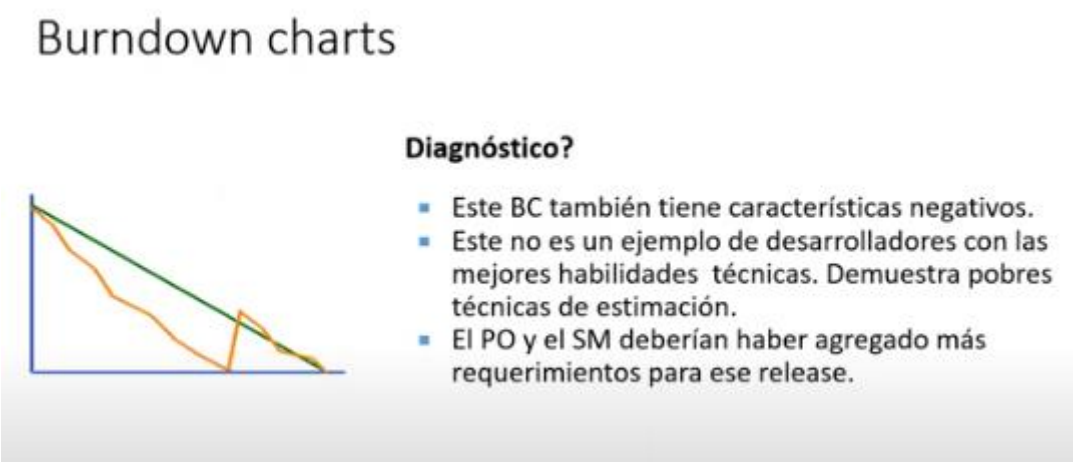
Diagnóstico?



- Este BC es mucho mejor que el anterior
- Al comienzo del sprint hubo problemas, pero en el medio el equipo de trabajo reaccionó y tomo medidas correctivas. Por la línea se nota que el PO quitó requerimientos para mantener la fecha de release.
- También pone en evidencia un mejor manejo de riesgos lo que facilitó aumentar la velocidad en la segunda parte del sprint.

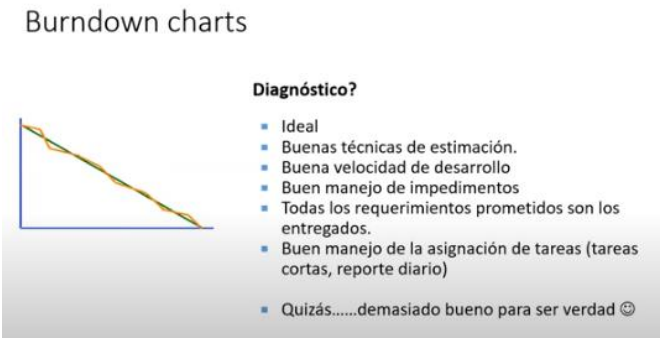
En este BC se nota una mejora con respecto al anterior, al comienzo del sprint hubo problemas igual que en el anterior, pero en el medio el equipo de trabajo reaccionó a tiempo y tomó medidas correctivas y se pudo resolver. Una posibilidad fue que Por la línea se nota que el PO quitó requerimientos para mantener la fecha de release acordando con el equipo, sin modificar el alcance.

También pone en evidencia un mejor manejo de riesgos, lo que facilitó aumentar la velocidad en la segunda parte del sprint.



Es posible que se haya estimado mal, se sobreestimó. Lo que llevó a agregar funcionalidades. Lo mas probable es que se hayan agregado requerimientos para que no queden dias sin hacer nada.

ESCENARIO IDEAL



Hicimos todo lo que prometimos, prácticamente vamos en línea con la línea perfecta.

“SCRUM NO ES PARA TODOS, SINO PARA AQUELLOS QUE TIENEN QUE TRABAJAR CON SISTEMAS FUNCIONANDO CON LA COMPLEJIDAD DE TECNOLOGÍAS INESTABLES Y EL SURGIMIENTO DE REQUERIMIENTOS. “

Este concepto es importante, es un poco lo que hablamos antes, dominios impredecibles, tecnología, cambio de requerimientos, en esos contextos scrum tiene utilidad y es útil y funciona, pero no es para todos los proyectos ni es para todos los equipos. Si consideramos los cimientos hablamos de personas que deben hacer todo todos, autoorganización. No todos los equipos pueden configurarse de esa manera, no todos pueden aplicar scrum y no en todos los proyectos resulta útil.

MÉTRICAS ÁGILES

A diferencia de las métricas tradicionales, en las métricas agiles hay una mirada minimalista. Las tradicionales apuntan a actividades de monitoreo y control.

LA MEDICION ES UNA SALIDA, NO ES UNA ACTIVIDAD.

Cuando hablamos de agilismo y métricas agiles hablamos de medir solamente lo estrictamente necesario y nada más. ¿Y cuando hablamos de lo estrictamente necesario tenemos en cuenta una definición que es muy importante y que viene del manifiesto ágil y que nos dice que la principal medida de progreso cual era? EL SOFTWARE FUNCIONANDO. Este criterio es el que nos va a servir de base para definir cuáles son las métricas que nosotros vamos a utilizar. Que la principal medida del progreso es el sw funcionando.

Acá también vamos a tener una mirada mínima sobre las métricas que vamos a utilizar.

Otros dos lineamientos que también viene del manifiesto ágil:

- NUESTRA MAYOR PRIORIDAD ES SATISFACER AL CLIENTE POR MEDIO DE ENTREGAS TEMPRANAS Y CONTINUAS DE SOFTWARE FUNCIONANDO.
- EL SOFTWARE TRABAJANDO ES LA PRINCIPAL MEDIDA DE PROGRESO.

Estas son las 2 premisas o los 2 principios del manifiesto ágil que nos guían en la elección de las métricas agiles.

LA PRIMERA METRICA ES LA CAPACIDAD.

La vamos a definir en términos de estimación. Es loco, pero tiene que ver justamente con tomar una medición de algo que ya ocurrió. LA CAPACIDAD SE DEFINE COMO UNA METRICA, PERO ESTÁ BASADA EN UNA ESTIMACIÓN

¿QUÉ SIGNIFICA LA CAPACIDAD? Es cuanto trabajo puede completarse en un periodo de tiempo dado. ¿Cuál sería el periodo de tiempo dado? El sprint. Lo que nosotros vamos a buscar con la capacidad es definir cuanto trabajo podemos hacer en el sprint. ¿PARA QUE QUIERO SABER ESTO? Voy a usar la capacidad en la planning. Porque si yo no defino cuanto trabajo puedo hacer en el sprint, ¿cómo voy a saber cuales son las features que yo voy a incluir en el sprint? ¿Cuándo corto? Corto en función del trabajo que yo puedo hacer.

¿Qué problema habría en definir la capacidad en términos de story points? EQUIVOCARSE POR SER UN EQUIPO INEXPERTO. Porque en realidad el story point es una medida que surge de la experiencia que tiene el equipo y que es propia de ese equipo. Entonces normalmente lo que se hace primeramente es medir el esfuerzo en horas, dado que es una medida mas tangible, mas fácil. Cuanto va a trabajar cada uno. 8hs. – Y de es as 8hs cuanto vamos a considerar que son productivas. 5,5hs – cuantos dias a la semana vamos a trabajar. Tanto. Tal va a estar de vacaciones en tal sprint Etc. En función a esa operación vamos a tener la cantidad de esfuerzo o la cantidad de horas.

Capacidad

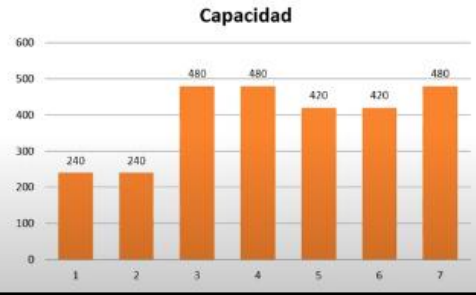
- **Horas de Trabajo Disponibles por día** (WH) X **Días Disponibles Iteración** (DA) = Capacidad

WH x DA = Capacidad

- Ejemplo:
  - Equipo de 8 miembros
  - 4 miembros disponibles los 2 primeros sprints.
  - 1 miembro se casa en sprints 5 y 6
  - 6 horas de trabajo

Capacidad

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



- La capacidad es una estimación de cuanto trabajo puede completarse en un periodo de tiempo dado. Basado en la cantidad de tiempo ideal disponible del equipo.

La capacidad se RATIFICA en la planning.

No necesariamente hay una relación lineal entre las horas y los puntos de historia! Justamente eso aprendemos en base a experiencia.

Se puede medir en

- Esfuerzo (horas)
- Puntos de historia (Story points)

CONSIDERACIONES DE LA CAPACIDAD.

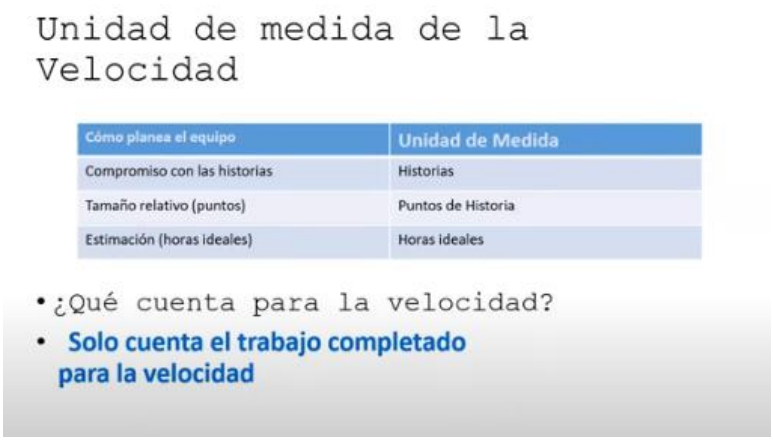
- Tener una capacidad realista del calculo (no decir trabajamos 8 hs, sabemos que no son las hs reales de trabajo), es decir, individuos calculan capacidad realista.
- Aplicar estimaciones honestas a sus tareas.
- Considerar una capacidad máxima de 50% - 70% (tiene que ver con ser realista)
- Comprender la capacidad a largo plazo con la velocidad y los puntos de historia.
- Cual es el promedio de finalización de un punto de historia para un equipo/individuo.
- Y poder empezar a modelar esa capacidad en el tiempo en función de lo que realmente va pasando, de la experiencia que vamos adquiriendo.

- No es la idea dejar un margen en el sprint hasta acumular experiencia necesaria sino mas bien hacerlo lo mas a conciencia que se pueda sabiendo que nos vamos a equivocar. Normalmente en los primeros sprint cuando uno termina probablemente en términos de velocidad la velocidad sea 0, habían estimado cosas que no pudieron ni siquiera empezar. Pero no importa porque los primeros sprints son de aprendizaje. Pero para poder hacer ese aprendizaje tengo que ser lo mas honesto posible.
  En función de eso ir aceitando esta estimación de la capacidad conforme pasa el tiempo y poder ir ajustándola.

  LA MEDIDA DE LA VELOCIDAD, OTRA METRICA IMPORTANTE
  Es la 2da métrica que vamos a utilizar, a diferencia de la capacidad que se trabaja en la planning, la velocidad si es concretamente UNA MEDIDA, una observación y que la vamos a obtener completamente en la REVIEW.
  ¿CÓMO LA VAMOS A OBTENER?
  Vamos a sumar la cantidad de puntos de historias de las features/user terminadas que han sido probadas por el PO.
  ¿POR QUÉ DECIMOS QUE LA VELOCIDAD RETROALIMENTA LA CAPACIDAD?
  Porque en función de lo que me pasó, en función de los puntos de historia que realmente me aceptaron en el sprint, me va a servir a mi de indicador a la hora de plantear próxima planning. No solo me sirve para eso sino que además podemos ver que es o cual es la medida del trabajo que el equipo va realizando. Nos permite ver como trabaja e equipo, qué es lo que logra.
  En los primeros sprints sin experiencia del equipo la velocidad probablemente sea 0, y probablemente no tengamos ningún valor de velocidad que resulte satisfactorio. Y no importa porque esto es parte del aprendizaje, el equipo tiene que conocerse y nuestro objetivo es que a medida que los sprint transcurran el equipo logre una velocidad estable y que se pueda predecir salvo que un miembro del equipo se vaya o no este trabajando. Si la velocidad es estable, la definición de la capacidad se vuelve mas fácil porque ya puedo predecir que va a pasar.

- Es una observación empírica de la capacidad del equipo para completar el trabajo por iteración.
- Comparable entre iteraciones para un equipo dado y un proyecto dado.
- NO ES una estimación
- NO ES un objetivo a alcanzar
- NO ES comparable entre equipos
- NO ES comparable entre proyectos.

Tanto la velocidad como la capacidad no son comparables, ya que depende de un equipo.



Para la velocidad solo cuentan las historias TERMINADAS.

TESTING (Prueba de software)

Importante tener en cuenta, el testing no es la actividad que nos permite ni asegurar la calidad ni desarrollar un sw de calidad.

Es una disciplina en donde contemplamos aspectos de productos o requerimientos de calidad. ¡¡Pero el testing NO es la actividad que nos permite asegurar que nuestro proceso de construir el sw sea de calidad!!

¿Por qué? Porque la calidad del sw no abarca solamente aspectos del producto. Si nos concentráramos en aspectos del producto estamos viendo solo una parte de lo que tiene que ver con la calidad del sw.

Y por otro lado porque hay otras actividades o disciplinas que incluso se ejecutan antes del testing y que también nos permite asegurar la calidad del producto que vamos a construir. Esas actividades tienen que ver con la inspección o revisiones técnicas.

¿Por qué el testing es necesario?

- **Porque la existencia de defectos en el sw es inevitable**. Esto es verdadero porque el testing es un proceso que nosotros iniciamos asumiendo que va a haber defectos en el sw porque el sw está hecho por personas, escritos por desarrolladores. El objetivo de ejecutar la actividad de testing tiene que ver con que suponemos que vamos a encontrar defectos en el sw.

- **Para llenar el tiempo entre fin del desarrollo y el día del reléase.** FALSO, Claramente no.

- **Para probar que no hay defectos**. Esta premisa es falsa ya que está en contraposición de lo que decimos en la 1er premisa. Si decimos que hacemos testing porque asumimos que hay defectos, no podemos probar que no haya.

- **Porque el testing está incluido en el plan de proyecto**? FALSO. El testing no se incluye como un requerimiento, sino que yo decido hacer testing e incluirlo en el plan de proyecto porque es una actividad del proceso necesaria.

- Porque debuggear mi código es rápido y simple -- FALSO, no tiene NADA que ver con el testing. Tiene confusión con algún nivel de testing. Pero no tiene que ver.

- **Para evitar ser demandado por mi cliente**, VERDADERO, claramente SI

- **Para reducir riesgos**, VERDADERO, Claramente si.

- **Para construir la confianza en mi producto**. VERDADERO, Claramente si.

- Porque las fallas son muy costosas. VERDADERO, Claramente si.

- **Para verificar que el sw se ajusta a los requerimientos y validar que las funciones se implementan correctamente**. VERDADERO, Claramente si.

Todas esas son razones por las cuales el testing es necesario.

SIN EMBARGO, HAY QUE ROMPER MITOS.

El testing no es igual a calidad de producto ni tampoco calidad de proceso.

¿¿El testing es una etapa que comienza al terminar de codificar?? NOOOO **Puede comenzar mucho antes.** Incluso desde el momento que empiezo a definir los requerimientos, o incluso cuando empiezo a definir el plan de proyecto.

El sw no es probar que el sw funcione... Es un PROCESO DESTRUCTUVO cuyo objetivo es encontrar defectos.

El tester no es enemigo del programador, solo que debería ser un trabajo de sinergia.

PROCESO DESTRUCTIVO, PORQUE SU OBJETIVO ES ROMPER EL SW, ENCONTRAR DEFECTOS. DE TRATAR DE ENCONTRAR DEFECTOS EN EL CODIGO.

DEBE SER UNA ACTITUD NEGATIVA PARA DEMOSTRAR QUE ALGO ES INCORRECTO, YA QUE SE ASUME QUE VAMOS A ENCONTRAR DEFECTOS. "TENGO QUE ENCONTRAR LOS DEFECTOS QUE YA SE QUE ESTAN" UN TESTING EXISTOSO ES AQUEL QUE ENCUENTRA DEFECTOS.

MUNDIALMENTE, **CUANDO HABLAMOS DE SW CONFIABLE, HABLAMOS DE QUE DESDE EL 30 AL 50% DEL COSTO DE UN SW CONFIABLE SE NOS VA EN EL TESTING.**

Cuando tenemos que pensar en una estimación de esfuerzo relacionada con el testing, y asociado a ese esfuerzo el costo, el testing se lleva esa parte del esfuerzo de construir un sw confiable.

PRINCIPIOS BASICOS DEL TESTING.

- El testing muestra presencia de defecto. Busca los defectos.
- El testing exhaustivo es IMPOSIBLE. No es viable en términos de costo, de esfuerzos y de tiempo probar el sw de manera completa. ¿Qué vamos a hacer nosotros? abarcar la mayor cantidad de casos posibles dentro del testing con el menor esfuerzo posible. NUNCA VOY A PODER PROBAR EL 100% DE MANERA COMPLETA.

- Testing temprano. Empieza desde que yo empiezo a construir los casos de prueba, desde el momento que los empiezo a planificar y empiezo a definir el nivel de cobertura del testing.
- Agrupamiento de defectos. En función del universo que yo voy probando y de los CASOS DE PRUEBA que voy definiendo.
- Paradoja del Pesticida. A medida que vamos ejecutando el testing de sw nos vamos a encontrar con esta paradoja, cuando uno utiliza muchas veces un pesticida probablemente no tenga problemas con la plaga que yo quiera matar, pero si con otras que no veo porque estoy concentrada en una sola plaga.
  Yo quizá pueda armar casos de prueba que se ejecuten de manera automática y poder correrlos y eliminar todos los defectos y dejar cosas afuera que no estoy viendo porque me estoy concentrando en los defectos que estoy tratando en ese contexto. ¡Ocurre bastante en testing automatizado! --> **por eso muchas veces el testing automatizado va muy de la mano con algún testing manual.**
- El testing es dependiente del contexto.
- Falacia de la ausencia de errores.

Encontrar defectos corregirlos y volver a testear es todo parte del testing.

¿Cuándo sabemos cuándo vamos a dejar de testear? No es cuando encontramos un defecto sino cuando corregimos o cuando termino mi ciclo de prueba y decido que los defectos que quedan asumo que los dejo porque no tienen el suficiente nivel de severidad. Todo eso es parte del testing.

El 100% de cobertura es imposible.

El programador debería evitar probar su propio Código.. es evidente que es difícil encontrar defectos en mi propio Código. (solo en el primer nivel de testing - testing unitario - es donde un programador puede testear pero sino NO).

Ni una unidad de programación debería probar sus propios desarrollos.

Nunca debemos planificar el esfuerzo de testing suponiendo o planificando que no vamos a encontrar defectos. Ya que asumimos que los defectos están.

¿Hacia dónde apuntamos para encontrar esos defectos? Debemos tener en cuenta de buscar los defectos pensando en que el sw haga lo que se supone que debería hacer y también **pensando en lo que el sw NO se supone que debe hacer.**

¿CUANTO TESTING ES SUFICIENTE?

- **Cuando se terminó todo lo planificado** FALSO
- **Nunca es suficiente** FALSO
- **Cuando se ha probado que el sistema funciona correctamente** FALSO
- **Cuando se tiene la confianza de que el sistema funciona correctamente** VERDADERO
- **Depende del riesgo de tu sistema** VERDADERO

¿Como se cuándo dejo de probar?

Supongamos un sistema con 20 pantallas, supongamos promedio de 3 menus por pantallas con 4 opciones en cada menu, con un promedio de 10 campos x pantalla. y dentro de cada campo la opcion de ingresar numeros de 2 digitos con 101 valores posibles.

Si suponemos que cada caso de prueba nos lleva 1 min tenemos para una persona si hace testing exhaustivo 1 año y medio. Sin incluir testing de regresion.

Debemos asegurarnos de encontrar la mayor cantidad de defectos

entonces vamos a considerar que realizamos la cantidad de ciclos de prueba suficiente cuando tengo la confianza de que mi sw funciona correctamente.

¿Cómo obtenemos esa medida para determinar que el sw que estoy entregando es confiable?

¡Depende también del riesgo de tu sistema! No es lo mismo si tenemos que inyectar una droga a un paciente, a enviar pedidos a una casa.

CONCLUSIONES

- Una variable a tener en cuenta para saber cuándo dejar de testear es el nivel de exposición al riesgo. Cuanto mayor es el riesgo que manejo más voy a tener que testear, más esfuerzo y dinero tengo que poner.

- Otra variable es los costos asociados al proyecto. Va en conjunto con el riesgo.

En el caso de los riesgos vamos a determinar en función del riesgo, las funcionalidades que vamos a testear primero, a qué le vamos a dedicar esfuerzo y que es lo que por ahora podemos no testear.

¿Cómo medimos todo esto que hablamos antes? con los CRITERIOS DE ACEPTACIÓN

Es algo que yo acuerdo con mi cliente. Mi cliente tiene que acordar conmigo hasta donde yo voy a probar. Esto me va a servir de respaldo en el caso que después se encuentren defectos.

¿Cómo los defino? tiene que ser con variables que puedan medirse.

Puede ser en términos de costos, "voy a usar tantas horas en el testing"

Voy a ejecutar hasta que haya determinada cantidad de casos no tenga fallas.

Voy a probar en tanto sigan apareciendo defectos de severidad ALTA. Si tenemos de severidad leve o menor, no sigo testeando eso.

Se acuerda todo esto para saber hasta dónde aprobar. Son ejemplos de cómo cuantificar los criterios de aceptación. para no tener ambigüedades en el medio. Pueden existir otras formas de definir los criterios de aceptación. Lo importante es que el cliente los conozca.

Fallas predichas aún permanecen en el ws --> quiere decir que recordemos cuando nosotros entreguemos el sw, el sw va a seguir teniendo defectos y eso lo tenemos que saber, y también hacerlo saber al cliente. por qué? PORQUE EL TESTING NO ES EXHAUSTIVO. Che ppero vos no probaste. NONO si, hice el testing con este criterio de aceptación. Los defectos permanecen en el sw porque no tengo manera de probar el sw de manera completa. Hasta hay funcionalidades que quedan relegadas de su uso y luego se empiezan a utilizar.

Es una locura que haya gente que crea que no va a haber posibilidades de que haya un defecto. Queda en la habilidad del líder de proyecto de que se le advierta que no va a existir ni un defecto.

En proyectos agiles es más fácil que en los tradicionales. Es mejor ser claros y explicarles esto y no prometer algo que no se puede cumplir dado que es una realidad. básicamente el que avisa no traiciona jajaja

Cuando le contas el costo ahí te das cuenta de que a veces es mejor no asumir el testing de algunas cosas porque si no te terminaría saliendo más caro el testing que desarrollar el sw.

La psicología del testing.

- la búsqueda de fallas puede ser visto como una crítica al producto y/o su autor.

- la construccion del sw requiere otra mentalidad a la de testear el sw.

la mentalidad de quien testea no es criticar al desarrollador o que hizo mal su trabajo. Nosotros nos paramos en un lugar donde la presencia de defectos EXISTE. parados en ese lugar o si todo el equipo comprende este contexto evitamos esta rivalidad entre el tester y programador. La búsqueda de defectos es una tarea o actividad orientada a la calidad del producto que estamos construyendo.

Cada uno tiene su rol y cada rol es igual de importante.

ERROR VS DEFECTO.

- cuando hablamos de testing usamos el termino defecto, no error! Porque cuando hablamos en testing que encontramos defectos en esa etapa de testing, fueron generados en una etapa anterior, en una etapa de implementación es decir, cuando se escribió el código se generó el error. Y cuando ese error entró en la etapa siguiente ese error se transformó en defecto.

El error se genera en la misma etapa que se genera. El defecto se detecta en una etapa posterior.

SEVERIDAD VS PRIORIDAD

- Severidad: bloqueante, critico, mayor, menor, cosmético --> tiene que ver con qué pasa si el defecto permanece en el sw. por ejemplo: bloqueante- si el defecto existe no puedo seguir ejecutando el CP. Critico y mayor parecidos. que la funcionalidad no bloquea pero no va a tener la funcionalidad correcta. Menor ejemplo: Hay una funcionalidad que tiene un curso alternativo y no me tiene que dejar seguir avanzando y me muestra y el mensaje dice algo que no corresponde. El sw no va a ejeutar lo que tiene que ejecutar y el mensaje al usuario no es claro. Y asi va bajando. Cósmeticos (errores de ortografia, etiquetas boludeces)
- Prioridad: Urgencia, alta, media, baja --> tiene que ver con la prioridad de la funcionalidad y del CP en el contexto de la funcionalidad dentro del negocio. Sería la prioridad de tratamiento a la hora de resolver el defecto. No hay una relación directa entre severidad y prioridad.

Bloqueante y critico sería en función del CASO DE PRUEBA que estoy ejecutando. No tiene que ver con que tan crítico es en el negocio.

(existe el testing funcional y no funcional (el no funcional sería seguridad, performances))

NIVELES DE PRUEBA

Tienen que ver como yo voy escalando en términos de lo que voy probando haciendo un enfoque desde la menor a mayor granularidad.

1 nivel: testing unitario. donde yo Hago foco sobre un componente. termino de construir el componente de manera individual y lo testeo, pero de una manera independiente (no hago testeo vinculado con otros componentes) este es el que hace el mismo programador. Es el primer testing que es candidato a ser automatizado.

Hay manera de construir Código escribiendo el testing automatizado y a partir de ahí construir el Código. TDD SE LLAMA.

Se ejecuta el testing unitario, se corrigen los errores y sigue adelante.

2 nivel: testing de integración: empezar a juntar las partes/componentes de manera integrada. Normalmente cuando se empieza a juntar, hay un esquema de integrar esos componentes de manera INCREMENTAL. No es que los vamos a integrar a todos juntos. A medida que los vamos integrando, vamos probando y a medida que se van integrando de manera exitosa continuamos integrando otros.

Siempre se integra primero los módulos críticos para el negocio y de ahí extensivo hacia las otras funcionalidades secundarias para el negocio o de menor prioridad.

3 nivel: testing de sistema: probamos la aplicación funcionando como un todo. Ya empezamos a hacer pruebas funcionales y no funcionales. Y tratamos de probar en un ambiente lo más parecido al entorno de producción. (preproductivo) acá si tenemos en cuenta seguridad, carga, performance.

4 nivel: testing de aceptación: lo hace el usuario, no es un testing técnico. Lo hace el usuario para ver si se ajusta a sus necesidades. acá no espera destruir, su foco es no encontrar defectos. Sino que sea lo más parecido al entorno final de producción (se hacen pruebas alfa en ambiente de laboratorio, o beta con datos reales) EL objetivo es que el usuario nos dé el OK.

Si tenemos que hacer una relación con el framework de scrum se haría en el momento de la review. (demo)

PROCESO DE PRUEBA

En métodos tradicionales o procesos definidos, la prueba es también un proceso. Es un proceso que tiene distintas etapas. En scrum es diferente (no ocurre este proceso)

1. planificación: armar plan de pruebas, definimos los criterios de aceptación. Plan subsidiario al plan de proyecto.

- construir el plan de pruebas

- acordamos metas y objetivos con nuestro cliente. criterios de aceptación.

- definimos cual va a ser la estrategia de testing.

- cómo vamos a hacer las pruebas. cuales manuales cuales automatizadas.

- cuales van a ser nuestras coberturas.

la planificación es algo VIVO. se va retroalimentando permanentemente. y en cualquier momento puedo modificar.

2. Identificacion y especificacion: de los casos de pruebas, qué es lo que vamos a probar. ambas van de la mano

- tiene que ver con determinar qué vamos a probar.

- definir los casos de prueba

- identificar los datos para esos casos de prueba. Priorizarlos

- Definir y diseñar el entorno de prueba sobre el cual vamos a ejecutar los casos.

Aca podemos ejecutar los casos aunque no tengamos escrita una sola linea de codigo. Aca necesitamos tener definidos los req. funcionales y no funcionales. El codigo no es necesario tenerlo. Es mas, es mejor que se haga esta actividad de manera paralela con el desarrollo.

3. Ejecución: ejecutamos los CP. Y vemos cuales son los resultados. Si hay defectos o si no llegamos a los criterios de aceptacion, volvemos a ejecutar y volvemos a analizar las fallas. ¿HASTA CUANDO? hasta cumplir con los criterios de aceptacion. Ejecutar los casos de pruebas definidos. QUE IMPLICA? ejecutarlos y automatizar los que se puedan si es que esperamos automatizar.. se empezaría desde los unitarios. Definimos los prioritarios para ejecutarlos. Definimos cuales constituyen un ciclo de prueba.

Una vez terminada la ejecucion lo mas importante es (teniendo en cuenta que en la etapa de identificacion y especificacion definiamos los RESULTADOS ESPERADOS de los casos de prueba entonces...) lo mas importante una vez terminada la ejecucion es --> COMPROBAR QUE LOS RESULTADOS QUE OBTENEMOS SEAN O NO LOS RESULTADOS QUE HABIAMOS DEFINIDO COMO ESPERADO. Si los resultados son los esperados, el CP pasa. Si no es esperado, FALLA y hay que corregir y volver a ejecutar. A su vez definiremos el defecto que se haya encontrado definiendo su severidad y prioridad de corrección.

4. Evaluación y reporte, tiene que ver con, una vez que terminamos el ciclo de prueba decimos, cuáles son los defectos que encontramos? ¿que severidad tiene? llegamos a cumplir los criterios de aceptación o no llegamos a cumplir? seguimos probando o no seguimos probando??

CUANTAS LINEAS DE CODIGO NECESITO PARA EMPEZAR A HACER TESTING? CERO Porque todas las actividades de planiificacion, identificacion, de preparar el ambiente no necesito escribir ninguna linea de codigo.

EN SCRUM ? COMO HACEMOS ESTE PROCESO?

La actividad de testing queda embebida dentro de las actividaddes que se ejecutan en el contexto del sprint. El testing se HACE.

Tiene en terminos de CP y de ejecucion tiene las mismas caracteristicas que cuando trabajamos en meotodologia tradicional, solamente que no vmos a tener un proceso definido con estas caracteristicas vistas recien. Si no que queda embebida dentro del sprint y tratando de ENCOTNRAR LOS DEFECTOS LO ANTES POSIBLE Y CORREGIRLOS DE LA MANERA MAS RAPIDA POSIBLE.

QUE SERÍA UN CASO DE PRUEBA (CP)??

Tiene que ver con escribir una secuencia de pasos que tienen condiciones y variables con las que yo voy a ejecutar el sw y me van a permitir saber si el sw esta funcionando correctamente o no.

La buena definicion del CP nos ayuda a reproducir defectos!

EJEMPLO: Supongamos que estamos probando un sw para un GPS, y el caso de prueba es PROBAR BUSCAR UNA CALLE.

condicion o variable que yo defino a la hora de pensar en hacer el caso de prueba: voy a buscar una calle que ya esté cargada en el gps.

Entonces si vamos a buscar la calle san lorenzo, yo me voy a asegurar que cuando ejecute el CP esa calle esté cargada en el gps con una altura determinada

Ingreso en el GPS la calle SL, ingreso la altura 720 y pongo BUSCAR y cual es el resultado esperado? que el sistema me ubique cual es la calle y altura que ingrese.

QUE ES LO IMPORTANTE DE LA DEFINICION DEL CP??

primero, que yo tengo esas condiciones y variables determinadas claramente especificadas y tengo bien definidas cual es la secuencia de pasos que tengo que ejecutar con esas variables para lograr el resultado esperado.

Despues si da o no el resultado esperado es otro asunto.

Todo esto es fundamental prque si el rstado esperado no se logra, quien tiene que reproducit un defecto para corregirlo no tiene mas que seguir los pasos enunciados en el CP.

Si esto no esta bien definido, y me pongo a probar lo que me parezca, cuando el defecto aparezca puede que no sea facil de reproducir el defecto. Puedo tener suerte, o si no es facil de reproducir, porque yo no me guie con un CP que estaba definido.

EL CP ES FUNDAMENTAL PARA QUE EL TESTER SEPA QUE ES LO QUE ESTA TESTEANDO, Y SEPA QUE ES LO QUE EJECUTA, SOBRE TODO PARA VALIDAR SI EL RESULTADO QUE OBTENGO ES EL ESPERADO Y SI NO LO ES, ES FUNDAMENTAL PARA SABER COMO HICE PARA LLEGAR A ESE DEFECTO Y PODER CORREGIRLO.

CONDICIONES DE PRUEBA

En el caso de prueba del ejemplo, una condicion seria "que la calle san lorenzo esté cargada en el gps"

Estas son fundamentales, porque no es lo mismo que la calle esté o no cargada. Porque me cambiaria el resultado esperado!

CARACTERISTICAS DEL CP

tiene un objetivo que tiene que ver con qué es lo que yo voy a probar-

datos o condiciones previas o precondiciones que voy a tener que contemplar.

Una vez que ejecuto el caso, tiene siempre un coportamiento esperado.

Con el ejemplo que dimos tenemos como minimo 2 ejemplos: probar con una calle que exista y despues una que no exista "no se puede localizar la ubicación ingresada"

LOS BUGS SE ESCONDEN EN LAS ESQUINAS Y SE CONGREGAN EN LOS LIMITES:

uno de los aspectoss a tener en cuenta (que vamos a ver en el practico) dado que el testing exhaustivo es imposible la idea es probar lo más que podiamos del sistema con el menor esfuerzo posible. Pensar como hago para ejecutar la menor cant de casos de prueba pero que me aseguren la mayor cobertura que me permita cubrir varios aspectos de la funcionalidad.

Una de las premisas con el planteo de casos de prueba es que los bugs y los errores se concentran en los lugares limites. esta premisa es a la hora de definir casos de prueba. **hablamos de lugares limites cuando trabajamos con rangos.**

EJEMPLO DEL GPS: si estamos hablando de que tienen las calles alturas. y la calle va del 0 alm 790 entonces se va a probar con el limite, que sea 791, el 0 y el 1. Para contemplar la mayor cantidad de universos posibles. Por que¨? porque si pruebo el 790 y anda, es muy probable que el 500 tambien me funcione. Entonces probemos donde es mas probable que haya errores.

Hay una tecnica, prueba de caja negra por valores limites. lo vamos a ver en profundidad en el practico.

CÓMO ENUNCIO LOS CP?

Si tenemos requerimientos planteados o US podemos empezar desde ahi. Si tenemos US NO SON LAS PRUEBAS DE USUARIOS ESCRITAS EN LA USER. Las pruebas en las US tienen que ver con los criterios de aceptacion y pruebas de aceptacion que se hacen en la review que tiene que ver con a donde testeo!! **Pero no son los casos de prueba que estamos hablando ahora que son del equipo pra adentro.**

Obvio las pruebas de aceptacion nos van a ayudar a definir los nuestros porque es lo que se espera para ver si se acepta o no! PERO SE HACEN EN ESE ULTIMO NIVEL DE ACEPTACION.

Estos casos de prueba que estamos viendo son de los que estamos viendo ahora, de pruebas unitarias, de integracion y de sistema.

Mientras mejor esten especificados los requerimientos, mas facil eshacer los casos de prueba.

CONCLUSIONES SOBRE LA GENERACION DE CASOS DE PRUEBA.

- vamos a tener distitnas tecnicas! (caja negra, caja blanca, combinadas) ninguna tecnica es completa! Ninguna tecnica me permite abarcar todos los casos de prueba. Lo que nosotros buscamos es usar distintas tecnicas complementarias entre si para tratar de definir o de **abarcar la prueba de la mayor cantidad de sw posible con la menor cant de CP posible**. CÓMOO LO VAMOS A LOGRAR? USANDO ESTAS TECNICAS que nos ayudan a enunciar los CP de tal manera que esos nos abarquen la mayor cantidad de escenarios posibles.

Si no tenemos requerimientos definidos, es mucho mas dificil. Si estuvieran bien especificados, el trabajo se nos simplifica de manera exponencial.

Todas ls tecnicas tienen ventajas y desventajas que apuntan a cuestiones diferentes. En el practico vemos caja blanca y negra. Apuntan a cosas distintas.

CICLO DE TEST

Supongamos que tenemos un conjunto de casos de pruebas con distintas tecnicas ya definidas, con 10 CP. Ejecutamos los 10. Algunas veces obtenemos resultado esperado y otros con defectos de distinta severidad. Si terminamos la ejecucion y enccontramos 20 defectos bloqueantes, y ahi que hacemos? tenemos que ir a corregirlos. y directamente derivado de eso, hay que volver a probar para ver si se corrigieron o no. **Cada vez que yo ejecuto todos los CP definidos es a una misma version del sistema, un ciclo de test.** Hago una regresión, vuelvo a ejecutar. otro ciclo. Siempre hablando de la misma version! Cuantos ciclos se van a ejecutar? Depende de los resultaados de la ejecucion de los CP. Si dijimos en los criterios de aceptacion que vamos a ejecutar hasta que no haya mas defectos bloqueantes, **entonces Se van a ejecutar tantos ciclos de test como sean necesarios hasta que no haya mas bloqueantes.**

De la mano, **va la REGRESIÓN.** La regresion está directamente relacionada con el ciclo de prueba.

Si la regresion forma parte del testing.

La regresion nos dice que cuando terminamos un ciclo de prueba y reemplazamos la version habiendo corregido todos los defectos, **hay que volver a verificar no solamente los CP donde encontramos los defectos, sino que hay que volver a ejecutar TODOS los CP de maenra completa**, porque probablemente cuando intentamos corregir un defecto normalmente se produce un nuevo defecto y si nos limitamos solamente a probar los CP en los que habiamos detectado los defectos, se nos pueden pasar por alto algunos defectos que saltaron con la correccion.

CANDE MIRA EL EJEMPLO DEL VIDEO DE TESTING DE CAJA NEGRA, PARTE 3 MINUTO 4:20

Probablemente arreglando e casp de prueba 1 se rompe el 2.

En la regresion hay que probar TODOOOO de nuevo.

SMOKE TEST

Tiene que ver con los test de humo, con hacer una primera corrida del sistema en términos generales, en terminos basicos, no exhaustivo. Una pasada rapidita de todo el sistema para ver que no haya ninguna falla catastrofica. que mas o menos anda. para no abocarme a la ejecucion detallada de los CP cuando hay alguna cosa mas grosera que se me esta pasando.

DISTINTIOS TIPO DE PRUEBA (FUNCIONAL Y NO FUNCIOANL)

- TESTING FUNCIONAL --> BASADO EN LOS REQUERIMIENTOS

- NO FUNCIONAL --> ESTRES, MANTENIMIENTO, PERFORMANCE, SEGURIDAD, USABILIDAD, POSTABILIDAD, FIABILIDAD, CARGA

EL no funcional TAMBIEN ES IMPORTANTE. tiene vital importancia para las pruebas preproductivas! porque tiene que ser practicamente un ambiente igual al entorno para ver si soporta todo esto el sistema porque sino no tendrian sentido.

- PRUEBAS DE INTERFACES DE USUARIOS. Cada vez son mas complejas. Son pruebas mas especificas hoy en dia.

- PRUEBAS DE PERFORMANCE

- PRUEBA DE CONFIGURACION --> prueba de mi entorno cn una determinada configuracion de cada uno de sus componentes.

Todos estos que estan arriba son ejemplos de testing NO funcional.

TESTINGG Y CICLO DE VIDA

Existen distintos tipos de CICLO DE VIDA.

verificación y validcion

- verificacion ESTAMOS CONSTRUYENDO EL SISTEMA CORRECTAMENTE?

-VALIDACION ESTAMOS CONSTRUYENDO EL SISTEMA CORRECTO? el que nos estan pdiiendo?

Dependiendo del ciclo de vida, el testing se inserta de una manera distinta.

Es imporatnte usar aquellos ciclos de vida donde el testing se comience a ejecutar lo mas temprano posible. POR QUE? porque nos da visibilidad de que tenemos que probar, hasta nos ayuda a ver si los req func y no func estan correctamente especificados, y ademas cuanto mas temprano lo introducimos, los costos de correcciones de defectos disminuyen.

Existen 2 estrategias para el diseño de casos de prueba.

CAJA BLANCA Y NEGRA.

En las de caja blanca --> nosotros podemos ver el detalle de la implementación de la funcionalidad, vamos a poder ver el codigo y vamos a poder diseñar nuestro CP para poder garantizas COBERTURA (si quiero cubrir todos los if, todos los else, si quiero cubrir todas las ramas de ejecucion de cierta funcionaldidad, como yo veo el codigo puedo guiar la ejecucion de cp por donde yo quiera, conozco la estructura interna)

En la de caja negra yo no conozco la estructura interna de la imple, solo voy a analizar en terminos de entradas y salidas. Voy a identificar cuales son las diferentes entradas que una funcionalidad puede tener y voy a elegir los valores con los cuales voy a ejecutar esa funcionalidad y finalmente voy a comparar las salidas obtenidas contra las que esperaba tener.

Dentro de las estrategias tenemos distintos metodos. Para que usamos esos metodos? **para maximizar la cantidad de defectos encontrados.**

Dentro de caja negra tenemos 2 tipos de metodos en particular:

-> los basados en especificaciones

- PARTICION DE EQUIVALENCIAS o clases de equivalencias: analiza primero cuales son las diferentes condiciones externas (tanto entradas como salidas posibles) que van a estar involucradas en el desarrollo de una funcionalidad. Me planto frente a la funcionalidad e identifico las entradas y salidas posibles, las entradas pueden ser cualquier tipo de variable que puede estar en juego tanto sea un campo de texto en el cual yo escribo o un combo de seleccion en el cual elijo alguna opcion posible, o tambien pueden ser las coordenadas en las que yo em encuentro actualmente, o la medicion de un sensor de temperatura, etc. Puede ser cualquier condicion que va a guiar la ejecucion de la funcionalidad. La salida puede ser mostrar un mensaje en pantalla, o la luz en un crontrol remoto, o la emision de un mensaje a traves de frecuencia modulada. (nos tenemos que abstraer a cualquier producto de sw. por eso esos ejemplos, no pensemos solo en una interfaz grafica)

Una vez que identifique estas condiciones externas (entrada/salida) para cada condicion externa yo voy a analizar cuales son los subconjuntos de valores posibles que pueden tomar cada una de esas condiciones externas que produce un resultado equivalente.

DOS PASOS

1. Identificar las clases de equivalencias (validas y no validas) Son estos subconjuntos que cualquier valor que yo tome de subconjunto ante la ejecucion de la funcionalidad produce un resultado equivalente por ejemplo, un app que vende bebidas alcoholicas lo primero qe te va a pedir es la edad, si tenes 18. En ese caso tenes como variable de entrada la edad, y como variable de salida si entras o no dependiendo la edad. Entonces por ejemplo, un subconjunto equivalente sería todos los números menores a 18 van a producir el mismo resultado, es decir no poder ingresar. Es lo mismo si pones que tenes 12 a si tenes 15.

Entonces a simple vista nos surgen a simple vista al menos 2 clases de equivalencia: "números enteros mayores o iguales a 18" o "números enteros menores a 18" "números enteros menores a 0" --> esta ultima seria una clase de equivalencia no valida.

- Rango de valores continuos
- Valores discretos
- Seleccion simple
- Seleccion multiple

2. Identificar casos de prueba

Una vez que identifique esas clases de equivalencia, identificams los casos de prueba, acá es donde el método tiene su resultado. (los metodos solo sirven para poder diseñar los casos de prueba). Yo lo que voy a hacer es tomar de cada una de esas condiciones externas, una clase de equivalencia en particular y Voy a elegir un valor representativo de esa clase de eq para ese caso de prueba.

El caso de prueba que es? es una receta, es un conjunto de pasos ordenados que yo debo seguir para ejecutar la funcionalidad, con una especificacion de los valores que yo voy a ingresar.

La idea de este metodo es que cada caso de prueba tenga UN REPRESENTANTE de cada clase de equivalencia. Osea es un metodo que claramente va a depender de qué tan bien hagas las clases...

PRÁCTICO DE CAJA NEGRA - VER VIDEO TESTING CAJA NEGRA PARTE 2 4K4 MINUTO 10:10

Por ejemplo cuando tengamos que ponerle prioridad al caso de prueba-- LOS DE PRIORIDAD ALTA VAN A SER LOS DE CAMINO FELIZ. Los de prioridad baja van a estar relacionados con validaciones, valores no ingresados, valores que no se corresponden con el formato esperado. Prioridad MEDIA, todo lo que esta al medio.... PRACTICÁ. Combinaciones de valores que puedan ejecutarse bajo ciertas condiciones que quiza produzcan una falla o camino no feliz.

PRECONDICIONES - Es todo el conjunto de valores o caracteristicasq ue tiene que tener mi contexto para que yo pueda llevar adelante este CP particular. Si estuvieramos hablando de una funcionalidad en al que se requiere que el usuario esté logueado o tener ciertos permisos. EJ: El usuario "Juan" está logueado con permisos de administrador.

PASOS - Conjunto de operaciones ordenadas que debe ser totalmente claro sin ambiguedades que el tester debe ejecutar para conseguir el resultado esperado -

1."El ROL ingresa a la opc "ingresar" RESULTADO ESPERADO: "bIENVENIDO AL SITIO WEB"

2. El cliente ingresa "18" en el campo de "edad"

3. El cliente selecciona a la opc "ingresar"

- ANALISIS DE VALORES LIMITES - una implementacion o particularidad de particion de equivalencuas.

-> los basados en la experiencia

- ADIVINANZA DE DEFECTOS

- TESTING EXPLORATORIO

Esta es la variante al metodo de particion de equivalencias.

En vez de tomar cualquier valor de la clase de equivalencia, vamos a tomar los bordes de una clase. Aca vamos a poder llegar a tomar un duplciado.

CAJA BLANCA

En este metodo si disponemos de los detalles de implementacion, es decir, que podemos ver el codigo y en base a eso poder diseñar nuestros casos de prueba.

Se basa en el análisis de la estructura interna del sw o componente de sw. **Se puede garantizar el testing coverage.**

Nos permiten diseñar casos de prueba maximizando la cantidad de defectos encontrados con la menor cantidad de pruebas posibles disponiendo de los detalles de la implementación. Existen diferentes coberturas que permite garantizar.

ESAS COBERTURAS SON (**no son las unicas coberturas**, son en las que nos centramos para el practico)

COBERTURA--> FORMA DE RECORRER LOS DISTINTOS CAMINOS QUE NUESTRO CODIGO NOS PROVEA PARA DESARROLLAR UNA FUNCIONALIDAD.

- Cobertura de enunciados o caminos básicos --> busca poder garantizar que a todos los caminos independientes que tiene nuestra funcionalidad los vamos a recorrer al menos una vez. Podemos decir que nuestro Código ha sido ejecutado pasando por todos los caminos independientes. Nos va a dar una idea de cuantos CP debemos hacer para poder recorrer todo los caminos indep.

Se requiere poder representar la funcionalidad a traves de un diagrama de grafos.

Se calcula la complejidad ciclomatica --

M=Complejidad ciclomática

E=Número de aristas del grafo

F= Números de nodos del grafo

P=Numero de componentes conexos, nos de salida

M = E - N + 2*P --> Nos da la medida de complejidad ciclomática, es decir, la medida de caminos independientes.

M = Número de regiones + 1

Dado un diagrama de flujo se pueden generar casos de pruebas.

-Bucle WHILE DO WHILE

-IF ELSE

-SECUENCIA

Este sería el metodo mas sencillo pero no lo cubrimos. Es una métrica de sw que provee una medicion cuantitativa de la complejidad lógica de un programa

Usada en el contexto de testing, define el número de caminos independientes en el conjunto basico y entrega un limite inferior para el numero de casos necesarios para ejecutar todas las instrucciiones al menos una vez.

EJEMPLO: https://www.youtube.com/watch?v=6IrH0k-2KQo MINUTO 8:19

Tenemos 3 Decisiones que son nuestros IF. (los rombos)

Los caminos independientes serían: desde la raiz tendriamos:

derecha - derecha

derecha - izquierda

izquierda - izquierda

izquierda - derecha

serian 4 caminos independientes. Y si vamos al diagrama de gragos, hay 3 estructuras cerradas entonces y segun la formula M = Numero de regiones + 1 podemos comprobar que en este caso srría 3 + 1= 4 caminos indep.

Para poder garantizar que nosotros cubrimos todos los caminos independientes, cual es la cantidad minima de casos de prueba que necesitamos hacer? Con solo 4 garantizamos la cobertura o de caminos basicos o de caminos independientes. Podemos escribir mas de 4 ? Por mas qe estuviera cubierta la cobertura en si, con los 4 CP, no quita que yo pudiera escribir 100 o 1000 CP. Solo que no es la cantidad minima. Y lo que busca esto es escribir la menor cantidad de CP. Durante todo el practico vamos a intentar esto, **buscar descubrir cual es la cntidad minima de casos de prueba para garantizar la cobertura determinada.**

- Cobertura de sentencias min 20:32

if --> decisiones

cuantas sentencias tenemos en esta funcionalidad del ejemplo?

Hay dos sentencias --> eso es x = x+1 --> asignacion de variable

Una sentencia es cualquier instruccion como asignacion de variable, invocacion de metodos, mostrar un mensaje, lanzar una excepcion. Cualquier instruccion que nosotros demos. Mientras no sean estructura de control! LAS ESTRUCTURAS DE **CONTROL (LOS IF) SON DECISIONES.**

Cual es el objetivo de la cobertura de sentencias? Darle cobertura a todas las sentencias. Buscar cual es la cantidad minima de CP que me permiten pasaar o ejecutar o evaluar o recorrer todas las sentencias. EN el ejemplo, con un solo CASO DE PRUEBA, va. No son excluyentes. con elegir los correctos valores para las 4 variables dsitntas, con un solo caso de prueba puedo curbir todas las sentencias.

- Cobertura de decisión

Una decisión es una estructura de control completa. Cada una de las estructuras de control que nosotros tenemos va a ser cada uno de esos rombos en los cuales podamos tomar un camino o el otro segun si el resultado de esa estructura puede dar true or false.

SI VAMOS AL IF, ES CADA UNO DE LOS PARENTESIS DE LOS IF. IF((A>0) && (C==1))

Podemos tener estructuras de control de if anidadas.

Probar toda las decisiones. buscar la minima cantidad de CP que pruebe todas las decisiones PERO ADEMAS, DEBERIAMOS CUBRIR AMBOS CASOS TANTO SI VA POR EL VERDADERO O FALSO. Lo que queremos ver es si cada una de nuestras decisiones funcionan correctamente y para ver si funcionan bien tengo que ver que si se vaya por la rama del true cuando necesito que sea true y que si se vaya para la rama del false, cuando sea false. Voy a buscar la min cantidad de CP para poder probar todas las decisiones que tengo en mi trozo de codigo, ya sea forzarlo en las 2 ramas T O F. Aca no nos interesa probar las sentencias sino nuestras decisiones.

Cual creemos que es la cantidad de casos de prueba para este ejemplo?

Tenemos que tener en cuenta que adentro del rombo tenemos condiciones, y que cuando estamos en la cobertura de decision NO NOS IMPORTA la combinacion de condiciones de adentro, lo unico que le importa es que de alguna manera salga un true o de alguna manera salga un false!

EN ESTE CASO SERIAN 2 por las variables, YA QUE hay 4 variables, y no estan anidadas! entonces puedo pasar de la primer decision a la 2da si o si!

Si tuviera por ejemplo la variable A tambien como condicion en la 2da decision, si me afectaria!

- Cobertura de condición

Las condiciones son cada una de las evaluaciones logicas unidas por operadores logicos adentro de una decision.

Busca encontrar la menor cantidad de casos de prueba que nos permita la menor cantidad de CP que nos permiten valuar tanto las condiciones en su valor V o F independientemente de por donde salga la decision. Acá es al reves que antes, no nos interesa si sale T O F aca nos interesa hacer todas las combinaciones! de lo que esta adentro de las decisiones.

Para los objetivos del testing de caja blanca no le interesa hacer la salvedad con respecto al cortocircuito de los operadores logicos (esto quiere decir que puedo valuar en el ejemplo, por ejemplo la A Y C en falso al mismo tiempo) Solo le interesa evaluar cada una de las condiciones en su valor V F minimizando la cant.de CP. En este caso al ser todos independientes se puede hacer como min 2 casos de prueba en donde hago todos los valores en false en un CP y todos en true en otro CP.

- Cobertura de decisión/condición

La unica variante con las anteriores es no solamente busca valuar las decisiones en su valor V o F sino tambien valuar todas las condiciones en su valor V y F.

En este caso sería como minimo 2 CP. Si hago que las 2 condiciones sean verdad entonces como es un & salgo por V. Si hago que las 2 condiciones posteriores sean verdaderas tmb al ser un OR sale V. Ahi pude cubrir todas las condiciones verdaderas y a todas las decisiones V. Lo mismo con la falsa.

- Cobertura múltiple

Busca valuar el combinatorio de todas las condiciones en todos sus valores de verdad posible. Platear el combinatorio de los valores de verdad para toda la combinacion de condiciones disponibles. El combinatorio seria:

VV

VF

FV

FF

VV

VF

FV

FF

Considerando que estas variables son independientes se puede hacer como minimo 4 CP. Si estuvieran anidadas uso: El mismo CP donde me da VV lo uso para el resto de las 4 combinaciones! osea que no necesariamente es eque necesitamos 8 CP.

FF

FV

VF

VV VV VV VV

VV VF FV FF

En este caso particular podrian ser 7 CP.

El objetivo de la cob multiple es hacer el combinatorio de los valores de verdad de todas las condiciones adentro de las decisiones. Teniendo en cuenta todos los operadores logicos.

Pra hacer los ejercicios nos conviene hacer el diagrama de flujo.

Framework para Escalar SCRUM

Es un framework de trabajo de desarrollo de sw ágil el cual gestiona PROYECTOS que crean productos de sw, dende tenes un equipo que consiste en un scrum, un po y un equipo.

Los 3 artefactos son el sprint backlog, el producto backlog y el increment.

QUE PASA SI HAY CAMBIOS EN LAS FEATURES DEL PRODUCTO EN EL TIEMPO? El producto BACKLOG se va adaptando, se pueden y modificando, agregando, sacar, priorizar nuevamente. Hay una frase “los cambios son bienvenidos aun en etapas avanzadas del proyectos”.

El refinamiento había que realizarlo previo a la planning, redefinir los ítems, si hace falta reestimarlo y hasta agregarlos o eliminarlos. SERÍA UNA ADAPTACIÓN DEL BACKLOG A LOS CAMBIOS. ES LO QUE NECESITAMOS PARA QUE ESE BACKLOG SE MANTENGA VIVO Y RETROALIMENTADO DURANTE LAS EJECUCIONES DEL SPRINT. A diferencia de las otras, es **opcional**.

El PO es un rol dentro del equipo, es parte y tiene que tener la suficiente claridad y visión del producto y el suficiente involucramiento. Parte del acuerdo para que esto funcione es que durante de la ejecución del sprint.

Hay una fuerte diferencia entre ser ágil y hacer ágil.

Ser ágil significa implementar las prácticas del agilismo. No implementarlo en términos de culturas de la organización, sino es implementar las practicas.

Los beneficios se obtienen cuando nosotros podemos incorporar esta cultura de SER AGIL. Las practicas son la parte del iceberg que se ve. Pero debajo de todo eso, está la institucionalización de la cultura de agile, que es lo que hace estas prácticas se puedan materializar o cumplir, sin sentir que lo que estoy haciendo es en vano, tener una piedra en el zapato, sino que se naturaliza en todo el equipo de trabajo donde el PO es uno mas del scrum team. EL PO NO ES CLIENTE.

INSPECCION – ADAPTACIÓN:

EN LA DEMO, sobre qué hacemos la adaptación e inspección? Sobre el producto

EN LA RETRO, sobre qué hacemos la adaptación e inspección? Sobre el proceso

La idea de "escalar SCRUM" se refiere a llevar las reglas, reuniones, toda la metodología en sí, a la aplicación de muchos equipos para empresas más grandes o complejas. Esto se realiza porque vivimos en un mundo complejo, y la agilidad pretende reducir esa complejidad, y si usásemos solo el equipo de 7 personas aproximadamente, NO SERÍA SUFICIENTE.

CÓMO PONEMOS A PRUEBA TODO ESTO QUE PLANTEAMOS EN EL FRAMEWORK LLAMADO SCRUM?

En todas las técnicas o herramientas que uno utiliza, lo que hace o pone a prueba la efectividad de lo que estamos trabajando es que, de alguna manera, en este caso este framework, es que sea potencialmente escabable. Si no lo es, entonces la complejidad que conlleva construir software no lo va a resolver.

"vivimos en un mundo complejo, la agilidad pretende reducir esa complejidad"

QUÉ DIFICULTAD NOS PUEDE DAR TENER UN EQUIPO CON LA CARACTERISTICA DE 7+-2?

Lo que nos planteamos es si podemos construir todos los productos GRANDE que productos con un equipo de esa característica? La respuesta es que no. Entonces surge lo de que quiero ESCALAR. Quiero poder trabajar con lo que plantea este framework de scrum, los beneficios o las bondades que promete pero en un contexto de equipos mas grandes.

ES POSIBLE ESCALAR. CÓMO HACEMOS?

CYNEFIN ES UN FRAMEWORK – DAVE SNOWDEN: se plantea que de acuerdo a los entornos en lo que nos encontramos vamos a tener distintas formas de resolverlo o de encarar la complejidad o el escenario que el entorno plantea.

Arrancamos por el contexto de entornos **simples:** tengo una causa y una única consecuencia. "me duele la cabeza, tomo una aspirina" – implementando las mejores practicas, si sabemos que ante una causa siempre tenemos las mismas consecuencias, es muy fácil resolverlas.

Avanzando tenemos escenarios complicados en donde tenemos para una causa, varias consecuencias, no como en los escenarios simples. Tenemos más de un efecto, nos requiere usar las buenas practicas para abarcar a partir de una causa los posibles efectos que nosotros tenemos identificados para poder resolver.

Hasta acá ninguno de estos dos universos está relacionado con lo que plantea scrum para resolver la problemática de sw. LO QUE SE PLANTEA EN SCRUM PARA RESOLVER LA PROBLEMÁTICA DEL SW, es lo que entra dentro de este framework en el cuadrante de **COMPLEJO. En ese cuadrante de los escenarios complejos, no se puede asociar una causa y un efecto fácilmente, ni una causa a muchos efectos. Nos damos cuenta de la relación causa efecto después de que el efecto ocurra.** La clave en este tipo de escenario, y en esto que se llama ´practicas emergentes tiene que ver con lo que propone scrum, de inspección y adaptación. No tengo manera de resolver de antemano a partir de la causa cuales van a ser los efectos sino que a partir de lo que realmente va pasando, yo voy adaptando mi escenario a partir de estas practicas que emergen haciendo ese análisis de las relaciones causa-efecto para poder ir acomodándome en esos escenarios.

En el escenario CAOTICO, se intenta buscar practicas novedosas, ideas, para salir de ese escenario caotico, ósea, quiere decir que en verdad en escenarios caóticos no hay nada que se pueda resolver, uno en esos contextos no debería pasar demasiado tiempo y las practicas novedosas son para salir de ese caos que no puede manejarse. UN EJEMPLO: LA PANDEMIA. Nadie sabe como resolverlo, todos proponemos ideas para salir buscándole la vuelta de ese lugar.

DESORDEN: LO QUE ESTA EN EL MEDIO, DESORDEN ES FALTA DE INFORMACIÓN. NO tengo info para catalogar nignuno dee esos 4 escenarios, entonces directamente no puedo trabajar porque necesito saber en donde estoy parado.

Cuando trabajamos tanto con scrum como con los framework pares al scrum, estamos parados en los escenarios complejos, con la particularidad de que cuando hablamos de scrum, estamos hablando de escenarios complejos, y cuando hablamos de ESCALAR, seguimos estando en escenarios complejos pero un poco más complejos, porque se nos agregan variables que utilizando el framework como está planteado no están, no existen, no aparecen.

Cynefin: Complejo (prácticas emergentes, frente a escenarios complejos y su incertidumbre), Complicado (buenas prácticas), Caótico (prácticas novedosas) y Simple (mejores prácticas).

#Objetivo: Construir productos, no proyectos (lo que nos permite construir un producto) #La diferencia entre producto y proyecto, es que este ultimo muere tras cumplir su objetivo el producto prevalece.

NOSOTROS HACEMOS PRODUCTOS O HACEMOS PROYECTOS? QUÉ CONSTRUIMOS?

Construimos productos, el proyecto es una unidad de gestión que sirve para guiar la construcción de ese producto.

LO QUE NOSOTROS CONSTRUIMOS CUANDO HABLAMOS DE GENERAR SW ES QUE CONSTRUIMOS PRODUCTOS.

Necesitamos un framework para poder escalar lo que scrum plantea en términos mas sencillos. VAMOS A TRABAJAR CON NEXUS. Sin embargo hay 4 FW.

- **El primer FW QUE SALIÓ es SAFE**, dean leffingwel: es bastante complejo, es mas bien un fw organizacional, no está circunscripto solo a la escalabilidad de scrum, mezcla cosas. Algunos conceptos buenos son el de la **arquitectura**. Es el único FW que plantea en su esquema un **porfolio** de productos (bastante complicado de entender), todo el resto solo trabajan con un UNICO PB. Tiene una complejidad que va un poco en contra de como se supone que deberían ser los FW agiles, que deberían ser mas livianitos, dado que promueven el agilismo, priorización. Está mas orientado a organizaciones muy grandes, de gran escala.

Lo de arquitectura y porfolio es lo que lo destaca de los otros que no tratan estos temas, lo de porfolio es lo que lo hace disruptivo con respecto a lo que se plantea en el FW de scrum.

- **OTRO FW ES LESS, CRAIG LARMAN:**
  Plantea este FW que se llama less, que tiene 2 versiones:
  - Una que es la básica, que se banca hasta 8 equipo
  - Otra llamada less huge, que es ya para más de 8 equipos.

Es importante les, solo se eligió nexus porque la bibliografía esta en español.

SCALE Y NEXUS, las 2 personas aparecieron con el FW de scrum, no se pusieron de acuerdo y por eso crearon 2 por separado.

SCALE, también es un poco engorroso.

NEXUS ES MUY PARECIDO A SCRUM, es como que se le pusiera una capa de scrum arriba del fw scrum que ya conocemos. Todo es muy parecido. Simplemente plantea la integración entre lo que todos los equipos que trabajan y la minimización de dependencias entre todos lo que trabajan.

Nexus: Framework para escalar scrum, sencillo de entender a comparación de otros como Safe, LeSS o Scale. Además, cuenta con la documentación en español.

Este framework permite el escalamiento entre 3 y 9 equipos de scrum, si se necesitasen más, es posible que convenga utilizar Safe u otro.

Esta planteado con un UNICO PB. No está planteado para que haya más de uno. La ceremonia de refinamiento tiene que ser obligatoria a diferencia de scrum.

La lista de pendientes del sprint nexus es el sprint backlog de scrum.

El incremento del producto es un incremento del producto para todos los equipos que están trabajando. No tenemos un pedacito de incremento por cada equipo sino que el incremento de producto, una vez terminada la ejecución del sprint, ese artefacto es UNO SOLO. No tengo n incrementos del producto. Con lo cual la reveiw es la única ceremonia que es única, es decir, no hay una por equipo!

La planificación del sprint nexus, se hace una planificación del sprint, pensando en como se van a particionar la parte del producto backlog en cada uno de los equipos. Y cada uno de los equipos va a hacer su propia planning a partir de eso que le entregamos.

DAILY cada uno tiene la suya, la de scrum y después la nexus daily scrum que Se enfoca en el impacto de cada equipo sobre el incremento integrado y discuten:

- **El trabajo del día anterior fue integrado exitosamente, si no fue así, por que no?**
- **Cuales nuevas dependencias han sido identificadas?**
- **Que info necesita compartirse entre los equipos de nexus?**
-

RETRO también hay combinación entre lo que hace el equipo y luego lo que se hace en términos de nexus (integración)

¿Cómo funciona?:

-En Nexus el **product backlog** es uno solo (por ende, solo hay un PO y un solo producto). **Esta planteado con un UNICO PB. No está planteado para que haya más de uno.**

-**Planificación del Sprint Nexus**: Aquí habrá un **sprint backlog** por equipo (para que trabajen como lo harían normalmente en scrum).

#Es importante que no haya dependencias entre los sprint backlogs de diferentes equipos, pero que sus objetivos estén coordinados, ya que en algún momento habrá que integrarlo. (De esto se encarga el **Equipo de Integración Nexus**).

#Todos los integrantes de todos los equipos forman parte de esta ceremonia. -**Scrum Diario Nexus**: En Nexus hay 2 dailys, la del equipo (como en scrum) y la propia del framework que es ejecutada por el equipo de integración para ir subsanando las dependencias entre los equipos. Se enfoca en el impacto de cada equipo sobre el incremento integrado y se discuten temas como:

*¿El trabajo del día anterior fue integrado exitosamente?, ¿Por qué?

*¿Cuáles nuevas dependencias han sido identificadas?

*¿Qué información necesita compartirse entre los equipos del Nexus?

Primero se hace la de cada equipo, la daily scrum. Participa el NIT, por eso también es importante que los que conformen el nit sean los mismos que conforman los otrs equipos.

-**Sprint Review Nexus**: Es uno solo, al igual que en scrum (solo que tendrá mayor volumen).

-**Sprint Retrospective Nexus**: Consta de tres partes:

1) Representantes de todo un Nexus identifiquen problemas que hayan impactado en más de 1 equipo (Transparentar problemas compartidos) que también tiene que ver con la integración y dependencia. Tengamos en mente los problemas que se están dando en mas de un equipo.
2) Cada equipo Scrum realice su propia retrospectiva y aborda lo siguiente:
    a. ¿Se dejó algún trabajo sin realizar? ¿El Nexus generó deuda técnica? La deuda técnica es la atada de alambre, no es defecto. El producto funciona, hay cosas que tengo que emprolijar. No es particular de nexus.

b. ¿Todos los artefactos se integraron con éxito, al menos 1 vez al día?
c. ¿El software se desplegó con la frecuencia suficiente para prevenir la acumulación abrumadora de dependencias sin resolver?

3) Representantes apropiados de Equipos Scrum se reúnan y acuerden cómo visualizar y rastrear las acciones identificadas.

Roles en Nexus:

-**Nexus Integration Team (NIT)**: (no existe en scrum) Se compone por el PO + SM + 1 (o varios) NIT (miembros del equipo) Members, pudiendo ser este parte del NIT, pero debiendo priorizar su trabajo como NIT Member. **Por que suele ser así de que se incluyan miembros de los scrum individuales? Porque eso hace que la composición del NIT vaya cambiando en el tiempo en función de las necesidades. Por ejemplo si yo necesito en algún momento revisar algo sobre la integración e independencia de la arquitectura, en ese caso me conviene más un NIT que tenga más experiencia en arquitectura.**

Responsabilidades:

* Declaración del DoD (Definition of Done), aplicable al incremento integrado. Porque el incremento es el artefacto, uno solo, tiene que estar abordado por todos los equipos.
* Coaching sobre estándares de desarrollo, arquitectura e infraestructura.
* Consultoría
* Generación de advertencias sobre dependencias y problemas entre equipos
*Puede trabajar en el producto backlog #La

composición del NIT puede cambiar en el tiempo

-PO en NIT: Mismo que en Scrum.

-Scrum Master en NIT: Mismo que en Scrum.

-NIT Members: Mismo que en Scrum.

Tiene como función resolver los aspectos que tienen que ver con la integración y con las dependencias que surgen producto de que muchos equipos hasta 9 estan trabajando sobre un mismo producto backlog para obtener un único incremento del producto.

Justamente lo que plantea este FW es incluir este nuevo sol para poder resolver estos 2 inconvenientes que surgen de trabajar con mas de

un equipo que tiene que ver con la integración (porque el artefacto es un incremento del producto, no partecitas del incremento) para que esto funcione así necesitamos resolver la integración. Sería inviable platear un FW de estas características si no tengo incorportado al menos la integración continua.

¿Cual es la traba que me presentan las dependencias? Que haya ciertas dependencias que condicione el avance de un equipo a lo que otro equipo puede estar haciendo. ELIMINAR LA DEPENDENCIA ES IMPOSIBLE, pero al menos hay que minimizarla.

Artefactos:

-Product Backlog

-Nexus Sprint Backlog repartija de qué le va a tocar a cada equipo resolver en el sprint.

-Incremento Integrado **El incremento del producto es un incremento del producto para todos los equipos que están trabajando. No tenemos un pedacito de incremento por cada equipo sino que el incremento de producto, una vez terminada la ejecución del sprint, ese artefacto es UNO SOLO. No tengo n incrementos del producto. Con lo cual la reveiw es la única ceremonia que es única, es decir, no hay una por equipo!**

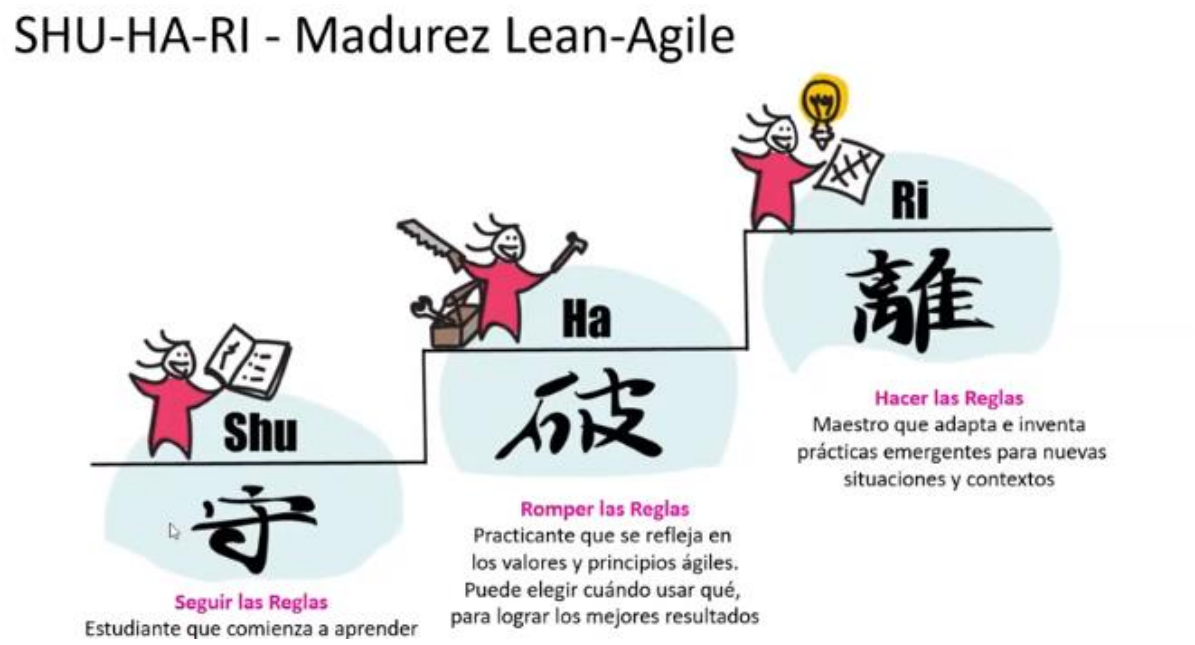
Acá la planning va al revés, primero se hace la planning de partición para tener el sprint backlog de nexus, donde particiona para cada equipo y luego cada equipo hace su sprint planning.

La daily primero la de cada uno de los equipos y después la de nexus.

En la retro es un mix, por un lado hacen los de nexus, por otro la de los equipos y después se integran ambas.

#No es lo mismo aprender como funciona scrum, e implementar las buenas practicas de scrum, que tener institucionalizadas las maneras de pensar y los valores relacionados a dichas prácticas.

#Es diferente hacer agile, que ser agile.



La ruta japonesa es la maestría

SHU HA RI MADUREZ LEAN AGILE

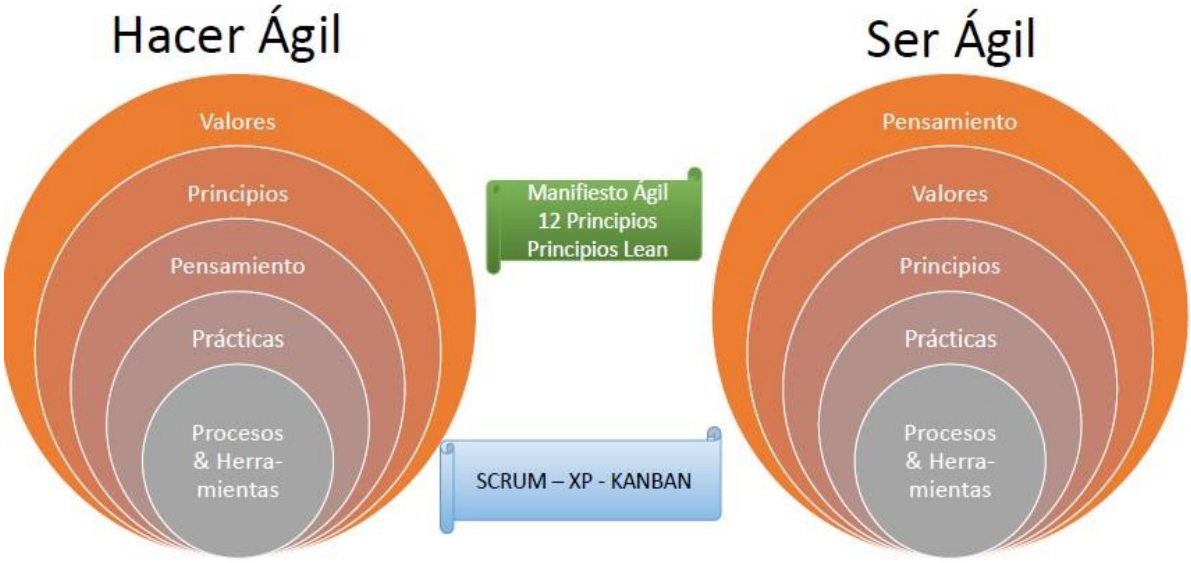
Cual debería ser el camino que nosotros deberíamos seguir para ser capaces a partir de nuestro aprendizaje de definir las reflas, nuevas reglas. Antes que nada de romper, aprende bien lo que esta definido, toma la experiencia que implica seguir las reglas, tomalo, dominalo, entendelo, usalo.

Después de ahí recién, Romper las reglas, significa que puedo elegir qué usar y que no, que usar en un contexto si y en otro no.

Lo primero que hacen las organizaciones en ver las reglas y adaptarlas de una sin saberlas.

Y al último recién plantear practicas emergentes.

PARA QUE TODO FUNCIONE tenemos que hacer un cambio en la cultura para poder implementar las practicas! Para que lo que plantea scrum funcione, para que nos de los beneficios realmente de lo que queremos obtener, la base de la pirámide (valores, creencias de la cultura) es cambiar la cultura para que cada uno de los individuos que conforman nuestra organización tengan estos valores, es el primer paso. Sino no estamos aplicando ni nexus ni scrum. Y de ahí parte la diferencia de ser ágil y hacer ágil



Hacer ágil, nos paramos en el contexto simplemente de implementar métodos y practicas agiles, estamos haciendo algo para lograr la agilidad, nos sirve para tener los artefactos, para tener definidas las practicas PERO NO TENEMOS LOS VALORES INCORPORADOS, NI EL PENSAMIENTO QUE TIENE QUE VER CON SER AGIL A NIVEL ORGANIZACIONAL. ESO ES UN CAMBIO CULTURAL, CAMBIA HASTA EL ESTILO DEL LIDERAZGO Y ES LO QUE SE PROMUEVE EN TODO EL MTERIAL RELACIONADO CON LOS VALORES, PRINCIPIOS AGILES

Filosofía Lean

Los principios Lean:

Surge en el contexto de la producción, y a partir de acá surgen algunos conceptos que después se extrapolan en agile, como por ejemplo, el just in time, que después lo vemos reflejado en las entregas frecuentes. Cuando hablamos de lean no hablamos de proyectos. Hablamos de un trabajo continuo que se mantiene durante todo el tiempo.

Tiene 7 principios.

-Eliminar Desperdicios (Producir de más, defectos, esperas, etc.) evitar que las cosas se pongan viejas antes de terminarlas o evitar retrabajo, Se relaciona con **Raleases Frecuentes (de 2 a 4 semanas):** División de trabajo en fases productivas. (y también tiene que ver con el principio ágil de software funcionando y el de simplicidad

Se relaciona con el just in time y tiene que ver con tener la menor cantidad posible de stock que genera desperdicio porque tiene un costo de mantenimiento. Tenerlo cuando lo necesito! Reducir el tiempo removiendo lo que no agrega valor.

Donde mas vemos en termino de desperdicios es en la parte de requerimientos, en scrum de hecho se trbaaja, no me pongo a identificar y especificar todos los requerimientos cuando quizás los voy a implementar y esos requerimientos ya no están vigentes. Perdi tiempo al pedo. Perdió valor. Lleva mucho trabajo los requerimientos. Hay que lograr que lo que construyamos en termino de sw tenga valor para el cliente o que no se usa.

Desperdicio es cualquier cosa que interfiere con darle al cliente lo que el valora en tiempo y lugar donde le provea mas valor.

En manufactura, el inventario

En sotware: es el trabajo parcialmente hecho y las características extra

El 20% del sw que entregamos contiene el 80% del valor.

(arte de maximizar lo que no hacemos, tiene que ver con el software funcionando es lo que da valor y tratar de optimizar las tareas que nosotros hacemos para evitar retrabajo o evitar construir cosas que después no se usan.

Los siete desperdicios Lean (en software):

- Producción en exceso porque me implica tener disponible al menos estructura de logística y almacenamiento para decidir que hago con esa producción en exceso

- Stock me genera desperdicio porque me genera un gasto en logística y en infraestructura innecesario.

-Pasos extra en el proceso

-Búsqueda de información

-Defectos

-Esperas

- Transportes

-Ampliar Aprendizaje

-Embeber la Integridad Conceptual

-Diferir compromisos

-

Dar poder al equipo

-Ver el todo

-Respetar a la gente

Los 4 valores de los principios Ágile:

-Individuos e interacciones por sobre procesos y herramientas

-Software funcionando por sobre documentación detallada

-Colaboración por sobre negociación con el cliente

-Responder a cambios por sobre seguir un plan

12 principios del Manifiesto Agile:

-**Satisfacer al cliente con entregas frecuentes y tempranas**: Darle pequeñas entregas para contentar al cliente e ir teniendo un feedback de parte del cliente. #Productos de valor que cubran una necesidad.

-**Cambios de Requerimientos Son Bienvenidos**: Aceptar los cambios provenientes del feedback del cliente.

#Cambiar sobre la marcha no es dar un paso atrás.

-**Raleases Frecuentes (de 2 a 4 semanas):** División de trabajo en fases productivas.

-**Técnicos y no técnicos juntos**: Los líderes de los proyectos deben ejercer su labor en el mismo terreno donde tienen lugar las tareas y no desde los despachos.

-**Individuos Motivados**: Los procesos solo tendrán éxito si quienes los llevan a cabo son personas motivadas y que están en un clima de confianza.

-**Medio Comunicación: cara a cara**: El gestor responsable debe comunicar de forma eficaz sus mensajes (óptimamente de manera presencial, tanto para el cliente como sus colaboradores).

-**Métrica de progreso: Software funcionando**.
#Medir con indicadores concretos la evolución de los procesos.

-**Ritmo de desarrollo sostenible**: La forma de ejecutar los proyectos debe garantizar en si misma su continuidad.

-**Atención continua a la excelencia técnica**: Las formas nunca deben perderse, así como tampoco la calidad del trabajo.

Simplicidad: Maximización del trabajo no hecho.
#Las tareas han de ser lo más sencillas posibles (dividir tareas grandes en pequeñas).

-**Arquitecturas, diseños y requerimientos emergentes**: Los proyectos no suelen terminar de la misma forma que empezaron, resulta indispensables que quienes los ejecutan se adapten ante las circunstancias.

-

-**A intervalos regulares el equipo evalúa su desempeño**: Los equipos deben ser capaces de organizarse por sí mismos (aun así, debe existir una figura que los monitorice).

Significado de Ágil: Balance entre ningún proceso y demasiado proceso. La diferencia inmediata es la exigencia de una menor cantidad de documentación, pero lo más importante es:

-La adaptabilidad de los métodos en lugar de predictivos.
-La orientación a la gente en lugar de hacia los procesos.

#Triangulo agile, valor, calidad, alcance (costo y tiempo)

Kanban

Palabra japonesa creada para líneas de producción en un inicio, significa tarjeta señalizada. El punto es el trabajo que se debe hacer en esa tarjeta y lo que se debe ver en ella.

Kanban: Es un método para introducir cambios en un proceso de desarrollo de software o una metodología de administración de proyectos.

#No se utiliza para construir software, se usa para el mantenimiento.

Just in Time: Que no se genere desperdicio por tener más oferta que demanda, eliminando el sobrecosto.

Básicamente consiste en la cadena de trabajo, pero una flexible, cuando alguien este desocupado de una tarea, colabore con otra. Aquí el flujo esta centrado en el cliente

Principios:

-Visualizar el Flujo: Hacer el trabajo visible.

-Limitar el Trabajo en Progreso (WIP).
-Administrar el flujo: Ayudar a que el trabajo fluya.

-Hacer explicitas las políticas. -
Mejorar colaborativamente.

Kanban aprovecha muchos de los conceptos probados de Lean:

-Definiendo el Valor desde la perspectiva del Cliente.

-Limitando el Trabajo en Progreso (WIP).

-Identificando y Eliminando el Desperdicio.

-Identificando y removiendo las barreras en el Flujo.

-Cultura de Mejora Continua.

Kanban fomenta la evolución gradual de los procesos existentes.

-

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

-Kanban está basado en una idea muy simple:

*Limitar el trabajo en progreso (WIP).

-El Kanban (o tarjeta de señal) 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 límite acordado.

¿Cómo aplicar Kanban?

-Empezar con lo que se tiene ahora.

-Entender el proceso actual.

-Acordar los límites de WIP para cada etapa del proceso.

-A continuación, comienza a fluir el trabajo a través del sistema tirando de él, en presencia de señales Kanban.

#Dividir el trabajo en piezas, y las US son buenas para eso.

#Utilizar nombres en las columnas para ilustrar donde está cada ítem en el flujo de trabajo.

#Distribuir el trabajo en las columnas: el trabajo fluirá de izquierda a derecha en las columnas.

#Pull, no push.

#Limitar la cantidad de tareas simultaneas. #Ayudar a que el trabajo fluya.

¿Cómo aplicar Kanban en nuestro proyecto?

-**Proceso**: modelar nuestro proceso.

-**Trabajo**: decidir la unidad de trabajo-

-**Límites de WIP**: limitar el WIP para ayudar al flujo de trabajo.

-**Política**: definir políticas de calidad

-**Cuellos de Botella y Flujo**: mover recursos a los cuellos de botella.

-**Clase de Servicio**: diferentes trabajos tienen diferentes políticas definición de hecho (“done”), para cada estado.

-**Cadencia**: Releases, planificaciones, revisiones.

Métricas Clave:

-Lead Time = Vista del Cliente

-

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

Cycle Time = Vista Interna

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

#Medición más mecánica de la capacidad del proceso.
#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 bloqueado o sin trabajo del equipo sobre el mismo.

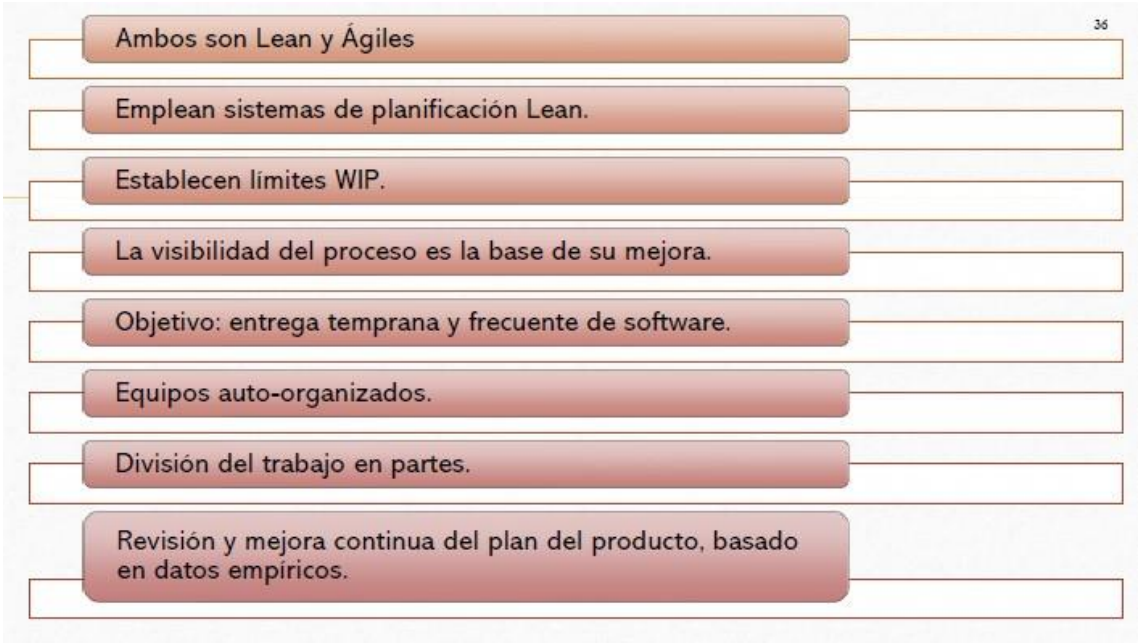
$$Touch\ Time \leq Cycle\ Time \leq Lead\ Time$$

-Eficiencia del Ciclo de Proceso:

% Eficiencia ciclo proceso = Touch Time / Elapsed Time

#En términos de procesos, scrum es más prescriptivo que Kanban, pero este es más adaptativo.

Similitudes de Scrum y Kanban:



-

Diferencias de Scrum y Kanban:

-

Scrum	Kanban
Iteraciones de tiempo fijo.	**Tiempo fijo es opcional**. La cadencia puede variar. Pueden estar marcadas por la previsión de los eventos en lugar de tener un tiempo prefijado.
Equipo asume un compromiso de trabajo por iteración.	El **compromiso es opcional**.
Métrica para planificación y mejora: **Velocidad.**	Métrica por defecto es **Lead Time** (Tiempo de Entrega o tiempo medio)
Equipos Multifuncionales.	**Equipos Multifuncionales o especializados.**
Funcionalidad divididas para poder completarse en un Sprint.	No hay prescripción respecto del tamaño de la funcionalidad.
Deben emplearse gráficos **Burndown chart**.	No se prescriben diagramas de seguimiento.

Scrum	Kanban
Limitación **WIP indirecta** (por Sprint).	**Limitación WIP directa** (marcada por el estado del trabajo)
Se deben realizar estimaciones.	**Las estimaciones son opcionales**.
No se puede agregar alcance en medio de una iteración.	Siempre que haya capacidad disponible **se puede agregar trabajo.**
Sprint Backlog pertenece a un equipo determinado.	Varios equipos pueden compartir pizarra Kanban.
Se prescriben tres roles (PO / SM/ Equipo).	No hay roles prescritos.
En cada sprint **se limpia el tablero** de seguimiento.	El tablero Kanban **es persistente**.
Product Backlog Priorizado.	**La priorización es opcional.**

Métricas de Kanban

Promedio de Cycle Time es alto: tenemos un **Product Owner** que no hace bien su trabajo, una complejidad elevada **o un proceso excesivamente burocrático** para la aprobación de un PBI.
Promedio del Lead Time sube: tenemos **sobrecarga de trabajo**. Hay algún cuello de botella, el equipo es insuficiente, el ritmo no es adecuado, etc.

Promedio del Lead Time baja: no tenemos suficiente trabajo porque no hay pedidos y el equipo está sobredimensionado, o si somos positivos, el equipo es muy bueno y está trabajando a un ritmo muy bueno.

Promedio de Touch Time por historia alto: tenemos muchos impedimentos que requieren una mayor atención por parte del Scrum Master.

Variabilidad del Lead Time: complejidad muy distinta entre unos issues y otros.

-

Indudablemente del análisis de estas tendencias, tenemos que sacar acciones de mejora que variarán en función de cada proyecto /equipo. **¿Qué monitorear?**

En mi experiencia, es importante revisar periódicamente o durante retrospectivas los números a continuación. En la tabla también explico posibles interpretaciones de las tendencias y algunas pistas para mejorar:

-Tendencia
-Posibles Interpretaciones
-Posibles Acciones

Promedio del Cycle Time Está bajando

- El equipo está mejorando **-** Proceso más fluido **-** ¡Seguir así!

Promedio del Cycle Time Está subiendo

-Se complejiza el trabajo

-Hay bloqueos y/o cuellos de botella

-Se desmotiva el equipo

-Analizar cuellos de botella y bloqueos de trabajo

-Revisar motivación del equipo y/o capacidad del equipo para el trabajo

Promedio del Lead Time Está bajando

-No hay pedidos

-El equipo está sobredimensionado

-Mejoramos

-Analizar si no deberían llegar más pedidos

-Analizar dimensionamiento del equipo

Promedio del Lead Time Está subiendo

-Hay muchos pedidos

-El equipo está sub-dimensionado

-El ritmo de trabajo no es suficiente

-Revisar validez de los pedidos

-Analizar dimensionamiento del equipo

-Revisar motivación del equipo y/o capacidad del equipo para el trabajo

Variabilidad Cycle Time Muy variable

-

-Muchas diferencias de complejidad entre los ítems

-Deuda técnica acumulada surge en cualquier ítem en forma aleatoria

-Separar ítems en varias clases de servicios o según complejidad (alta, media, baja)

-Atacar la deuda técnica e implementar prácticas para bajarlas (XP)

Promedio Touch Time / Promedio Cycle Time Baja proporción

-Mucha espera en el proceso

-Muchos bloqueos en el proceso y cuellos de botella

-Mucho mult-tasking- Bajar los límites de WIP y trabajar sobre puntos de bloqueos y cuellos de botella Identificar puntos de espera externa y trabajar sobre su optimización

Promedio Cycle Time / Promedio Lead Time Baja proporción

-El equipo está sub-dimensionado

-El ritmo de trabajo no es suficiente

-Analizar dimensionamiento del equipo

-Revisar motivación del equipo y/o capacidad del equipo para el trabajo