Gestión del backlog

Playbook para desarrolladores de software

Transformación desarrollo software - #ONE

Control de versiones

Versión	Fecha de Creación	Responsable	Descripción
1.0	{27-4-2023}	Global Software Development - GSD	Primera versión oficial de los playbooks

Índice

1. Introducción

1.1. Acerca de este playbook

Este playbook es un componente fundamental para la práctica de desarrollo de software en el Banco, cubre todos los principios y directrices necesarios para una adecuada gestión del backlog de las piezas de software a nivel global.

Define los principios, las prácticas, las herramientas y los indicadores que todos los equipos de desarrollo de software[1] del Banco deben adoptar para la construcción y gestión del Product Backlog.
Como equipos ágiles, la gestión del backlog es un proceso continuo que ayuda a los equipos a mantenerse centrados y alineados en el trabajo que hay que hacer para asegurar el éxito en el desarrollo de productos digitales. Es una parte importante de la gestión de proyectos, que hace hincapié en la flexibilidad y mejora continua.

El playbook debe ser actualizado regularmente basándose en las prácticas y el panorama tecnológico del Banco. El equipo responsable del documento será el responsable de actualizarlo.

Figura 1: Ciclo de vida del desarrollo de software

[1] Queda fuera del alcance de este playbook, todos aquellos equipos/proyectos que no trabajan realizando desarrollo de software

1.2. Principios básicos

Este playbook se guía por un conjunto de principios que todos los desarrolladores debemos seguir:

- **Generar Backlog:** El backlog es la única fuente de la verdad para cada equipo independientemente de la tecnología. Todo lo planificado y no planificado debe estar incluido en él.
- **Globalidad:** Existirá una taxonomía y un proceso global de trabajo para todos los servicios en el Banco independientemente de la tecnología (ej., mainframe, APP) y la geografía, y una herramienta global siempre que sea posible.
  La migración a la nueva taxonomía global y herramientas se hará de forma manual.
- **Documentar:** Todos los informes, métricas, documentación y cuadros de mando son parte integrante de las herramientas de gestión del backlog disponibles.
- **Estandarización:** Usar una taxonomía global y formas de trabajo alineadas entre los equipos y geografías permitirá a los equipos colaborar mejor, compartir mejor el conocimiento y rotar entre los equipos.
- **Simplicidad:** Los sistemas de software rara vez se escriben una vez y no se modifican. Más bien, muchas personas trabajan con el mismo sistema a lo largo del tiempo. Por eso es importante buscar las soluciones más sencillas que puedan comprenderse fácilmente y evolucionar en el futuro.
- **Automatización:** Los procesos manuales que son repetitivos deben automatizarse para mejorar nuestra productividad y enfocarnos en el desarrollo.

1.3. Niveles de exigencia

Este playbook utiliza intencionadamente las siguientes tres palabras para indicar los niveles de exigencia según la norma **RFC2119**:

- **Debe** - Significa que la práctica es un requisito absoluto.
- **Debería** - Significa que pueden existir razones válidas en circunstancias particulares para ignorar una práctica, pero deben comprenderse todas las implicaciones y sopesar cuidadosamente antes de elegir un camino diferente.
- **Podría** - Significa que una práctica es realmente opcional.

2. Prácticas

Los equipos del Banco para el desarrollo de software **deben** seguir las **tres prácticas** siguientes relativas al análisis, construcción y gestión del Backlog, independientemente de la tecnología y el lenguaje de programación utilizados:

1. Gestionar el backlog a nivel portfolio.
2. Gestionar el backlog a nivel equipo.
3. Gestionar la documentación de los proyectos.

A continuación se definen cada una de las prácticas siguiendo la siguiente **estructura**:

- Sus beneficios (que conseguimos).
- Precondiciones (condiciones para poder aplicar la práctica).
- Adopción (cómo implementarla).
- Herramientas necesarias.
- Indicadores para los equipos (como la medimos).
- Enlaces de interés.

Para ayudar a la lectura de este capítulo se han identificado los siguientes puntos:

1. Se adjunta la información de los principales actores que participan en la gestión del backlog y se realiza una revisión de sus **principales funciones** (Tipos de roles).

2. Dada la **inclusión del Servicio**[2] como eje para los desarrollos, se identifica necesario relacionarlo con el trabajo de los equipos, permitiendo de este modo tener identificados todas las evoluciones realizadas sobre los mismos.

3. Se identifica el siguiente gráfico como un resumen de la relación entre los distintos artefactos definidos en el modelo de trabajo. Todos los elementos de la gestión del backlog **aplican tanto a los nuevos desarrollos o evoluciones como a las actividades de mantenimiento y operación.**

[2] Los Servicios son canales o dominios de negocio que son consumidos por el cliente u otros servicios. Ejemplo: retail web channel, banking app, ATM & self service. Los servicios están registrados en la Herramienta Global Nucleus perteneciente a los equipos de Operaciones.

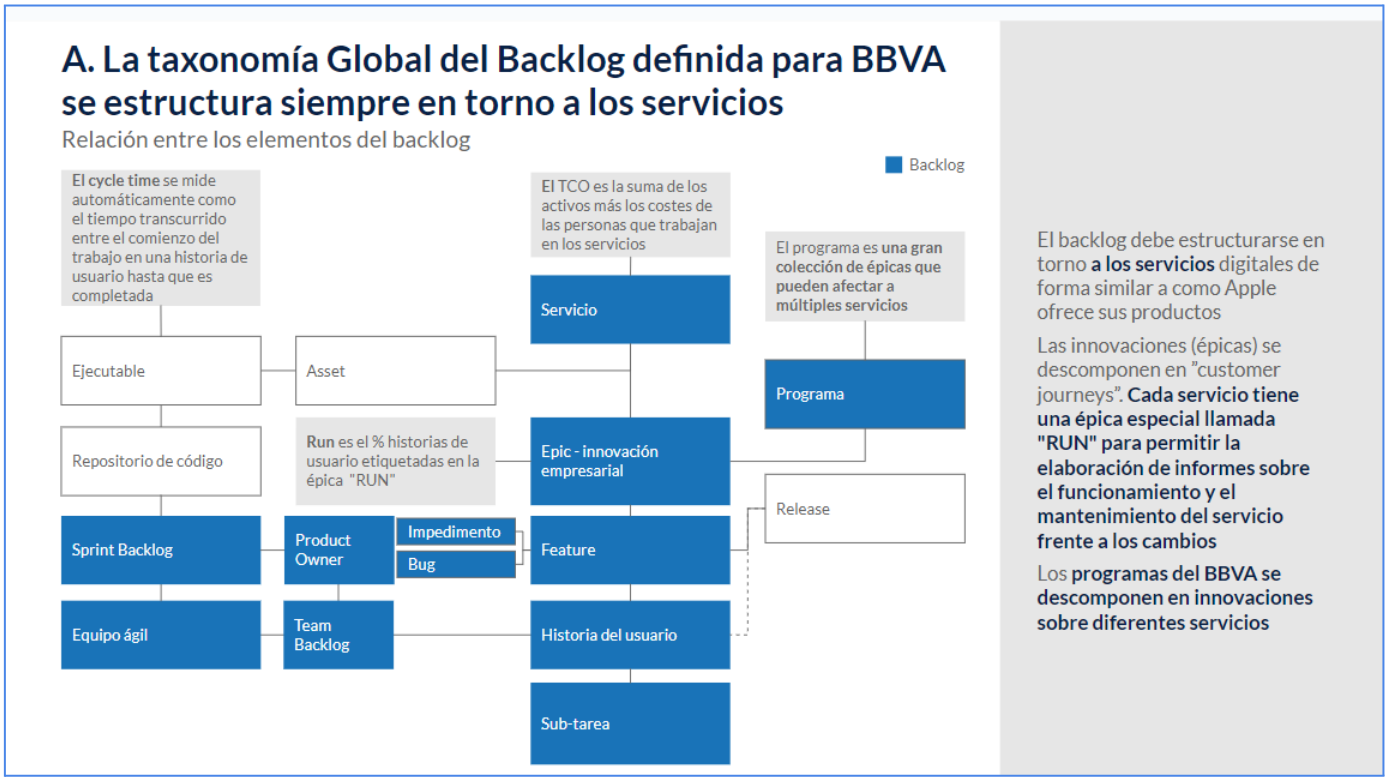


Figura 2: Ejemplo ilustrativo de las relaciones entre los elementos del backlog

2.1. Gestionar el backlog a nivel Portfolio.

Gestionar el backlog a nivel de portfolio[3] debe ser una parte importante de la gestión de proyectos y debe aportar flexibilidad para centrarse en la mejora continua.
Dicha gestión permitirá posteriormente a los proyectos tener visibilidad del trabajo que se tiene planificado a largo plazo, asegurando el alineamiento posterior con el trabajo que realizan los equipos de desarrollo.

Beneficios

Los beneficios que aporta una adecuada gestión del backlog a nivel portfolio son:

- Visibilidad del trabajo a largo plazo.
- Proceso continuo que ayuda a los equipos a trabajar sobre las prioridades definidas.
- Mantiene a los equipos alineados en el trabajo.
- Priorización eficiente a la hora de trabajar.
- Gestión de dependencias.
- Alineamiento con los objetivos estratégicos de negocio.

[3] Portfolio refiere a una organización de alto nivel estándar, si bien según la geografía podría denominarse de otros modos como: Fábrica, Valor de Negocio, Propuesta de valor, Value Stream Management (VPM)…

Precondiciones

Para poder aplicar la práctica se deben dar las siguientes **precondiciones**:

- Disponer de una herramienta global de gestión de backlog (Jira, excepto en países donde no se pueda aplicar).

Adopción

2.1.1. Gestionar el backlog a nivel Portfolio.

Para la correcta gestión de los proyectos, es importante tener identificados las evoluciones o nuevas funcionalidades que se desean realizar a largo plazo (Épicas) y que posteriormente serán desglosadas en funcionalidades más pequeñas y que veremos posteriormente (Features).

La **Épica** es uno de los principales artefactos a gestionar a nivel portfolio y **debe** ser el Program Manager (PM), que puede ser de ingeniería para features que tengan como objetivo la entrega de valor al servicio, y/o el Service Owner (SO) quien las debe definir siguiendo las siguientes directrices:

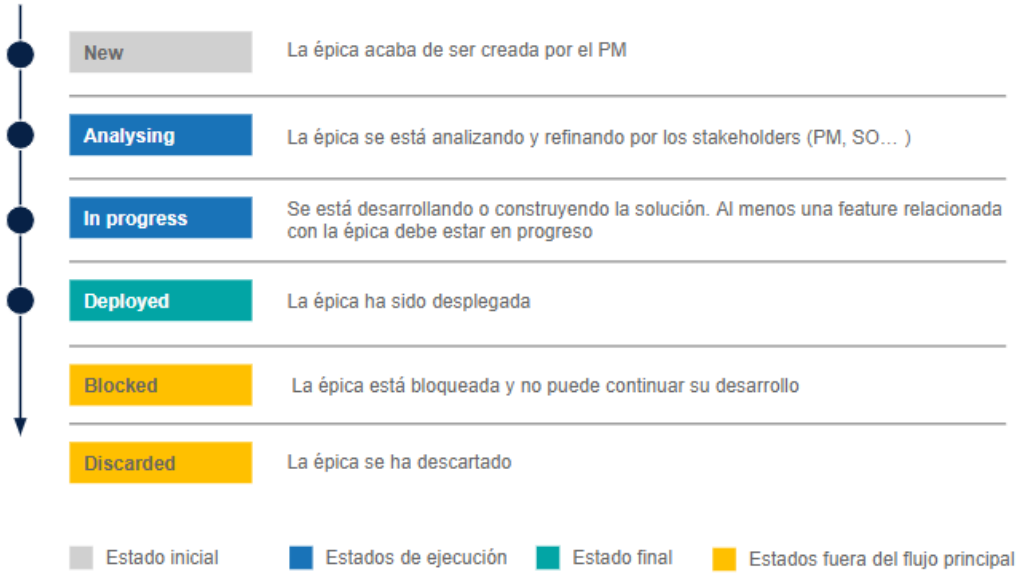
- Las Épicas **deberían** conllevar una nueva evolución funcional que se llevará a cabo con un **horizonte temporal** estimativo, que comprenda un máximo de 3 Qs.

- Atendiendo a aspectos relacionados con la **trazabilidad**:
  - La Épica **debe** componer de una agrupación de más de una Feature, es decir, la relación es 1 -> n.
  - Las Épicas **deben** agruparse en programas para todos los proyectos que entran dentro del ámbito de la SDA.
  - Las Épicas **deben** impactar a uno o varios servicios.

- En función de la naturaleza de la Épica ésta **debe** dividirse en dos **categorías** diferentes:
  - Épica “CHANGE”: los equipos **deben** definir como Épica Change aquellas derivadas de programas priorizados por la SDA que introducen nuevas funcionalidades o innovaciones en el servicio.
  - Épica "RUN": los equipos **deben** definir como Épica Run aquellas que se centran en el apoyo al servicio (por ejemplo, mantenimiento, operaciones, seguridad, obsolescencia, fiabilidad), con independencia de que por temas de gestión pudiera existir un código de la SDA para dicho apoyo.
    Ejemplo: las partidas anuales de la SDA para el FIE (caso de España).

- En el caso de proyectos normativos regulatorios se deja a criterio de los equipos determinar si se deben dar de alta como épica CHANGE o RUN en función de si entran priorizados dentro de la SDA o si se centran en apoyo al servicio.

- Es el Program Manager (PM) y/o Service Owner (SO) en función de si la épica es de Desarrollo o Mantenimiento quienes **deben** tener en cuenta que el **flujo de una Épica** es el mostrado a continuación y **debe** pasar por los estados indicados para poder desplegarla:



- Teniendo en cuenta si son obligatorios o no, para crear una Épica **debemos** tener en cuenta los siguientes **campos**:

Campo	Descripción	Obligatorio	Opcional
Summary	Resumen de la funcionalidad	✓	
Description	Descripción en detalle de la funcionalidad	✓	
Epic Program	Agrupador de épicas		✓
DoR	Puntos que se deben cumplir para que la épica sea considerada lista para que pueda comenzar a desarrollarse		✓
DoD	Puntos que se deben cumplir la épica para que pueda considerarse como terminada acorde con los requisitos establecidos		✓
Acceptance Criteria	Requisitos de negocio que cada épica debe cumplir para considerarse como completada		✓
Priority	Prioridad asignada		✓
Labels	Agregador de épicas		✓
Attachment	Conjunto de archivos que se pueden adjuntar para definir la épica		✓

✓ Obligatorio cuando se transiciona de New a Analysing

Una parte importante de la gestión del portfolio es la **coordinación y seguimiento** del trabajo de los equipos, por ello se establece que:

- **Deben** ser los Program Manager y los Service Owner los encargados de gestionar y priorizar sus correspondientes épicas.

- **Deberá** existir una estrecha relación entre los Program Manager y los Service Owner, dado que ambos tienen una visión portfolio de los desarrollos y de sus prioridades. En ocasiones el Service Owner **deberá** hablar con los Program Manager para solicitar una mayor prioridad dentro de los equipos de desarrollo en temas relacionados con pequeños evolutivos y/o deuda técnica, que afectan a sus correspondientes desarrollos.

- Los responsables del Portfolio del Banco **deben** definirse las siguientes vistas, que nos dan agrupaciones del trabajo que se está desarrollando para un servicio (Vista Servicio), lo que se está desarrollando para un proyecto (Vista de Proyecto) o para un Programa (Vista de Programa). Es importante destacar que son simplemente vistas de backlog y que dentro de los espacios de trabajo de la herramienta global de gestión de backlog tendremos únicamente el Team backlog, sobre el cual trabajará el equipo de desarrollo. En estas vistas podemos ver:

    - **Vista de Programa:** Es una lista priorizada de trabajo derivada de los team backlogs que conecta el Programa y su Program Manager.
        Ejemplo: por Programa, relación de épicas (especificando estado y Change/Run) y por cada una de ellas, toda la info relevante: Features asociadas, servicio al que pertenece, SDA Tool, estado, etc. Está compuesto por los siguientes campos:
        - Epic Program
        - Epic (Summary y enlace directo)
        - Estado de la epic
        - Run o Change
        - Feature (relacionada con la Epica)
        - Team (Backlog que desarrolla la Feature)
        - Servicio (asociado a la Feature y por tanto a la épica)
        - SDA (asociado a la Feature)
        - Status (asociado a la Feature)

    - **Vista de Servicio:** Es una lista priorizada de trabajo derivada de los team backlogs que conecta al Servicio y su Service Owner.
        Ejemplo: por servicio, qué feature se están trabajando y equipos (con todos los datos asociados) y a que Epicas/Programas están ligadas esas Features. Podrían tener los siguientes campos:
        - Servicio, Feature (Summary y enlace directo)
        - Epic (Summary y enlace directo)
        - Change / Run

- Epic Program
- SDA Tool
- Team Backlog
- Status
- Commitment Type
- PI
- Type of Delivery

- **Vista de Proyecto:** Es una lista de elementos compuesta por diferentes backlogs de equipo y gestionada por diferentes servicios.

  Ejemplo: Por equipo, mostrar las Feature en las que se está trabajando por PI y tipo de compromiso, añadiendo todos los atributos relevantes asociados a la Feature:

  - Team Backlog Name
  - Feature (summary)
  - Status
  - Epic
  - Change/Run
  - Epic Program
  - SDA Tool
  - Commitment Type
  - PI
  - Type of Delivery

Herramientas necesarias

La adopción de esta práctica por parte de los equipo del Banco requiere el uso de la siguiente herramienta:

- **Jira:** Servicio global para la gestión de proyectos.

Indicadores para los equipos

Los indicadores que apliquen, **deben** ser seguidos por los equipos, para conocer el grado de madurez respecto a la práctica:

Indicador	Descripción
Nº Épicas RUN	Nº de Épicas RUN creadas
Nº Épicas CHANGE	Nº de Épicas CHANGE creadas
Nº Épicas RUN descartadas	Nº de Épicas RUN descartadas
Nº Épicas CHANGE descartadas	Nº de Épicas CHANGE descartadas

Nº Epicas RUN entregadas	Nº épicas RUN entregadas
Nº Epicas CHANGE entregadas	Nº épicas CHANGE entregadas
Tiempo de entrega	Tiempo de entrega de la Épicas

Nota: Dichos indicadores son mediciones que deberán utilizar los equipos de forma interna para conocer mejor cómo están trabajando y poder mejorar de forma objetiva.

2.2. Gestionar el backlog a nivel equipo.

Todos los equipos de desarrollo del Banco **deben** garantizar que la gestión del backlog a nivel equipo sea un proceso continuo que ayude a los equipos a mantenerse centrados y alineados en el trabajo que hay que hacer para asegurar el éxito en el desarrollo de productos digitales.
Para poder cumplir con este cometido es esencial aplicar las directrices recogidas en esta sección.

Beneficios

Los beneficios que aporta una adecuada gestión del backlog por parte de los equipos ágiles son:

- Proceso continuo que ayuda a los equipos a mantenerse centrados.
- Mantiene a los equipos alineados en el trabajo.
- Asegura el éxito en el desarrollo de equipos digitales porque permite hacer hincapié en la flexibilidad y mejora continua.
- Definir los mismos elementos del backlog a nivel global mejorará la forma en que diseñamos y desarrollamos servicios digitales a nivel global.

Precondiciones

Para poder aplicar la práctica se deben dar las siguientes **precondiciones**:

- Disponer de la herramienta global de gestión de backlog (Jira, excepto en países donde no se pueda aplicar).

Adopción

2.2.1. Gestionar el backlog a nivel Program Increment (PI)

Para la correcta gestión de los proyectos, es importante tener identificados las evoluciones que se desean realizar durante la siguiente iteración (Feature) y que posteriormente serán desglosadas en funcionalidades más pequeñas y que veremos posteriormente (Historias de Usuario).

Los equipos de desarrollo deben tener muy presente que la **Feature** (funcionalidad que queremos desarrollar para ser puesta a disposición del cliente final o para algún servicio, aportando valor a cualquiera de ellos) **debe** ser uno de los principales artefactos a gestionar a nivel PI y **debe** ser el

Product Owner (PO) con la ayuda del Program Manager (PM) o Service Owner (SO) quien **debe** definir todas las Features atendiendo a los siguientes aspectos:

- Toda Feature **debe** ser **autocontenida funcionalmente**, es decir, no depende de otras para dar valor una vez finalizada.

- El responsable funcional del equipo (normalmente el Product Owner (PO)), con la ayuda del Program Manager (PM) o Service Owner (SO), es quien **debe** definir los requisitos de negocio que cada Feature debe cumplir para considerarse como completada.

- Una Feature **debe** llevar implícito **todas las fases del ciclo de vida de desarrollo** (Refinamiento/Diseño, Desarrollo, Pruebas y Despliegue) evitando así, crear Features para solo alguna de estas fases. La Feature será E2E, naciendo en la definición y finalizando en el despliegue a entornos de producción. La definición del **MSA queda fuera del alcance de la Feature**, al ser un ejercicio previo a la ejecución y cross a todo el desarrollo.

- No se **deberían** incluir nuevas Features que no se han planificado para el PI en curso. **Una vez iniciado el PI solo se deberían ejecutar las Feature comprometidas al inicio**. Esto podría ser excepcionado en casos en los que las Features planificadas queden bloqueadas/descartadas por algún imprevisto y por lo tanto, si se podrían incluir nuevas Features.

- En proyectos donde participa Holding y existen equipos mixtos con desarrolladores holding y locales, **deberán** no duplicar Features y compartir su trabajo sobre la misma Feature.

- Cuando existan **tareas o funciones dentro de los equipos de desarrollo** de software del Banco, que siendo necesarias trabajar **no implican desarrollo de software**, por ejemplo aquellas que el resultado de su consecución es la entrega de información/documentación, creación de prototipos o modelos teóricos de información, diseño de la feature, tareas más propias de gestión necesarias para la finalización de la Feature y que no conllevan desarrollo de software, los equipos de desarrollo del Banco **deberían** utilizar el artefacto denominado **Task**[4]. Este artefacto podría estar relacionado con una Épica al igual que la Feature y la duración de la resolución depende del tamaño, no se estipula referencia temporal.
  Dada su naturaleza, se define dicha Task con un flujo no guiado, permitiendo así adaptar los estados a la naturaleza de la actividad que se desee realizar en cada momento. Esto dará versatilidad, permitiendo así utilizar los estados según la necesidad de cada equipo (se podrán utilizar todos los estados, sin un orden predefinido).

  Si fuera necesario desglosar estas tareas para que puedan ejecutarse por integrantes diferentes del equipo o bien para simplificar el trabajo, los equipos **pueden** utilizar las Subtareas, con nivel de detalle más exhaustivo.

[4] Se utilizará dicho artefacto para las antiguas Features de tipo “Discovery” y “Functional”.

- Atendiendo a criterios de **implantación** los equipos deben tener en cuenta que:
    - Por regla general la Feature **debe** ser implantable (se puede subir a producción) por sí sola (sin depender de otras Features, excepto de las Features Enabler Delivery) en entornos de producción, permitiendo de esta manera eliminar posibles dependencias y aportando nuevas capacidades/servicios a los clientes.
    - No **debe** llevar asociado el código de las pull requests, ya que serán en las Historias de Usuario donde se deberán asociar.
    - Los equipos del Banco al finalizar una Feature **deberían** asociar una **release** (versión de un producto de software entregable) que incluye un conjunto de Historias de Usuario asociadas a una Feature.
    - Los equipos del Banco no **deben** cerrar una Feature que no ha sido finalizada o descartada. Entendiéndose por finalizada, cuando su funcionalidad está 100% disponible y el cliente (interno o externo) puede hacer uso de la misma.
    - Los equipos del Banco no **deben** cerrar una Feature con Historias de Usuario y/o Dependencias abiertas.

- **Debería** ser desarrollada con un **horizonte temporal** de 3 meses (1Q). Siendo complicado cumplir este objetivo temporal, es importante mantener y respetar siempre que sea posible, para cumplir la máxima de realizar entregas continuas a nuestros clientes, y establecer de este modo una mentalidad de entrega continua dentro de los equipos. Para ello en la creación de las Features, se deberá descomponer en la funcionalidad más pequeña posible, que permita entregar valor, ser implantable (se puede subir a producción) y realizarlo en el menor tiempo posible (idealmente en un trimestre).
  El equipo **podría** añadir en un campo label o junto a la descripción corta de la Feature, el tamaño de la Feature con tallas de camisetas (XL, L, M, S, XS). Esta estimación será orientativa y solo tendrá validez dentro del mismo equipo, siendo útil para estimar internamente la capacidad de ejecución.
  Nota: En los casos en los que la Feature no pueda ser finalizada en el Q previsto, no se deberá cerrar y abrir una nueva. La Feature se deberá planificar para el siguiente Q y se seguirá trabajando sobre la misma.

- Atendiendo a aspectos relacionados con la **trazabilidad**:
    - Una Feature sólo **debe** estar asociada a una Épica y **debe** ser una descomposición con más nivel de detalle de una Épica.
    - Es el Product Owner (PO) quien por norma general **debe** asociar la Feature a su correspondiente Épica.
    - Toda Feature **debe** componerse de más de una Historias de Usuario (para el desarrollo de software del propio equipo de trabajo) y/o Dependencias para solicitar capacidades a terceros si fuera necesario.
      En los casos en los que la Feature sea de Mantenimiento, debido al tipo de trabajo que se realiza en dichos equipos, se **podría** crear una Feature más general para albergar varias historias/bugs y evitar así tener que crear Features para temas muy específicos de una sola Historia de Usuario. Esto ayudará a los equipos de Mantenimiento a

realizar una gestión más eficiente de su trabajo, si bien cuando dichos equipos tengan Features de desarrollo de funcionalidades, deberán cumplir los criterios marcados en el playbook.

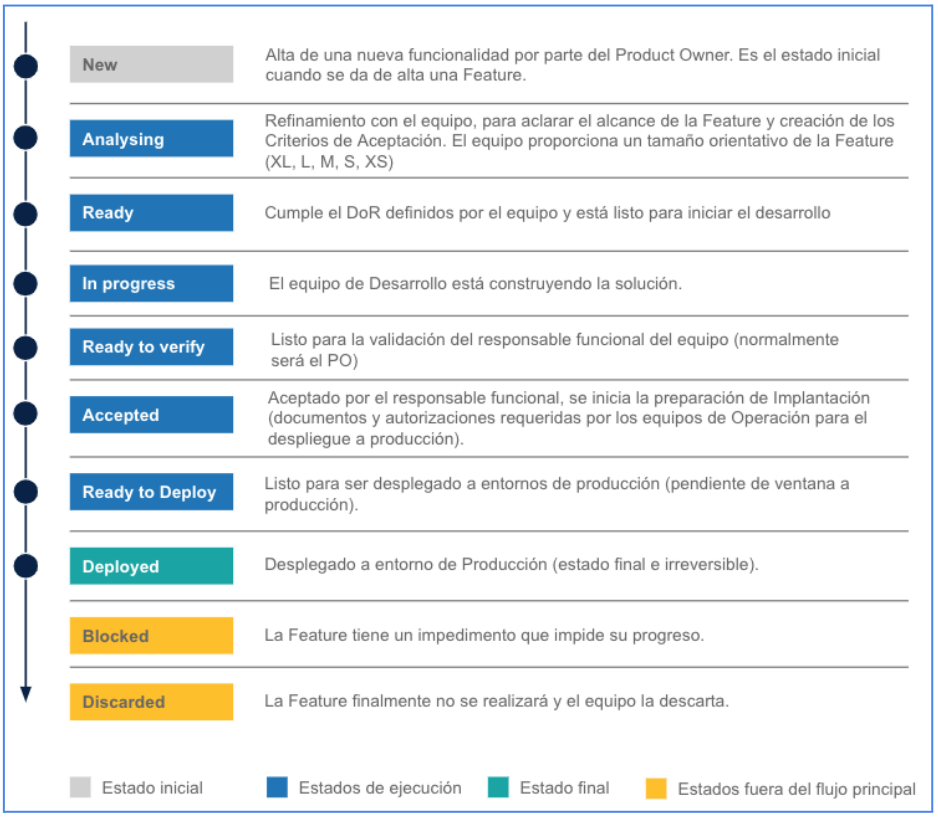
- Una Feature **debe** pertenecer a la funcionalidad de un Servicio. Esto no implica que el equipo que desarrolla dicha Feature no pueda realizar evoluciones sobre otros servicios siempre y cuando el equipo tenga los perfiles necesarios y haya sido coordinado con los Service Owners de dichos Servicios.
  En dichos casos en los que el equipo pueda trabajar en desarrollos de Servicios de terceros, estos **podrían** ser trazados con el Servicio correspondiente, teniendo así la trazabilidad completa.
  En el caso en el que no se disponga de los perfiles, se **debería** coordinar con el Servicio y Service Owner afectado, para que esté segundo pueda gestionar una Feature Enabler. Esta Feature Enabler **debería** estar enlazada a una Dependencia que será gestionada por el equipo peticionario.
- Existen requisitos dentro de una Feature que están asociados a otros equipos y/o servicios. Cuando estos requisitos no puedan ser realizados por el propio equipo de desarrollo (emisor) y el esfuerzo pueda ser realizado en un sprint, se **debe** abrir una **Dependencia** (explicado posteriormente) al otro equipo (receptor).
  Cuando el desarrollo requiere de un mayor esfuerzo, se **debe** conversar con el equipo receptor y con el (Service Owner en caso de ser necesario) para que se puedan crear una Feature de tipo Enabler (explicada más abajo). Esta Feature **debería** estar asociada a la Dependencia del equipo emisor, pudiendo así tener trazabilidad entre la petición del equipo emisor (con la Dependencia) y el trabajo del equipo Receptor (con la Feature Enabler). Adicionalmente destacar que dichas Features Enabler podrán a su vez tener Historias de Usuario asociadas.
  La identificación de Dependencias, **debe** ser un trabajo dentro de la planificación de las Features y en la medida de lo posible, todas deben ser creadas antes de comenzar los desarrollos, previo al inicio del PI.
  Durante las sucesivas sesiones de refinamiento de las Features, se identificarán necesidades que el equipo no podrá abordar, al tratarse de alguna acción centralizada por otro equipo o con un nivel de especialización con el que el equipo no cuenta. En este caso debemos avisar al responsable de cubrir nuestra necesidad y posteriormente dar de alta la Dependencia en la herramienta global de gestión de backlog asociada a la Feature de la que depende.

- La Feature **debe** dividirse en dos **categorías** diferentes (Ejemplos):

  - Feature "**CUSTOMER**". Será el Product Owner (PO) y Program Manager (PM) o Service Owner (SO), en función de si la feature depende de una Épica RUN o CHANGE, quienes con la ayuda, si fuera necesario, del Service Owner (SO) definirán este tipo de Feature siempre y cuando **conlleve desarrollo de software e implique la entrega de funcionalidades a nuestros clientes del Banco o empleados** (ej: Consulta de Movimientos, Login...).

- Feature "**ENABLER**". Será el Product Owner (PO) y Program Manager (PM) o Service Owner (SO), en función de si la feature depende de una Épica RUN o CHANGE, quienes definirán este tipo de Feature siempre y cuando **conlleve desarrollo de software el cual habilita una funcionalidad a un frontal u otras piezas core de producto o aplicación**, si bien una vez implantado, tanto nuestros **clientes del Banco como los empleados, no tendrán ninguna nueva funcionalidad o evolución** de una existente.

- El Product Owner (PO), Program Manager (PM) y Service Owner (SO) **deben** conocer que el **flujo de una Feature** es el mostrado a continuación y **debe** pasar por los estados indicados para poder desplegarla:



- Para crear una Feature debemos tener en cuenta los siguientes **campos** teniendo en cuenta si son obligatorios o no:

Campo	Descripción	Obligatorio	Opcional
Summary	Resumen de la funcionalidad	✓	
Description	Descripción en detalle de la funcionalidad	✓	
Epic	Enlace a la Épica que la engloba	✓*	
Team Backlog	Equipo que desarrollará la funcionalidad	✓	
Program Increment	Trimestre en el que se ha planeado su desarrollo		✓
SDA Project	Proyecto de la SDA	✓	
Delivery type	Tipo de funcionalidad que se desea desarrollar: Customer or Enabler	✓	
DoR	Puntos que se deben cumplir para que la feature sea considerada lista para que pueda comenzar a desarrollarse		✓
DoD	Puntos que se deben cumplir la feature para que pueda considerarse como terminada acorde con los requisitos establecidos		✓
Acceptance Criteria	Requisitos de negocio que cada feature debe cumplir para considerarse como completada		✓
Commitment type	Indica si la funcionalidad se ha comprometido para un trimestre o si se realizará si finalmente hay espacio pero sin compromiso de hacerse		✓
Priority	Prioridad asignada		✓
Labels	Agregador de features		✓
Attachment	Conjunto de archivos que se pueden adjuntar para definir la feature		✓

✓* Obligatorio cuando se transiciona de New a Analysing

Haciendo énfasis en los campos de Criterios de aceptación, DoD y DoR se deben tener en cuenta las siguientes pautas a la hora de informar estos campos:

- Criterios de aceptación: Los criterios de aceptación son una de las **prácticas más importantes y útiles** para el correcto funcionamiento de los equipos. Dichos criterios son el “contrato” entre el responsable funcional del equipo (normalmente el PO) y el equipo de Desarrollo, donde se determina las condiciones y requisitos que deben considerarse para dar la Feature por completada.
  - El responsable funcional del equipo (PO) **debe** asegurar que se cumplen dichos criterios de aceptación. Esta validación se realizará en el estado Ready to Verify y será dicho responsable quien **debe** transicionar al estado Accepted, evidenciando así que está todo correcto.
  - **Debe** especificar los límites y confirmar cuándo funciona según lo previsto.
  - **Debe** explicar en detalle la funcionalidad y los resultados que esta funcionalidad ofrecerá a los usuarios.
  - **Podría** marcar cuándo se puede probar (se sugiere seguir la norma "Given-When-Then") que define los criterios de aceptación y utilizarlos durante las pruebas de aceptación.

Ejemplo:

"Dado un usuario cliente que está logado cuando accede a la Posición Global entonces se le mostrará una lista con sus productos contratados."

- DoR (Definition of Ready).
  - Se **debería** crear DoR a nivel de Feature, para asegurar que no se inician las mismas sin tener todos los puntos relevantes para su ejecución resueltos.
  - Es el Product Owner (PO) con la ayuda del Program Manager (PM) o Service Owner (SO) quien **debería** definir claramente qué se debe cumplir para que la Feature sea considerada lista para que pueda comenzar a desarrollarse.
  - Siempre que exista **debe** haber un DoR por equipo.
  - El DoR **podría** ir adaptándose y revisarse siempre que ocurran cambios en los entornos o procesos y se identifiquen oportunidades de mejora.
  - Ejemplos:

    "Las dependencias están identificadas y resueltas."

    "No tiene bloqueos que impidan su ejecución."

- DoD (Definition of Done).
  - Se **debería** crear DoD a nivel de Feature, para asegurar que no se finalizan las mismas sin tener todos los puntos relevantes para su finalización resueltos.
  - Es el Product Owner (PO) con la ayuda del Program Manager (PM) o Service Owner (SO) quien **debería** definir el conjunto de criterios para saber que deben cumplir las Features para que puedan considerarse como completadas.
  - Siempre que exista **debe** haber un DoD por equipo.
  - El DoD **podría** ir adaptándose y revisarse siempre que ocurran cambios en los entornos o procesos y se identifiquen oportunidades de mejora.

    Ejemplos:

    "Documentación realizada"

    "Aprobado por el usuario"

Herramientas necesarias

La adopción de esta práctica por parte de los equipo del Banco requiere el uso de la siguiente herramienta:

- **Jira:** Servicio global para la gestión de proyectos.

Indicadores para el equipo

Los indicadores que apliquen, **deben** ser seguidos por los equipos, para conocer el grado de madurez respecto a la práctica:

Indicador	Descripción
Predictibilidad	Features entregadas vs comprometidas
Features finalizadas	Número de features finalizadas en un Q
Features no planificadas	Número de features no planificadas al inicio del Q e incorporadas al equipo posteriormente
Features en curso	Número de Features en curso
Lead Time	Medición del tiempo transcurrido entre el momento en que el equipo comienza a dedicar recursos (estado análisis) hasta su finalización (estado Deployed)
Promedio de Dependencias por Feature	Promedio de Dependencias asociadas a una Feature
Promedio de Historias de Usuario por Feature	Promedio de Historia de Usuario asociadas a una Feature
Features descartadas	Número de Features iniciadas y posteriormente descartadas
Features bloqueadas	Número de Features bloqueadas por Q
Tiempo de bloqueo	Medición del tiempo medio de bloqueo de una Feature

Nota: Dichos indicadores son mediciones que deberán utilizar los equipos de forma interna para conocer mejor cómo están trabajando y poder mejorar de forma objetiva.

Enlaces

- How To Define Features in Agile Methodology?
- Everything You Need to Know About Acceptance Criteria

2.2.2. Gestionar el backlog a nivel Sprint

La gestión del backlog a nivel sprint es importante para que los equipos de desarrollo puedan detallar el trabajo que realizan. Dicha información podrán detallarla en las Historias de Usuario funcionales (desglose de las Features que permita realizar incrementos en sprints de 2 semanas), Dependencias (peticiones a terceros que no suponen un elevado esfuerzo (aprox 1 sprint) y Bugs donde los equipos detallan los errores que se identifican bien durante las pruebas o en producción, siempre siguiendo las siguientes directrices:

- Los equipos **deben** incluir en el backlog a nivel sprint todo lo planificado, en caso que exista alguna tarea que no haya sido planificada pero que se considere que es estrictamente necesario realizar, también deberíamos incluirla en el backlog.

- Para definir las **Historias de Usuario** se debe tener en cuenta que:

    - La Historia de Usuario **debe** ser una **descripción breve de una funcionalidad** contada desde la perspectiva del cliente y escrita en el lenguaje común del usuario, y que son resultado de descomposición de las Features por parte del equipo (preferiblemente el Product Owner), aunque en situaciones excepcionales también podrán ser creadas por el equipo de desarrollo, aunque siempre debe ser revisada y aprobada por el PO.

    - Se **podrá** utilizar el **método INVEST** para asegurar la calidad en la escritura de las Historias de Usuario.

    - La medición de los **puntos de historia (SP**) **debería** hacerse utilizando el método de **Fibonacci** con la participación del equipo. En la práctica, los equipos podrían utilizar una secuencia exponencial de números como puntos de historia para representar el tamaño, la complejidad, la incertidumbre y el esfuerzo necesarios para completar o implementar una Historia de Usuario.

    - Estas Historias de Usuario **deben** ser **priorizadas en función de su importancia y de los recursos necesarios** para completarlas y es el equipo el que va trabajando sobre ellas para ejecutar el trabajo.

    - Las Historias de Usuario **deben** ser **creadas de forma previa al sprint** donde se van a desarrollar los trabajos y el equipo tendrá que refinar las Historias de Usuario para establecer el compromiso del siguiente Sprint.

    - Las Historias de Usuario **deben** llevar **asociado el código software**[5] que le corresponde, y por tanto, se deberán relacionar con las PRs de integración (los commit de código deberán estar asociadas a la issue de tipo Historia). De este modo se tendrá **trazabilidad entre el requerimiento funcional (Feature), las tareas que son necesarias para su consecución (Historias de Usuario) y el código generado** para dicho

[5] Esta relación solo será obligatoria para todo el código que esté almacenado dentro del repositorio global de código (Bitbucket)

requerimiento. Esta trazabilidad es fundamental para un mejor Mantenimiento y para la resolución de problemas.

- Se **podría** asociar la Historia de Usuario con los tickets ASO y APX, creados para cubrir el objetivo de dicha Historia de Usuario.

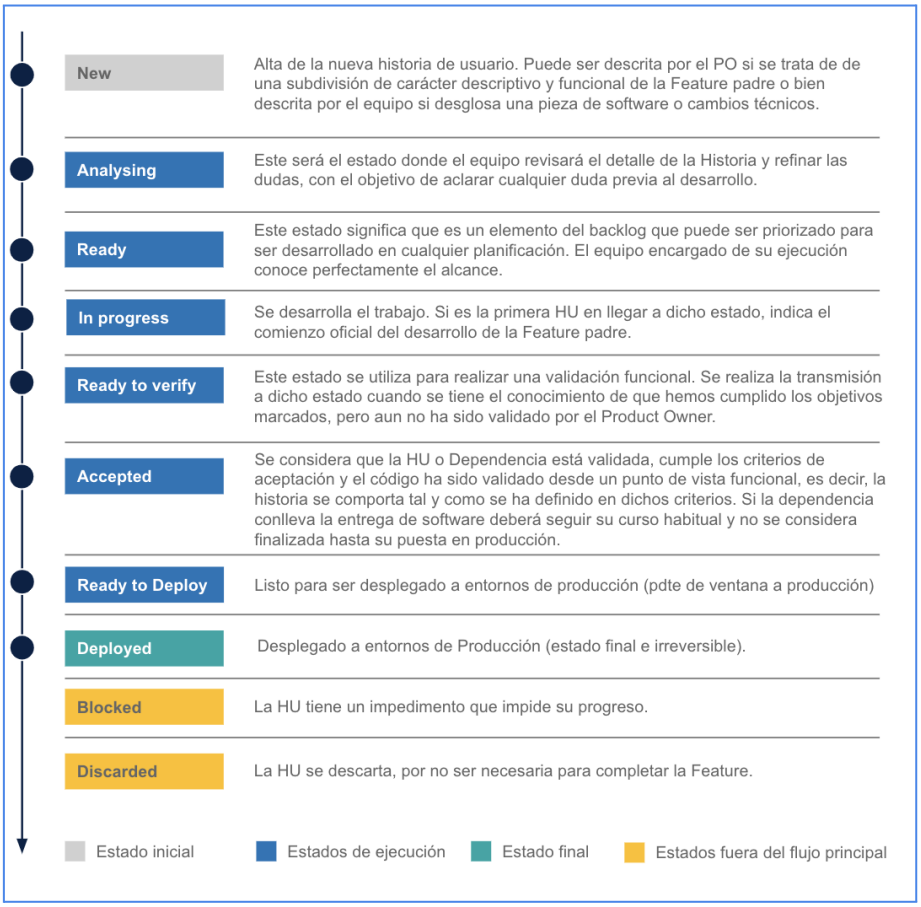
- Si necesitamos desglosar las Historias de Usuario para que puedan desarrollarse por integrantes diferentes del equipo o bien para simplificar el desarrollo, los equipos **podrían** utilizar las **Subtareas,** estas deberían contener definiciones técnicas como por ejemplo, definir la estructura de la base de datos, implementar una función de código para ordenar determinadas transacciones por fecha.

- En caso de ser necesario, se **podría** hacer uso de **Impedimentos** para visualizar las Dependencias que no se han resuelto a tiempo.

- **Debería** tener un **horizonte temporal** de un sprint para completarse, es importante mantener y respetar siempre que sea posible este espacio temporal, para cumplir la máxima de realizar entregas a nuestros clientes. En cualquier caso, al ser una parte de una Feature, esta debe tener un peso menor en cuanto a esfuerzo de trabajo. Si la Historia de Usuario está comprometida en el sprint, entonces **debería** finalizar antes de que el sprint termine.

- Atendiendo a aspectos relacionados con la **trazabilidad**:
  - **Debe** estar siempre ligada a una Feature (donde se va a desarrollar la funcionalidad).

- El equipo de desarrollo debe conocer que el **flujo de una Historia de Usuario** es el mostrado a continuación y debe pasar por los estados indicados para poder desplegarla:



- Para crear una Historia de Usuario debemos tener en cuenta los siguientes **campos** teniendo en cuenta si son obligatorios o no:

Campo	Descripción	Obligatorio	Opcional
Summary	Resumen de la funcionalidad	✓	
Description	Descripción en detalle de la funcionalidad	✓	
Feature Link	Enlace a la Feature que la engloba	✓	
Team Backlog	Equipo que desarrollará la funcionalidad	✓	
User Story Type	Se especifica el tipo de historia de usuario: New Functionality, Tech Debt o Platform Improvement		✓
DoR	Puntos que se deben cumplir para que la historia sea considerada lista para que pueda comenzar a desarrollarse		✓
DoD	Puntos que se deben cumplir la historia para que pueda considerarse como terminada acorde con los requisitos establecidos		✓
Acceptance Criteria	Requisitos de negocio que cada historia debe cumplir para considerarse como completada		✓
Story Points	Tamaño de la historia de usuario (usando valores de secuencia de Fibonacci)		✓
Priority	Prioridad asignada		✓
Sprint Estimate	Indicará cuándo estima que se implementará la funcionalidad		✓
Fix Version	Versión en la que se entrega la historia		✓
Affects Version	Versión afectada por la historia		✓
Labels	Agregador de historias		✓
Attachment	Conjunto de archivos que se pueden adjuntar para definir la historia		✓

✓ Obligatorio cuando se transiciona de New a Analysing

Haciendo énfasis en los campos de Criterios de aceptación, DoD y DoR se deben tener en cuenta las siguientes pautas a la hora de informar estos campos:

- Criterios de aceptación.
  - El Product Owner junto con el equipo de desarrollo, si fuera necesario, **deben** definir los requisitos de negocio que cada Historia de Usuario debe cumplir para considerarse como completada siguiendo las siguientes pautas:
  - **Debe** especificar los límites y confirmar cuándo funciona según lo previsto.
  - **Debe** explicar en detalle la funcionalidad y los resultados que esta funcionalidad ofrecerá a los usuarios.
  - El responsable funcional del equipo (PO) **debe** asegurar que se cumplen dichos criterios de aceptación. Esta validación se realizará en el estado Ready to Verify y será dicho responsable quien **debe** transicionar al estado Accepted, evidenciando así que está todo correcto.
  - **Debería** marcar cuándo se puede probar una historia (se sugiere seguir la norma "***Given-When-Then***" que define los criterios de aceptación y utilizarlos durante las pruebas de aceptación.
  - Ejemplo:

    Dado un usuario cliente que está en la página de búsqueda

cuando selecciona la opción “Ver Productos” entonces se le mostrará una lista con los productos en venta .

- DoR (Definition of Ready).
  - El Product Owner junto con el equipo de desarrollo, si fuera necesario, **debería** definir de forma clara el DoR (Definition of Ready) que debe cumplir una historia para ser considerada lista para el desarrollo.
  - **Debería** haber una DoR general por equipo y tipo de tarea.
  - Ejemplos:
    Las dependencias están identificadas y resueltas.
    No tiene bloqueos que impidan su ejecución.

- DoD (Definition of Done).
  - El Product Owner junto con el equipo de desarrollo, si fuera necesario, **debería** definir el DoD (Definition of Done) que debe cumplir una Historia de Usuario por equipo y tipo de tarea para que pueda considerarse como completada. Definir esto debe garantizar que el trabajo es de alta calidad y cumple los requisitos establecidos en los criterios de aceptación del proyecto.

  - Ejemplo:
    Documentación requerida.
    Alineado con las normas de codificación.
    Cumple los criterios de aceptación.
    Mantiene la estabilidad del servicio.

- Otro artefacto a la hora de gestionar el backlog a nivel sprint que los equipos de desarrollos tienen que tener en cuenta es el **Bug** (error o fallo en un software que causa un comportamientos no deseado o incorrecto) y seguirá las siguientes directrices:

  - **Debe** ser solucionado por el equipo responsable de la versión que produce el fallo o en su defecto el equipo encargado del mantenimiento del servicio.

  - El error puede aparecer en producción o en entornos anteriores. Si el error se produce en un entorno productivo **debemos** abrir un bug, pero si se produce en entornos previos se **podría** gestionar (a criterios del equipo) bien abriendo un bug o pasando a estado To Rework la Historia de Usuario indicando el problema sucedido.

  - Dadas las circunstancias de los equipos del Banco, un bug **podría** ser resuelto mediante cambios en el código del software o por otras acciones como por ejemplo cambios en infraestructura, cambios en parametrías, reinicios de máquinas.

- El **horizonte temporal** de la resolución depende del tamaño del bug.

- Atendiendo a aspectos relacionados con la **trazabilidad**:
  - Un bug **debería** estar vinculado a una Historia de Usuario o a una Feature aunque para los casos en los que el bug es identificado en producción tiempo después de la implantación, no será obligatoria dicha relación.
  - **Debe** sólo implicar a un servicio.
  - **Podría** tener algunas Dependencias con algunos servicios secundarios.

- El equipo de desarrollo debe conocer que el **flujo de un bug** es el mostrado a continuación y debe pasar por los estados indicados para poder desplegarla:

- Para crear un bug debemos tener en cuenta los siguientes **campos** teniendo en cuenta si son obligatorios o no:

Campo	Descripción	Obligatorio	Opcional
Summary	Resumen de la funcionalidad	✓	
Description	Descripción en detalle de la funcionalidad		✓
Feature Link	Enlace a la Feature que la engloba		✓
Team Backlog	Equipo que desarrollará la funcionalidad	✓	
Stage	Entorno en el que se ha detectado el bug	✓	
Origin	Indica la forma en que se ha localizado el bug		✓
Priority	Prioridad asignada	✓	
Support cases	Código del ticket de Remedy/Helix		✓
DoD	Puntos que se deben cumplir el bug para que pueda considerarse como terminado acorde con los requisitos establecidos		✓
Fix Version	Versión en la que se entrega el bug		✓
Affects Version	Versión afectada por el bug		✓
Story point	Tamaño del bug (usando valores de secuencia de Fibonacci)		✓
Labels	Agregador de historias		✓
Attachment	Conjunto de archivos que se pueden adjuntar para definir el bug		✓

- Y el tercer y último artefacto a la hora de gestionar el backlog a nivel sprint sería la **Dependencia** (requisitos dentro de una Feature que un equipo no puede abordar con autonomía al estar asociados a otros servicios o equipos y tiene la necesidad de la colaboración para entregar lo descrito en la Feature) y estas **deben** ser gestionado por los equipos de desarrollo de la siguiente manera:
  - Las Dependencias deben permitir a otros equipos proporcionar las capacidades necesarias para completar el trabajo.

  - Una Dependencia podría ser solicitada a un Team Backlog del propio servicio o de un Servicio de terceros.

  - Al ser una tarea que forma parte de un objetivo concreto, como es una Feature, esta debe tener un alcance muy específico, pactado con el equipo receptor de la misma, para que pueda ser planificada, priorizada y trabajada en un sprint o un corto espacio de tiempo (máximo 3 sprint) . En caso de que no sea posible su ejecución en un corto periodo de tiempo (máximo 3 sprint), se debe gestionar como hemos comentado en el apartado anterior, abriendo una Feature Enabler.

  - Podríamos decir que que la Dependencia es como la Historia, siendo las historias las realizadas por el propio equipo y las Dependencias las realizadas por un equipo tercero.

- El equipo receptor **debe** trabajar sobre la propia Dependencia, permitiendo de ese modo, tener informado al equipo peticionario, sobre el estado de desarrollo de la misma.

- Una parte importante de la **gestión del backlog es la coordinación y seguimiento** del trabajo del equipo, por ello se establece que todos los equipos de desarrollo del Banco **deberían** tener creados los siguientes tipos de backlog:
  - **Sprint Backlog** (conjunto de elementos del Backlog seleccionados para el Sprint)**:** todos los equipos de scrum **deberían** hacer uso de este tipo de backlog y debe ser el plan para entregar el incremento y alcanzar el objetivo del Sprint. Debería ser una estimación realizada por el equipo Scrum sobre qué funcionalidad formará parte del siguiente incremento y el esfuerzo necesario para entregarla.
  - **Backlog** (lista ordenada de elementos con todo lo que se sabe que se necesita para un resultado concreto)**: debe** ser la única fuente de requisitos para cualquier entrega o cambio que se vaya a realizar a utilizar por todos los equipos de desarrollo. El Backlog **debe** enumerar todas las funcionalidades, requisitos, mejoras y correcciones que se deben realizar en un servicio para futuras entregas.

- Existen **métricas** a nivel de Historia de Usuario que **deberían** ser utilizadas por los equipos de desarrollo del Banco para identificar su velocidad y nivel de madurez. Estas métricas deben apoyar al equipo y su trabajo y tiene como beneficios mejorar la previsibilidad en la medida que los equipos puedan planificar y cumplir de forma coherente el trabajo que se comprometieron a realizar para apoyarles en la planificación futura. Aumentar el flujo para mejorar la rapidez y eficacia con que los equipos pueden ofrecer valor a los clientes y responder a las necesidades del mercado y mejorar el compromiso de los equipos con el trabajo que realizan y el entorno en el que se encuentran maximizando de esta manera la eficacia de los desarrolladores. Por lo tanto, teniendo en cuenta esto:

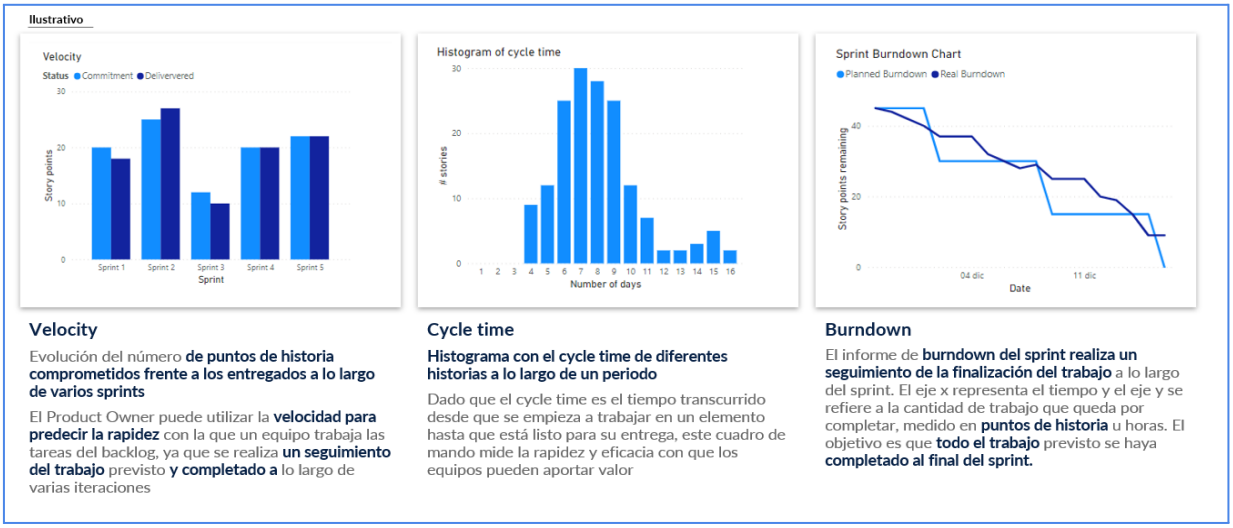
  - Se debería analizar la pendiente de la curva definida por el número de puntos de la historia (eje y) completados a lo largo del tiempo (eje x), **Burndown chart,** con el objetivo de realizar un seguimiento del trabajo (por ejemplo, la finalización del trabajo a lo largo del sprint).
    Un gráfico de burndown satisfactorio debe tener una pendiente lo suficientemente pronunciada como para cruzar el eje x antes de la fecha límite (final del sprint).

  - Se debería medir la cantidad de puntos de historia completados por el equipo durante un sprint, **User story velocity**, con el objetivo de identificar la productividad de un equipo scrum.

  - Se debería medir el tiempo transcurrido desde el inicio del desarrollo en una Historia de Usuario hasta su finalización, **User Story cycle time**, con el objetivo de determinar cuánto tiempo (en tiempo de calendario) se tarda en completar una Historia de Usuario.

- Se debería medir el número de días necesarios para completar una Historia de Usuario desde su concepción (Analizando) hasta que cumple el DoD (Desplegada), **User story lead time**, con el objetivo de determinar cuánto tiempo (en tiempo de calendario) se tarda en preparar y completar una Historia de Usuario.

- Se debería calcular el porcentaje de Historias de Usuario que se completan sobre el número de Historias de Usuario que se planificaron para el sprint, **Ontime user stories completion**, con el objetivo de identificar la previsibilidad del equipo.



Dashboards que se crearán para hacer un seguimiento de los KPI clave.

Herramientas necesarias

La adopción de esta práctica por parte de los equipo del Banco requiere el uso de la siguiente herramienta:

- **Jira:** Servicio global para la gestión de proyectos.

Indicadores

Los indicadores que apliquen, deberían ser proporcionados a los equipos para conocer el grado de madurez respecto a la práctica:

Indicador	Descripción
Incremento Nª Puntos Historia	Diferencia del número de puntos de historia completados entre un sprint y otro.
Incremento Tiempo Medio Desarrollo	Tiempo transcurrido desde el inicio del desarrollo en una Historia de Usuario hasta su finalización
Incremento Nª Historias	Número de historias entregadas entre un sprint y otro.

Incremento Nº Dependencias realizadas	Diferencia del número de dependencias realizadas en un trimestre / número total de dependencias estimadas
Incremento Nº Bugs detectados	Número de bugs detectados entre un trimestre y otro
Historias de Usuario descartadas	Número de Historias de Usuario iniciadas y posteriormente descartadas
Historias de Usuario bloqueadas	Número de Historias de Usuario bloqueadas por Sprint
Tiempo de bloqueo	Medición del tiempo medio de bloqueo de una Historias de Usuario

Nota: Dichos indicadores son mediciones que deberán utilizar los equipos de forma interna para conocer mejor cómo están trabajando y poder mejorar de forma objetiva.

Enlaces

- How to Write User Stories in Agile Software Development
- What's the Difference between DoR, DoD and Acceptance Criteria?
- Detalle para un mejor entendimiento del método Fibonacci
- Método INVEST
- Como diferenciar la Definición de Done y Criterio de aceptación
- Five agile KPI metrics you won't hate

2.3. Gestionar la documentación de los proyectos.

Los equipos de desarrollo de software **deben** utilizar una herramienta de colaboración y gestión de documentos para poder ordenar la información funcional del ciclo de vida de desarrollo y aplicar las directrices recogidas en esta sección cuando la tecnología lo permita.

Beneficios

El propósito y beneficios de utilizar una herramienta de colaboración y gestión de documentos es:

- Proporcionar una ubicación central para almacenar y organizar todo el trabajo que debe realizarse en un proyecto de software.
- Permite a los miembros del equipo colaborar en el desarrollo del proyecto creando y compartiendo contenido, realizando un seguimiento del progreso y comunicándose entre sí.
- Crea, colabora y organiza todo tu trabajo en un único lugar.

Precondiciones

Para poder aplicar la práctica se deben dar las siguientes **precondiciones**:

- Disponer de la herramienta Confluence.

Adopción

2.3.1. Gestionar la documentación de los proyectos.

Los equipos del Banco deben adoptar las siguientes directrices relacionadas con la práctica:

- **Deben** crear un espacio de proyecto para todos los conocimientos relevantes en Confluence.
- El espacio de proyecto **debe** actuar como un único punto de contacto para consultar los detalles del proyecto, con el objetivo de que el equipo pueda acceder fácilmente a la información.
- **Deben** crear una jerarquía organizada y estructurada en Confluence bien:
    - creando un espacio para cada equipo funcional y páginas para cada iniciativa o proyecto importante.
    - creando un espacio para cada proyecto y páginas para cada ciclo o lanzamiento de publicación.
- **Deben** vincular las páginas de Confluence con los proyectos de la herramienta global de gestión de backlog. De este modo, el equipo dispone de todo el contexto, la información y la trazabilidad que necesita.

Herramientas

La adopción de esta práctica por parte de los equipo del Banco requiere el uso de la siguiente herramienta:

- **Confluence:** herramienta de colaboración y gestión de documentos.

Indicadores

Los indicadores que apliquen, **deben** ser proporcionados a los equipos para conocer el grado de madurez respecto a la práctica:

Indicador	Descripción
TBD	TBD

Enlaces

- Confluence best practices

3. Proceso

Durante la fase de despliegue de la transformación del software debe realizarse una migración a las definiciones, procedimientos y directrices globales de backlog definidas en este playbook.

La migración no se ejecutará sobre el backlog histórico (por ejemplo, el trabajo realizado en sprints anteriores**) sino solo del trabajo actual o futuro**, en caso de incompatibilidad. En caso contrario, sí se llevará ese histórico al nuevo formato.

Con el fin de **garantizar que todas las regiones y equipos adopten el nuevo proceso de gestión de backlog** habrá una estrategia de gestión del cambio centrada en alinear a todas las partes interesadas, crear una historia del cambio, identificar los canales para implicar a todos los segmentos y desarrollar un plan de comunicación.

La generación y gestión diaria del backlog sigue **una serie de ceremonias** y **un proceso determinado articulado en torno a ellas** y que podemos ver de forma resumida en el siguiente gráfico:

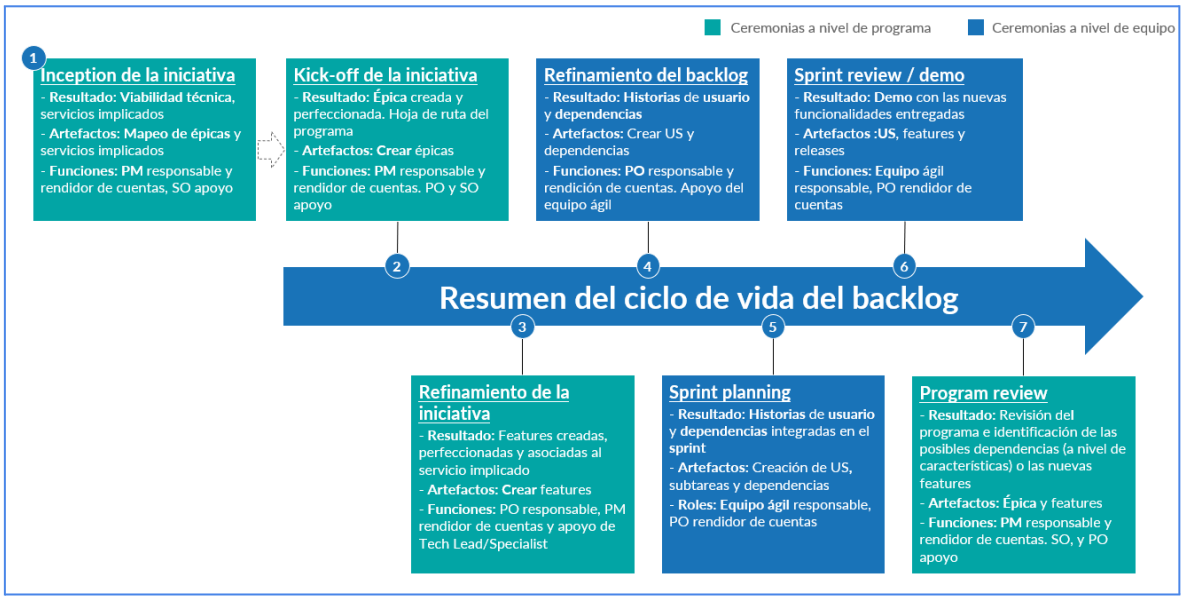


Ilustración del flujo de trabajo de la generación y gestión del backlog

Para las Épicas CHANGE aplicarían todas las ceremonias expuestas anteriormente, sin embargo para las Épicas RUN al menos deberían celebrarse el refinamiento de la iniciativa, el refinamiento del backlog, sprint planning y sprint review/demo.

De la ilustración anterior pueden derivarse los siguientes pasos sobre cómo pasar del nivel de iniciativa al de Historia de Usuario a través de las ceremonias:

1. **Inception de la iniciativa:** El PM responsable de la iniciativa de negocio solicita un refinamiento a los principales SOs afectados por la iniciativa para identificar todos los servicios implicados y comprobar la viabilidad de la iniciativa.
    - **Resultado:** Viabilidad técnica, servicios implicados.
    - **Cuando:** Antes de la priorización de la SDA.

- **Artefactos:** Mapeo de las Épicas y servicios implicados.
- **Roles**: PM responsable y accountable, apoyo SO.

2. **Kick-off de la iniciativa:** Una vez que el PM obtiene el equipo necesario (idealmente 100% autónomo) para poner en marcha la iniciativa, se crean las Épicas, que se perfeccionan con los PO recibiendo el apoyo de los SO (si es necesario).
   - **Resultado:** Épicas creadas y refinadas con PO de la iniciativa o de otras iniciativas.
   - **Cuando:** Después de la priorización SDA.
   - **Artefactos:** Épicas creadas.
   - **Roles**: PM responsable y accountable, PO y SO de apoyo.

3. **Refinamiento de la iniciativa:** Cuando los PO reciben las Épicas refinadas, las dividen en Features para su equipo o para equipos de terceros si se trata de Dependencias. Además, refinan las Features con el apoyo del Tech Lead (si es necesario).
   - **Resultado:** Features creadas, asociadas al servicio implicado y refinadas con el PM.
   - **Cuando:** Después de que las Épicas estén bien definidas.
   - **Artefactos:** Features creadas.
   - **Roles**: Responsable el PO, accountable el PM y apoyo del Tech Lead.

4. **Refinamiento del backlog:** Una vez que el PO ha creado las Features, se perfeccionan con el equipo para crear las respectivas Historias de Usuario (US), subtareas y dependencias en el backlog del equipo y se ordenan por prioridad.
   - **Resultado:** US y dependencias creadas.
   - **Cuando:** Antes de que comience el sprint.
   - **Artefactos:** US, subtareas y dependencias creadas.
   - **Roles**: PO responsable y accountable, apoyado por el equipo de desarrollo (**Nota**: Para las subtareas o dependencias técnicas, el equipo de desarrollo será responsable de la creación)

5. **Sprint planning:** Cuando el equipo tiene listo el team backlog con las Historias de Usuario y las dependencias, comienza a asignar puntos de historia (PE) y decide qué Historias de Usuario se compromete y priorizan para el siguiente sprint.
   - **Resultado:** US y dependencias integradas en el sprint backlog.
   - **Cuando:** Antes de que comience el sprint.
   - **Artefactos:** US, creación de subtareas y dependencias.
   - **Roles**: Equipo de desarrollo responsable, PO accountable.

   Durante la ejecución del sprint, el equipo de desarrollo junto con el PO actualizará los artefactos del sprint (es decir, Features, Historias de Usuario, Subtareas y Dependencias).

6. **Sprint Review/Demo:** Una vez que el equipo ha finalizado el sprint y se han entregado las Historias de Usuario, se demuestra la funcionalidad conseguida idealmente en entornos productivos. Cuando no sea posible se podrá mostrar en pre producción o desarrollo. En el

caso de presentar una Feature en una demo, siempre debe mostrarse en entornos productivos
    - **Resultado:** Demostración con las nuevas funcionalidades entregadas (entorno de producción, preproducción o desarrollo)
    - **Cuando:** Después del sprint.
    - **Artefactos:** US, Features y releases.
    - **Roles**: Equipo de desarrollo responsable, PO accountable.

7. **Program review:** Periódicamente, se sincroniza el trabajo en curso para el programa y se identifican nuevas prioridades y dependencias.

Además, las siguientes ceremonias deberían tener lugar periódicamente como parte de la gestión o revisión del backlog:

- **Program retro:** Revisión en cada ciclo (trimestre) de la entrega y los mecanismos buscando oportunidades para optimizar los procesos de entrega.

- **Sprint retro**: Reunión al final de cada sprint para reflexionar sobre el rendimiento pasado e identificar posibles mejoras. Esta reunión debería servir como punto de cierre al sprint y una vez realizada esta ceremonia se volvería a refinar el backlog (sprint planning) y el ciclo de vida del sprint comienza de nuevo.

- **Daily stand-up:** Reunión diaria con todos los miembros del equipo para revisar el plan de trabajo del día e identificar posibles impedimentos.

- **Service maintenance review:** Periódicamente, el SO revisa el estado de cada servicio comprobando las nuevas características, dependencias o fallos integrados en el service backlog.