SYSONE

DISCOVERY PATH

El camino ágil para la implementación de soluciones inteligentes.

SYSONE

Discovery Path

El camino ágil para la implementación de soluciones inteligentes.

Sobre SysOne

Somos el socio tecnológico y estratégico para compañías de seguros que buscan transformarse digitalmente, con el impulso de la gestión de prácticas ágiles dentro de un mercado complejo y en constante evolución. Somos SysOne y estamos para hacer simple lo complejo.

Índice

Introducción

En la era digital, la relación entre empresas y clientes está cambiando significativamente. Durante años, incluso en la actualidad, gran número de compañías padecen los efectos limitantes de contar con sistemas heredados y estructuras organizacionales que poco tienen que ver con las demandas del nuevo cliente digital.

Como respuesta a este panorama de necesidad, creamos “Discovery Path”, un libro práctico y sencillo que combina nuestro know how en el mercado asegurador, experiencias en el desarrollo e implementación de software de alta complejidad y, la puesta en marcha de metodologías ágiles, situando a estas últimas dentro de un ecosistema en constante mutación, donde el mayor desafío es la capacidad de adaptación al cambio.

Ahora, la pregunta sería, ¿Usted está interesado en cómo adaptarse a estos grandes cambios y emprender el mayor desafío dentro de su compañía?

En SysOne sostenemos que esta ruptura es inevitable, ¡pero el cambio es completamente posible!

No entre en pánico, relájese. Este libro está aquí para ayudarlo. Siga leyendo y verá cómo transformarse en un jugador clave en el campo digital.

¿A quiénes se encuentra dirigido?

Muchas personas y equipos pueden beneficiarse más de este libro, pero nos tomamos la libertad de asumir lo siguiente:

✓ Está buscando poner a prueba un proyecto utilizando metodologías ágiles. Es un director de proyecto, un líder técnico o quiere lanzar un nuevo producto al mercado y quiere adoptar prácticas ágiles, pero no está seguro de por dónde empezar. Es posible que haya probado algunas prácticas agiles de manera ad hoc y haya encontrado algunas dificultades. No se preocupe; muchos equipos experimentan algunos pasos en falso cuando se mueven por primera vez al mundo ágil.

✓ Ha tenido algún éxito en el proyecto y está buscando hacer crecer la práctica ágil más allá de su equipo. Está buscando formas de coordinar varios equipos con los mismos resultados que experimentó en su pequeño equipo.

✓ Quiere probar la agilidad, pero su entorno tiene complejidades que deben abordarse. Tal vez tenga equipos distribuidos globalmente o esté sujeto a mandatos de cumplimiento normativo. Se pregunta si las prácticas ágiles pueden ser efectivas en este entorno.

En definitiva, este libro ayuda a explicar y reforzar las prácticas exitosas de desarrollo de software disponibles en la actualidad. Aquí hay mucho que pensar, incluso si su equipo u organización actual aún no está listo para dar el salto ágil.

Objetivo del libro

Los principios de desarrollo ágil han pasado de ser algo que solo utilizan los equipos de vanguardia a un enfoque general utilizado por equipos grandes y pequeños para cosas tan variadas como las siguientes:

✓ Proyectos de desarrollo de software en su etapa inicial.

✓ Esfuerzos de desarrollo de tamaño empresarial.

✓ Iniciativas complejas de ingeniería de sistemas a gran escala.

✓ Sistemas legacy o heredados.

✓ Sistemas integrados en tiempo real.

✓ Entornos de alto cumplimiento (como atención médica, seguros o banca).

¿Qué deseamos con este libro?

1. Explicar de qué manera SysOne opera con metodologías ágiles para la implementación de proyectos.

2. Servir de guía para el entendimiento de nuestros clientes y nuestros colaboradores de cómo SysOne implementa y gestiona proyectos.

3. Mantener la simplicidad en la gestión de proyectos promoviendo la colaboración con el cliente y empleando algunos instrumentos de gestión para ser más efectivos, tales como: user story map, historias de usuario, backlog, sprint, entre otros.

4. Emplear este framework para la gestión de cualquier cambio o iniciativa que promuevan nuestros clientes para generar valor al negocio.

5. Lograr implementar el sistema central como un mínimo producto viable (MVP) que genere rentabilidad y resultados a corto plazo, para

luego realizar entregas incrementales de mejora continua para llegar al ideal.

6. Fomentar el uso del sistema central en forma temprana por todos los colaboradores, con la finalidad de obtener feedback y propuestas de mejoras de valor agregado.

7. Llevar a cabo una transformación cultural y organizacional dentro de las compañías de seguros, que propicie la innovación y transferencia de conocimientos para la implementación de soluciones ágiles.

¿Qué son las metodologías ágiles?

En febrero de 2001, un grupo de desarrolladores interesados en promover metodologías de desarrollo ligero se reunieron para hablar sobre sus puntos de vista y encontrar puntos en común, y así nació el concepto de agilidad. Los desarrolladores que crearon este enfoque entendieron la importancia de crear un modelo en el que cada iteración en el ciclo de

desarrollo "aprendiera" de la iteración anterior. El resultado fue una metodología más flexible, eficiente y orientada al equipo que cualquiera de los modelos anteriores.

Todos los métodos ágiles buscan orientación en el Manifiesto Ágil y los 12 principios básicos. La adherencia a la guía proporcionada por el manifiesto y los principios es lo que hace que un equipo de desarrollo de software sea ágil, no un proceso, herramienta o etiqueta específicos.

Principios del Manifiesto Ágil

1. Considerar prioritario la satisfacción del cliente. En consecuencia a ello, garantizar la entrega temprana y continua de software de valor.

2. Entender la dinámica y cambios del mercado, para adaptarnos ágilmente a ellos. La implementación se basa en una constante iteración que se va “moldeando” de forma incremental.

3. Iterar de forma constante para la entrega incremental del proyecto. Definir períodos

breves de trabajo (en general entre dos semanas y un mes), para desarrollar historias de usuario previamente priorizadas y definidas.

4. Trabajar de forma conjunta durante todo el proyecto, entre los responsables del negocio y los desarrolladores, ya que entendemos que las implementaciones exitosas son transversales a las compañías.

5. Desarrollar e implementar los proyectos en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo.

6. Ser más eficientes y efectivos a la hora de comunicar información al equipo de desarrollo y entre sus miembros, a través de la planificación de reuniones diarias de seguimiento a fin de compartir avances, impedimentos, etc.

7. Velar siempre por la disponibilidad del software, en todo momento y lugar.

8. Promover el desarrollo sostenible. Los promotores, desarrolladores y usuarios

debemos ser capaces de construir un ambiente de comunicación continua, clara y efectiva.

9. Perseguir la excelencia técnica y la mejor experiencia de usuario, mejora la agilidad.

10. Maximizar la cantidad de trabajo y esfuerzo.

11. Construir equipos auto-organizados.

12. Reflexionar en equipo sobre cómo ser más efectivos para emprender una mejora continua de actividades y procesos.

El Manifiesto Ágil es sencillo y poderoso. Las siguientes secciones desglosan los componentes del manifiesto:

- **Individuos e interacciones sobre procesos y herramientas:** Reconociendo que el software está hecho por personas, no por procesos o herramientas, las metodologías ágiles otorgan una mayor importancia a las personas que trabajan juntas de manera efectiva. Los procesos y las herramientas pueden ayudar en eso, pero no pueden reemplazarlo.

- **Software de trabajo sobre documentación completa:** Valorar el software de trabajo por encima de la documentación completa se opone rotundamente al modelo Waterfall (enfoque tradicional o de cascada). Un documento de especificaciones muy detallado, preciso y completo no tiene valor si no da como resultado un software funcional que satisfaga las necesidades de los usuarios. El software de trabajo puede involucrar documentación, pero el enfoque ágil solo lo usa para crear software de trabajo, no como un fin (casi) en sí mismo.

- **Colaboración con el cliente sobre la negociación del contrato:** Si bien la agilidad no ignora la realidad de los contratos, valora la colaboración activa a lo largo del proceso de desarrollo de software como una mejor manera de ofrecer valor en lugar de un contrato redactado cuidadosamente. Un contrato no es un sustituto de la comunicación real cuando estás haciendo algo tan desafiante como crear software.

- **Respuesta al cambio sobre el cumplimiento estricto de un plan:** A excepción de los sistemas más increíblemente simples, es

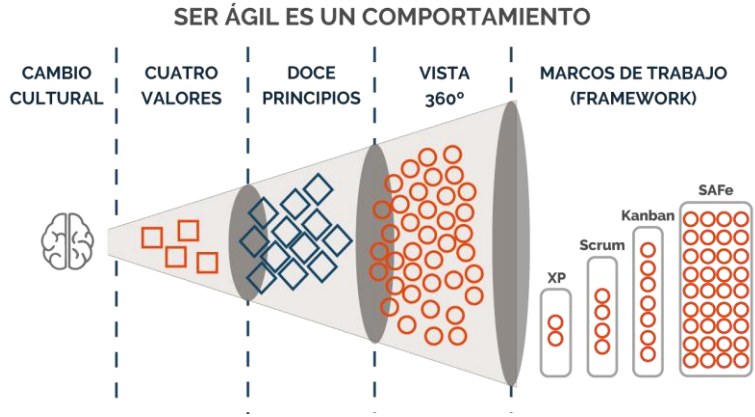
enormemente difícil pensar en todas las funciones, todos los datos y todos los casos de uso posibles del software. Eso significa que, en un proceso de colaboración con el cliente, se descubre mucho durante el proceso de desarrollo de software.

Además, el mundo cambia bastante rápido: Las necesidades y prioridades de la empresa pueden cambiar en los meses o incluso en los años que puede llevar la construcción completa de un sistema grande. Agile valora la capacidad de cambiar en respuesta a nuevos descubrimientos y necesidades en lugar de ceñirse a un plan creado antes de que se supiera todo.

¿Por qué utilizamos frameworks ágiles?

El entorno de trabajo de las empresas del conocimiento, se parece muy poco al que originó la gestión de proyectos tradicional. Ahora se necesitan estrategias para el lanzamiento de productos orientadas a la entrega temprana de resultados tangibles, y a la respuesta ágil y flexible, necesaria para trabajar en mercados de evolución rápida como lo es la industria del seguro.

Las metodologías ágiles son un intento de hacer que el proceso de desarrollo de software y su implementación sea mejor, más efectivo y más exitoso. Descubrirá cuán ágil es un enfoque incremental e iterativo para entregar software de alta calidad con entregas frecuentes para garantizar el valor durante todo el proceso. Da un gran valor a las personas, la colaboración y la capacidad de responder al cambio.



1. **Cambio Cultural:** Cambio organizacional y de mindset dentro de las compañías. Para realizar un cambio verdadero, también en necesario reestructurar todas las áreas.

2. **Cuatro Valores:**

a. Individuos e interacciones sobre procesos y herramientas.
b. Software de trabajo sobre documentación completa.
c. Colaboración con el cliente sobre la negociación del contrato.
d. Respuesta al cambio sobre el cumplimiento estricto de un plan.

3. **Doce Principios:** Ver apartado “Principios del Manifiesto Ágil” del presente libro.

4. **Vista 360º:** El cliente/usuario es el centro del Negocio. Optimización del trabajo para responder rápido a los cambios.

5. **Marco de Trabajo:** Utilizar un marco definido y ser disciplinado en su ejecución. Promover la mejora continua. Establecer los circuitos de retroalimentación continua.

Ahora se construye el producto al mismo tiempo que se modifican e introducen nuevos requisitos. El cliente

parte de una visión medianamente clara, pero el nivel de innovación que requiere, y la velocidad a la que se mueve el entorno de su negocio, no le permiten prever con detalle cómo será el resultado final.

Ya no hay "productos finales", sino productos en continua evolución y mejora.

Orientados a la generación de valor

La mejor forma de explicar por qué SysOne basa estratégicamente su cultura en agilidad es con el siguiente ejemplo:

"Hoy hay directores de producto que no necesitan conocer cuáles van a ser las 200 funcionalidades que tendrá el producto final, ni si estará terminado en 12 o en 16 meses. Hay clientes que necesitan disponer de una primera versión con funcionalidades mínimas en cuestión de meses o en su caso extremo semanas, y no un producto completo dentro de uno o dos años.

Aseguradoras cuyo interés es poner en el mercado rápidamente un concepto nuevo, innovando, siendo pioneros y desarrollar de forma continua su valor. Hay proyectos que no necesitan gestionar el seguimiento de un plan, y cuyo fracaso puede ser la consecuencia de un modelo de gestión inapropiado.

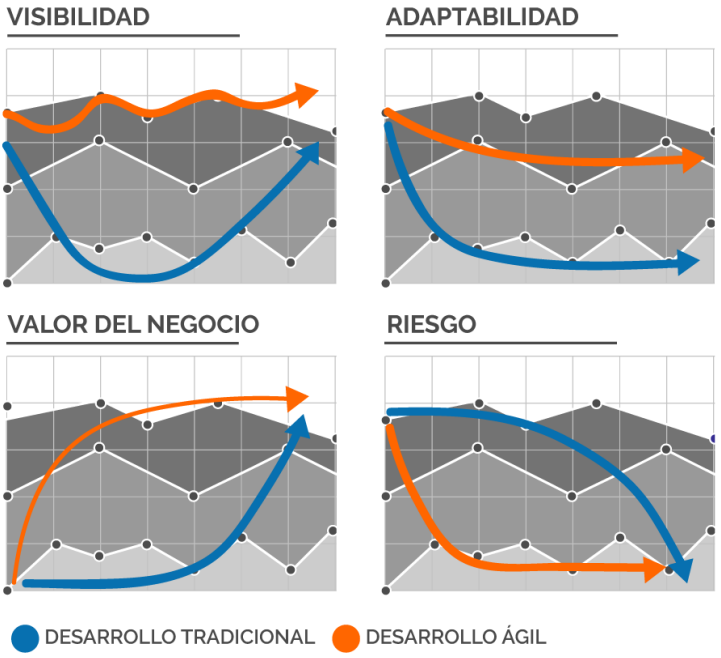
La mayoría de los fracasos se producen por aplicar ingeniería secuencial y gestión predictiva tanto en el proceso de adquisición (requisitos, contratación, seguimiento y entrega) como en la gestión del proyecto, en productos que no necesitan tanto garantías de previsibilidad en la ejecución, como respuesta rápida y flexibilidad para funcionar en entornos de negocio que cambian y evolucionan rápidamente".

SysOne entiende que el negocio de seguros está bajo constante cambio, sobre todo con modelos de competitividad y cambios regulatorios que generan una gran cantidad de cambios y necesidades, por lo tanto, enfoca su estrategia en la generación de valor mediante la gestión ágil.

***Esto último no quiere decir que no se planifique o no se tenga un plan, sino que se planificará de forma diferente, siempre teniendo en cuenta que lo importante es la entrega de valor agregado al negocio en forma incremental y sistemática**.*

El framework de gestión no soluciona los problemas, sino que los hace más evidentes y visibles para que se puedan comprender. Es responsabilidad del equipo de implementación solucionarlos.

Beneficios



En SysOne se utiliza metodologías ágiles para poder organizarnos internamente pero también para dar una mejor atención a nuestros clientes. Capacitamos a nuestros colaboradores y clientes en agilidad, adaptamos distintos frameworks (Scrum en su mayoría) para llevar a cabo proyectos complejos como la implementación de sistemas centrales.

Entregas sobre períodos cortos

Esto nos ayuda a mantener el riesgo bajo:

1. Entregas incrementales.
2. Iteraciones cortas.
3. Priorización de expectativas de clientes.
4. Sinergia.

Las prioridades del cliente son clave

El planeamiento del proyecto debe ser basado en las prioridades del cliente. En base al sistema central provisto por SysOne y la plataforma de desarrollo le vamos a ir parametrizando y agregando piezas en forma incremental para complementar la visión del cliente y mitigar sus dolencias.

Iteraciones cortas

Siempre trabajamos con ciclos cortos para limitar la complejidad de los requisitos venideros y obtener una pronta retroalimentación de las expectativas del cliente, los resultados del producto y el rendimiento de los procesos de trabajo.

Incremental

Utilizamos iteraciones incrementales para centrarnos en tener productos estables de excelente calidad. Crear sinergia para ser más productivos: Nos

aseguramos de compartir, aprender y sincronizar equipos diariamente para identificar impedimentos y siempre reflexionar eventos pasados (retrospectivas).

Simplicidad y transparencia

Nuestro acercamiento a la implementación y gestión para sistemas centrales se basa en una serie de elementos:

- **Artefactos:** Elementos indispensables y simples de entender, los mismos indican, qué y cómo se deben hacer las tareas, otorgando en todo momento transparencia y visibilidad de la entrega de valor.

- **Eventos:** Diferentes situaciones y evaluaciones a las cuales se someten las tareas para determinar si las mismas están entregando el valor apropiado al negocio.

- **Roles:** Posiciones y responsabilidades del equipo para poder generar valor al negocio mediante la ejecución de las tareas.

Artefactos

- **Product Backlog:** Las historias jerarquizadas por los responsables del negocio.

- **Sprint Backlog:** Es la lista de todo lo que el equipo scrum se compromete a alcanzar en el sprint. Una vez creado nadie puede agregar nada a excepción del equipo de desarrollo.
- Incremento de la estrategia o entregable.

Eventos

- **Sprint o iteración:** Es el corazón de Scrum. Espacio de tiempo para realizar una iteración con tareas. En nuestro caso será de dos semanas para ejecutar las estrategias, pudiéndose ajustar si es necesario.

- **Sprint Planning:** Reunión de planificación, donde todos los integrantes del equipo definen las actividades a realizar en un intervalo de tiempo, estimando el esfuerzo de cada una.

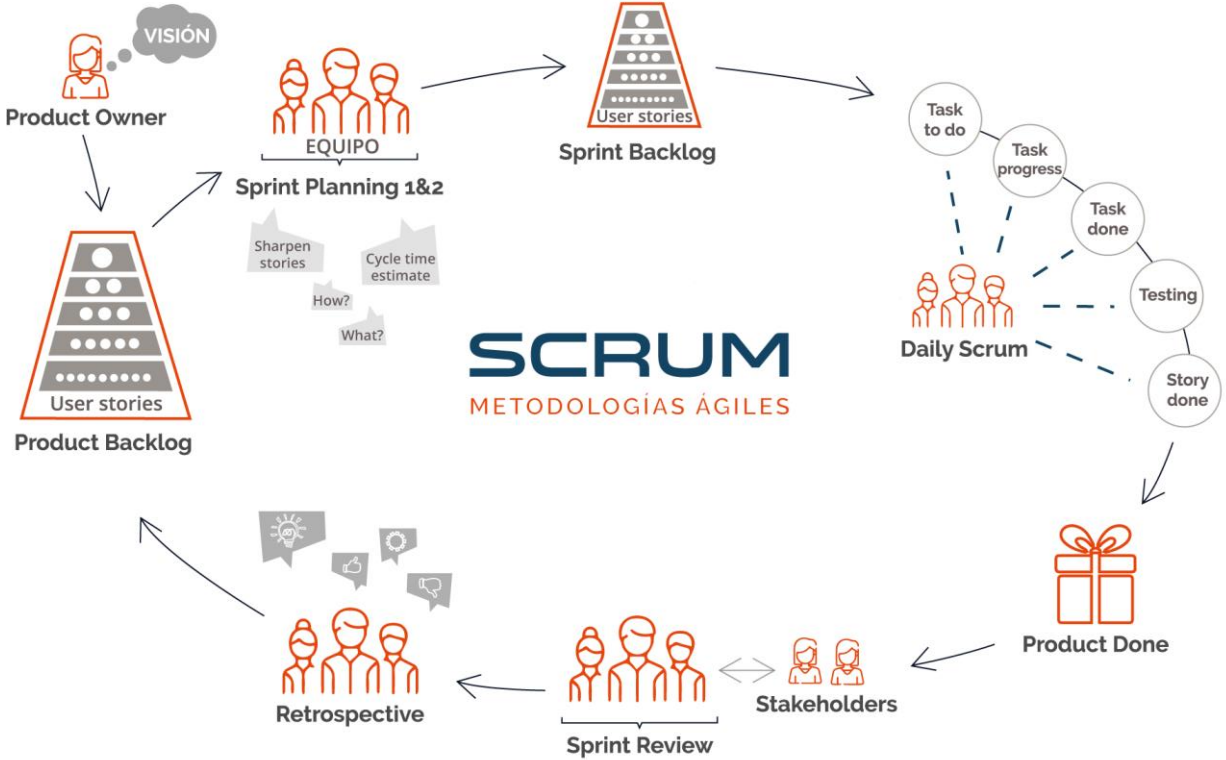
- **Sprint Backlog:** Luego de haber planificado las actividades, el Sprint Backlog es el conjunto de actividades priorizadas jerárquicamente.

- **Sprint Review**: Reunión al final del Sprint estratégico que valora los resultados del trabajo realizado durante el Sprint.

- **Retrospectiva**: Reunión que analiza qué hemos hecho, cómo lo hemos hecho y cómo mejorar el desempeño del equipo en futuros sprints durante la ejecución de la estrategia.

Roles

- Product Owner (PO).
- Scrum Master (SM).
- Equipo de colaboradores y desarrolladores / implementadores.


VISIÓN
Product Owner
EQUIPO
Sprint Planning 1&2
User stories
Sprint Backlog
Task to do
Task progress
Task done
Testing
Story done
Daily Scrum
Product Done
Stakeholders
Sprint Review
Retrospective
Sharpen stories
Cycle time estimate
How?
What?
User stories
Product Backlog
SCRUM
METODOLOGÍAS ÁGILES

Colaboración con nuestros clientes

La práctica ágil sistematiza la colaboración entre el cliente y el equipo:

- El equipo participa con el cliente en la creación del Backlog priorizado por el proyecto, en las reuniones de planificación del producto o proyecto, proporcionando la estimación de su esfuerzo y aportando mejoras, nuevas ideas e innovación.

- En el inicio de cada iteración, en la reunión de planificación de la iteración el equipo pregunta al cliente los detalles que pueda necesitar de los requisitos para poder dimensionar mejor el contenido de la iteración.

- Al finalizar cada iteración el equipo realiza una demostración al cliente de los requisitos completados.

Scrum sistematiza la colaboración dentro del equipo mediante las siguientes actividades:

1. Reunión de planificación de la iteración o Sprint.

2. Reunión diaria de sincronización del equipo o Daily Scrum.
3. Retrospectiva.
4. Sprint Review.

Concepción de un proyecto

SysOne entiende a los proyectos como una implementación ágil que consta de varias etapas de análisis y planeamiento, configuración de productos y reglas de negocios, pruebas continuas y deployment.

Derribando mitos

Los modelos ágiles o frameworks ágiles, no solo se utilizan para proyectos de desarrollo puro, sino que también aplican perfectamente para aquellos donde ya se encuentran ciertas funcionalidades desarrolladas y se deben configurar y adaptar algunas partes del mismo.

En la búsqueda del MVP

Concebimos los proyectos como mínimos productos viables, los cuales proveen una forma clara de ver las facilidades que otorgan las herramientas de SysOne para la configuración, construcción y migración de core de seguros.

Un producto mínimo viable es la versión mínima de un producto, que permite recolectar el mayor volumen de información relevante para el proyecto con el menor esfuerzo posible. Consiste en focalizarse en las características mínimas y necesarias para que el producto pueda salir al mercado, lo cual posibilita entre otras cosas poder sondear el mercado y maximizar el capital invertido (ROI).

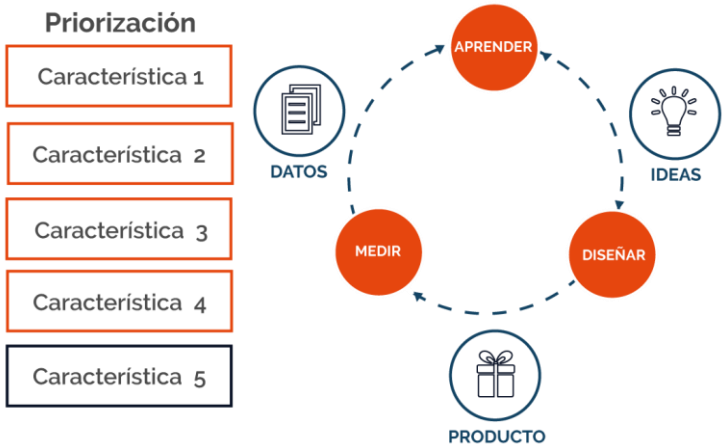
A pesar de su nombre, un MVP no se trata de algo que se construye por el simple hecho de probarlo o hacer algo rápido. El mismo solo vale si otorga valor al negocio, es utilizable en producción con los ajustes necesarios y está alineado con la estrategia de la aseguradora.

Un MVP se trata de crear negocios de seguros funcionales mediante la mayor frecuencia de interacción de entregas con los futuros usuarios. Es decir, que en vez de hacer un gran plan de proyecto de años y terminar un producto mucho tiempo después (algo así como un ciclo de vida en cascada en el mundo software), con el riesgo gastar mucho tiempo y dinero en algo que no vale para nada, la idea es concentrarse en el desarrollo de un MVP, es decir, una porción del producto, que englobe todas las funcionalidades y

características necesarias y pautadas para que el cliente pueda salir a producción.

Un MVP es parte central de la estrategia de éxito para la implementación de un core de seguros. Es un elemento central en un proceso iterativo de generación de ideas en base a la plataforma provista por SysOne, creación de productos, recopilación de datos, análisis y aprendizaje. Con un MVP se busca generar una primera versión del producto. El proceso se itera hasta que se obtiene un producto que se ajusta al mercado Minimum Marketable Product (MMP).

Al ponerle fecha a un MVP se convierte en una meta, si a la meta la dividimos en iteraciones se convierte en un plan, ese plan e iteraciones se apoyan en sprints y acciones y se convierten en entregables.



En toda metodología ágil, el producto debe construirse de forma incremental, en pequeñas entregas. La división del proyecto en estas entregas, debe ser llevada a cabo de forma criteriosa, con el fin de que cada sprint aporte valor para el cliente. Estos grupos de características forman el MMP, el conjunto más pequeño de funcionalidad que tiene valor en el mercado.

***Luego de la determinación y definición del MVP de un proyecto, se pueden planificar releases incrementales con versiones superadoras. No obstante, el MVP siempre debe ser una versión mínima productiva, es decir, que le brinde*

*al cliente un valor agregado listo para salir al mercado y obtener ROI***

¿Por qué concebimos así nuestros proyectos?

En la actualidad, todos los procesos de recambio de core de seguros implican altos riesgos para las aseguradoras y sus correspondientes negocios, los mismos van desde la continuidad de la operación, pasando por migración y exigencias regulatorias.

Como este proceso implica una alta inversión con alto riesgo, es necesario mitigar este esfuerzo con una metodología que otorgue la previsibilidad necesaria de que se está invirtiendo de la forma correcta.

La forma correcta de otorgar esta previsibilidad es sometiendo el proceso a entregas iterativas, probadas y verificadas por el cliente, que permitan en todo momento medir si el esfuerzo realizado está entregando valor al negocio.

Algunas observaciones que responden la pregunta de por qué utilizamos y concebimos los proyectos de esta forma son:

- No es rentable invertir una gran cantidad de dinero, recursos y esfuerzo para implementar una herramienta la cual no se sabe si es apta para un determinado tipo de organización, más aún detectar que no es apta luego de haber transcurrido un largo período de tiempo (años).

- SysOne y las aseguradoras necesitan tener en forma temprana feedback del avance de los proyectos y sus correspondientes riesgos.

El triángulo de hierro tradicional vs. ágil

En todo proyecto, gestionado de la forma tradicional existen tres variables implícitamente relacionadas:

- **Alcance:** Determinación de cuántos requisitos o tareas hay que realizar.

- **Tiempo:** Planificación de cuánto durará el proyecto.

- **Costo:** Estimar los recursos (dinero, personas), es preciso para emprender un determinado proyecto.

A esto es lo que se denomina triangulo de hierro tradicional, donde cualquier cambio en una de las variables afecta de forma directa a por lo menos una de las otras dos y de forma implícita a la calidad del producto. Aquí se estima que el cliente pueda fijar dos de estas variables, dando flexibilidad a la tercera:

Por ejemplo: si se fija el alcance y el costo, se debe dejar que sea el equipo del proyecto quien fije el tiempo. Sin embargo, en la realidad el cliente casi siempre desea tener el control sobre las tres variables, lo que hace que no exista flexibilidad.

Jim Highsmith propuso el triángulo ágil, donde se considera que el alcance, costo y tiempo son más bien restricciones y los reúne en una sola variable. Es así, como éste triángulo tiene como variables:

- La calidad.
- El valor.
- Las restricciones (alcance, costo, tiempo).

Las restricciones deben ser gestionadas para maximizar la calidad y el valor esperado por el cliente, esto concuerda con los principios ágiles.

El valor es la medida de satisfacción lograda en el cliente cuando el producto o servicio le brinda lo que desea y necesita. Bajo este esquema un proyecto satisfactorio es aquel que provee un producto o servicio de calidad y que genera el más alto valor posible al cliente.

Entendemos y manejamos el riesgo

Con el uso de un framework ágil y visionando un mínimo producto viable, es más fácil de entender y manejar el riesgo de cada proyecto. También estos frameworks conducen al equipo a que tenga una disciplina para identificarlos y tratarlos a tiempo.
Para mantener el riesgo bajo siempre se tienen en cuenta los siguientes ítems:

1. **Las prioridades del cliente son clave:** Un plan de proyecto debe basarse en las prioridades del cliente. Primero obtenemos un producto "central" tangible y agregamos "piezas" gradualmente para completar la visión total del negocio.

2. **Repeticiones cortas:** Siempre trabajamos con ciclos cortos para limitar la complejidad de los requisitos venideros y obtener una retroalimentación temprana de todas las expectativas del cliente, los resultados del producto y el rendimiento de los procesos de trabajo.

3. **Incremental:** En SysOne utilizamos iteraciones incrementales para centrarnos en tener productos estables de excelente calidad.

4. **Crear sinergias para ser más productivos:** Todos nos aseguramos de compartir, aprender y sincronizar equipos diariamente para

identificar impedimentos y siempre reflexionar sobre eventos pasados (retrospectivas).

Propósitos del MVP

El mínimo producto viable MVP, persigue los siguientes propósitos en la concepción de un proyecto:

- Con la herramienta, recursos y equipos de profesionales provistos y configurados por SysOne, poder probar que la misma es suficiente y superadora a la competencia.

- Gestar entregas funcionales de productos de seguro a bajo riesgo.

- Generar aprendizaje para acelerar la implementación o migración del resto de los productos de una aseguradora.

- Reducir la cantidad de horas de ingeniería en funcionalidades no relevantes para los usuarios y el negocio.

- Demostrar la habilidad de SysOne y las competencias técnicas del producto en etapas tempranas de implementación.

- Probar que la estructura brindada por la aseguradora para llevar a cabo el proyecto es satisfactoria y soporta la implementación del proyecto en todas sus etapas.

Definir un MVP maximizando su valor

Queremos que el Mínimo Producto Viable nos sirva para maximizar la entrega de valor al cliente en la iteración planteada. Para poder dar este enfoque al MVP debemos seguir los siguientes principios:

- **Funcional:** El Mínimo Producto Viable debe ser funcional. Aunque estamos hablando de una versión reducida de un negocio de seguros, debe ofrecer cierta integración con el resto del ecosistema de la aplicación (Ver capítulo catálogo de interfaces). El objetivo es que el usuario pueda utilizar el MVP en un contexto normal o el más aproximado a la realidad posible.

- **Analítico:** Cada entrega de MVP nos debe proporcionar aprendizaje y valor sobre el negocio de seguros que estamos resolviendo, debe ser una fuente de nuevas correcciones y

tareas a realizar para la próxima iteración de manera tal que genere más valor al negocio.

- **Consistente:** Debe ser consistente con el resto del ecosistema donde está inmerso este proyecto, de manera tal que funcione en forma armónica con el resto de las aplicaciones y negocios con los cuales convive.

Roles y equipos de SysOne

En un proyecto ágil y disciplinado, cualquier persona puede estar en uno o más roles, y también puede cambiar su (s) rol (s) con el tiempo. Los roles no son puestos ni están destinados a serlo.

Los equipos que practican agile se adaptan a sus necesidades y pueden variar en su ejecución exacta. Sin embargo, todos tienden a tener los mismos tipos de roles y procesos muy similares en su núcleo.

Agile resta importancia a los roles especializados y considera a todos los miembros del equipo iguales: todos trabajan para ofrecer una solución independientemente de su trabajo. Con la excepción de las partes interesadas, todos desempeñan efectivamente el papel de miembros del equipo.

En este capítulo, explorará los roles, proporcionándole una base para comprender cualquier estilo que pueda experimentar.

Cabe destacar que existen 3 (tres) perfiles primordiales en agilidad:

- Product Owner (PO).
- Scrum Master (SM).
- Equipo de Desarrolladores / Implementadores.

Dentro de este último ítem (equipo de desarrolladores / implementadores) pueden encontrarse los desarrolladores propiamente dicho, así como también los colaboradores y perfiles técnicos (por ejemplo: líderes de equipo, arquitectos, equipo de UX/UI, y todo el equipo de testing quienes aportan calidad al producto final).

Todo el equipo de trabajo

Actividades

01
PARTICIPAR
En las Retrospectivas.

02
UTILIZAR
Jira como la herramienta formal de tareas.

03
SOLICITAR
Las necesidades requeridas por el equipo (lugar, insumos).

04
FOMENTAR
La comunicación y estar alerta acerca de novedades.

05
GESTIONAR
La formación de equipos con los integrantes necesarios.

Responsabilidades

01
VELAR
Por la puesta a producción.

02
ACOMPAÑAR
A los nuevos integrantes en el proceso de inducción.

03
DEFINIR
El mecanismo de soporte SLA (dirección y comercial).

04
EMPLEAR
La metodología de trabajo acordada en equipo.

05
VERIFICAR
La disponibilidad de los ambientes para realizar pruebas.

Product Owner (PO)

Actividades

01
REALIZAR
El User Story Mapping (alcance).

02
COORDINAR Y MEDIAR
Reuniones periódicas con el cliente (manejar expectativas).

03
PRESENTAR Y DEFINIR
Las historias en la planning, y compromiso en los sprint.

04
EJECUTAR
Pruebas funcionales, reportes de bugs, recopilación de métricas, informe de avance.

05
DEFINIR CON EL CLIENTE
El calendario de Sprints, tiempos y releases (negociar los cambios futuros).

06
DOCUMENTAR
Los procesos de negocio.

Responsabilidades

01
TENER Y TRANSMITIR
La visión del producto.

02
COORDINAR Y MEDIAR
Comunicación fluída con el cliente, usuarios y el equipo.

03
CONOCER Y COMUNICAR
Las tareas técnicas (de seguros) y alcance del proyecto.

04
PLASMAR Y GESTIONAR
La funcionalidad y estado de todos los sprint en Jira.

05
ACOMPAÑAR
En los procesos de ventas para definir el acance total del proyecto en sus diferentes etapas.

06
PARTICIPAR
En las reuniones de PO.

Scrum Master (SM)

Actividades

01
FACILITAR
Al equipo de trabajo.

02
MAXIMIZAR
El valor del negocio dentro de un marco ágil.

03
PROPICIAR Y FORMAR
Equipos auto-organizados y multifuncionales.

04
DETECTAR
Posibles mejoras surgidas en las retrospectivas.

05
SOLUCIONAR
Posibles impedimentos que pudieran surgir durante el sprint.

06
DOCUMENTAR
Los procesos de negocio.

Responsabilidades

01
PROMOVER
Las buenas prácticas.

02
CONOCER Y COMUNICAR
Las características y entendimiento de la agilidad.

03
PLASMAR Y ACTUALIZAR
El estado de todos los sprint en Jira.

04
GARANTIZAR
Que haya una definición clara y concisa de "Done".

05
ENSEÑAR
Al Product Owner (PO) a priorizar y gestionar efectivamente el Product Backlog.

06
PARTICIPAR
En las reuniones de equipo.

Equipo de Implementadores

Actividades

01
REALIZAR
Todos los commits.

02
COMUNICAR
Si al resolver un bug se halla un problema en el producto.

03
ALERTAR
En caso de detectar la falta de definiciones en historias.

04
DOCUMENTAR
Acerca del desarrollo en el fuente y en confluence.

05
EJECUTAR
Pruebas unitarias y funcionales para cumplir los criterios de aceptación.

Responsabilidades

01
RESPETAR
Convenciones de desarrollo.

02
MERGEAR Y GARANTIZAR
El correcto funcionamiento del sistema al comitear.

03
ESTIMAR
Las historias de usuario en Jira, actualizando su estado.

04
PROPICIAR
La mejora continua en el código.

05
INFORMAR
En caso de no llegar o tener impedimentos para cumplir con los issues pactados en el sprint.

Equipo de Testing

Actividades

01
REALIZAR
Las pruebas de la solución.

02
EJECUTAR
Testing técnico, funcional y de interfaces.

03
REALIZAR
Redacción, refinamiento y priorización de bugs.

04
EMPLEAR
Las historias de usuario y necesidades del cliente.

05
DOCUMENTAR
Todo lo relacionado con las pruebas.

Responsabilidades

01
CREAR
Criterios de aceptación.

02
REPORTAR
Potenciales errores ó problemas de la plataforma.

03
COORDINAR
La ejecución de pruebas manuales y automáticas.

04
REVISAR
Los reportes de SLA con el cliente.

05
OPTIMIZAR Y MANTENER
El entorno de pruebas (hardware, software y red).

Análisis Funcional

Cuando enfrentamos un análisis de un sistema central, priman las siguientes 3 (tres) verdades:

- Es casi imposible tener todos los requerimientos documentados y analizados en profundidad desde el principio del proyecto.

- Los requerimientos siempre van a cambiar, y, es crucial contar con los skills y flexibilidad para adaptarse a ellos.

- Siempre van a existir más funcionalidad para implementar que dinero disponible para hacerlo.

Dentro del proceso de análisis se desarrollan las siguientes etapas que son fundamentales:

User Story Map

¿Qué es un User Story Map?

El User Story Mapping (denominada también en algunas ocasiones como Visual Story Map es una técnica creada por Jeff Patton comúnmente utilizada

para representar de forma visual necesidades o una idea de producto, brindando como resultado versiones del producto y plan de entregas, ayudando a tener un panorama más claro de lo que se va a construir.

Ofrece una vista general de todas las funcionalidades que lo componen, convirtiéndose en una forma de reorganizar el Product Backlog en dos dimensiones: una para el tiempo (medido en Releases) y otra dimensión para las funcionalidades.

¿Cómo se construye?

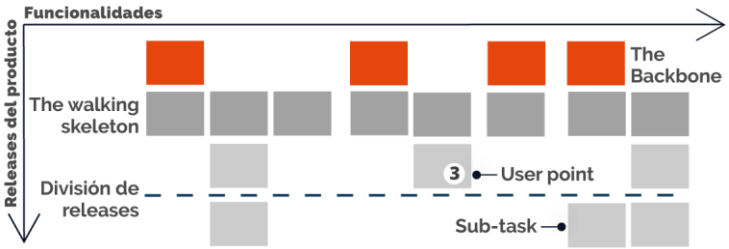
Identificar las distintas grandes funcionalidades en las que se divide la plataforma. Estas serán como la columna vertebral del proyecto. (The Backbone).

Descomponer esa actividad en las historias de usuarios más básicas que ofrecen funcionalidad de punta a punta en el sistema. Estas no deben priorizarse y permiten dar contexto a las User Stories. (The Walking Skeleton).

Identificar las historias de usuarios más específicas que deben ser desarrolladas, es importante tener en cuenta que cada una de ellas debe entregar valor y estar ordenadas, siendo las de más arriba más críticas

o prioritarias (User Task). Cada una de estas tareas va a tener incorporado un story point, que será un valor que se le otorga en función del esfuerzo requerido para su resolución (este valor debe ser consensuado entre el equipo, puede ser numérico o por alguna escala previamente definida como small, medium, large, etc).

Luego se realizan recortes de funcionalidad a entregar con líneas dibujadas. Cada una de estas representará una release del producto, reflejando visualmente todas las funcionalidades que incluye cada una.
El siguiente gráfico esquematiza lo desarrollado recientemente:



¿Por qué es importante?

Muchas veces para entender el alcance total (y haciendo uso de las herramientas que nos dan las metodologías ágiles) se realiza un User Story Map

durante la venta del proyecto. Esta actividad nos resultó muy beneficiosa y sirve mucho para dar una buena dimensión del alcance del proyecto en una etapa muy preliminar. En general en esa actividad suelen participar distintos interesados que luego no formarán parte del día a día del proyecto.

Si bien esta actividad es muy importante no tenemos que confundirla con un compromiso o con el alcance del proyecto actual. Dependiendo el nivel de ese User Story Map se podrá utilizar más o menos de ese artefacto para realizar el User Story Map del proyecto que mostrará con más información como se desarrollará el proyecto.

El User Story Map debe marcar el inicio del proyecto y es el que definirá un alcance inicial. Si bien está basado en el relevamiento y en el User Story Map de ventas (en el caso de que se haya realizado) se propone tomarse un período prudencial en el que el Product Owner (PO) puede definir a alto nivel todas las historias y proponer el mapeo al cliente, y si es posible al equipo de desarrollo.

Es fundamental que cuando el Product Owner tiene su primer borrador del User Story Map, se realice una jornada con todos los involucrados: dirección del

proyecto (Stakeholders), representante del cliente, equipo de desarrollo y Product Owner.

En esta reunión se debe acordar las tareas a realizar y el User Story Map que se va a utilizar en el desarrollo. Este User Story Map puede cambiar con el tiempo, pero siempre como un acuerdo entre SysOne y el cliente.

El User Story Map propuesto es del libro "User Story Mapping: Discover the Whole Story, Build the Right Product" de Jeff Patton.

Este se basa en tener tres dimensiones:

- Una primera dimensión que es un gran agrupamiento que nosotros llamamos componentes.

- Un segundo nivel de pasos para completar ese componente que son las historias de usuario épica.

- Un tercer nivel donde debajo de cada épica se enumeran los requerimientos a tomar que serían las historias de usuario. Las historias de usuario deben

ser pequeñas y deben entrar en un sprint y ser redactadas en un lenguaje funcional para que el cliente pueda validarlas. Estas serán pasadas a Jira (https://jira.sysone.com), que es la herramienta empleada por SysOne para el seguimiento y avance del Proyecto.

¿Qué son las historias de usuarios?

La gestión de requisitos es un proceso clave del proyecto, el inconveniente se presenta en cómo conseguir que lo que se encuentra escrito en los documentos sea realmente lo que el usuario necesita. Agilidad aborda este inconveniente de una manera diferente, sin que esto quiera decir, como a veces se mal comprende, que no se debe realizar gestión de requisitos. Mike Cohn en su libro “User stories applied for agile software development”, confirma que la gestión de requisitos, sea cual sea la metodología que se use, es vital para un proyecto.

Las historias de usuario son una descripción breve, de una funcionalidad tal y como la percibe un usuario, son el puente de colaboración con el cliente que debería por lo menos tener los siguientes elementos:

- **Identificador:** Un identificador para la historia de usuario.

- **Título:** Título que describe a la historia de usuario.

- **Descripción:** Mike Cohn recomienda seguir el patrón como [ROL] quiero [FUNCIONALIDAD] para [BENEFICIO].

- **Pruebas de aceptación:** Criterios de aceptación consensuados con el equipo para definir cuando la tarea se da por completada.

- **Estimación:** Normalmente medida con puntos de historia.

- **Valor:** El valor que tiene la historia para el usuario, este elemento permitirá priorizar el trabajo en un momento posterior.

Algo muy importante es que el detalle de las historias debe salir de conversaciones con el usuario, tomando en cuenta que puede darse casos en los que, a pesar de la flexibilidad de esta herramienta, no sea la apropiada para algún contexto específico, en el cual

sea necesario expresar la necesidad de una forma diferente.

¿Cómo descomponer historias de usuarios?

Es importante para poder lograr una comprensión de una historia de usuario, poder desglosarlas en unidades pequeñas y simples. Hay muchas técnicas diferentes para desglosar historias, pero el más efectivo al momento es el método “SPIDR” de Mike Cohn. También es uno de los métodos más simples una vez que se comprenden los conceptos básicos.

“SPIDR” es un acrónimo de 5 (cinco) técnicas utilizadas para dividir historias de usuarios:

Spike

Es una técnica que se utiliza como último recurso, no debería ser muy usual; es cuando la tarea que vamos a enfrentar requiere de más investigación y comprensión.

Se dice que, con este aumento del conocimiento, es más probable que comprendamos mejor la historia y que luego dividamos una historia grande en muchas más pequeñas.

Muchas veces los Stakeholders o Product Owners no proveen los detalles correspondientes para poder lograr comprender una historia en su totalidad, esto puede ser dado su complejidad.

Por ejemplo: motor de capitalización de vida con inversión en nuestro core de vida. Sin una comprensión mucho mejor de la historia, el equipo de desarrollo y los Product Owners corren el riesgo de entregar un producto de calidad inferior al esperado por el negocio.

Path

Un camino (path) describe un escenario donde existen flujos alternativos hacia el mismo objetivo final. Tomemos un ejemplo de un proyecto reciente. La historia de alto nivel es que un cliente necesita poder

pagar un seguro dentro de una tienda. Esta historia tiene muchos caminos disponibles. *Por ejemplo, la capacidad de pagar en efectivo, con tarjeta de crédito (tanto con chip como con PIN y sin contacto), cuenta de crédito del cliente y pago por móvil, incluidos tanto el pago de Apple como el de Android.*

En este caso, la historia original se puede dividir en 6 (seis) historias más pequeñas, cada una con sus propios detalles específicos y pruebas individuales.

Como Product Owner, tendríamos que considerar el valor de negocio por cada camino individual de la división, analizando si 6 historias ofrecen más valor que 4 (cuatro) historias, en donde 2 (dos) de ellas tienen el mismo requisito que cubre múltiples métodos de pago.

Podríamos entonces estimar cada historia y en lugar de hacer todo en un sprint, podríamos hacer lo más valioso por ahora y trabajar en otro más adelante en el proyecto.
Cada camino (path) nos permite especificar cada ruta en su propia historia, luego volver a agrupar ciertas rutas si no ofrecen ningún valor adicional que se traten como historias individuales.

Interfaces

Las interfaces se refieren a la cantidad de detalles a través de uno o muchos diferentes canales. Tomemos otro ejemplo. Una vez completado, este proyecto ofrecerá al cliente la capacidad de realizar cotizaciones en varios canales, incluido el sitio web y el sitio móvil (optimizado para teléfono y tableta), pero para la fase inicial, hemos decidido ofrecer el servicio solo a través del sitio web.

Si bien los clientes de dispositivos móviles son un subconjunto importante de clientes, el mayor valor es sacar el proyecto, obtener retroalimentación de los clientes y medir el rendimiento inicial para que podamos atender mejor al cliente.

Seguirá Mobile, y cuando lo haga, probablemente será una oferta de cliente aún mejor y más pulida basada en el hecho de que hemos dividido la historia inicial en canales individuales.

La división de historias basada en interfaces ofrece una mayor cantidad de flexibilidad y la capacidad de estimar con mayor precisión el tiempo de desarrollo y prueba requerido para cada interfaz.

Data
Dividimos nuestras historias en función de los datos más valiosos y que ofrece la mejor rentabilidad en el tiempo invertido para el cliente / negocio. Dividir historias basadas en datos proporciona un mayor grado de flexibilidad.

Rules
Reglas comerciales, cuestiones regulatorias, canales, etcétera; todas ofrecen el potencial de dividir historias.

Tomemos como ejemplo una aseguradora de personas que posee un producto que requiere de una suscripción. Como sabemos la suscripción de personas está basada en varias variables como pueden ser: declaración de salud, coberturas, sumas aseguradas, tipo de actividad de la persona, riesgo crediticio, entre otras.

Como sabemos para que una persona o riesgo sea asegurable tendremos una regla de análisis de riesgo para cada una de las variables que previamente mencionamos, cada regla tiene una naturaleza completamente diferente. Para dividir la historia, podemos hacer una regla (la que entregue más valor al negocio en una etapa temprana) y omitir el resto de las primeras iteraciones. Las reglas adicionales se

entregarán en iteraciones posteriores, pero para probar la funcionalidad de suscripción y todo su circuito con una regla es suficiente, de esta manera, podemos entregar valor de forma temprana sin necesidad de tener toda la complejidad de la suscripción implementada. Luego el negocio priorizará que otras reglas se van a implementar en futuras iteraciones.

Dividir la historia según la inclusión / exclusión de las reglas permite una entrega y una prueba más rápida. El valor de las reglas se puede realizar en una iteración posterior cambiando la versión que ahora funciona.

Esto obviamente no es viable si no se posee un producto orientado a servicios o reglas donde la dinámica de agregar o modificar reglas tiene un costo mínimo o nulo. En el caso de la plataforma de SysOne esto es así, por lo tanto, se debe aprovechar mucho esta facilidad que la plataforma ofrece.

Criterios de aceptación de Historias

Un criterio de aceptación ayuda a determinar si la definición de una historia de usuario se desarrolló en función de las expectativas del propietario del producto. Normalmente será cuando se cumpla con todos los criterios de aceptación cuando se dará por

terminada la tarea. Una técnica muy importante para escribir los criterios es aquella que viene dada por el comportamiento BDD (Behavior driven development).

Para escribir criterios guiado por comportamiento se debe tomar en cuenta tres partes fundamentales: Dada una condición (describir el contexto), cuando ocurre un evento o acción (o varias utilizando operadores lógicos), entonces se describe las acciones que se deben realizar. Esta técnica ayuda mucho para realizar pruebas automatizadas.

Sea cual sea la técnica utilizada para la escritura de los criterios de aceptación es muy importante tomar en cuenta que estos posibilitan las siguientes 3 (tres) misiones:

- Definir detalladamente todas las historias de usuario.

- Especificar los requisitos funcionales de un software en su totalidad.

- Refinar las historias de usuarios mediante la comunicación entre todos los miembros del equipo, en la planificación de una iteración o sprint.

Cuando la historia cumple con todos los criterios de aceptación la historia se da como hecha o finalizada. El propietario del producto es quien comprueba el cumplimiento de los criterios de aceptación.

Los elementos básicos que debería tener un criterio de aceptación guiado en comportamiento son:

- Identificador del escenario.
- Título.
- Contexto.
- Evento.
- Resultado.

Mapa de impacto (Impact Map)

¿Qué es un Impact Map?

El Impact Map es una técnica de planificación que permite entre otras cosas poder generar una versión inicial de un Product Backlog.

La misma se concentra en determinar con la mayor precisión posible qué es y cómo se debería actuar para obtener aquello que se desea conseguir

(objetivo/metas). Para ello, se estipulan cuatro preguntas que serán la base de esta herramienta: ¿Por qué?; ¿Quiénes?, ¿Cómo?; ¿Qué?

Backlog

Antes de comenzar a trabajar ya sea configurar la plataforma o desarrollo, se deberá poseer un backlog del producto a implementar. Generalmente se generan backlogs por producto de seguros a convertir en MVP (mínimo producto viable). Como resumen un backlog es una lista priorizada y ordenada de funcionalidades o historias de usuarios desde la perspectiva del cliente. Es la lista de funcionalidades que se van a entregar en un cierto time box o sprint. Es responsabilidad del Product Owner y Scrum Master de mantenerlo actualizado con el cliente durante todo el ciclo de vida del proyecto. El backlog de producto existe y evoluciona durante toda la vida del proyecto, es la hoja de ruta del mismo.

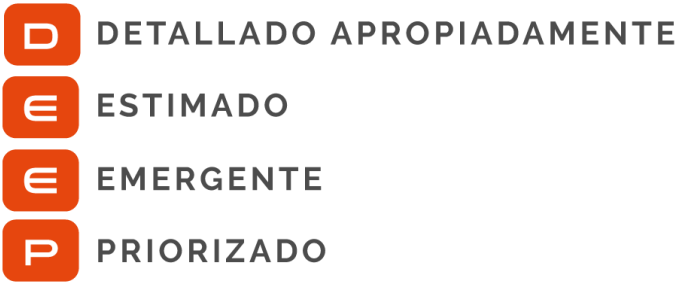
En cualquier momento, el backlog de productos es la visión única y definitiva de “todo lo que va a ser realizado en algún momento por el equipo en orden de prioridad”. Sólo existe un backlog de producto para cada producto de seguro, lo cual significa que cada product owner deberá tomar decisiones de prioridad sobre todo el abanico de opciones en representación

de los intereses de los stakeholders (incluyendo el equipo).

Prioridad	Historias de usua	Detalles	Estimación	Sprint (1,2,3,4,"n")
1	Como vendedor quiero cotizar los productos de vida con inversión desde una tablet.	...	5	
2	Como vendedor quiero que las cotizaciones se realicen en menos de dos segundos.	...	10	

Es importante notar que una entrada del backlog no solo satisface funcionalidad, sino que también se pueden tomar como elementos del backlog requerimientos asociados a la velocidad o performance, como es en el ejemplo la segunda entrada. Las historias de usuario del backlog de producto se articulan de cualquier manera que sea clara y sostenible, sea cual sea la aproximación, la mayor parte de los elementos deberían enfocarse en proporcionar valor a los clientes.

Un buen backlog es DEEP (Profundo):



Detallado apropiadamente

Las historias de usuario de mayor prioridad están más detallados y especificados que los de menor prioridad, ya que se trabajará antes en los primeros que en los segundos.

Por ejemplo, el 10% inicial del Backlog puede estar compuesto elementos muy pequeños y bien analizados, mientras que el 90% restante pueden estarlo mucho menos. Por otro lado, es muy relevante, que el detalle y refinamiento de los elementos se haga por anticipado a su ejecución, esto quiere decir que si nos encontramos en el sprint 1, el Product Owner y los Stakeholders deberían estar definiendo y detallando los elementos con mayor profundidad del sprint 2 y 3.

Estimado

Los elementos de que componen la siguiente versión de una entrega (release) deben estar estimados y, lo que es más, deberían revisarse y re-estimarse cada Sprint conforme vamos obteniendo nueva información. El equipo proporciona al Product Owner estimaciones de esfuerzo para cada elemento del Backlog, y quizás también proporcione estimaciones de riesgo técnico.

El Product Owner y otros Stakeholders proporcionan información sobre el valor de cada petición, lo cual puede incluir ingresos producidos, reducción de costes, riesgo de negocio, importancia para distintos Stakeholders y otros datos.

No existe elemento de Backlog sin estimación del equipo de implementación, es ideal que un elemento o historia de usuario sea lo suficientemente pequeña para que entre dentro del lapso de un sprint.

Emergente

Debido a la variabilidad y al aprendizaje experimentado, el Backlog es refinado regularmente. En cada sprint se pueden añadir, eliminar, modificar, dividir o cambiar la prioridad de los elementos. Así, el Product Owner actualiza constantemente el Backlog

para reflejar cambios en las necesidades del cliente, nuevas ideas o percepciones, movimientos de la competencia, obstáculos técnicos que aparecen, etc.

Debido al proceso de aprendizaje continuo sobre las necesidades del cliente, será lógico producir mayor cantidad de elementos a medida que evolucionan los Sprints.

Priorizado

Los elementos del Backlog están priorizados de manera ordenada, de 1 a “N”. En general, las historias de usuario de máxima prioridad deberían producir el mayor valor agregado al negocio. Otro posible motivo para priorizar una historia de usuario es atacar los riesgos de forma temprana, antes de que los riesgos te ataquen a vos.

¿Dónde gestionamos el Backlog?

En SysOne, el único lugar dentro del proyecto donde se detallan todas las historias de usuarios y el Backlog es en la aplicación Jira. Desde esta aplicación el equipo de desarrollo e implementación puede no solo documentar las historias de usuarios y saber quién es el responsable por la misma, sino también es posible medir de forma precisa la velocidad con la cual se

resuelven para aportar métricas a la gestión al proyecto.

Esta aplicación siempre se encuentra visible y accesible no solo para el equipo de desarrollo sino para todos los participantes del proyecto, aportando transparencia en el proceso de implementación y compartiendo las métricas a todos en tiempo real.

Clasificación de esfuerzo dentro del Backlog

- Configuración.
- Desarrollo de reglas.
- Desarrollo de servicios y software a medida.

Una métrica por la cual podemos cuantificar la priorización es utilizando a Pareto, el principio de Pareto (también conocido como la "Regla 80/20") se puede aplicar a muchas áreas de negocios.

Por ejemplo:

- *En ventas, el 80% de su negocio proviene del 20% de su base de clientes.*

- *En productividad, el 80% de sus logros provienen del 20% de su lista de tareas.*

- *En las empresas del día a día, el 80% del trabajo que realiza proviene del 20% de sus ofertas de servicios.*

- *En marketing, el 80% de la respuesta lograda proviene del 20% de sus esfuerzos de marketing.*

- *En el servicio al cliente, el 80% de las quejas provienen del 20% de sus clientes.*

- *Y, en su backlog, el 80% del valor entregado vendrá del 20% de lo que hay en su backlog.*

(Fuente empleada para ejemplificar la distribución de Pareto: http://en.wikipedia.org/wiki/Pareto_distribution).

Con el espíritu de administrar eficazmente el trabajo con el objetivo de brindar un gran valor y satisfacción a nuestros clientes, recomendamos seleccionar cuidadosamente la lista de tareas pendientes en el Backlog.

Esto significa identificar el 80% de las características que no hacen felices a los clientes y eliminarlos. Encontrar el 20% que satisface a los clientes, implementarlos y repetir el proceso.

El principio de Pareto reconoce el desequilibrio en el esfuerzo y los resultados; y nos permite usar ese desequilibrio a nuestro favor. Pero eso no significa que podamos cancelar todo lo que cae en el 80% no crítico o menos crítico. No podemos ignorar los requisitos regulatorios, por ejemplo; sin embargo, podemos modificar las acciones para que nos concentremos más donde sea necesario (entrega de valor).

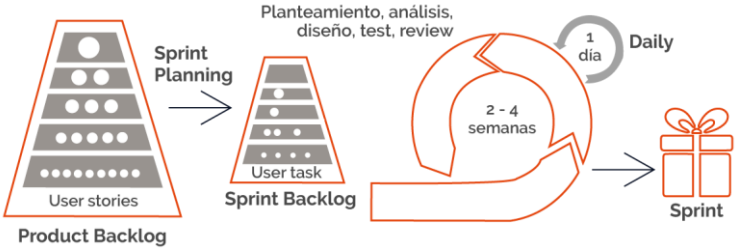
El principio de Pareto se trata de reconocer el desequilibrio de lo que se necesita hacer para alcanzar la satisfacción / finalización / felicidad y hacer solo eso, no un trabajo adicional que no genere un valor positivo. Este esfuerzo adicional debe reservarse para el crítico 20%.

El objetivo de aplicar el principio de Pareto al Backlog es que debe centrarse principalmente en el crítico 20% para alcanzar el 80% de satisfacción del cliente. Luego ejecutar y repetir.

Por supuesto, esta es una regla general, no una estadística irrefutable; para algunas acciones, el desglose puede estar más cerca de 90/10 o 70/30. Pero el punto es el mismo, y puede ser inquietantemente preciso cuando comienzas a ver qué entregas no son

agradables para los clientes (o características que tus clientes no están usando en absoluto). El desafío radica en identificar ese crítico 20%. Con algunas áreas que tienen métricas mensurables (como el número de clientes, cantidad de productos, la cantidad de pólizas), es una obviedad. Pero puede ser difícil hacer el mismo análisis y aplicarlo a su Backlog, especialmente cuando tiene una larga lista de cosas por hacer con muchos elementos que deben completarse.

Sprint, time box



El corazón de nuestra metodología se basa en el concepto de “Sprint”. Un sprint es un intervalo de tiempo prefijado durante el cual el equipo crea, configura o desarrolla una entrega o “incremento” de un producto determinado acorde a las “definiciones de terminado” (Definition of Done). Como principal característica es que al final del sprint el incremento tiene que ser utilizable, generando valor al negocio.

A lo largo del desarrollo hay entregables consecutivos de duración constante. Cada uno de ellos entregando valor en forma continua.

Cada sprint lo podemos comprender como un mini proyecto, el cual tiene un propósito específico planteado y especificado por el Product Owner con el negocio.
Por ejemplo: "Este sprint se basa en la cotización de vida con inversión para dos fondos mutuos". Ese propósito alinea a todos los integrantes a focalizar sus esfuerzos con el fin de hacer el propósito planteado realidad.

Cada sprint cuenta con una definición de lo que se va a lograr, un diseño y un plan flexible que guiará la construcción, las pruebas y la entrega esperada. Dentro del marco de desarrollo de un Sprint se llevan a cabo diferentes etapas que colaboran a la efectividad en la implementación de un Core de SysOne.

Las etapas son:

- Reunión de planificación.
- Reunión diaria.
- Reunión de revisión del sprint.
- Reunión de retrospectiva.

SPRINT
Ciclo de la iteración

01. Mercado: Preguntas y respuestas de funcionalidades y procesos a realizar. Analizar el mercado de seguros en distintos escenarios. Esta etapa es un trabajo en conjunto, de intercomunicación y feedback.
02. Análisis / Conclusiones: Se evalúa las alternativas para llevar a cabo cada tarea, determinando las mejores prácticas para cada caso.
03. Documentación: Se documenta la información analizada y tareas a ejecutar. Se proporciona material didáctico a todos los actores claves.

SPRINT PLANNING
Planificación de sprints

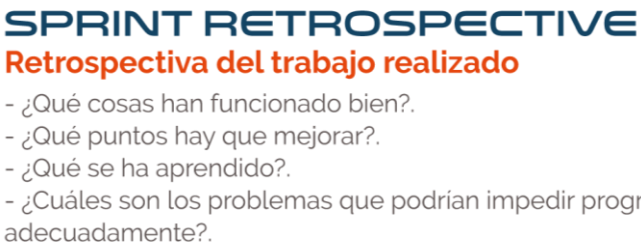
- ¿ Qué vamos a hacer?, ¿ Cómo?.
- Listado de requisitos.
- Priorización de tareas.
- Elaboración de estrategia y objetivos.
- User stories por sprint.
- Estimación de esfuerzo.

SPRINT REVIEW
Presentación de prototipo

- Requisitos realizados.
- Obtención de feedback.
- Registro de adaptaciones necesarias.
- Replanificación de nuevos cambios.

DAILY SPRINT
Reunión diaria de seguimiento

- ¿Qué he hecho la última reunión?.
- ¿Pude hacer todo lo planeado?.
- ¿Tuve problemas en el trabajo?.
- ¿Pude hacer todo lo planeado?.
- ¿ Qué voy a hacer en el próximo sprint?.
- ¿ Existen impedimientos o dependencias?.



Reunión de planificación: Sprint Planning

El trabajo a realizar en este período de tiempo se prevé en la reunión de planificación de sprint, este plan se crea con la colaboración de TODO el equipo. Se entiende que previamente el Product Owner revisó la estrategia y actividades con el negocio.

Esta reunión para un sprint de dos semanas no debería durar más de 4 (cuatro) horas, si en ese período de tiempo no se completó, es síntoma que hay algo que no está bien, ya sea el alcance para el sprint lo la definición previa de los issues a realizar no es clara.

En la reunión de planificación, se define el incremento planeado y cómo el equipo creará este incremento. La salida/entrega/resultado de esta reunión es definir el objetivo del sprint y los compromisos a asumir por el equipo.

La reunión de planificación de un sprint consta generalmente de dos partes, cada una de la mitad de tiempo de duración del total. En la misma nos respondemos las siguientes preguntas:

- ¿Qué va a ser entregado en el incremento resultante del próximo sprint?.

- ¿Cómo se va a realizar el trabajo seleccionado?.

Cabe destacar que el objetivo del sprint puede ser un hito en el objetivo más amplio de la hoja de ruta del proyecto (roadmap).

Aclaración: Una vez acordadas las historias de usuario, no se deberán realizar cambios que afecten el objetivo del Sprint

Reunión diaria: Daily Sprint o Scrum Diario

Es un evento de no más de 15 minutos, cuyo objetivo es que el equipo de desarrollo sincronice actividades, y cree un plan para las próximas 24 horas. Esto se realiza mediante la inspección del trabajo desde la última reunión, y la previsión del trabajo que se puede hacer antes del próximo.

Las reuniones diarias se llevan a cabo en la misma hora y lugar cada día para reducir la complejidad, las mismas pueden ser presenciales o remotas, pero es necesaria la presencia de todo el equipo. El equipo de desarrollo utiliza estas reuniones para evaluar el progreso hacia la meta del sprint y evaluar la tendencia del progreso en finalizar el trabajo en el backlog del sprint.

Cada miembro del equipo debe responder las siguientes preguntas en un timebox de cómo máximo 15 minutos:

- ¿Qué he hecho desde la última reunión de sincronización? ¿Cuál fue el problema?.

- ¿Qué impedimentos tengo o voy a tener para cumplir mis compromisos en esta iteración y en el proyecto?.

- ¿Qué voy a hacer a partir de este momento?.

Cada día, el equipo de desarrollo debe ser capaz de explicar al dueño del producto y al scrum master cómo van a trabajar juntos como un equipo auto-organizado para lograr el objetivo y crear el incremento previsto en el resto del sprint.

Un beneficio adicional es que las reuniones diarias mejoran las comunicaciones, eliminan otras reuniones, identifican y eliminan obstáculos para el desarrollo, destacan y promueven la rápida toma de decisiones, y mejoran el nivel de conocimiento del proyecto del equipo de desarrollo. Esta es una reunión clave de inspección y adaptación.

Reunión de revisión del sprint: Sprint Review

Se lleva a cabo al final del sprint, para inspeccionar el incremento y adaptar, si es necesario, el Backlog del producto. El equipo y las partes interesadas colaboran durante la revisión de lo que se hizo en el sprint.

Basado en ese y cualquier cambio en el Backlog del producto durante el sprint, los asistentes trabajan en las próximas cosas que se podrían hacer. Esta es una reunión informal, y la presentación del incremento está destinada principalmente a obtener retroalimentación y fomentar la colaboración.

La revisión de Sprint incluye los siguientes elementos:

- Los asistentes son el equipo y los interesados clave invitados por el dueño del producto.

- El propietario del producto identifica lo que se ha "hecho" y lo que no se ha "hecho".

- El equipo de desarrollo demuestra el trabajo que se ha "hecho" y responde preguntas sobre el incremento a entregar.

- El propietario del producto analiza el estado actual del Backlog del producto, y estima fechas de finalización basado en el progreso hasta la fecha.

La revisión del Sprint es una porción del producto revisado que muestra funcionalidad y permite obtener feedback y potencialmente generar nuevas historias de usuarios.

Reunión retrospectiva: Sprint Retrospective

Es una oportunidad para el equipo de inspeccionarse a sí mismo y crear un plan de mejoras para ejecutar durante el siguiente sprint.

La retrospectiva tiene 4 (cuatro) objetivos:

- Revisar cómo fue el último sprint en lo que respecta a las personas, relaciones, procesos y herramientas.

- Identificar y ordenar los temas principales que salieron bien y las potenciales mejoras.

- Crear un plan para la implementación de mejoras con respecto a cómo el equipo hace su trabajo.

- Debatir en equipo sobre lo que anduvo bien durante el sprint, qué problemas hubo y cómo se resolvieron.

Refinado de Historias de usuario

El refinamiento de historias de usuarios o User Stories es una sesión clave en el proceso de implementación donde se clarifican ideas y especificaciones para que los miembros de la compañía o integrantes del equipo puedan tener una base consistente sobre el negocio de seguros que queremos implementar, cuales son las responsabilidades y compromisos para entender a estas tareas como terminadas.

Existen 3 (tres) tipos de refinamiento:

1. Aquellos que se realizan durante la incepción de un proyecto o en etapas tempranas para refinar los conceptos iniciales del producto.

2. Los refinamientos esporádicos o refinamientos de la solución (qué).

3. Los que se realizan dentro del sprint cada intervalo regular o lo que llamamos refinamientos periódicos del problema (cómo).

Un error muy común dentro de la práctica es que no se realicen refinamientos, se deja para el último día o se hace dentro del sprint planning (planeamiento del sprint) como parte del mismo.

En general, este último error tiene como consecuencia de que los equipos aleguen no tener tiempo para poder hacer la sesión, se les quite su foco sobre las tareas diarias o no existan historias o problemas concretos a resolver hasta el día anterior a un sprint planning.

Es recomendable también trabajar en un borrador de la definición de listo o terminado (Definition of Ready).

Ésta es el acuerdo entre todas las partes a seguir a la hora de que el Product Owner y el equipo de desarrolladores puedan considerar una historia lista para ser pasada de refinamiento a candidata de un sprint. Nadie quiere, ni es saludable, que aparezca sin previo aviso una historia de usuario un minuto antes de comenzar un sprint planning. Ello bajaría la moral del equipo, destruiría la confianza y traería un número indeseable de efectos negativos en el corto y mediano plazo.

Es responsabilidad absoluta del Product Owner y scrum master poder analizar y refinar historias por adelantado a los sprints, de manera tal que a la hora de comenzar un sprint planning todas las historias se encuentren con la mayor claridad y especificación posible, incluyendo como estas historias se van a probar para darlas por terminadas.

**Como regla general un Product Owner debería estar refinando historias de usuario candidatas a que se encuentren dos o tres sprints por delante del corriente*.*

Definición de terminado

La definición de terminado es algo que aplica a las historias de usuario y refiere al acuerdo de un equipo

de trabajo sobre una lista de criterios que deben cumplirse para la entrega de un incremento de producto. Es así, como el Product Owner junto con el equipo, consensuan cuándo se determina que algo está terminado.
El incumplimiento de estos criterios normalmente implica que ese trabajo no puede ser computado a la velocidad del sprint.

Debajo se citan ejemplos de varios elementos de una definición de listo (DOR) para aplicar a las sesiones de refinamiento, puede haber muchos otros:

- La historia de usuario está bien definida. Todo el equipo entiende lo mismo sobre la historia en cuestión.

- El criterio de aceptación se entiende y se comparte por todos.

- Las dependencias de alto nivel se conocen, así como los riesgos.

- El equipo tiene una idea a alto nivel sobre la experiencia de usuario actual y deseable.

- La historia tiene un tamaño asignado.

- El equipo se siente confortable que realmente lo que se está tratando de resolver es un problema para el cliente.

- Se han pasado por al menos 2 (dos) refinamientos de la historia antes de un sprint planning.

- La historia tiene un valor asignado de puntos de negocio (Cuánto vale la historia para la empresa y por qué).

- El equipo de trabajo entiende cómo se va a probar la historia y que tipo de pruebas se esperan de la misma: stress, funcionales, otras.

Generalmente los desarrolladores de software tienen una reputación de ser un tanto descuidados al responder la pregunta “¿Terminaste con esta tarea?”.

Para ser justos, esta es una pregunta ambigua: puede significar "programación hecha" y esto es generalmente lo que un desarrollador tendrá en cuenta al responder. Sin embargo, el significado suele ser "¿Has terminado la programación, la creación de

datos de prueba, las pruebas unitarias, despliegue en pruebas, documentación...”?

Características requeridas para alcanzar el “Definition of Done”

- El código se encuentra bien escrito, ha sido chequeado y actualizado en el repositorio de versiones.

- El código tiene el nivel de pruebas unitarias establecidas, generalmente en la práctica son de 3 a 5 pruebas.

- El código ha sido desarrollado de a pares o ha sido inspeccionado por un tercero.

- Los documentos y modelos de prueba han sido actualizados.

Beneficios esperados

Los beneficios que esperamos en SysOne sobre la utilización de DOR son:

1. Proveer un checklist que sirva de guía de las actividades previas a realizar la implementación: estimación, diseño, pruebas.

2. Reducir los costos de trabajo. Si el equipo tiene claro que es todo lo que se tiene que tener en cuenta para la implementación de una funcionalidad, raramente el trabajo volverá con errores o partes faltantes.

3. Elaborar un contrato explícito de lo que se debe realizar, con el fin de mitigar el riesgo de malentendidos y conflictos entre el equipo de desarrollo y el Cliente o Product Owner.

Demo / Sprint Review o Die

Al finalizar un sprint, es importante poder preparar una demo de manera tal que se puedan tangibilizar los resultados del esfuerzo del equipo y recibir feedback al respecto. Como el título lo describe, este evento es casi mandatorio para continuar con el avance los siguientes sprints. Por otro lado, es parte fundamental de comprobar la forma de trabajo siguiendo el manifiesto ágil.

"Software funcionando sobre documentación extensiva".

Para poder avanzar correctamente en el proyecto y cumplir efectivamente con la demo es necesario prepararla de manera efectiva.

En los siguientes puntos se presentan temas a tener en cuenta para realizar una buena demo:

1. Hacer foco en los criterios de aceptación que se están cumpliendo y cubriendo con la demo. Se supone que ya existen historias, entonces es necesario demostrar que las mismas han sido cumplidas, total o parcialmente.

2. Comenzar el desarrollo con la demo en mente. El equipo no debería esperar hasta la demo para analizar que mostrar, todo se debería enfocar en la misma, por ejemplo, durante el desarrollo se deberían generar scripts de tests para mostrar que la historia está cubierta y funcionando.

3. Planificar con anticipación. Pensar en un escenario interesante para demostrar que se han satisfecho los criterios básicos de aceptación.

4. Practicar la reunión. Ejecutar la demo pensada al menos una vez con todo el equipo de trabajo.

5. Contar la historia. Centrar la demo desde un rol y usuario realista para resolver un problema concreto de la historia analizada. El objetivo no es solo mostrar que el software funciona sino que a su vez es valioso para el usuario.

6. Mantener el foco de la demo, destacando cuáles son los puntos importantes y cuál es la entrega de valor al negocio.

La demo debe ser la parte más emocionante de esta práctica. Es el momento donde el equipo puede mostrarle a todos los valores que está entregando, por eso vale la pena invertir tiempo en prepararlo para dicha demostración sea lo más valiosa posible. Es vital para poder involucrar a las partes más desinteresadas en el proyecto.

¿Qué medir?

Antes de profundizar en las métricas específicas de proyectos, es importante tener en cuenta y derribar el mito acerca de que en las metodologías ágiles no se planifica ni se mide el avance de los proyectos. Muy por

el contrario, el espíritu que persigue el uso de metodologías ágiles es la continua planificación y revisión del avance de todas las tareas.

Las métricas ágiles se centran en obtener un conjunto comprensible de indicadores para respaldar un objetivo empresarial claro y al mismo tiempo reforzar un ciclo de retroalimentación positiva hacia los comportamientos de las personas y los procesos de la aseguradora. La métrica principal en nuestra metodología es una pieza de software que cumple con las expectativas del cliente.

Como mencionamos previamente, la planificación ágil no se debe interpretar como no tener un plan o métricas, el objetivo de una implementación de sistema central en forma ágil es ofrecer la mejor relación valor / costo. Por lo tanto ¿qué medimos?

La información es la materia prima para la toma de decisiones, y la que puede ser cuantificada proporciona criterios objetivos de gestión y seguimiento.

Desde el nivel concreto de la programación, hasta los más generales de la gestión global de la organización,

tres son los fondos de escala o niveles de zoom con los que se puede medir el trabajo:

- **Gestión de entrega de valor al negocio:** Cuánto valor estamos entregando con las entregas incrementales realizadas. Cuán satisfecho está nuestro cliente con la dirección estratégica que tienen las entregas de valor.

- **Gestión de la solución técnica:** Con el valor entregado al negocio, que calidad poseen nuestras entregas. ¿Cuánta deuda técnica tenemos? ¿La implementación posee los estándares dictados por el mercado y el producto?

- **Gestión de la organización:** ¿Estamos felices con lo que entregamos? ¿Los individuos de los equipos se encuentran motivados para generar más valor?

Algunos principios sobre mediciones

- Medir es costoso. Métricas serias requieren de esfuerzo y tiempo de los equipos sin importar que sean del cliente o de SysOne.

- Comprender por qué necesitamos un indicador, qué objetivo respalda y sus beneficios. Debe quedar claro para las personas qué significa un indicador específico y qué suposiciones se hicieron para elaborarlo.

- Cuestionar su valor antes de implementar una nueva herramienta de medición, así como también definir quién o quiénes serán los responsables de desarrollar la métrica, y la forma en la cual se aplicará.

- Garantizar que todas las mediciones persigan un fin. Medir no es un fin en sí mismo.

- Apoyar un proceso de mejora continua para ayudar a adaptar las prácticas e ir evolucionando en cada iteración.

- Emplear aplicaciones como Jira para generar métricas con información precisa sobre lo que sucede en el proyecto. Como la información actualizada es clave para una correcta gestión, es importante que todos los participantes del proyecto sean responsables por actualizar sus tareas y valores durante el sprint y diariamente.

Métricas de productividad

Las métricas más importantes que SysOne persigue dentro de su metodología son las siguientes:

- Velocidad.
- Tiempo.
- Trabajo por hacer.

Dónde: Velocidad = Cantidad de puntos por Sprint

En Scrum la métrica por excelencia es la velocidad, la cual se evidencia posterior al Sprint y hace referencia a la cantidad de puntos que se han realizado en cada uno de los Sprint. Es una métrica alineada con el Manifiesto Ágil en el sentido de que la mejor medida de avance es el software funcionado, y esta se mide una vez entregado el incremento.

La velocidad no puede predecir el futuro, sólo dice lo que ya ha sucedido y ofrece una sugerencia de lo que se puede esperar en el futuro. Desde el punto de vista de la naturaleza humana es una métrica que crea un impulso sano en poner el esfuerzo en la mejora de la velocidad. Pero también aquí hay que tener cuidado, la velocidad es propia de cada equipo y específica para ese producto/entorno/clientes concretos, nunca

debemos de comparar velocidades entre equipos ni hacer benchmarking con la misma.

Por lo que ahí cabe destacar que la velocidad no es lo mismo que capacidad, ya que esta última se mide en horas y es previa al Sprint, donde se realiza foco en estimar la cantidad de puntos que se podrán atender en cada uno de los Sprint.

Funcionales

- Velocidad planificada.
- Velocidad actual.
- Número de historias de usuario planificadas.
- Número de historias de usuario aceptadas (que cumplen el Definition of Done).
- Porcentaje de historias de usuario aceptadas (que cumplen el Definition of Done).

Calidad

- Porcentaje de cobertura de test unitarios.
- Número de bugs.
- Número de casos de prueba.
- Número de casos de prueba automatizados.
- Total de pruebas.
- Total de pruebas automatizadas.
- Número de refactorizaciones.

Métricas financieras

Las siguientes métricas son importantes para poder controlar el proyecto en cuanto al Retorno de la Inversión (ROI) que proporciona el proyecto a lo largo de la implementación:

1. Retorno de la inversión pendiente. El valor pendiente de entregar respecto al costo, cuánto presupuesto tenemos para finalizar el proyecto.

2. Presupuesto disponible vs. presupuesto insumido.

3. Desviación financiera vs. la planificación inicial.

Análisis Técnico

Por cada sprint planning, es necesario realizar el análisis técnico de las tareas a realizar, en este caso surgen generalmente todas las incógnitas de diseño de cómo vamos a resolver un determinado desafío siguiendo los lineamientos del producto y la plataforma.

Por tal motivo es inminente antes de realizar un desarrollo o nuevo servicio preguntarse lo siguiente:

- ¿Ya existe un servicio o funcionalidad de la plataforma que me pueda resolver el problema que estoy enfrentando?

- Si no existe, ¿hay alguien de mi equipo que posee un problema similar?

- ¿El equipo de ingeniería de producto me puede proveer la solución? ¿Puedo trabajar con ellos en la búsqueda de la misma?

Generalmente hay muchos nuevos desarrollos relacionados con interfaces e intercambio de datos que tenemos que llevar adelante con el resto del ecosistema donde se implanta la plataforma. Por eso es de vital importancia en la etapa de User Story Map, poder realizar el catálogo de interfaces.

Catálogo de interfaces

Es claro que a la hora de implementar un core o un nuevo negocio de seguros, el mismo no puede desenvolverse aislado del ecosistema de la aseguradora. Por eso, a la hora de integrar el core con el resto de las plataformas, es importante recopilar todas las interfaces a desarrollar y probarlas para la implementación.

Para la correcta elaboración y conocimiento del catálogo de interfaces es necesario recopilar los siguientes datos que muestra la tabla:

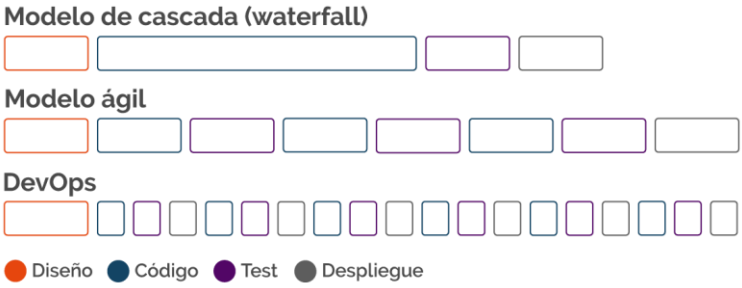
Elemento	Descripción
Interfaz con otra plataforma	Cada interfaz que intervenga en el proceso de seguros deberá ser explicitada, la misma deberá interactuar de forma no intrusiva dentro del core, de manera tal que la misma se pueda envolver dentro de un servicio del ISB.
Complejidad	Estimar la complejidad de la interfaz respecto al esfuerzo para dejarla funcionando dentro de la plataforma. La estimación deberá ser realizada en puntos de historia de usuarios.
Responsable	Obtener por cada interfaz una persona responsable de la misma, a la cual podamos acudir en caso de malfuncionamiento o necesidad de asistencia. Identificar correos electrónicos y teléfonos de cada responsable.
Tipos de mensaje	Se deberá especificar que tipo de mensajes recibe cada interfaz, siempre es necesario generar uno o más ejemplos de mensajes de solicitud y respuesta (input/output). Es necesario para cada tipo de mensaje poseer una especificación clara del formato de intercambio, por ejemplo: XML, JSON, texto u otros.
Validación de mensajes	Para cada tipo de mensaje del punto anterior se deberán poseer las especificaciones de las validaciones a realizar. Es importante entender que quizás lo que a nuestra vista es obvio en otros sistemas no lo es y corresponde validarlo.
Transformaciones	En este caso se deberá estimar y especificar el esfuerzo de transformar un mensaje previamente mencionado al modelo de datos de Sysone, por ejemplo si tomamos una cobranza de un sistema mediante una interfaz, cuál es el esfuerzo en convertir dicho mensaje en un objeto BillingStatement.

Transporte	Se deberá identificar el protocolo de transporte para cada interfaz, por ejemplo HTTP, REST, WSDL, FTP, MQ, etc.
Auditoría	Para cada mensaje enviado y recibido desde la interfaz es necesario especificar el nivel de auditoría y control de los mismos. Por ejemplo, cada vez que recibimos un cliente crear una entrada en nuestro modelo AuditEvent con un AuditEventType de tipo “NEW CUSTOMER”.
Endpoints	Establecer cuáles serán las locaciones donde estarán disponibles las interfaces. Por ejemplo https://int.sysone.com/customers o ftp://www.cliente.com
Seguridad	Detallar para cada interfaz cuales son los mecanismos de seguridad aceptados por la misma, por ejemplo SSL, autenticación previa con usuario y password, LDAP u otros.
Tratamiento de errores funcionales	Para este ítem, debemos reconocer para cada interfaz, los tipos de errores que la misma arroja y generar las analogías correspondientes en el sistema central. Nunca se deberá dejar un error sin tratamiento, incluyendo time outs o no respuesta de un mensaje.
Factores de rendimiento	Como nuestra plataforma y su desempeño dependen en este caso del rendimiento y performance de la interfaz, se deberá establecer la tolerancia y cotas mínimas de rendimiento de cada una de ellas. Por ejemplo, no podemos tardar más de 3 segundos en recuperar los datos de una persona de un sistema de CRM.

Integración y Delivery continuo

La Integración y Delivery continuo evitan que los equipos de desarrollo trabajen de forma aislada. Con esta técnica se le da más visibilidad al proceso de desarrollo en general, a todos los pasos requeridos desde que se empieza a programar una historia de usuario hasta que está en producción.

Así, todo el equipo sabe las fases por las que va pasando el código, y el estado del software en cada momento (si compila, si pasa las pruebas, en qué entorno está cada versión, qué versión se está probando, etc.), dando más visibilidad a la estrategia de gestión de configuración, política de ramas del control de versiones, entre otras cosas.



Aclaración: Cada metodología propuesta en el gráfico no es excluyente*. Por ejemplo, el modelo ágil puede incluir DevOps.*

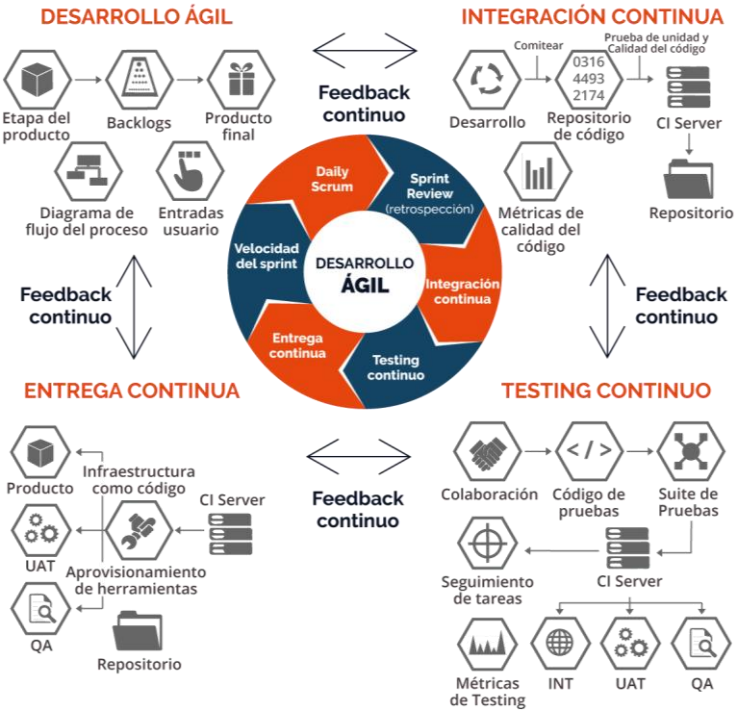
En cuanto a la calidad, se destacan 2 (dos) mejoras sumamente relevantes para los proyectos:

- **Incremento en la calidad de producto:** Ya que su principal objetivo es detectar los errores lo más pronto posible, en fases tempranas del desarrollo, para poder solucionarlos rápidamente. Así se introducen varios tipos de pruebas y comprobaciones, minimizando los riesgos, y haciendo que el software tenga menos bugs. También se lanzan inspecciones continuas de código, análisis periódicos para detectar problemas de calidad en él. Los desarrolladores siempre deben velar por suplir esas deficiencias para que todas las versiones del código cumplan con los estándares de calidad previamente definidos.

- **Incremento en la calidad de las personas:** El equipo se vuelve más auto-organizado y sus integrantes aprenden a hacer distintos tipos de pruebas (unitarias, de integración), adoptan

mejores prácticas de programación y en general a desarrollar código de mayor calidad.
Este escenario le da más confianza al equipo. Así es capaz de afrontar nuevos retos, mejoras y cambios y está más motivado. También se maximizan los tiempos a través del empleo de automatizaciones de procesos repetitivos, como por ejemplo ciertas pruebas de regresión.

Estas 2 (dos) mejoras en la calidad, propician un aumento en la satisfacción del cliente, y una mejor redistribución e inversión de recursos económicos de la compañía a través del perfeccionamiento de procesamientos.



Integración continua

Fowler define la integración continua como: *"Una práctica de desarrollo software donde los miembros del equipo integran su trabajo frecuentemente, al menos una vez al día. Cada integración se verifica con un build automático (que incluye la ejecución de pruebas) para detectar errores de integración tan pronto como sea posible."*

Muchas veces, se tiende a pensar que la integración continua es tener instalado el servidor de integración continua (por ejemplo, Jenkins) y que este compile el código periódicamente; o tener automatizados los despliegues dándole a un botón desde dicho servidor.

La integración continua engloba mucho más que esto: es una serie de buenas prácticas, de comprobaciones interconectadas entre sí para conseguir software de mejor calidad.

¿Cuáles son esas buenas prácticas?

1. **Práctica de desarrollo software:** La integración continua afecta al proceso de desarrollo del software, incorporando nuevas prácticas o interconectando las que ya había. Para implementar una integración continua solemos definir un “pipeline”, un conjunto de etapas o fases por las que va pasando el software y se va automatizando.

   Un ejemplo de un pipeline podría ser que con cada subida de código al repositorio de control de versiones este se descargue y compile.

Si está todo correcto, que se ejecuten una serie de pruebas unitarias, o se despliegue el código a otro entorno para hacer pruebas de sistema. Por último, que se despliegue al entorno de QA, de pruebas, para realizar otro tipo de pruebas manuales. (Recuerda ¿Pruebas de integración, funcionales, de carga...?)

Esta es una de las primeras cosas que hay que definir, saber cómo es el desarrollo, cuál es el criterio para que el código evolucione de un entorno a otro, y qué se hace en cada entorno. Si el código no pasa algún punto hay que establecer cuál es la estrategia para resolver el error, quién se encarga de ello, cómo se gestiona.

2. **Práctica de integración frecuente del trabajo de todos los miembros del equipo:** El código realizado por cada uno de los colaboradores debe estar integrado dentro de un ecosistema en común para todos, a lo que denominaremos “Control de Versiones”.

   Cuando cada miembro del equipo comienza a trabajar, normalmente se consumen otros componentes del software que todavía no

están implementados o que está programando otra persona. Y hasta que no juntamos todo ese código, no nos damos cuenta de los errores. Antes, lo que se tendía a hacer es que cada desarrollador trabajara de forma independiente y luego al final se realizaba la integración de todo el código.

Esto se traduce en integraciones difíciles que tardan mucho en completarse, ya que hay muchos cambios, mucho código que integrar. Uno de los motivos por los que surge la integración continua es para evitar esto. La idea es que, en vez de dejar la integración para el final, se vayan haciendo pequeñas integraciones de código frecuentemente.

La frase que podríamos aplicar aquí es que, si algo te cuesta, debes hacerlo más a menudo y poco a poco, para que con el tiempo te vaya costando cada vez menos esfuerzo.

Prueba continua

Para poder regresionar pruebas y proveer de seguridad a nuestra plataforma, es esencial poder automatizar la mayor cantidad de pruebas e

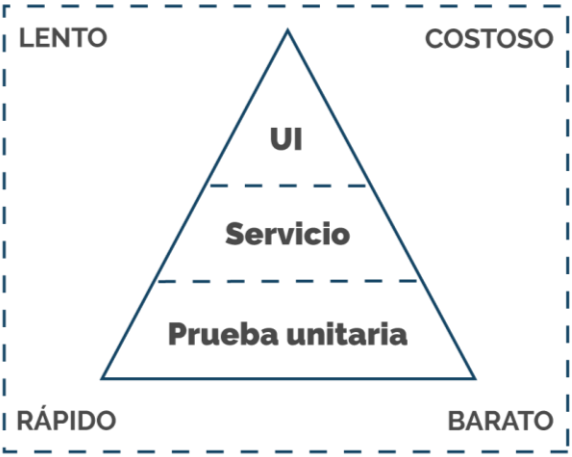
integrarlas a nuestra máquina de integración continua. Para poder lograr esto no sólo se debe codificar pruebas automáticas sino poder pensar en las mismas antes de realizar cualquier módulo o desarrollo. Esto último tiene más relación con la cultura de cómo desarrollamos que con la prueba en sí.

Un conjunto de pruebas unitarias es esencial para verificar el correcto funcionamiento de cualquier proyecto de seguros. Pero, ¿cómo podemos asegurarnos de que estamos escribiendo buenas pruebas que les den valor al cliente y al equipo, tanto a corto como a largo plazo?

Pruebas unitarias automatizadas

Una prueba de unidad es simplemente eso; una prueba que cubre la pieza más pequeña de nuestra base de código, una 'unidad' si lo desea, aislada de otras partes del sistema. *Por ejemplo, en nuestra plataforma (Java), esto normalmente se refiere a probar métodos individuales. Las pruebas unitarias son distintas de las pruebas de integración, que tienen como objetivo probar cómo varias piezas del sistema se integran y se integran con partes de la arquitectura, como las bases de datos. También son diferentes a las pruebas funcionales, que prueban porciones de funcionalidad sin tener en cuenta cómo funciona el sistema técnicamente.*

En el siguiente gráfico piramidal, las pruebas de unidad situadas en la parte inferior deben tener muchas más pruebas de unidad que cualquier otro tipo de prueba.



El principal beneficio de las pruebas unitarias es la posibilidad de verificar que lo que hace una parte del código que ha escrito, sea lo que realmente se necesita y se planificó, sin tener que acceder previamente a una interfaz específica. Las pruebas unitarias también dan la confianza de que el código puede hacer frente a las entradas negativas o inesperadas, y tratarlas adecuadamente.

Las pruebas unitarias deben proporcionar una red de seguridad para todo el equipo de desarrollo. Si una

base de código está comprensivamente cubierta por pruebas, los desarrolladores pueden refactorizar, modificar y extender cualquier código, con la seguridad de que el conjunto de pruebas les mostrará si la funcionalidad subyacente ha cambiado de una manera que no fue intencionada.

Despliegue continuo

El despliegue continuo es la práctica mediante la cual SysOne utilizando herramientas de integración continua (ver Jenkins), intenta en todo momento del proyecto realizar entregas de software funcionando en un ambiente de pruebas, dando el poder al Product Owner de determinar en qué momento se pasa la funcionalidad del ambiente de pruebas a producción.

El despliegue continuo requiere de máxima responsabilidad de los participantes del proyecto, dado que todo lo que se sube al repositorio de versiones se despliega en forma automática en el ambiente de pruebas.

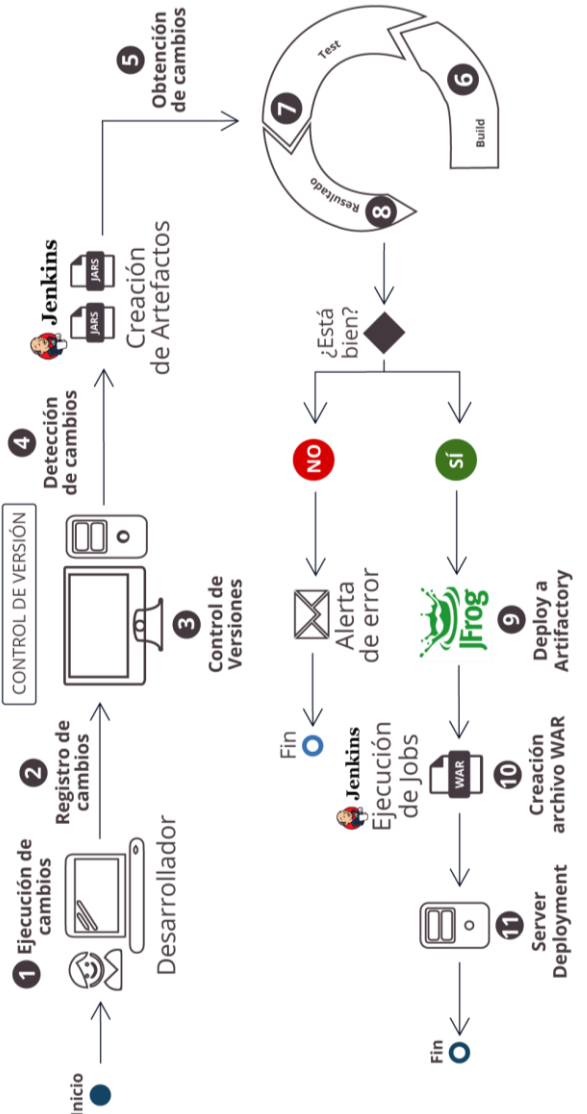
Las ventajas de esta técnica son:

1. Perfeccionar desde el inicio del proyecto el pasaje a producción, dado que esta práctica se está realizando todo el tiempo y preferentemente de forma automática. De esta manera también se minimizan los riesgos.

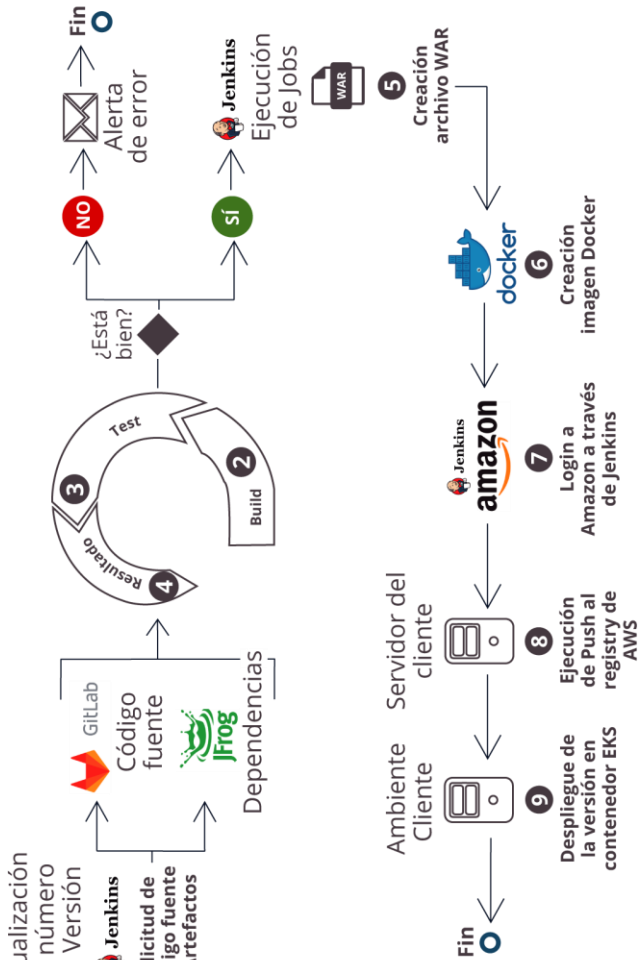
2. Proyectar en los ambientes de forma casi inmediata los cambios realizado por el equipo de desarrollo. De esta manera el usuario puede detectar tempranamente la aparición de nueva funcionalidad y proporcionar su respectivo feedback.

3. Contar en todo momento con una versión de la plataforma funcionando.

Workflow proceso diario: Pipelines



Workflow despliegue al cliente



Glosario

A

- **Agilidad:** Metodología de trabajo que permite adaptar la forma de trabajo a las condiciones del proyecto, consiguiendo flexibilidad e inmediatez en la respuesta para amoldar el proyecto y su desarrollo a las circunstancias específicas del entorno.

- **Artefactos:** Son los elementos que se producen como resultado de la aplicación de Scrum. Los tres principales artefactos o herramientas son: el Product Backlog, Sprint Backlog, el incremento de la estrategia o entregable y las historias de usuario/épicas.

B

- **Backlog:** Lista de Historias de Usuario ordenadas según el valor de negocio que establece el cliente, y que trata de cubrir todas las funcionalidades necesarias. El product backlog se puede ver desde la perspectiva de una iteración o sprint, de una release o de todo el producto.

E

- **Épica:** Epic, historia de usuario que por su gran tamaño, el equipo descompone en historias con un tamaño más adecuado para ser gestionada con los principios y técnicas ágiles.

F

- **Framework:** Marco de trabajo que engloba un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas.

H

- **Historia de usuario:** User story, descripción de una funcionalidad que debe incorporar al sistema, y cuya implementación aporta valor al cliente.

I

- **Iteración:** Iteración significa repetir varias veces un proceso con la intención de alcanzar una meta deseada, objetivo o resultado.

M

- **Mapa de historias de usuario:** User Story Map, framework de trabajo ágil que permite generar una representación visual del proyecto en su totalidad.

- **Mapa de Impacto:** Impact Map, técnica de planificación que permite entre otras cosas poder generar una versión inicial de un Product Backlog.

- **MMP:** Minimum Marketable Product, conjunto de características mínimas que debe tener un MVP.

- **MVP:** Minimum Viable Product, producto mínimo viable es la versión mínima de un producto, que permite recolectar el mayor volumen de información para el proyecto con el menor esfuerzo posible.

P

- **Puntos de Historias de usuario:** Story points, unidad de medida para expresar un estimado del esfuerzo requerido para implementar una porción de trabajo.

R

- **Release:** Proceso de entregas de software nuevo o de actualizaciones del software.

- **ROI:** Return On Investment, retorno de la inversión.

S

- **Scrum:** Marco de trabajo en el que se aplica de manera regular un conjunto de buenas prácticas para trabajar colaborativamente, en equipo, y obtener el mejor resultado posible de un proyecto. El trabajo se divide en Sprint, donde al finalizar cada uno de ellos se valida el trabajo realizado. En función de esto, se priorizan y planifican las actividades en las que invertiremos nuestros recursos en el siguiente Sprint.

- **Sprint:** Intervalo prefijado durante el cual se crea un incremento del producto, potencialmente entregable.

- **Stakeholders:** Personas que tienen un interés, directo o indirecto, en el trabajo del Equipo.

W

- **Waterfall:** Metodología de trabajo también conocida como modelo de desarrollo en cascada. Consiste en el desarrollo de un proyecto de manera secuencial, donde se redacta entre el proveedor y cliente una lista de requisitos que el producto final debería tener, presentando serios inconvenientes en aspectos como la adaptabilidad y flexibilidad.

- **Workflow:** Flujo de trabajo, donde se automatiza y estandariza los procesos de negocio.

SYSONE

Confidencialidad

La información presente en este libro es de carácter confidencial, siendo todos los derechos y su propiedad intelectual perteneciente a SysOne S.A. Sus receptores no obtendrán derecho alguno, de ningún tipo, sobre la información ni sobre el uso de su contenido. Su divulgación, ya sea parcial o total, e independientemente de su formato y canal, deberá contar con explícita autorización de SysOne S.A.

Para más información, póngase en contacto con nosotros escaneando el siguiente código QR.

www.sysone.com

"Discovery Path", es un libro de autoría exclusiva de SysOne Latam que combina nuestro know how en el mercado asegurador, experiencias en el desarrollo e implementación de software de alta complejidad y, la puesta en marcha de metodologías ágiles, donde el mayor desafío es la capacidad de adaptación al cambio.

¿Está interesado en cómo adaptarse a estos grandes cambios y emprender el mayor desafío dentro de su compañía?

¡Este libro está aquí para ayudarlo!

ISBN 978-987-47225-0-8