UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Centro de e-Learning

Product Owner:

Gestión de Productos con Metodologías Ágiles

Centro de e-Learning SCEU UTN - BA.
Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Unidad 4:

Gestión de backlog

Presentación:

En la presente unidad vemos cómo un product owner realiza una gestión efectiva del backlog. La idea es aprender cómo coordinar la entrega de valor con la adquisición de conocimiento de manera sustentable.

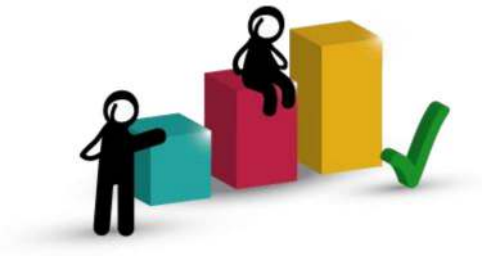
Objetivos:

Que los participantes:

- Comprendan qué es la visión de producto y para qué se necesita.
- Entiendan las herramientas para consensuar y compartir la visión.
- Se introduzcan en las prácticas más modernas para definir un producto.

Bloques temáticos:

1. El Backlog del producto y sus componentes (PBIs).
2. Estimar o No Estimar, esa es la cuestión.
3. Técnicas de Estimación y no estimación.
4. Validación frecuente con el usuario.
5. Priorización de corto plazo y repriorización.
6. Manejo del flujo de trabajo con Kanban.
7. La Deuda Técnica, sus consecuencias y cómo gestionarla.



Consignas para el aprendizaje colaborativo

En esta Unidad los participantes se encontrarán con diferentes tipos de actividades que, en el marco de los fundamentos del MEC*, los referenciarán a tres comunidades de aprendizaje, que pondremos en funcionamiento en esta instancia de formación, a los efectos de aprovecharlas pedagógicamente:

1. Los foros proactivos asociados a cada una de las unidades.
2. La Web 2.0.
3. Los contextos de desempeño de los participantes.

Es importante que todos los participantes realicen algunas de las actividades sugeridas y compartan en los foros los resultados obtenidos.

Además, también se propondrán reflexiones, notas especiales y vinculaciones a bibliografía y sitios web.

El carácter constructivista y colaborativo del MEC nos exige que todas las actividades realizadas por los participantes sean compartidas en los foros.

** El MEC es el modelo de E-learning colaborativo de nuestro Centro.*

Tomen nota;

Las actividades son opcionales y pueden realizarse en forma individual, pero siempre es deseable que se las realice en equipo, con la finalidad de estimular y favorecer el trabajo colaborativo y el aprendizaje entre pares. Tenga en cuenta que, si bien las actividades son opcionales, su realización es de vital importancia para el logro de los objetivos de aprendizaje de esta instancia de formación. Si su tiempo no le permite realizar todas las actividades, por lo menos realice alguna, es fundamental que lo haga. Si cada uno de los participantes realiza alguna, el foro, que es una instancia clave en este tipo de cursos, tendrá una actividad muy enriquecedora.

Asimismo, también tengan en cuenta cuando trabajen en la Web, que en ella hay de todo, cosas excelentes, muy buenas, buenas, regulares, malas y muy malas. Por eso, es necesario aplicar filtros críticos para que las investigaciones y búsquedas se encaminen a la excelencia. Si tienen dudas con alguno de los datos recolectados, no dejen de consultar al profesor-tutor. También aprovechen en el foro proactivo las opiniones de sus compañeros de curso y colegas.

1. El Backlog del producto y sus componentes (PBIs)

Podríamos definir al backlog del producto como; *"Una lista priorizada de características a implementar sobre el producto por el equipo..."* Esta definición puede aplicar para cualquier tipo de enfoque y fuera de contexto no aporta mucho más valor. Al utilizar el paradigma de construcción ágil, una de las principales diferencias es que no es necesario (ni deseable) tener al inicio del proyecto una definición detallada de todos los ítems, así como tampoco esperar que sea una versión final de los requerimientos. Típicamente se inicia (como hemos visto en la unidad anterior) con definiciones de alto nivel para luego detallar lo más próximo a empezar a construir (just in time). La segunda (y tal vez principal) diferencia es que el backlog en un enfoque ágil es construido colaborativamente; incluyendo a usuarios, desarrolladores y demás involucrados en el producto de una u otra manera. De esta manera **será consensuado tanto para los que lo construyen como por quienes usarán el producto**.

Tipos de componentes del Backlog (PBIs)

Personas que han tenido una introducción a los marcos de trabajos ágiles conocen que comúnmente se expresan funcionalidades a través de historias de usuario, lo cual puede ser practico por simpleza y facilidad de entendimiento, pero de ninguna manera es el método mandatorio. A continuación describiremos diversas posibilidades de como expresar ítems en el product backlog:

- Características o funcionalidades:
  - Historias de Usuario (user stories), Son breves y simples descripciones de una funcionalidad deseada desde el punto de vista de un usuario. Por ejemplo, podemos decir: *COMO comprador QUIERO visualizar los ítems cargados en el carrito antes del check-out PARA revisar mis selecciones previo al pago.*
  - Casos de uso: Si bien no es común, es posible la utilización de estos documentos como ítem del backlog. Habiendo dicho esto, en general los casos de usos son bastante inclusivos y bajan a detalle la implementación por lo cual tienen a ser costosos para un paradigma de ciclos cortos como Scrum, pero si se utiliza un proceso como Kanban puede aprovecharse, aunque se recomienda que sean lo más desagregados posibles y sean acompañados con conversaciones (como veremos después se hace para las historias de usuario).
- Errores o Bugs: Aquellas fallas detectadas en entregables terminados un tiempo atrás e implementados nos generan una necesidad de cambio que, a menos a nivel práctico, no tiene gran diferencia con la construcción de una nueva funcionalidad, por lo tanto, se los incluye en el backlog y compite con las prioridades de los demás ítems.
- Trabajo técnico: En este caso hacemos referencia a necesidades que se pueden requerir para ajustes tecnológicos, así por ejemplo la instalación de un parche del sistema operativo, la migración a una nueva versión de base de datos o trabajos de performance. Estos trabajos pueden no ser requeridos de manera directa por el Product Owner, pero son

necesarios para el mantenimiento y evolución del producto por lo cual deben verse reflejados y ser priorizados en un equilibrio con las necesidades del negocio.
- Adquisición de conocimiento - Spike: Este tipo de ítems se puede usar para poder obtener la información que nos permita definir nuevos ítems, estimar impacto o ganar conocimiento. Si bien no entregan valor directo al usuario, sirven como base para avanzar. Siempre debemos asignarle una duración limitada y alcance concreto.

Información necesaria en cada PBI

No existe un template perfecto para preparar los ítems y depende mucho del acuerdo que exista en el equipo y el nivel de comunicación. Habiendo dicho eso, lo más común es que contengan algunos de los siguientes datos para considerarlas "listas" (o criterio de ready que veremos en la próxima unidad en detalle)[1]:

- Descripción: Información que permita entender el objetivo buscado (o el para qué).
- Estimación: Suele ser de utilidad tener un orden de magnitud de cada ítem que permita compararlos (luego hablaremos sobre las estimaciones y cuando podemos evitarlas).
- Orden de importancia: El PO es el responsable de situar la posición relativa de cada ítem dentro del backlog del producto.

Atributos que involucran a los PBI

- Valor: Hace referencia al beneficio que otorga al usuario o el retorno de la inversión que nos representa.
- Riesgo: Las dificultades o peligros que se relacionan al ítem y que nos motivan a implementarla o a no realizarlo.
- Costo: El valor monetario que nos representa implementar el ítem.
- Recursos: Implica los distintos costos requeridos (skills, infraestructura, licencias, etc.) para la implementación del ítem.
- Conocimiento: La necesidad de obtener información sobre la implementación o sobre el comportamiento del usuario.
- Dependencias: Las necesidades de otros desarrollos que lo afectan.

Que no es un PBI

Tan importante como conocer que es un PBI es conocer que no lo es y por lo tanto no debe formar parte del Product Backlog. Es común en equipos en estadios iniciales, querer incluir toda actividad realizada por los integrantes (desarrolladores, analistas, diseñadores, etc.) como un ítem (en ocasiones reuniones, lectura de documentos, tareas administrativas, etc.) Esta práctica

[1] Fuente: ScrumInc, Jeff Sutherland.

debemos evitarla ya que el backlog del producto no es una lista de tareas o representación del trabajo sino de entregables del producto.

2. Estimar o No Estimar, esa es la cuestión

¿Para qué se estima?

Cada persona puede tener una respuesta propia a esta pregunta... Para poder tomar una decisión; para realizar una acción con la estimación; para controlar la performance; para poder planificar el trabajo, etc. Por supuesto que todo esto depende del contexto y la necesidad. Desde un enfoque ágil debemos preguntarnos el objetivo y el valor de esta actividad y asi actuar en consecuencia.

¿Cuándo se estima?

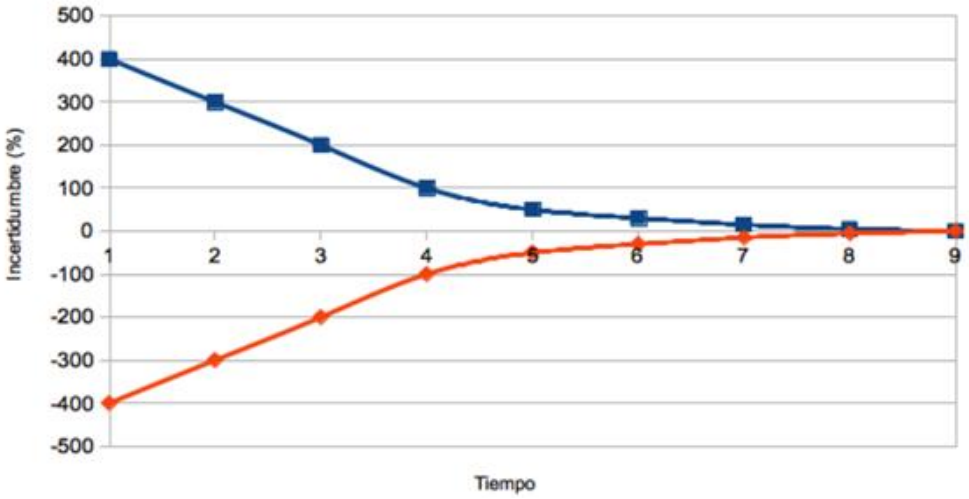
Hay varios tipos de estimaciones con momentos para realizarlas:
Al momento de evaluar potenciales proyectos: necesidad de comparación de oportunidades de valor, ¿Qué proyecto conviene realizar?
Al iniciar un proyecto: necesidad de conocer un roadmap y poder coordinar recursos, personas e hitos.
Al inicio de una etapa: necesidad más detallada de coordinar acciones, recursos de la organización.
Diariamente: necesidad de coordinar tareas y esfuerzos en respuesta a las actividades que cada uno tiene para hoy y cómo se coordinan las dependencias.

Costo de la estimación

La estimación no es algo que agrega valor al entregable que se desarrolla, este es un entregable interno del proyecto. Entonces sabiendo que la estimación tiene un costo, debe realizarse en la forma necesaria para dar valor y sobre todo no tomarlo como una manera de presión al equipo, pensando más en la pregunta ¿Cuándo va a estar? En lugar de: ¿Cuál es la hipotesis que queremos validar? ¿Cuál es el valor al usuario?

Riesgo o Incertidumbre al momento de estimar

El *Cono de la Incertidumbre de Bary Boehm* ***[1],*** muestra la relación entre incertidumbre de una estimación (exactitud) y tiempo:



Cono de Incertidumbre – Barry Boehm

En este diagrama se muestra la dispersión del esfuerzo real al finalizar un proyecto versus la estimación inicial, en función del tiempo. Se puede observar que al inicio de un proyecto (tiempo = 1 en el gráfico), la incertidumbre en cuanto a la estimación va de -400% a 400% del valor estimado. Luego va bajando a lo largo del tiempo del proyecto hasta llegar a la certeza (100%) al finalizar el proyecto. Es entonces fundamental manejar bien las expectativas respecto a las estimaciones dado el nivel de incertidumbre. Una estimación temprana y de alto nivel intrínsecamente tendrá mucha incertidumbre, y una estimación tardía y detallada tendrá menos incertidumbre.

Estimaciones ágiles

Recordando que los marcos ágiles tienen por objetivo los escenarios complejos, se debe entender a las estimaciones **como pronósticos y no compromisos**. Justamente por trabajar en situaciones donde la estimación tiene mucha incertidumbre, se utilizan iteraciones fijas y breves, que entregan valor para el cliente.
El objetivo es obtener conocimiento, aprendizaje y feedback, de esta manera se va logrando en cada iteración (de duración fija) más precisión en los pronósticos. Esto aplica a todo tipo de estimaciones: tiempos de construcción, calidad esperada, satisfacción del cliente y alcance a construir en la iteración. Debemos tener presente que **el motivo de que la estimación no sea exacta no es el uso de un marco de trabajo ágil, sino el escenario complejo** en el cual las estrategias tradicionales de estimación no son certeras o son muy costosas. El escenario no cambia por cambiar la técnica, pero al tratarlo con un enfoque ágil puedo obtener estimaciones útiles en menor tiempo.

Estimación Relativa

Una característica común de las estimaciones en los marcos ágiles es que sean *relativas*. Eso significa que no se busca una estimación universal de la complejidad o del esfuerzo de construcción de un ítem o agregado de valor (funcionalidad o features), sino una comparación de su complejidad con otros ítems conocidos.
En general no se suele apuntar a una estimación en horas del esfuerzo de construcción del ítem, sino poder comparar su complejidad con otros ítems. Un Story Point es una unidad arbitraria pero fija que describe cuánto esfuerzo requiere un ítem para ser entregado al cliente.
Esta medida relativa suele ser útil para pensar en forma más abstracta las estimaciones y compararlas contra elementos conocidos. La escala de *Story Points* es propia de cada equipo, por lo tanto, no se deben comparar entre sí.

Puntos de Historia = Esfuerzo + Riesgo + incertidumbre

Los puntos pueden utilizar valores numéricos (por ejemplo, la serie de Fibonacci, o las potencias de 2, etc) o alguna escala de magnitud que sirva para comparar (ejemplo talles de remera: S, M, L, XL). Se suelen utilizar escalas de crecimiento exponenciales porque a mayor complejidad la incertidumbre suele ser equivalentemente mayor

Tiempo Ideal (Ideal Time)

En algunos contextos, se puede preferir una unidad de estimación más concreta y relacionada directamente con el tiempo de desarrollo. El Tiempo Ideal es una unidad de estimación usada en los equipos de desarrollo más tradicionales. Con el tiempo y el pasar de las iteraciones, contando con datos históricos estimados y reales, suele emerger una relación entre tiempo ideal y tiempo real, a veces del orden de 2 veces.
Esta técnica suele ser contraproducente porque tiende a generar la idea de un compromiso (en particular para el área de management).

3. Técnicas de Estimación y no estimación

Técnicas de estimación ágil

Las siguientes son las más difundidas en los equipos que trabajan con marcos agiles:

1- Planning Poker
Esta técnica permite hacer una estimación inicial en forma rápida y fiable, dado que todos los miembros del equipo comparten sus diferentes informaciones y expresan su opinión sin sentirse condicionados por el resto.
Es una estimación relativa en la que comúnmente se usan Story Points como unidad. Se puede utilizar una baraja de cartas especiales de "planning poker" para cada miembro del equipo (figura a continuación), así como también las ya existentes aplicaciones para celulares, herramientas on-line o cualquier manera de presentar valores numéricos.

[2]

El proceso utilizado sigue los siguientes pasos:

1. El responsable del negocio (o Product Owner) presenta brevemente un ítem a ser estimado.
2. Cada participante realiza su estimación en forma secreta, sin influenciar al resto del equipo, y luego pone su carta elegida boca abajo sobre la mesa.
3. Una vez que todos los integrantes han estimado, se dan vuelta las cartas.
4. Si hay muchas diferencias entre los valores, se discuten estos valores para entender el punto de vista correspondiente.
5. Al finalizar la discusión se levantan las cartas y se vuelve a estimar (pasos 2 y 3), esta vez con mayor información que la que se tenía previamente.

[2] imagen: software testing help.

6. Las rondas siguen hasta que se logra **consenso** en el equipo. No se debe simplemente asumir un promedio, sino que el valor clave es la discusión del disenso para ganar alineamiento.
7. Se repite el proceso para cada ítem a estimar.

Destaco que el puntaje del PBI corresponde a todo el trabajo involucrado para llevarlo a DONE.

2- Estimación por afinidad (Affinity Estimating)
Esta técnica también permite una estimación relativa en grupo con Story Points realizando un mapeo de los mismos. Al inicio de la actividad, el representante del negocio (Product Owner) presenta los ítems a estimar, cada uno en un post-it separado.
Luego se siguen los siguientes pasos:

1. El equipo tiene que ubicar en una pizarra los "post-its" en orden de complejidad: desde los más simples a la izquierda hasta los más complejos a la derecha. Este paso se debe hacer en forma grupal, pero en silencio.
2. Se tienen que agrupar los post-its de acuerdo a valores de complejidad, que empiezan a formar columnas en la pizarra (Story Points). Se suele usar una adaptación de los números de la secuencia de Fibonacci (ver Planning Poker). Este paso se hace en grupo, y se permite hablar para tomar las decisiones correspondientes. Se debería llegar a un consenso sobre la valoración de story points de todos los features.

Esta técnica es muy interesante y se puede trabajar en grupo de 5 a 45 personas, en un tiempo razonable (aprox. 2 horas) para estimar muchos features (+40).

3- Grande/ Desconocido / Pequeño
Es una estimación rápida, en la cual se pueden catalogar a las historias con tres tamaños: grandes, pequeñas o desconocido (se puede también incluir algún valor intermedio). Usualmente se toma el acuerdo de trabajo que solo entrarán en el sprint una vez catalogadas como pequeñas o simplemente tenerlo como referencia. El grupo comienza por debatirlas y en caso de no llegar a un acuerdo se cataloga como desconocido para poder refinarla con el PO y los stakeholders.

4-T-Shirt Sizing
Estimación relativa de alto nivel que utiliza valores tipificados de talles de remeras para indicar la magnitud (S, M, L, XL, XXL, etc). El funcionamiento es similar al poker planning y se suele utilizar para estimar épicas y/o para obtener valores rápidamente.

5- Wideband Delphi
La técnica *Wideband Delphi* es una técnica basada en el consenso para estimar esfuerzo. Fue derivada por Barry Boehm y John Farquhar de la técnica *Delphi,* desarrollada en los años 1950-1960 en *RAND Corporation* como mecanismo de previsiones.
Se siguen los siguientes pasos:

1. *Eligiendo al equipo*: Se selecciona un equipo de estimación y un moderador. El equipo debería ser de 3 a 7 personas y debería incluir representantes de los sectores involucrados en el desarrollo del producto a estimar.
2. *Reunión de Kick-off*: el moderador prepara al equipo y facilita un debate sobre supuestos, genera un WBS [2] y decide de las unidades de estimación.
3. *Preparación Individual:* Luego de la reunión de Kick-off, cada miembro del equipo de estimación estima individualmente cada tarea del WBS, anotando eventuales observaciones y supuestos.
4. *Sesión de Estimación:* el moderador lleva al equipo en pasos iterativos hasta llegar a un consenso sobre las estimaciones. En cada iteración, se muestran los rangos de estimaciones de los participantes, y se revisan las variaciones importantes y eventuales issues para generar nuevas estimaciones. El ciclo se repite hasta llegar a un consenso o hasta que se agote el tiempo asignado.
5. *Cierre:* Se compilan las estimaciones finales y se revisan.

No Estimates

Basado en el concepto Lean que la estimación no da valor intrínseco al cliente, esta idea se focaliza en dedicarle el menor esfuerzo a estimar utilizándolo como proceso de acuerdo. De ser conveniente directamente no se hace. Lo que se busca es debatir donde está la entrega de valor del ítem y acordar en equipo cual es el posible MVP sobre el mismo, siendo el tiempo dedicado el de una mini iteración (un día o a lo sumo 3).
Otra manera posible de trabajarlo es acordar un tamaño relativamente estándar de corte (ejemplo entre lo que el equipo considera 3 y un 5 story points) para hacer un slicing de todas las historias hasta que entren en ese tamaño, de esa manera solo se discute alcance (que deberia ser el foco).

Técnicas tradicionales de estimación:
Mencionamos algunas de las técnicas que pueden usarse en gestiones tradicionales:
1- Por Analogía
En base a experiencias similares pasadas. Para esto es necesario algún tipo de registro o sino basarse en la "memoria" de los participantes.
2- Paramétrica
Este tipo de estimación se realiza cuantificando uno o más parámetros (por ejemplo, cantidad de clases, metros a construir, formularios de diversa complejidad, etc). Se suele construir hojas de cálculos con los parámetros o aplicaciones (ejemplo COCOMO) y estos devuelven la estimación. Es una práctica que se inició muy fuerte en los '80 y aún continúa en algunas organizaciones, pero no han mostrado brindar certidumbre en ambientes complejos.
3- Por 3 valores
Se utilizan los valores Más probables, Optimista, y Pesimista (3 valores) y mediante una fórmula matemática se estima un valor final, un ejemplo de este caso es la estimación que se conoce como PERT (program evaluation and review technique).

En PERT para cada actividad se hacen tres estimaciones: el tiempo pesimista (Tp), el tiempo más probable (Tm) que es la mejor estimación consensuada y el tiempo optimista (To). De estas tres estimaciones, mediante la siguiente fórmula se obtiene el valor de la estimación final:
Te = (To + 4*Tm + Tp) / 6
El estimar un tiempo pesimista y uno optimista da una gama de probables resultados, a su vez el tiempo más probable es nuestra mejor estimación y todo el conjunto otorga una estimación final más refinada.
4- Juicio de Expertos
Otro nombre para “experiencia”, en muchos casos no existe una mejor estimación que aquella que puede realizar una persona o un equipo con conocimientos sobre el tema, muchas veces es cuestionada, aunque existen evidencias que reivindican este tipo de estimaciones.
Top-Down vs Bottom-Up: Define el “modo” en que se hace una estimación, Top-Down (o también de lo general a lo particular) implica trabajar en un nivel general para luego profundizar a detalle, por el contrario, Bottom-Up se centra más en el nivel de detalle y luego “sumariza” hacia una estimación general. Normalmente Bottom-Up tiene mayor precisión, pero demora más tiempo.

4. Validación frecuente con el usuario

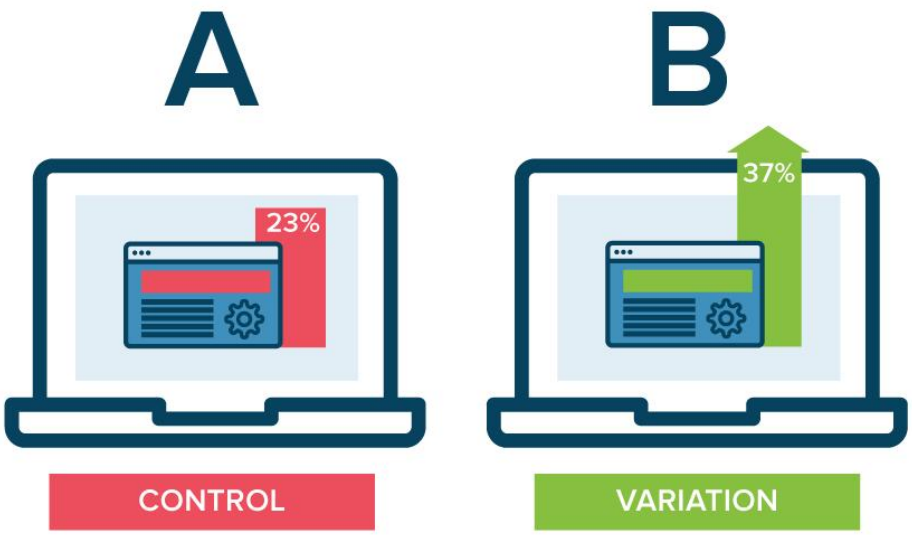
Una de las características más importantes del paradigma ágil es la entrega iterativa e incremental de valor, lo que nos permite obtener un feedback que consideramos crítico para lograr construir un producto satisfactorio. Las técnicas más comunes para obtener feedback y validar hipótesis son:

Revisiones (Reviews)

Este es un evento de Scrum que se acostumbra replicar en otros marcos de trabajo. Consiste en mostrarle al usuario el desarrollo realizado como consecuencia del PBI, para conocer su opinión y permitirle visualizarlo en funcionamiento; así validar como responde su necesidad y actuar en consecuencia. Es por este motivo que se insiste en denominarla revisión (o review) y no demo (por demostración), la cual se entiende como mostrar lo que hicimos y limitarnos a enseñarle a usarlo (o en el mejor de los casos simplemente una aceptación o rechazo) pero sin verlo como una oportunidad de aprender en equipo y escalar en base a eso. Aquellos que practican Scrum suelen hacerlo en fecha fija, pero tanto en este marco de trabajo como en otros, la búsqueda de feedback debe ser una constante,

La técnica de pruebas A/B (A/B testing)

También denominada en algunas bibliografías como split testing o bucket testing, consiste en comparar dos versiones de un mismo producto para determinar cuál tiene un mejor desempeño para ciertas características. Esto permite hacer experimentos donde 2 o más alternativas son otorgadas al azar a los usuarios para analizar los resultados con herramientas analíticas y sacar conclusiones de acuerdo a cómo se comportan. El uso más clásico es en los e-commerce para aumentar la conversión o mejorar alguna otra métrica, las cuales se controlan por ejemplo con Google Analytics y se les puede dar un seguimiento por una cantidad de horas o días y en consecuencia avanzar con el cambio, seguir iterando para encontrar una versión mejor o descartar la idea.



[3]

Esta manera de validar los cambios elimina los supuestos para tomar decisiones de optimización para tomar decisiones orientadas por datos (data-driven) y permite debatir en torno a ellos y asegurarnos que los cambios que se realicen tengan un impacto positivo al expandirse a toda la población.
Esta práctica permite hacer cambios sobre productos ya lanzados al mercado y validar hipótesis sobre el comportamiento de los usuarios. Es muy utilizada particularmente en ecommerce para realizar cambios que lleven a una mejor experiencia usuario y aumentar la conversión (ventas).

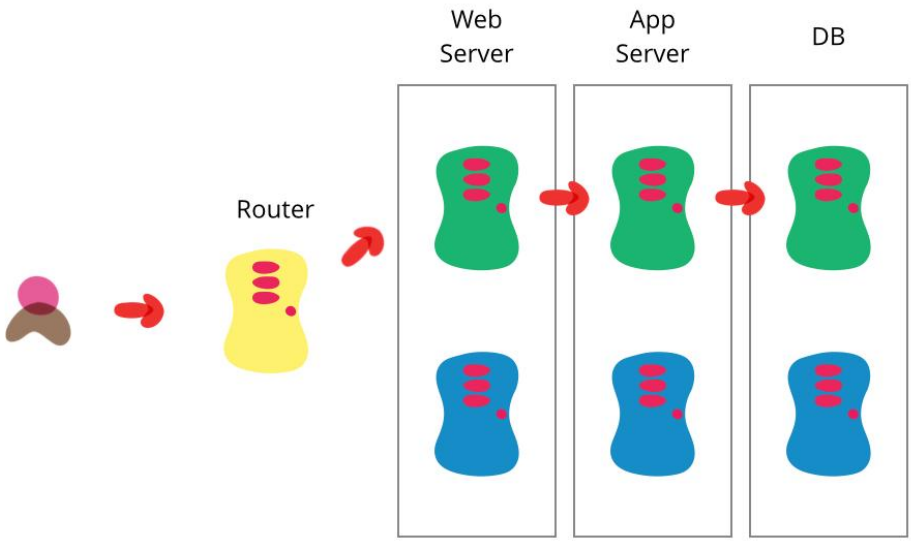
Algunos recomiendan no implementar múltiples A/B test sobre las mismas operaciones, en la práctica esto no es una restricción completamente mientras se pueda identificar con claridad cada resultado.

Implementación Azul-Verde (Blue-Green deployment)

La implementación en ambiente productivo es uno de los grandes hitos por lo que pasa todo nuevo desarrollo. En los casos de estar tocando un producto operativo puede agregar la complejidad de necesitar reducir al mínimo el tiempo en que no está disponible para los usuarios. Considerando que muchas aplicaciones se esperan funcionen 24x7 esto lo vuelve un problema de difícil solución.
La implementación Azul-verde busca solucionar este problema teniendo dos ambientes de las mismas características que intercambian el rol productivo. Para pensarlo con un ejemplo, podria verse que en un momento el llamado ambiente “Azul” está funcionando como producción y la nueva versión en la que estamos trabajando por un nuevo release es el “Verde”, una vez decididos a disponibilizar la versión a los usuarios se realiza el cambio para que las nuevas

[3] Imagen: Optimizely.

solicitudes sean redireccionadas a este ambiente pasando a ser el productivo y el “Azul” pasará a quedar ocioso. Si bien no se reduce a cero el tiempo de indisponibilidad por el tiempo que puede tomar la configuración del redireccionamiento queda extremadamente reducido.
Como puede resultar obvio esto nos deja una posibilidad muy simple para realizar una vuelta atrás (rollback) en caso de problemas con la nueva versión. En algunos casos incluso se dejan espejados momentáneamente para eliminar posibilidad de pérdidas y/o agilizar el rollback.



[4]

Una vez que se está conforme con la nueva versión, se disponibiliza la nueva versión productiva en el otro ambiente y queda disponible para ser utilizado como ambiente pre productivo para poder realizar en el próximo release el mismo proceso de cambio de “Verde” a “Azul”.
La realización de este ciclo de cambio constante entre ambiente productivo y preproductivo tiene como ventaja el mantener aceitado el proceso de recuperación de ambientes en caso de problemas, algo que si bien muchos tienen un ambiente preproductivo suelen no practicar de manera periódica.
La manera de implementarlo varía en las organizaciones (hablando principalmente de productos de software), si bien la implementación más completa es utilizar entornos completamente independientes, (base de datos, hardware, etc.) que se redirecciona desde el DNS. Algunos tienen versiones intermedias como utilizar la misma base de datos (dependiendo de la magnitud del cambio esta opción puede no ser viable en algunas ocasiones), realizar el redireccionamiento desde el webserver, tener instancias en el mismo servidor, etc.
Esta práctica es particularmente utilizada en ambientes de entrega continua (continuous delivery) y con la disponibilidad de la tecnología actual (servidores en la nube, procesos automáticos de deploy, etc) se simplifica su uso.

Implementación de Canario (Canary Deployment)

[4] Fuente de la Imagen: Martin Fowler.

Esta técnica de implementación busca reducir el riesgo en la implementación de nuevas versiones a través de una implementación acotada (un canario que sirva para probar, tal cómo se realizaba en las minas antiguamente) a un pequeño grupo de usuarios que nos sirva para comprobar el correcto funcionamiento para progresivamente incorporar segmentos mayores hasta la implementación total.
Similar a la implementación Azul/Verde se comienza a "deployar" la nueva versión en una parte de la infraestructura en donde no se direccionan usuarios
Este tipo de implementación es muy común (aunque no se la llame de esta manera en todas las organizaciones). La selección de los primeros usuarios de prueba puede ser variada, usualmente se selecciona un grupo de usuarios internos los cuales pueden ser empleados y conocidos de ellos que están al tanto de los cambios y reportan en detalle el comportamiento, otros optan por un público al azar y evaluar los datos que ellos generan, así como también puede optarse por segmentos regionales, demográficos, etc.
Este tipo de implementación facilita el obtener validación en entorno productivo del comportamiento de la nueva versión con métricas y datos de los usuarios, manteniendo la posibilidad de deshacer los cambios (rollback) muy fácilmente.
Un ejemplo muy conocido que utiliza este método de implementación es Facebook, haciendo visibles los cambios a usuarios internos manteniendo activas todas las aplicaciones de medición para evaluar el comportamiento.
A diferencia de la implantación de tipo Azul-Verde, en este caso no requerimos dos ambientes espejados para hacer de producción y preproducción alternativamente, lo cual evita esta redundancia y costo a través del uso de la misma infraestructura.
Puede verse muy parecida a la técnica de validación de hipótesis A/B testing que vimos en la unidad anterior, pero si bien las características técnicas de implementación pueden ser parecidas los objetivos son distintos. Con el A/B testing buscamos validar hipótesis para realizar cambios, mientras que con la implementación de un canario ya hicimos el cambio y estamos probando la estabilidad en producción y poder deshacerlo fácilmente.

Implementación progresiva (Rolling deployment)

En este tipo de implementaciones no tenemos ambientes replicados, sino que el ambiente productivo está compuesto por nodos o instancias más chicas que lo conforman y se reparten la carga (es muy común que empresas tengan estos equipos conectados una balanceador de carga para distribuirla de manera equitativa), entonces al momento de implementar una nueva versión, se lo va realizando progresivamente de un servidor a la vez, pudiendo evaluar el comportamiento a medida que se incorporan nuevos nodos.
Este método permite un rollback muy simple ya que es dejar de usar momentáneamente los nodos con la nueva versión y reinstalar en ellos la anterior. Como limitante pueden estar los cambios que haya en la base de datos, lo cual suele no ser tan flexible, por eso esto suele utilizarse en aplicaciones principalmente de interfaz usuario.

5. Priorización de corto plazo y repriorización

Priorizando los próximos ítems a construir

Ya sea que utilicemos Scrum, Kanban u otro marco ágil de trabajo, todas tienen en mayor o menor medida un tiempo de planificación de corto plazo en la cual se acuerda los ítems más prioritarios y se conversa en detalle para que todo el equipo pueda entenderlos, consultarlos y si ser necesarios desagregarse en ítems más chicos que permitan implementaciones más frecuentes. Pudiendo algunas partes ser menos prioritarias y dejadas para más adelante y haciendo foco en las partes más críticas las planificadas para el corto plazo.
Estos ítems más prioritarios deberán cumplir ciertas condiciones para que puedan ser tomados por el equipo para su construcción y tener un criterio de aceptación claro para que pueda probarse antes de presentarlo al PO una vez construido.
En el caso de Scrum, al inicio de cada sprint tendremos un evento de planificación que nos permitirá tomar los ítems más prioritarios que el equipo considera puede implementar en el periodo para buscar maximizar la entrega de valor con los recursos disponibles. No es inusual que se incluyan algunos ítems de menor prioridad pero que tienen una relación muy fuerte con los ítems a trabajar para tratar de buscar más eficiencia en la construcción.

Repriorización del backlog

Como mencionamos previamente, en el paradigma ágil se da la bienvenida al cambio de acuerdo al aprendizaje que vamos obteniendo y a la evolución de las necesidades del negocio. Por lo tanto, si bien se realiza una inversión considerable en armar un roadmap, plan de releases (y habiendo expresado los beneficios de tenerlos) este no está escrito en piedra y si la dirección actual no nos está llevando al objetivo deseado o simplemente las necesidades del negocio cambian, podemos (o debemos) ajustar las épicas o historias de usuario del backlog acorde a esto. Este ajuste puede implicar cambios en la prioridad, así como también incorporar o eliminar ítems.
En Scrum, la menor planificación que hacemos con el PO es a nivel sprint, en la cual se acuerda entre él y el equipo el objetivo y los ítems a incluir. Idealmente existe el llamado “blindaje del sprint” por el cual no se debería cambiar el alcance durante ese lapso (entre 1 y 4 semanas) obviamente esto no es una restricción total, si la necesidad del negocio o una emergencia así lo amerita puede incorporarse algún ítem (quitando otro/s de tamaño equivalente) y en casos extremos en la cual la planificación realizada ya no aplica, deshacer para barajar y dar de nuevo totalmente.
En Kanban se suele estar trabajando un ítem a la vez, aunque se suele refinar y dejar preparados un mínimo de ítems como buffer para tener un flujo constante, por lo tanto, en caso de cambio de prioridades el PO simplemente reorganiza los próximos ítems para trabajar por el equipo.

Valor del Negocio vs Valor de Conocimiento

En la era de la tecnología y de los productos digitales, el conocimiento se ha convertido en el componente dominante para lograr entregar valor al usuario. Este cambio con relación a épocas anteriores en que los recursos físicos eran el activo principal ha llevado a que la nueva economía sea liderada por las compañías que mejor gestionan el conocimiento, que crean, encuentran y combinan conocimiento en nuevos productos y servicios más rápido que sus competidores.

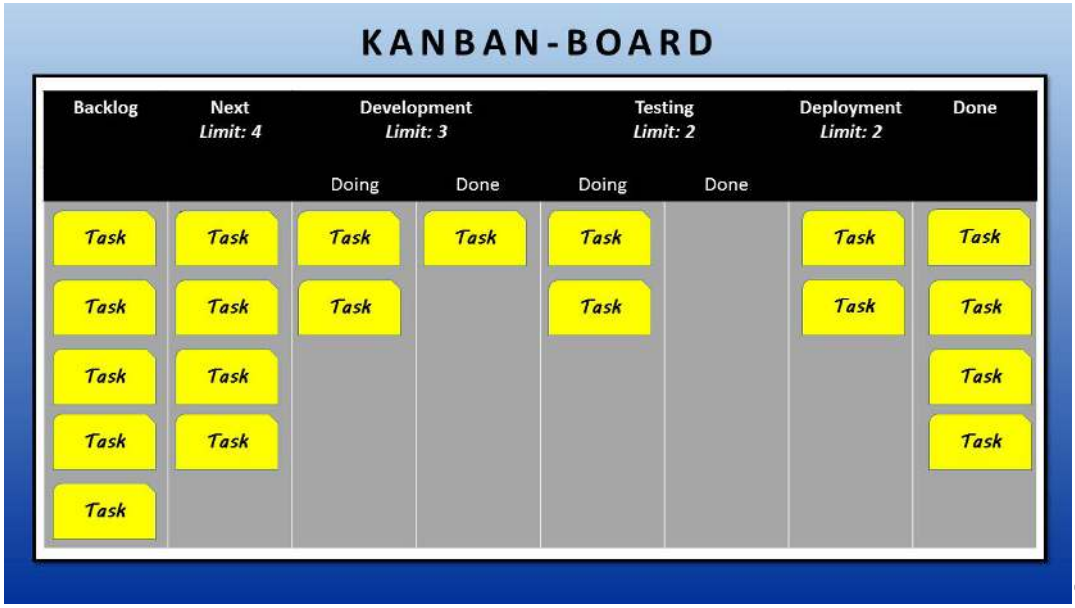
Es por eso que un buen product owner es aquel que lograr entender dónde está el valor del negocio de acuerdo a su conocimiento y a como lo enriquece constantemente con la validación de hipótesis.

6. Manejo del flujo de trabajo con Kanban

El método Kanban nos brinda un entendimiento compartido de cómo es nuestro proceso de trabajo, las reglas que tenemos, cuanto podemos manejar al mismo tiempo y cuan bien podemos entregar el resultado a los clientes. Este entendimiento nos permite trabajar en la mejora del mismo, para ser más predecibles y trabajar a un ritmo sostenible. Este fue diseñado por equipos de Toyota y su nombre se traduce literalmente del japonés "tarjeta" por la entidad física utilizada para representar el trabajo, el cual "viaja" por una serie de estados o etapas representados en columnas.

La tarjeta sirve como mecanismo visual, las cuales se ponen a disposición en una cantidad correspondiente a la capacidad total del sistema. Un nuevo ítem de trabajo puede iniciarse únicamente cuando se dispone de una tarjeta libre. Esta tarjeta libre se adjunta al ítem de trabajo para que pueda avanzar a través del sistema. Cuando no hay más tarjetas libres, no se pueden iniciar nuevos trabajos.

Este mecanismo es conocido como sistema "Pull" (arrastre), porque un nuevo trabajo es introducido en el sistema ("Pulled") únicamente cuando hay disponibilidad para procesarlo, en lugar de ser introducido "Pushed" (empujado) en el sistema[5]



[6]

Valores de Kanban

El método Kanban tiene una serie de valores que son su guía, no solo para el éxito de la empresa, sino para darle un fundamento al trabajo de las personas:

[5] David Anderson
[6] Digital Guide

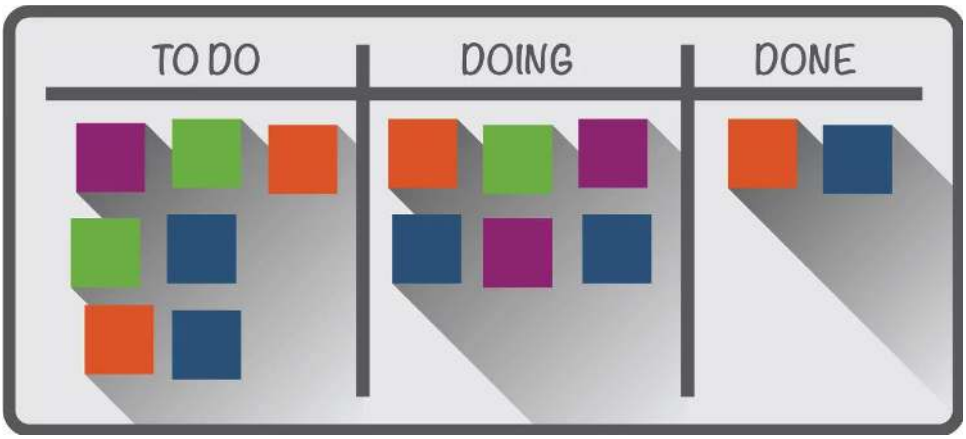
- Transparencia: Compartir abiertamente la formación, esto facilita la mejora y aprendizaje.
- Equilibrio: Lograr que la demanda, capacidad y demás puntos involucrados estén equilibrados para evitar el colapso del sistema por la degradación causada.
- Colaboración: El método en si fue concebido para facilitarle a las personas trabajar juntas.
- Foco en el cliente: El outcome de cada sistema Kanban es la entrega de valor al cliente.
- Flujo: La realización del trabajo está dada por un flujo de valor.
- Liderazgo: No en el sentido jerárquico sino en la búsqueda de valor y mejora desde todos los niveles.
- Entendimiento: Conocerse y aprender, en este método la mejora continua esta embebida.
- Acuerdo: El consenso y compromiso para ir juntos al objetivo.
- Respeto: Valorar y empatizar con las personas.

Las tres reglas de Kanban

Este marco de trabajo es realmente útil en todos los rubros y hasta a nivel personal, por lo cual es altamente recomendable para organizar el trabajo y visualizar las optimizaciones. Si buscamos hacer foco en el desarrollo de software, estos mecanismos se tuvieron que adaptar a la naturaleza virtual del software y el resultado de esta adaptación puede resumirse en las siguientes reglas:

1. Visualizar el proceso

Con esta regla se busca mostrar los ítems y el proceso de trabajo en un tablero (puede ser físico o digital). El flujo es expresado en columnas por cada estadio o etapa, esto debemos relevarlo para conocer cómo funciona en la realidad, **Kanban promueve mostrar el proceso actual tal cual es** sobre intentar redefinirlo desde el inicio. En particular esta primera regla busca brindar visibilidad a la organización para entender el proceso actual de trabajo y los problemas que pueden surgir. También permite una mejor comunicación a la hora de definir cambios en la forma de trabajar. No hay manera específica de dibujar este tablero, depende de cada contexto y comportamiento, aunque podemos ver un diseño elaborado en la imagen anterior, o un estilo más básico a continuación:



2. Limitar el trabajo en curso

En un sistema Kanban se busca limitar el trabajo en curso (también llamado work in progress –WIP-) para asegurar un flujo de producción continuo y optimizado, sin espera o sobreproducción que lleven a trabajo de inventario o sobre stock (muda, como lo vimos en Lean). Se logra esta limitación con dos mecanismos básicos:

- Se utiliza un número fijo y limitado de tarjetas Kanban.
- El proceso posterior solamente retira partes terminadas del proceso anterior cuando las necesita.

3. Optimizar el flujo

El flujo del sistema debe maximizar la entrega de valor, minimizar los tiempos de entrega y ser tan fluido como sea posible. Llevar el control del trabajo en curso y visualizar el proceso es por lejos una herramienta poderosísima que cualquier equipo puede aprovechar.

4. Explicitar las reglas

Las políticas son una manera de definir el proceso más allá del flujo per se. El proceso expresado por el flujo y las reglas permite tener las características emergentes que pueden ser ajustadas por medio de experimentos. Estas políticas deben ser pocas, concretas, visibles para todos y siempre aplicarse.

5. Canales de retroalimentación (feedback)

Los canales de retroalimentación son una parte esencial de cualquier proceso controlado y claves para la mejora continua. Si bien no hay timebox en kanban, si se espera tener una cadencia para estos procesos.

7. La Deuda Técnica, sus consecuencias y cómo gestionarla

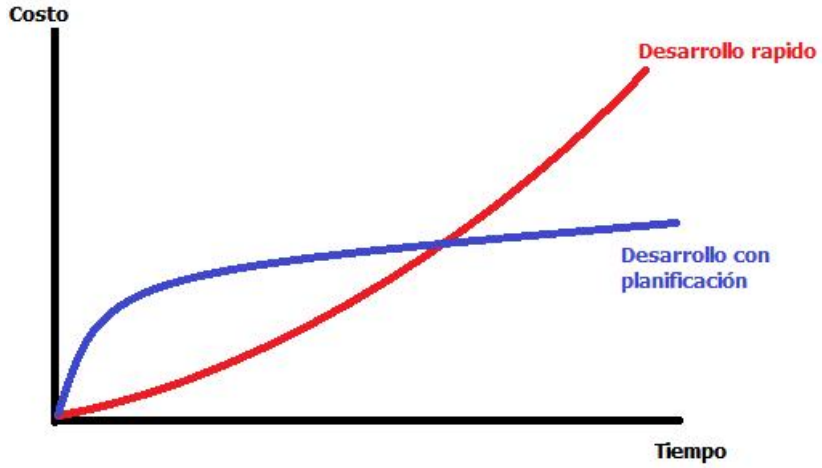
¿Qué es la deuda técnica?

Las primeras menciones a este término suelen atribuirse a Ward Cunningham, uno de los autores del Manifiesto Ágil, comparando las decisiones de realizar pobres implementaciones en pos de acelerar los tiempos de implementación (el clásico "lo atamos con alambre" o "harcodeamos") con estar emitiendo una deuda a futuro. Lo cual no es inherentemente malo mientras que se paguen los intereses refactorizando el código o haciendo las correcciones requeridas para una implementación de calidad[7].

El problema, como nos podemos imaginar, viene cuando esta deuda no es pagada y sus "intereses" comienzan a acumularse y deberemos "pagarlo" al evolucionar el producto con mayores costos. Estos se entienden como mayor dificultad para realizar cambios, mayor presencia de bugs en producción, desmotivación en el equipo, etc.

El equivalente a una "crisis de deuda" se da cuando el sistema ya no puede seguir modificándose por tener un código tan incomprensible que cada cambio es extremadamente costoso y causante de problemas.

Vale aclarar el alcance de lo llamado como deuda técnica no incluye los bugs (o defectos productivos) en sí mismos, ni la falta de un proceso de software, así como tampoco las funcionalidades no deseadas por el usuario.



Tipos de deuda técnica

[7] OOPSLA conference.

El cuadrante desarrollado por Martin Fowler nos permite categorizar la deuda técnica utilizando un gráfico de dos dimensiones, por un lado, si es deliberada o no deliberada y prudente o irresponsable obteniendo cuatro tipos diferenciados[8]:

Prudente y deliberada
En este caso el motivo se debe a una búsqueda de competitividad y lograr maximizar el ROI del producto. Esto puede deberse a querer adelantarse a un competidos, a la necesidad (legal o contractual) de cumplir con una fecha o un periodo comercial de gran importancia (las ventas de un hot-sale o día de celebración).

Imprudente y deliberada
Caer en este cuadrante se debe a una mala gestión, ya sea propia del proyecto o de la organización que empuja al cumplimiento de fechas no realistas en pos de objetivos que no son realmente la necesidad prioritaria del producto. Estas malas decisiones de la gestión llevan a un producto y un proceso de construcción no sustentable.

Imprudente e inadvertida
Esto se suele dar por bajo seniority o experiencia en el equipo, que puede llevar a tomar pobres decisiones, causantes de incrementar la deuda técnica sin saberlo. Es por esto que se debe contar con un proceso que lo compense (revisión de código, programación de a pares, plan de capacitación, etc) y herramientas que acompañen (para integración continua, chequeo de código, etc)

Prudente e inadvertida
Este tipo de deuda es bastante común y equipos maduros lo deben revisar periódicamente para reducirlo y aprender a evitarlo a futuro. Puede darse por cambios en los objetivos, el haber avanzado por otro camino al planificado, etc.

Prácticas para evitarla
Entre las prácticas más comunes que se utilizan mencionamos las siguientes[9]:
1- Trazar una línea
Esto implica un cambio cultural, definiendo que no se acepta más deuda técnica salvo las excepciones por decisiones estratégicas.
2- Una fuerte definición de hecho (definition of done). En este aspecto los equipos maduros suelen tener (pero no limitarse) a:

- *Revisión de código*
- *Pruebas unitarias documentadas*
- *La no aceptación de bugs pendientes*
- *Automatización de pruebas de regresión.*

[8] Nita Andansare, Agile Record.
[9] Nita Andansare, Agile Record.

- *Utilización de herramientas de control de código.*
- *Mayor trabajo en equipo para el diseño y compartir conocimiento.*

3- Desarrollo orientado a Pruebas (Test-Driven Development)
Es un proceso de desarrollo en el cual se escribe un caso de prueba, esta falla, se escribe el código requerido para que pase. Este proceso continúa hasta concluir con todos los casos de prueba validados. Se incluye una refactorización cuando se necesita para mantener "saludable" el código.

Prácticas para gestionar la deuda

Cada organización tiene sus propias políticas en lo que respecta al tratamiento de la deuda técnica, tal como puede tenerlas con la deuda financiera. Se puede estar en un extremo del espectro no aceptando tomar deuda (un enfoque más tecnócrata), otros la consideran aceptable mientras que se haga de manera consciente realizando revisiones frecuentes para reducirla y recuperar la velocidad (más pragmático) y otros pueden simplemente hacer foco en ignorarla enfocándose solo en el desarrollo inmediato pudiendo obtener algunos resultados rápidos en el corto plazo a costas del colapso del sistema en el largo.
La deuda técnica es un problema que suele salirse completamente de control en los sistemas legacy, llegando a situaciones donde el consumo de recursos es a causa mayoritariamente de dar soporte (pagar los intereses de la deuda) no pudiendo evolucionar (al menos de manera importante). La solución más común que suele darse en muchas organizaciones (que creo todos hemos visto) es que luego de años de una mala performance y equipos sin motivación se reemplaza el sistema desarrollando uno nuevo.
Esto no necesariamente debe ser así, es posible controlar la deuda técnica a través de técnicas tales como desarrollo orientado a pruebas. Obviamente se debe tener paciencia y disciplina para poder obtener resultados y una vez controlado lograr mejoras en la calidad y productividad.
Una práctica que suele dar buenos resultados es incluir la deuda técnica como ítems del backlog, los cuales pueden simplemente competir en prioridad con los demás, incluir cierta cantidad fija por iteración (se puede utilizar su tamaño en story points), aprovechar momentos de menor carga en desarrollo, etc.

¿Se puede aceptar deuda técnica?

Posiblemente no haya una respuesta absoluta a esa pregunta. Como estuvimos viendo, al igual que en la deuda financiera, si esta se utiliza como una inversión nos puede ayudar a obtener una ganancia que permita pagar y seguir creciendo. Lo más importante que debemos recordar es:

- Nunca ignorarla,
- Evitarla salvo que los beneficios excedan cómodamente su costo,
- Tener el compromiso de su "pago" o solución, tanto del equipo de elaboración como del negocio.

Bibliografía utilizada y sugerida

Libros y otros manuscritos

Carlucci, Daniela, Schiuma; Gianni: The knowledge value chain: how intellectual capital impacts on business performance, 2014.

Cohn, Mike: Agile Estimating and Planning, Prentice Hall, 2005.

Humble, Jez, Molesky Joanne, O'Reilly, Barry: Lean Enterprise: How High-Performance Organizations Innovate at Scale, O'Really, 2015.

Patton, Jeff: User Story Mapping, O'Really, 2015.

Pitchler, Roman: Strategize: Product Strategy and Product Roadmap Practices for the Digital Age; 2016.

v7

Lo que vimos:

Es esta unidad profundizamos en las responsabilidades más cotidianas del product owner, como es el mantenimiento de un backlog y cómo actuar ante los cambios.

Lo que viene:

En la próxima unidad nos enfocaremos en el uso de las historias de usuario como herramienta para poblar un backlog efectivo.