Ingeniería de Software

Introducción

¿Qué es software?

El software es, en general, un set de programas y la documentación que acompaña. Entonces, ¿Qué entendemos por software? Lo instintivo es asociar el software a algún ejecutable o a algo instalado. Lo que pasa es que si nosotros nos quedamos con eso solamente tenemos una visión acotada que nos está resolviendo una parte del problema, pero no todo el problema.

El software es conocimiento empaquetado y ese conocimiento puede venir en distintos formatos y en distintos niveles de abstracción, pero el software es conocimiento. Por ejemplo, si yo tengo un requerimiento en el formato que sea o un diseño de base de datos, eso también es software. Cualquier cosa que se genere como información resultante del proceso de hacer software es software, y ese conocimiento se va refinando hasta que llego al código que es la granularidad más fina.

Entonces, ¿Dónde encontramos software hoy en día? En todos lados. Hoy en día todas las industrias son industrias de software. Si vemos alguna industria, su parte principal contiene software.

Razones para no comparar software y manufactura

Una de los motivos principales es que el software es intangible. Mi producto no se puede tocar, por ende, no tenemos mucha visibilidad del avance de un determinado producto. Esto le da un nivel de complejidad diferente. El software es una actividad humano intensiva, es decir, que la materia prima somos nosotros.

Un software no se rompe o no se gasta, pero ¿por qué tenemos tantos problemas? Esto se debe a que el software funciona inserto en un determinado ambiente, que puede ir cambiando. El software no funciona de forma aislada. La problemática que tenemos con la evolución tiene que ver más con cuestiones personales, de cultura de trabajo, organizacionales, etc.

El software no es fácil de modificar, básicamente, porque siempre estamos apurados o porque estimamos mal.

Además, otros de los problemas que tenemos son: la versión final del producto no satisface las necesidades del cliente, no es fácil extenderlo y/o adaptarlo, hay mala documentación, mala calidad y nos lleva más tiempo y costos que los presupuestados.

Buenas prácticas

* El cliente o usuario tiene que participar ya que reduce la probabilidad de fracaso.
* Sponsoreo de la gerencia. Los niveles que toman decisiones en la empresa nos tienen que apoyar.
* La especificación de los requerimientos es muy importante porque si no están claros, podemos llegar a hacer cualquier cosa.
* Planificar.
* Hitos intermedios: Hay que dejar de engañarnos, a veces mentimos para safar.
* Personas involucradas competentes: Si tengo gente que sabe hacer lo que tiene que hacer, nos va a reducir dolores de cabeza.
* Expectativas: a veces el cliente quiere un software que sale 1000 pesos pero pagar solamente 20, debemos reducir las expectativas de los clientes.

El 80 por ciento del costo de incidencia en un producto de software es el esfuerzo, es decir el trabajo de la gente. Si subestimamos esto, le vamos a errar a los costos.

En conclusión, con saber programar no alcanza, es necesario, pero no alcanza. Tenemos que saber conocimientos organizacionales, humanos, de gestión y no sólo conocimiento técnico.

Ingeniería de Software

La intención de la ingeniería de software es tener un enfoque formal y sistemático para hacer software. Mucho de lo que se ha pensado se ha copiado de hacer hardware, pero no es lo mismo. Por ejemplo, el Hardware se puede tocar, el software.

Cuando hablamos de ingeniería de software tenemos que hablar de la gente y de la dimensión de los procesos que vamos a usar para hacer software, de los proyectos como unidad de gestión para hacer software y del producto resultante.

Relación entre procesos, proyecto y producto

Podemos decir que un proceso es un conjunto de actividades estructuradas con un objetivo en común que utilizan recursos para transformar una entrada en una salida. Pero esta definición es una definición genérica de proceso.

Cómo el software es una actividad humano intensiva, la incidencia de la gente sobre el proceso es muy importante. Entonces, tenemos que complementar la definición con herramientas (que hoy en día se busca que sea lo más automatizada posible) y con la gente que tiene que tener habilidades, estar capacitadas y motivadas.

Por esto, podemos decir que un proceso de software es un conjunto de actividades, métodos, prácticas y transformaciones que la gente usa para desarrollar o mantener software y sus productos asociados

El proceso nos define el camino a seguir, pero esto no tiene vida. Se usa en el contexto de un proyecto. Por esto, la relación entre proceso y proyecto es que el proceso se adapta al proyecto. El equipo decide qué proceso quiere usar, qué necesita y qué no necesita, que actividades van a llevar a cabo, y aplicarlo al proyecto.

Tenemos diferentes tipos de procesos

* Procesos empíricos: son otra forma de llegar a lo mismo que proponen los procesos definidos. Aprovecharnos de las experiencias. Para eso, tenemos que establecer períodos cortos de la ejecución de las cosas para poder obtener retroalimentación y así aprender. Funcionan con dos pilares que son la inspección y con la adaptación.
* Procesos definidos: funcionan de una manera más repetitiva y anticipatoria. Tienen una concepción de que yo tengo un conjunto de actividades establecidas con entradas y salidas bien establecidas y una intención de tener visibilidad basada en la repetibilidad. Si logro repetibilidad, puedo lograr control y previsión. Apuntan a que nosotros hagamos las cosas de la misma manera así obtenemos los mismos resultados.

A veces, se comete el error de utilizar los procesos definidos, pero no adaptarlos al proyecto.

Ahora, ¿Qué obtengo en el proyecto? Primero tengo que juntar todos los recursos que me hace falta, integrar a las personas y entonces surge la relación entre proyecto y producto. La relación es que el proyecto va a obtener como resultado un producto, que en nuestro caso es un producto de software.

El proceso se adapta a un determinado proyecto en particular, que va a obtener como resultado un producto de software.

Dentro de los enfoques empíricos, tenemos la filosofía Lean y Ágil.

Cada enfoque ha planteado sus propios frameworks y metodologías para gestionar proyectos.

Ciclos de vida

Un ciclo de vida, o modelo de proceso, de un proyecto de software es una representación de un proceso. Grafica una descripción del proceso desde una perspectiva en particular. En resumen, el ciclo de vida define el orden en que se van a realizar las tareas, cuántas tareas se van hacer en cada momento y el momento de ejecución

Tipos de ciclos de vida.

* Secuencial: Por ejemplo, ciclo de vida en cascada.
* Iterativo: Por ejemplo, ciclo de vida iterativo/incremental.
* Recursivo: Por ejemplo, ciclo de vida en espiral.

Los ciclos de vida no son lo mismo que procesos. La diferencia es que, en el proceso se define lo que tenemos que hacer, que entra, qué sale, quién las hace, etc., y a ese proceso yo le tengo que aplicar el ciclo de vida.

El ciclo de vida me define en qué orden tengo que hacer las cosas y en qué momento. Lo que hace es completar la definición del proceso con cómo lo voy a implementar en mi proyecto.

Planificación de un proyecto

Un proyecto es una unidad de gestión o unidad de organización que administra recursos para obtener un resultado. Algunas características son:

* Se usa como medio para obtener un producto o servicio.
* El resultado que se obtiene de su ejecución tiene que ser único y diferente.
* Tiene una fecha de inicio y una de fin.
* Afectan recursos.
* Tiene que tener un objetivo con alcance gradual, es decir, tener tareas interrelacionadas que se realizan gradualmente. Este objetivo tiene que cumplir dos características:
  + Claro: Que se entiende y no tenga ambigüedades.
  + Alcanzable: Con las condiciones de recursos que tenemos, el equipo sienta que el objetivo se pueda lograr.

Un proyecto que se administra puede fracasar y uno que no se administra va a fallar seguro. Por esto, aparecen las disciplinas de gestión cuyo propósito es definir una planificación del proyecto y luego definir lineamientos de cómo vamos a ir supervisando si esos planes se cumplen o no se cumplen.

La realidad es que los planes pueden llegar a cambiar. Esto no es excusa para no planificar. Si se desvía, se replanifica.

De la mano de planificación, viene el monitoreo y control. El monitoreo es permanente, y el control es a hito determinado. Un hito es un punto de control, un momento especificado que yo defino que voy a detenerme un momento para saber cómo me está yendo. El control se hace a hitos y el monitoreo es más frecuente. En la gestión de un proyecto, se deben considerar ambos, el monitoreo y el control.

Plan de proyecto basado en un proceso definido

Lo que necesitamos para planificar un proyecto de software, según el enfoque basado en los procesos definidos, es:

1. Objetivo del proyecto: Determinar para qué se va hacer este proyecto. Es el objetivo del proyecto, no del producto.
2. Alcance del proyecto: Un alcance del proyecto es el trabajo, y sólo el trabajo, que se tiene que hacer para cumplir con el objetivo de proyecto. Cabe aclarar, nuevamente, que son los alcances del proyecto y no del producto. Por ejemplo, un alcance de un proyecto es relevamiento de requerimientos o testing, en cambio, el alcance del producto es lo que quiero que haga el producto, por ejemplo, gestión de clientes.
3. Proceso y ciclo de vida del proyecto: Un proyecto usa un proceso porque lo necesita como marco de referencia para definir tareas que contiene de manera interrelacionadas. Acto seguido al elegir el proyecto es seleccionar el ciclo de vida. Un proyecto se puede usar en un tipo de ciclo de vida o en otro ciclo de vida. Cabe aclarar que se habla del ciclo de vida del proyecto y no del producto.
4. Estimaciones: En este punto, definimos el dimensionamiento del proyecto. ¿Qué se estima?
   1. Tamaño: Se estima el tamaño del producto, no del proyecto. Responde a la pregunta: ¿qué voy a construir? Como tenemos que obtener un número, tenemos que determinar en base a qué voy a estimar el tamaño. Al principio se estimaba en base a líneas de código, pero esto no sirve porque varía demasiado. Además, si tengo que planificar para un producto que no construí, entonces no sé cuántas líneas de código voy a necesitar. Los lenguajes de programación también hacen variar las líneas de código. Hoy en día, se utilizan para realizar cambios, por ejemplo: cuántas líneas de código tengo que tocar para realizar un cierto cambio. Entonces, surgieron otras formas, como contar requerimientos, módulos, casos de uso, clases, entre otras. Se tiene que elegir algo que tenga al principio del proyecto.
   2. Esfuerzo: Esta estimación se deriva de la estimación del tiempo. Esta estimación está basada en el tamaño y me índica cuántas horas personas lineales necesito para construirlo. Horas personas lineales: asumo que para el esfuerzo trabaja una sola persona haciendo una sola cosa a la vez. De esta estimación obtengo una bolsa de horas.
   3. Tiempo: Después de la estimación de esfuerzo derivo a la estimacion del tiempo. En este punto, empiezo a pensar en la cantidad de diís. Por ejemplo, si la gente trabaja 8 horas, que días trabaja la gente, etc.
   4. Costo: Lo último que tenemos que estimar es el costo. El costo se mide en base del esfuerzo. Si se quiere apurar el proyecto, va a salir más caro porque vamos a necesitar más esfuerzo. El costo es una estimación monetaria
5. Riesgos: El riesgo tiene asociada una probabilidad. No es lo mismo riesgo que problema. El riesgo es la probabilidad de ocurrencia de algo que me impacte negativamente en mi proyecto. Es la probabilidad de algo que pase me dañe el proyecto. Los riesgos se miden con la exposición, que tiene dos variables. Por un lado, la probabilidad y por el otro lado el impacto. La probabilidad mide la posibilidad que el problema se presenta. Y el impacto mide cuanto daño me va a hacer el problema. No están alineados, podemos tener alto impacto con baja probabilidad, alta probabilidad con bajo impacto o tener la misma medida de impacto que de probabilidad.

Tenemos que decidir de que riesgos nos vamos a ocupar porque los proyectos tienen muchos riesgos y no nos vamos a poder a ocupar de todos. Para saber si ocuparme del riesgo o no, tenemos que unificar la probabilidad y el impacto a través del cálculo de la exposición que es igual a la probabilidad por el impacto. Puedo ordenar los riesgos a través de la exposición.

Se tiene que tener una estrategia proactiva frente a los riesgos, no reactiva. El que tiene una actitud reactiva frente a los riesgos es el que dice: espero que sean problema. La proactiva trabaja con anticipación: identifica los riesgos, calcula la exposición, ve si se puede mitigar (bajar la exposición, es decir, qué puedo hacer para bajar la mitigación).

Cuando no se puede mitigar la exposición, entonces se realiza la contingencia. La contingencia es decir, por ejemplo, "me voy a preparar para cuando esto ocurra saber que hacer".

También tengo que pensar cuando pongo en práctica la mitigación y/o la contingencia, eso se llama evento disparador. Los riesgos pueden vivir a lo largo del proyecto, o que pueden aparecer después, o desaparecieron, por ende, hay que supervisándolo.

1. Calendarización o programación del proyecto: En este punto, se definen las qué tareas hacer, cuándo empiezan, cuándo terminan, las dependencias entre tareas y quién las va a hacer. Para armar la calendarización, necesito las tareas que se eligieron en el proceso. Para determinar la programación, tenemos que tener en cuenta dos conceptos diferentes: el tiempo y el esfuerzo. El tiempo es en cantidad de días pero el esfuerzo es cuantas horas personas lineales se necesita. Por ejemplo, una tarea requiere un esfuerzo de 10 horas, y el que realiza la tarea trabaja 4 horas por día, entonces, se tiene un tiempo de 2 días y 2 horas de un esfuerzo de 10 horas. Una cosa es gestión del tiempo y otra cosa es gestión del esfuerzo. En resumen, el esfuerzo es el trabajo que realmente se le dedica a la ejecución del trabajo, que no es lo mismo que el calendario
2. Métricas: Se define que métricas se van a obtener en el contexto del proyecto. Una métrica es un número. Su definición formal es el grado de presencia de una característica en el ámbito que yo quiero medirla, entonces, tiene que ser un valor cuantitativo que me de visibilidad sobre lo que yo quiero medir. Acá, se planifica que métricas puedo tomar yo. Los dominios de las métricas de software son: métricas de producto, proyecto y proceso. Tengo que definir en el plan que métricas de producto (ej.: tamaño, cantidad de efectos por severidad), de proyecto me interesan (ej.: cuantas horas llevo consumida, cuanta plata lleva consumida) y de procesos me interesan (estas son más globales, a nivel de organización, no se definen en este plan de proyecto)
3. Seguimiento y control del proyecto: Como voy a hacer el seguimiento y el control del proyecto. También, se planifica como se va a monitorear y supervisar. En este punto, tenemos que definir, por ejemplo, qué reuniones vamos hacer, qué reportes vamos a generar y a quién se lo vamos a dar. En resumen, como voy a hacer yo el seguimiento y control del proyecto.

User stories o historias de usuario.

Lo complicado que hay en la industria del software es decidir qué producto se quiere construir. La técnica trata de resolver esta problemática: evitar los problemas de comunicación que se establecen cuando tenemos que compartir la visión de que producto queremos realizar. Y lo próximo más difícil es estimar.

Asociado a lo que es la gestión de requerimientos, hay una serie de principios, tanto para ágil como para lean, que justifican por qué nosotros vamos a abordar la problemática de los requerimientos de la manera que decidimos abordar la problemática de requerimientos.

Lo primero tiene que ver con que hemos acordado entregar algo de valor para el cliente y de manera frecuente, es decir, ciclos de entrega de producto funcional en ciclos cortos de tiempo. Las interacciones en el ambiente ágil son interacciones que deben durar de máximo 1 mes. Su justificación descansa en la idea de que tenemos que tratar de hacer iteraciones en un tamaño en el que el cliente no cambie de idea en ese período de tiempo y porque la retroalimentación es menos costosa.

Otro principio, tiene que ver con recibir cambios en los requerimientos aún en etapas tempranas, quiere decir, que no tenemos que resistirnos a los cambios. (Explicación de las variables y de los triángulos).

Los enfoques tradicionales trabajan a partir de que es lo que el cliente nos define a nosotros. Nosotros que trabajamos en software tenemos tres variables asociadas como dimensión de un proyecto de software, ellas son: alcance (ósea, los requerimientos), recursos (son transformables en costos, se derivan en dinero) y el tiempo. La gestión tradicional es resistente a los cambios porque empieza negociando con el cliente los alcances y trabajando con él para determinar un alcance, y ese alcance es el que sirve para luego derivar los recursos que hacen falta para poder construir ese producto y el tiempo necesario para poder entregar esa cantidad de funcionalidad que se pactó con el cliente. En resumen, del alcance se derivan los recursos y el tiempo. Cuando el cliente cambia algún alcance, por cualquier razón, la consecuencia natural es que uno pudiera ajustar los recursos y el tiempo, pero es muy difícil que esto pase. Por esto, es que se intenta resistirse a los cambios. Es decir, se intenta que los alcances sean fijos porque los recursos y el tiempo son variables. La idea, como líder de proyecto, es controlar dos de las tres variables.

Los ágiles, para poder ser flexibles con los requerimientos, invirtieron el triángulo. Entonces, dejan fijo el tiempo y los recursos y dejan variable los alcances. De esta manera, el cliente tiene la libertad de ir cambiando de opinión. Ágil, para esto, realiza iteraciones de duración fija, por ejemplo 1 mes. Esto implica que los recursos me queden fijos. Con el cliente se negocia: con este tiempo y estos recursos puedo entregarte esta cantidad de requerimientos. El dueño del producto, o **product owner**, es una persona que conoce del dominio del producto, viene del lado del negocio, y tiene capacidad para tomar decisiones, no sólo presupuestarias sino que de priorización de características.

El product owner trabaja con el equipo para cumplir con el principio de mejorar la comunicación.

Este product owner nos va a responder todas las dudas que tengamos, por ende tenemos que hacerlo parte del equipo. Si yo quiero aplicar esta técnica, el product owner tiene que estar dispuesto a ser parte del equipo porque si no la técnica me va a fallar.

La gestión ágil de requerimientos trabaja con un enfoque de tipo “just in time”, y tiene que ver de eliminar desperdicios. Uno de los desperdicios más importantes es escribir requerimientos que después van a cambiar.

El **principio de diferir** compromisos funciona de esta manera: dado que el software es conocimiento, al principio del proyecto uno no lo tiene. Entonces si uno toma decisiones tempranas, respecto a los requerimientos, arquitectura, etc., tiene una altísima probabilidad de equivocarse por la falta de información. El principio dice que mientras más tarde yo pueda decidir, voy a ver recopilado más información y voy a estar en mejores condiciones de tomar una decisión, entonces, el principio completo dice diferir compromisos hasta el último momento responsable, porque en algún punto la decisión hay que tomarla.

La técnica que utilizamos para bajar a tierra estos enfoques se llama USER STORY que tiene tres partes: conversación (charla que vamos a tener entre el equipo y el product owner), tarjeta (parte visible que expresa la descripción corta de una funcionalidad, que tiene una sintaxis especial) y la confirmación (forma concreta de poder demostrar que nosotros al trabajo lo hicimos)

La conversación, si bien no se registra, es la más importante, porque ahí es dónde está la adquisición de conocimiento entre el product owner y nosotros.

Las ideas de metodologías ágiles están incompletas por definición, porque le dejan al equipo la decisión de que cada uno de los miembros puedan aportar y tomar la decisión respecto de qué procesos quieren usar, y eso no significa que lo único que se tiene que hacer son las 4 cosas que dice SCRUM, porque SCRUM es sólo para gestionar equipos. Todas las metodologías son completas por definición, nosotros como equipo tenemos que decidir con qué prácticas vamos a completar en cada caso en particular.

Las tarjetas tienen una sintaxis. Tiene tres partes: un quién (rol de usuario), un qué (funcionalidad del producto) y un para qué o por qué (representa el valor del negocio). Esta sintaxis apunta a que el valor de negocio es fundamental para justificar y para priorizar esta user en relación de otras users. Como el product owner tiene que priorizar características, con esto va a tener la posibilidad de elegir que tiene más valor de negocio.

La mayoría de la gente apurada, con el who. Todas las historias ponen “como usuario”. Lo que cada rol va a hacer con el producto determina también la experiencia de usuario que vamos a tener que usar para cada usuario. Y si no invertimos tiempo en especificar quiénes son los que van a usar el producto, qué características tienen, qué expectativas tienen, etc., es muy probable que hagamos un producto que no le sirve o que no le guste. Entonces, la identificación de las user parte, la mayoría de las veces, de quiénes van a ser los usuarios. Poner “usuario” y no poner nada, es lo mismo. Y finalmente, hay que poner lo que va a hacer el sistema.

**¿Qué son la user story?** Representan una necesidad de usuario. Una user story es un requerimiento a nivel de usuario o a nivel de negocio. Su nivel de granularidad es mucho mayor que un caso de uso. La user story está a un nivel más arriba es una necesidad de usuario expresada en términos de usuario que es un rol no técnico porque las user stories las define el product owner. En la técnica dice “las escribe el PO” pero en la práctica quién usa el teclado, o el marcador, etc., es un detalle. Lo importante es que el conocimiento del producto salga de la cabeza del product owner (PO).

**¿Dónde se ubican las user stories?** Se ubican en algo que se llaman product backlog PBI (lista priorizada de producto). El P.O. es el dueño de este artefacto. Es un contenedor genéricamente de ítems de product backlog porque en el product backlog puede haber user stories pero también otras cosas, por eso se lo denomina ítems de product backlog.

La granularidad de una user story es la necesaria como para que entre en una iteración. De todas formas, el tamaño depende de las decisiones tomadas por el equipo, es decir, si el equipo decidió que las iteraciones sean de 15 días el tamaño de la US será acorde a ese tiempo pero si definimos iteraciones de un mes, el tamaño será otro. Entonces, depende del time box del spring.

En el product backlog no solo hay US, hay defectos de iteraciones anteriores, deuda técnica (la deuda técnica es un concepto en el desarrollo de software que refleja el costo implícito del retrabajo adicional causado por la elección de una solución fácil ahora en lugar de utilizar un mejor enfoque que llevaría más tiempo).

El PO se dice que es el dueño del product backlog porque tiene la potestad de priorizar, eliminar, agregar, modificar, repriorizar con total libertad. Esto está relacionado con la idea de flexibilidad con los requerimientos y que en el ambiente ágil no tiene que estar terminado para empezar a trabajar.

**¿Cuánta es la cantidad mínima de US en un product backlog (PBI) para empezar a trabajar?** La cantidad de US necesarias para poder completar una iteración de trabajo. No tiene que estar el producto completo especificado. En el product backlog nunca van a estar todos los requerimientos del producto porque vamos empezando y vamos trabajando y segundo porque los que se van implementando se sacan del PBI.

Lo que se ve reflejado en el PBI son cosas que están pendientes de hacer para el producto, si quiero saber todo lo que hace el producto voy al manual de usuario no recurro al PBI.

Las US son porciones verticales, debe cortar transversalmente a las capas para poder entregar una característica de producto que sea de valor para el cliente.

Técnicas para trabajar roles

Los roles son por lo que uno primero empieza. Quiénes son los que lo van a usar.

**Primer técnica base: Descripción del rol de usuario**

Se debe encontrar un nivel de granularidad adecuado para el rol de usuario para que se pueda potenciar el “qué quiere hacer”. Es decir, si describimos los roles de la facultad decimos que hay alumnos, docentes, bedeles, etc. Dentro del rol de docentes estaría el jefe de cátedra, adjunto; por lo tanto, si solo ponemos docente es una descripción muy genérica del rol.

**Segunda técnica: Personas**

Resuelve la problemática de los roles genéricos haciendo una descripción de un conjunto de usuarios definiéndolos individualmente para tratar de construir una experiencia de usuario que sea acorde con cada perfil.

**Tercer técnica: Personajes extremos**

Buscar situaciones que están en el borde y ver cómo impactaría en las características del producto. Por ejemplo: Qué pasaría si el que va a usar el sistema es ciego, anciano.

Determinar el Product Owner no es tarea sencilla, ya que los más capacitados para estar en ese lugar están muy ocupados para llevar a cabo el rol. Una solución para esto son los Proxies.

Proxies son intermediarios del negocio que funcionan como un representante del que debería ser el PO. Este debe comunicarse con el PO para informarle de los avances, decisiones, problemáticas, etc.

Los proxies debería tomar ciertas decisiones, no tienen que depender al 100 % del PO, sino sería como una “puerta giratoria” que nadie toma las decisiones.

Recordar: El tiempo es un factor que no se mueve, por lo tanto, hay que ser rápidos en la toma de decisiones.

La clave fundamental del éxito del proyecto está en el PO.

Criterios de aceptación de US: No tienen un lugar formal dentro de la US. Las ponemos en las notas de la Card. Me sirven para basarme en la construcción de las pruebas.

Pruebas de aceptación: Me van a permitir demostrar si se cumplió o no con los criterios. Como mínimo tengo que tener dos pruebas por cada criterio una para el camino feliz “PASA” y otra para el camino triste “FALLA”. Conjunto de cosas que tenemos que construir después en los casos de prueba.

Criterio de aceptación vs Prueba de aceptación: Los criterios vienen de las reglas de negocio o requerimientos no funcionales. Las pruebas se derivan de los criterios. En resumen,

Criterios importantes asociados a las US: Ayudan a determinar:

1. **Definition of Ready:** Conjunto de preguntas que tengo que hacerme para saber si la US está en condiciones o no de ser incluida en la siguiente iteración de desarrollo. Asociado a un modelo de calidad que se llama “INVEST Mode”.   
   El INVEST mode es un acrónimo, cada letra representa una palabra. Es algo que la US tiene que tener para poder cumplir de **mínima** con el criterio de si está lista o no.
   1. I: Independiente→ Cada US puede ser elegible por el PO en cualquier orden. No deben relacionarse las US entre sí.
   2. N: Negociable → La US debe decir qué es lo que quiero y no cómo lo voy a implementar.
   3. V: Valuable → Tiene que ser de valor para el cliente.
   4. E: Estimable → Podamos asignarle una story point (valor, peso) para poder determinar qué tamaño tiene esa US. El problema de la estimación es cuánta información y cuánta incertidumbre tengo. Si no puedo estimarla, no puedo incluirla en la próxima iteración porque no tengo información suficiente para desarrollarla. Si es **no** estimable se le asigna un nombre: SPIKES.   
      Las SPIKES se clasifican en dos tipos: Técnicas y funcional. **Técnica**: Tecnologías que no se han usado y por lo tanto no sé si van a andar. **Funcional**: El cliente todavía no tiene claro cómo quiere una determinada cosa, entonces no estoy en condiciones de estimar si él no está en condiciones de decir lo que quiere.  
      En una US pueden existir los dos tipos de SPIKES o cualquiera de ellas.   
      La US queda en el product backlog pero no puede entrar en la iteración hasta tanto no sea estimable.
   5. S: Small (Pequeña) → Una US debe ser consumible en una iteración. Las US pueden dividirse para poder ser hechas en una iteración.
   6. T: Testeable→ Relacionado con las pruebas de aceptación. Si no puedo mostrar de alguna manera que a esto lo implementé, tampoco lo voy a poder construir. Por lo tanto, si no es testeable tampoco es implementable.
2. **Definition of done:** Esta relacionado con la US terminada. Es una lista de pregunta que ayudan al equipo a tener un mismo criterio de cuándo está algo listo para ser mostrado al cliente. Cada equipo construye su propio definition of done. Si puedo decir si ya terminé un US o aún no.

Diferencia entre definition of ready y definition of done: DOR-> Si una US puede estar en una iteración para ser concluida. DOD-> Con la idea de si terminé realmente la US o no.

Como hay distintos niveles de granularidad sobre las US puede haber DOD asociado a una US o a toda una iteración completa del producto donde se van a sumar otras características de calidad diferentes por ejemplo: Una prueba de sistema completo que eso es un DOD de la iteración y no de una US en particular.

En el product backlog vimos que hay tamaños distintos. Los más grandes pueden ser **temas**, un tema es un conjunto de US o épicas (una US muy grande, es decir, que no entra en una iteración). Las cosas más “gruesas” se ubican abajo del product backlog porque para poder implementar una épica tengo que descomponerla en US y recién ahí se puede subir de prioridad. La responsabilidad de esto la tiene el PO, es el que se tiene que hacer cargo de ese trabajo y lo puede hacer en cualquier momento. La actividad se llama **refinamiento de product backlog**.

16/08/2019

El contenedor de características de producto se llama, según SCRUM, es el product backlog.

Product backlog es una lista priorizada de productos. En términos generales, independientemente de la técnica, esto representa las características del producto que queremos construir. Sumatorias de los items (IPB).

Las técnicas para trabajar items de PB:

· User story

· Épicas

· Temas

SCRUM parte desde un producto backlog.

En términos generales, uno tiene una idea, como por ejemplo: hacer una app. Más allá de la cuestión del dinero, esta profesión es una profesión de creación, entonces la idea es que nuestro producto, la gente lo use. Otra realidad, la gente usa un porcentaje del producto. Entonces, si tenemos la visión de entregarle al cliente el 100 por ciento de la funcionalidad, es una pérdida de esfuerzo. La realidad hace que se dispare el enfoque de lean ágil: si sólo se usa el 20 por ciento, hagamos eso y después vemos si sacamos otras funcionalidades.

Tenemos que empezar a pensar para que nosotros queremos entregarle el producto al cliente. El usuario puede confiar en los resultados que se están entregando

El gran desafío que tiene que soportar un producto de software para trascender en el mercado es poder cruzar la línea que divide lo supuestamente básico llegamos a que el foco de lo que yo quiero entregarle al cliente no es característica de software sino valor de negocio.

¿Cómo empieza todo producto de software? Se empieza con “visión”: poder crear una visión de lo que nosotros queremos que el producto haga. Esa visión no es un conjunto de características de software, una visión es “quiero mejorarle la vida a la gente a través de esto” o “quiero ganar mucha plata”. La visión es como quiero cambiar el mundo con mi producto. El primer filtro tiene que ver con cuestiones estratégicas. Por ejemplo: voy a tener alguien para ofrecerle esto? Quiero dedicar mi vida para hacer esto?. Son características más allá de la tecnología y la técnica. Esto empieza a formar la visión. Si empezamos a desarrollar la visión con el enfoque lean ágil, el ciclo de vida de la evolución del producto va a ser iterativo. La unidad de entrega del producto se llama “reléase”. Release es una versión de producto que está en condiciones de ponerse en producción. Con el enfoque Lean agil no va a estar 100 por ciento completa su funcionalidad, casi que nunca va a estar completa. Como por ejemplo: Word.

El foco de SCRUM es generar un producto que si el cliente quisiera, podrías ponerlo en producción o no, pero la clave es entregar algo que funciona. No entregamos prototipo, diseños, ni documento de arquitectura. Entrego producto, que se puede poner en producción. De acá el fundamento de que las user story son porciones verticales

Esa visión de producto se materializa en un primer nivel en algo que se llama Mapa de ruta o Route Map. Es una planificación a mediano plazo. El route map está planteado a nivel de temas o de épicas (alto nivel). Del route map, se deriva el product backlog. De ese product backlog hace la planificación de los reléase. Cada release yo identifico una cierta cantidad de sprints (iteración de scrum). Cada sprints integra el release plan que planifica a alto nivel que es lo que se espera entregar al final de cada sprints. Y después para cada sprints en particular hay que hacer la primera ceremonia de SCRUM, que se llama sprint planning, tiene que tener como resultado el sprint backlog.

Tenemos que ver cuántas de las cosas que tenemos que hacer es esfuerzo y cuánto es desperdicio. Tenemos que entender la clave para ver qué lo que es valor y qué lo que es desperdicio lo va a definir el cliente. Básicamente, el desafío es no perder de vista que lo que tengo que construir es poquito cada vez tiene que ser el poquito de conjunto de características correctas, no que funcione bien.

El concepto sobre el que vamos a trabajar es el MVP o producto mínimo viable o minimal viable product. Ese MVP está pensado en términos de generación de valor para el cliente y tratar de eliminar lo más que se puede desperdicios. El primer release que vamos a sacar en el mercado eso es lo que llamamos MVP. MVP es una oportunidad de generación de conocimientos respecto de que si es el respecto del producto que se va a vender con la total convicción de que tenemos una única oportunidad para dar una buena primera impresión. Son pocas veces las que un producto primero ha fracasado y después si ha sido bueno. La premisa básica del MVP es que primero, tiene que ser un producto (que el usuario lo pueda ver funcionando) pero que no necesita tener tanto funcionamiento sino que características creemos nosotros que tiene que tener para enganchar al cliente y en algún caso, para buscar inversores, etc. Nos da fundamentalmente, un ciclo de realimentación sobre el producto que en muchos casos va a determinar si lo tiro a la basura o lo sigo produciendo.

Después del MVP se obtiene la primera versión del product backlog. Mínimo de características del producto que tiene que tener un product backlog: lo mínimo que necesite para poder dar una vuelta de iteración.

23/8

Estimaciones

Cuando uno estima con las estimaciones basadas en procesos definidos el proceso de estimación tiene un orden que hay que respetarlo:

Primero se estimaba el tamaño (del producto) en términos de software pueden ser clases, requerimientos. Se recomienda que se puede estimar con algo que se pueda contar lo más antes posible. Debemos estimar algo que podemos obtener pronto, y eso puede ser feature, caso de uso, requerimientos. Algo con lo que uno cuente para hacer las estimaciones tempranas. Las estimaciones se hacen en la etapa del pre proyecto. Esta es la base para derivar lo siguiente que se estima que es el esfuerzo. Este tiene una única medida para contar que es en horas personas lineales: asumiendo que una única persona va a trabajar haciendo una única cosa a la vez. Una bolsa de horas que harías ese producto con ciertos recursos. De acá se deriva el calendario. Ahí si tenemos en cuenta cuanta gente va a trabajar, si va a trabajar en paralelo, las dependencias entre las actividades, cuántas horas reales están las empresas y cuántas horas trabajan. Luego del esfuerzo, se estima el costo.

Este proceso de estimación para trabajar en procesos definidos. Hay varios puntos fijos de reestimación. Y después hay reestimaciones cuando hay cambios muy grandes. Todo esto en el enfoque de trabajar con procesos definidos. En resumen, queda tres tipos de estimaciones: estimación preliminar, estimación cuando planificás el proyecto, y las reestimaciones.

La precisión en la estimación se alcanza cuando se está llegando al final de la construcción del producto.

Quién estima en este tipo de proceso tradicional? Es el líder del proyecto cuando estamos en la planificación del proyecto. Pero cuando estamos en la etapa anterior, en la preestimación, ahí aparecen distintas técnicas para estimar y la que más se utiliza es la que se llama juicio experto. Tenemos el juicio experto individual o puro (es un gurú estimador que hay en la organización y es el que dice “tanto”.) y el juicio experto grupal (se escucha la opinión de varios expertos). Otra técnica que es por analogía (se empieza a recolectar información histórica sobre proyectos. El problema es que puede llegar cambiar las circunstancias y cambia todo.). Otras técnicas que son más complejas que son algorítmicas: punto de función, concomo (técnica algorítmica de estimación de esfuerza basada en una estimación de tamaño basada en código), punto de objeto.

El otro enfoque

Time boxing: Significa que yo tengo iteraciones de duración fija que no se puede cambiar. Acá negociamos los alcances. Por esto se insiste en que las US que sean porciones verticales.

Cómo estiman?

En primer lugar dice que las estimaciones ágiles son relativas, es decir, que hacés una aceptación de que somos mejores comparando que haciendo estimaciones sin tener marcos referenciales. La US que usamos para comparar se llama canónica. Cómo surge la canónica? Las estimaciones ágiles dicen que estima el que hace el trabajo. Hay puntos distintos de estimación: hay un primer momento de estimación que se hace estimando el product backlog, es decir, los ítems de product backlog se estiman, sólo que en ese momento se los puede estimar con un nivel de granularidad más gruesa. En Lean las estimaciones son opcionales porque son costosas si no tenemos precisión porque es un valor que es difícil alcanzar. Las estimaciones ágiles buscan certeza, no precisión.