**Scrum – Clase 28/09/2023**

Scrum framework agil para la gestión de proyectos

Sirve como es mas eficiente, esta pensado para desarrollo de productos donde hay ambientes de incertidumbre y cierta complejidad que la incertidumbre te da. Mas incertidumbre es mas complejo.

Scrum practicas. Trabajar en sprint. Un spritn es una iteración de duración fija. Máximo 4 semanas o 30 dias.

Roles po, scrum master, developers. Developers son los programadores, pero es una generalización. Developers no necesariamente son los programadores, son personas que crean producto o artefactos, no importa el trabajo que hacen, pueden ser arquitectos, médicos.

Soluciones adaptativas para problemas complejos. Framework o marco de trabajo que ayuda a personas o grupos a gestionar. Scrum es un framework para gestión porque básicamente no apunta a ninguna actividad de ingeniería, no habla de software. Se quiere extender a cualquier ámbito de trabajo.

Ceremonias, reuniones eventos. 4 actividades mas una que es continua. Esta ultima se utiliza en demanda, cuando se necesita. Es el refinamiento del producto backlog. Antes se llamaba grooming. Es continua, todo el tiempo estamos haciendo esto, on demand, vamos a armar una reunión de refinamiento cada vez que lo necesitemos.

Time box. Todas las actividades tiene que tener duración fija. El time box dice que la duración del sprint es fija y la acuerda el equipo pero tiene un máximo de 30 dias, de preferencia menos. Todas las ceremonias tienen time box. La daily 30 minutos y todos tienen que hablar y asistir. Hay time box en todas. Esta la recomendación de la duración de cada una de las ceremonias.

El tiempo es complejo de administrar porque no lo pueden controlar, entonces hay que gestionarse para aprovechar mejor el tiempo. Es difícil porque hay mucho distractor, mucha fuente que interrumpe la concentración, mucha necesidad de inmediatez. Entonces un poco la idea de ayudar a organizarse forzadamente con el tiempo es el time box.

No discutir un sprint entero las estimaciones de las user. El concepto de time box nos obliga a ser respetuosos con nuestro tiempo y el de los demás. El agilismo no es legalizar el quilombo.

Si tenemos la cultura arraigada la practica es mas fácil de sostener. Las mejoras de proceso hay que sostenerlas hasta que se hace el cambio cultural. Una de las restricciones de scrum es el time box. Se termino el tiempo y no llegamos con todo, entregamos menos. Restringe la manera que vamos a trabajar. A partir de ese time box uno va a incorporar lo que dicen las ceremonias.

Iterativo incremental de duración fija y ahí es donde engancha el time box

Sprint planning, la planificación del sprint. A que nos comprometemos. Asisten todos al sprint planning. El producto owner es parte del equipo. Principio ágil técnicos y no técnicos trabajando juntos. En la daily no participa el producto owner porque si el no tiene asignaciones de trabajo completas dentro del sprint, no tiene la obligación de asistir pero si puede. De mas no es el problema, de mas si. El po esta en la planning, en el refinamiento del backlog porque es su ceremonia, porque es el dueño del producto backlog. Puede que no todos los developers estén en la ceremonia de refinamiento, pero el po si o si, sino no hay refinamiento.

El fin ultimo del sprint planning es la creación del artefacto sprint backlog

Artefactos: producto backlog, sprint backlog y el incremento del producto

El soft es abstracto, y antes de que existieran las descargas, el soft se vendía en cajas, donde venían los manuales y los cds/dvds/disquetes/blueray. Algunas cosas se vendían en el supermercado. Por eso la representación del incremento de producto es una caja

Es importante que este el po porque ahí le mostramos el producto. En la sprint review. Además, el po puede invitar a otras personas, usuarios, jefes. De mínima el equipo completo, puede haber más.

La retrospectiva vemos que hicimos bien que hicimos mal, con el propósito de mejoras sobre el proceso. En la review se mejor producto, en la retrospectiva el proceso. Usamos los pilares del empirismo: transparencia, adaptación, introspección.

En la daily hay inspección y adaptación. El propósito de la daily todos hablan y lo que dicen es concreto, que hiciste en que estas trabajando que pensar hacer. El equipo tiene libertad de hablar o preguntar de otra manera. Para que no se resista el equipo a la daily. La necesidad sigue existiendo, no sacarlas, sino aprender a hacerlas bien, y en ese contexto la daily va a servir.

En la review se calcula la velocidad del equipo. La palabra velocidad tiene que ver con desplazarse rápido. Y no es eso en scrum lo que quiere representar. La velocidad mide producto, la unidad de medida del producto son los story point. La mejor métrica de progreso es el soft funcionando. Mide producto que fue aceptado por el producto owner. Definición: punto de historia que el producto owner acepto al final de una iteración. Me acepto la user o no, no hay trabajo a medias. Si no la acepto vuelve al producto backlog. En la review hacemos ese calculo de velocidad. En los primeros sprint la velocidad puede ser 0. Y eso puede pasar si metemos una cascada en el sprint.

La retrospective es el equipo. Hay una recomendación de que venga un moderador de afuera y el scrum master puede participar. Si el sm esta moderando no puede participar. Acá puede pasar que el producto owner no este si tuvo una actitud distante o el equipo no lo invita, pero lo ideal es que este. Esta relacionado con el ultimo principio que dice que a intervalos regulares el equipo se reúne y reflexiona sobre lo que puede mejorar, del PROCESO. Que hicimos bien, que hicimos mal, que podemos hacer para que funcione mejor. La retrospectiva es una ceremonia especifica que alguien tiene que moderar, y eso no significa que el equipo pierda su capacidad de autoorganización.

El ciclo termina con improvments, mejoras. Al iniciar la siguiente planificación del siguiente sprint necesitamos considerar los resultados de la retrospective. Cualquier mejora de la retrospective uno la tiene en cuenta al momento de iniciar el siguiente sprint. Seteemos un conjunto de condiciones al inicio del sprint y no cambiemos nada. No se aceptan cambios durante el sprint. Si el cambio no afecta el cumplimiento del objetivo del sprint puedeo considerarlo pero solo si esta alineado con los objetivos del sprint, sino no se acepta.

El tiempo del sprint no se puede cambiar durante el sprint, porque ahí rompes el time box, si te puede pasar que canceles un sprint, si un incremento ya no lo necesitas mas no lo seguis. La decisión de cancelar el sprint es del producto owner. Si el objetivo ya no es mas relevante. También se puede cancelar por cuestiones internas del equipo, que haya cosas que no funcionen bien, o que se haya estimado mal. Puede pasar que el equipo no este a la altura y no lleguen con algo comprometido, entonces se cancela para no perder tiempo y planificar de nuevo. Se intenta que no pase porque uno de los principios es que se asumen equipos capacitados y motivados. Con eso me va a ir bien con cualquier framework porque la cuestión esta en el equipo, no en Scrum. Si no esta capacitado el equipo no es culpa de Scrum.

El compromiso del prodcut backlog. Es importante tener un norte. Porque nos ajusta a que las decisiones que tomamos estén alineadas con el objetivo. Si lo formalizo, los ítems que meta en el producto backlog están alineadas con el objetivo, y de paso pelear contra el yagni

En el caso del sprint backlog ahora es obligatorio definirlo (el objetivo). Los cambios en el sprint los aceptamos solo cuando no comprometen los objetivos del sprint. Se hace necesario tener claro los objetivos del sprint para ver si se aceptan cambios o si afecta al objetivo del sprint. Si el objetivo del sprint es desarrollar la capacidad del producto de cobrar y en la planning decidís que vas a empezar con la user de cobrar con tarjeta de crédito, si no la empezaron y el po dice que prefiere cobrar con billetera se puede aceptar. Hay que ver los puntos de historia para ver si podes absorber ese cambio o no.

El incremento de producto es el DoD, lo que tiene que tener de madurez el producto para poder decir que esta terminado y presentarlo al producto owner. El compromiso es pegar el DoD. Puede pasar que la característica cumple el DoD pero el producto owner no lo acepta, porque no es lo que pensaba o cambio de opinión.

El DoR es para ver si una user esta en condiciones de ingresar al sprint backlog. El DoD se usa para hacer la entrega del producto. Incremento del producto potencialmente entregable. Potencialmete porque depende si el po lo quiere poner en producción o no, pero si quiere el producto tiene que estar en condiciones de, si quiere o si quiere hacer una reléase del producto una vez al año. En producción es software que esta para usarse. En scm hay tres entornos: de desarrollo, testing, (idealmente de pre-producción), y de producción. La review se hace en el de pre producción si lo tenes, sino en el de producción. El pre suele no existir por temas económicos, que tenga exactamente las mismas condiciones de hardware que el de producción.

Calculo de capacidad de un equipo. Nos hace falta en la planning. La capacidad es la otra métrica que se usa fuertemente en scrum. La velocidad es la otra. La velocidad no se estima, se calcula sumando los story point de las user que el po nos acepta. La capacidad es la otra que se usa en la planning y si se estima. Se mide de dos maneras, o se mide en puntos de historia para los equipos mas experimentados (podemos hacer 15 puntos de historia por ej), la otra para equipos no tan experimentados es medir la capacidad en horas ideales (el tiempo que yo puedo estar concentrado efectivamente haciendo cosas vinculadas a las users que tengo que completar). El equipo al inicio de la planning hace el calculo de capacidad. Con una recomendación de trabajar por rangos. Yo creo que puedo trabajar entre 4-7 horas por día y voy a estar disponible 8 dias, entonces puedo aportar 32 horas de trabajo (rango inferior 8\*4) o 56 (rango superior 8\*7). Y cada persona del equipo determina eso. Cada uno tiene que hacer el calculo de cuantas horas puede aportar y ahí se arma el calculo de capacidad. Esta forma de trabajo asume developers que pueden tomar una user del sprint backlog, empezarla y terminarla. Scrum define condiciones de seteo. Asume equipos multifuncionales que todos pueden hacer todo. En ese contexto un calculo de capacidad así sirve como estimación. Las estimaciones se realizan no importa el enfoque. Y hace falta reestimar, si cambia una condición y necesito reajustar o si me equivoque.

La herramienta fundamental que elige scrum para visualizar el trabajo es el tablero o board. La característica de estos tableros es que duran lo que dura el sprint. Se configuran inicialmente en el sprint planning y luego al final se borran para finalizar el sprint para el siguiente. Tiene 3 columnas el To do, del producto backlog aparecen las users estimadas en story points y ese es el to do, las cosas que tengo que hacer. Después el doing, ahí están las asignaciones de trabajo, yo voy a buscar a la columna de to do lo que me comprometo a hacer, paso la user de la columna de to do al doing y le pongo mis iniciales. La hacer el propio developer, es un compromiso que asume.

Entra al to cuando cumple el DoR, se puede mover del doing al done cuando cumple el DoD. En la daily se actualiza el tablero. En el to do tenes todo lo que tenes que hacer en el sprint. En el doing tene una sola cosa a tu nombre, y cuando la terminas vas a otra, no hacer multitasking.

Esa es la configuración básica. Solo se puede hacer si tenes developers en condiciones de agarrar una user, empezarla y terminarla. Sino usas configuraciones alternativas. Story del backlog para acá, en la primera parte de la planning. El equipo descompone en tareas y aparece el to do, cada user tiene varias tareas. Separa todas las tareas que tenes que hacer en esa user para cumplir con el DoD y las estimas en horas ideales. De ahí ya se puden asignar a personas, las tareas. Pasa a in process. Después a in verify para ver si ya podes juntar las tareas y llevarlas al done. Done Done, mueve la user de story al final cuando la user esta terminada. No hay una sola configuración del tablero, depende de lo que el equipo considere. La granularidad del tablero tiene que ser fina. Si es grande no tenes margen de maniobra. Users grandes hay riesgos de no terminarla en el sprint y además no tenes margen de monitoreo en el tablero.

La descomposición en tareas de la user se hace en la planning. Un primer momento en que esta el po que es el del que y otro que ya no se requiere que este que es el como.

Trabajo quemado, cuanto trabajo hemos terminado. Son gráficos que se blamnquean al final del sprint y se actualizan en la daily. Puntos de historia en eje de las y, y días del sprint en el eje de las x. No horas, porque si gestionas por hora estas gestionando el trabajo, y acá lo importante es el producto. Puedo consumir todas las horas y no necesariamente lograr todo. Se limpia al final del sprint. Me comprometo a una tal cantidad de puntos de historia, y lo ideal seria que queden 0 al final. Sirve para ver si voy a poder cumplir un sprint o si lo voy a tener que cancelar.

En una organización hay distintos niveles de planificación. El scrum lcubre el nivel de día con la dayly, y la iteración con el sprint planning. Por arriba de eso no. No vamos a poder entregar un producto en un sprint. Aparece la planificación del reléase. Una versión del producto con cierta cantidad de características que se libera al mercado. Después esta el producto. Si la organización tiene más de un producto empiezan a competir los productos entre si, hay un portafolio, y hay planificación a nivel de portafolio. El producto backlog es producto, no tareas, pero producto. Las tareas pueden aparecer en el sprint si decido descomponer las users en tareas. Si lo hago la estimación es en horas ideales.

El artefacto que sale de la planificación del reléase es el reléase planning. Cuantos sprint vamos a ejecura en este reléase y mas o menos un planteo deol objetivo y alcance de cada sprint, y la duración para estiamr la duración del sprint. Tengo la decisión de trabajar este backlog, en el spint 1 va esto, el 2 esto. Voy a tener sprint de duración de un mes cada uno y estimo como capacidad de tal cantidad de puntos de historia por sprint. Si tengo tal capacidad no me voy a comprometer a mas.

Es un plan que se hace a la realidad de hoy, mañana cmabia la realidad y tengo que cambiar el plan. En ágil si se planifica, y a veces mas frecuentemente que en el ortro enfoque. Solo que la importancia del artefacto es menor, porque lo importante es el acto de planificar, no la planificación en si. Entonces para cada reléase cuantos incrementos de producto nos comprometemos a entregar. Hay veces que el primer reléase puede tener como objetivo desarrollar el mvp.

Cadencia de los sprints, cada cuanto tengo un reléase. Es una decisión de negocio.

Release plan, cuantos sprint necesito para cumplir con el reléase. Vamos a hacer consideraciones. Cuantos sprints de cuanto tiempo de duración y mas o menos plantear las users que iriran en el reléase, que te gustaría que este reléase haga y ahí se decide que historias entran, teniendo en cuenta la capacidad y la velocidad.

La salida del sprint planning es el sprint backlog. Cuanto dura el sprint, cual es el objetivo, quien es el equipo, que capacidad tiene estimadas en horas ideales. Hay que matchear la suma de las horas de las tareas con las estimadas de la capacidad. No necesariamente una user de 2 puntos va a tener la misma cantidad de horas

**Testing de Software – Clase 05/10/2023**

Aseguramiento de calidad de proceso y producto

La palabra aseguramiento apunta a un conjunto de acciones que podemos hacer para evitar los errores y evitar que esos errores se conviertan en defectos.

Asegura proceso y producto. No alcanza el tiempo para ver los dos así que vamos a hacer foco en producto. Para hacer aseg de calidad del producto, una practica que ha trascendido en todas las metodologías, son las revisiones de a pares o revisiones técnicas. Lo importante es que no las hace el jefe, el dueño o el cliente. Lo hace alguien que tecnincamente esta igual de preparado que yo o mejor, y que no tenga un vinculo de relación jerárquica y que no sea un auditor tampoco. Las revisiones técnicas, pares o de colegas son la técnica por excelencia para hacer este trabajo de prevención. Hay varios métodos algunos mas formales otros mas formales.

Las revisiones técnicas se hacen al código, al diseño, la documentación, los reqs, los planes. Conclusión a todo. Cualquier artefacto que alguien construye, alguien lo puede revisar. Es una actividad humano intensiva y los humanos los pueden cometer.

Hay muchos frameworks agiles que hablan de programación de a pares. Si haces xp tenes que hacer programación de a pares. Uno programa y el otro corrige. Tengo dos programadores por puesto de trabajo. Es la forma mas inmediata de revisión técnica.

Las revisiones técnicas las trajo el agilismo. La otra técnica son las auditorias y esas al agilismo no le gusta porque tiene que venir alguien de afuera.

Testing ágil es una concepción diferente. Pasa a tener mucha importancia y mucho del trabajo de reqs se mueve al testing. Las pruebas no pueden quedar desactualizadas porque sino no podes probar. Los reqs pueden quedar desactualizados y en principio “no pasa nada”. Movamos el esfuerzo del detalle de los reqs al testing que es algo que necesariamente tenes que tener actualizado. Es un enfoque conceptual distinto.

Hay cosas que en un enfoque son iguales que otros. Otra de las herramientas para hacer aseguramiento son las auditorias

Como encaja el testing cuando puedo hacer la ejecución del testing. Es cuando el producto ya esta construido. No es aseguramiento, es parte de la disciplina, pero es control de calidad lo que hace. Una actividad de validación y verificación. Esos son los dos procesos principales. Verifica que el sistema funciona correctamente y valida que el sistema sea el sistema correcto. Hago eso con un producto construido, lo que detecto con el testing son defectos, que se trasladaron. Es control de calidad.

Tiene que haber una buena scm para el aseguramiento de calidad

Hacer testing. Proceso destructivo de tratar de encontrar defectos cuya presencia se asume. El testing lo único que puede hacer es identificar defectos. Si no encontraste no significa que no hay.

No esta bueno que una actividad de control la ejecture la misma persona que vos queres controlar. Testing unitario si es trabajo del desarrollador. Salvo ese los otros niveles de testing lo recomienda que lo haga gente distinta a la que programo

El testing es la actividad mas cara de al construcción del soft. Entre el 30 y el 50% del costo. Mientras mas critico es el producto en términos de manejo de dinero importante o la vida de las personas, nos acercamos al 50%. Reservar tiempo para el testing, no asumir que no vas a tener errores.

Cuando uno hace el testing del producto no usar la palabra certificación. El testing no certifica nada. No se puede asegurar que son todos los defectos que hay. Hay una relación costo-beneficio, asique en un punto hay que dejar de probar, y no siempre es suficiente.

Diferencia entre error y defecto es el momento en que se advierte. Cuando encuentro un problema en la misma etapa en que ese introdujo es un error. Si se traslado a la etapa/workflow siguiente ya es un defecto. Un error de reqs encontrado en la revisión de pares se corrige ahí. En la etapa de diseño es un defecto. Testing corrige defectos.

Errores y defectos pueden provocar fallas en el sistema o no. No todos provocan fallas en el sistema. Si yo permito ingresar letras en un campo de fecha en algún momento va a provocar fallas.

Diferencia entre severidad y prioridad. La severidad tiene que ver con que tan grave es el defecto encontrado y me ayuda a determinar el impacto. La severidad de los defectos se suelen establecer SLA, cuanto tiempo podes tomarte para corregir. Si es bloqueante el sistema no anda, tenes que resolverlo ya, pero si es cosmética tenes mas tiempo. Se discute la severidad por rrazones equivocadas, no arreglar un problema eliminándolo.

1. Bloqueante: el sistema no se puede usar
2. Cosmetico de visualización de interfaz, ortografía, como muestro los datos
3. Mayor menor o critico depende de la situación. La severidad viene asosciada a un contexto. El mayor o menor da mas problemas para describir.

Prioridad viene del lado del cliente, el producto owner. Desde el punto de vista de mi negocio que tan rápido lo necesito. Pensamos que los bloqueantes son urgente, los de prioridad baja son cosméticos, pero no necesariamente es así. Puede ser bloqueante pero de una parte que no esta en producción.

Define la urgencia del cliente de que tan rápido lo necesita pero visto del negocio y determina cuanto me va a costar ami. A veces un error de cosmética es costoso arreglarlo, y el bloqueante es algo simple de resolver.

Niveles de pruebas. Existen tanto en agile como en tradicional. Esto es lo que se automatiza con practiccas continuas. Están relacionadas con que testing automatizamos. Las pruebas unitarias las hace el desarrollador.

Mientras mas podemos automatizar el testing mejor. Mientras mas nos podemos ayudar mejor. Porque el testing es una actividad carísima. Nivel de testing lo hace cada uno. Si sabemos como lo vamos a probar la calidad del código es mejor. Si vos te podes imaginar como probar significa que estas entendiendo lo que estas haciendo. El tdd es una técnica de desarrollo que construye primero la prueba unitaria. El testing de integración, también conocido como prueba de interfaces. El foco de este testing es si dos componentes que funcionan en la prueba unitaria funcionan bien juntos. Si yo eso lo automatizo tengo continuous integration. La automatización del buil es el ci. En el PUD es una actividad del workflow de testing. A veces la gente de testing no esta capacitada para hacer la integración, así que la suelen hacer los desarrolladores. Finalmente tenemos el testing del sistema o de versión. Lo que vos estas testeando es una versión del producto.

El ultimo nivel es pruebas de aceptación de usuario. Lo hace el usuario. Va en el workflow de despliegue. En scrum va en la review.

Ambientes. Tienen que estar separados, separación de intereses porque uno tiene que llegar con una versión de producto lo mas limpia posible. Los desarrolladores no tienen acceso al ambiente del testing. Ellos integran y generan un build, y testing tiene que ejecutar un ciclo de prueba, ejecutando casos de prueba definidos para una versión, y termina cuando haces todas las pruebas o cuando una te bloquea y no te deja seguir. No hay que arreglar el error y después seguir, tiene que empezar el ciclo de prueba de nuevo. Porque cuando un desarrollador corrige un error introduce mas, por eso es ambiente separado.

Las pruebas de aceptación se pueden hacer en pre-producción para no hacer lio en producción. El problema es que no siempre se incorpora el ambiente de pre-producción por costo. Las pruebas de aceptación no se pueden hacer en prueba porque después hay que desplegarlo y ahí se hacen pruebas también.

Un defecto que no se puede reproducir no es un defecto, porque como se lo reportas al desarrollador. Tener un caso de prueba nos ayuda porque tenemos registrado el paso a paso de lo que hice para poder reportarlo. El uso de casos de prueba me permite reproducir los defectos y hace que una prueba sea sistemática. El otro tipo de pruebas son ad hoc o no sistemáticas.

Normalmente los casos de prueba trabajan sobre la idea de que es imposible probar todo, entonces tenes que elegir. Hay que elegir métodos de prueba que ayudan a definir casos de prueba que sean eficientes y sean la menor cantidad posible. El propósito que tienen es achicar la cantidad de casos de prueba sin reducir la calidad del testing. Buscar donde están los problemas, suelen estar en el borde al limite.

Los casos de prueba tienen datos concretos, sino están mal.

De donde saco info para crear casos de prueba. Se puede sacar de muchos lugares. Uno de los peores lugares es el código, porque voy a poder verificar que funcione bien pero no voy a poder validar, hay que definir los casos de prueba contra los reqs, de ahí hay que sacar. Si no escribieron los reqs como validas. Si los reqs van a los casos de prueba no tenes que hacerlos dos veces.

Condición de prueba. Sobre un caso de prueba lo completa para ver que hace falta para completar esa prueba.

Un caso de prueba tiene lo que quiero probar, lo que necesito y el paso a paso.

Estrategias de prueba tenemos dos. Caja negra y caja blanca. Caja negra se definen las entradas y salidas esperadas. Los desarrolladores que hacen caja negra no tienen acceso al código, seria bueno que tengan acceso a la bd para ver si se graban bien los datos.

Un ciclo de prueba es la ejecución de un conjunto de casos de prueba sobre una versión del producto. El ideal seria tener dos (uno es utopico). El tester define los casos de prueba y ejecuta ese ciclo de pruebas. El primer ciclo de prueba que se ejecuta se llama ciclo 0, porque es la base para poder identificar defectos y trabajar hacia adelante. Hay definiciones respecto de la calidad del producto vinculado a la cantidad de defectos. Hay empresas con definiciones de calidad vinculado a la cantidad de defectos.

Otro temino importante asociado a los ciclos de prueba es el concepto de regresión. Si hacemos pruebas sin regresión, recibo una versión del producto, lo pruebo, genero un reporte y se lo mando a los desarrolladores. Pruebo los defectos que encontré y veo si pasan, no vuelvo a probar todo, solo los que reporte. En ese caos pasa que uno había pasado y se introdujo un error que no vi porque no pase de nuevo por ahí porque la versión anterior andaba. Eso es testing sin regresión. Las pruebas de regresión hacen todo de nuevo, todos los ciclos se comportan como ciclo 0, cuando haces testing automático no cuesta nada hacer regresión, el problema es cuando haces testing manual, muchas veces no te da el tiempo. Esta asociado a los ciclos de prueba

El proceso de pruebas. Se lo llama diseño porque diseñamos los casos de prueba. Hay una planificación, decis que casos de prueba vas a usar con que estrategias de donde vas a sacar los datos quien va a ser responsable de cada cosa. Puedo hacer el plan de prueba a partir de que tengo los reqs. Diseño identificar la necesidad y crear los casos de prueba. Ejecutamos y se arman los ciclos. Encuentro defectos, los reporto y va otro ciclo, hasta definir cuando dejar de probar. Proceso y nivel no es lo mismo, proceso y estrategia tampoco. Los entregables que se generan es el plan de prueba, los casos de prueba (dentro están los datos de prueba que son importantes), reporte de defectos (muy importante en el sentido que hay que documentar de la forma mas detallada posible el defecto para que se pueda reproducir, imágenes, capturas, todo lo que se pueda, y asignarle la severidad y la prioridad).

Se desarrolla de lo general a lo particular y se prueba al revés. Cuando probas es al revés, empezas por pruebas unitarias y de ahí vas para arriba.