**Ingeniería de software 1**

**Practico**

Alumno: Santiago Vietto

Docente: Julio Emanuel Castillo

DNI: 42654882

Institución: UCC

Año: 2021

**Los procesos de software**

Objetivo de la materia

\_ El objetivo principal de la materia es presentar al estudiante los métodos fundamentales para el desarrollo de software en forma disciplinada y precisa. Para ello se inicia con la descripción general del proceso de desarrollo de software analizando los diferentes modelos y sus casos típicos de aplicación. Se analiza todo lo relativo a la gestión de procesos en general y a la gestión particular de procesos de desarrollo de software poniendo énfasis en los estándares de calidad. Se introduce al estudiante en la gestión y liderazgo de proyectos de desarrollo de software.

Ingeniería de software

\_ Decimos que la ingeniería de software es una disciplina de ingeniería que se interesa por todos los aspectos de la producción de software, desde las primeras etapas de la especificación del sistema hasta el mantenimiento del sistema después de que se pone en operación. En esta definición se presentan dos frases clave:

Disciplina de ingeniería: los ingenieros hacen que las cosas funcionen. Aplican teorías, métodos y herramientas donde es adecuado. Sin embargo, los usan de manera selectiva y siempre tratan de encontrar soluciones a problemas, incluso cuando no hay teorías ni métodos aplicables. Entonces, tenemos un problema, planteamos una solución usando las diferentes herramientas viendo las diferentes etapas.

Todos los aspectos de la producción del software: la ingeniería de software no sólo se interesa por los procesos técnicos del desarrollo de software (hacer el código), sino también incluye actividades como la administración del proyecto de software y el desarrollo de herramientas, así como métodos y teorías para apoyar la producción de software. La ingeniería tiene una visión mas grande, cubre mas que solo saber un lenguaje de programación o hacer un diagrama que explique el sistema, sino que también, sino que también como se va a ejecutar, cuales son los tiempos, cuando se va a entregar y como va a ser después el mantenimiento una vez que sea entregado.

\_ La ingeniería de software es un concepto, metodología y disciplina que utiliza herramientas para poder crear el software, donde tiene un análisis de procesos, gestión de proyectos, una planificación, etc. No está centrada en un tipo de software, sino que es para cualquier software que nosotros tengamos.

**Ejercicio 1**

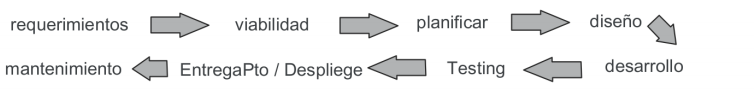
1)\_ ¿Qué es la ingeniería de software?

* Conceptos y metodologías que se aplican en las distintas etapas de un ciclo de desarrollo de un software.
* Es una disciplina que utiliza estructuras, herramientas y técnicas informáticas para la creación de sistemas informáticos programados.
* Método de programación que ayuda a que todo sea mas universal, lo que ayuda a los programadores a entenderse de una mejor forma generando así una mayor organización. Creación de software a través de una planificación o método conceptualmente establecido para llevar a cabo este proceso de una forma eficiente. Estudio sobre las mejores técnicas, modelos y diseños para implementar y planificar el desarrollo de software. Consiste en la metodología y comprensión de los softwares y programas.
* Minimizar errores
* Análisis de procesos, minimizar errores, gestión de proyectos, técnicas y herramientas para desarrollar software. Desarrollo y mantenimiento de sistemas de software, buenas practicas de desarrollo, uso de tecnologías y prácticas de las ciencias de la computación, ingeniería, etc. Es el estudio de los principios y metodologías para el desarrollo y mantenimiento de sistemas de software.

2)\_ ¿Para qué creen que sirve la ingeniería de software?

* Para establecer un flujo de elaboración de un software de manera de que la colaboración no sea interrumpida por ninguna parte que se encuentre involucrada en el proyecto. Para realizar un software de calidad, que cumpla con las necesidades del cliente y sea de fácil mantenimiento. Desarrollar software de excelencia.
* Sirve para que todo sea más sencillo tanto como para usar y programar. Sirve para el desarrollo de software, el cual lleva a cabo un proceso que se divide en etapas. Ser mas eficiente y tener las respuestas adecuadas frente al diseño e implementación del software.
* Sirve para coordinar a los integrantes que forman parte del equipo (incluyendo clientes). Organización de equipos de software. Mejorar la calidad de los productos de software. Resolución de problemas relacionados al software. Aplicar prácticamente el conocimiento científico al diseño y construcción de programas de computadora. Optimizar procesos del desarrollo de software. Estimar tareas, costos y plazos de ejecución de proyectos. Establecer los principios y métodos de la ingeniería a fin obtener software de modo rentable, que sea fiable y trabaje en máquinas reales.
* Poder coordinar el desarrollo de softwares difíciles. Crear programas informáticos que satisfagan las necesidades de las personas. Poder diseñar y desarrollar programas informáticos.

3)\_ Pasos del desarrollo de software:



Desarrollo del software

\_ A continuación, vamos a denotar paso a paso las etapas para un desarrollo de software adecuado, teniendo en cuenta las respuestas del ejercicio 2, entonces:

Requerimientos: algo que siempre vamos a tener son los requerimientos, y esto no solamente es para ingeniería de software, sino que también es para cualquier cosa que queramos hacer, por ejemplo, si queremos hacer una mesa primero tenemos que averiguar que mesa queremos hacer, como la vamos a hacer, cuales van a ser las medidas, donde la vamos poner y sobre todo para que queremos hacer la mesa, entonces con el software es lo mismo y viene del lado del proyecto. Si uno no sabe lo que hace, es muy difícil que el cliente nos valide los requerimientos. Y cuando uno escribe un requerimiento es importante tener en cuenta lo que se hace y lo que no se hace.

Viabilidad: realizamos un estudio de viabilidad, por el hecho de que se pueda o no realizar el proyecto, por ejemplo, verificamos que el cliente gane dinero y nosotros también, es decir, un análisis financiero, un análisis de utilidad, entre otros.

Planificación: una vez que tenemos los requerimientos, planificamos como vamos a hacer el proyecto, en que fecha, que vamos a hacer primero, como y quien lo va a hacer. Esta etapa es crucial en la ingeniería de software y también para cualquier proyecto que hagamos.

Diseño: el diseño puede estar dentro del mundo de la planificación, pero podemos considerarlo como un proceso más.

Desarrollo: una vez finalizada la planificación y el diseño, largamos con el desarrollo del proyecto, donde apretamos tecla y lo codeamos.

Testing: realizamos un testeo del producto desarrollado para solucionar errores y chequear que este todo en orden.

Despliegue/Entrega: una vez que funciona correctamente realizamos el despliegue del mismo, que básicamente consiste en la entrega del producto al cliente.

Mantenimiento: luego de entregar el producto sigue el mantenimiento. El mantenimiento es algo previo a las cosas que hagamos y es una parte importante dentro del desarrollo de software y la ingeniería, y generalmente siempre vamos a tener mantenimiento independientemente de lo que hagamos ya que justamente es como se va a mantener el sistema. Lo puede hacer uno mismo, el cliente o un tercero.

Conclusión

\_ El software se puede hacer de muchas maneras (Modelos de Proceso de Software) los cuales se ajustan mejor a los contextos o necesidades en que surge el mismo. Estos procesos o “maneras”, son algunos más estrictos que otros, pero todos tienen sus etapas. Sobre los roles o personajes dentro del proceso de software, tenemos varios (algunos muy puntuales según el tipo de proceso o metodología), pero los que siempre están son:

* Cliente
* Equipo desarrollo (desarrollador, diseñador, tester, product owner, arquitectos)

\_ Por otro lado uno cuando toma un proyecto tiene las 4 P de la ingeniería de software, que son:

* Personal: sin las personas no se puede ejecutar el proceso o desarrollar algo.
* Producto: es lo que vamos a desarrollar, esto es algo chico.
* Proceso: es como vamos a desarrollar el producto, mediante las etapas que vimos (requerimiento, viabilidad, etc).
* Proyecto: es lo que envuelve al producto, es decir, incluye mas que solamente hacer el producto.

Proceso de software

\_ Un proceso es un conjunto estructurado de actividades necesarias para desarrollar un sistema de software. Muchos de los procesos de software son diferentes, pero todos implican: especificación, diseño, desarrollo, validación, evolución. Es decir, todos los desarrollos de software más allá del proceso que elijamos necesitamos saber que queremos hacer, tenemos que hacer un diseño, saber como lo vamos a hacer, tenemos que desarrollarlo y después lo tenemos que validar. En la etapa de especificación nos planteamos que queremos hacer el software de tal manera, y en la validación lo que se hace es comparar lo que desarrollamos contra la especificación, y la evolución hace referencia al mantenimiento de lo que es el proceso o como va evolucionando.

Especificación: la definición de lo que el sistema debe hacer, así como también lo que no debe hacer ya que muchas veces los clientes asumen cosas que nosotros no lo dejamos explicito.

Diseño e implementación: la definición de la organización del sistema y la implementación del sistema.

Validación: la comprobación de que hace lo que quiere el cliente.

Evolución: el cambio del sistema en respuesta a las necesidades cambiantes de los clientes.

Tipos de proceso

\_ En el día a día hay situaciones en donde el software que queremos hacer se amolda mejor a algo ágil y se amolda mejora a algo mas estructurado, habiendo casos en los que se combinan, no son contradictorios, sino que a veces alguno se amolda mejor a las condiciones. El desarrollo de software depende muchos del contexto de lo que queremos hacer y de las condiciones.

Por plan: este es un tipo más estructurado, dividiéndose en etapas, donde hay que cumplir una para pasar a la siguiente. Mas allá de que lo hagamos por reutilización o incremental, tenemos un plan y una fecha establecida.

Ágil: van por pequeños objetivos y permite mostrarle los cambios o avances más cercanos al cliente. Vamos haciendo las cosas sobre la marcha y se lo vamos mostrando al cliente. No tiene una fecha final calculada, sino que esa última fecha no se sabe ya que depende de la velocidad, del equipo, que tanto se avance y otros factores.

Modelo de proceso de software

Modelo de cascada: este es un modelo dirigido por plan, tiene fases separadas y distintas, de especificación y desarrollo. Generalmente suele ser más estructurado, es decir, necesitamos terminar una fase para recién avanzar a la otra. Este modelo se suele utilizar en software que necesitan demasiada precisión o que necesitamos que funcione bien desde el principio y no tantas iteraciones. Con este modelo nos aseguramos que cuando pasamos de una etapa a la otra, estamos seguros de que se terminó y testeamos todo lo que teníamos que hacer. Además, en cascada con los requerimientos que tenemos al principio son los mismos que manejamos durante todo el proyecto.

Desarrollo incremental: la especificación, desarrollo y validación se intercalan. Puede ser el dirigido por plan o ágil. Este modelo es algo que lo vamos a ir haciendo con iteraciones y le vamos a ir mostrando al cliente los cambios que el va queriendo, y no le damos todo entero como el modelo en cascada. Entonces vamos haciendo el software en pedazos y le damos un incremento.

Ingeniería de software orientado a reutilización: el sistema se ensambla a partir de componentes existentes Puede ser el dirigido por plan o ágil. Esto básicamente consiste en que, si ya tenemos una porción de software que hace tal cosa, por ejemplo, un login, no hace falta que lo volvamos a desarrollar, sino que lo podemos reutilizar para el nuevo desarrollo de software. Un ejemplo básico de esto es cuando tenemos librerías, cuando consumimos algún repositorio, algún código que ya soluciona alguna cosa, entonces la idea es reutilizar y no tener que volver a desarrollar. El uso de esto es muy común sobre todo en la programación, a menos que necesitáramos hacer algo puntual, y en ese caso deberíamos desarrollarlo de nuevo. Tenemos que diferenciar la reutilización con tomar cosas que ya están hechas y así ensamblarlas, sino que la reutilización es de módulos que nos permitan evitar tener que codear.

\_ Los requerimientos influyen mucho hasta inclusive en como vamos a encarar las cosas. Utilizamos el modelo en cascada cuando tenemos algo en lo que necesitamos cierta estructura, cuando es algo riesgoso como, por ejemplo, tenemos el tren de aterrizaje, el control de un motor, el panel de control de un avión, es decir, cosas en las que no podemos fallar cuando salen a producción o tenemos que tratar de fallar lo menos posible, y donde sabemos que los requerimientos no van a fallar. Cuando tenemos muchos cambios el desarrollo incremental sirve un montón, porque se adapta más, además lo vamos haciendo de a pasos o iteraciones, y se lo mostramos al cliente de una manera más frecuente no como en cascada donde se lo mostramos al final.

**Ejercicio 2:**

\_ Para los siguientes ejemplos definir qué Modelo de proceso podrían aplicar y su justificación:

1)\_ Software que controla la bomba de combustible de un motor: quiero un software que controle la bomba de combustible conocimiento su presión, cantidad de combustible y que regule el mismo dependiendo del andar del motor.

\_ Se concluyo que el mejor modelo es CASCADA, porque los requerimientos son complejos y no cambiantes, y además necesitamos seguridad entre cada etapa, ya que pone en riesgo vidas. La bomba de combustible es un proceso delicado y si algo sale mal, va a andar mal el motor pudiendo generar un accidente, entonces con la cascada nos aseguramos de que cuando pasamos de una etapa a la otra estamos seguros de que la anterior se hizo correctamente. Por otro lado, es muy raro que la bomba cambie los requerimientos, entonces por eso nos sirve cascada ya que son los mismos requerimientos del inicio a fin.

2)\_ Software de control de mesas de un bar: quiero hacer un programa que me permita controlar las mesas de un bar pudiendo ver las personas que tenemos en las mesas, que pidieron y el costo total de la consumición. EL software me debe permitir agregar y sacar consumiciones (productos de mi bar) y también manejar roles por mozo (cuando tengamos cambio de turno).

\_ Se concluyo que el mejor modelo es INCREMENTAL, porque los requerimientos son cambiantes, por ejemplo, a futuro, y podemos ir mostrando de a poco, es decir, tener un rápido despliegue. También vemos que puede ser de REUTILIZACION para casos de login, es decir, un módulo de login (esto hace referencia cuando habla de manejar roles), donde usamos algún modulo o librería que ya existe.

3)\_ Software que controla el tren de aterrizaje de un avión: el software deberá poder subir y bajar el tren de aterrizaje, emitiendo alertas en el caso de que tenga algún problema. Mostrándolo en el panel de control del avión.

\_ Al igual que el ejercicio 1, se concluyó que el mejor modelo es CASCADA, porque los requerimientos son complejos y no cambiantes, y además necesitamos seguridad entre cada etapa, ya que pone en riesgo vidas. Necesitamos que salga bien desde cero, no es bueno que tengamos un error, sino que cada paso que hagamos tiene que ser pisando sobre seguro.

4)\_ Página Web para control de una playa de estacionamiento: el software deberá ser una página donde pueda ver toda la playa, también deberá poder calcular el costo de la estadía de un cliente, teniendo en cuenta su entrada y salida (hora). También se deberá dar un indicador en tiempo real de cuanto lugar tengo libre en el estacionamiento. También quiero obtener métricas mensuales de rendimiento y sectores que más se usan de mi playa.

\_ Se concluyo que el mejor modelo es un NO CASCADA, porque los requerimientos son cambiantes y podemos ir mostrando de a poco los resultados. Surgió un enfoque INCREMENTAL, ya que lo podemos hacer de forma iterativa, pero también vemos que puede ser de REUTILIZACIÓN para los casos como login, donde usamos algún modulo o librería que ya existe, ya que seguramente una persona que está usando una playa se tiene que loguear.

5)\_ App de empresa New Smart TV que se instala en un Smart TV: esta App te permitirá ver contenido en línea y a demanda, poder agregar contenido a mis favoritos y tendré la posibilidad de bookmarks, para no volver al principio de una peli. Quiero poder ver películas, series, cualquier contenido que sea un video.

\_ Se concluyo que el mejor modelo es un NO CASCADA, porque los requerimientos son cambiantes y podemos ir mostrando de a poco los resultados. Surgió un enfoque INCREMENTAL pero también vemos que puede ser de REUTILIZACIÓN dependiendo del contexto, por ejemplo, si hago la App es para todos los TV o solo de algunas empresas en particular. A veces vamos a poder reutilizar una cosa puntual o mas cosas dependiendo del contexto o los requerimientos, y a prior conviene hacerlo incremental.

6)\_ Software de control de un brazo robótico: este software permitirá la comunicación entre los sensores del brazo y las partes mecánicas.

\_ Se concluyo que el mejor modelo es CASCADA, por el que los requerimientos son complejos y no cambiantes. Vimos que el contexto necesita que el brazo salga funcionando bien, con los menos errores posibles, por eso creemos que cascada se adapta mejor.

7)\_ Software para controlar el antibloqueo de frenos en un automóvil: este software permitirá controlar la intensidad en que frenamos, según condiciones como la velocidad, la dirección del auto.

\_ Se concluyo que el mejor modelo es CASCADA, porque los requerimientos son complejos y no cambiantes, y además necesitamos seguridad entre cada etapa, ya que pone en riesgo vidas.

8)\_ Software de aula virtual para estudiantes: actualmente ya tenemos un software así, pero queremos modernizarlo. Queremos que se pueda ingresar y autenticar como alumno de la facultad y ver sus materias. Donde verá el programa y las clases.

\_ Se concluyo que el mejor modelo es un NO CASCADA, porque los requerimientos son cambiantes y podemos ir mostrando de a poco los resultados. Surgió un enfoque INCREMENTAL pero también vemos que puede ser de REUTILIZACIÓN dependiendo de cómo se interpretaba el enunciado (vimos lo importante de los requerimientos dentro del desarrollo de software). Es importante tener en cuenta los requerimientos.

9)\_ Software interactivo de programación de viajes que ayude a los usuarios a planear viajes: este software me mostrará las opciones de transporte para el lugar y fecha donde quiero ir. También me dirá si puedo ir en avión o en por tierra. Me mostrará los costos y tarifas extras (impuestos) como el total a pagar. Me permitirá pagar de manera online en la página también.

\_ Se concluyo que el mejor modelo es INCREMENTAL porque los requerimientos son cambiantes y podemos ir mostrando de a poco. Y podemos usar REUTILIZACION por el módulo de login o algún módulo de pago. Casi siempre vamos a reutilizar alguna cosa, y la reutilización no quiere decir que no vamos a codear, sino que lo que quiere decir es que tenemos requerimientos, pero a algunos los vamos a ver solucionados por algún modulo externo.

10)\_ Software que controla un ascensor: este software permitirá controlar el ascensor mediante un panel, donde seleccionaremos el piso y nos llevará al mismo. Tendrá la inteligencia para organizarse según el piso. Ejemplo, si muchas personas van al piso 7, las agrupa a todas en un ascensor y usamos solo ese. El software debe poder manejar un grupo de ascensores.

\_ Se concluyo que el mejor modelo es CASCADA, porque los requerimientos son complejos y no cambiantes, y además necesitamos seguridad entre cada etapa, ya que pone en riesgo vidas.

**Desarrollo ágil**

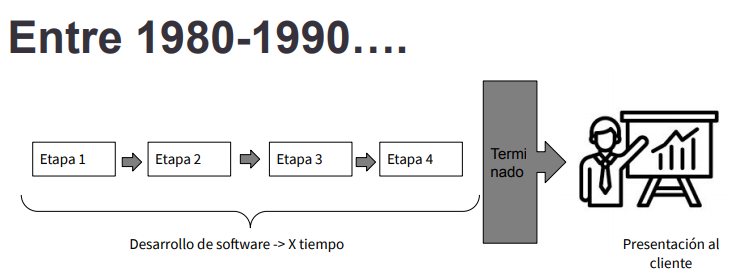
Desarrollo rápido de software

\_ El rápido desarrollo y la entrega son ahora los requisitos más importantes para los sistemas de software, es decir, cuando se creó el software o se empezaron a hacer los desarrollos primero era todo un poco más estático, donde muchas cosas se aplicaban en cascada porque se necesitaba que el sistema sea rígido, hacía falta validarlo mucho, corrían riesgo muchas veces las vidas de las personas, pero después empezó a haber software para cosas menos complejas, es decir, hubo situaciones, en donde necesitamos software, que no son tan complejas. Entonces frente a esa necesidad el software se tuvo que adaptar, ya que ahora no necesitamos tanta documentación y tanto proceso (una etapa después de la otra), sino de que ahora tenemos un desarrollo rápido y una entrega inmediata, y este empezó a se lo mas importante en el desarrollo de software, ya que:

* Los requisitos cambian rápidamente, donde ahora los requisitos van cambiando muy rápidamente por lo que necesitamos un proceso que también se adapte a eso.
* El Software tiene que evolucionar rápido, debido a lo anterior.

\_ Gracias a todo esto surge el desarrollo rápido de software, en donde:

* Especificación, diseño e implementación están intercalados. Como vimos antes, estas estaban separadas, y ahora se empiezan a mezclar en un proceso más rápido y ágil.
* El sistema está desarrollado como una serie de versiones con las partes interesadas involucradas en la evaluación de versiones. Antes en el modelo en cascada teníamos que la entrega del producto era al final entonces lo que entregábamos era casi el producto terminado, pero ahora con el desarrollo ágil tenemos muchas versiones porque tenemos muchas entregas, le mostramos más rápido al cliente por ende tenemos un feedback más temprano, cambios más rápidos y generamos versiones.
* Las interfaces del usuario se desarrollan a menudo utilizando un IDE y herramientas gráficas. Ahora nosotros podemos mostrar las interfaces de usuario a través de un IDE sin la necesidad de codear, esto también nos ayuda a ver los cambios más rápidamente.

\_ Viendo el grafico que tenemos a continuación este nos dice que entre 1980 y 1990 se desarrollaban muchas cosas en cascada, por eso las N etapas, y llevaban un X tiempo, donde una vez que se terminaba lo que se hacía era presentárselo al cliente, pero nunca en el medio, y el cliente cuando se lo mostrábamos recién empezaba justamente a ver el producto, en donde si a este no le gustaba algo y pedía un cambio, la desventaja de un modelo en cascada frente a los cambios era que se tenía que hacer toda la iteración de nuevo, donde lo que nos llevó X tiempo ahora nos va a volver a llevar otro X tiempo más porque al cambio lo vimos al final y no en el medio, por lo que el cambio es igual a mayor dinero y mayor tiempo. Entonces es por este problema por lo que empezaron a surgir las metodologías agiles.

Métodos agiles

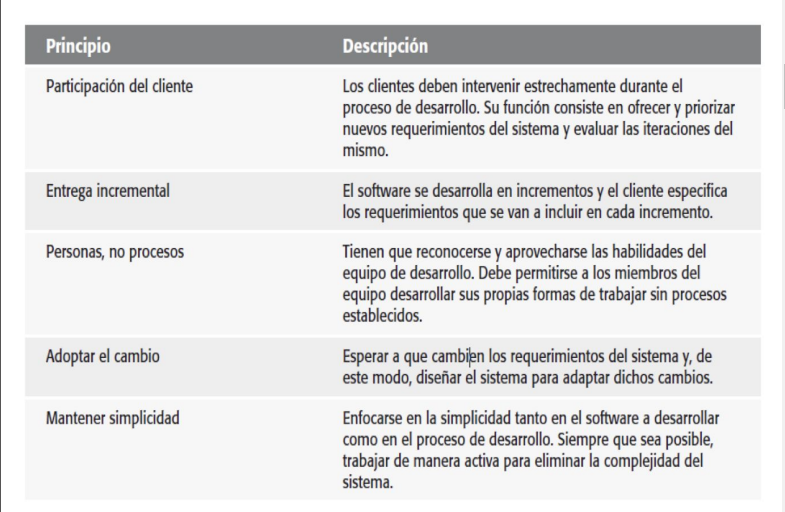
\_ El objetivo de los métodos agiles es reducir los gastos generales del proceso de software(por ejemplo, limitando la documentación) y ser capaces de responder rápidamente a las necesidades cambiantes sin rehacer trabajo de manera excesiva, porque cuando nosotros realizamos el software, si el cliente piensa una cosa y nosotros pensemos otra, es algo totalmente natural y puede pasar porque somos personas y tenemos percepciones totalmente distintas, por eso es que es importante especificar bien los requerimientos (que hace y que no hace el sistema), para que de esa forma podamos evitar los cambios y saber cómo manejarnos frente a esos cambios. De ahí es cuando surge todo lo que son las metodologías o manifestó ágil.

Manifestó ágil

\_ Estamos descubriendo mejores formas para desarrollar software, al hacerlo y al ayudar a otros a hacerlo. Gracias a este trabajo llegamos a valorar:

* A los individuos y las interacciones sobre los procesos y las herramientas.
* Al software operativo sobre la documentación exhaustiva.
* La colaboración con el cliente sobre la negociación del contrato.
* La respuesta al cambio sobre el seguimiento de un plan.

\_ Todo esto quiere decir que ahora nos acercamos más al cliente, no lo vemos como una persona lejana al final del proyecto, sino que lo hacemos parte del proyecto, le mostramos el software operativo funcionando (página web, sistema de login, sistema de autogestión, etc), para que nos pueda decir si es lo que esperaba o no, donde ese feedback temprano y continuo es lo que marca la diferencia en el agilismo.



\_ Para agregar, lo de adoptar el cambio es crucial, y no solamente para metodologías agiles sino también para cualquier contexto que tengamos. Siempre van a surgir cosas, siempre vamos a tener opiniones distintas, siempre vamos a creer tal cosa que después el cliente espera otra. Entonces las metodologías agiles también nos enseñan a abrazar el cambio y no verlo como algo no tan catastrófico. El cambio es algo que se puede dar en el proceso de software, y nosotros tenemos que saber entenderlo y manejarlo y no tratar de esquivarlo o temerle, ya que es algo totalmente natural, aunque si bien no podemos tener un software que cambie siempre ya que para eso tenemos los requerimientos y dejamos establecido el alcance, porque si no vamos a estar en un proceso donde nunca se va a terminar el software ya que siempre lo vamos a ver con otros ojos. Entonces una cosa es abrazar el cambio y la otra es estar cambiando constantemente.

Problemas con métodos ágiles

\_ No todo es fantástico con el agilismo porque este tiene tanto ventajas como desventajas, y los principales problemas son:

* Puede ser difícil mantener el interés de los clientes que están involucrados en el proceso. Por ahí el cliente no se quiere meter en el proceso, delegándonos todo el desarrollo. Traerlo al cliente al proceso es un cambio de paradigma.
* Los miembros del equipo pueden ser inadecuados para la intensa participación que caracteriza a los métodos ágiles. En el proceso ágil tenemos mucha participación del equipo, porque el equipo es fundamental y es el que marca que es lo que vamos a tomar y como lo vamos a tomar, y es el que realmente termina desarrollando. Entonces esto también implica un cambio de paradigma en el equipo, el hecho de hacerse mas participe y mas owner del proyecto y no solamente ejecutor.
* Priorizar cambios puede ser difícil donde hay múltiples partes interesadas. Esto tiene que ver con el agilismo, es decir, nosotros tenemos cambios constantes por lo que tenemos que elegir como los vamos a hacer. Negociar siempre es importante.
* Mantener simplicidad requiere un trabajo extra. Hablamos de mantener simplicidad porque parte del agilismo es no enroscarse y hacer las cosas simples, como entregar algo y después irlo iterando.
* Los contratos pueden ser un problema, porque cuando hacemos software planteado con el agilismo por ahí no hacemos bien el alcance y sobre todo con los tiempos los contratos son difíciles de establecer.

Plan vs Ágil

\_ El desarrollo guiado por plan:

* Identifica etapas separadas en el proceso de software con salidas asociadas a cada etapa.
* Las salidas de una etapa se usan como base para planear la siguiente actividad del proceso.
* Las iteraciones se producen dentro de las actividades.

\_ Y en el desarrollo ágil:

* Especificación, diseño, implementación y pruebas están intercalados.

\_ La mayoría de los proyectos de software incluyen prácticas de los enfoques ágil y basado en un plan. Por ejemplo, si decimos que, en el desarrollo de una calculadora, el proceso va a tardar tanto tiempo, a eso lo hacemos basándonos en un plan, pero que después en la implementación utilicemos técnicas agiles como una planning, sprints, daily, retros, desarrollo incremental, entre otros, a eso lo manejamos nosotros, pero al cliente le vamos a decir una fecha basada en un plan. Entonces en el día a día se combinan las dos, técnicas agiles con planificación, donde utilizamos lo mejor de las dos cosas para llegar al objetivo.

Agilismo

Extreme Programming (XP): se utiliza principalmente para proyectos de desarrollo de software, siendo uno de los muchos tipos de metodologías agiles. Con esta, vamos construyendo un producto muy ajustado a los requerimientos del cliente. También se logra la programación de a pares, donde esta busca tener diferentes putos de vista y este es el objetivo.

Scrum: su atención se centra en la gestión del desarrollo iterativo en lugar de prácticas ágiles específicas. Es decir, tenemos un framework o flujo que se basa en sprints. Hay tres fases en Scrum:

* La fase inicial es una fase en fase de anteproyecto, donde se establecen los objetivos generales del proyecto y se diseña la arquitectura de software.
* Esto es seguido por una serie de ciclos (sprints), donde cada ciclo desarrolla un incremento del sistema. Cada vez que nosotros terminamos un ciclo en scrum, tiene que tener un incremento de producto, ya que ese es nuestro objetivo, nuestro conjunto de tareas a tomar tiene que tener como finalidad eso, es decir, sumarle un incremento a nuestro sistema, sea chico o grande es un incremento ya que cada vez que terminemos un sprint no podemos quedar en la misma situación que como estábamos al inicio, sino que tenemos que tener un progreso.
* La fase de cierre del proyecto concluye el proyecto, completa documentación requerida, tales como marcos de ayuda del sistema y manuales de usuario y evalúa las lecciones aprendidas del proyecto.

\_ Ahora analizando el grafico, tenemos el siguiente proceso, para empezar tenemos la planeación del bosquejo y diseño arquitectónico, donde seguido de este tenemos la valoración donde hacemos el análisis, luego seleccionamos a lo que vamos a hacer, lo desarrollamos y lo revisamos, toda esta parte pertenece al ciclo sprint que se repite cada un cierto tiempo en donde tenemos una lista de los requerimientos y por cada ciclo tomamos un pedazo y lo ejecutamos, y así sucesivamente en otros ciclos, y para saber cuándo va a terminar se utilizan diferentes técnicas como por ejemplo estimar el backlog, calcular la velocidad de nuestro equipo y de ahí sacar cuantos sprints nos llevaría para luego transformarlo en tiempo, y con esto se complejiza el hecho de tener que dar una fecha final cuando se trabaja con el agilismo.

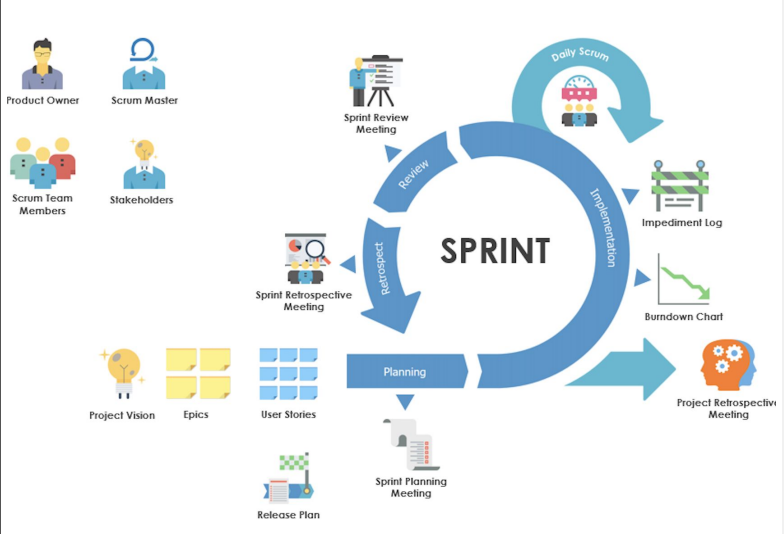


\_ En scrum tenemos:

* Los Sprints que pueden durar entre 1-4 semanas, donde uno es muy poco porque quizás con el equipo de desarrollo no desarrollamos mucho en una semana, y cuatro es un montón porque el agilismo consiste en hacer algo y mostrarlo al momento y no a un mes después ya que de esta forma se pierde el feedback y el cambio constante cuando hacemos un sprint de cuatro semanas, entonces generalmente se manejan entre 2 a 3 semanas.
* Ceremonias, que pueden ser por ejemplo el planning, daily, sprint review, retrospective.
* Roles, como, por ejemplo:
  + Scrum master: es la persona que vela por la ejecución de la metodología de scrum que tengamos. Este lo que hace es velar por el marco de scrum, es decir, por las retros que tengamos, que tengamos las daily, si estamos bloqueados como se puede resolver, que tengamos la planning, entre otras cosas mas.
  + Scrum team: es justamente el equipo que hace el desarrollo.
  + Product owner: es nuestro principal contacto que tenemos con los stakeholders. Esta es la persona encargada de saber cuáles son nuestros requerimientos, además es quien ayuda a la planificación y la priorización del backlog y es nuestra cara con el cliente. Es el que prioriza como tenemos que hacer las cosas, además es quien ayuda a seleccionar y es nuestra cara con el cliente. También ayuda en la formación de las historias de usuario, donde es la persona que mas conocimiento tiene sobre el producto a nivel de negocio.
  + Stakeholders: son justamente como la representación de nuestro cliente, es decir, personas que están directamente vinculadas con ellos.

\_ El nombre scrum se basa en la posición del rugby en la que el equipo está unido, se apoyan y van juntos para el mismo lado.

\_ A continuación vemos una imagen del framework, donde cómo podemos observar empieza con la planning, después tenemos la implementación, la review y la retroalimentación. Este framework de scrum se basa en un aprendizaje constante, también basado en la mejora continua o retroalimentación constante. Podemos ver también los roles y los requerimientos principales del sistema (project visión), que son transformados en épicas y luego en historias de usuario, y esta es la forma en la que nosotros contamos los requerimientos, pero desde un punto de vista más del usuario.



\_ En la visión tenemos los objetivos, lo más abstractos, donde después hacemos las épicas y después los objetivos. Entonces el ciclo empieza con una planning donde se revisa el backlog grande y el sprint backlog, en esta planning participan el product owner, el scrum master y el scrum team, donde se definen entre todos que es lo que vamos a hacer en el sprint basado en las prioridades que nos va a decir el product owner. Una vez que tenemos el sprint backlog, empezamos a codear, y aparece el burndown, es un grafico descendente que marca como vamos consumiendo nuestras tares y cuantas tareas nos quedan, la métrica que nos indica cuantos puntos o story points consumimos por sprint se llama la velocity del equipo, es decir, cada equipo de scrum tiene una velocidad que lo caracteriza y con esta logramos obtener o proyectar la fecha de finalización. Luego tenemos la daily scrum que lo que hace es ir contando como estamos durante el día, también tenemos la review en la que se muestra el incremento, y después la retro, este es un espacio de crecimiento del equipo donde se analizan que cosas salieron bien, que tenemos para mejorar y se sacan accionables para evitar o mejorar las cosas el próximo sprint. A cada user story se le agrega un puntaje que se lo conoce como story point (basados en la complejidad), entonces en la planning lo que se hace es estimar las historias de usuario y se las carga en el backlog.

Users tory

\_ Estas son la forma en que representamos los requerimientos del sistema desde una visión de usuario. Cuando iniciamos un software, lo primero que tenemos que hacer es dejar bien establecido que es lo que queremos hacer y lo que no. Entonces, como usuario queremos hacer tal cosa para obtener tal beneficio. En las user story debemos tener en cuenta lo siguiente:

* **Como** <quien> **Quiero** <que> **Para** <objetivo>

\_ Por ejemplo, como Vendedor, quiero registrar los productos y cantidades que me solicita un cliente para crear un pedido de venta. Como vemos esto lo que esconde es una funcionalidad, que en este caso el sistema tiene que poder registrar producto y cantidades y tiene que poder crear un producto de venta.

\_ La historia de usuario no solamente queda en plasmar el requerimiento, sino que también entramos en una validación. Cuando nosotros desglosamos los requerimientos y los transformamos en historias de usuario, nosotros tenemos que aceptar esa historia de usuario, y para eso en el agilismo tenemos lo que se llaman criterios de aceptación, es decir, como nosotros a esa historia de usuario la marcamos como terminada o resuelta, entonces para validar una historia de usuario lo hacemos con los criterios de aceptación, donde en estos se especifica qué salidas obtendremos cuando finalice el proceso de ejecución de la funcionalidad, y nos sirve para verificar que está terminada la funcionalidad. Está relacionada con las pruebas que se realizarán para verificar el cumplimiento de la expectativa de diseño, usabilidad, rendimiento, y la satisfacción del usuario.

\_ Los criterios de aceptación es muy común que después se terminen utilizando como casos de uso de testeo, ya que cuando uno se empieza a desarrollar y se basa mucho en lo que es historias de usuario, podemos ver que los criterios de aceptación se utilizan como casos de uso de testeo para validar esas funcionalidades, pueden ser test automáticos o manuales con una persona que los valide, pero estos criterios de aceptación muchas veces se trasladan a casos de uso de testeo.

Criterios de aceptación

\_ Generalmente ayudan al equipo de desarrollo a validar que:

* Se ha construido el producto correcto.
* Se ha construido el producto correctamente.

\_ Los criterios de aceptación deben describir siempre un contexto, un evento y la respuesta o consecuencia esperada del sistema. La forma más utilizada para describir los criterios de aceptación es conocida como Given-When-Then. Aquí un ejemplo:

* **Dado** un usuario que aún no se ha identificado en el sistema.
* **Cuando** intenta acceder a alguna funcionalidad de la parte privada.
* **Entonces** se le redirige automáticamente a la página de login para que pueda identificarse.

\_ Entonces los criterios de aceptación son dado un contexto, cuando queremos hacer algo y entonces pasa tal cosa. Aunque describen comportamiento de la aplicación, se utiliza siempre un lenguaje de negocio, no técnico. A continuación, tenemos un ejemplo:

User story:

**COMO** usuario **QUIERO** acceder a la agenda de enfermería de forma segura **PARA** usar sus funcionalidades.

Acceptance criterio:

* **Dado** un usuario que quiere loguearse en nuestro sistema, **cuando** ingresa su nombre de usuario y contraseña y estos son correctos, **entonces** el usuario puede ingresar exitosamente a la Home.
* **Dado** un usuario que quiere loguearse en nuestro sistema, **cuando** ingresa el nombre de usuario DEBE tener forma de email, en caso contrario se mostrará el mensaje de error pertinente.
* **Dado** un usuario que quiere loguearse en nuestro sistema, no ingresa nada en los campos usuario y contraseña, **entonces** el sistema muestra un mensaje de error indicando que debe ingresar esos campos obligatorios.

\_ Entonces, la historia de usuario marca una funcionalidad o se desprende de una funcionalidad, y los criterios de aceptación nos dicen que para obtener esa funcionalidad tenemos que hacer tal cosa, es decir, son las condiciones para nosotros asegurar esa funcionalidad. Podemos tener varios criterios de aceptación para una misma historia de usuario, y lo importante es definir que tantos criterios de aceptación sumamos para esa historia. Podemos tener varias historias de usuario y lo podemos hacer iterativo, es decir, tenemos una historia y hacemos cinco criterios, lo mismo con la siguiente y así para no hacer muchos criterios en una sola historia.

\_ En el siguiente grafico del user story tenemos el tema (theme) o el objetivo principal del cual sacamos las épicas, luego las historias de usuario y de estas las tareas.



**Ejercicio 3:**

\_ Nos dividimos en grupos, donde cada uno deberá hacer user stories para los siguientes casos. A continuación, tenemos descripciones porque es como seria en la vida real, entonces desde un párrafo tenemos que sacar los requerimientos y trasladarlos a historias de usuario con sus criterios de aceptación.

1)\_ Sistema de alerta de tsunami: un sistema de **monitoreo** y **alerta** de **tsunami** posee un centro de mando que **recibe información** de lectura de 500 **boyas** antena flotantes. Cada boya posee una antena que se comunica vía satélite con el centro de mando, y **envía** la **información** relacionada a su **posición geoespacial**, la **presión atmosférica**, **velocidad del viento, temperatura superficial** y **humedad**. A su vez, **la boya unifica** la **información** proveniente **de 20 celdas subacuáticas** que se encuentran depositadas a una distancia no mayor a 5 km. Dichas celdas miden la presión del agua y la velocidad de rotación de las corrientes y envían la información a su boya madre. Las celdas pueden además comprobar el estado de su batería, para poder alertar cuando la vida útil de una batería se está agotando. Los profesionales a cargo pueden consultar el centro de mando en cualquier momento, obteniendo un mapa marítimo con distintos niveles de alerta. El sistema a su vez tiene la capacidad de procesar las lecturas recibidas y pronosticar con un modelo matemático, la posibilidad de que exista un tsunami. Cuando la probabilidad es superior al 50%, el sistema automáticamente envía una alerta a las principales agencias de los gobiernos afectados.

\_ Historias de usuario:

* **Como** usuario, **quiero** consultar el estado de la batería de las celdas **para** poder cambiar la batería cuando se esté agotando.
* **Como** usuario **quiero** que el sistema unifique la información proveniente de celdas subacuáticas **para** generar un reporte matemático.
* **Como** usuario **quiero** que el sistema genere un reporte matemático **para** mantenernos alertados
* **Como** usuario **quiero** que el sistema procese las lecturas recibidas **para** generar un pronóstico de ocurrencia de un tsunami.
* **Como** usuario **quiero** que el sistema me muestre un mapa marítimo **para** poder revisar los niveles de alerta.
* **Como** usuario, **quiero** consultar el centro de mando **para** obtener un mapa marítimo.
* **Como** agencia de un gobierno afectado, **quiero** recibir una alerta de tsunami **para** pronosticar la posibilidad de que exista un tsunami.

\_ Por ahí las historias de usuario están más definidas como alguien que utiliza el sistema y no como alguien que forma parte, por ejemplo, en este caso la boya es como que forma parte del sistema proveyendo datos, y las historias de usuario están más enfocadas a los profesionales que consultan y monitorean, es decir, a las personas que usan los datos y no a los elementos que forman parte del sistema.

\_ Las preguntas sobre un ejercicio son importantes porque en el desarrollo de software estas determinan el alcance, es decir, que es lo que vamos a hacer. Casi siempre hay que consultarle al cliente.

\_ Las historias de usuario tienden a ser cortas, ya que la idea es tratar de que sean atómicas porque lo que van a marcar son pequeñas funcionalidades. Las historias no tienden a ser grandes y súper complejas, es decir, no tenemos que verlo como un sistema grande sino empezar a desglosarlo y hacer cosas más chicas. As historias de usuario tienen como objetivo partir los requerimientos, tienen que ser atómicas y sencillas, y si utilizamos sprints, las historias de usuario se tienen que poder hacer en dos semanas. Por otro lado, tenemos que tener cuidado de pensar el sistema como una sola historia de usuario (sistema entero), por ende, tenemos que fraccionarlo, es decir, transformar lo implícito en explicito. Tienen que ser lo suficientemente atómicas para que entren en un sprint, y si se pasa hay que revisarla.

2)\_ Plataforma de capacitación online: los profesionales que desean capacitarse en áreas específicas, pueden ingresar a una plataforma de capacitación online que ofrece una gran cantidad de cursos clasificados por temas de interés. Para poder utilizar la plataforma, la persona debe poseer un usuario registrado. Un usuario registrado posee datos personales tales como nombre, correo electrónico, empresa en la que trabaja y además se le permite asociar un CV. El CV consta de varios ítems que pueden ser personalizados para reflejar información de estudios, experiencia laboral y logros alcanzados. El usuario puede buscar un curso en particular, navegando el catálogo de cursos por tema de interés. Cada curso posee un identificador único, nombre del curso y descripción de sus contenidos. Además, cada curso posee una clasificación de dificultad (básico, intermedio, avanzado) que le permite a los usuarios evaluar cual es el curso más adecuado a sus necesidades. Cuando el usuario selecciona un curso del catálogo, puede agregarlo a su lista de cursos favoritos o bien optar por comenzar el cursado. En tal caso, el sistema evalúa la dificultad del curso. Si se trata de un curso intermedio o avanzado y el usuario no ha tomado el curso de dificultad anterior, el sistema rechaza la solicitud y no le permite ingresar al curso, indicándole que debe cumplimentar un curso de inferior dificultad antes de tomar el de mayor complejidad. Cuando el usuario toma y finaliza un curso, el sistema emite un comprobante de cursado. El usuario puede imprimir el certificado, enviarlo por correo electrónico y/o agregarlo a su CV digital.

\_ Usuario = profesional que desea capacitarse.

\_ Historias de usuario:

* **Como** usuario, **quiero** ingresar a la plataforma online **para** consultar sobre los distintos cursos que el sistema (academia, institución, etc) ofrece.
* **Como** usuario, **quiero** que el sistema clasifique a los cursos por temas de interés **para** facilitar el encuentro de los mismos.
* **Como** usuario **quiero** registrar mis datos personales **para** poder utilizar la plataforma.
* **Como** usuario **quiero** asociar un CV a mi perfil/cuenta **para** reflejar la información de mis estudios, experiencia laboral y logros alcanzados.
* **Como** usuario **quiero** que el sistema me permita buscar un curso en particular **para** encontrar el que más se amolde a mis necesidades.
* **Como** usuario, **quiero** que el sistema identifique a los cursos mediante nombre y descripción **para** obtener la información del mismo.
* **Como** usuario, **quiero** que el sistema clasifique los distintos cursos por dificultad (b, i, a) **para** evaluar cuál se adecue más a mi necesidad.
* **Como** usuario **quiero** que si el curso tiene una dificultad avanzada el sistema lo nivele con un curso inferior **para** mantener una organización.
* **Como** usuario, **quiero** que el sistema me permita “darle fav” a un curso **para** agregarlo a mi lista de favoritos.
* **Como** usuario, **quiero** que el sistema me permita realizar una inscripción a un curso **para** comenzar su cursado.
* **Como** usuario, **quiero** que el sistema emita un comprobante de cursado **para** utilizarlo como certificado de reconocimiento de aprendizaje.
* **Como** usuario, **quiero** que el sistema me permita imprimir un certificado **para** contar con la documentación física que avale mi cursado.
* **Como** usuario, **quiero** que el sistema me permita enviar el certificado por correo electrónico **para** contar con una copia digital.
* **Como** usuario, **quiero** que el sistema me permita agregar el certificado a mi CV virtual **para** demostrar mi aprendizaje.

\_ Cuando marcamos en el texto o consigna palabras claves, es una técnica que nos ayuda a identificar casos de uso o actores, tenemos en cuenta que las palabras que marcamos son sustantivos o verbos que nos ayudan a nosotros a entender como son los requerimientos, quienes son los actores y cuales son las acciones, y después a nivel de código, esta técnica nos ayuda a después a hacer los diagramas de clases, de dominio, de uso, etc.

\_ Es importante también agregar los criterios de aceptación ya que estos nos ayudan a validar las historias de usuario. En la vida real, a muchos de los criterios de aceptación, los da el cliente, ya que a todos los sacamos de entrevistas que tenemos con el cliente. Los criterios de aceptación ayudan a definir la historia.

Repaso desarrollo rápido de software

\_ El rápido desarrollo y la entrega son ahora los requisitos más importantes para los sistemas de software. En estos:

* Los requisitos cambian rápidamente.
* El Software tiene que evolucionar rápido.

\_ Además en el desarrollo rápido de software tenemos que:

* Especificación, diseño e implementación están intercalados.
* El sistema está desarrollado como una serie de versiones con las partes interesadas involucradas en la evaluación de versiones. La idea del desarrollo rápido es hacerlo a través de versiones y no hacer todo el producto entero.
* Las interfaces del usuario se desarrollan a menudo utilizando un IDE y herramientas gráficas.

Scrum: aplicar 100% scrum puro es muy complicado de hacer, con todas las ceremonias, con todas las capacitaciones que tenemos que tener sobre el equipo, entonces lo que generalmente se hace es que se trata de utilizar opciones agiles, es decir, dinámicas agiles como, por ejemplo, tenemos retro pero no la tenemos a la finalización del sprint, sino que la tenemos cada dos sprints, pero aun la tenemos. Entonces lo que pasa es que nos marcamos estrictamente en lo que es scrum o utilizamos las técnicas agiles, y generalmente lo que pasa es que terminamos utilizando las técnicas agiles, algunos hacen daily, otros planning, retro, y eso se va adaptando según la empresa y según el quipo, pero lo bueno es que usemos las herramientas que nos dé scrum.

Backlog: el Product Backlog o pila de producto en un proyecto que sigue la metodología Scrum consiste en una lista con todos los requerimientos iniciales del producto que se va a desarrollar. Se trata de una lista dinámica, que irá evolucionando a medida que lo hace el producto y el entorno del proyecto.

Kanban: es un pizarrón que se puede utilizar para planning, retros, dinámicas de roles, y todo esto lo hacemos con mural ya que es una aplicación muy potente. Kanban es una forma de organizar el trabajo, y además es una herramienta que se utiliza para los proyectos agiles, es decir, es una forma rápida en la que vemos todo lo que hay que hacer, que es lo que estamos haciendo y que es lo que ya terminamos. Es un pantallazo general que nos permite organizar el trabajo de una buena manera y de forma consistente.

\_ Dado el ejercicio 4 que fue en Mural sacamos las siguientes conclusiones:

* Lo que salió bien: presentación, divertido el scrum master, se quemaron todas las tareas del sprint, buena distribución de tareas, se pudo hacer todo a tiempo.
* A mejorar: falta de comunicación entre integrantes del equipo y falta de trabajo.
* Accionable/compromiso: continuar mejorando la comunicación entre los pares y capacitarse en la herramienta para un mejor uso.

**Ingeniería de requerimientos**

Ingeniería de requerimientos

\_ La ingeniería de requerimientos es una parte super importante en la ingeniería de software porque es lo que nos define el alcance, si nosotros a esto no lo hacemos de la mejor manera podemos después tener fricciones con el cliente a la hora de los cambios y el proceso como así también el proyecto se pueden ver afectados.

\_ El proceso de establecimiento de los servicios que el cliente necesita de un sistema y las limitaciones con las que opera y se desarrolla. Esta ultima parte es crucial porque tenemos que describir lo que hace y no hace en el alcance, por eso habla de limitaciones, ya que los clientes suelen asumir implícitamente cosas que nosotros no, por eso es importante pasar de lo implícito a lo explicito.

\_ Los requisitos, en sí, son las descripciones de los servicios del sistema y las limitaciones que se generan durante el proceso de ingeniería de requerimientos.

Requerimientos

\_ La definición de un requerimiento puede variar de una declaración abstracta de un servicio o de una restricción de sistema a una especificación funcional matemática detallada. La forma en la que nosotros escribimos los requerimientos puede variar (lenguaje natural o lenguaje estructurado) pero lo que si es importante es que nuestro requerimiento no tiene que ser ambiguo, es decir, tiene que ser claro y lo mejor detallado posible.

Tipos de requerimientos

Requerimientos funcionales: enunciados acerca de los servicios del sistema que debe proporcionar, como el sistema debe reaccionar a entradas generales y cómo el sistema debe comportarse en situaciones particulares. Pueden explicar lo que el sistema no debe hacer. Relacionados justamente con la funcionalidad. Por ejemplo: necesito loguearme y que autentifique contra una base de datos. Estos requerimientos están muy vinculados a las cosas que tiene que hacer el sistema para poder saciar la necesidad por la cual fue creado el sistema.

* Requerimiento del usuario: deben ser escritos en lenguaje natural, más los diagramas de los servicios que proporciona el sistema y sus limitaciones operacionales. Escrito para que los clientes entiendan. Orientados a que queremos hacer como usuarios. Por ejemplo: Yo como usuario quiero poder loguearme y poder acceder para ver el catálogo.
* Requerimiento del sistema: un documento estructurado que establece las descripciones detalladas de las funciones del sistema, los servicios y las limitaciones operativas. Define todo lo que debe ser implementado así que puede ser parte de un documento entre el cliente y el desarrollador. Se desprenden de los de usuario y son más a detalle, y en este caso entra el desarrollador que no entra en el de usuario, acá hacemos más detalle. Por ejemplo: El sistema tienen que lograr autenticarse contra una base de datos que esta en la nube, y que lo pueda devolver al catalogo y si soy un usuario valido o no.

Requerimientos no funcionales: limitaciones en los servicios o funciones que ofrece el sistema, como restricciones de tiempo, restricciones del proceso de desarrollo, normas, etc. A menudo se aplica al sistema en su conjunto, en lugar de a las funciones o servicios individuales. Tienen que ver con el contexto en si de donde va a estar la aplicación o el servicio que va a prestar. Por ejemplo: no quiero que el login tarde mas de 5 minutos en autenticarse y devolver una respuesta.

* Requerimientos del producto: requerimientos que especifican que el producto entregado debe comportarse de una manera particular, por ejemplo, velocidad de ejecución, tiempos de ejecución, disponibilidad, fiabilidad, etc.
* Requerimientos organizacionales: requerimientos que son consecuencia de las políticas y procedimientos de la organización, por ejemplo, estándares de procesos utilizados, los requisitos de implementación, etc. Tiene que ver con cosas culturales de la empresa.
* Requerimientos externos: requerimientos que surgen de factores que son externos al sistema y su proceso de desarrollo, por ejemplo, requisitos de interoperabilidad, los requisitos legislativos, etc. Tiene que ver con normas que nos regulen.

Requerimientos de dominio: las restricciones en el sistema segun el dominio de operación.

Ejemplos de requerimientos no funcionales

\_ Requerimientos del producto:

* El MHC-PMS estará a disposición de todas las clínicas durante las horas normales de trabajo (lun-vie, 08:30 a 17:30). El tiempo de inactividad dentro de las horas normales de trabajo no excederá de cinco segundos en cualquiera de un día.

\_ Requerimiento organizacional:

* Los usuarios del sistema MHC-PMS deberán autenticarse usando su tarjeta de identidad autoridad sanitaria.

\_ Requisito externo:

* El sistema deberá aplicar las disposiciones de privacidad del paciente según lo establecido en HStan-03-2006-priv.

**Ejercicio 4:**

\_ Nos dividimos en grupos, cada grupo clasificar los siguientes requerimientos en Funcionales y No Funcionales, con sus correspondientes Sub-Clasificaciones. Justificando la respuesta.

1)\_ El sistema debe ser capaz de procesar N transacciones por segundo.

\_ **Requerimiento** **no funcional** porque no hace referencia a la funcionalidad que resuelve la necesidad sino a cosas a que el sistema tiene que responder (contexto) en donde se especifica la velocidad, **requerimiento de producto**.

2)\_ El sistema enviará un correo electrónico cuando se registre alguna de las siguientes transacciones: pedido de venta de cliente, despacho de mercancía al cliente, emisión de factura a cliente y registro de pago de cliente.

\_ **Requerimiento funcional** hace referencia a la funcionalidad del sistema que debe satisfacer una necesidad, **requerimiento de usuario** porque está escrito en lenguaje natural.

3)\_ Toda funcionalidad debe responder al usuario en menos de 3 segundos.

\_ **Requerimiento** **no funcional** porque no hace referencia a la funcionalidad que resuelve la necesidad sino a cosas a que el sistema tiene que responder (contexto) en donde se especifica el tiempo y es una limitación, **requerimiento de producto**.

4)\_ El sistema permitirá aprobar, cambiar o actualizar planes y cronogramas de proyecto.

\_ **Requerimiento funcional** hace referencia a la funcionalidad del sistema que debe satisfacer una necesidad, **requerimiento de usuario** porque está escrito en lenguaje natural.

5)\_ El sistema permitirá el envío automatizado de cartas de entrega de órdenes directamente al almacén.

\_ **Requerimiento funcional** hace referencia a la funcionalidad del sistema que debe satisfacer una necesidad, **requerimiento de usuario** porque está escrito en lenguaje natural.

6)\_ El sistema debe contar con un sistema de soporte en línea.

\_ **Requerimiento** **funcional**, es un requerimiento de origen no funcional ya que no tiene que ver en si con la funcionalidad para saciar la necesidad, pero nos termina generando un requerimiento funcional porque lo tenemos que desarrollar, **requerimiento de usuario**.

7)\_ El sistema de Historia Clínica Electrónica debe cumplir con la legislación vigente relativa a la protección de datos médicos.

\_ **Requerimiento** **no funcional**, **requerimiento externo** ya que tiene que ver con una normativa externa.

8)\_ El sistema debe cumplir con la accesibilidad para personas con discapacidad.

\_ **Requerimiento** **funcional**, es un requerimiento de origen no funcional ya que no tiene que ver en si con la funcionalidad para saciar la necesidad, pero nos termina generando un requerimiento funcional porque lo tenemos que desarrollar, **requerimiento de usuario**.

9)\_ Todas las comunicaciones entre servidores, aplicaciones y clientes del sistema deben ser encriptadas utilizando el algoritmo RSA.

\_ **Requerimiento** **no funcional**, **requerimiento externo** ya que el algoritmo tiene que asegurar la comunicación entre ambas partes y generalmente viene impuesto desde afuera y no maneja la organización.

10)\_ La tasa de errores cometidos por el usuario deberá ser menor al 5% de las transacciones totales ejecutadas por el sistema.

\_ **Requerimiento** **no funcional** porque no hace referencia a la funcionalidad que resuelve la necesidad sino a cosas a que el sistema tiene que responder (contexto) y tenemos una métrica, **requerimiento de producto**.

11)\_ El software debe poder emitir los siguientes estados financieros: Balance general, Estado de ganancias y pérdidas, Estado de flujos de efectivo. Además, debe poder emitir un listado de mayor general y mayor analítico.

\_ **Requerimiento funcional** porque nos dice lo que tiene que hacer el sistema, es decir, forma parte de la funcionalidad, **requerimiento del usuario**.

12)\_ El sistema debe poder operar de forma adecuada con 10.000 usuarios en sesiones concurrentes.

\_ **Requerimiento** **no funcional** porque no hace referencia a la funcionalidad que resuelve la necesidad sino a cosas a que el sistema tiene que responder (contexto) y tenemos una métrica, **requerimiento de producto**.

13)\_ Se debe permitir el registro de pedidos de compra con datos obligatorios incompletos, los cuales podrán completarse posteriormente modificando el pedido. Antes de poder aprobarse los datos del pedido deben estar completos.

\_ **Requerimiento funcional**, ya que justamente hace referencia a aquellas actividades específicas o servicios que debe realizar el sistema ante datos de entrada, es decir, se especifica lo que el sistema debe hacer, **requerimiento del usuario**.

14)\_ La aplicación web debe poseer un diseño “Responsive” para garantizar la adecuada visualización en todos los dispositivos (PC, Tablet, Smartphone).

\_ **Requerimiento no funcional** y **requerimiento de producto**, ya que hace referencia a una funcionalidad que debe cumplir el producto para casos específicos, como en este caso que la app web sea responsive, por lo que se especifica que el producto debe comportarse de una forma particular.

15. El sistema debe tener una disponibilidad del 99,99% de las veces en que un usuario intente acceder.

\_ **Requerimiento no funcional** y **requerimiento de producto**. Este es un requerimiento no funcional y de producto ya que especifica mediante una medida métrica (en este caso, 99,99% de las veces) una propiedad que el sistema debe cumplir.

16. El Motor de Base de Datos será Oracle 11g.

\_ **Requerimiento no funcional** y **requerimiento organizacional**, decimos que este requerimiento es no funcional y organizacional ya que suponemos que es una restricción que nos la ha impuesto el cliente, que puede llegar a ser una empresa con una disponibilidad o previa implementación de dicho motor de bases de datos

17. Es necesario contar con una replicación constante de los datos en un servidor distinto al que usa (idealmente en un edificio distinto).

\_ **Requerimiento no funcional** y **requerimiento organizacional**, decimos que este requerimiento es no funcional y organizacional ya que suponemos que es una restricción que nos la ha impuesto el cliente, que posee distintos edificios con la capacidad de replicar una versión constante de los datos.

18. El uso del sistema debe ser intuitivo. El tiempo de aprendizaje para el uso del sistema de un usuario promedio debe ser menor a 30 minutos.

\_ **Requerimiento no funcional** y **requerimiento del producto**. El requerimiento es no funcional ya que se define una propiedad del sistema que debe lograr al cumplir el requisito de aprendizaje promedio en cierto tiempo (métrica). Es un requerimiento del producto ya que garantiza el funcionamiento y facilidad de uso al producto final entregado. Si dijese solamente intuitivo es funcional con origen no funcional.

19. El sistema debe proporcionar mensajes por cada transacción realizada (error, éxito, descargando…). Los mensajes deben ser informativos y orientados al usuario final.

\_ **Requerimiento funcional** y **requerimiento de usuario**. Decimos que este es un requisito funcional y de usuario ya que describe una funcionalidad que el sistema debe cumplir de una forma breve en la que el cliente la puede entender sin complicaciones

20. El tiempo para iniciar o reiniciar el sistema no podrá ser mayor a 2 horas, avisando previamente de esta maniobra.

\_ **Requerimiento no funcional** y **requerimiento del producto**. Decimos que este es un requerimiento no funcional y de producto ya que especifica una propiedad que el sistema debe cumplir (en este caso el tiempo máximo de reinicio) mediante el uso de una métrica (en este caso horas

**Preguntas:**

1)\_ ¿Qué implica el desarrollo rápido de software?

\_ Los requerimientos cambian y el software debe evolucionar.

2)\_ ¿Cuál es el objetivo de la metodología ágil?

\_ Responder rápidamente a las necesidades cambiantes, es decir adaptarse a los cambios y no solamente hacer las cosas rápido.

3)\_ ¿Cuál es la afirmación que forma parte del manifiesto ágil?

\_ Al software funcionando sobre la documentación exhaustiva, es decir, el manifiesto ágil nos dice que no prioricemos la documentación excesiva, sino que prioricemos un software funcional.

4)\_ ¿Cuáles son algunos de los problemas de la metodología ágil?

* Cuesta entender cuando termina el proyecto.
* Priorizar cambios puede ser difícil donde hay múltiples partes interesadas.
* Necesitamos capacitación para el equipo.

5)\_ Actores que correspondan al marco Scrum:

* PO
* Equipo de trabajo

\_ No participan líder, tester ni Scrum Líder (no es lo mismo que Scrum Master).

6)\_ ¿Como se escribe una user story?

\_ Como <quien> quiero <que> para <objetivo>.

6)\_ ¿Para qué se usan los criterios de información?

\_ Validar que se cumpla una User Story.

7)\_ ¿Cuáles son las divisiones dentro de un tablero Kanban?

\_ ToDo, WiP, Done.

Documento de requerimiento

\_ El documento de requisitos de software es la declaración oficial de lo que se requiere de los desarrolladores del sistema, es decir, este documento nos dice como va a funcionar el sistema sea a través de requerimientos funcionales como no funcionales. Debe incluir tanto una definición de los requisitos del usuario y una especificación de los requisitos del sistema. NO es un documento de diseño. En la medida de lo posible, debería establecer de lo QUE el sistema debe hacer y NO COMO es que debe hacerlo.

**Usabilidad**

Usabilidad

\_ Facilidad con que las personas pueden utilizar una herramienta particular u otro objeto fabricado por humanos con el fin de alcanzar un objetivo concreto. Dentro de la usabilidad tenemos:

* Efectividad
* Eficiencia
* Satisfacción

\_ Tanto efectividad y eficiencia están directamente relacionados al objetivo que tiene el sistema, en que, si lo hace bien o no, y después la satisfacción es importante porque esta parte es del cliente. Antes cuando se empezaban a hacer los sistemas no le daban mucha importancia a la visión del usuario, es decir, como se siente, como la maneja, si le es cómodo o no, sino de que se hacia el sistema y se adaptaba el usuario, mientras que ahora eso cambio, se piensa mas en el usuario y se adapta el sistema. Están importante la experiencia del usuario que moviliza cambios.

\_ La usabilidad se mide:

Empírica: porque no se basa en opiniones o sensaciones, sino en pruebas de usabilidad realizadas en laboratorio u observadas mediante trabajo de campo.

Relativa: porque el resultado no es ni bueno ni malo, sino que depende de las metas planteadas.

\_ Para medir la usabilidad se hacen diferentes encuestas para ver el grado de satisfacción o no de la aplicación, un ejemplo son los ratings que les ponemos a las aplicaciones nativas, donde lo ideal es tener todas las aplicaciones con 5 estrellas y si una tiene 1 se trata de mejorarla.

\_ Entonces la usabilidad es muy importante porque nos da cierto posicionamiento en el mercado.

Principios de la usabilidad

Facilidad de aprendizaje: hace referencia a que no nos cueste como utilizar el sistema o la aplicación.

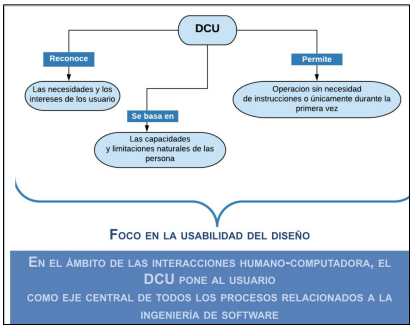
Facilidad de uso: esta tiene que ver con el aprendizaje, que no sea muy complejo poder usar la aplicación y que sea intuitiva.

Flexibilidad: relativa a la posibilidad de poder cambiar la información, de que no sea algo muy estructurado, sino que se pueda ordenar y cambiar la información, y también tiene que ver con el dialogo y poder realizar varias tareas

Robustez: es el nivel de apoyo al usuario que facilite el cumplimiento de sus objetivos (ayudas, carteles, bot de consultas, etc).

Diseño centrado en el usuario

\_ Antes de salir a la producción tenemos que tener nuestro sistema centrado en el usuario. Le preguntamos al usuario que le parece lo que estamos desarrollando.



**Gestión de proyectos**

Gestión de proyectos

\_ La gestión de proyectos es algo que cubre varias cosas, no solamente el desarrollo de software. Una cosa es desarrollar nosotros el producto y otra cosa es gestionar el proyecto. Siempre gestionar el proyecto es mas grande que gestionar el producto. Gestionar el producto es hacer justamente en nuestro caso el software, por ejemplo, hacer una pagina web, pero gestionar el proyecto es mas grande, en donde tenemos que ver los interesados, los riesgos, el alcance, el acta de constitución, es decir, tenemos cosas que van más allá de desarrollar el producto.

¿Qué es un proyecto?

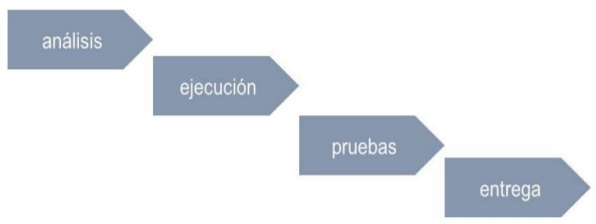
\_ Podemos decir que un proyecto es un conjunto de etapas, actividades y tareas, un esfuerzo, para alcanzar un objetivo que implica un trabajo no inmediato, a un plazo relativamente largo. Lo importante de un proyecto es que surge de una necesidad o de un caso de negocio, todos nuestros proyectos surgen porque queremos resolver algo.

* Los proyectos implican un principio y un final, es decir, nosotros iniciamos el mismo y lo terminamos en algún momento, nunca queda colgado en el tiempo.
* Utiliza diversos recursos finitos y cuenta con un presupuesto, no tenemos un presupuesto infinito que dure para siempre y eso por eso que tiene un inicio y fin.
* Sus actividades son únicas y esencialmente no repetitivas.
* Tiene un objetivo.
* Requiere un jefe de proyecto y personal de desarrollo, cuyos roles y estructura de equipo deben definirse y desarrollarse, es decir, una persona que lidere y que tenga una visión de ese proyecto, también tenemos personas que hagan el desarrollo cuyo roles y estructuras de equipo deben definirse y desarrollarse.
* Tiene que planificarse, tenemos que idear como vamos a hacerlo, que vamos a hacer primero, que vamos a hacer después y sobre todo cuando va a empezar y cuando va terminar, después eso puede cambiar, pero tiene que tener la planificación.
* Debe medir su progreso frente al plan.
* Suele coexistir con otros proyectos y competir por los recursos, esto suele pasar en grandes empresas cuando tenemos proyectos en paralelo.
* Existen fuerzas internas y externas, que deben identificarse y tratarse, que influyen en él. Estas fuerzas tienen que ver un poco con lo que son los riesgos, donde algunos tienen mas o menos probabilidades de surgir y los tenemos que tener en cuenta y los tenemos que clasificar.

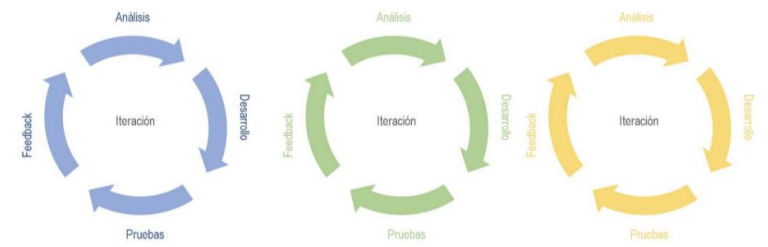
Ciclo de vida de un proyecto

\_ El ciclo de vida se refiere a cómo va transaccionando el proyecto, es decir, se refiere a la serie de fases que atraviesa un proyecto desde su inicio hasta su conclusión. Los ciclos de vida pueden ser:

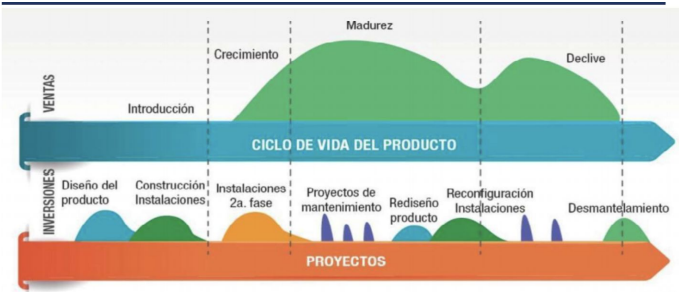
Predictivo: se determina el alcance, el costo y el tiempo al inicio del proyecto. Y después se hace el ciclo de vida en cascada. Una vez que termina una actividad pasamos a la otra



Adaptativo: ciclos agiles, iterativos e incrementales. Las estaciones de costos, tiempo y alcance se van ajustando a lo largo de ciclos (iteraciones) de trabajo. Incremento sucesivo de las funcionalidades del producto. Estas se relacionan con el agilismo, es decir, en un proceso circular que se va repitiendo cada determinado tiempo, apareciendo conceptos como scrum, etc. Ciclos agiles, donde por ejemplo tenemos un recurso inicial y se va adaptando a medida que avanzamos en el proyecto.



\_ Por ahí nosotros confundimos que el producto es igual que el proyecto. El producto suele se mas chico que el proyecto, y el proyecto tiene otras etapas que el producto por ahí no. A continuación, vemos la diferencia de etapas de ambos:



Fases de un proyecto

\_ Una fase del proyecto es un conjunto de actividades del proyecto, relacionadas de manera lógica, que culmina con la finalización de uno o más entregables. Un ejemplo de fase puede ser la fase del acta de constitución en donde tenemos el caso de uso de negocio y empezamos a definir el acta, y tiene como salida el acta de constitución.

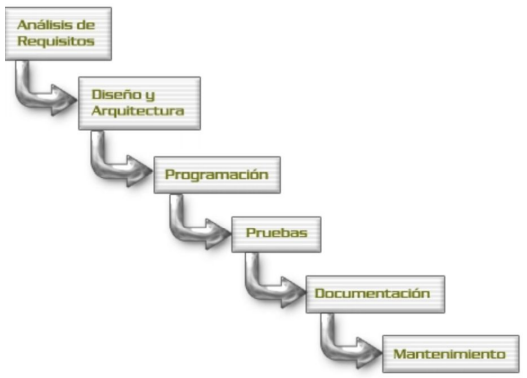
\_ Al finalizar cada fase del proyecto se encuentra un punto de revisión, en el cual se comparan el avance y el desempeño general del proyecto contra la planificación. Cada fase tiene entradas y salidas, generalmente las salidas de una fase anterior son las entradas para la fase siguiente.

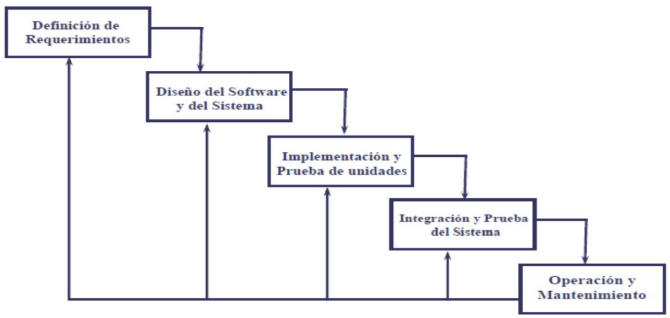
\_ El proyecto tiene estas fases comunes, pero nosotros podemos agruparlas según nuestro criterio:



¿Y en un proyecto de software?

\_ Hacemos referencia al modelo en cascada:





PMBook

\_ Es una referencia fundamental para cualquiera que esté interesado en la profesión de la Dirección de Proyectos, es decir, es algo general y no particular del software, e imprescindible para la preparación del examen de certificación PMP, los alumnos del Master en Project Management están en condiciones de presentarse al examen de certificación PMP-PMI. Lo que hace es dar los lineamientos para dirigir un proyecto, donde hay cosas que las podemos usar y otras no, pero nos da un pantallazo general.

\_ Incorpora prácticas tradicionales en Project Management generalmente empleadas en la Dirección de Proyectos agiles, también incorpora nuevas prácticas en Dirección de Proyectos después de haber comprobado su utilidad en los Proyectos y su uso por parte de los Project Manager.

\_ El PMBOK es aplicable en la mayoría de los Proyecto, estas buenas prácticas de Dirección de Proyectos recogidas en el PMBoK son aplicables a la mayoría de los Proyectos y la mayor parte del tiempo, existe un consenso entre los Gerentes de Proyecto, a nivel internacional, acerca de su valor y utilidad cuando se aplica a los Proyectos, de ahí su enorme éxito internacional.



