AGILE OPEN CAMP
BARILOCHE 2015

COMPILADO POR NICOLAS PAEZ

agiles Argentina

HERRAMIENTAS AGILES

TÉCNICAS Y PATRONES PARA EQUIPOS Y PROFESIONALES

Tabla de contenido

Introducción 0
San Saru: selección natural de equipos 1
Integración de prácticas para lanzamiento de proyectos de software 2
Value Stream Mapping 3
Elaboración de historias de usuario centradas en comportamiento 4
Técnicas de OnBoarding para la gestión de conocimiento 5
SHU-HA-RI: Un Modelo de Aprendizaje 6
Continuous Discovery: Validación de ideas para el Backlog 7
Prácticas eficaces para aplicar en Reuniones (In)eficientes 8
Introducción a Visual Management 9
Revisión Triangular de Documentos 10
SEF: Sesión Exprés de feedback 11
Improvement Kata 12
Guardián de un equipo con múltiples asignaciones 13
Coding Dojo: técnica de entrenamiento 14
Automatización a través de Git hooks 15
Versionado de código, configuración y ambientes 16
Referencias 17

Introducción

Por Nicolás Paez, @inicopaez

Sobre este libro

Durante la primera edición del Agile Open Camp celebrada en 2015, escribimos un libro que titulamos *Experiencias Ágiles: Relatos de experiencias del uso de métodos ágiles en Argentina [Libro 2015]*.

Cuando se anunció el Agile Open Camp 2016 y se estableció el sistema de inscripción basado en postulaciones y propuestas (ver capítulo *San Saru: selección natural de equipos*), envié inmediatamente mi postulación proponiendo la escritura de un segundo libro que continuase la tradición establecida en el primer AOC. Este libro es la materialización de esa propuesta.

A diferencia del primer libro que reunía experiencias, este segundo libro reúne un conjunto de técnicas. Considero que esta es una evolución natural, o sea, en primera instancia uno enfrenta diversas situaciones las cuales permiten ir ganando experiencia. Luego, a partir del análisis de esas experiencias es posible descubrir patrones y generalizarlos en técnicas reutilizables en situaciones similares.

Los capítulos han sido escritos de forma totalmente independiente por distintos autores siguiendo una mínima estructura dada por 4 secciones:

- Palabras clave
- Intención
- Motivación
- Descripción

Dentro de esta estructura de primer nivel, cada autor tuvo la libertad de agregar sub secciones acorde a la técnica presentada.

Algunas de las técnicas presentadas son creaciones originales de los autores de este libro, mientras que otras son creaciones de terceros que han sido descriptas/refinadas por los autores aquí presentes.

El primer capítulo del libro no tiene relación directa con el desarrollo de software, sino que describe el método utilizado para la selección de participantes del AOC 2016.

Orden de lectura

Los capítulos pueden ser leídos en cualquier orden ya que como se mencionó anteriormente, son totalmente independientes y su orden de aparición no sigue ningún criterio particular. El libro ofrece una interesante diversidad de técnicas, algunas de índole técnica y otras más orientadas a cuestiones de organización.

Créditos

Autores: Thomas Wallet, Tomás Christie, Pablo Lischinsky, Pablo Tortorella, Juan Daza Arévalo, Vanesa Savino, Omar Fernández, Alejandro Faguaga, Soledad Pinter, Virginia Brassesco, Natalia Baeza, Leonardo Barrientos Silva, Hiroshi Hiromoto, Fernando Di Bartolo, Nicolás Paez

Revisión: Natalia Baeza, Virginia Brassesco y Nicolás Paez

Figuras: Juan Daza Arévalo

Arte de tapa: Mauro Strione

Foto de tapa: Diego Gómez

Idea y coordinación: Nicolás Paez

Agradecimientos

A Mariano Correa, Rosemery Restrepo, y Deiby Ordóñez Díaz por su colaboración en el proceso de escritura y revisión.

A los sponsors del Agile Open Camp 2016, su apoyo fue fundamental para la realización del evento:

- Agilar
- FDV Solutions
- Patagonian Tech
- INVAP
- Kleer
- PetroVR
- Micracel
- Kinetica

San Saru: selección natural de equipos

Por Thomas Wallet, @WalletThomas y Tomás Christie, @tommychristie

Palabras clave

auto-organización, descentralización, propuesta de valor, selección, san saru

Intención

En abril del 2015 se realizó el primer *Agile Open Camp* (AOC), cuyos tres fundadores (Mauro Strione, Tomás Christie y Thomas Wallet) no tenían experiencia alguna en organización de eventos.

Si bien fueron rápidamente respaldados por otros organizadores más experimentados, se pusieron a la venta 50 entradas con poca expectativa de venderlas. A pesar del período de fiestas de fin de año y vacaciones de verano, a las 3 semanas se habían agotado todas las entradas. Luego, se habilitó la venta de 25 entradas más al doble de precio, que se agotaron en menos de una semana. Cuando finalmente se cerraron las inscripciones, quedaron más de 30 personas en lista de espera.

Para la segunda edición del AOC se amplió la capacidad a 100 personas y desde las primeras charlas de organización, surgió la preocupación de evitar una selección de participantes basada en una carrera contra-reloj por comprar entradas.

Durante otro evento de la comunidad ágil [Ágiles 2015], se formó un pequeño grupo luego de las actividades del día, que debatió apasionadamente para revisar ideas previas y explorar alternativas superadoras al mecanismo anterior de inscripción. Tomando como base una idea original de Mauro Strione de selección distribuida logramos entre todos asentar las bases del mecanismo de inscripción *San Saru,* que se describe más adelante.

Motivación

Partiendo del supuesto que la demanda de entradas para el AOC iba a superar ampliamente la cantidad disponible, se diseñó el mecanismo de inscripción llamado *San Saru* con el objetivo de cumplir con las siguientes características:

- **Descentralizado,** para que la selección no dependa de un comité restringido.

- **Activo,** para que los interesados tengan que exponer sus motivaciones.
- **Abierto,** para que los criterios de selección no sean pre-definidos ni cerrados.
- **Asincrónico,** para poder ejecutar en paralelo sus distintas etapas.

Descripción

Metafora San Saru

San Saru es un término japonés que se puede traducir como “los tres monos sabios”. Entre las múltiples explicaciones existentes del concepto, destacamos la siguiente [Román 2006]:

Cuenta la leyenda que tres monos fueron enviados por los dioses para delatar y castigar las malas acciones de los humanos:

- *Kikazaru,* el mono sordo, era el encargado de utilizar el sentido de la vista para observar a quienes realizaban malas acciones y comunicárselo a Mizaru, mediante la voz.
- *Mizaru,* el mono ciego, no necesitaba su sentido de la vista, puesto que tan sólo se encargaba de transmitir al tercer mono, Iwazaru, los mensajes que le pasaba Kikazaru.
- *Iwazaru,* el mono mudo, escuchaba los mensajes transmitidos por Mizaru para decidir la pena de los dioses que le caería al desafortunado humano que lo mereciese y observar que se cumpliese.

Colocados los tres monos según sus habilidades y limitaciones, obtenemos un mono que ve, otro que escucha y otro que habla. Los monos, juntos y organizados, pueden alcanzar metas que no lograrían por separado. Si bien existen varias posibilidades de colocar a los tres monos, todas ellas son situaciones de comunicación fallida o de colaboración imposible, excepto una: *Kikazaru* (sordo) > *Mizaru* (ciego) > *Iwazaru* (mudo).

La metáfora San Saru, con su fundamento de integración por afinidad y complementariedad, inspiró el mecanismo de inscripción del AOC, con sus dos etapas: **postulación** y **selección.**

La postulación

Se solicita a los interesados en participar del AOC postularse, contestando las siguientes preguntas:

- **¿Qué puedo aportar yo al evento?** ¿Por qué el evento va a ser mejor con mi participación?
- **¿Qué espero recibir del evento?** ¿En qué creo que me va a ayudar?
- **¿Quién soy?** ¿Cuál es mi ocupación, formación, empresa/institución, pasión, etc.?

Todas las postulaciones son públicas.

La selección

El **San Saru Primario,** compuesto por los tres fundadores del AOC inicia la selección de los participantes.

Cada uno de los tres miembros del **San Saru Primario** elige a dos personas postuladas para formar un nuevo **San Saru Secundario,** con las siguientes restricciones:

- **Mis propios criterios:** cada persona elige dos participantes evaluando las postulaciones para que el evento sea el mejor posible de acuerdo a sus propios criterios.
- **Una persona desconocida:** se sugiere elegir por lo menos una postulación de una persona que no sea compañero de trabajo, de estudio, o cercano por otra vía, para evitar el sesgo de pertenencia.

Se comunica a las nuevas personas elegidas su selección, lo cual habilita, por un lado su inscripción al evento, y por otro lado su responsabilidad de generar un nuevo **San Saru Terciario,** eligiendo dos personas cada una dentro de las postulaciones restantes.

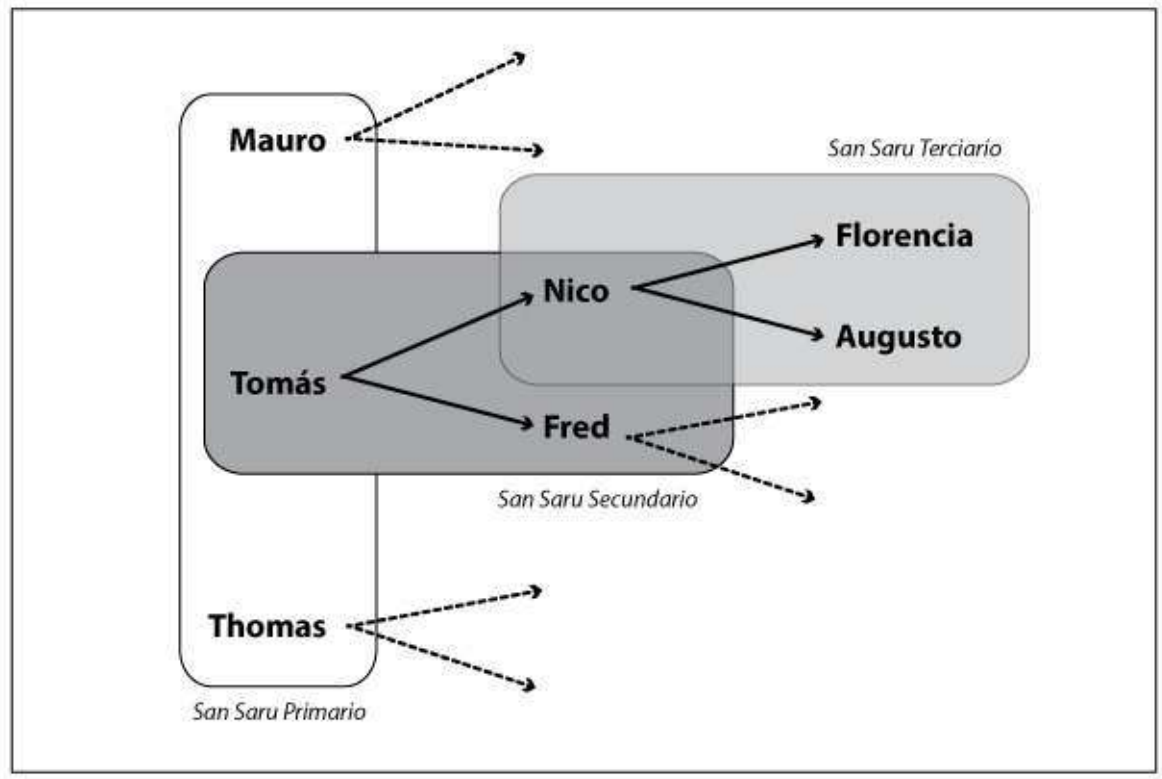

Figura 1.1. *Ejemplos de San Saru Primario, Secundario y Terciario.*

Se repite el ciclo hasta generar todos los San Saru necesarios para cubrir las vacantes disponibles con las personas seleccionadas. Si bien la etapa de postulación se inicia un tiempo antes que la etapa de selección, una vez iniciada la etapa de selección las dos se

desarrollan en paralelo hasta agotar las vacantes disponibles.

Resultados

El primer experimento de inscripcion San Saru para el *Agile Open Camp 2016* [AOC 2016] arrojó los siguientes resultados:

- Se inició la etapa de postulación el 1/12/2015, recibiendo 119 postulaciones hasta el 25/2/2016, de las todas fueron seleccionadas.
- Se inició la etapa de selección el 23/12/2015, generando 119 selecciones hasta el 26/2/2016, de las cuales 24 personas no se inscribieron o cancelaron su inscripción por motivos varios.

Conclusión

(Des)control

Una vez armadas las herramientas de soporte, hechas las explicaciones correspondientes [AOC 2016], y generadas las selecciones del San Saru Primario, el mecanismo San Saru empezó a funcionar por sí solo, sin necesidad de intervención por parte de la organización sobre sus resultados.

Se necesitaron algunas comunicaciones de seguimiento para evitar que los seleccionados demoren su inscripción al evento y/o su elección de postulantes. También se centralizaron en un grupo de 6 personas las tareas de comunicación de los pasos a seguir para la inscripción al evento.

En varias oportunidades surgió la tentación de intervenir para frenar estas comunicaciones para demorar las selecciones posteriores (por ejemplo cuando quedaban pocas postulaciones disponibles para elegir), pero al final no hubo grandes intervenciones por parte de la organización y se dejó fluir solo el mecanismo San Saru.

Lo que aprendimos

Se diseñó el experimento de inscripciones San Saru para el AOC suponiendo que las entradas habilitadas eran pocas para el público interesado, lo cual no se cumplió: todas las postulaciones fueron elegidas, y sobraron 5 vacantes.

Identificamos algunas oportunidades de mejora:

- Dejar más tiempo entre el inicio de las postulaciones y el inicio de las selecciones, para lograr mayor volumen de postulaciones.
- Evitar los meses de diciembre y enero, en los cuales suele haber mucha carga de

actividad laboral y personal, dificultando el esfuerzo requerido para las postulaciones.
- Facilitar las postulaciones de personas con poca experiencia en este tipo de evento o metodologías, ya que expresar el valor que uno puede aportar al evento puede generar cierta inhibición.

El experimento también permitió refinar sobre la marcha varios aspectos logísticos y de comunicación del mismo, lo cual seguramente habilitará futuros usos más fluidos.

A pesar de estas limitaciones, el balance del experimento es positivo. Se destaca en particular lo bien que funcionó la auto-organización, la diversidad y riqueza de los criterios de selección que aportó cada uno, y la sana reflexión de cada postulante para definir sus motivaciones de participación.

Quizás se pueda extrapolar el mecanismo de selección San Saru a otros dominios para los cuales sea útil tener un mecanismo de selección descentralizado y auto-organizado basado en propuestas de valor.

Integración de prácticas para lanzamiento de proyectos de software

Por Pablo Lischinsky, @pablolis

Palabras clave

Acta de proyecto, visión del producto, empoderamiento de equipos, alineamiento, gestión de producto.

Intención

Para dar inicio a un proyecto de desarrollo de software (o relanzar uno en marcha), existen varias técnicas que aportan valor. Sin embargo, es conveniente combinarlas para tener una mejor visión compartida del proyecto y preparar adecuadamente al **equipo de *delivery*** (más que desarrollo) y demás participantes e interesados en el desafío a enfrentar

Motivación

En una empresa de desarrollo de software surgió la necesidad de relanzar un proyecto relacionado a un producto *core* en el área de gestión, específicamente liquidación de haberes. Se trataba de un producto *legacy* con más de 10 años en el mercado que evolucionó en distintas direcciones a necesidad de cada cliente. Al plantearse relanzar el producto con versiones web más modernas y una versión *mobile*, fue necesario re-delinear el alcance y la visión del proyecto. No había un equipo unificado ni de soporte ni de desarrollo, había pocas personas con conocimiento profundo del producto y había poca documentación del mismo.

Descripción

Cuando se inicia un proyecto es conveniente que el *Product Owner* o *Agile Product Manager*, junto al equipo de *delivery*, la gerencia y otros involucrados como clientes y responsables financieros, **creen y mantengan actualizada** la visión, el propósito, la hoja de ruta y el alcance del producto, es decir, contextualizar el mismo. El objetivo es mantener al

equipo alineado y con foco para evitar malentendidos y retrabajo, buscando siempre la **esencia: la simplicidad inherente** en cada proceso o sistema que aporta mayor valor al negocio.

Técnicas y herramientas

Las técnicas Agile Inception Deck [Rasmusson 2010a] y [Rasmusson 2010b] y User Story Mapping [Patton 2014] y [Buonamico 2013], fueron desarrolladas para aplicarse en este contexto de lanzamiento de proyectos de software.

Agile Inception Deck

Consiste en diez actividades de alto nivel realizadas en forma de taller participativo para intentar contextualizar el proyecto:

- Cinco para crear y consensuar una visión de alto nivel:
  - Preguntar por qué estamos aquí.
  - Crear un *elevator* pitch.
  - Diseñar una caja de producto.
  - Crear la lista de los NO.
  - Conocer a los vecinos.
- Cinco para consensuar cómo llevarlo a tierra:
  - Muestra la solución.
  - Qué te quita el sueño en la noche.
  - Determina su tamaño.
  - Lo que va a dar.
  - Qué se va a tomar.

User Story Mapping

Es una herramienta que permite generar una representación visual de un sistema completo. Ofrece una vista general de todas las funcionalidades que lo componen. Permite identificar y planificar *Releases* cortando en rebanadas (*slicing*) y visualizando cómo se distribuyen las funcionalidades de acuerdo a las diferentes áreas del sistema.

También puede verse como una forma de reorganizar el *Product Backlog* en dos dimensiones, una dimensión para el tiempo y otra dimensión para las funcionalidades.

Actividades a realizar:

- Identificar los procesos de negocio, su desglose en actividades y secuencia.
- Identificar los usuarios y las actividades que realizan.

- Identificar las funcionalidades del software a construir.
- Representarlos visualmente como un mapa con *notas autoadhesivas*.
- Priorizar las actividades por valor de negocio,
- Agruparlas en *Releases*.

La idea es **descubrir juntos el contexto del proyecto** realizando estas actividades en forma de taller: es una **búsqueda activa de todos los involucrados**, para evitar cometer errores en la toma de decisiones basadas en asunciones falsas.

Otras técnicas

Sin embargo, es conveniente combinarlas con otras técnicas, como por ejemplo, el *Product Vision Board* [Pichler 2013] y [Hiromoto 2013] y otras actividades de planificación y preparación, ver [Larsen 2013], para tener una mejor visión y para mejor preparar al equipo de *delivery* y demás interesados.

¿Cómo hacerlo?

Planificar y preparar reuniones de trabajo con todos los involucrados, incluyendo al equipo de delivery, al promotor del proyecto, a los analistas de negocio, soporte y otros interesados. No descuidar los detalles: espacio adecuado (amplio, luminoso, silencioso), materiales, refrigerios, sin interrupciones, etc.

Desarrollar juntos las distintas actividades relacionadas a establecer un acta del proyecto (agile chartering), todo esto en forma de talleres para apoyar el lanzamiento del proyecto (liftoff). ¡Aquí hay mucho espacio para la facilitación, participación con creatividad y diversión! ¡La idea es crear y compartir conocimiento mediante el ensayo y el error!

¿Qué se logra?

Algunos logros con estas actividades de lanzamiento de proyectos son:

- Que los involucrados se conozcan y den a conocer al grupo su experiencia, destrezas y expectativas. Si los participantes no se conocen es recomendable utilizar dinámicas para romper el hielo y ayudar a crear un ambiente de trabajo en equipo.
- Que todos los involucrados **participen en crear una visión compartida del producto** y una hoja de ruta del mismo, usando una o varias de las técnicas mencionadas:
  - Una *Agile Inception Deck.*
  - Un tablero de Visión del Producto.
  - Un *User Story Mapping* que da origen a un plan de entregas y a una primera versión de alto nivel del *backlog.*
- Se fortalece al equipo y se crea un mayor sentido de propósito y empoderamiento.

- Se definen los roles y las responsabilidades.
- Se establecen acuerdos de trabajo, tales como canales de comunicación y almacenamiento de la documentación, frecuencia y lugar de las reuniones y ceremonias, horario de trabajo, herramientas a utilizar, etc.
- Se aclaran los recursos requeridos y asignados al proyecto.
- Se estudian las dependencias con otros proyectos.
- Se analizan y priorizan los riesgos y se establece una estrategia para irlas despejando en el tiempo.
- Se crea documentación mínima, preferiblemente usando técnicas de facilitación gráfica, que estará visible para todos los interesados y que servirá para el proceso de inserción de nuevos miembros al equipo (*onboarding*).
- Se nivelan las destrezas de los miembros del equipo en aspectos de metodológicos tales como principios y valores del agilismo y del proceso a utilizar, por ejemplo Scrum.
- Se definen y programan otros talleres para nivelar o desarrollar otras competencias técnicas.
- Se comienza a **crear un ambiente de alta colaboración, transparencia, compromiso y seguridad.**
- **¡Celebrar** juntos el trabajo realizado!

Resumen

Algunas de las actividades propuestas pueden solaparse y en una primera impresión, confundir a quienes las usan. Sin embargo, ayudan a reforzar el aprendizaje y el entendimiento compartido, así como a **validar** distintos aspectos del proyecto desde diferentes puntos de vista.

Las herramientas a utilizar dependen también del contexto de negocio y de la etapa en que se encuentra el producto dentro de su ciclo de vida.

Estas actividades de lanzamiento aportan valor también para aquellos proyectos en ejecución: en contextos complejos e inciertos (como lo son casi todos los proyectos de software) de vez en cuando es necesario hacer un alto para **reflexionar** y responderse: ¿quiénes somos? ¿qué hemos hecho?, ¿dónde estamos hoy?, ¿hacia dónde vamos?, ¿cuáles son los desafíos, restricciones, riesgos y oportunidades?

[Larsen 2013] nos provee una visión más amplia y de cierta forma integradora que la que provee cada herramienta por separado, centrada en el equipo de *delivery* y demás involucrados y que nos ayuda a responder, de forma activa, estas interrogantes.

Value Stream Mapping

Por Pablo Tortorella, @pablitux y Pablo Lischinsky, @pablolis

Palabras clave

Lean, procesos, optimización, visualización, eficiencia

Intención

Si un equipo tiene un objetivo claro y todos sus integrantes quieren alcanzarlo, es probable que quieran **optimizar** su forma de trabajar para lograr ese cometido. Si el objetivo está relacionado con la realización de tareas o actividades que se repiten, estarán frente a uno o más **procesos**. En ese contexto de optimización de procesos, existe una técnica que permite visualizar, analizar y trabajar en su optimización. Su nombre original es en inglés: ***Value Stream Mapping*** [Martin 2013]. Se trata de una técnica que se suele llevar adelante en reuniones grupales (aunque también podría ser realizada individualmente) en las que se trabaja con elementos y características del **proceso que se desea optimizar**.

Esta técnica es una forma de bajar a tierra los principios Lean, oriundos del mundo de la manufactura y muy utilizados también más tarde en industrias de diversa índole. En Lean, todo el esfuerzo se dedica a mejorar los procesos de forma tal que se logre minimizar el tiempo entre que un cliente realiza una solicitud y esa solicitud se transforma en el producto o servicio requerido. Suele utilizarse tanto en procesos producción como en servicios.

Un **ejemplo** que puede ilustrar los conceptos que se mencionan, es el caso de un **restaurante** y una familia que decide comer allí el viernes por la noche.

Motivación

Dar a conocer una técnica que puede potenciar a los equipos que quieran mejorar sus procesos para alcanzar sus objetivos con mayor eficiencia, focalizados en el **valor** que aportan a sus clientes y en la disminución del **desperdicio** que generan en el camino.

Descripción

El *Value Stream Mapping* significa mapear (es decir, crear un mapa) con el **flujo de valor**.

En un taller de *Value Stream Mapping*, suelen llevarse adelante los siguientes pasos:

- Trabajo sobre el flujo de valor actual.
  - Elección del **Proceso** que se quiere mejorar.
  - Selección del **inicio** y el **final**, dejando por fuera lo que ocurre antes del inicio y después del final.
  - Identificación y visualización de las **actividades** que forman parte del proceso.
  - Identificación de los **tiempos** que llevan la **realización** de dichas actividades.
  - Identificación de los **tiempos** de **espera** que hay entre las actividades.
  - Cálculo del ***lead time***.
  - Selección de las actividades que aportan **valor**.
  - Cálculo de la **eficiencia** del proceso.
- Optimización del flujo de valor.
  - Diseño de un **experimento** para mejorar el proceso.
  - Cálculo de la **hipotética nueva eficiencia** del proceso.
- **Experimentación** y análisis posterior.
  - Llevar adelante el experimento.
  - Realizar un nuevo análisis con los resultados del experimento.

A continuación se detalla cada uno de los pasos.

Trabajo sobre el flujo de valor actual

Elección del Proceso

Saber qué proceso queremos mejorar es un buena forma de comenzar. Se dejarán de lado todos los demás procesos en los cuales el equipo participa. Tener foco posibilita un mejor análisis que potencie el flujo de valor (también conocido como cadena de valor o cadena crítica).

En el caso del restaurante, se podrían elegir distintos procesos. Se tomará uno en particular: recibir y dar de comer a una familia, un viernes por la noche. Se dejarán de lado en este análisis la limpieza nocturna del salón luego de cerrar el local, la preparación del salón para abrir el local, la compra de insumos para la cocina, la difusión y el *marketing* que se hace en redes sociales y otros procesos que hacen posible que el restaurante opere normalmente. El foco de la sesión de trabajo será la cena de la familia.

Selección de Inicio y Fin

Dado que todo proceso tiene su contexto, es necesario elegir puntos desde y hasta los cuales se realiza el análisis.

En el ejemplo, se tomará como punto de inicio el momento en el que la familia entra a local y como punto de fin el momento en el que sale del mismo. Queda fuera del análisis la forma en la cual la familia se enteró de la existencia del restaurante, el trayecto desde la casa hacia el restaurante y viceversa.

Identificación de Actividades

En esta etapa se desglosa el proceso en actividades.

Es aquí donde, durante el taller *Value Stream Mapping*, suelen aparecer las notas adhesivas de colores: cada actividad queda representada en un papel. También se puede dibujar el flujo en una hoja que esté visible para todos los participantes del taller.

Una de las ventajas de usar notas adhesivas, en comparación con los dibujos, es que con las notas adhesivas se puede modificar fácilmente tanto el nombre de una actividad (descartando la nota y usando una nueva) como también el orden relativo entre actividades (despegando las notas adhesivas y cambiándolas de lugar), sin tener que borrar o tachar lo que se ha dibujado.

Algo valioso de esta etapa es la búsqueda de un lenguaje común. Cuando se van mencionando las actividades y se van escribiendo sus nombres en el mapa, pueden aparecer diferentes formas de nombrar cada actividad. La identificación de los puntos de vista da lugar a una conversación al respecto que suele servir para llegar a acuerdos y unificar el lenguaje.

Es habitual que en este momento ya empiecen a surgir ideas para mejorar el flujo de valor.

En el ejemplo del restaurante, las actividades podrían ser: Sentarse en una mesa, Elegir y ordenar la comida, Llevar el pedido a la cocina, Cocinar, Llevar la comida a la mesa, Comer, Pedir la cuenta, Pagar e irse.

Identificación de Tiempos de las actividades

Luego de identificadas las actividades del proceso, se procede a trabajar con métricas. El llamado *Process Time*, también conocido como *Cycle time* o *Value-added time*, es el tiempo que lleva cada actividad o una parte parcial del proceso. Es un tiempo necesario para agregar valor.

Si los tiempos se han medido, se utilizan directamente esos datos. Si no se han medido, se puede apelar pragmáticamente a la memoria y/o a la estimación de los participantes del taller.

En el ejemplo, el tiempo que lleva cada actividad podría medirse en minutos o en minutos y segundos, dependiendo de qué tipo de análisis o mejora se quiere llevar adelante. En otros casos se podrían usar días o alguna otra unidad de medida.

Sentarse en una mesa (1 minuto), Elegir y ordenar la comida (10 minutos), Llevar el pedido a la cocina (2 minutos), Cocinar (18 minutos), Llevar la comida a la mesa y servirla (4 minutos), Comer (25 minutos), Pedir la cuenta (2 minutos), Pagar (10 minutos) e Irse (1 minuto).

Identificación de Tiempos de espera

También se miden tiempos entre actividades. El tiempo en espera -o *Idle time*-, es el tiempo que transcurre esperando a que una actividad sea realizada o esperando a ser pasado a una próxima etapa o actividad.

En el ejemplo: Esperar una mesa disponible (5 minutos), Esperar al camarero para pedir la comida (5 minutos), Esperar a que los cocineros comiencen a procesar el pedido (1 minuto), Esperar al camarero para que lleve la comida a la mesa (3 minutos), Esperar al camarero para pedirle la cuenta (5 minutos), Esperar el vuelto (3 minutos).

Cálculo del *Lead Time*

El ***Lead Time*** o *Throughput time*, es el **tiempo total** transcurrido entre el momento en que se recibe una solicitud de trabajo hasta que el cliente está satisfecho.

Lead Time = Process Time + Idle time

En el ejemplo, la sumatoria total de los tiempos de actividades y de espera da 95 minutos. Es el tiempo que pasa entre el momento en el cual llega la familia para comer y el momento en el que sale satisfecha del restaurante.

Selección de actividades que aportan Valor

De la secuencia de las actividades se determinan aquellas que le dan valor al cliente final. El criterio para definir si una actividad aporta o no aporta valor suele estar basado en el punto de vista del cliente y debería validarse.

En el ejemplo, las actividades que aportan valor a esta familia son: Elegir la comida (pues hay platos que en su casa no pueden preparar), Cocinar (pues es tiempo que la familia dedica a compartir conversaciones en lugar de estar cocinando) y -por supuesto- Comer.

Cálculo de la Eficiencia

Esta y todas las etapas previas sirven para establecer el modelo actual del flujo de valor.

El cálculo de la eficiencia se mide a partir de la división del tiempo de actividad entre el *Lead Time en porcentaje.*

En el ejemplo, el tiempo de actividades valiosas es 53 minutos y el tiempo total es 95. La eficiencia del proceso es de 55,8%.

Optimización del flujo de valor

Diseño del experimento de mejora

En esta etapa se rediseña el modelo del flujo de valor a partir de un experimento, el cual luego será llevado adelante. El experimento cuenta con una hipótesis que se pretende validar o refutar. La hipótesis suele estar relacionada con las mejoras resultantes de remover algún cuello de botella específico.

Lo primero que se hace es identificar los tipos de desperdicio presentes en el flujo. También son llamados restricciones o cuellos de botella. Por ejemplo: esperas, retrabajos, transportes innecesarios o trabajo en progreso y no terminado.

Una vez identificados todas las restricciones críticas, se priorizan y se diseña una mejora concreta alrededor de las más crítica. Algunos ejemplos de mejoras concretas son: agrupar tareas, unificar actividades, agregar recursos o capacitar personas.

Siempre se recomienda utilizar la ley de Pareto: con pocos cambios se pueden lograr grandes impactos para disminuir el *Lead time* y/o aumentar la eficiencia.

En el ejemplo del restaurante, se puede pensar en varias optimizaciones posibles analizando cuál es más factible y conveniente teniendo el cuenta las restricciones económicas u otras. Una optimización en particular puede ser mejorar los tiempos de espera relacionados con el pago y el vuelto. Para esto, los camareros llevarán consigo los instrumentos necesarios para cobrar (POS, del inglés *Point of Sales*) y el dinero suficiente para dar el vuelto en el momento. Se espera que ese tiempo de espera del vuelto desaparezca y que el pago tarde solo 5 minutos. Ahora el *Lead Time* sería 87.

Cálculo de nueva eficiencia esperada

Hacemos una estimación de la nueva eficiencia esperada con la optimización del flujo de valor, pasando por el cálculo del tiempo de actividades, tiempos de espera y *Lead time.*

Con la mejora realizada, si el experimento tiene éxito, sería 60,9%.

Es decir, la eficiencia mejora en un 5,1% y el *Lead time* se redujo en un 8,42%.

Experimentación y análisis posterior

Fuera del alcance del taller del *Value Stream Mapping* nos abocamos a implementar los cambios identificados, llevando adelante un experimento y analizando los resultados.

Aquí se evalúa el nuevo modelo implementado y pasamos a repetir el ciclo: siempre habrá posibilidades de mejora.

En el caso del restaurante el experimento controlado se podría realizar con alguno(s) de los camarero(s) en determinados turnos, luego de lo cuales, se deberá repetir el ciclo contemplando los nuevos tiempos observados.

Elaboración de historias de usuario centradas en comportamiento

Por Juan Daza Arévalo, @juanenlasala

Palabras clave

Historias de usuario, comportamientos, valor, épicas, temas

Intención

Esta técnica ha sido diseñada con el fin de apalancar la idea de la “oportunidad para una conversación” que viene de la mano de una historia de usuario. Que ese diálogo sea poderoso al contar con un texto donde están consignados los comportamientos que queremos acompañar con la solución que se propone.

Motivación

Las historias de usuario cuentan con un formato, una plantilla que permite un encuentro para validar qué se espera lograr en el próximo sprint. Sin embargo, el poder no radica en la forma en que se redacta, en cada historia reposan comportamientos que se desean modificar o a los que se quieren apelar cuando el usuario esté al frente de la solución. Tener presente el comportamiento como eje permite trabajar con una óptica de User Centered Design o Diseño Orientado al Usuario.

Descripción

Las historias de usuario son la respuesta a la forma tradicional en la que los “requerimientos” se convertían en un listado de tareas y acciones que, casi siempre, desembocaban en situaciones abiertas a interpretaciones, o en un listado de acciones cerradas. Son la evolución de una tarea redactada en forma de historia que recoge el verdadero problema que se desea resolver.

Los primeros rastros en la historia de las historias de usuario conducen a Steinberg & Palmer [Steinberg-Palmer 2003] y al artículo de Bill Wake [Wake 2003] en el que propone, a partir de las siglas, INVEST y SMART la construcción de historias: independientes, negociables, valiosas, estimables, pequeñas y que se puedan testear (probar) o si se toma el segundo acrónimo que sean: eSpecíficas, medibles, alcanzables, relevantes y con tiempo regulado.

En su artículo, Wake cita a Ron Jeffries y describe las historias de usuario en XP (Extreme Programming) como herramientas que deben tener tres componentes: *Cards* (Tarjetas) como medio físico; *Conversation* (Conversación) o la discusión que genera la propuesta de dicha historia, y *Confirmation* (Confirmación) o la manera de probar que se ha cumplido lo esperado. Es frecuente encontrar discusiones frente a las sugerencias de redactar historias de usuario independientes y valiosas y que a la vez sean pequeñas.

Justamente ese es el punto de partida de Gojko Adzic [Adzic 2014] cuando afirma que “software valioso es un concepto vago y esotérico en el campo de los usuarios de un negocio, pero el tamaño de la tarea es algo que se puede tener bajo control para un equipo de desarrollo, por eso muchos equipos terminan escogiendo tamaño sobre valor.” El valor se convierte en un adjetivo que genera dudas y cada quien apropia a su manera. Por eso surgen expresiones como “una página dinámica” para expresar lo que se desea, frases que aparecen como muestra de un problema importante de fondo.

En XP se sugiere que la historia de usuario sea escrita por el cliente. Desde otros marcos de trabajo y metodologías es una herramienta a la que se llega en conjunto, como un ejercicio colectivo perfecto para cubrir temas de experiencia de usuario (UX), prioridad, etc. Nuevamente es una apropiación que cada equipo va refinando a su modo de trabajo. Las dudas posibles del uso de historia de usuario pueden resolverse en la medida que se explica el contexto y las oportunidades asociadas a otro nivel para el proyecto, el negocio y el usuario.

Jeremy Jarrel [Jarrel 2014] describe las diferencias entre historias de usuario, temas y épicas, como un conjunto de textos que en distintos momentos describe las necesidades del proyecto. La historia de usuario dice, es una “unidad auto-contenida de trabajo acordada entre el equipo de desarrollo y el stakeholder.” Los temas, agrega, son “ideas expuestas en historias que se pueden agrupar” en atención a un tema, funcionalidades similares, etc. Mientras que las épicas “comprometen un flujo completo para un usuario” a diferencia del lugar común de verlas como historias grandes, que necesitan refinamiento.

Lo que se quiere hacer supera el tradicional guión de: "Como un [rol] yo deseo [características] para que así exista [beneficio]." porque las conversaciones deben confirmar no lo que se desea hacer sino lo que se quiere lograr. No toda acción o desarrollo apoya un

objetivo. Estas conversaciones son momentos para confirmar una y otra vez qué es lo que realmente se quiere hacer. De ahí que Gojko Adzic y David Evans [Evans 2014] hablen de ver las historias de usuario como una oportunidad de “modificar comportamientos”.

¿Realmente la funcionalidad que tiene una solución apela a un comportamiento? ¿No se trata únicamente de un proceso donde el código activa tareas y funciones? Cada funcionalidad debería responder a una hipótesis o la suma de un supuesto con una medición, más una alta dosis de empatía.

Wendell [Wendell 2013] describe la forma en la que trabaja en su organización donde implementan “algunos elementos de agilidad” y se ve una correlación con el Pensamiento de Diseño o Design Thinking. En la base de todo desarrollo está un proceso de “entendimiento” que en la agilidad se ha recopilado en distintas prácticas conocidas como Inception. Un ejercicio atento de conexión con el usuario donde se identifican los problemas y posibles usos para que la solución propuesta confirme que está clara la visión que se tiene del producto o servicio.

Al apoyar procesos de desarrollo desde el diseño de la propuesta de valor, he querido darle un alcance distinto a las épicas para encontrar en ellas los comportamientos que se van a acompañar. Es imposible cambiar un comportamiento con sólo una historia de usuario pero sí se puede influenciar un comportamiento para que, poco a poco, se convierta en un hábito.

El flujo de trabajo descrito en una épica está poblado de comportamientos: miedo a entregar información, dudas por el uso de la información privada, reservas por el sentido del proceso, molestia por tener que memorizar una nueva contraseña, sospechas por la relación que se establece entre una red social y la solución que está usando, etc. Estos y otros comportamientos se van asentando a medida que las rutinas asociadas al proceso se repiten, son más sencillas, transparentes, etc.

El poder detrás de la experiencia de usuario no radica en una sensación de gusto o satisfacción, el poder se consolida si hay un comportamiento que queda satisfecho. Por eso, desde la empatía, se conversan cuáles son los comportamientos que puede tener un usuario en distintos momentos de uso de una solución. Parte de los diálogos necesarios al diseño de la épica.

Para esos diálogos diseñé un formato, una plantilla llamada LYPS como acrónimo de Love Your Epics. Por un lado, busca resignificar la idea de que una épica es nociva por no tener foco o estimación y convertirla en una oportunidad de ampliar la mirada del proyecto. Por otro, el juego de palabras y el término “lips” o labios que resuena con conversación, encuentros, “besos”, etc.

LYPS se usa en el Inception o gestación de la idea, durante el diseño de la propuesta de valor, a medida que vamos descubriendo las posibilidades de la solución que se va a programar, etc. y propone una serie de campos para ser usados con los interesados en cada fase del proyecto. Puede ser desde el equipo de desarrollo en pleno o en sesiones con cliente y desarrolladores.

Los campos descritos en la imagen se utilizan con la misma libertad de apropiación de otras herramientas ágiles. Sin embargo, el grupo debe tener en cuenta qué entiende por “comportamiento que espera modificar” porque es sobre eso que se puede medir el impacto de la solución, por ejemplo "Disminuir en un 20% la cantidad de formularios rechazados". Este camino permite oportunidades de mejora y de modificación constante de la solución.

Al respecto, y aunque este capítulo no se refiere al comportamiento humano es importante tener en cuenta que las conductas humanas y los comportamientos tienen relación y sutiles pero importantes diferencias. Una conducta se refiere a acciones asociados a un código propuesto en grupo y con implicaciones morales. El comportamiento habla de respuestas, acciones y actividades de un organismo. La conducta se refiere también a una lectura trazable en el tiempo, normalmente en una institución mientras que el comportamiento responde a interacciones inmediatas.

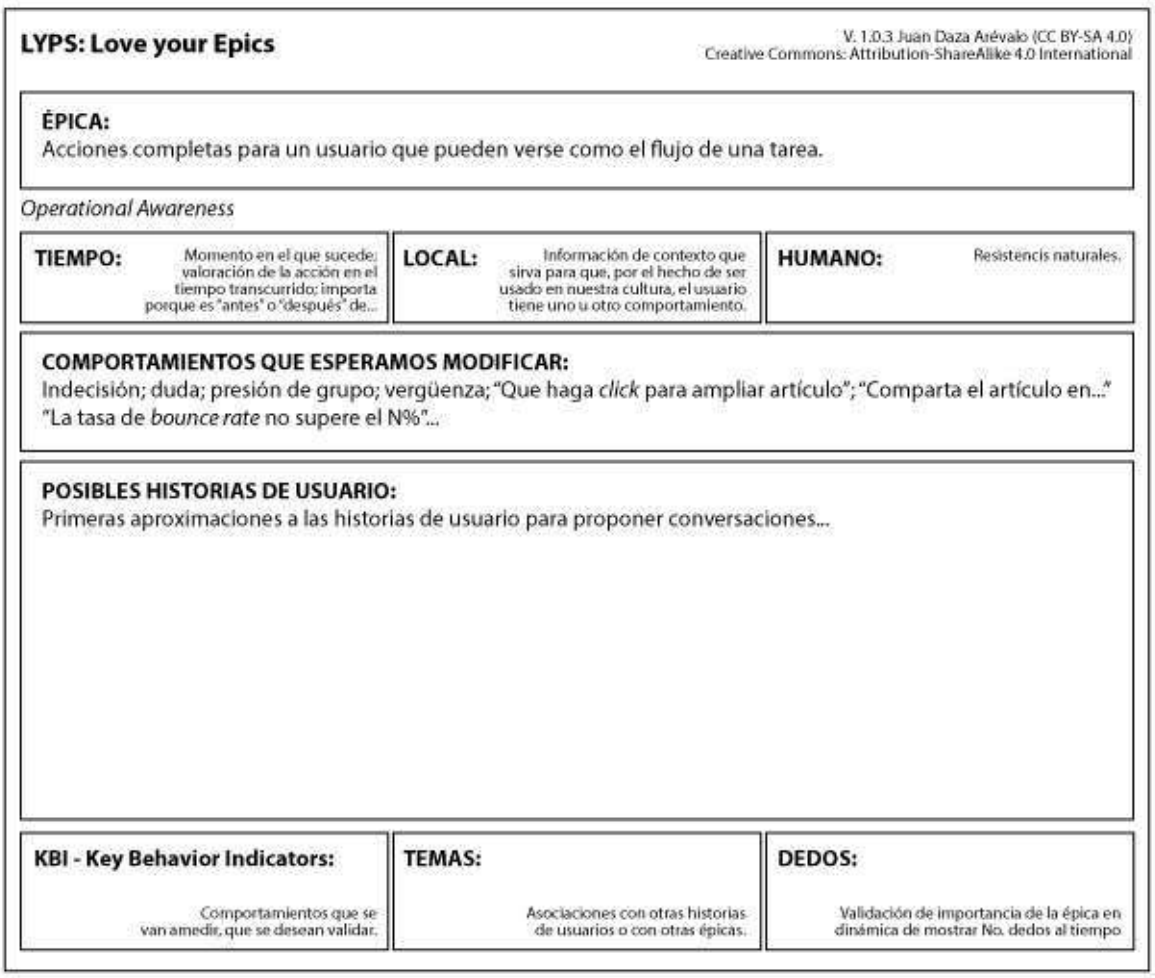

Figura 4.1_. LYPS - Love your epics

LYPS ha sido usado para la identificación de comportamientos en aplicaciones móviles y sitios web para encontrar si las implementaciones confirman los números esperados. La hemos puesto en marcha usando a la par herramientas de graficación de tráfico y donde los usuarios hacen realmente click (por ejemplo http://www.crazyegg.com o https://mouseflow.com) con el fin de validar los comportamientos esperados.

La idea de encontrar un mecanismo que permita modificar, con precisión, un comportamiento humano es poco menos que una fantasía. Nada puede predecir cómo los individuos y los grupos de personas reaccionan frente a una señal o un impulso; podemos entender y anticipar algunos pero estamos siempre frente a la complejidad propia de los seres humanos. Contemplar los comportamientos es preparar los límites suficientes para contar con metas y objetivos no sólo alineados con el negocio sino con el uso de las soluciones.

En nuestros proyectos LYPS nos ha dado foco en tres aspectos que aborda Gojko Adzic [Adzic 2016] y que llama operational awareness y que en la plantilla se ven como Tiempo, Local y Humano. Los comportamientos van de la mano de esas variables que nunca podemos controlar. La incertidumbre no es otra cosa que la realidad convertida en una historia cruel cuyo autor definitivamente no nos hace caso. Por eso, antes de medir resultados nos hemos propuesto a buscar cómo confirmar comportamientos y ojalá este formato sea un aporte para una meta también difícil.

Técnicas de OnBoarding para la gestión de conocimiento

Por Vanesa Savino, @VaneSavino

Palabras claves

Transferencia de conocimiento, Coaching, Roles variables.

Intención

¿Cómo trasmitir el conocimiento a los nuevos miembros de un equipo?

Motivación

Con la llegada de un nuevo miembro a un equipo nos encontramos con el dilema de transmitirle conocimiento relacionado con nuestro proyecto. Además de las pruebas automatizadas de código, en muchos casos hay que brindar información conocida por los expertos del negocio.

Los métodos expuestos a continuación intentan que los expertos puedan dividir su trabajo entre la formación de los nuevos integrantes y sus tareas habituales.

Descripción

Las ocho prácticas que aquí se mencionan surgieron de la necesidad de reorganizar un equipo de desarrollo. Sólo permanecieron dos miembros del equipo original: un desarrollador y el analista funcional/tester que llevaban trabajando tres y siete años, respectivamente, en el proyecto. El desarrollador fue designado como líder del equipo y al mismo tiempo ingresaron tres nuevos desarrolladores, uno de ellos con experiencia previa y dos más que iniciaban su carrera en sistemas.

Al principio el caos era tal, que el líder siempre estaba atrasado con su trabajo, y justamente era quien resolvía los temas más urgentes o importantes.

Pasada la etapa de adaptación, de tres meses aproximadamente, los desarrolladores observaron que el líder continuaba sobrecargado y decidieron proponer algunos métodos para distribuir el conocimiento, aliviar la presión del líder y al mismo tiempo, sentirse confiados al realizar un requerimiento complejo.

La aplicación de los métodos fue un proceso iterativo. Algunos métodos fueron creados dentro del equipo y otros adaptados de la bibliografía.

A continuación se muestra un listado con la organización de los mismos:

- Regla de la mano izquierda.
- *Coaching.*
- Más que programación en parejas.
- El héroe nuestro de cada semana.
- Resolución que hace eco.
- Repositorio de conocimiento compartido.
- Capacitación.
    - Clases particulares.
    - Tarde de películas.
- Diseño colaborativo.
    - Adaptación de Kata de arquitectura.
    - Reunión de diseño más informal.

Regla de la mano izquierda

Este método previene que el experto sea consultado todo el tiempo, escalando las dudas por intermediarios, hasta agotar las instancias y consultarle directamente. Además contribuye a que el conocimiento fluya a través del grupo y no esté centralizado en el experto.

Cuando surge una duda, el primero a ser consultado es nuestro compañero de la izquierda. Si la duda se puede resolver, no se escala más. De lo contrario nuestro compañero se la transmite a su compañero de la izquierda, y así sucesivamente hasta llegar al experto. En este caso el experto explica la solución para todo el equipo.

Coaching

Este método tiene por objetivo que el desarrollador se sienta seguro al introducirse en un tema complejo ya que su trabajo está respaldado por los conocimientos brindados por el experto.

Cuando un miembro del equipo necesita resolver una tarea que requiere del conocimiento del especialista, solicita su asesoramiento. El especialista aporta su visión y experiencia, discutiendo juntos la forma de aproximarse a la solución con menos trabas y escollos en el camino.

Más que programación en parejas

En programación de a pares (*pair programming*) el grupo es conformado por dos programadores, pero en esta propuesta uno de los miembros es el experto que no necesariamente es un programador. A diferencia de *coaching*, el experto permanece constantemente junto al desarrollador hasta finalizar la solución.

Este método favorece la cooperación entre diferentes roles, aportando visiones que ayudan a comprender el problema o fijar conceptos.

Se arma un equipo temporal para trabajar en el código. Si el especialista programa, puede codificar las partes más complicadas de la solución y explicarle su funcionamiento al compañero. Cuando el especialista no sea técnico puede validar la solución que construyan.

Siempre que se crea conveniente, se puede solicitar la colaboración temporaria de miembros con otros roles, para que aporten otros enfoques del problema, formando un equipo de tres miembros. Un ejemplo de esta colaboración puede ser un grupo conformado por un especialista técnico y otro desarrollador que le solicitan ayuda al analista funcional para despejar una duda de negocio que surgió en medio de su tarea.

Este método puede aplicarse junto con la “regla de la mano izquierda” y armar un equipo con el compañero de la izquierda.

El héroe nuestro de cada semana

Este método tiene un doble propósito, el primero es que cada miembro del grupo se encuentre capacitado para resolver las urgencias del sistema. El otro es que el experto pueda continuar con sus tareas y no se encargue exclusivamente de los casos urgentes, que provocan demoras en su trabajo si las interrupciones son muy frecuentes.

Del Agile Open Camp 2015 se tomó una idea interesante: un miembro del grupo va a tener el rol de "Héroe" para ocuparse de los problemas urgentes del sistema.

El rol es ocupado por cada miembro durante una semana, hasta que todos lo hayan ocupado una vez y luego se vuelve a comenzar. Para hacerlo identificable frente a los demás miembros del equipo, cada miembro puede tener un muñeco o imagen de su héroe

favorito a la vista, sobre la que pegan una estrella de Sheriff para informar que están de “guardia”.

Quien desempeñe este rol continúa con sus tareas habituales; en caso de haber un problema urgente, pospone su tarea y se dedica a la resolución del problema. Dependiendo de la complejidad de la situación, un experto o cualquier otro miembro del equipo es requerido para ayudar a resolver la urgencia. Así cada miembro aprende a resolver errores críticos sintiéndose confiado bajo la tutela de alguien con conocimiento sobre el tema.

Este método puede combinarse con “*coaching*” y “más que programación en parejas” para facilitar un arreglo urgente.

Resolución que hace eco

Este método pretende unificar criterios, lo que facilita el entendimiento de los temas. También propone que se incorporen ideas generales del funcionamiento del sistema cuando se investiga en busca de comportamiento común dentro de la aplicación.

Al recibir un requerimiento para cambiar una característica en particular, se tiene que investigar si hay otras funcionalidades con similar comportamiento que requieran ser modificadas y también cambiarlas.

En una aplicación grande y de larga data, suele ocurrir que la solución a un problema ya está implementada en otro lugar, de modo que es importante asegurarse que realmente sea necesario dedicar tiempo a construirla desde cero, cuando lo más probable es que lleguemos a un código y lógica duplicados, que traerá consecuencias si el día de mañana uno de estos comportamientos se modifica y el otro no. A no repetir código!

Este método se puede combinar libremente con cualquiera de los métodos que consisten en desarrollo, ya que no interfiere con su misión.

Repositorio de conocimiento compartido

Construir un repositorio en la nube y compartir la documentación ha desplazado al uso de otros sistemas más formales. Hoy en día existen muchas plataformas y servicios gratuitos que brindan acceso desde diferentes dispositivos, lo que facilita la distribución de la documentación.

Este método tiene por objetivo tener disponibilidad online de la documentación con todas las ventajas que eso representa, entre ellas: accesibilidad móvil, y un mecanismo más simple para creación de contenido, ya que se evita la instalación de programas específicos.

Es muy importante mantener un orden conocido por todos, para que cada miembro sepa dónde ubicar el contenido que produce o donde buscar la información que necesita. Dependiendo de la organización se pueden otorgar diferentes permisos de acceso y edición, lo que requiere una moderación del contenido que se cree colaborativamente.

Capacitación

Se van a explicar dos maneras de comunicar información. El objetivo es incorporar conceptos a través de diferentes formatos, el más moderno, audiovisual y el tradicional con material escrito.

Clases particulares

Consiste en preparar clases con temas especiales y brindar material para los asistentes que ayude a la compresión del tema tratado. Es fundamental que luego el material se suba al repositorio para que pueda ser consultado en cualquier momento. Además conviene tener un modelo de contenido para implementar las charlas, dando un marco mínimo de información que deben cubrir.

Tarde de películas

Existen muchos videos de tutoriales online que nos enseñan desde cómo hacer el nudo de una corbata hasta como aprender a tocar la guitarra. Esta forma de enseñanza es cada vez más habitual, incluso se pueden encontrar clases de prestigiosas universidades disponibles online.

¿Por qué no aplicarlo entonces a nuestro sistema? Para partes que sufren pocas modificaciones, como la configuración de un entorno, es de mucha utilidad contar con tutoriales que muestren la configuración correcta, de esta forma se ahorra mucho tiempo de búsqueda infructuosa.

Diseño colaborativo

Como ya se sabe la visión de grupo en sinergía supera la suma de la visión de sus miembros.

En esta instancia se propone utilizar diseño colaborativo para llegar a la solución óptima de un problema.

Dependiendo de la estructura y la cantidad de miembros del equipo se pueden hacer reuniones más informales o adaptaciones de katas de arquitecturas.

El término kata proviene del karate y consiste en una serie de ejercicios establecidos que se realizan sólo. Este término se aplicó al diseño de código cuando Dave Thomas creó las “Code Kata”.

Las katas de arquitecturas se realizan en grupos, tienen reglas y roles definidos. Se acuerda un tiempo de duración para todo el ejercicio, los miembros se dividen en grupos de desarrolladores y se designa un moderador, responsable de responder todas las dudas de los equipos. Si el moderador es el líder del equipo, también se encarga de guardar las buenas prácticas de programación.

Los equipos se separan una distancia prudencial para hablar sin interferirse mutuamente. Después de finalizado el tiempo, un orador elegido por los miembros del grupo expone la solución frente al interesado. En este caso el rol del interesado puede ser ocupado por el analista funcional.

El moderador, el interesado y los restantes participantes formulan todas las preguntas necesarias para entender la propuesta. Al finalizar esta etapa todos votan al mismo tiempo la solución.

La votación se realiza con la regla del pulgar: pulgar para arriba para indicar que me gustó la propuesta. Horizontal para indicar que no me convenció del todo y pulgar para abajo para indicar que no me agradó.

Luego de la votación comienza la exposición del siguiente equipo hasta terminar, al finalizar se trabaja con la solución más votada.

Si el equipo es pequeño (3 o 4 personas) puede trabajarse más informalmente, reuniéndose frente a una pizarra y diagramando una solución. Aquellos con más conocimiento pueden proponer una solución tentativa y los demás pueden dar su *feedback* para ver en que se puede mejorar.

Es importante a tener en cuenta el *feedback* en ambos métodos, pues a veces la solución más brillante la aporta el miembro menos pensado.

Este métodos propician la participación de todos en la construcción de la solución, pero como la experiencia de cada miembro es diferente se pueden complementar con “*coaching*” o “más que programación en parejas” dependiendo de si es una solución más técnica o de negocio.

Los métodos expuestos tienen como eje principal compartir conocimientos, algunos han surgido por la necesidad de resolver un problema, otros han sido adaptados de la bibliografía o tomados literalmente. Estos métodos pueden aplicarse en forma iterativa e incremental; lo fundamental es lograr la comodidad del equipo y una transferencia de conocimiento efectiva.

La intención de exponerlos aquí es que sirvan de base para generar otros métodos que se adapten a cada situación particular.

El progreso que se realizó después de su implementación ha sido muy significativo; además contribuyó a crear un equipo muy sólido, con confianza en sus miembros y auto-organizado.

SHU-HA-RI: Un Modelo de Aprendizaje

Por Omar Fernández, @omarfl7

Palabras clave

Aprendizaje, mejora continua, shu-ha-ri

Intención

Cómo enfrentar técnicas nuevas o rescatar las ya conocidas, haciendo énfasis en la búsqueda del aspecto fundamental de éstas, lo esencial, lo que hace de esta técnica singular.

Motivación

Si te encuentras estudiando y buscando alguna nueva técnica pero buscas ir más allá de sólo aprenderla. Deseas comprender el espíritu de ésta, de forma tal que puedas transportar dicho espíritu en diferentes contextos. Entonces reconocerás que todo proceso de aprendizaje lleva tiempo y pasa por diferentes etapas.

Descripción

Es posible que hayas escuchado sobre Shu-Ha-Ri como es posible que no, bien sea en el contexto de software o para otro contexto. Podrías revisar el artículo de Martin Fowler [Fowler 2014] para tener un primer acercamiento o el artículo de Alistair Cockburn [Cockburn 2008] para una revisión más extensa en el contexto de software.

Si realizamos una descripción breve, Shu-Ha-Ri es un concepto del arte marcial Japonés que describe las etapas de aprendizaje de un estudiante para lograr la maestría en el arte. Ésta consta de tres etapas cuya descripción es la siguiente:

Shu: significa mantener o proteger. En esta etapa el estudiante sigue los pasos indicados por el maestro sin preocuparse demasiado en los aspectos subyacentes de la técnica.

Ha: significa separar. En esta etapa el estudiante empieza a aprender los principios y la teoría subyacente detrás de la técnica. El estudiante formulará muchos porqués sobre la técnica.

Ri: significa trascender. En esta etapa el alumno ha aprendido la técnica del maestro y empieza un aprendizaje adaptado y creado bajo su propia experiencia. La relación estudiante-alumno ya no es debido al aprendizaje de la técnica sino mucho más cercana, manteniendo la independencia del alumno.

De lo anterior podemos apreciar etapas bien definidas, tareas concretas por etapas y un maestro el cual pueda guiarte. Estas etapas pasan por generar una conciencia en aquello que se practica para en algún momento asimilarlo y mejorarlo. Sin embargo, esto nos lleva a entender que no basta con sólo leer o practicar tempranamente, hay que asimilar su contenido, su fundamento y esencia para poder apreciar el trasfondo de esta práctica. Es decir entenderlo para apreciarlo y mejorarlo. No basta haber leído la biblia para convertirse en santo y en ese sentido y en ese camino, no podemos detenernos y decir que ya lo entendemos si no hemos generado conciencia en ello.

Ciertamente al aprender una técnica no siempre tendremos la compañía de un maestro, lo cual sería bastante importante pero, sin embargo existe un grupo de actitudes que se manifiestan en cada etapa. Las describo a continuación:

Shu: en esta etapa temprana es importante practicar mucho y entender lentamente los pasos de la misma forma como si aprendiéramos a caminar. Seguramente no tendremos conciencia clara del porqué de cada paso pero podremos ver coherencia entre paso y paso. En este punto es importante medir y reflexionar, porque nos llevará a la siguiente etapa. El hecho de medir nos ayuda a comparar que tanto hemos avanzado nosotros mismos o con respecto a aquello que queremos lograr con esta práctica.

Ha: si ya hemos entendido la técnica, ciertamente podemos desenvolvernos con facilidad, pero se sentirá como quien sigue una receta, y es en este punto donde debemos cuestionar si lo aceptamos o no tal como lo hemos aprendido y entonces nos percatamos que sólo aprender la técnica no basta, la sucesión de pasos uno tras otro tiene un porqué que es importante encontrar y ello nos lleva a buscar e investigar. Si mantenemos dicho espíritu de no simple aceptación y de considerar el aprendizaje como algo no terminado, entonces nos llevará a la siguiente etapa.

Ri: aunque parezca extraño es importante olvidar lo aprendido porque estos fueron sólo una secuencia de pasos lógicos pero que no develan el espíritu del mismo. Nos sirvió para aprender la técnica pero en este punto es el velo que no deja ver más allá. El olvidar permite desapegarse de los pasos aprendidos para que poco a poco podamos apreciar que los pasos fueron una forma de mostrar la práctica pero que oculto por debajo se encuentra el espíritu de la misma. El desapego se manifiesta también al retirar los pensamientos que

llevan a considerar como única la interpretación de la técnica que hemos aprendido y de esta forma, empezar a aceptar nuevas interpretaciones de la misma como un vaso que puede verse de distintas posiciones. Posiblemente volverás a recorrer los pasos iniciales pero esta vez será como una danza natural, sin esfuerzo, pero que en sí misma muestra el espíritu de la práctica. En este punto se percibirá que se debe seguir aprendiendo, y es entonces cuando estamos comprendiendo el espíritu de la técnica. Ahora es cuando empezaremos a extenderla y visualizarla en múltiples y nuevos contextos

De forma breve podemos identificar en qué etapa nos encontramos apoyándonos en algunas actitudes que mostramos:

1. Si sigues estrictamente los pasos como una secuencia, bien sea porque recién inicias con la técnica o porque así aprecias que debe ser, estás en Shu.
2. Si necesitas saber el porqué de dicha secuencia de pasos, y buscas el porqué de los enlaces de los mismos en dicho orden y no en otro, y ensayas o experimentas una y otra vez, estás en Ha.
3. Si te percatas que empiezas a describir la técnica pero en distintas formas como quien describe un vaso en distintos ángulos pero sin apegarte a esa descripción y comienzas visualizar la misma en otros entornos, entonces estás en Ri.

Otro aspecto importante a tener en cuenta es que cada etapa no es una secuencia lineal sino más bien éstas se auto contienen como se puede ver en la figura siguiente. Pero también en ese sentido vamos incrementando las actitudes que manifestamos y, que en un principio te parecerán contradictorias pero con la práctica te percatarás que éstas se refuerzan entre una y otra.

Puedes considerar encontrarte en alguna etapa, pero las actitudes que manifiestes son las que realmente describirán en qué etapa te encuentras. Esto es un trabajo de autoconciencia que te indicará a tí mismo con qué actitud enfrentas el aprendizaje.

Es usual mantenerse en la etapa Shu y Ha y sentirse conforme, pero es importante llegar a la etapa Ri, en la cual no sólo se aprecia el sentido de una técnica o práctica, sino que también se develan aspectos de ella que la hacen singular, encontrarás patrones y nuevas formas de mejorarla. Apreciarás que el aprendizaje de algo no es un tema terminado sino más bien un camino que se debe seguir recorriendo.

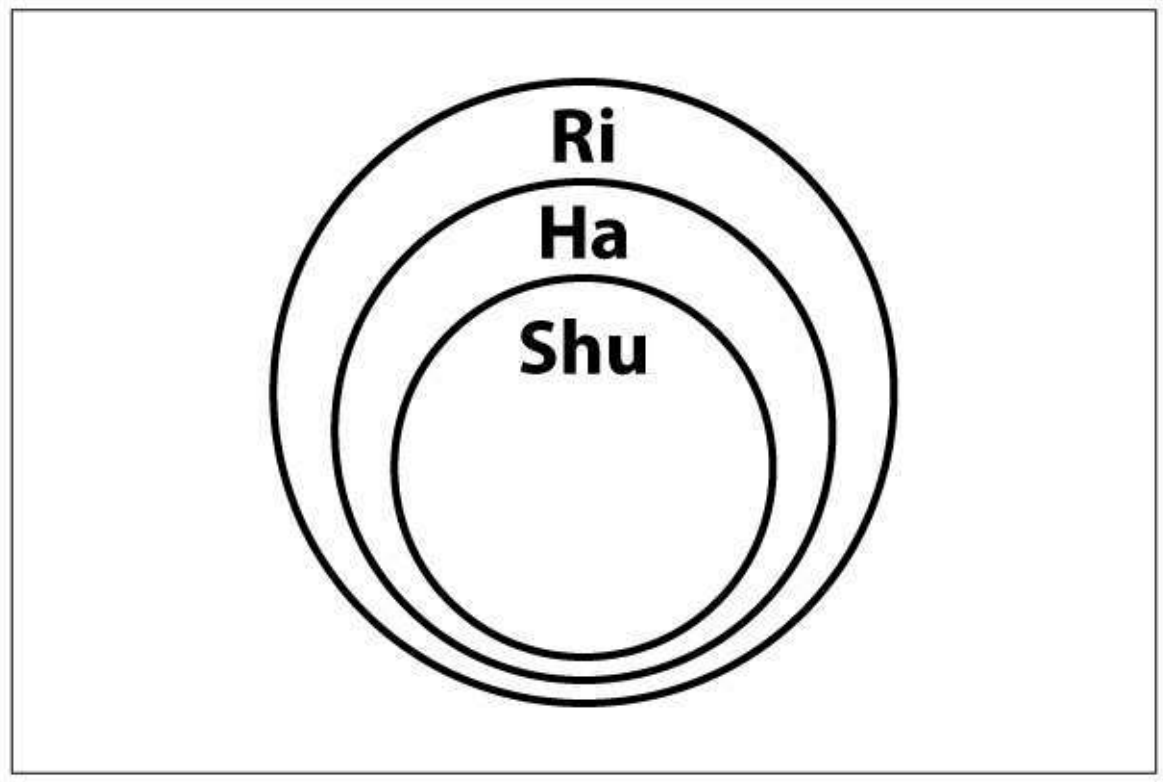

***Figura 6.1**. Relación entre Shu, Ha y Ri*

Además, una misma persona puede estar en distintos estados en cada una de sus destrezas en un instante de tiempo, y, en un equipo, cada integrante puede estar en distintos estados con respecto a una destreza. Por otro lado podríamos hablar del estado de todo el equipo en una destreza o técnica en particular.

Conclusión

Las tres etapas nos ayudan a reconocer la esencia de una técnica, y, como consecuencia de esto habilitarnos a poder transportar dicho conocimiento a diferentes contextos. Podemos, sí, mantenernos en las dos primeras etapas, pero esto sólo nos mantendrá como conocedores. Pero si deseamos trascender la técnica, deberemos llegar a la tercera.

Algo que también podemos percatarnos es que cada etapa no es un empezar y terminar, sino en la repetición de las etapas anteriores impulsa sutilmente a llegar a la siguiente etapa siempre que no descuidemos **las actitudes que manifestamos**.

Te invito entonces a que selecciones alguna técnica que tal vez consideres ya aprendida o alguna que desees aprender, revises las actitudes que manifiestas y realices éste recorrido sin fin del Shu-Ha-Ri.

Finalmente quisiera dejar una cita tomada de Shunryu Suzuki [Suzuki 1987], la cual he adecuado para este contexto de aprender:

“Nuestra comprensión de una técnica, no es meramente una comprensión intelectual. La verdadera comprensión es la práctica misma”

Continuous Discovery: Validación de ideas para el Backlog

Por Alejandro Faguaga, @afaguaga

Palabras clave

Dual track scrum, inception, discovery, productos con impacto, business value, UX

Intención

En general en los proyectos ágiles, la figura del *Product Owner* es la que trae nuevos requerimientos o necesidades del negocio. A veces como simples ideas o requerimientos de alto nivel, y otras -con suerte- en forma de *User Stories*. Sobre ésto le pide al equipo técnico que analice y estime el esfuerzo de desarrollo de dicha funcionalidad.

En este contexto, no sólo dependemos en primera instancia del conocimiento y buen criterio del *Product Owner* para detectar las necesidades reales de los usuarios, sino también, de su capacidad para explicitar y clarificar correctamente esas ideas al equipo de desarrollo, y luego además dependemos del buen entendimiento e interpretación del equipo sobre las ideas que el *Product Owner* pone sobre la mesa.

Esto comúnmente puede generar ítems del *Product Backlog* definidos de manera pobre o poco claros que pueden atentar contra las reuniones de planificación de los *Sprints*, haciéndolas tediosas o muy extensas. Incluso puede afectar la velocidad del equipo, que termina analizando, refinando y entendiendo las funcionalidades en paralelo con el desarrollo del Sprint.

El resultado de esta situación por lo general es la pérdida de tiempo y el re trabajo debido a que estos ítems no han sido completamente validados a tiempo. A priori pareciera que hay demasiadas variables en juego en esta cadena de gestación y concepción de nuevas funcionalidades a ser desarrolladas, y que dicha cadena está compuesta por eslabones endebles que pueden romperse en cualquier momento.

Todo lo anterior nos da la percepción de que en muchos proyectos la precisión y la correctitud del *Backlog* dependerá en gran parte de la fortuna.

Motivación

Si partimos del supuesto que la suerte está de nuestro lado, o que las capacidades del *Product Owner* hacen que sus pedidos coincidan con los requerimientos del usuario, en verdad aún quedaría mucho camino por recorrer. Cuando el equipo de desarrollo empiece a analizar lo que hay que hacer y el esfuerzo necesario para llevarlo a cabo, en reiteradas oportunidades podríamos caer en cuenta que lo que pide el negocio es inviable desde el punto de vista técnico, o que demandaría un esfuerzo prohibitivo para sus expectativas y/o posibilidades.

Y podría haber una variable más… Aún cuando la solución se pueda implementar desde el punto de vista técnico, el equipo de desarrollo y el *Product Owner* puede que no tengan en cuenta la usabilidad o la experiencia del usuario respecto de esa solución (ya sea por falta de tiempo o de conocimientos en el área de UX).

Constantemente, requerimientos válidos, e implementados con soluciones técnicamente apropiadas y correctas, podrían no alcanzar el objetivo final de satisfacer la necesidad del usuario, ya que este mismo podría no encontrar dicha solución como “usable” o ventajosa respecto de su situación actual.

Descripción

Hay que romper con la idea que el Product Owner trae los requerimientos y el equipo de desarrollo es un ente que ejecuta en función de ese *input* funcional una solución técnica, que la gente de UX validará desde el punto de vista del usuario final. En definitiva hay que romper esa cadena o secuencialidad en la interacción de los equipos o perfiles. Y una buena forma de cambiar esto es hacer que el Product Owner, algún representante del Equipo de Desarrollo y un representante de UX trabajen de forma colaborativa y conjunta desde la concepción misma de los requerimientos o User Stories.

Se forma así un equipo con las 3 “patas” que trabajará sobre ideas, validando de manera conjunta que: las funcionalidades que conformen el Product Backlog cubran la necesidad funcional del negocio (Product Owner); la solución planteada sea factible técnicamente (Desarrollo) y la experiencia de usuario en dicha funcionalidad sea tenida en cuenta (UX). Este proceso se realiza de manera continua y en paralelo con el desarrollo de los Sprints y se denomina *Continuous Discovery* [Cagan 2012a] o como lo llama Jeff Patton, *Dual Track Scrum* [Cagan 2012b]. Justamente porque plantea un *track* o *thread* de “Descubrimiento” (Discovery) de ítems del Backlog en paralelo con el *Delivery* de funcionalidad que se va desarrollando en los Sprints. Personalmente, prefiero el término *Continuous Discovery*, porque eso nos abstrae de Scrum, ya que este tipo de técnicas funcionan muy bien con prácticas como *Kanban* y otras también.

UX, Desarrollo y Producto se complementan para pensar cómo cubrir las necesidades del negocio juntos, aportando cada uno su experiencia y asegurando que la funcionalidad planteada será la mejor posible teniendo en cuenta los 3 puntos de vista. Si se detecta que la solución planteada originalmente no es factible técnicamente o no es usable, se ahorra mucho tiempo y además permite plantear alternativas de manera temprana.

Así el track de Discovery se preocupa exclusivamente de generar PBIs (*Product Backlog Items*) validados y el track de Delivery se enfoca en generar software funcionando basado en ese Backlog validado, e implementarlo lo antes posible en producción.

Adicionalmente, no solo queremos obtener soluciones validadas y realizables, sino que además queremos asegurarnos lo antes posible que realmente estamos cubriendo la necesidad del usuario final, y la mejor manera de lograr esto es obteniendo su feedback de manera rápida y ajustando en función de la retroalimentación recibida.

Continuous Discovery se basa en que en general, aproximadamente el 50% de las ideas propuestas para ser desarrolladas en un proyecto de desarrollo son erróneas o no cubren las necesidades reales de los usuarios.

Para esto la técnica lleva al extremo el concepto de *Fail Fast*, generando prototipos ejecutables para permitir probar la funcionalidad rápidamente y detectar si estamos en el camino correcto o no.

Hay muchas maneras de implementar esto. Puede ser a través de *A/B Testing*, prototipos operativos, funcionalidades que puedan ser activadas o desactivadas mediante un “*switch*”, *Mock Objects*, entre otros.

Independientemente de la forma, lo importante aquí es permitirle al usuario probar la funcionalidad y obtener su feedback lo antes posible.

Hace unos años tuve la oportunidad de trabajar en un equipo Scrum desarrollando una aplicación web de viajes, y aplicábamos varias técnicas. Por un lado se armaba un prototipo ejecutable, con objetos *mock* que simulaban la lógica de negocios y el acceso a datos y una pantalla de UI con validaciones y el aspecto visual que queríamos que tenga la página. Esto se instalaba en *tablets* o teléfonos móviles y luego salíamos y pedíamos a la gente en la calle que use la aplicación y nos diera *feedback* directo, en ese mismo instante.

También armábamos laboratorios de pruebas, donde traíamos a usuarios específicos, les dábamos instrucciones para que ejecuten una funcionalidad, mientras observábamos y tomábamos nota de los problemas que surgían al momento de usar la aplicación.

En otras oportunidades nos sentábamos al lado y los guiábamos para que usen la funcionalidad desde una computadora y les íbamos pidiendo *feedback* al respecto.

Hay muchas formas, pero lo importante más allá de cómo se obtiene el *feedback*, es lo que uno hace luego con ello.

¿Cómo se relaciona el track de *Continuous Discovery* con las conocidas sesiones de refinamiento (*backlog refinement sessions*)[Cohn 2015]? La realidad es que se complementan. El *Continuous Discovery* generalmente es previo, ya que tiene que ver con la creación y validación de ideas que luego se podrán o no transformar en PBIs. Y una vez que empieza el *Discovery* no termina nunca, con lo cual en algún momento, cuando empezamos a hacer refinamiento, vamos a tener las dos actividades en paralelo.

Lo interesante del *Discovery* es que de alguna manera "filtra" y valida los PBIs que luego serán refinados como parte del track de *Delivery*, es decir, si hacemos bien el *Discovery*, deberíamos tener PBIs a refinar pre-validados y de "mejor calidad" en cuanto a valor agregado para el producto, y adicionalmente no deberíamos tener PBIs que al refinarlos nos demos cuenta que son técnicamente inviables, o que aportan poco valor.

Ahora bien, la pregunta es, ¿toda "idea" debe pasar por una fase de Discovery antes de ser refinado en las *refinement sessions*? Eso depende del proyecto, del equipo, del contexto. A veces el equipo que lleva adelante el Discovery es cuello de botella, entonces hay que elegir. Lo ideal sería que, en la medida de lo posible, todo requerimiento o característica crítica de nuestro producto pasará por una instancia de Discovery previo a ingresar al *Product Backlog* y ser refinada, y las *features* menos relevantes quizás puedan ir directo a las sesiones regulares de refinamiento.

La dedicación del equipo de Discovery también dependerá del contexto y las posibilidades, teniendo en cuenta siempre que el tiempo y esfuerzo dedicado a esta tarea ayudará a evitar que ideas o funcionalidades que no son útiles sean analizadas en detalle o incluso desarrolladas más adelante (en el *track* de *Delivery*), con lo cual es tiempo bien invertido.

Conclusión

Un efecto colateral y muy positivo de la práctica de *Continuous Discovery* tiene que ver con la situación que se presenta continuamente en equipos ágiles, que terminan haciendo una especie de “mini cascada” dentro del marco de Scrum. El Product Owner trabaja en los “requerimientos”, que son pasados a los diseñadores, quienes generan los artefactos visuales que finalmente son pasados al equipo de desarrollo para construir y testear la funcionalidad. *Continuous Discovery* no se basa en cada rol entregando artefactos a los demás roles sino que se enfoca en tener al Product Owner, un desarrollador y un diseñador trabajando juntos, hombro a hombro en la validación de los PBIs.

Las ideas que no son bienvenidas por el usuario, se descartan automáticamente y las que son bien recibidas se ajustan en base al feedback obtenido y se convertirán en User Stories que pasarán a formar parte del Backlog. Desde este punto el Product Owner no tendrá más

que priorizar dicha funcionalidad, pero ya con el pleno convencimiento que tiene entre manos una funcionalidad que será útil para el usuario, usable y factible desde el punto de vista técnico, con una UI pre probada y validada y con el simple costo de haber tenidos un par de reuniones e invertido algo de tiempo en un prototipo.

Pero es muy importante no “casarse” con ninguna idea por más buena que pueda parecer a priori y si vemos que no funciona como esperábamos, descartarla automáticamente.

Prácticas eficaces para aplicar en Reuniones (In)eficientes

Por Alejandro Faguaga, @afaguaga

Palabras clave

Proyectos ágiles, comunicación efectiva, colaboración, ceremonias, reuniones eficientes

Intención

En muchas organizaciones la palabra reunión es prácticamente mala palabra, sinónimo de pérdida de tiempo, debido a que las reuniones son ineficientes y poco eficaces.

Esto mal predispone a la gente y genera mucha pérdida de tiempo de las personas.

Presentamos aquí una serie de técnicas o prácticas muy simples pero eficaces que permiten conducir reuniones más eficientes y productivas con un mínimo esfuerzo y organización previa.

Motivación

La previa

Supongamos que estamos empezando a acompañar a un equipo en su primer proyecto ágil, dentro de una gran organización tradicional, un gigante que se mueve muy lento.

Todavía no conocemos bien a los participantes del proyecto, pero como esto de “Agile” viene muy “*sponsoreado”* e impulsado desde arriba, nos empiezan a llegar mails e invitaciones a reuniones compulsivamente.

Una de esas invitaciones dice en el asunto “Revisión de Nuevo Proyecto”. Somos 12 personas invitadas a compartir 2 horas de nuestro tiempo, de las cuales no conozco a casi nadie.

En el cuerpo de la invitación no hay una agenda definida tampoco. La reunión es a la tarde y no sabemos porque nos invitaron, ni para qué, ni quienes van a participar, o mejor dicho qué roles van a desempeñar en el proyecto.

De la extensa lista de nombres nos parece reconocer a una persona y vamos a su escritorio a consultarle cual es el objetivo de la reunión. Nos responde que no tiene idea, pero que debe ser importante porque en la lista de invitados está su jefe y varios líderes de equipos que mantienen sistemas *core* de la organización. Además que hay varios “pesos pesados” de producto.

Volvemos a nuestro escritorio un tanto desilusionados.

La reunión

Entramos a la sala, hay 10 personas que no conozco. Sigue llegando gente hasta que claramente hay más personas en la sala de las que había en la invitación.

Llega una mujer que saluda y agradece porque estamos ahí, aún sin saber por qué y para qué. Claramente es la persona que envió la invitación.

Como el objetivo de la reunión y la agenda no están para nada claros, hay varios líderes que “por las dudas” llevaron a un analista y a un desarrollador de su equipo, por si se tratan temas funcionales o técnicos respectivamente. Esto hace que la cantidad de gente crezca desmesuradamente.

Empieza la reunión con todos hacinados en la sala y ocurre el caos, nadie sabía para qué era la reunión específicamente, pero como hay mucha gente invitada, varios aprovecharon para llevar una lista de sus problemas o inquietudes, que mejor que plantearlas en un ámbito donde están varios líderes juntos. Se empiezan a plantear entonces muchos problemas distintos, de forma desorganizada, muchos de ellos no relacionados con nada, ante el estupor del “organizador” que ve cómo su reunión se va por cauces inesperados.

Luego de varias discusiones sin un hilo conductor, el organizador logra encauzar la reunión hacia un tema que si tiene que ver con el proyecto. Para eso ya pasaron 40’ y ya hay varias personas que se dan cuenta que están de más, que no pueden aportar nada y que la reunión tampoco les está dejando gran cosa.

Adicionalmente, nos damos cuenta que para terminar de definir algunas cuestiones, está faltando gente fundamental que no fue invitada.

Pero claro, como la agenda no era explícita, nadie pudo detectar previamente que esas personas iban a ser necesarias.

Ya pasada la hora de reunión empieza la catarsis, producto de la frustración de la mayoría de los asistentes porque en “esa organización siempre pasa lo mismo”, que las “reuniones no sirven para nada”, que “son una pérdida de tiempo”, etc. Hasta que finalmente se cumple el horario y nos piden que entreguemos la sala.

Esto es un claro ejemplo de una extensa reunión de más de dos horas donde se hizo mucha catarsis pero nada productivo, y se plantearon diversos problemas pero ninguna solución. Dos horas de más de 20 personas totalmente estériles.

Descripción

Para intentar evitar todo lo anterior se pueden implementar una serie de prácticas que nos permitieron optimizar el tiempo de los asistentes, que no les quite las ganas de volver a tener una reunión y que por sobre todo sean simples, eficaces y sencillas de realizar, es decir implementables.

Prácticas

Enviar Agenda Visual: básicamente lo que se propone aquí es definir agendas digitales de manera gráfica para incluir en las invitaciones de las distintas reuniones o ceremonias, ya sean de Scrum o no.

Se muestra en la Figura 7.1 un ejemplo de agenda digital que se podría utilizar para la invitación a una reunión de planificación de Sprint.

Esta práctica permite clarificar varios temas, a saber:

- Tener un objetivo claro a cumplir en la reunión: explicitar el propósito de la misma.
- Definir la agenda y el tiempo necesario (Formato de la reunión): que todos sepan exactamente qué vamos a hacer en la reunión y también lo que NO vamos a hacer (una especie de *DOs* & *DONTs*).
- Ajustar y precisar la audiencia (participantes): describirla de antemano en forma clara, para que no sobre gente pero también para dar la posibilidad de que si falta alguien los convocados lo puedan plantear de antemano basándose en el objetivo y la agenda de la reunión.
- Impulsar “el hacer”: describir en la sección final de la agenda digital, lo que no podemos dejar de realizar inmediatamente después de abandonar la reunión (acciones concretas). En nuestro ejemplo de la Figura 7.1 esta sección se denomina llamada “*Al finalizar*…”

Esta práctica además de brindar mucha información útil previa a la reunión, es mucho más efectiva que poner el texto en la agenda de la invitación, ya que por lo general la gente no se detiene a leer el texto (sobre todo si ve que es muy extenso).

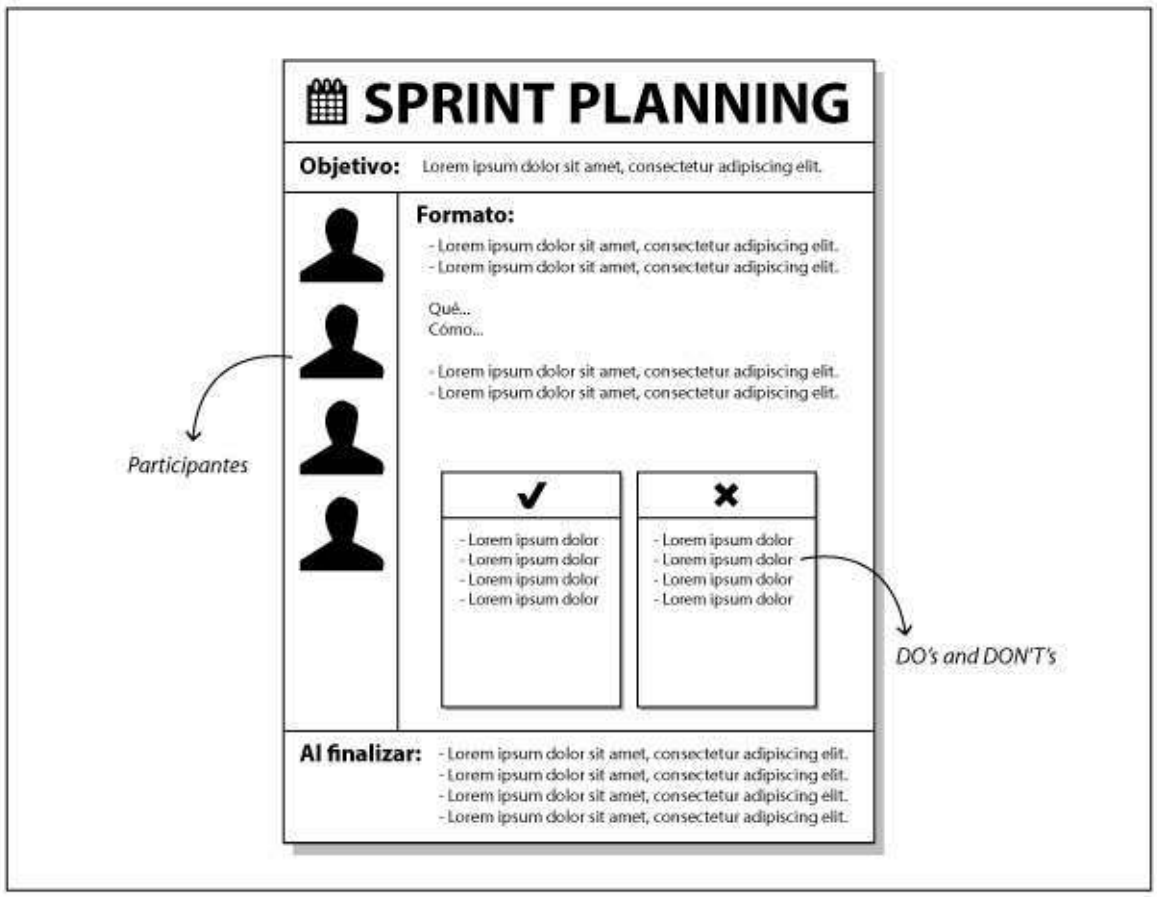

Figura 7.1. Plantilla para reuniones de planificación de Sprint

Cronomertar las reuniones: comúnmente en las reuniones dedicamos excesiva cantidad de tiempo a ciertas tareas o discusiones que no llevan a nada productivo por el simple hecho de que las personas pierden la noción del tiempo. Cuando definimos de antemano y de manera conjunta (por ejemplo por votación) el tiempo que le vamos a dedicar a cada actividad de la reunión y además hacemos explícito y visual el paso del tiempo es increíble como la situación cambia. Por ejemplo, en algunas reuniones de *Inception* hemos definido como equipo que no le dedicaríamos más de 15' a cada requerimiento para generar las *User Stories* iniciales. Esto hizo que las personas tomaran conciencia y optimizaran ese tiempo por el simple hecho de ver en un cronómetro el tiempo restante. Esto no significa que pasados los 15 minutos tengamos que terminar si o si la actividad, quizás solo puede servir para tomar conciencia de que estamos excediendo el límite y eso nos quitaría tiempo para analizar el siguiente requerimiento. Otro ejemplo es cronometrar las reuniones diarias de Scrum. Una vez excedido el límite de tiempo establecido, se puede dar por terminada la reunión o simplemente hacer saber que nos estamos excediendo. Eso dependerá de cada equipo. Lo importante es tener la noción del tiempo, tanto el destinado a cada actividad (*time-boxing)*, como el registro del tiempo transcurrido, siempre presente.

Distribución de roles y auto-organización de tareas: es importante repartir las tareas simples de forma tal de distribuir los esfuerzos y organizar rápidamente las reuniones. Por ejemplo, designar quién va a escribir o actualizar las User Stories, quien va a cronometrar

las actividades, quien va a registrar los impedimentos y las acciones a realizar, quien va a tomar notas para la minuta en caso de ser necesaria o requerida, quien va a facilitar la reunión, etc. Los roles y asignaciones pueden variar de persona de una reunión a otra y lo ideal es que las personas se auto organicen para distribuir la carga.

Creación de posters visuales: siguiendo con la idea que lo visual tiene un gran impacto, especialmente en organizaciones tradicionales y más conservadoras, una buena práctica es tener *posters visuales* que guíen la reunión y permitan tener el foco todo el tiempo en lo importante. En la figura 7.2 y 7.3 vemos ejemplos de posters que fueron creados para una reunión de revisión de Sprint (*Sprint Review*):

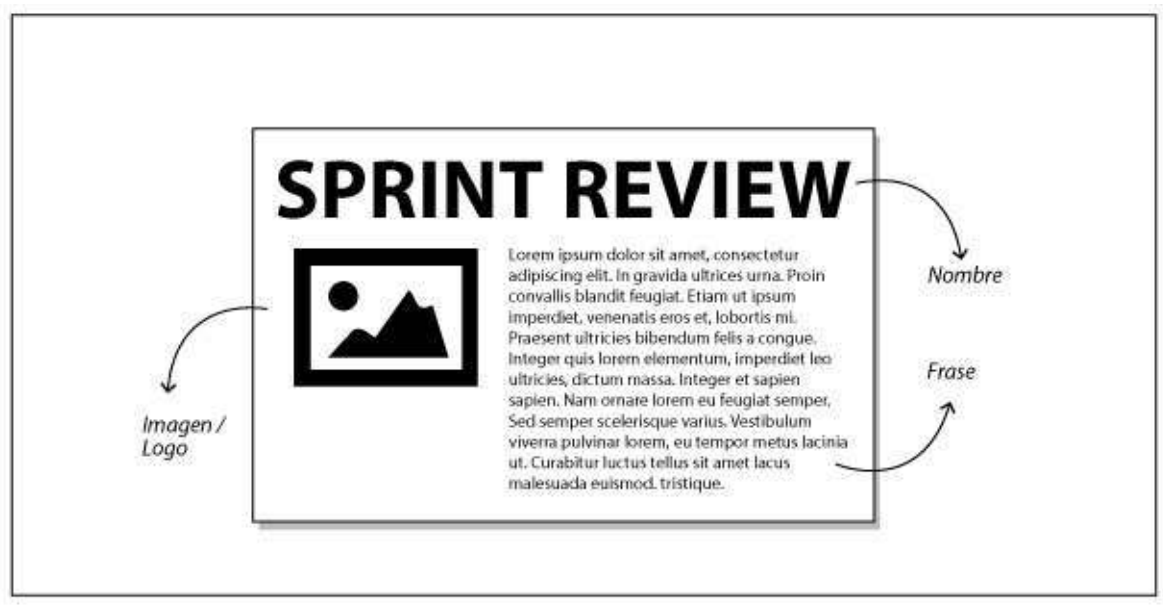

Figura 7.2. Poster digital para la ceremonia de revisión del Sprint

Estos *pósters* se pueden imprimir a color en el tamaño que nos parezca útil y pegarse en la sala de reuniones para que todos los vean.

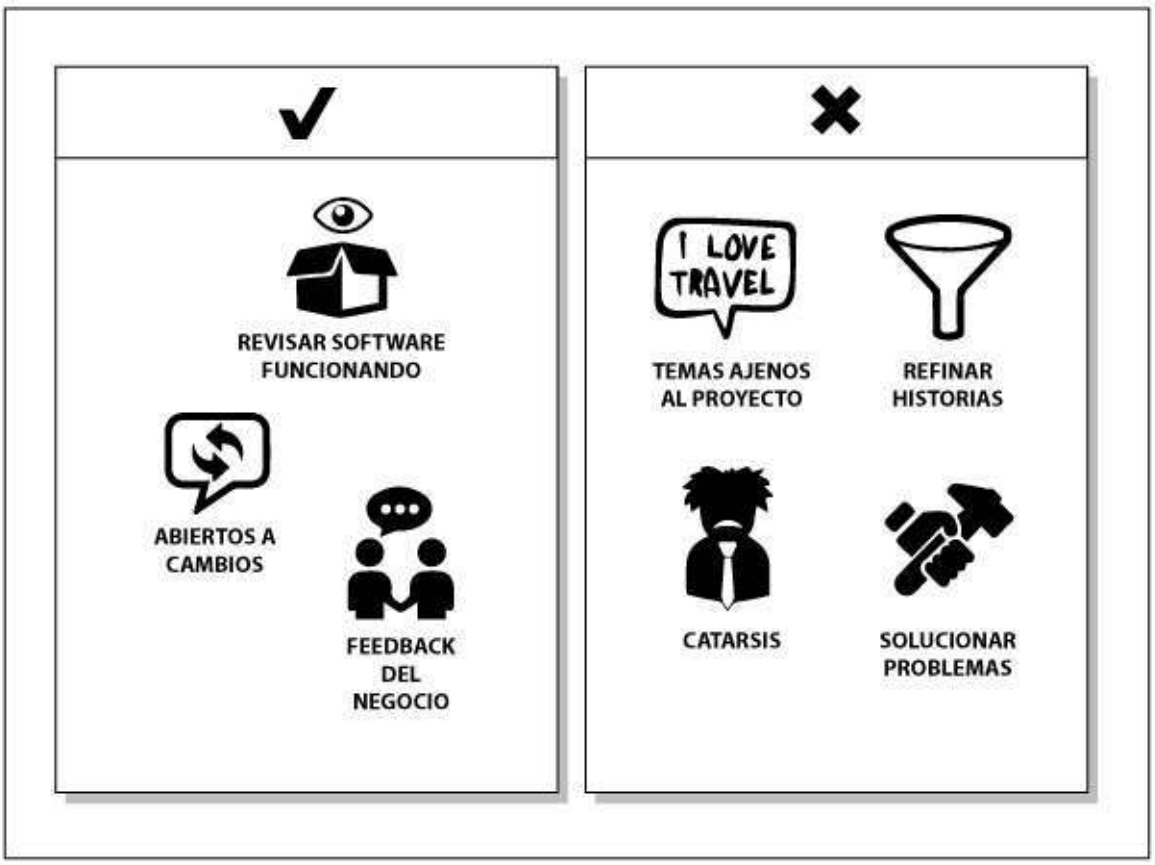

Figura 7.3. *Posters de "DOs & DONTs" para una reunión de revisión de Sprint*

Facilitación Gráfica: luego de empezar a usar pósters digitales nos fuimos dando cuenta que en muchas ocasiones realizar afiches gráficos o incluso facilitar las reuniones gráficamente "en vivo" tenía aún más impacto y captaba mucho más la atención de la audiencia. Con lo cual fuimos incorporando esta técnica para transmitir mensajes importantes o resaltar temas que queríamos que los asistentes se lleven incorporados.

En la Figura 7.4 podemos ver un ejemplo de afiche utilizado para describir a través de la facilitación gráfica los conceptos básicos de Scrum durante una reunión. Esta lámina fue creada "en vivo" y de forma colaborativa por los asistentes.

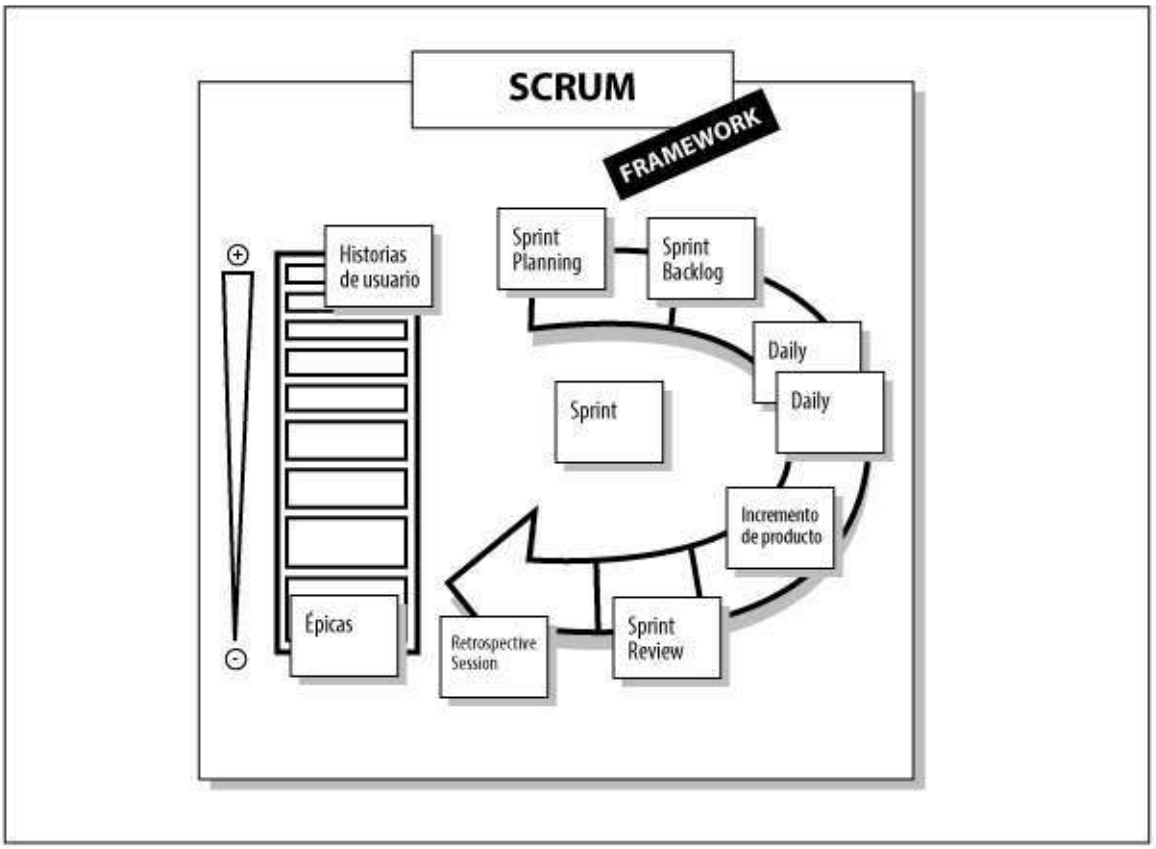

Figura 7.4. *Afiche descriptivo del proceso de Scrum*

Mini *kick-off*: en muchas actividades que involucran varias reuniones o tiempos más extensos, como puede ser la Sprint 0 o *Inception* de un proyecto ágil, optamos por realizar una reunión de kick-off de la actividad, en la cual no se habla de los requerimientos ni nada especifico del proyecto, sino que se describe que es lo que va a suceder en dicha fase o actividad en las próximas reuniones, semanas o incluso meses. Esto sirve para bajar la ansiedad de los asistentes, alinear expectativas, pulir la audiencia que va a participar y alinear la visión de lo que viene, como así también para que los asistentes entiendan el porqué de lo que estaremos haciendo, la importancia de hacerlo, los beneficios aparejados, las desventajas de no hacerlo y que sientan que las próximas reuniones serán una inversión y no un pérdida de tiempo.

Conclusión

Existen muchas prácticas simples que podemos realizar para optimizar el tiempo y realizar reuniones o ceremonias más efectivas. Y adicionalmente a la eficiencia y las prácticas eficaces, siempre es bueno tratar de generar un clima cálido y agradable en las reuniones. Si logramos tener reuniones eficientes y transitar las actividades inmersos en un clima de buen humor, lograremos que las personas sientan que se llevaron algo, invirtieron su tiempo y se divirtieron, con lo cual estaremos dando un paso firme en el afán de lograr que esas

personas vuelvan a participar con buena predisposición cuando sean convocadas. En definitiva un poco de eso se trata todo esto, lograr buenos resultados y a la vez compartir un buen momento en equipo.

Introducción a Visual Management

Por Soledad Pinter, @solepinter

Palabras clave

Visual management, kanban, tableros.

Intención

¿Cómo visualizar el estado de las tareas en curso? ¿Qué información es importante compartir con el equipo? ¿Cómo la compartimos?

Motivación

Cuando trabajamos en equipo, necesitamos conocer los diferentes estados de las tareas en las que está trabajando cada integrante. También necesitamos visualizar información valiosa e importante para todo el equipo y muchas veces no sabemos cómo, ni cuál.

Descripción

El visual management es un conjunto de técnicas de visualización para administrar información, en particular en este capítulo será en contexto del seguimiento de un proyecto.

Aquí encontrarás una lista de ideas que se pueden realizar para que los equipos cuenten con la información que necesitan visible cerca de su área de trabajo. Se trata principalmente del uso de láminas con diagramas UML, frases clave del proyecto en el que trabajan y láminas con gráficas acerca de temas importantes,como por ejemplo: decisiones de diseño, arquitectura, infraestructura. Todo eso, en las paredes visibles a todo el equipo. Y con colores que permitan visualizar rápidamente lo que es relevante.

Radiadores de Información

El término Radiadores de Información (*Information Radiators*) fue usado por primera vez en contextos Ágiles por Alistair Cockburn. Él se refería en particular a los *Taskboards*, Charts and Continuous Integration Build Health Indicator.

Todo tipo de recurso que puede ayudar al equipo a mejorar su colaboración y su comunicación los llamamos “Radiadores de Información”; en adelante “RI”. Son aquellos artefactos capaces de transmitir toda la información con solo pasar y mirarlos rápidamente. ¿Qué información irradian? Aquella específica que nosotros queremos que transmita.

La información será leída y tenida en cuenta por aquellos miembros del equipo cuando la necesiten. Lo importante es tenerla a mano y todo el tiempo disponible.

Además, tener información disponible y visible a todos, facilita la transparencia y la autoorganización.

Algunos consejos para lograr buenos RI. Éstos deben ser:

Accesibles. Deben estar ubicados en un lugar cercano al equipo, para que pueda ser visible por todos. Si lo ubicamos lejos o difícil de visualizar, pronto dejará de ser útil.

Simples. La información debe ser precisa, fácil de entender y sobre todo fácil de mantener actualizada.

Personalizados. Es importante que el equipo le encuentre sentido al RI, sino no lo van a cuidar, utilizar y con el tiempo lo van a dejar de usar.

A continuación enumeramos algunos RI que son fáciles de implementar rápidamente y ayudan a una mejor organización, coordinación y colaboración del equipo.

Tableros Kanban

Este es el tablero más famoso y utilizado en los equipos. Con el podemos visualizar el flujo del proceso y el estado de cada una de las tareas dentro del mismo.

Para identificar las tareas en el tablero utilizamos tarjetas, una por cada tarea. Por ejemplo se podrían utilizar notas autoadhesivas para cada tarea. Lo importante es que las tareas se irán moviendo entre las columnas.

La versión más simple para comenzar, es un tablero de tres columnas de izquierda a derecha: Para hacer, En Curso y Terminado. En la primera columna: “Para Hacer”, colocamos todas aquellas tareas que identificamos y aún no comenzamos. Las ubicamos priorizadas dentro de esta columna: las primeras o más importantes arriba, y hacia abajo las de menor prioridad.

En la columna ‘en curso’, ubicamos las tareas que actualmente se están trabajando. Y en la tercer columna ‘Terminado’ se ubican aquellas tareas que ya han sido finalizadas.

Kanban apunta a reflejar todo tu proceso sobre el tablero, para ello necesitamos conocer todos los pasos para concretar una tarea. De esta manera, entendiendo de manera holística el flujo de trabajo, podemos aventurarnos a superar la versión básica de 3 columnas y

organizar el tablero en nuevas columnas, reflejando en ellas cada etapa que genera valor dentro del proceso completo.

De esta manera podremos detectar fácilmente:

- Los cuellos de botella: aquellos pasos del proceso donde las tareas se retrasan o bloquean más a menudo.
- Limitación del trabajo en curso: no comenzar una nueva tarea hasta no terminar la actual. Disminuyendo así el tiempo basura entre cambios de múltiples tareas.

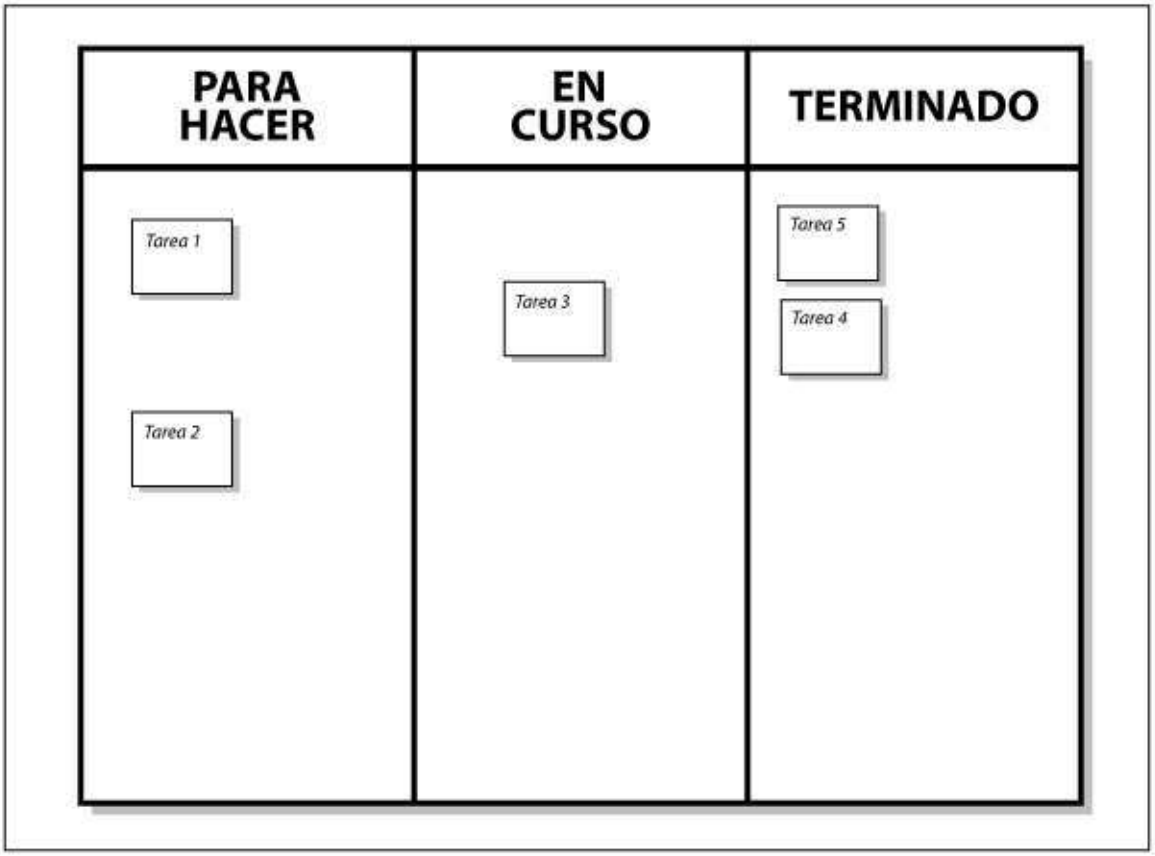

***Figura 9.1**. Tablero Kanban*

Una vez ya implementados, y luego de las primeras experiencias, se pueden aplicar estas sugerencias en el Tablero Kanban Avanzado:

- Tareas de un dia.
- Carril de emergencia: la primera fila del tablero la usamos para colocar aquellas tareas que entrar de emergencia al backlog, por ejemplo los errores en producción.
- Tags con nombres: utilizar notas autoadhesivas o algún identificador más pequeño para señalar los nombres de los responsables en la nota autoadhesiva de cada tarea.
- Fotos con los miembros del equipo.
- Métricas como el Leap Time y el Cycle Time permiten conocer, ajustar y mejorar los tiempos de desarrollo y entrega de una tarea, asi como tambien ayudan a identificar los

cuellos de botella en el proceso.

Calendario del Equipo

Si el equipo se autoorganiza para coordinar y organizar las vacaciones y ausencias planificadas (turnos médicos, preparación de exámenes, vacaciones), alcanza con imprimir o dibujar un calendario mensual en blanco en una hoja de papel. Luego, los miembros del equipo escriben su nombre en los días que estará fuera de la oficina, Pueden acordar un símbolo o color distinto para cada tipo de ausencia planificada por estudio, vacaciones, etc. También se pueden variar los colores o símbolos para cada integrante del equipo. Y en ocasiones, para evitar colapso de información podemos tener un calendario en la misma pared por cada integrante del equipo.

Este calendario debe estar visible junto al tablero de tareas o área de trabajo del equipo para que todos puedan verlo fácilmente.

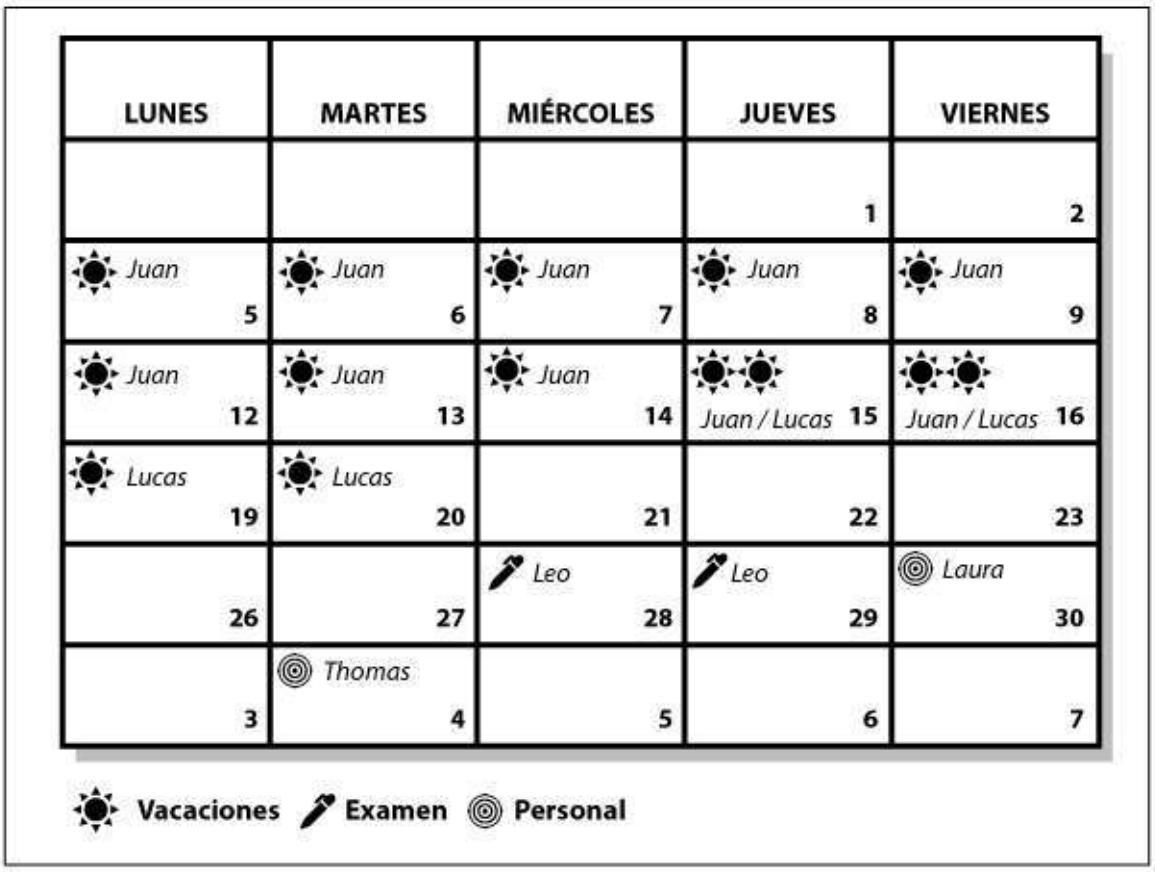

***Figura 9.2**.Calendario de Equipo*

Criterio de Terminado (*Definition of Done*)

Cuando el equipo trabaja con historias de usuario, necesita acordar los Criterios de Terminado, más allá de los Criterios de Aceptación de cada Historia de Usuario o Item del Backlog.

Para ello, se suele colgar una lámina con estos criterios listados, para que todos puedan tenerlos presentes siempre que necesiten recordarlos.

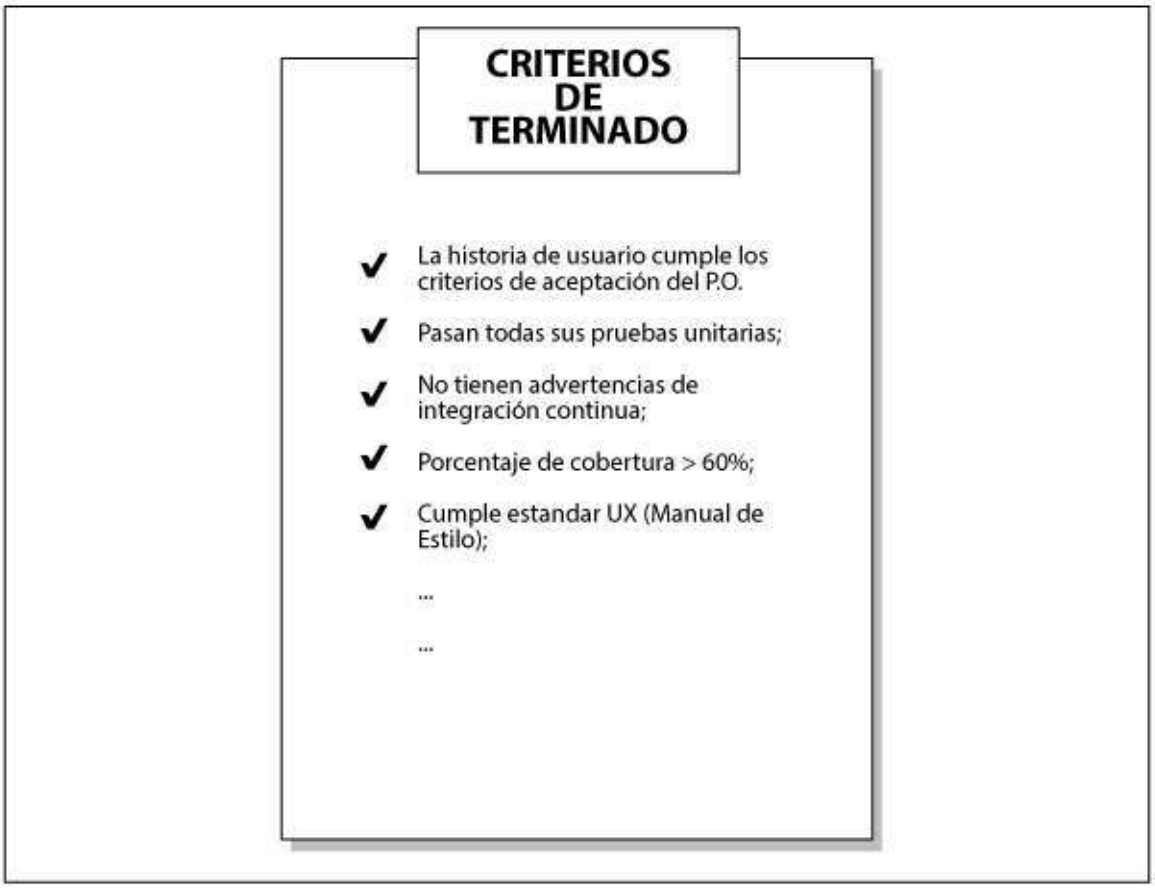

***Figura 9.3**. Criterio de Terminado*

Matriz de Programación de a Pares

Cuando un equipo realiza programación de a pares, es una buena práctica llevar registro de cómo lo hacemos para no repetir u olvidar alguna combinación. Para eso, dibujamos una matriz de doble entrada donde en las filas y columnas ponemos los nombres de los programadores del equipo, con esto logramos una cuadrícula de pares. Cada vez que dos miembros del equipo programaron juntos, colocan una cruz en la celda que intersecta sus nombres en fila y columna. De esta manera sabremos quiénes restan aún trabajar juntos.

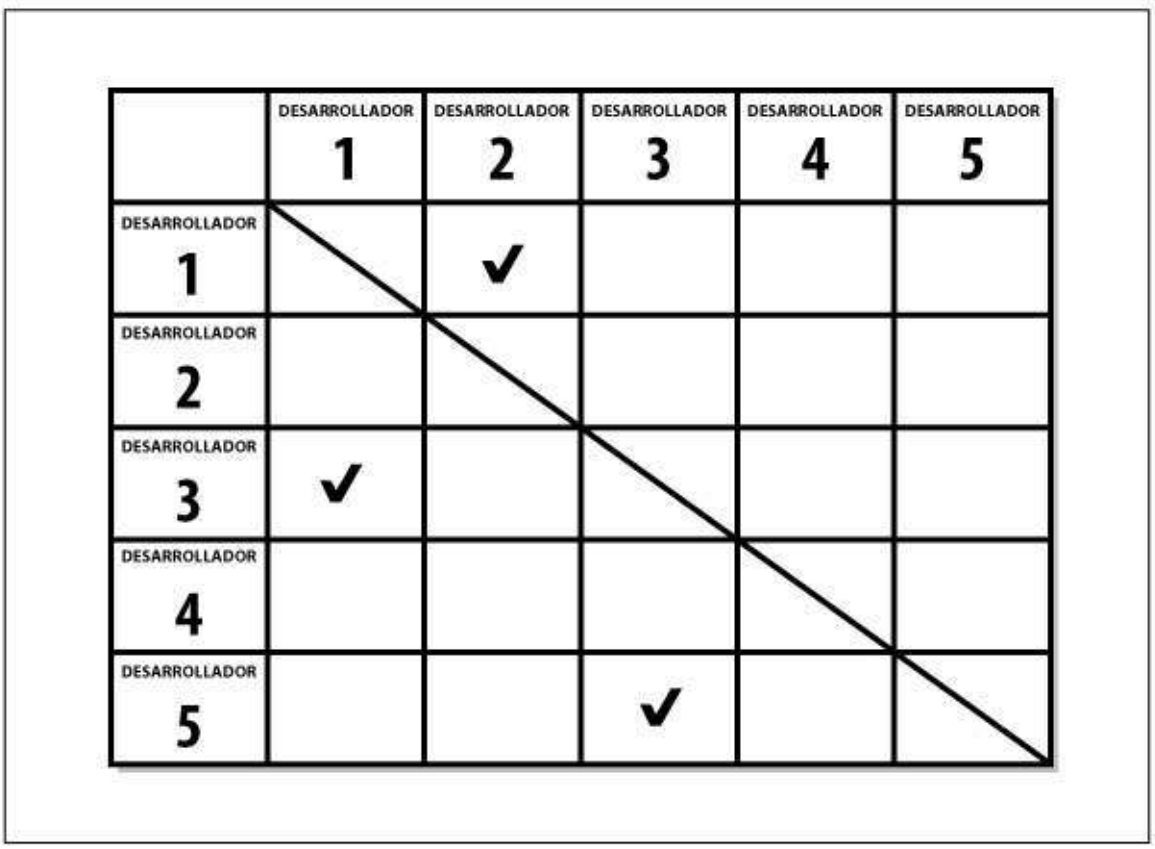

***Figura 9.4**. Matriz de Pair Programming*

Araña de Dependencias

Es una lámina donde dibujamos todas las dependencias que identificamos como aquellas que pueden condicionar la entrega de nuestro trabajo.

En la lámina ubicamos al equipo en el centro. Alrededor de éste, simulando las patas de la araña, todas aquellas fuentes de dependencias que pueden bloquear o pueden hacer peligrar la entrega a tiempo de las tareas del equipo.

A un costado de la araña replicamos estas fuentes en forma de lista bajo el título de “Resueltas”. Luego a medida que fuimos atacando cada dependencia acomodamos junto a cada fuente las tareas que se fueron desbloqueando. Esta métrica sirve para entender qué tipo de dependencias tiene el equipo y así poder pensar en acciones para poder removerlas.

Sprint Information

Cuando trabajamos con metodología Scrum, siempre es muy útil tener junto al tablero de tareas del equipo los datos importantes correspondientes al Sprint en curso. Sirve a modo de recordatorio a los miembros actuales y también para ayudar a nuevos integrantes.

En una hoja A4 escribimos aquellos datos que pensamos son complementarios y necesarios a la hora de transitar un sprint: Número de Sprint en curso, fecha de inicio y fecha de fin, Objetivo del Sprint.

***Figura 9.5**. Sprint Information*

Acuerdos de Trabajo

Cada equipo tiene su dinámica de trabajo. Siempre es bueno al comenzar, establecer entre todos los miembros del equipo, aquellas reglas o acuerdos que pueden ayudar a la convivencia y el desempeño del equipo. Por ejemplo: horario laboral, hora de la Daily meeting , cómo rotan la facilitación de las reuniones, a qué temperatura es aceptable el aire acondicionado y todas aquellas cosas que entre todos accedan a incorporar como "Acuerdo de trabajo".

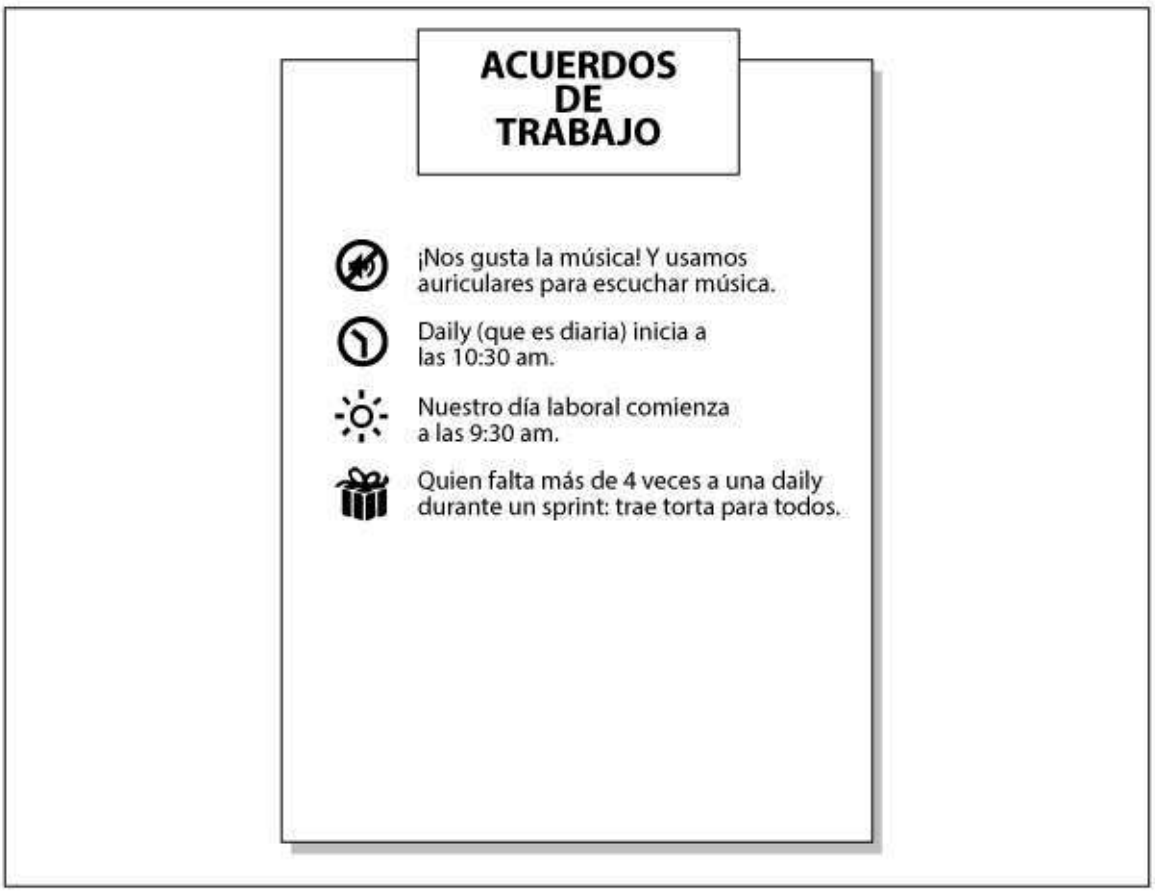

***Figura 9.6**. Acuerdos de Trabajo*

Revisión Triangular de Documentos

Por Natalia Baeza, @Naty3Baeza y Virginia Brassesco, @virbrassesco

Palabras clave

Revisión por pares, aprendizaje colaborativo, *feedback*, edición, documentación

Intención

Se presenta un patrón de revisión de documentos para tratar de evitar:

- Ambigüedades.
- Uso de vocabulario inapropiado.
- Redundancias.
- Textos innecesariamente extensos.
- Errores ortográficos.
- Estética general inadecuada para el contexto del documento bajo revisión.
- Uso de documentación obsoleta.

Este patrón facilita:

- El aprendizaje y maduración del tema a tratar.
- La comunicación entre las personas involucradas en la elaboración de documentos.
- Lla transmisión de conocimiento.
- La elaboración misma del documento.
- El mantenimiento de base de conocimientos documental.

Motivación

Este patrón resulta útil cuando un documento, público o privado, debe ser elaborado y se requiera *feedback* para mejorar la calidad del contenido, la redacción y la presentación del mismo.

También puede usarse cuando se desee evaluar y/o ampliar el conocimiento de una persona sobre un tema dado.

Descripción

Bajo el paradigma Ágil, el código funcionando tiene más valor que la documentación como se puede leer en el [manifiesto ágil]. En general hacemos documentos que refieren a algún producto de software. Sin embargo suele emerger la idea de que éste quedará obsoleto ante algún cambio en el producto, o que escribirlo nos haría perder tiempo productivo, y por eso la documentación no es prioritaria. Pero, ¿qué ocurre cuando el documento es el objetivo de nuestro trabajo, el producto final?

Muchas veces ciertos documentos deben poseer una estructura específica que podría introducir ruido al que intenta comprender una idea. Podría ocurrir que el autor principal no sea experto del dominio o su lenguaje nativo no sea el usado en el documento y, por ende, le podría faltar vocabulario específico.

Algo que solemos olvidar es que los textos deben ser comprensibles no sólo para su redactor, sino para el público al que están orientados.

Recordemos además que no a todos les gusta escribir y no siempre todos los que contamos con el conocimiento vamos a tener voluntad de trasladarlo a un texto publicable. Para que esta tarea no resulte tediosa, trabajar colaborativamente y tener una metodología que nos guíe puede ser el puntapié inicial.

Partiendo de la bien conocida idea de revisión de pares [Crespo-Villena 2005], aplicada a la elaboración de textos, se propone un patrón de revisión triangular como estrategia de aprendizaje y así lograr mejores documentos que faciliten la tarea de compartir conocimiento.

Aspectos que matizan al patrón:

- *Roles y tareas:* Identificar las funciones de los actores participantes con su tipo de revisión asociada; separar revisiones conceptuales de las ortográficas y gramaticales.
- *Expertise:* Acceder a “diversidad” de personas que estén calificadas para hacer tu revisión.
- *Workflow:* Contar con un método práctico para seguir el estado del documento.
- *Soporte a la comunicación:* Herramientas que faciliten la visualización de cambios y el estado de la revisión.

Roles

Son tres (así naciendo la idea de triángulo), el *autor*, el *revisor* y el *editor*.

El *autor* es quien sabe del tema y quiere exponerlo al mundo, entiende bien de qué se trata y cuál es el alcance del documento. Identificó la necesidad de escribir sobre ello y tiene la voluntad de hacerlo.

El *revisor* es el experto, sabe más o igual del tema que el autor y es capaz de decidir si el autor se está expresando bien sobre aquello que quiere contar. Conoce sobre el vocabulario del dominio y los detalles de qué es bueno decir y qué no, para cada caso.

La pregunta es, ¿por qué no lo escribió él al documento?

Todo documento surge por una necesidad, pues si no hay necesidad no hay motivos para hacerlo ya que no le aporta valor a nadie. Esa necesidad fue identificada por el autor, quien es el que conoce la intención de plasmar el tema en un documento.

El *editor* es quien se encarga de darle formato y revisar ortográfica y gramaticalmente el texto. Esto es tan importante como el contenido. Nadie quiere leer un documento con errores de ortografía, mal redactado o donde los colores cansan la vista. Por otro lado, puede ser interesante y hasta necesario en ciertas ocasiones, saber a quién pertenece el documento, introduciendo en él algún símbolo que lo identifique: el autor, la compañía o el nombre del producto (sea comercial o no). El editor debe revisar también que el documento tenga la información necesaria para la publicación, sea agradable a la vista, y “que cuente bien la historia”.

Tareas, ¿Quién se ocupa de qué?

El autor y el revisor evalúan básicamente *contenido*, definen el *estilo* de la escritura y parcialmente formato a sólo efecto de no agregar confusión en el documento por ejemplo identificando jerarquías de títulos o definiendo colores que no pueden ser de otro modo. Además, el editor debe saber exactamente a qué le está dando formato, por ende el autor debe entregar un documento con ideas claras al editor.

El rol del editor es muy distinto al del revisor. Él estará concentrado en los aspectos gramaticales, formato y estilo visual del documento. Será el encargado de dejar la versión final publicable. Si es un experto en el idioma de redacción del documento, puede aportar en mejoras del estilo de escritura.

Expertise

Escribir sobre un tema implica entenderlo en plenitud. Solo puedes explicarlo bien si puedes entenderlo muy bien. La revisión de un experto en etapas tempranas ayuda al aprendizaje y balance de conocimiento entre el autor y el revisor. Así el texto es finalmente construido con el conocimiento de las partes involucradas, consensuando los alcances del documento actual y decidiendo qué documentación elaborar en un siguiente ciclo de revisión.

Cuando el revisor cuenta con mayor conocimiento específico de ese dominio que el autor, se debe asegurar de que el texto final sea comprensible para el público al que está destinado.

Si en tu equipo diario no encuentras a alguien cercano que cuente con el *expertise* para hacer tu revisión, debes buscarlo en otro lugar, si no deberás hacerlo tú mismo y el resultado será carente de objetividad y no tendrás *feedback*.

Pero…, ¿qué pasa si el revisor tiene menor *expertise* que el autor? ¿Es factible?

La revisión que se pretende incorporar en este patrón tiene intención de ser vista como un método de aprendizaje para todos los que escriben en él, no sólo para el autor.

Alguien que no domine el tema en plenitud puede aportar valor. Es importante que pueda tener lugar para formular todas las preguntas necesarias y así comprender lo que está leyendo. A través de estas preguntas el autor puede detectar qué información es faltante o poco clara en el documento.

En este caso, el experto es el autor, y el revisor se introduce en un *workflow* de revisión sobre algo pre-elaborado. Notemos que en este caso la confianza del revisor al autor para poder realizar preguntas y del autor al revisor para dar lugar a edición es fundamental. Motivemos al revisor a preguntar, pero también a reformular textos para hacerlos más comprensibles.

Este tipo de revisión conviene hacerla en etapas avanzadas del documento, cuando éste ya fue revisado en al menos una iteración por un revisor experto y el editor. Esa persona debe estar preparada para modificar un documento que trata de un tema en el que no es experta, por lo cual no debe ser elegida al azar.

Por supuesto, se puede contar con varias personas para el mismo rol: puede haber más de un autor, más de un revisor y más de un editor.

Con respecto al *expertise* del editor, debe conocer bien las reglas del lenguaje en el que se escribe el documento. En el caso que el documento deba publicarse en un idioma diferente al usado por el autor, el editor debe contar con conocimientos expertos en ambos lenguajes, y hasta podría realizar una traducción del mismo, o crear las versiones necesarias del documento en diferentes idiomas.

Por otra parte, si el diseño del documento es muy complejo, se podría recurrir a algún diseñador que colabore agregando estilos gráficos, corrigiendo imágenes, y dando el marco acorde que identifica al documento como parte de una base de conocimiento específica.

Comunicación fluida

La interacción entre las personas que redactan y editan el documento es clave, debe ser fluida. Esto no implica tener reuniones constantemente para poder redactar cada línea; aquí prima la confianza en nuestros pares, donde el grupo de revisión está enfocado a lograr un producto de calidad, construyendo y aprendiendo en el camino.

Antes de comenzar la elaboración del documento hay que elegir las herramientas que soporten esta comunicación.

Un ejemplo muy práctico es la plataforma de Google, Google Docs. Lo valioso de esto es que permite volver atrás cualquier cambio realizado e identificar el momento y el autor del cambio. Por supuesto esto no es para identificar culpables, sino para acudir a esa persona y entender el cambio si es que no está claro.

Aquí es necesario comunicarse en tiempo real. Pueden ser útiles herramientas de *chat*, o llamados telefónicos. El objetivo es visualizar la traza del cambio, aclarar las dudas de inmediato y decidir y resolver en función de ello.

Workflow: Revisión iterativa en ciclos cortos

Para no caer en el cansancio o creencias de que se “pierde tiempo” en un documento, la redacción de documentos debe ser de manera iterativa, con el cubrimiento de todos los roles aportando cambios incrementales pequeños, con lo cual se requiere menos tiempo para tomar una decisión sobre ellos, y por ende son más simples de incorporar (descartar) en el documento, o bien crear nuevos documentos a partir de ellos. Además así se mantienen actualizados a lo que realmente ocurre en ese momento.

Es común que al elaborar un documento hagamos referencias a otros existentes ya publicados. Si se descubre un documento obsoleto, es necesario actualizarlo de inmediato resolviendo lo que esté desactualizado. Si el documento es completamente obsoleto es recomendable hacer “borrón y cuenta nueva”.

Para poder hacer un seguimiento de documentos publicados se recomienda que cada documento tenga una fecha o versión de publicación que coloque el editor final.

Aún así, ¿quién empieza?

El autor es el que da el paso inicial. Manifiesta ideas lo más claramente posible y en un muy corto plazo, por ejemplo 1 día o 2, el revisor ya se hace parte del documento.

Alinea los tópicos, pide explicaciones verbales sobre los mismos (esto puede implicar una reunión inicial) y da lugar al autor con indicaciones, ideas, ayudas fuertes de cómo seguir. El resultado es un acuerdo entre ambas partes.

Si al escribir cualquier cosa del documento hay un “impedimento” se debe acudir de inmediato al revisor o viceversa (revisor a autor).

Y aquí hemos trazado un canal de ida y vuelta en la comunicación dando forma a un lado de nuestro triángulo, Figura 10.1.

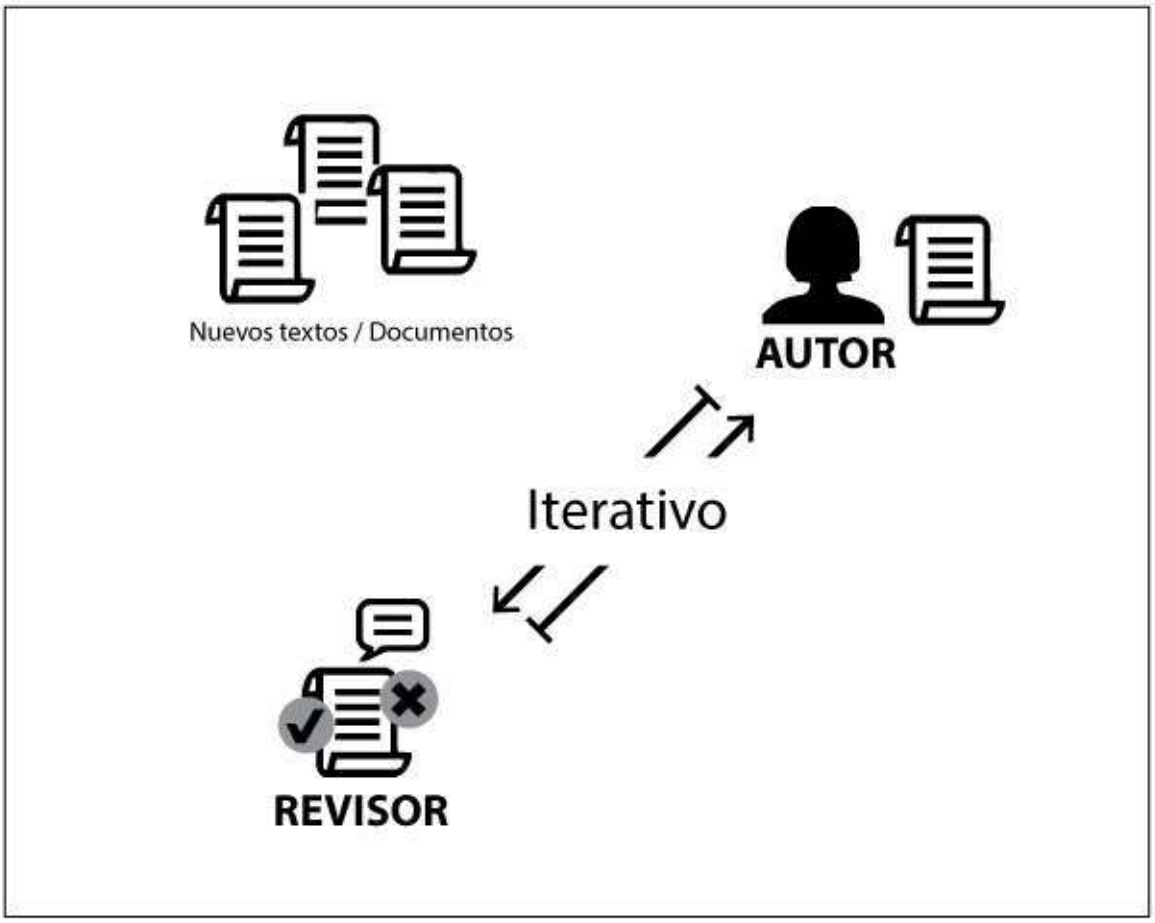

Figura 10.1.Interacción autor-revisor

En esta etapa de la revisión pueden surgir documentos anexos, secciones no previstas o nuevos documentos a generar a posteriori. Hasta aquí es claramente una revisión de pares.

En estas dos instancias, el editor cumple un rol pasivo. Se le pueden preguntar cuestiones puntuales pero aún no edita el documento.

Cuando ya hay una versión lo suficientemente madura del documento que contenga las ideas principales del mismo y que estén explicadas de forma clara para autor y revisor, se hace parte activa del *workflow* el Editor.

Es posible que el editor no requiera ayuda del autor o revisor para realizar su tarea, pero si hay algo de fondo que se deba cambiar o alguna idea no está clara, el editor debe consultar al autor.

Aquí ya tenemos dos aristas del triángulo

Figura 10.2. Interacción autor-editor

Si el autor no es capaz de responder a la pregunta o no está disponible en ese momento, el editor debe acudir al revisor.

La tercer arista es más liviana (editor al revisor y viceversa) ya que quien intermedia cada revisión en el *workflow*, es el autor.

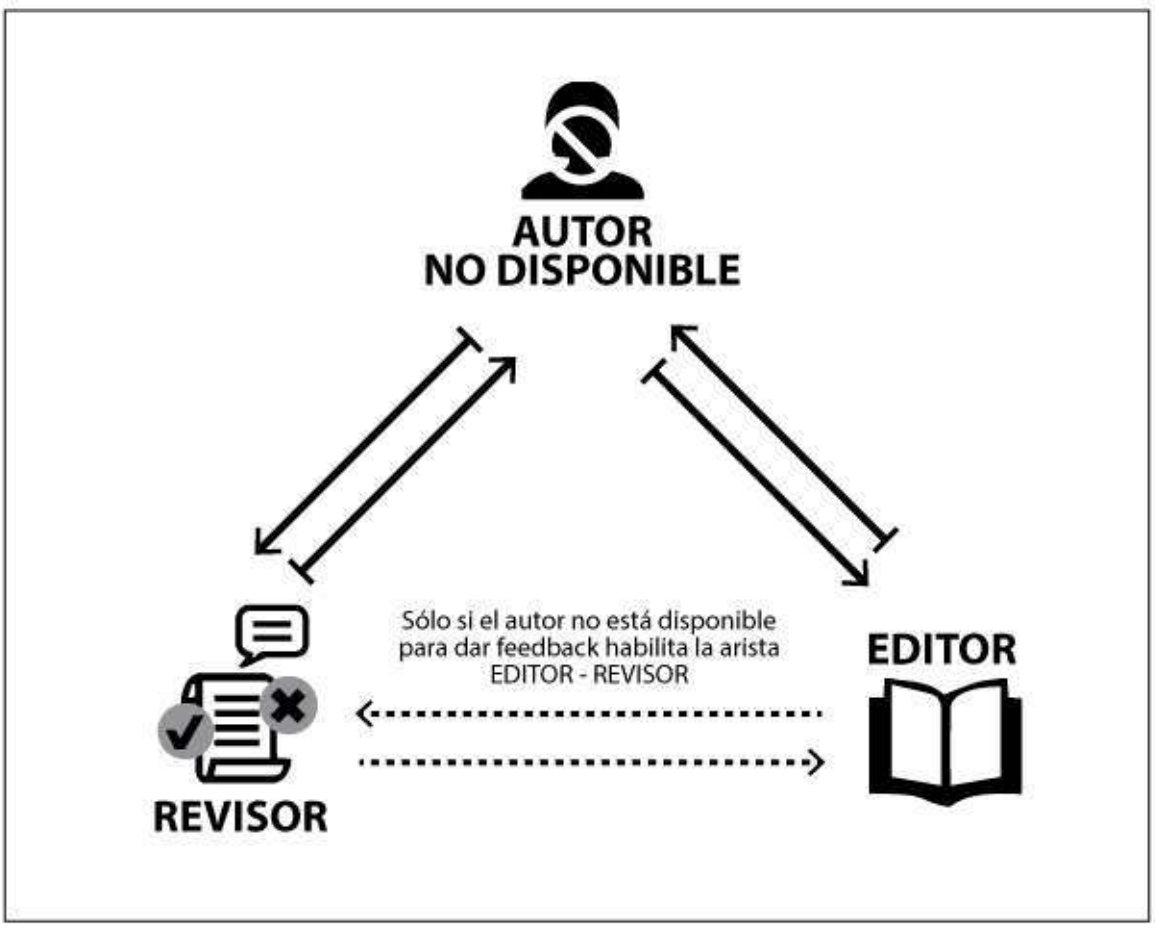

***Figura 10.3**. Interacción editor-revisor*

Este ciclo de 3 etapas, debe repetirse cuantas veces sea necesario.

Durante la generación del documento la revisión se realiza de manera cruzada e iterativa hasta tener la versión final que también es revisada por el autor previa a su publicación.

Otros detalles

Cada equipo de revisión debe generar su propio mecanismo para saber cuándo le toca a quién (siguiendo el *workflow* anterior) y no pisarse en los cambios. No es recomendable que revisor, editor y/o autor hagan cambios en simultáneo sobre la misma parte del documento. Cada uno debe decir explícitamente si el documento completo o qué parte del él está lista desde su perspectiva.

Otro punto inflexible es que **aquel que decide que algo debe escribirse de otro modo, borra lo escrito y escribe su idea. No pide permiso.** Esto puede sonar agresivo pero no lo es y el resultado es favorablemente bueno, ya que quien escribe, lee y elabora su idea sin pensar en una frase o concepto que ya decidió que debía reformularse. La revisión y elaboración de la revisión es un proceso de aprendizaje. Tomo mi mejor decisión, aprendo y mejoro mi decisión en la siguiente iteración.

Por esta razón es fundamental que la herramienta de soporte de la edición, guarde las versiones y los cambios con autor y *timestamp*. De ese modo es posible tener y visualizar la traza a versiones anteriores y resolver dudas aclarando ideas nuevas.

Conclusiones

Este patrón se propone como un mecanismo de aprendizaje extendiendo la idea de revisión de pares. Se definen 3 roles principales, cada uno con su expertise aportando valor en un proceso iterativo de revisión.

A la hora de implementarlo seamos concientes que los participantes van a ser enriquecidos en conocimiento y nos ayudará a mantener la información actualizada no solo escrita, sino también en las mentes de todos los participantes.

De este modo, se facilita la elaboración de documentos que soporten la gestión del conocimiento a partir de un documento de calidad, dividiendo y distribuyendo la revisión en pequeñas tareas y actuando de forma colaborativa. Aplicado el patrón, el producto debe lograrse en cortos períodos de tiempo dando valor y resolviendo una necesidad actual de aprendizaje y divulgación, e incrementando una base de conocimiento que intenta mantenerse actualizada.

Agradecimientos

Este capítulo se ha realizado utilizando este patrón de revisión triangular a través de 2 iteraciones. Agradecemos a Pablo Tortorella y Nicolás Páez, como revisor y editor respectivamente, por su colaboración en la primer iteración de revisión triangular que dio forma a este documento.

SEF: Sesión Exprés de feedback

Por Leonardo Barrientos Silva, @leobarrientos

Palabras clave

retrospectiva dual, personas, feedback, 5 minutos, retroalimentación.

Motivación

Los procesos de maduración de equipo son, bajo mi punto de vista, muy distintos y pueden o bien encontrar su óptimo al cabo de un par de *Sprints* o requerir un tiempo no menor de conocerse y generar confianza. Recordemos que todo es cuestión de relaciones humanas y comunicación.

Valores como el compromiso, coraje, foco, respeto y apertura no emergen de forma espontánea de un equipo sin conocerse. Si bien cada integrate del equipo puede valorar más uno que otro y actuar como quiera, claramente no está asegurado que el equipo tenga el compromiso de entregar valor al cliente de forma temprana, iterativa e incremental.

Lo que queremos es llegar a un ritmo que permita transmitir tranquilidad a nuestros clientes para bajar la ansiedad de que el equipo ágil esté en su desempeño óptimo lo antes posible, además de ser flexibles, y por sobre todo desarrollar de forma incremental con calidad, *Time to Market* y con un presupuesto controlado.

Fomentar el sano vínculo dado por relaciones humanas sólidas permitirá llevar un proceso de maduración más tranquilo, sostenible y por sobre todo defendible ante el cliente.

Descripción

Se presenta una actividad que permitirá de forma clara entregar feedback a los miembros del equipo en forma objetiva, respetuosa, certera, oportuna, rápida y concisa para corregir un comportamiento o acción que provoque un declive en el desempeño del equipo, por ejemplo alguna mala práctica de desarrollo, uso de algún patrón, antipatrón de diseño. Es necesario potenciar el proceso creativo de desarrollo de software.

Comúnmente los miembros del equipo no saben cómo comunicar a sus pares los defectos que han encontrado en sus desarrollos o cómo enfrentar las situaciones post crisis esto es, la corrección de un incidente en producción producto de un mal patrón de diseño por ejemplo.

La transparencia y la confianza son claves al momento de construir lazos en los equipos.

La actividad se denomina Sesión Exprés de feedback (SEF) y está enfocada a que dos miembros del equipo se comuniquen de tal forma en que uno de ellos da el feedback y el otro escuche, analice y se propongan acciones de mejora.

La actividad se divide en cuatro fases:

- La primera fase la inicia la persona que quiere dar feedback y consiste en determinar a quién se le dará, cuál es la problemática a comunicar y los hechos concretos.
    - Para mostrar claramente la problemática y hacer que la reunión sea expres y no desviarnos en la conversación con cualquier cosa, nos apoyaremos con notas autoadhesivas para anotar e individualizar los hechos y objetivos.
    - Luego se deben priorizar y elegir un solo una nota autoadhesiva. El cual será el foco de la reunión.
- Comunicar y Agendar. Es básicamente concertar e invitar a la persona a una SER.
    - Se puede enviar una cita electrónicamente o directamente se solicita cara a cara. Se debe privilegiar e incentivar la solicitud verbal cara a cara para favorecer los lazos humanos.
- Ejecutar la SEF: es la reunión en donde daremos el feedback.
    - Los primeros minutos deben destinarse a contextualizar el tipo de feedback.
    - Mostrar las notas autoadhesivas a conversar
    - Hablar siempre calmado y concentrado.
    - Exponer los hechos e interpretaciones para demostrar que te importa el hecho a discutir. Hablar desde las emociones.
    - Pedir que és lo que se quiere.
- Cierre y Compromiso, etapa que busca:
    - Conocer la opinión del otro.
    - Formular acciones que conduzcan a mantener y mejorar los hechos.
    - Formular acciones que generen un compromiso de cambio.

Improvement Kata

Por Hiroshi Hiromoto, @hhiroshi

Palabras clave

mejora continua, generación de valor, auto-organización, kaizen, lean

Intención

En muchas organizaciones algo recurrente es el deseo por ser mejores en algún aspecto, ya sea en reconocimiento, en ganancias, en la calidad de su producto o felicidad de sus colaboradores. Algo igual de recurrente, independientemente de sus desafíos, es que existe una gran dificultad para convertir esos deseos en mejoras concretas y una mayor dificultad aún, para sostenerlas en el tiempo.

Improvement Kata es un patrón que nos ayuda a lograr convertir esos deseos en mejoras concretas.

Motivación

Si bien Improvement Kata puede ser utilizado a diferentes niveles de una organización (C-Level, *middle management*, áreas, equipos, etc), este capítulo estará enfocado en su uso a nivel de equipo; en particular en equipos que se desenvuelven en el trabajo del conocimiento.

En estos equipos, independientemente de si están desarrollando software, definiendo estrategias, creando campañas de marketing, transformando una organización o diseñando un servicio, existe un alto grado de incertidumbre en cómo superar los desafíos a los que se enfrentan. Esta característica en particular hace que la posibilidad de crear una planificación prescriptiva para afrontar sus desafíos sea poco realista, y que aparezca la necesidad de utilizar un marco que pueda ser adaptativo.

Por ejemplo en el *Agile Manifesto* [Manifesto 2001], la necesidad de poder generar mejoras a la forma de trabajo de forma adaptativa se ve reflejada en el siguiente principio ágil:

“A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia.”

Donde el uso de intervalos de tiempo, permite hacer ajustes y reflexionar sobre los cambios que se vienen adoptando.

Asimismo, si revisamos la definición [Scrum Guide 2016] de la reunión de retrospectiva en Scrum (una práctica que se basa en el principio que acabamos de revisar), nos encontramos con:

“La Sprint Retrospective es una oportunidad para el Scrum Team para inspeccionarse y crear un plan de mejoras a ser ejecutadas durante el siguiente Sprint.”

Donde los intervalos regulares han sido reemplazados por el Sprint y se hace más foco en la inspección que en la reflexión. Además, al ser una reunión, la incorporación de las mejoras queda fuera del alcance de la misma (dado que se realiza durante el Sprint).

Si tenemos en cuenta que la retrospectiva es la práctica más utilizada [ScrumAlliance 2015] por los practicantes de Scrum (80%), siendo este el framework más utilizado en el mundo ágil, uno podría pensar que muchos de los equipos ágiles están constantemente moviéndose hacia el logro de sus desafíos.

Lamentablemente, en mi experiencia, pocos son los equipos que están logrando mejorar continuamente con cadencia y resultados tangibles. Muchos podrán argumentar que la mayoría de retrospectivas están llenas de anti-patrones que las hacen inefectivas (con lo que estoy de acuerdo), pero creo que no es la única razón por la que no se ven resultados.

Mi hipótesis (que he ido validando con equipos a los que acompaño) es que hay tres elementos que hacen falta en la mayoría de sesiones de mejora para que estas sean efectivas.

- El primer elemento es la ausencia de un desafío claro y preciso, que vaya mucho más allá de “ser más efectivos” (como dice el principio ágil), y que permita generar foco en las iniciativas de mejora que nos lleven de la pregunta “qué podemos mejorar” a la pregunta “qué debemos mejorar”.
- El segundo elemento es un mecanismo que nos ayude a recorrer el camino hacia ese desafío. Que evite que tengamos sesiones Whack-a-Mole (como el juego de arcade), en donde simplemente estemos reaccionando de forma instintiva ante lo que aparece al frente de nuestros ojos, sin tener en cuenta todo el panorama ni las causas raíces de los problemas.
- El tercer y último elemento es la experimentación sobre los planes de acción como indica Hiromoto [Hiromoto 2014], que permite hacer foco en los impactos que queremos generar y amplifique el aprendizaje cuando no logramos esos impactos con nuestras hipótesis.

Si bien es cierto que he visto equipos aplicar estos tres elementos en una retrospectiva, me parece más natural utilizar un patrón que ya las contenga por diseño, y es ahí donde entra Improvement Kata.

Descripción

Antes de profundizar en Improvement Kata, quisiera comentar brevemente sobre el concepto de la *kata*, ya que da las bases para entender la importancia del patrón.

Kata

La kata es un concepto que viene de las artes marciales. De forma general es un patrón que repites continuamente de forma deliberada para generar memoria muscular y luego realizarlo casi sin pensar. Por ejemplo en la película Karate Kid [Karate Kid 1984], Daniel repetía el movimiento de *encerar y pulir* para tenerlo interiorizado y poder utilizarlo en la pelea de karate de forma natural, casi automática.

En el caso de Improvement Kata, este es un patrón que repetimos para generar una memoria muscular de mejora, de forma que se vuelva un hábito.

Improvement Kata

Improvement Kata [Rother 2009] está enfocado en ayudarnos a dar pasos que nos permitan ir acercándonos al desafío mejorando continuamente. A grandes rasgos tiene dos etapas: una de planificación y otra de experimentación, que se repiten cíclicamente.

Etapa de planificación

En esta etapa se genera la parte estratégica de Improvement Kata, así como el seteo inicial que nos permitirá experimentar. Existen cuatro elementos a tomar en cuenta:

El desafío

El desafío es aquel reto, alineado con una visión, que tiene el equipo y es el motivo por el cual se está usando Improvement Kata. Uno puede decir que ese desafío es relevante cuando cumple las siguientes tres características:

- Dirección: Brinda una dirección al equipo, que les permite avanzar.
- Tensión positiva: Genera una tensión positiva hacia el desafío, parecida a la fuerza de la gravedad. Ya que la forma en cómo hacemos las cosas (*status quo*) genera una tensión que evita cambios, la tensión del desafío debe ser superior para permitir que el equipo se mueva.

- Significado de ganar: Tener un desafío claro nos permite entender qué significa ganar en nuestro contexto, dentro de la definición de *ganar* de Kofman [Kofman 2006].

Situación Actual

La situación actual son los hechos y datos que te dicen dónde está el equipo hoy en día en función del desafío. La situación actual usualmente tiene varios componentes como:

- Diagrama de bloque del proceso: Una representación de cómo se hacen las cosas hoy en día.
- Métricas de proceso: Indicadores claves de performance que nos permitan evaluar cómo funciona el proceso.
- Métricas de resultado: Indicadores claves de performance que nos permitan evaluar el resultado generado.
- Características del proceso: Datos adicionales contextuales que nos ayuden a entender mejor la situación.

Siguiente Condición Objetivo

La siguiente condición objetivo es la descripción de dónde quiere estar el equipo en un determinado periodo de tiempo. Funciona como punto intermedio entre la situación actual y el desafío. Además nos permite enfocarnos entre lo que podemos hacer y lo que tenemos que hacer.

La siguiente condición objetivo usualmente tiene varios componentes como:

- Fecha límite: Momento en el tiempo en donde el equipo espera estar en la situación descrita.
- Diagrama de bloque del proceso: Una representación de cómo se harían las cosas.
- Métricas de proceso: Números que deberían alcanzar los indicadores claves de performance que miden el proceso.
- Métricas de resultado: Números que deberían alcanzar los indicadores claves de performance que miden el resultado generado.
- Características del proceso: Datos adicionales contextuales que deberían ser diferentes en la situación descrita.

Algo a tener en cuenta es que una siguiente condición objetivo no requiere cambios en cada uno de los componentes descritos. Esto significa que un equipo puede plantearse una siguiente condición objetivo modificando únicamente el valor de un indicador y dejando el resto tal como está.

Obstáculos

Los obstáculos son una lista de elementos que te impiden estar en la siguiente condición objetivo. Estos pueden tomar forma de problemas, impedimentos o contextos.

Etapa de Experimentación

En esta etapa es donde ocurre toda la magia y es el momento en dónde se ponen a prueba las hipótesis y se genera el aprendizaje.

Ciclos PHVA

Una vez que están identificados los obstáculos se procede a elegir uno y tratar de removerlo a través de experimentos. La ejecución se realiza siguiendo ciclos de PHVA (Planear, Hacer, Verificar y Actuar), en donde primero se define un experimento que incluye la generación de una hipótesis y su resultado esperado. Luego de la definición del experimento, se procede a ejecutarlo, para luego revisar los resultados y capitalizar el aprendizaje.

Algo importante a notar es que Improvement Kata promueve el uso de los experimentos de factor único (*single-factor experiments*), que implican hacer un solo cambio a la vez, lo que permite capitalizar el aprendizaje de forma más efectiva.

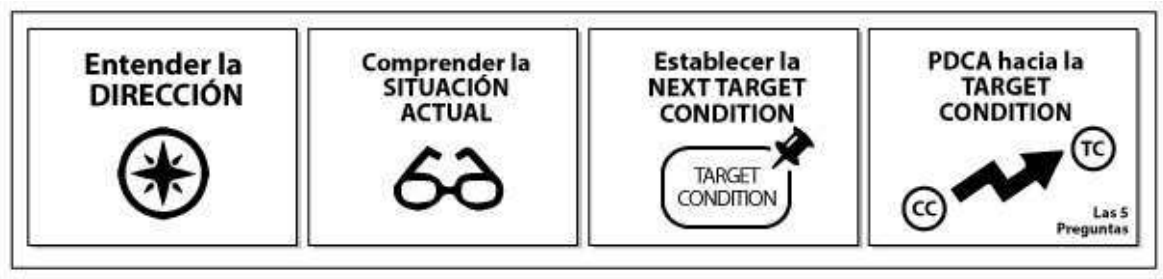

***Figura 12.1**. Improvement Kata*

Ejemplo de aplicación de Improvement Kata

Para hacer más didáctica la explicación usaré un ejemplo simplificado de aplicación en un equipo de desarrollo de software.

Les presento a Juan, Karl, Martín, Carlos y Lisbeth (Team Leader). Ellos son un equipo de desarrollo que trabaja desarrollando y dando soporte a la aplicación de ventas de su organización.

***Figura 12.2**. El equipo*

En la empresa donde trabaja el equipo, el área de tecnología tiene una visión común que es:

“Ser un brazo estratégico del desarrollo del negocio a través de tecnología."

El desafío

Con la visión en mente tanto Lisbeth como el director de IT llegaron a un consenso sobre el desafío que tienen tomando en cuenta la visión del área:

“Entregar software de valor continuamente con menos del 5% de funcionalidades con bugs en producción por release."

***Figura 12.3**. El desafío*

Situación Actual

Con el desafío claro, Lisbeth se reúne con su equipo en el lugar del trabajo, para que juntos determinen la situación actual.

Ellos actualmente utilizan un tablero kanban y tienen algunas métricas tanto del proceso como del resultado. Basado en eso la condición actual es:

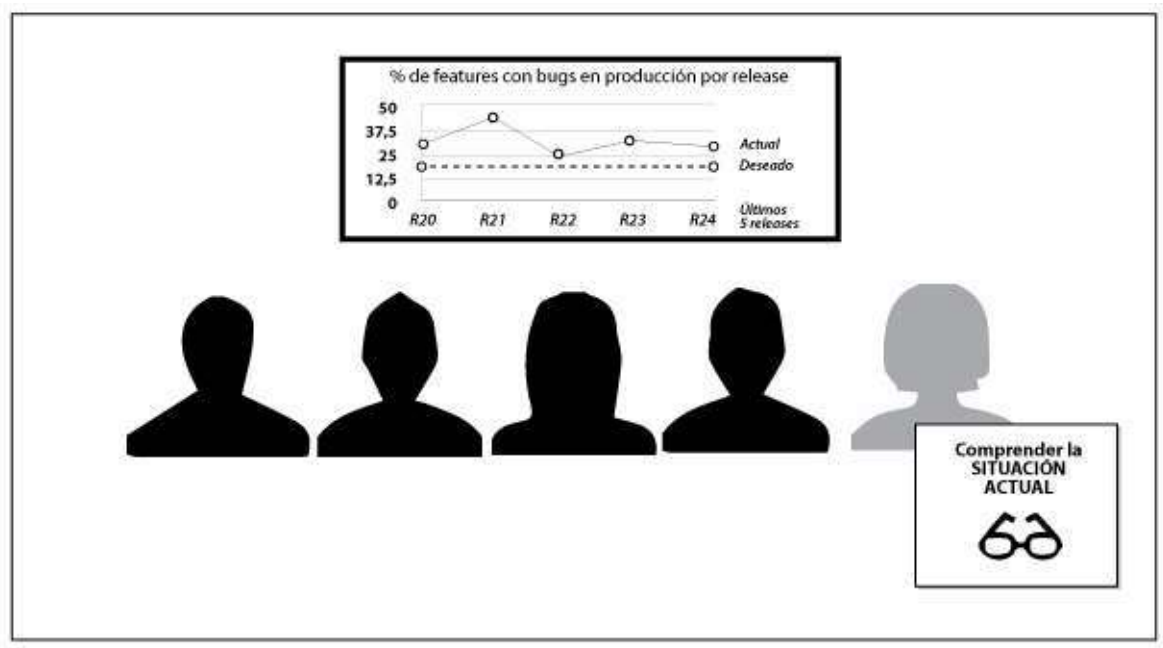

***Figura 12.4**. La situación actual*

Nota: La definición de la situación actual está simplificada para el ejemplo.

Siguiente Condición Objetivo

Una vez conscientes de la situación actual, Lisbeth junto con el equipo se proponen una siguiente condición objetivo que debería ser cumplida en dos meses. Consiste en reducir el porcentaje de bugs:

“15% de features con bugs en producción por release”

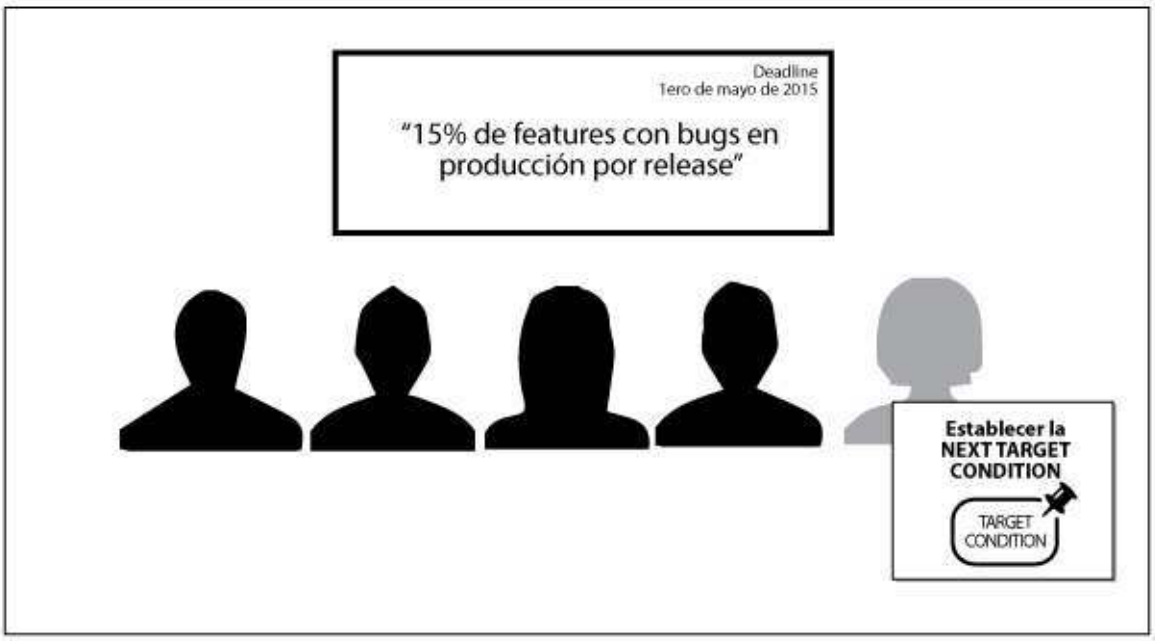

Figura 12.5. La siguiente condición objetivo

Obstáculos

Ahora que Lisbeth y el equipo tienen una meta clara a cumplir en los dos siguientes meses, pueden comenzar a pensar en los obstáculos que actualmente están evitando que estén en esa situación deseada. Luego de un poco de brainstorm, análisis y discusión, llegaron a la siguiente lista:

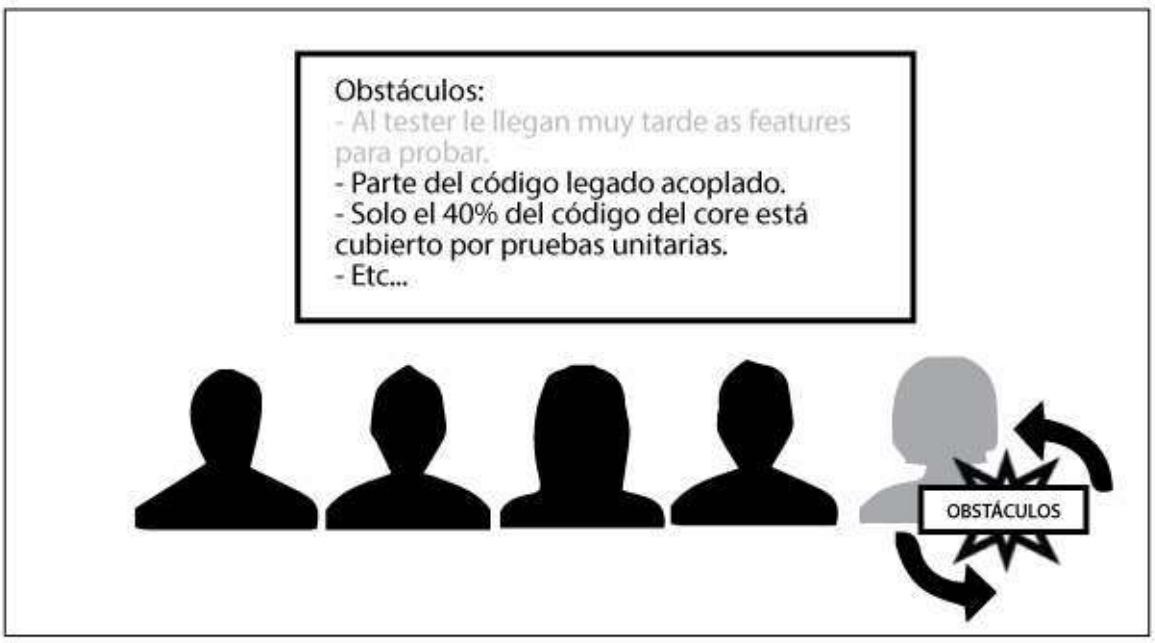

***Figura 12.6**. Los obstáculos*

De todos estos obstáculo, ellos han decidido atacar el obstáculo en naranja primero.

Con esto el equipo ha terminado la etapa de planificación de Improvement Kata y están listos para iniciar la etapa de ejecución, es decir, están listos para experimentar.

Ciclos PDCA

Luego de una conversación sobre cuál debería ser su primer experimento para buscar remover el obstáculo seleccionado, Lisbeth y el equipo determinan que probarán lo siguiente:

- Experimento: Hacer pairing con el tester antes de iniciar el desarrollo de una nueva funcionalidad.
- Resultado esperado: Que el tester tenga contexto del feature y pueda preparar con tiempo el set de pruebas. Con esto aumentar, en al menos 1 día, el tiempo de ejecución efectiva de pruebas, incluyendo las exploratorias.

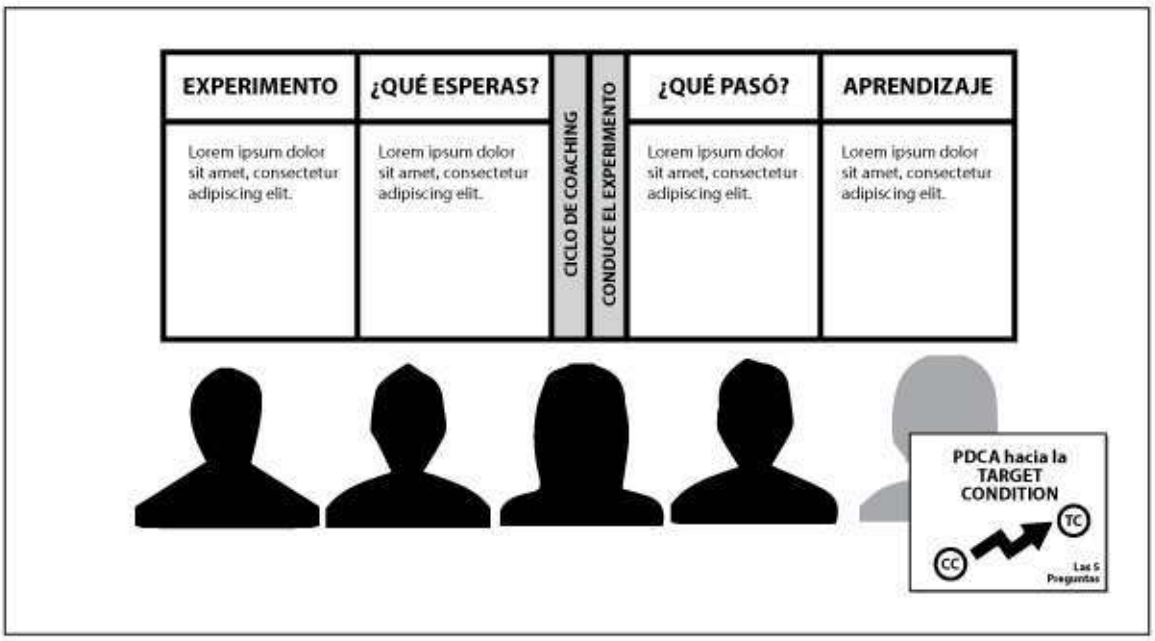

***Figura 12.7**. Experimentos*

Finalmente ahora que entendemos cómo funciona Improvement Kata solo queda resaltar que dado que las katas están diseñadas para ayudar a la persona a recorrer un camino, es relevante entender que el uso del patrón tal como está presentado solo es el inicio. Así que hago la invitación a ¡comenzar a recorrerlo!

Guardián de un equipo con múltiples asignaciones

Por Tomás Christie, @tommychristie

Palabras clave

Scrum, mantenimiento, desarrollo, multi-proyecto

Intención

La intención de esta técnica es aislar al equipo de las interrupciones, permitiéndoles desarrollar tranquilos.

Motivación

En equipos pequeños que atienden múltiples proyectos en forma simultánea, los integrantes suelen sufrir interrupciones permanentemente. Se pierde el foco, existe un costo altísimo de cambio de contexto y puede traer desmotivación. En general estas interrupciones son por fallas en los sistemas, que requieren de atención inmediata por parte de alguno de los integrantes del equipo, o más de uno.

Descripción

Se presenta típicamente en los casos de equipos pequeños que tienen una gran cantidad de proyectos, en todas las etapas del ciclo de vida. En estos casos suele suceder que las tareas de mantenimiento correctivo entran fácilmente en conflicto con los nuevos desarrollos, al punto que en algunos casos se vuelve muy difícil planificar.

Este conflicto típicamente lleva a que se le dedique poco tiempo a realizar las tareas de mantenimiento y a que los problemas se solucionan con “parches”, sin resolver realmente el problema de fondo. Irremediablemente, los parches se empiezan a acumular, parche sobre parche.

Esta situación hace que la calidad del software se vuelva progresivamente peor, llevando a que se generen cada vez más incidencias de mantenimiento correctivo, más interrupciones, menos foco por parte del equipo y más insatisfacción (tanto del cliente como del equipo).

Se propone tener a alguien que sea el guardián del equipo y que se encargue de que el resto del equipo no sufra interrupciones (o la menor cantidad posible). En una primera etapa a este guardián lo podríamos llamar “Bombero”, ya que mayormente atenderá urgencias, estará “apagando incendios”.

Como suele ser una tarea estresante, el equipo de desarrollo establece algún orden cíclico que incluya a todos. Luego, por sorteo (o alguna otra manera que el equipo decida), se selecciona al primer Bombero. Cada desarrollador tiene este rol por una semana, o el tiempo que el equipo considere conveniente. Personalmente, creo que es importante que sea un tiempo balanceado. Es decir, lo suficientemente corto como para que no sobrecargue al Bombero de turno (ya que es una tarea demandante) y lo suficientemente larga como para que los demás logren descansar del rol.

Cualquier pedido de acción que esté por fuera de la planificación, debe ser atendido por el Bombero. Esto en general debería responder a la resolución de bugs “urgentes”, del tipo “necesitamos facturar y no funciona tal o cual cosa”.

Algunas ventajas que vienen por añadidura:

- Distribución del conocimiento. El Bombero, al tener que resolver problemas de cualquier sistema, a la fuerza debe interiorizarse con cada sistema a medida que requiera atención.
- Proporciona tiempos de respuesta más rápidos ante bugs de los sistemas.
- Durante un período breve de tiempo, sólo uno sufre el constante cambio de contexto y luego tiene tranquilidad por un período proporcionalmente largo.

En general este estado de “bombardeo de bugs al equipo” es inicialmente fuerte y hace que el Bombero esté constantemente apagando incendios. Después de algún tiempo, esta situación tiende a mejorar, gracias a que la figura del Bombero tiene dedicación absoluta a estos temas y su tiempo no entra en conflicto con la planificación.

Cuando esto pasa, la figura del Bombero puede transformarse en la de un “Guardia”. De la misma manera que antes, el Guardia va rotando semana a semana. Hay dos variantes posibles para la figura del Guardia:

- Dedicar su tiempo a tareas preventivas o de saneamiento. Por ejemplo, implementar más tests, hacer refactoring de código con optimizaciones, etc. Muy probablemente, esta variante pueda aplicarse primero, a modo de transición entre el Bombero y la segunda variante del Guardia.
- Al Guardia se lo incluye dentro de la planificación. Realiza tareas de desarrollo al igual

que el resto, con la salvedad de que es el interrumpible. Es para equipos bastante evolucionados, con software de alta calidad, donde la incidencia del mantenimiento es baja. De esta manera, el Guardia colabora con el desarrollo siendo el único que “padece” las interrupciones.

Conclusión

La técnica funciona muy bien. Los resultados se ven en el corto plazo. El equipo en general adquiere un estado más relajado de trabajo, sabiendo que puede poner todo su foco en el desarrollo que tiene entre manos, sin tener interrupciones constantes.

En general, las tareas de apagar incendios merma y comienzan las tareas de prevención y Guardia. Vemos que al poder dedicarle tiempo a tareas de prevención, la calidad del software tiende a subir y las incidencias de bugs caen. ¡Se ingresa en un círculo virtuoso!

Coding Dojo: técnica de entrenamiento

Por Pablo Tortorella, @pablitux

Palabras clave

Coding dojo, conocimientos, experiencia, práctica, entrenamiento

Intención

Muchas empresas y profesionales creen que el **aprendizaje continuo** trae grandes beneficios. Por otro lado, los desafíos del mundo globalizado, competitivo y cambiante muchas veces llevan a los equipos de trabajo a focalizarse exclusivamente en resolver sus problemas, dejando así poco espacio para la reflexión, la mejora y la transferencia de conocimientos.

En este capítulo quiero dar a conocer las ventajas y contar cuáles son las características de un método de capacitación e integración que considero eficaz, eficiente, divertido y serio al mismo tiempo. Se trata del **Coding Dojo**.

Motivación

Messi entrena a diario. Corre, hace ejercicio físico y practica jugadas con pelota. Hace prácticas de tiros libres de todo tipo: con barrera, sin barrera, con arquero/portero, sin él, desde lejos y desde más cerca. Fue elegido como **el mejor** jugador de fútbol del planeta tierra, cinco veces. Cinco. Y aun así, el tipo **sigue entrenando**. A diario. Y lo mismo hacen todos los deportistas profesionales y de alto rendimiento. ¿Por qué lo hacen? Para mejorar su nivel, **para perfeccionarse**.

Así como Messi y los demás **deportistas**, también los **músicos** entrenan: **ensayan**. Aun siendo autores de las canciones. Aun conociendo cada una de las notas y de los ritmos con los cuales quieren interpretarlos en un concierto. Aun así, ensayan una y otra vez los temas, sus introducciones, sus estribillos y sus finales. Ensayan solos con sus instrumentos y también ensayan con los demás integrantes de las bandas u orquestas de las cuales forman parte.

Todas estas personas ejercitan sus habilidades y las potencian mediante práctica y más práctica. Saben que son capaces de mejorar y seguir mejorando. Refinando tanto detalles gruesos como finos. Mejoras que tal vez para otros sean imperceptibles. Mejoras que se identifican luego de miles de horas de entrenamiento.

¿Por qué nosotros, los profesionales del conocimiento, **no practicamos, no entrenamos y no ensayamos**? ¿Será nuestro ego? ¿Será un tema cultural? ¿Qué tan ciegos nos están dejando nuestras urgencias laborales? Creo que debemos cambiar esto: Aún los mejores en nuestras disciplinas, deberían seguir practicando, practicando y practicando para mejorar de forma permanente.

Es por eso, porque no practicamos -excepto algunos en los cursos y las carreras académicas que realizan por fuera del horario laboral- que considero que **el Coding Dojo puede ser de gran utilidad para todos nosotros**.

Dojo es una palabra japonesa. Significa "**el lugar del camino**", que hace alusión **al camino de la perfección** espiritual, física y mental. No es un camino con un destino, sino que es un camino en el que se ingresa para -ojalá- mantenerse toda la vida.

El Dojo, en Japón, es concretamente **el lugar físico** en el que se practica, donde se entrena. Existen dojos para variadas disciplinas, por ejemplo, para artes marciales. En este caso particular, en el Coding Dojo, se trata de la búsqueda de la perfección en el uso de las técnicas ágiles de desarrollo de software

Descripción

Roles

En un Coding Dojo la práctica suele hacerse en grupos. Específicamente suelen **ser grupos de 3 participantes**. Dentro de cada grupo hay roles y cada rol tiene sus responsabilidades. Los roles dentro de un grupo son Coder, Co-piloto y Asistente. El **Coder** también es llamado Driver, haciendo referencia al conductor de autos de Rally. Es quien puede tocar el teclado para escribir código fuente, con foco en lo que se está haciendo en ese mismo momento. El **Co-piloto** es llamado así haciendo referencia al acompañante del conductor de un auto de carrera; es quien se encarga de pensar estrategias de más mediano plazo, poniendo foco en los detalles a mejorar y en los escenarios que se irán resolviendo a continuación. El **Asistente** es como el Co-piloto, con la salvedad de que tiene una responsabilidad adicional, relacionada con un tablero del cual se habla más adelante en este mismo capítulo.

Estos tres roles suelen rotar a medida que se desarrolla la práctica, de forma tal que cada participante pueda practicar todos los roles. Ahondaremos más adelante al respecto de esta rotación.

En todo Dojo, también participa un **Sensei**. Es quien tiene la responsabilidad de guiar a los participantes, llamados también aprendices, en el camino de la mejora. También suele conocer en detalle las técnicas que se practicarán en su Dojo. Debe tener la tolerancia y la capacidad didáctica para lograr compartir sus conocimientos y su experiencia con los aprendices. También es recomendable que tenga la habilidad de transmitir esos saberes a diferentes asistentes, por ejemplo, de distintos niveles de experiencia.

Etapas

Un Coding Dojo es un evento que puede realizarse de muy variadas formas. Personalmente he facilitado más de 100 dojos en estos últimos años a lo largo y ancho del continente americano y tengo una agenda tentativa bastante refinada que casi siempre consta de las etapas que describo a continuación.

Invitación

Si querés que alguien participe de tu Coding Dojo, deberás -como mínimo- avisarle. Aprovechando el aviso, podés comentarle que se necesitarán **computadoras** (*notebooks*, *netbooks*, *ultrabooks* o las máquinas que se puedan conseguir). ¿Cuántas se requieren? Aproximadamente una computadora por cada dos a cinco participantes. Será provechoso si ya llevan instalado y configurado, antes de comenzar el evento, algún entorno de desarrollo (del inglés, IDE: *Integrated Development Environment*) con algún **framework de pruebas unitarias**, como es el caso de JUnit para Java, por mencionar un ejemplo.

En la invitación también es relevante que asistan con la **intención de aprender** cosas nuevas, dado que muchas personas participan de reuniones sólo para "difundir todo eso que ya saben a las personas ignorantes que las rodean" y se pierden grandes oportunidades de incorporar novedades a sus vidas. También les pido que traigan **buena onda**, pues es algo que nunca está de más y aporta grandes beneficios.

Preparación

En todos los casos que facilité un Coding Dojo permití, exceptuando un solo evento en una pequeña ciudad de la selva en el interior de Perú, que cada participante tuviera su silla o banqueta para sentarse. En aquel evento académico fueron 60 alumnos y docentes al Coding Dojo en el que -directivos y quien escribe- calculamos que irían sólo 15. A pesar de las condiciones, fue un hermoso Dojo en el que, algo incómodos, alcanzamos los objetivos.

Entonces, al preparar tu Coding Dojo deberás tener en cuenta lo siguiente:

1. Calculá cuánta gente entra cómoda, y evitarás así que te pase lo que a mí en el evento mencionado anteriormente.
2. Un Coding Dojo que tiene algo ligero para comer y tomar, es un mejor Coding Dojo.
3. Un Coding Dojo con comida y sin servilletas, será un desastre para los teclados.
4. Estos eventos duran entre una y dos horas, aproximadamente.
5. El foco suele tenerlo la práctica de desarrollo ágil de software, con lo cual vendrá bien que haya interés de los organizadores en prácticas tales como *Pair Programming* (Programación de a Pares), TDD (*Test Driven Development*) y *Refactoring*, así como también conocimiento en principios SOLID de diseño, por mencionar algunos temas ágiles relacionados con el desarrollo de software.
6. Los invitados pueden no saber qué es TDD, *Refactoring* ni SOLID. Lo aprenderán en el Dojo. Tampoco necesitan saber cómo escribir pruebas unitarias.
7. Si bien se espera que los participantes sepan programar, hubo excepciones en las cuales participaron exitosamente abogados, vendedores y otros perfiles no técnicos.
8. Deberás tener una agenda para presentarles a los que asistan. Posiblemente se parezca a un listado con los títulos que vienen a continuación.

Llegada y Presentación

La llegada al Dojo puede marcar el estado de ánimo del encuentro. Un ambiente de recepción con **música y comida** predispone bien a muchas personas. Luego, llegado el momento de iniciar, se hace una ronda para dar lugar a una **breve presentación** de cada uno de los participantes y de sus intenciones en el Coding Dojo. La disposición en ronda es intencional. Su objetivo es la participación activa, la horizontalidad y la posibilidad de que todo el que hable pueda ser escuchado y mirado por los demás.

Explicación de temas

Luego de la presentación, se da lugar a la **explicación** de varios conceptos, siempre y cuando esto se necesite. Es habitual describir **qué es** un Coding Dojo, cuáles son sus **objetivos**, cómo será el **mecanismo** de trabajo y cuáles son los **roles** que se practicarán. También se suele describir -en mayor o menor detalle- qué es **TDD** y técnicas de desarrollo ágil, como puede ser *Refactoring*. En general es útil ilustrar estos conceptos con **ejemplos** simples y concretos, llevados adelante en vivo, ahí mismo.

A partir de la explicación de TDD, suelo pedir a los participantes que esquematicen en una hoja de papel las etapas del ciclo de trabajo que esa técnica propone, llamado RGR (del inglés Red, Green, *Refactor*). Con la hoja del ciclo RGR, el Asistente guiará y controlará a

su grupo en el seguimiento del ciclo RGR. El uso que se da a esta hoja se asemeja al que tiene el **tablero** en un juego de mesa.

Esta etapa de explicación podría durar entre 15 y 45 minutos, dependiendo del grado de conocimiento que existe al respecto entre los participantes. Y, dado que el objetivo del Dojo es aprender y mejorar mediante la práctica, es importante **que la explicación no dure demasiado**, aún si quedan dudas. Durante la práctica y en las conclusiones finales, muchas de esas dudas suelen resolverse.

El desafío o la Kata

El Coding Dojo fomenta la práctica a través de ejercicios concretos y finitos que representen un **desafío** y resulten **motivadores**. Estos desafíos no son problemas que los equipos deban resolver para sus trabajos. Al tratarse de un entrenamiento, se proponen desafíos que les sirvan a los participantes para **aprender y mejorar**. Los desafíos se llevan adelante mediante la realización de una actividad. Esa actividad suele ser una **Kata**, palabra que en japonés significa “forma”. Las *katas* en las artes marciales son ejercicios que se realizan repetidamente, con el objetivo de incorporar movimientos de forma evolutiva. Así, en un combate, luego de muchas repeticiones, esos movimientos se podrán realizar naturalmente sin siquiera pensarlos.

Por ejemplo, una *Kata* que suelo proponer para aprender TDD es crear un componente de software que convierta números naturales a números romanos. Si el componente recibe un 1 deberá devolver “I”, si recibe un 2 deberá devolver “II”, si recibe 5 deberá devolver “V” y así para todos los números romanos hasta 3999. El objetivo de la *Kata* no es resolver el desafío por completo ni diseñar desde el inicio la solución final, sino practicar **paso a paso**, resolviendo cada una de las partes, tal como propone TDD.

Desarrollo de la Kata

Una vez que los conceptos han quedado claros y se ha elegido una *Kata*, **comienza la práctica**. Se invita a los participantes a que se organicen en **grupos** de 3 integrantes. Si son pares, que se armen todos los grupos de 3 que sean posibles y finalmente quedará algún grupo de 4 o de 2. Todo el grupo trabajará en **una misma computadora**.

Se trabajará en ciclos cortos, también llamados **iteraciones**. Son períodos de tiempo que pueden ir entre 5 y 10 minutos. En los Dojos en los cuales hay mayoría de grupos con 3 integrantes, la duración que he encontrado más conveniente es de 7 minutos.

Al cabo de los 7 minutos, pedir a los participantes que **roten de rol**. Entonces el *Coder* pasará a ser Co-piloto, el Co-piloto pasará a ser Asistente y el Asistente pasará a ser *Coder*. Eso será así durante los siguientes 7 minutos. Y así se repetirá durante 4 a 8

iteraciones, aproximadamente. Esta cantidad depende del tiempo que se quiere invertir en el Coding Dojo.

El Sensei deberá estar atento a las necesidades que los grupos puedan tener durante en el desarrollo de la *Kata*, compartiendo oportunamente su conocimiento y experiencia. También podrá realizar aclaraciones y comentarios entre ciclos, para compartir con todos los grupos algún conocimiento o hallazgo que haya ocurrido en algún grupo en particular.

Una posibilidad opcional es fomentar, cada 3 o 4 iteraciones, la rotación de los grupos para que los participantes trabajen también con el código fuente que fue desarrollado por otros. El objetivo de esta otra rotación es generar mayor conciencia al respecto de la mantenibilidad, un atributo de calidad esencial en el desarrollo ágil de software. Este tipo de rotación implica que **todo el grupo** (o parte del mismo) se mueva a otra computadora y trabaje allí durante algunas iteraciones. Si alguien del grupo se queda trabajando en la computadora original, es deseable que comience la iteración como Co-piloto o Asistente, para permitir, a sus nuevos compañeros de grupo, que conozcan la solución que se ha estado desarrollando antes de su llegada.

Conclusiones y cierre del Dojo

Un rato antes de terminar el encuentro, se suele realizar una **nueva ronda**. En esta ronda de cierre, se invita a los participantes a que **compartan** sus conclusiones, a partir de lo que han aprendido, de los desafíos que se han encontrado, de las emociones y sensaciones que han sentido. También se los invita a compartir cómo han pasado en general. Esto último apunta a que no solamente compartan cuestiones técnicas, sino que también puedan también conversar sobre lo que les haya resultado relevante a lo largo del Coding Dojo. No es raro escuchar **reflexiones** acerca del trabajo en equipo, la distribución de teclados ajenos, la tolerancia frente a diferentes formas de pensar soluciones para la *Kata* y las inquietudes más profundas relacionadas con las técnicas que se han estado utilizando durante la práctica.

Para poder mejorar entre un Coding Dojo y el siguiente, es habitual pedir retroalimentación a los participantes, para que compartan qué les pareció útil, qué les gustó, qué no les gustó y qué ideas tienen para mejorar de cara al siguiente encuentro.

Conclusión

Habiendo facilitado más de cien Coding Dojos en estos últimos cinco años, tanto en ámbitos comunitarios como académicos y corporativos, puedo asegurar que siempre ocurrió la magia: gente que aprendió cosas nuevas relevantes para su día a día y que pasó a ser más

consciente. A partir de participar activamente de un Coding Dojo, miles de personas empezaron a ver cosas a su alrededor que antes no habían visto: Temas técnicos, metodológicos, humanos y tecnológicos.

Gracias a esa experiencia, puedo garantizar que organizar eventos del tipo Coding Dojo en el marco de un equipo, compañía o comunidad de práctica, sirve para:

- Difundir y compartir conocimientos de todo tipo.
- Fomentar la posterior auto-capacitación.
- Mejorar el uso de técnicas y herramientas.
- Ampliar capacidades.
- Refinar habilidades.
- Establecer bases de un lenguaje común que mejora la comunicación.
- Crear y mejorar relaciones entre los participantes.
- Mejorar potencialmente la calidad del código fuente y por ende del producto.
- Establecer un ambiente en el cual la mejora continua sea posible y real.

¿Qué hacés ahí, leyendo? ¿Qué esperás para organizar y facilitar tu propio Coding Dojo?

Automatización a través de Git hooks

Por Fernando Di Bartolo, @fdibartolo

Palabras clave

desarrollo de software, git, automatización, entrega continua

Intención

Minimizar el error humano a través de la automatización de ciertas rutinarias tareas de un desarrollador de software.

Motivación

Somos humanos, y por ello, cometemos errores. Por otra parte, en la ingeniería de software, se menciona el hecho de que cuanto más tarde se descubre un *defecto*, mayor es el esfuerzo necesario para solucionarlo. Dicho esto, desde la perspectiva del desarrollador de software, si damos por sentado que vamos a introducir errores, defectos, y/o violación de estándares, ¿qué podemos hacer para detectarlos lo suficientemente temprano como para que corregirlos sea trivial?

A esta altura y por los años que corren, la respuesta a la pregunta de arriba podría ser: “escribe tus pruebas unitarias, escribe tus pruebas de integración, escribe tus pruebas de interfaz gráfica, bla bla bla”. Claramente yo apoyo esta moción (sujeto al contexto del proyecto en cuestión), pero aun así, y lo he visto, no todos los miembros de un equipo de desarrollo tienen la misma disciplina de correr dichas pruebas antes de enviar sus cambios al repositorio, o bien, solo corren aquellas que “teóricamente” están asociadas a su cambio en el código. Entonces, nuevamente, ¿qué podemos hacer?

Descripción

La idea planteada aquí no es bala de plata para cualquier implementación, ni tampoco la única. Sin ir más lejos, existen herramientas que incluso facilitan lo que estoy a punto de describir, pero la intención es plantear las bases, más aún cuando nuestro contexto nos

limita a poder hacer uso de aquellas herramientas (por ejemplo, si trabajamos con un cliente que ya tiene su set de herramientas predefinido; o bien que por políticas internas, las herramientas que podríamos usar no están permitidas).

Los *git hooks* (ganchos) son pequeños *scripts* que nos permiten ejecutar acciones en ciertos puntos del flujo de trabajo. Estas acciones podrían ser por ejemplo:

- Generar un nuevo objeto *commit.*
- Sincronizar código propio con un repositorio remoto.
- Hacer un *merge* de dos ramas del repositorio.

En pos de mantener la simpleza, aquí mencionaremos solo 3 *hooks*, aunque la lista es larga como indica Scott Chacon en su libro “Pro Git”, [Chacon 2009].

Entonces, en líneas generales, un desarrollador debe:

1. Cumplir con estándares de codificación.
2. Ejecutar la(s) suite(s) de pruebas localmente, contemos o no con un servidor de integración continua.
3. Desplegar código a un servidor/entorno de prueba.

Veamos cuáles *git hooks* nos ayudan a lograr que estas tareas rutinarias se ejecuten en el momento adecuado y de manera automática, sin riesgo de error humano (errores tales como olvido o mala lectura del resultado por falta de atención).

Nos encontramos escribiendo código y guardando los cambios a través del comando git add. Estamos listos para hacer un *commit*. Deberíamos ahora ejecutar manualmente la herramienta para asegurar que nuestros cambios no violan ningún estándar, pero frecuentemente, es algo que olvidamos y que terminamos chequeando *post-mortem*. He aquí un git hook al rescate: **pre-commit**. Creamos un archivo con ese mismo nombre dentro de *[repo]/.git/hooks*, con su correspondiente permiso de ejecución. Dentro del mismo, programamos aquello mismo que veníamos corriendo a mano, que dependiendo del lenguaje de programación en cuestión, será usando librerías tales como Rubocop, JsLint o CodeAnalysis. Una particularidad con este *hook* es que, llegado el caso que algún estándar no se cumpla y quisiéramos por ello rechazar el *commit*, solo tenemos que hacer que el *script* retorne un valor distinto de cero.

Tenemos ahora un conjunto de *commits* listos para empujarlos a un repositorio remoto. Si no lo hicimos antes, deberíamos ejecutar manualmente nuestro conjunto de pruebas, y nuevamente, solo lo hacemos cuando nos acordamos. El hook que ahora nos puede ayudar es el **pre-push**. Así como hicimos antes, creamos un archivo con el mismo nombre del *hook* (pre-push) y en el mismo lugar físico ([repo]/.git/hooks). Escribimos dentro de él aquel o aquellos comandos que corren nuestras pruebas. Ante una prueba fallida, haremos nuevamente que el *script* retorne un valor distinto de cero, y el *push* será rechazado. Como

habrán notado, todos aquellos *hooks* cuyo nombre comience con **pre-**, se ejecutan un instante antes de la acción en cuestión, con lo cual, haciendo que el *script* retorne un valor distinto de cero, impedimos que la acción se lleve a cabo.

Por último, necesitamos desplegar código a un entorno de pruebas, y esto puede requerir varios pasos: actualizar esquema de base de datos, minimizar javascripts y/o hojas de estilo, etc. Tareas rutinarias, algunas fácil de saltear, por lo cual, ¡automaticemos! Suponiendo que desplegar implica empujar código a un repositorio remoto (por ejemplo, de la forma que funciona Heroku), podemos hacer uso del hook **post-receive.** En este caso, a diferencia de los 2 primeros, debemos crear el archivo en el repositorio remoto, no en el que tenemos localmente. Una vez que empujemos código via git push desde nuestro repositorio local hacia el remoto, el *script* correrá en el servidor, aunque tendremos la posibilidad de ver el log de ejecución desde nuestra máquina, en tiempo real.

En resumen

No todos los pasos o procesos que automatizamos en el contexto de un proyecto, funcionan tal cual en otro. De la misma manera, la automatización está limitada por la creatividad del desarrollador; lo importante es hacer una buena lectura del contexto y aplicarla con criterio.

Versionado de código, configuración y ambientes

Por Nicolás Paez, @inicopaez

Palabras clave

Devops, versionado, entrega continua

Intención

¿Cómo organizar el versionado de artefactos de software en un contexto de entrega continua?

Motivación

Desde una perspectiva de entrega continua toda aplicación es **código** ejecutándose en un determinado **ambiente** con una determinada **configuración**. Algunos fanáticos podrían insistir en que los datos son también parte de la aplicación, pero este es un punto que puede resultar muy polémico y por ello se dejará de lado en esta ocasión. Entonces tenemos:

código + configuración + ambiente

Por ambiente entendemos como mínimo una máquina (física o virtual) con un cierto sistema operativo con determinado software de base, por ejemplo: un servidor de aplicaciones o un servidor web. En la actualidad existen diversas herramientas que permiten especificar los ambientes en forma de código (Chef, Puppet, Ansible, etc). Entonces con una vuelta de rosca más tenemos:

código de la aplicación + configuración de la aplicación + código que describe el ambiente de la aplicación.

Finalmente como nuestra aplicación deberá evolucionar a lo largo del tiempo, agregaremos a nuestra lista un paquete de código adicional que describa cómo instalar/desplegar nuestra aplicación, con su correspondiente configuración, en su correspondiente ambiente. Pasando en limpio, podemos decir que toda aplicación se compone entonces de 3 artefactos:

- El código fuente.
- La configuración.
- Los scripts de despliegue.

En un contexto de entrega continua es necesario gestionar y versionar cada uno de estos 3 artefactos. La pregunta es entonces cómo hacerlo.

Descripción

Estos tres artefactos tienen un frecuencia de cambio distinta: el código fuente suele estar en constante modificación, mientras que los scripts de despliegue suelen ser mucho más estables y en algunos casos no recibir cambios por semanas o meses. La frecuencia de cambio de la configuración de la aplicación se encuentra en un punto intermedio, no está en constante cambio pero tampoco es tan estable como los scripts de despliegue.

Por otro lado, dependiendo del contexto organizacional, pueden definirse distintas políticas de acceso a estos artefactos. En algunas organizaciones es muy común que el equipo de desarrolladores no tenga permisos para acceder a los parámetros de configuración del ambiente productivo. También suele ocurrir que los scripts de despliegue de la aplicación sean administrados por personas ajenas al equipo de desarrolladores (usualmente personas del área de operaciones).

Estas dos cuestiones, frecuencia de cambio y permisos de acceso, son las que nos motivan a separar estos tres artefactos del proyecto en diferentes repositorios.

El primer repositorio es el que almacena el código fuente de la aplicación. Dependiendo de la complejidad del proyecto, puede haber más de un repositorio para el código fuente. Un caso típico de esto son las arquitecturas basadas en microservicios, donde cada microservicio suele tener su propio repositorio.

El segundo repositorio es el que almacena la configuración de la aplicación. Este repositorio tiene típicamente un branch por ambiente (por ejemplo: desarrollo, testing, producción) ya que la configuración de la aplicación suele variar de un ambiente a otro. Hay que destacar que estos branches nunca se mezclan, sino que evolucionan a la par. Cuando la aplicación requiere de un nuevo parámetro de configuración, el mismo debe ser agregado simultáneamente a cada uno de los branches con el valor correspondiente al ambiente asociado.

Finalmente el tercer repositorio es el que contiene los scripts de despliegue. Dependiendo de la infraestructura del proyecto pueden ser simplemente scripts de Bash o de alguna herramientas específica como Ansible, Puppet o similar.

Referencias

[Ágiles 2015] Agiles 2015, VIII Jornadas Latinoamericanas de Metodologias Agiles, Montevideo, Uruguay - agiles2015.agiles.org

[Adzic 2014a] Adzic, Gojko. (2014). User stories should be about behaviour changes. gojko.net Recuperado de: https://gojko.net/2014/02/12/user-stories-should-be-about-behaviour-changes/

[Adzic 2014b] Adzic, Gojko. (2016) ABE15 (Agile By Example) Make Impact Not Software YouTube: https://youtu.be/-Hp9MEENlil

[Adzic-Evans 2014] Adzic, Gojko & Evans, David. (2014) Fifty Quick Ideas to Improve your User Stories. Neuri Consulting LLP.

[AOC 2016] Sitio del Agile Open Camp 2016, Descripción del Mecanismo San Saru - www.agileopencamp.com.ar/index.php/sansaru

[Buonamico 2013] Buonamico, Damián , Visual Story Mapping Aplicado, disponible en línea: http://www.caminoagil.com/2013/02/visual-story-mapping-aplicado.html, como estaba en Febrero de 2016.

[Cagan 2012a] Cagan, Marty, Continuous Discovery, disponible en línea: http://www.svpg.com/continuous-discovery/

[Cagan 2012b] Cagan, Marty, Dual Track Scrum, disponible en línea: http://svpg.com/dual-track-scrum/

[Chacon 2009] Chacon, Scott, Pro Git, Apress, 2009

[Cockburn 2008], Cockburn, Alistair, Shu Ha Ri, disponible en línea: http://alistair.cockburn.us/Shu+Ha+Ri, como estaba en febrero del 2016

[Crespo-Villena 2005] Raquel M. Crespo García, Julio Villena Román, Revisión entre pares como instrumento de aprendizaje. Una experiencia práctica, Universidad Carlos III de Madrid Serie de innovación docente, Marzo 2005.

[Fowler 2014] Fowler, Martin, ShuHaRi, disponible en línea: http://martinfowler.com/bliki/ShuHaRi.html, como estaba en febrero del 2016

[Garzás 2015] Garzás, Javier, Inception en contextos ágiles: dejemos las ideas claras desde el primer momento, disponible en línea: http://www.javiergarzas.com/2015/09/inception-en-contextos-agiles.html, como estaba en Febrero de 2016.

[Hiromoto 2013] Hiromoto, Hiroshi, Visión - Bonus Track: Product Vision Board, disponible en línea: http://scrumorganico.com/blog/visi%C3%B3n-bonus-track-product-vision-board, como estaba en Febrero de 2016.

[Jared 2014] Jared, Jeremy. (2014). Stories versus Themes versus Epics. Navigating the Waters en Scrum Alliance Member Articles Recuperado de: https://www.scrumalliance.org/community/articles/2014/march/stories-versus-themes-versus-epics#sthash.aqJoq80Z.dpuf

[Karen 2013] Karen, Martin, Value Stream Mapping: How to Visualize Work and Align Leadership for Organizational Transformation, McGraw-Hill Education, 2013.

[Larsen y Nies 2011] Larsen, Diana y Nies, Ainsley, Liftoff: launching agile projects and teams, Onyx Neon Press, 2011.

[Libro 2015] Disponible en linea en: https://www.gitbook.com/book/nicopaez/libroagileaoc2015/

[Manifiesto ágil] Disponible en linea en: http://www.agilemanifesto.org/iso/es/, como estaba en Febrero de 2015

[Cohn 2015] Product Backlog Refinement (Grooming), disponible en línea: https://www.mountaingoatsoftware.com/blog/product-backlog-refinement-grooming

[Mock Objects] Mock Objetcs, disponible en línea: http://www.mockobjects.com/

[Patton 2014] Patton, Jeff, User Story Mapping, O'Reilly, 2014.

[Pichler 2013] Pichler, Roman, Product Vision Board, disponible en línea: http://www.romanpichler.com/tools/vision-board/, como estaba en Febrero de 2016.

[Rasmusson 2010a] Rasmusson, Jonathan, The Agile Samurai: How Agile Masters Deliver Great Software, The Pragmatic Bookshelf, 2010.

[Rasmusson 2010b] Rasmusson, Jonathan, The Agile Inception Deck, disponible en línea: https://agilewarrior.wordpress.com/2010/11/06/the-agile-inception-deck/, como estaba en Febrero de 2016.

[Román 2006] Román José, Los 3 monos misticos, 2006 - http://www.emezeta.com/articulos/los-tres-monos-misticos

[Steinberg-Palmer 2003] Steinberg, Daniel H. & Palmer, Daniel W. (2003). Extreme Software Engineering A Hands-On Approach. Upper Saddle River, NJ, EE. UU. : Prentice-Hall, Inc.

[Suzuki 1987] Suzuki, Shunryu, Mente Zen, mente de principiante, Estaciones, 1987

[Wake 2003] Wake, Bill. (2003). INVEST in Good Stories, and SMART Tasks XP123 - Xplorations Recuperado de: http://xp123.com/xplor/xp0308/index.shtml

[Wendel 2013] Wendel, Stephen. (2013). Designing for Behavior Change: Applying Psychology and Behavioral Economics. Boston, MS, EE. UU. : O'Reilly Media.