## Procesos definidos vs procesos empíricos:

Existen dos tipos de procesos:

* Procesos definidos: hay un contexto organización, que define la forma de trabajar, y espera que los proyectos que se vayan a llevar adelante sigan con esta forma de trabajar. Lo que buscan es poder repetibilidad, es poder estimar nuevos proyectos a partir de proyectos pasados. Ejemplos, PDU/RUP.
* Procesos empíricos: se basan en la experiencia, pero plantean una forma de generar esa experiencia. Sostiene que la experiencia no es extrapolable, es decir, la experiencia de uno no se transmite a los demás. Plantean que la mejor forma de tomar decisiones es que la realice el que va a hacer el proyecto. Los equipos tienen que funcionar autónomamente. El empirismo trabaja con 3 pilares que son los que determinan que esto pueda funcionar, estos pilares son los que toman el agilismo:
  + Inspección: paro lo que se está haciendo y reviso. Se inspecciona el proceso (se hace con el equipo) y el producto (se hace con el cliente).
  + Adaptación: adaptarse a los cambios detectados en la inspección.
  + Transparencia: la información debe ser visible y disponible para todas las personas implicadas en el proceso.

En la inspección y adaptación es donde se da la iteración, la realimentación es la experiencia.

El proceso se adapta al proyecto.

## Manifiesto ágil:

Valores del manifiesto ágil:

1. Valoramos más el software funcionando que documentación exhaustiva.
2. Valoramos más los individuos e interacciones entre los individuos que procesos y herramientas.
3. Valoramos más el cambio (aceptar que los requerimientos pueden cambiar) que el cumplimiento de un plan.
4. Valoramos más la colaboración con el cliente que la negociación contractual.

El agilismo no implica que sea rápida, no implica que no se documente.

El agilismo para funcionar usa procesos empíricos.

### Los 12 principios del manifiesto ágil:

1. Nuestra mayor prioridad es satisfacer al cliente.
2. Aceptar que los requisitos cambien.
3. Entregar software funcional frecuentemente.
4. Los responsables de negocios, diseñadores y desarrolladores deben trabajar juntos día a día durante el proyecto.
5. Desarrollamos proyectos en torno a individuos motivados.
6. El método más eficiente de comunicar información es conversaciones cara a cara.
7. El software funcionando es la principal medida de éxito.
8. Los procesos ágiles promueven el desarrollo sostenible.
9. La atención continua a la excelencia técnica y al buen diseño mejora la agilidad.
10. La simplicidad es esencial.
11. Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados.
12. A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo y de acuerdo a esto ajustan su comportamiento.

### ¿Qué es ágil?

No es una metodología o un proceso. Ágil es una ideología con un conjunto definido de principios que guían el desarrollo del producto. Es una filosofía, es un pensamiento, es algo que apunta a la cultura de las personas que lo adoptan.

Valores de los equipos ágiles:

* Planificación continua, multi-nivel.
* Facultados, auto-organizados, equipos completos.
* Entregas frecuentes, iterativas y priorizadas.
* Prácticas de ingeniería disciplinadas.
* Integración continua.
* Testing concurrente.

Ágil es un balance entre ningún proceso y demasiado proceso. La diferencia inmediate es la exigencia de una menor cantidad de documentación, sin embargo, no es lo más importante.

* Los métodos ágiles son adaptables en lugar de predictivos.
* Los métodos ágiles son orientados a la gente en lugar de orientados al proceso.

#### Algunos frameworks ágiles:

Diagrama

Descripción generada automáticamente

### Requerimientos en agile:

* Usar el “valor” para construir el producto correcto.
* Usar historias y modelos para mostrar que construir.
* Determinar que es “sólo lo suficiente”.

Implica aplicar inspección y adaptación, es decir iterar, por ser un proceso empírico.

### Gestión ágil de requerimientos de software:

Los requisitos cambiantes son una ventaja competitiva si puede actuar sobre ellos.

Imagen que contiene Escala de tiempo

Descripción generada automáticamente

#### Tipos de requerimientos:

* Requerimiento de negocio: disminuir x% de tiempo invertido en procesos manuales relacionados con atención al cliente.
* Requerimiento de usuario: realizar consultas en línea del estado de cuenta de los clientes.
* Requerimiento funcional: generar reporte de saldos de cuenta. Recibir notificaciones por mail.
* Requerimiento no funciona: formato del reporte PDF. Cumplir niveles de seguridad para credenciales de usuarios según la ley bancaria 9999XX.
* Requerimiento de implementación: servidores en la nube.

### Tradicional vs ágil:

* Los cambios son la única constante.
* Stakeholders: no son todos los que están.
* Siempre se cumple eso de que: “el usuario dice lo que quiere cuando recibe lo que pidió”.
* No hay técnicas ni herramientas que sirvan para todos los casos.
* Lo importante no es entregar una salida, un requerimiento, lo importante es entregar, un resultado, una solución de “valor”.

### Principios ágiles relacionados a los requerimientos ágiles:

1. La prioridad es satisfacer al cliente a través de releases tempranos y frecuentes.
2. Recibir cambios de requerimientos, aún en etapas finales.

4. Técnicos y no técnicos trabajando juntos todo el proyecto.

6. El medio de comunicación por excelencia es cara a cara.

11. Las mejores arquitecturas, diseños y requerimientos emergen de equipos autoorganizados.

## Product Backlog:

Es un contenedor de ítems de product backlog, un ítem de producto backlog es una user story. Los ítems que tienen más prioridad son los de arriba, la prioridad no siempre se refiere a que sea más importante, también puede pasar que un ítem de product backlog sea menos importante, pero se requiera antes.

Es una cola o lista priorizada de ítems de product backlog (características de producto).

No puede haber tareas en un product backlog. Puede haber:

* Defectos.
* Deuda técnica.
* Épica.
* Temas.
* Spikes: aparecen cuando hay incertidumbre, cuando es tan grande que no sé porque falta información.

## User stories:

Son requerimientos de usuario o de negocio, el nivel de abstracción que tienen las user stories es más alto que el de los casos de uso.

Es una descripción CORTA de una funcionalidad esperada del producto que tienen que satisfacer. Describen una necesidad.

Las ideas de las US tienen que venir de product owner.

Se las llama “stories” porque se supone que cuenta usted cuenta una historia. Lo que se escribe en la tarjeta no es importante, lo que usted habla, sí.

Partes que conforman una user story:

* Conversación: es lo que se habla con el cliente, no se guarda en ningún lado, pero a partir de la conversación se obtiene la información para la US.
* Tarjeta: es la parte visible de la US, es lo que efectivamente escribimos y se ingresa en el product backlog. La sintaxis recomendada para la Card responde a: who, what y why (sería el titulo siguiente).
* Confirmación: serían las pruebas de aceptación de usuario.

### Forma de expresar las historias de usuario:

Como <nombre del rol>, yo puedo <actividad> de forma tal que <valor de negocio que recibo>.

* Nombre del rol: representa quién está realizando la acción o quién recibe el valor de la actividad.
* Actividad: representa la acción que realizará el sistema.
* Valor de negocio que recibo: comunica porque es necesaria la actividad.

Ejemplo:

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Lo de arriba se llama Frase Verbal, es un resumen corto de la descripción de la user story.

### ¿Qué es una User Story?

Las user stories son multipropósito.

Las historias son:

* Una necesidad del usuario.
* Una descripción del usuario.
* Un ítem de planificación.
* Token para una conversación.
* Mecanismo para diferir una conversación.

### El Product Owner prioriza las historias en el producto backlog:

Interfaz de usuario gráfica

Descripción generada automáticamente

### Modelado de roles: tarjeta de rol de usuario:

Texto

Descripción generada automáticamente

#### Usuarios representantes (proxies):

Tipos de usuarios representantes:

* Gerentes de usuarios.
* Gerentes de desarrollo.
* Alguien del grupo de marketing.
* Vendedores.
* Expertos del dominio.
* Clientes.
* Capacitadores y personal de soporte.

No son ideales como los usuarios verdaderos, hay que evitarlos.

### Criterios de aceptación de historias de usuario:

* Define límites para usa user story (US).
* Ayuda a que los PO respondan lo que necesitan para que la US provea valor (requerimientos funcionales mínimos).
* Ayudan a que el equipo tenga una visión compartida de la US.
* Ayudan a desarrolladores y testers a derivar las pruebas.
* Ayudan a los desarrolladores a saber cuándo parar de agregar funcionalidad en una US.

Ejemplo:

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

#### ¿Cuáles son los criterios de aceptación buenos?

* Definir una intención, no una solución. Ejemplo: el usuario debe elegir al menos una cuenta para operar.
* Son independientes de la implementación.
* Relativamente de alto nivel, no es necesario que se escriba cada detalle.

#### ¿Los detalles dónde van?

Ejemplo de detalles:

* El encabezado de la columna se nombra “Saldo”.
* El formato del saldo es 999.999.999,99.
* Debería usarse una lista desplegable en lugar de un check box.

Estos detalles que son el resultado de las conversaciones con el PO y el equipo puede capturarlos en dos lugares:

* Documentación interna de los equipos.
* Pruebas de aceptación automatizadas.

### Pruebas de aceptación de historias de usuario:

* Expresan detalles resultantes de la conversación.
* Complementan la user story.
* Se escriben: “Probar… (pasa/no pasa).”
* Es un proceso de dos pasos:

1. Identificarlas al dorso de la US.
2. Diseñar las pruebas completas.

Ejemplos:

Imagen que contiene Texto

Descripción generada automáticamente

Texto, Tabla

Descripción generada automáticamente

### Invest model:

* Independent: calendarizables e implementables en cualquier orden.
* Negotiable: se expresa en términos de “que” no el “cómo”.
* Valuable: debe tener valor para el cliente.
* Estimatable: para ayudar al cliente a armar un ranking basado en costos.
* Small: debe ser “consumidas” en una iteración.
* Testable: demostrar que fueron implementadas.

### Algo más sobre las User Stories:

* No son especificaciones detalladas de requerimientos (como los casos de uso).
* Son expresiones de intención, “es necesario que haga algo como esto…”.
* No están detallados al principio del proyecto, elaborados evitando especificaciones anticipadas, demoras en el desarrollo, inventario de requerimientos y una definición limitada de la solución.
* Necesita poco o nulo mantenimiento y puede descartarse después de la implementación.
* Junto con el código, sirven de entrada a la documentación que desarrolla incrementalmente después.

### Diferentes niveles de abstracción:

Imagen que contiene Gráfico

Descripción generada automáticamente

#### Spikes:

* Tipo especial de historia, utilizado para quitar riesgo e incertidumbre de una User Story u otra faceta del proyecto.
* Se clasifican en: técnicas y funcionales.
* Pueden utilizarse para:
  + Inversión básica para familiarizar al equipo con una nueva tecnología o dominio.
  + Analizar un comportamiento de una historia compleja y poder así dividirla en piezas manejables.
  + Ganar confianza frente a riesgos tecnológicos, investigando o prototipando para ganar confianza.
  + Frente a riesgos funcionales, donde no está claro como el sistema debe resolver la interacción con el usuario para alcanzar el beneficio esperado.

**Técnicas:**

* Utilizadas para investigar enfoques técnicos en el dominio de la solución.
  + Evaluar performance potencial.
  + Decisión hacer o comprar.
  + Evaluar la implementación de cierta tecnología.
* Cualquier situación en la que el equipo necesite una comprensión más fiable antes de comprometerse a una nueva funcionalidad en un tiempo fijo.

**Funcionales:**

* Utilizadas cuando hay cierta incertidumbre respecto de cómo el usuario interactuará con el sistema.
* Usualmente son mejor evaluadas con prototipos para obtener realimentación de los usuarios o involucrados.

Algunas User Stories requieren de ambos tipos de spikes. Por ejemplo:

* Como un cliente, quiero ver mi uso diario de energía en un histograma, para poder comprender rápidamente mi consumo de energía pasado, presente y proyectado.

En este caso un equipo puede crear dos spikes:

* Spike técnico:
  + Investigar cuanto tiempo requiere actualizar un display de un cliente al uso actual, determinando requerimientos de comunicación, ancho de banda y si los datos se actualizan en formato push o pull.
* Spike funcional:
  + Crear un prototipo de histograma en el portal web y obtener la retroalimentación de algunos usuarios respecto del tamaño, el estilo de la presentación y los atributos gráficos.

##### Lineamientos para spikes:

* Estimables, demostrables, y aceptables.
* La excepción, no la regla:
  + Toda historia tiene incertidumbre y riesgos.
  + El objetivo del equipo es aprender a aceptar y resolver cierta incertidumbre en cada iteración.
  + Los spikes deben dejarse para incógnitas más críticas y grandes.
  + Utilizar spikes como última opción.
* Implementar la spike en una iteración separada de las historias resultantes:
  + Salvo que el spike sea pequeño y sencillo y sea probable encontrar una solución rápida en cuyo caso, spike e historia pueden incluirse en la misma iteración.

### Algunas cosas para dejar en claro:

* Diferir el análisis detallado tan tarde como sea posible, lo que es justo antes de que el trabajo comience.
* Hasta entonces, se capturan requerimientos en la forma de “user stories”.
* Las user stories no son requerimientos de software, no necesitan ser descripciones exhaustivas de la funcionalidad del sistema.

### Tips para que las user stories sean útiles para el equipo:

* Un paso a la vez, evitar la “Y”.
* Usar palabras claras en los criterios de aceptación.
* No olvidarse la parte invisible: la conversación.
* Las user stories se escriben desde la perspectiva del usuario.
* No forzar todo para escribirlo como user stories.