En el siguiente documento profundizaremos en los contenidos relacionados con el estudio de las metodologías de desarrollo de software y sus aplicaciones tanto en el ámbito empresarial, como personal y móvil.

Contenidos Unidad 1

Metodologías de Desarrollo de Software

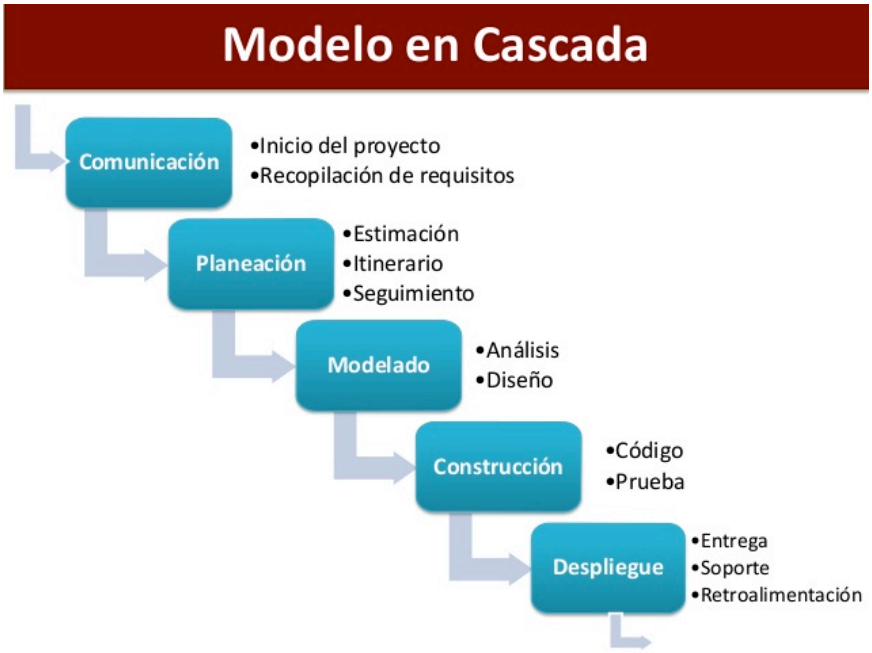
Contenidos Unidad 1

TEMA 1: "Definición de metodología de desarrollo de software"

Una metodología de desarrollo de software es un marco de trabajo (framework) para las tareas que se requieren en la construcción de software de alta calidad. La metodología de desarrollo de software puede estar basado en uno o varios *modelos de desarrollo de software*, los que se usarán para estructurar, planear y controlar el proceso de desarrollo de software.

- **EJEMPLIFICACIÓN**

En la siguiente figura se muestra un modelo clásico de desarrollo de software llamado modelo en cascada. Una metodología de desarrollo de software basada en este modelo debe seguir las etapas descritas en dicho modelo.



Como se desprende de la figura anterior, una etapa de desarrollo debe completarse antes de dar comienzo a la siguiente. Por ejemplo, cuando todos los requerimientos del cliente han sido identificados, analizados para comprobar su integridad y consistencia, y documentados en un documento de requerimientos, recién entonces el equipo de desarrollo puede seguir con las actividades de planeación.

- ## SITIOS DE INTERÉS

Te invitamos a revisar los siguientes enlaces como complemento al tema revisado:

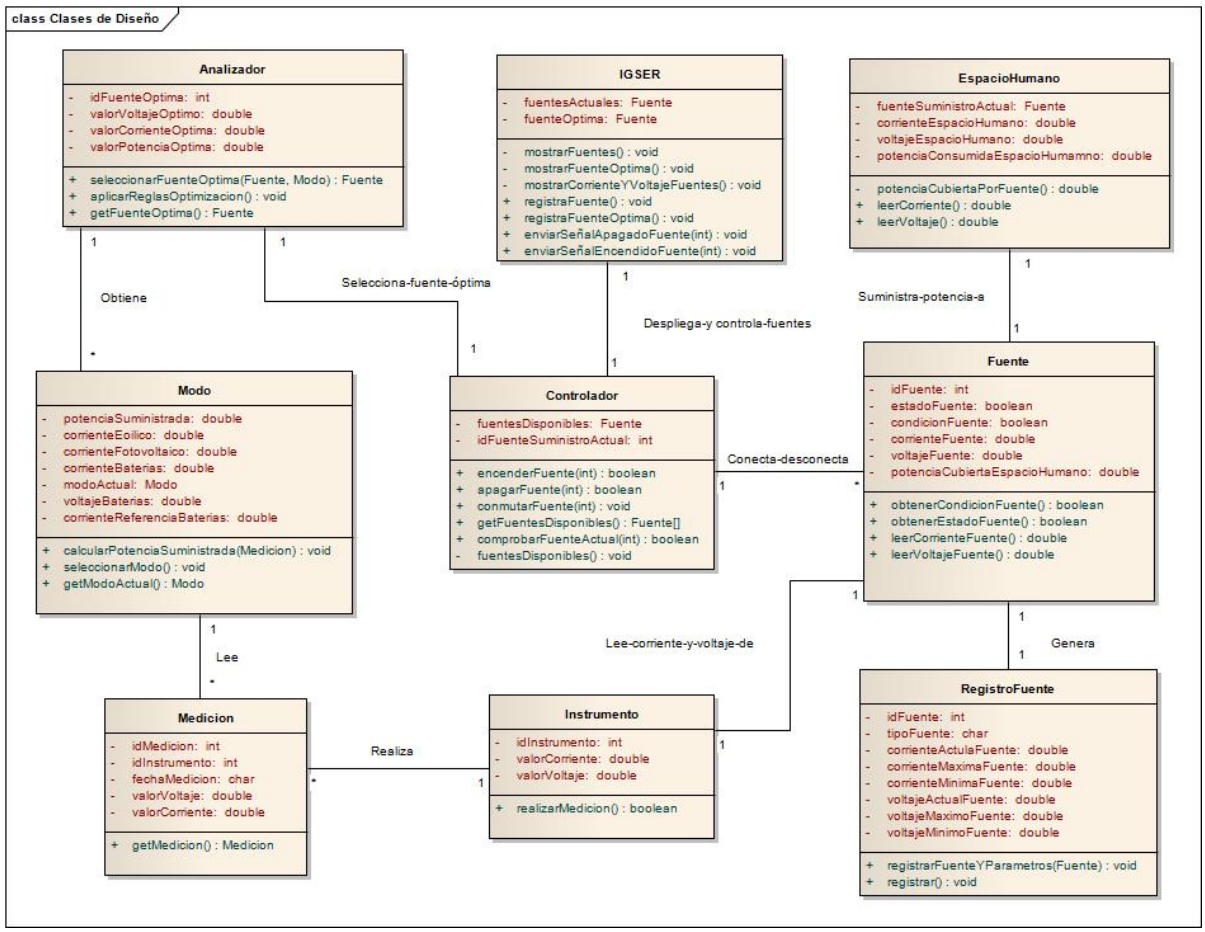
https://goo.gl/SjE9Y5

TEMA 2: "Etapas de una metodología de desarrollo"

Todo equipo de desarrollo de software debe describir las *etapas* o conjunto de actividades dentro del marco de trabajo para la metodología de desarrollo de software elegida. Es decir, cada etapa incluye un conjunto de actividades que deben realizarse y los productos que deben generarse en ella. También debe llenar cada actividad con un conjunto de acciones de desarrollo de software, y definir cada acción en cuanto a un conjunto de tareas que identifique el trabajo (y los productos del trabajo) que deben completarse para alcanzar las metas de desarrollo de software.

- ## EJEMPLIFICACIÓN

Una etapa central en el desarrollo de un software es la etapa de diseño. En ella se modela el software utilizando alguna herramienta como el Lenguaje Unificado de Modelado (UML) para representar la estructura del software resultante. En la siguiente figura se muestra un producto de la etapa de diseño, llamado diagrama de clases de diseño como resultado del diseño del software utilizando análisis orientado al objeto (AOO).



- ***SITIOS DE INTERÉS***

Para mayor profundización del tema tratado, te invitamos a revisar los siguientes enlaces:

Proyectos de Guerrilla

https://goo.gl/D5FhSJ

TEMA 3: "Objetivos de las etapas del desarrollo de software"

Los objetivos de cada etapa del desarrollo de software comprenden la entrega de ciertos productos o *artefactos*, como documentos de requisitos de usuario (DRU) o requisitos de sistema (DRS), modelos de análisis o modelos de diseño, prototipos o una primera versión del software requerido. Los objetivos también deben incorporar elementos como el cumplimiento de estándares de calidad, la localización temprana de defectos, restricciones del cronograma y el presupuesto.

- ***EJEMPLIFICACIÓN***

En el desarrollo de un software para control y monitoreo de sistemas de energía renovable (SER), que incluye la etapa de recopilación de requisitos de usuario se debía entregar un modelo de casos de uso. A continuación, se muestra el texto de un caso de uso tomado de dicho desarrollo.

Código: CU2	Visualizar voltaje y fuente que suministra actualmente energía eléctrica al espacio humano.		
Descripción	El usuario desea visualizar el voltaje (medido en Voltios) y la fuente de energía eléctrica del SER que actualmente suministra energía eléctrica al espacio humano.		
Actor principal	Usuario.		
Precondiciones	• Se debe haber seleccionado la fuente de energía eléctrica del SER de acuerdo a las reglas de uso óptimo de las fuentes renovables de energía eléctrica. • Se deben haber leído los valores de voltaje de cada fuente de energía eléctrica del SER.		
Garantías de éxito	• La selección de la fuente óptima de suministro de energía se ha realizado exitosamente. • La lectura de voltajes de las fuentes de energía eléctrica se realizó satisfactoriamente.		
Curso Normal			
1	El usuario se acerca a la pantalla de computador donde se encuentra desplegada la interfaz gráfica de usuario (IGSER) con el valor de voltaje y fuente actual óptima del SER.		
		2	El sistema debe obtener la condición enable/disable de las fuentes de energía eléctrica del SER.
		3	El sistema debe obtener el estado apagado/encendido de la(s) fuente(s) de energía eléctrica del SER.
		4	El sistema debe leer los valores de voltaje desde la(s) fuente(s) de energía eléctrica del SER que se encuentren en condición enable.
		5	El sistema debe seleccionar la fuente óptima de energía eléctrica del SER.
6	El usuario visualiza en la interfaz gráfica de usuario el valor de voltaje y la fuente óptima del SER.		
Cursos Alternos			
1a	La IGSER no muestra el valor de voltaje o la fuente óptima del SER. a. El usuario debe reiniciar el sistema. b. El usuario debe leer el valor de voltaje y la fuente óptima del SER.		
1b	La IGSER muestra un mensaje de error. a. El usuario debe reiniciar el sistema. b. El usuario debe leer el valor de voltaje y la fuente óptima del SER.		

- ## SITIOS DE INTERÉS

Para mayor profundización del tema tratado, te invitamos a revisar los siguientes enlaces:

Fases del modelo de cascada

http://fasesmodelocascada.blogspot.cl/

TEMA 4: "Metodología de desarrollo"

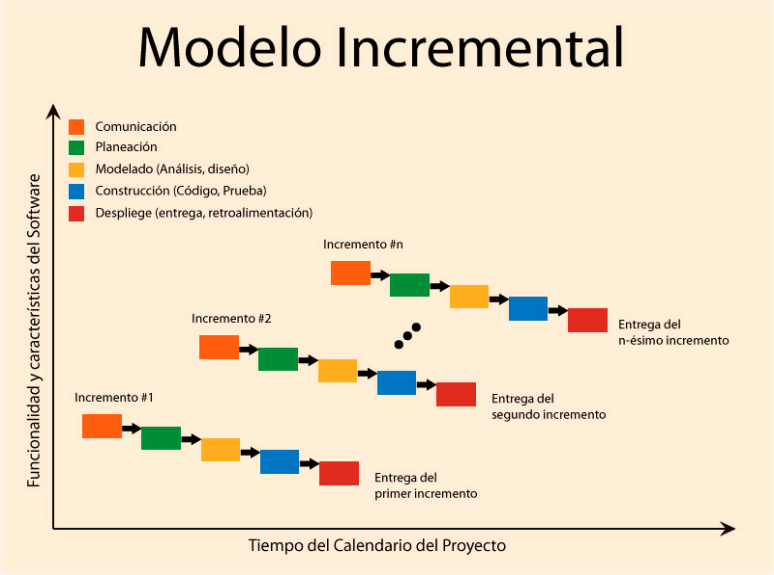
Las metodologías de desarrollo de software definen un conjunto distinto de actividades, acciones, tareas, fundamentos y productos (artefactos) de trabajo que se requieren para desarrollar software de alta calidad. Estas metodologías no son perfectas, pero proporcionan una guía útil para el trabajo de desarrollo de software. Sin importar el modelo de desarrollo de software en que está basada la metodología, pueden identificarse algunas actividades fundamentales, estas son:

- Especificación del software: tienen que definirse tanto la funcionalidad del software como las restricciones de su operación.
- Diseño e implementación del software: debe desarrollarse el software para cumplir con las especificaciones.
- Validación del software: hay que validar el software para asegurarse de que cumple lo que el cliente quiere.
- Evolución del software: el software tiene que evolucionar para satisfacer las necesidades cambiantes del cliente.

Se debe tener presente que el término proceso de software (o ciclo de vida del software) que encontrará en la literatura e internet, se refiere a las etapas que deben cumplirse para obtener un software que se ajuste a los requerimientos del cliente, pero de una forma más bien abstracta. Por otra parte, una metodología de desarrollo de software se refiere a la aplicación práctica de un cierto modelo de desarrollo de software, es decir, una metodología "aterriza" un modelo de desarrollo para ajustarlo al proyecto particular en que trabaja el equipo de desarrollo.

- ## EJEMPLIFICACIÓN

En la siguiente figura se muestran esquemáticamente las actividades que componen un modelo o paradigma tradicional de desarrollo de software llamada modelo incremental.



Fuente: imagen recuperada de https://goo.gl/w5D6Hp

- ***SITIOS DE INTERÉS***

Para mayor profundización del tema tratado, te invitamos a revisar los siguientes enlaces:

Implementación del modelo integral

https://goo.gl/uSJaqZ

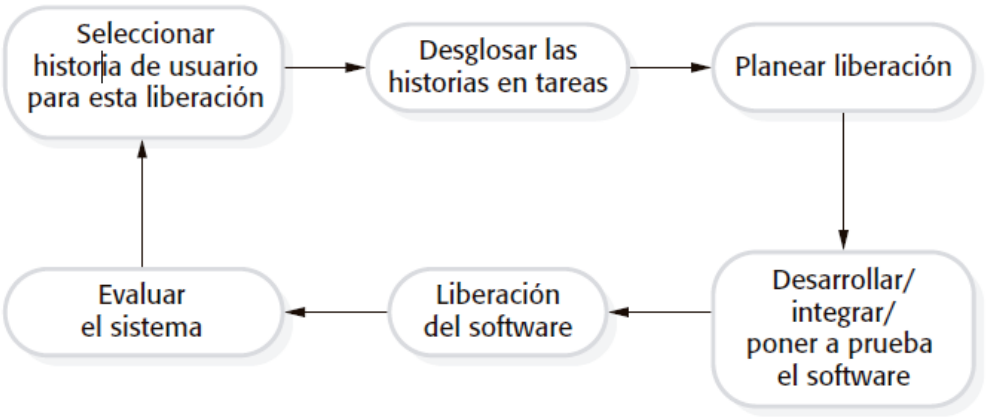
TEMA 5: "Metodología de desarrollo ágil"

Las metodologías ágiles son métodos de desarrollo de software incremental donde los incrementos son pequeños y, por lo general, se crean las nuevas liberaciones del sistema en forma frecuente (cada dos o tres semanas). Estas liberaciones se ponen a disposición de los clientes para que estos las evalúen y propongan cambios de ser necesario, es decir, involucran a los clientes en el proceso de desarrollo para conseguir una rápida retroalimentación sobre los requerimientos cambiantes.

Minimizan la cantidad de documentación con el uso de comunicaciones informales, en vez de reuniones formales con documentos escritos o "minutas". Una metodología ágil considera el diseño y la implementación como las actividades centrales dentro de la metodología de desarrollo de software.

- ## EJEMPLIFICACIÓN

En la programación extrema (XP), los requerimientos se expresan como escenarios (llamados historias de usuario), que se implementan directamente como una serie de tareas. Los programadores trabajan en pares y antes de escribir el código desarrollan pruebas para cada tarea. Todas las pruebas deben ejecutarse con éxito una vez que el nuevo código se integre en el sistema. Entre las liberaciones del sistema existe un breve lapso. La figura siguiente ilustra el proceso XP para producir un incremento del sistema por desarrollar.



- ## SITIOS DE INTERÉS

Para mayor profundización del tema tratado, te invitamos a revisar los siguientes enlaces:

Programación extrema

https://goo.gl/GzrN9g

TEMA 6: "Metodología de desarrollo tradicional"

Una metodología de desarrollo de software tradicional se refiere a una metodología que obedece a un plan establecido con anticipación (y a un modelo de desarrollo de software). Estas metodologías han sido creadas para poner orden al caos del desarrollo de software. La experiencia indica que estas han dado cierta estructura útil al trabajo de desarrollo de software y que constituyen una "hoja de ruta" eficaz para los equipos de software. Todas las actividades de la metodología se planean por anticipado y el avance del desarrollo del software se mide contra un plan. Dentro de las metodologías tradicionales más difundidas están las basadas en los modelos *cascada, incremental* y los modelos *evolutivos.*

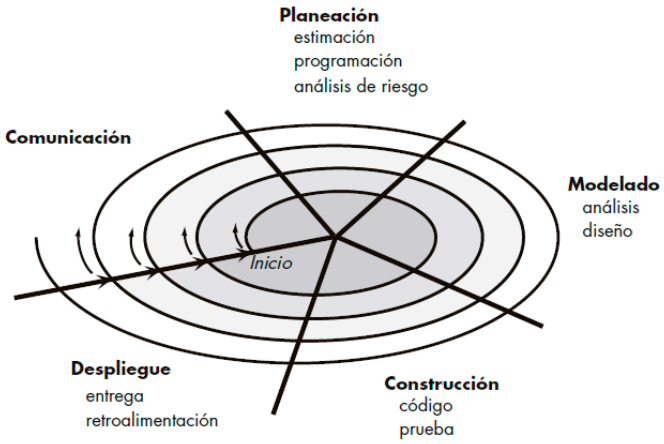
Cuando las metodologías se describen, por lo general se habla de actividades como especificar un modelo de datos, diseñar una interfaz de usuario, etcétera, así como del orden de dichas actividades. Sin embargo, al igual que las actividades, también las descripciones deben incluir:

1. Productos, que son los resultados de una actividad de la metodología. Por ejemplo, el resultado de la actividad del diseño arquitectónico es un modelo de la arquitectura de software.
2. Roles, que reflejan las responsabilidades de la gente que interviene en la metodología. Ejemplos de roles: gerente de proyecto, gerente de configuración, programador, etcétera.

Precondiciones y postcondiciones, que son declaraciones válidas antes y después de que se realice una actividad del proceso o se cree un producto. Por ejemplo, antes de comenzar el diseño arquitectónico, una precondición es que el cliente haya aprobado todos los requerimientos; después de terminar esta actividad, una postcondición podría ser que se revisen aquellos modelos UML que describen la arquitectura.

- ### EJEMPLIFICACIÓN

El modelo espiral es un modelo evolutivo de desarrollo de software que integra la naturaleza iterativa de hacer prototipos, con los aspectos controlados y sistémicos del modelo de cascada. Tiene el potencial para hacer un desarrollo rápido de versiones cada vez más completas. En la siguiente figura se muestra el esquema típico de este modelo.



Fuente: Ingeniería del Software: Un enfoque práctico Roger S. Pressman 7ª Edición McGraw Hill @2010. Página 46

- ## SITIOS DE INTERÉS

A continuación, se presenta el siguiente enlace:

Metodologías de Desarrollo de Software

https://goo.gl/EcQTbc

TEMA 7: "Fases del desarrollo de software"

Una *fase* dentro de una metodología de desarrollo de software, corresponde al desarrollo y termino (ojalá exitoso) de un conjunto de actividades. Estas actividades pertenecen a una cierta etapa del desarrollo del software. Las actividades terminan y entregan sus productos, entonces puede indicarse que la fase ha terminado. Cada fase debe tener como *output* un conjunto de productos o artefactos del proceso de desarrollo; modelos de análisis y arquitectónicos, documentos de requisitos de usuario o sistema, diseños de componentes o de despliegue en UML y otros.

- ## EJEMPLIFICACIÓN

En el caso de la metodología basada en el modelo en cascada, el resultado de cada fase consiste en uno o más documentos que se "firmaron". La siguiente fase no debe comenzar sino hasta que termine la fase previa. En la práctica, dichas fases se traslapan y se nutren mutuamente de información. Durante el diseño se identifican los problemas con los requerimientos. En la codificación se descubren problemas de diseño, y así sucesivamente. El proceso de software no es un simple modelo lineal, sino que implica retroalimentación de una fase a otra. Entonces, es posible que los

documentos generados en cada fase deban modificarse para reflejar los cambios que se realizan.

- ## SITIOS DE INTERÉS

A continuación, se presenta el siguiente enlace:

Fases de desarrollo de software
https://goo.gl/jv9wse

TEMA 8: "Tipos de metodología de desarrollo"

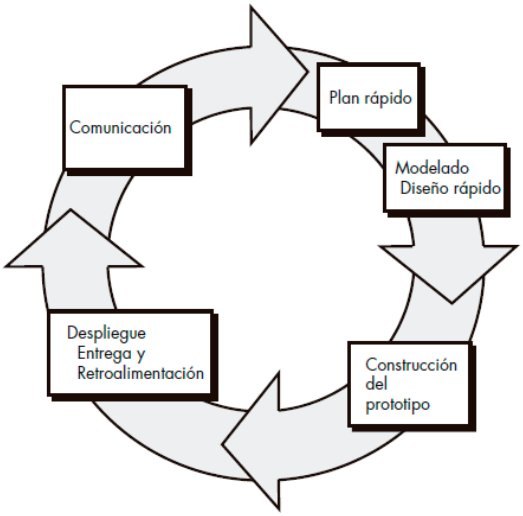
Los tipos de metodologías de desarrollo de software pueden clasificarse en dos grandes áreas; tradicionales y ágiles. Como se indicó antes, los enfoques ágiles en el desarrollo de software consideran el diseño y la implementación como las actividades centrales en el desarrollo del software. Incorporan otras actividades en el diseño y la implementación, como la adquisición de requerimientos y pruebas. En contraste, un enfoque tradicional basado en un plan para el desarrollo de software 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 metodologías tradicionales se caracterizan por obedecer a un determinado plan de desarrollo, con etapas y productos bien definidos por etapa; no se avanza a la siguiente etapa mientras la anterior no haya culminado. Por otra parte, en las metodologías ágiles, la planeación es incremental y es más fácil modificar el proceso para reflejar los requerimientos cambiantes del cliente. Algunas de las diferencias más notables entre ambos tipos de metodologías se muestran en la siguiente tabla.

Metodologías Tradicionales	Metodologías Agiles
Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo	Basadas en heurísticas provenientes de prácticas de producción de código
Cierta resistencia a los cambios	Especialmente preparados para cambios durante el proyecto
Impuestas externamente	Impuestas internamente (por el equipo)
Proceso mucho más controlado, con numerosas políticas/normas	Proceso menos controlado, con pocos principios.
El cliente interactúa con el equipo de desarrollo mediante reuniones	El cliente es parte del equipo de desarrollo
Más artefactos	Pocos artefactos
Más roles	Pocos roles
Grupos grandes y posiblemente distribuidos	Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio
La arquitectura del software es esencial y se expresa mediante modelos	Menos énfasis en la arquitectura del software
Existe un contrato prefijado	No existe contrato tradicional o al menos es bastante flexible

EJEMPLIFICACIÓN

En la metodología tradicional evolutiva de desarrollo de software creación de prototipos, un prototipo es una versión inicial de un sistema de software que se usa para demostrar conceptos, tratar opciones de diseño y encontrar más sobre el problema y sus posibles soluciones. La siguiente figura muestra las etapas típicas de desarrollo de software por prototipos.



Fuente: Ingeniería del Software: Un enfoque práctico Roger S. Pressman 7ª Edición McGraw Hill @2010. Página 43

SITIOS DE INTERÉS

A continuación, se presenta el siguiente documento como apoyo a los contenidos revisados.

Diferencias entre Metodologías Tradicionales y Ágiles

https://goo.gl/CqGJ21

TEMA 9: "Objetivos de las metodologías de desarrollo de software"

El objetivo de toda metodología de software es guiar al equipo de desarrollo en la construcción de software que esté de acuerdo con los requerimientos del usuario. Se hablará más adelante acerca de las metodologías que guian al equipo de desarrollo. Un objetivo adicional es que el software desarrollado sea de calidad. Respecto del objetivo de calidad puede indicarse lo siguiente:

1. Una metodología *eficaz de software* establece la infraestructura que da apoyo a cualquier esfuerzo de elaboración de un producto de software de alta calidad. Las prácticas de ingeniería de software permiten al desarrollador analizar el problema y diseñar una solución sólida, ambas actividades críticas en la construcción de software de alta calidad.
2. Un *producto útil* entrega contenido, funciones y características que el usuario final desea; sin embargo, de igual importancia es que entrega estos activos en forma confiable y libre de errores. Además, satisface el conjunto de requerimientos (por ejemplo, la facilidad de uso) con los que se espera que cuente el software de alta calidad.

Al *agregar valor para el productor y para el usuario* de un producto, el software de alta calidad proporciona beneficios a la organización que lo produce y a la comunidad de usuarios finales. La organización que elabora el software obtiene valor agregado porque el software de alta calidad requiere un menor esfuerzo de mantenimiento, menos errores que corregir y poca asistencia al cliente.

- ### EJEMPLIFICACIÓN

Uno de los objetivos básicos de la metodología de desarrollo de software es detectar los fallos del software antes de que este llegue al usuario. Para ello se implementan en alguna etapa del desarrollo las llamadas *pruebas de software.* Una prueba es el proceso de ejecutar un conjunto de elementos de software (código) con el fin de encontrar errores. Para construir una prueba deben formularse los denominados *casos de prueba,* que corresponde a un conjunto de condiciones, datos o variables bajo las cuales el desarrollador determinará si el o los requisitos del software se cumplen de manera parcial, completa o no se cumplen.

- ### SITIOS DE INTERÉS

A continuación, se presenta el siguiente enlace como apoyo a los contenidos revisados.

Pruebas de software
https://goo.gl/oq7E14

TEMA 10: "Clasificación de las metodologías de desarrollo de software"

Como se indicó en secciones anteriores, las metodologías de desarrollo de software están basadas en modelos de desarrollo de software, los que, como vimos, pueden ser tradicionales o ágiles. Una segunda clasificación obedece al *paradigma* utilizado para diseñar el software. Los paradigmas más aceptados son el *estructurado*, *orientado a objetos* y para sistemas en *tiempo real.*

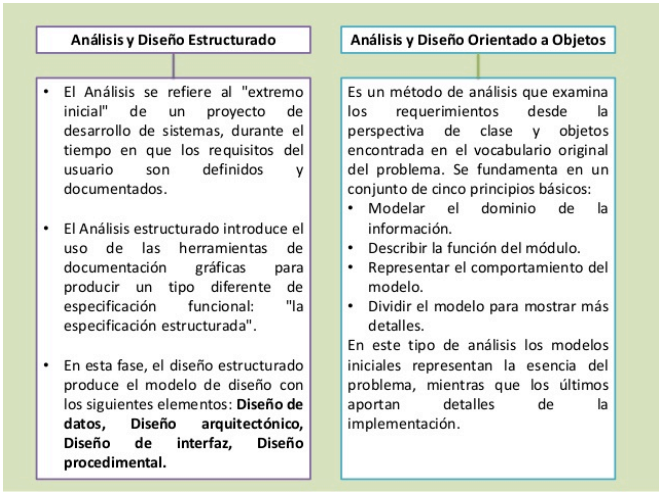
El paradigma estructurado tiene como fundamento el uso de análisis y diseño estructurado para su uso con herramientas CASE (Computer Aided Software Engineering, Ingeniería de Software Asistida por Computadora), incrementado la producción en el desarrollo e implantación de software. Propone la creación de modelos de sistemas que representan los procesos, los flujos y la estructura de los datos de manera descendente (*top down*). Se pasa de una visión general del problema (nivel alto de abstracción) hasta llegar a niveles más bajos de abstracción. El paradigma estructurado puede ser *orientado a procesos, orientado a datos* (jerárquico o no jerárquico) o mixto.

En el paradigma orientado a objetos (OO) los elementos del software se diseñan usando un enfoque de objetos, es decir, modelando los elementos de software en *clases* con características y comportamientos, o en lenguaje OO, con *atributos* y *métodos.* De esta manera, cada clase representa un elemento de software independiente que interacciona con otras clases, a través de sus métodos, almacenando información en los atributos. Dentro de las metodologías orientadas a objetos están Rational Unified Process (RUP) y Booch (OOD).

Por último, la metodología para desarrollo de software en tiempo real se orienta al desarrollo de software que procesan información orientada al control. Controlan y son controlados por eventos externos. Hacen manejo de interrupciones, comunicación y sincronización entre tareas, gestión de procesos concurrentes, respuesta oportuna a eventos externos, y pueden manejar tanto datos continuos como discontinuos.

- ### EJEMPLIFICACIÓN

En la siguiente figura se muestra un cuadro comparativo de algunas características adicionales de las metodologías estructuradas y orientadas al objeto.



- ## SITIOS DE INTERÉS

A continuación, se presentan los siguientes documentos como apoyo a los contenidos revisados.

Metodologías de desarrollo de software

https://goo.gl/FreyfR

TEMA 11: "El modelo en cascada"

El modelo en cascada o *ciclo de vida clásico* es un enfoque de desarrollo de software sistemático y secuencial hacia el desarrollo de software, que se inicia con la especificación de requerimientos del cliente y que continúa con la planeación, el modelado, la construcción y el despliegue para culminar en el soporte del software terminado. El modelo en cascada presenta una visión muy clara de cómo suceden las etapas durante el desarrollo, y sugiere a los desarrolladores cual es la secuencia de eventos que podrán encontrar.

Dentro de las fortalezas principales que podemos mencionar se encuentran:

- Soporta la baja experiencia de equipos y jefes de proyecto.
- La estricta secuencia de etapas de desarrollo asegura una documentación adecuada y una revisión del diseño que ayuda a asegurar la calidad, confiabilidad y mantenibilidad del desarrollo del software.
- El progreso del desarrollo es medible.

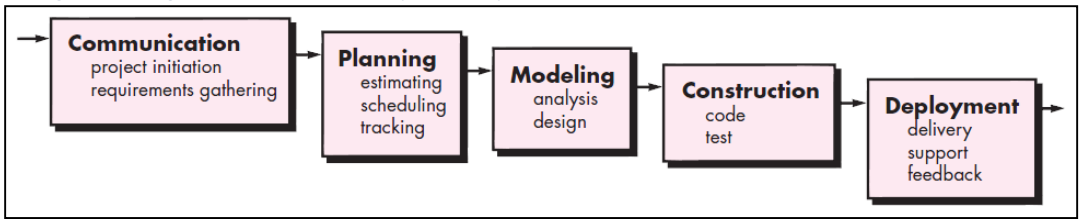
Mientras que dentro de las debilidades podemos mencionar:

- El avance del proyecto es hacia adelante, con pequeñas e indirectas iteraciones.
- Es difícil para el cliente establecer al comienzo del proceso todos los requisitos de manera explícita.
- La aparición de requerimientos inconsistentes, componentes de sistema faltantes e

inesperadas necesidades son detectadas en el diseño, desarrollo o incluso el testing.

- El desempeño del sistema no puede ser probado hasta etapas muy avanzadas del desarrollo.
- Dificultad para responder a los cambios.

La siguiente figura muestra un esquema típico del modelo en cascada.



Fuente: Ingeniería del Software: Un enfoque práctico Roger S. Pressman 7ª Edición McGraw Hill @2010. Página 39

EJEMPLIFICACIÓN

El modelo en cascada es especialmente adecuado en situaciones donde los requerimientos están fijos y bien definidos, y donde el trabajo se realiza hasta su concreción de una manera lineal. Estas razones, y la experiencia del equipo de desarrollo justifican la adopción del modelo en cascada en muchos proyectos complejos y de larga duración (años).

SITIOS DE INTERÉS

A continuación, se presentan los siguientes documentos como apoyo a los contenidos revisados.

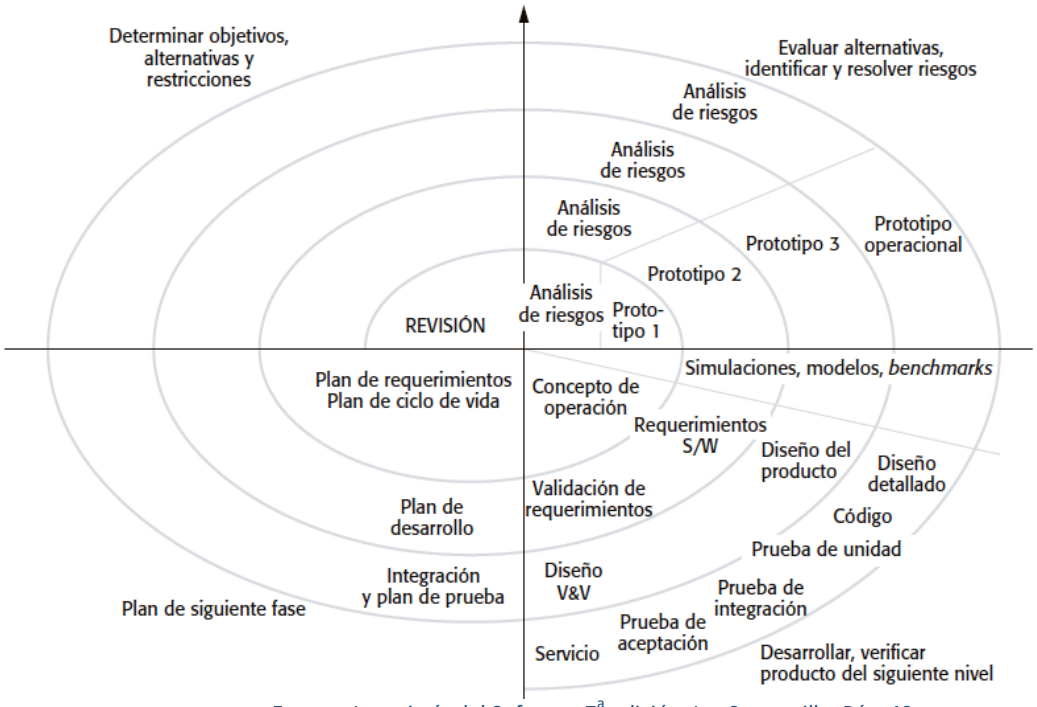
Modelo cascada

https://goo.gl/b8NMPB

TEMA 12: "Modelo espiral"

El modelo en espiral es un enfoque de desarrollo de software evolutivo que conjuga la naturaleza iterativa de la construcción de prototipos con los aspectos controlados y sistemáticos del modelo en cascada. En el modelo en espiral se desarrollan una serie de entregas evolutivas, donde las primeras entregas pueden ser un documento del modelo o un prototipo. Durante las últimas iteraciones se producen versiones cada vez más completas del software desarrollado. Un proceso de software en espiral se divide en un conjunto de actividades del marco de trabajo definido por el equipo de proyecto.

La figura siguiente muestra un esquema típico de un modelo de desarrollo de software en espiral.



Fuente: Ingeniería del Software 7ª edición. Ian Somerville. Pág. 49

- ## EJEMPLIFICACIÓN

En el modelo de desarrollo en espiral cada ciclo representa una fase del desarrollo de software. Por ende, el ciclo más interno puede relacionarse con la factibilidad del sistema, el siguiente ciclo con la definición de requerimientos, el ciclo que sigue con el diseño del sistema, etc. El modelo en espiral combina el cambio con la tolerancia a este. Lo anterior supone que los cambios son resultado de riesgos del proyecto e incluye actividades de gestión de riesgos para reducir tales riesgos.

- ## SITIOS DE INTERÉS

A continuación, se presenta el siguiente enlace como apoyo a los contenidos revisados.

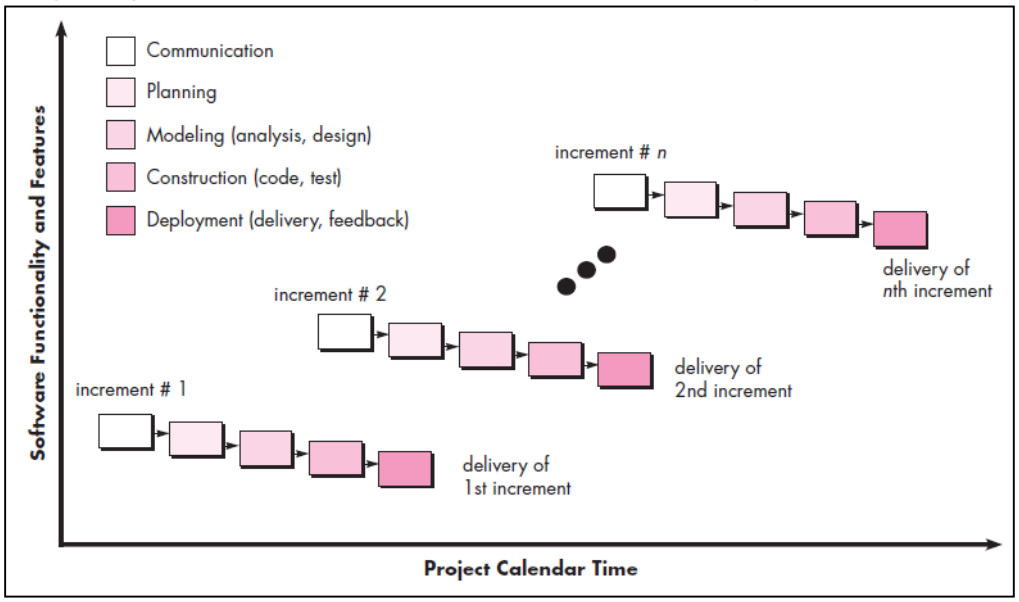
Desarrollo de modelo espiral

https://goo.gl/oY8mHv

TEMA 13: "Modelo iterativo-incremental"

El modelo incremental combina elementos del modelo en cascada aplicados en forma iterativa. Este modelo aplica secuencias lineales de manera escalonada según avanza el tiempo. Cada secuencia lineal produce incrementos del software especificados de acuerdo a su funcionalidad. Al utilizar el modelo incremental el primer incremento normalmente es un producto esencial donde se incorporan los requisitos básicos, luego como resultado de la evaluación, se desarrolla un plan para el incremento siguiente. El plan afronta la modificación del producto esencial con el fin de satisfacer de mejor manera las necesidades del cliente y la entrega de características y funcionalidades adicionales. Este proceso de evaluación se repite después de la entrega de cada incremento mientras no se haya elaborado el producto completo.

La figura siguiente muestra un modelo iterativo – incremental típico.



Fuente: Ingeniería del Software: Un enfoque práctico Roger S. Pressman 7ª Edición McGraw Hill @2010. Página 42

• EJEMPLIFICACIÓN

Muchas organizaciones utilizan el modelo iterativo – incremental porque:

1. El entrenamiento puede comenzar sobre una versión temprana del software bajo desarrollo, aún si se han omitido funcionalidades importantes. El proceso de entrenamiento permite que los desarrolladores observen como se ejecutan ciertas funciones, sugiriendo que se mejoren para versiones posteriores.
2. Pueden crearse tempranamente mercados para funcionalidades que nunca antes se habían ofrecido.
3. Las versiones frecuentes permiten que los desarrolladores arreglen problemas no anticipados de manera global.
4. El equipo de desarrollo puede centrarse en diferentes áreas de especialización con las diferentes versiones

• SITIOS DE INTERÉS

A continuación, se presenta el siguiente enlace como apoyo a los contenidos revisados.

Ciclo de vida iterativo

https://goo.gl/5zMjqr

TEMA 14: "RUP"

El Proceso Unificado Racional (RUP, por las siglas de *Rational Unified Process*) es un modelo de desarrollo de software moderno que se derivó del trabajo sobre el UML (Lenguaje Unificado de Modelado) y el proceso asociado de desarrollo de software unificado. Conjunta elementos de todos los modelos tradicionales de desarrollo de software; ilustra la buena práctica en especificación y diseño, y apoya la creación de prototipos y entrega incremental. El RUP reconoce que los modelos de desarrollo tradicionales presentan una sola visión del desarrollo.

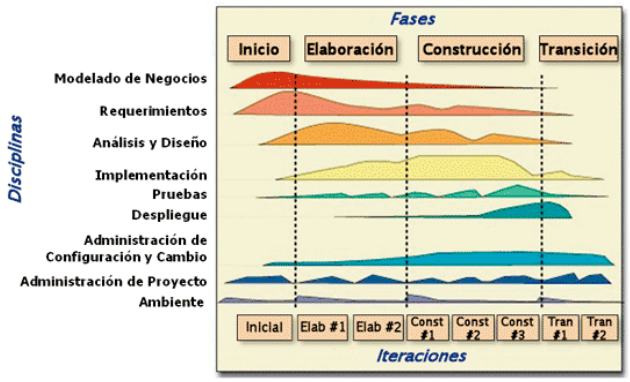
El RUP es un modelo que identifica cuatro fases discretas en el desarrollo de software. Sin embargo, a diferencia del modelo en cascada, donde las fases se igualan con actividades, las fases en el RUP están más estrechamente vinculadas con las necesidades de la empresa que con las preocupaciones técnicas.

1. *Concepción.* La meta de la fase de concepción es establecer un caso empresarial para el software. Deben identificarse todas las entidades externas (personas y sistemas) que

interactuarán con el software y se definirán dichas interacciones. Luego se usa esta información para valorar la aportación del software hacia la empresa. Si esta aportación es menor, entonces el proyecto puede cancelarse después de esta fase.

2. *Elaboración.* Las metas de la fase de elaboración consisten en desarrollar la comprensión del problema de dominio, establecer un marco conceptual arquitectónico para el software, diseñar el plan del proyecto e identificar los riesgos clave del proyecto. Al completar esta fase, debe tenerse un modelo de requerimientos para el software, que podría ser una serie de casos de uso del UML, una descripción arquitectónica y un plan de desarrollo para el software.
3. *Construcción.* La fase de construcción incluye diseño, programación y pruebas del software. Partes del software se desarrollan en paralelo y se integran durante esta fase. Al completar ésta, debe tenerse un sistema de software funcionando y la documentación relacionada y lista para entregarse al usuario.
4. *Transición.* La fase final del RUP se interesa por el cambio del sistema desde la comunidad de desarrollo hacia la comunidad de usuarios, y por ponerlo a funcionar en un ambiente real. Esto es algo ignorado en la mayoría de los modelos de desarrollo de software, aunque, en efecto, es una actividad costosa y en ocasiones problemática. En el complemento de esta fase se debe tener un sistema de software documentado que funcione correctamente en su entorno operacional.

La siguiente figura muestra un gráfico con las fases y *disciplinas* típicas de RUP



Fuente: http://metodoss.com/metodologia-rup/

• EJEMPLIFICACIÓN

La siguiente figura muestra un cuadro de las tareas típicas de modelo de desarrollo de software RUP

RUP			UML	RESUMEN
Fase	**Actividad**	**Entregable**		
Inicial	Modelamiento del Negocio	Documento de Visión	Extensión para Modelado Negocio	• Documento de Visión. • Plan de Desarrollo del Software. • Modelado de Use Case Del Negocio. • Entorno de Trabajo
		Plan de Desarrollo del Software		
		Modelo de Use Case del Negocio		
		Entorno de Trabajo		
Elaboración	Requerimientos	Modelo de Use Case	Diagrama de Use Case	Diagrama de Use Case
	Análisis y Diseño	Modelo del Análisis	Diagrama de Colaboraciones	Diagrama de Use Colaboraciones
		Diseño de Interfaces	Diagrama de Secuencia	Diagrama de Secuencia
		Diseño de Clases	Diagrama de Clases	Diagrama de Clases Diagrama de Vistas
		Plantilla de Clases		Plantilla de Clases
		Diseño de la Base de Datos.		Diseño de la Base de Datos.
		Modelo de Despliegue	Modelo de Despliegue	Modelo de Despliegue
		Prototipo Arquitectónico		Prototipo Arquitectónico
Construcción	Implementación	Modelo de Componentes	Diagrama de Componentes	Diagrama de Componentes Vistas de Componentes
	Prueba	Modelo de Caja Negra		Modelo de Caja Negra
		Prototipo del Software		Prototipo del Software
Transición	Despliegue	Prueba de Aceptación		Documento de Aceptación del Producto Software

• SITIOS DE INTERÉS

A continuación, se presenta el siguiente enlace como apoyo a los contenidos revisados.

Modelo RUP

https://goo.gl/LYLZSW

TEMA 15: "Programación Extrema (XP)"

La programación extrema usa un enfoque orientado a objetos como paradigma preferido de desarrollo, y engloba un conjunto de reglas y prácticas que ocurren en el contexto de cuatro actividades estructurales: planeación, diseño, codificación y pruebas. En el siguiente párrafo se explican cada actividad en mayor detalle.

Planeación. La actividad de planeación comienza *escuchando* —actividad para recabar requerimientos que permitan que los miembros técnicos del equipo XP entiendan el contexto del negocio para el software y comprendan acerca de la salida y características principales y funcionalidad que se requieren—. Escuchar lleva a la creación de algunas "*historias del usuario*" qué describen la salida que se necesita, características y funcionalidad del software que se va a elaborar. Cada *historia* es similar los casos de uso y es escrita por el cliente y colocada en una tarjeta indexadazada. El cliente asigna una prioridad (valor) a la historia con base en el valor general de la característica o función para el negocio. Después los miembros del equipo de desarrollo evalúan cada historia y le asignan un costo, medido en semanas de desarrollo. Si se estima que la historia requiere más de tres semanas de desarrollo, se pide al cliente que la descomponga en historias más cortas y de nuevo se asigna un valor y costo. Es importante observar que en cualquier momento es posible escribir nuevas historias.

Los clientes y desarrolladores trabajan juntos para decidir cómo agrupar las historias en la siguiente en el siguiente incremento de software que desarrollará el equipo XP. Una vez que se llega a un *compromiso* sobre la entrega (acuerdo sobre las historias por incluir, la fecha de entrega y otros aspectos del proyecto), el equipo XP ordena las historias que serán desarrolladas en una de tres formas: 1) todas las historias se implementarán de inmediato (en pocas semanas), 2) las historias con más valor entrarán a la programación de actividades y se implementarán en primer lugar o 3) las historias más riesgosas formarán parte de la programación de actividades y se implementarán primero. Después de la primera entrega del proyecto (también llamada incremento de software), el equipo XP calcula la velocidad de éste. En pocas palabras, la *velocidad del proyecto* es el número de historias de los clientes implementadas durante la primera entrega. La velocidad del proyecto se usa para: 1) ayudar a estimar las fechas de entrega y programar las actividades para las entregas posteriores, y 2) determinar si se ha hecho un gran compromiso para todas las historias durante todo el desarrollo del proyecto. Si esto ocurre, se modifica el contenido de las entregas o se cambian las fechas de entrega final.

Diseño. El diseño XP sigue rigurosamente el principio MS (mantenlo sencillo). Un diseño sencillo siempre se prefiere sobre una representación más compleja. Además, el diseño guía la implementación de una historia conforme se escribe: nada más y nada menos. Se desalienta el diseño de funcionalidad adicional porque el desarrollador supone que se requerirá después. XP estimula el uso de las tarjetas CRC (Clase-Responsabilidad-Colaboración) como un mecanismo eficaz para pensar en el software en un contexto orientado a objetos. Las tarjetas CRC identifican y

organizan las clases orientadas a objetos que son relevantes para el incremento actual de software. Las tarjetas CRC son el único producto del trabajo de diseño que se genera como parte del proceso XP. Si en el diseño de una historia se encuentra un problema de diseño difícil, XP recomienda la creación inmediata de un prototipo operativo de esa porción del diseño. Entonces, se implementa y evalúa el prototipo del diseño, llamado *solución en punta*. El objetivo es disminuir el riesgo cuando comience la implementación verdadera y validar las estimaciones originales para la historia que contiene el problema de diseño.

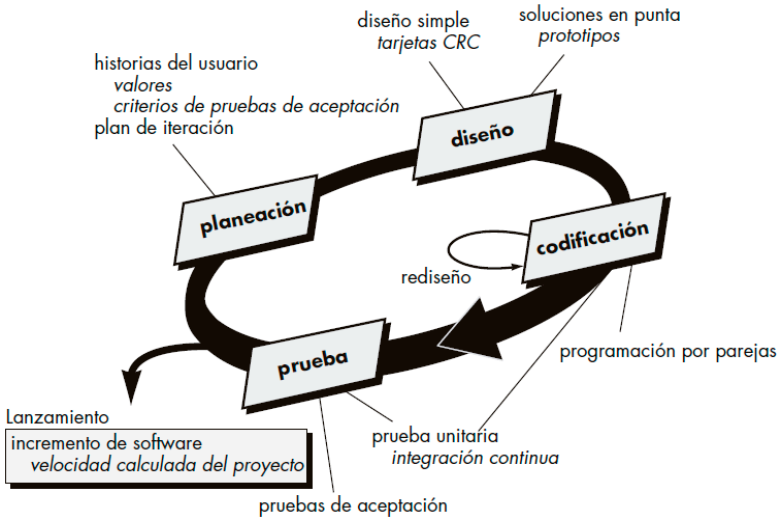
Un concepto central en XP es que el diseño ocurre tanto antes *como después* de que comienza la codificación. Rediseñar significa que el diseño se hace de manera continua conforme se construye el sistema. En realidad, la actividad de construcción en sí misma dará al equipo XP una guía para mejorar el diseño.

Codificación. Después de que las historias han sido desarrolladas y de que se ha hecho el trabajo de diseño preliminar, el equipo *no* inicia la codificación, sino que desarrolla una serie de pruebas unitarias a cada una de las historias que se van a incluir en la entrega en curso (incremento de software). Una vez creada la prueba unitaria, el desarrollador está mejor capacitado para centrarse en lo que debe implementarse para pasar la prueba. No se agrega nada extraño (MS). Una vez que el código está terminado, se le aplica de inmediato una prueba unitaria, con lo que se obtiene retroalimentación instantánea para los desarrolladores. Un concepto clave durante la actividad de codificación es la *programación por parejas*. XP recomienda que dos personas trabajen juntas en una estación de trabajo con el objeto de crear código para una historia. Esto da un mecanismo para la solución de problemas en tiempo real y para el aseguramiento de la calidad también en tiempo real. También mantiene a los desarrolladores centrados en el problema de que se trate. En la práctica, cada persona adopta un papel un poco diferente. Por ejemplo, una de ellas tal vez piense en los detalles del código de una porción particular del diseño, mientras la otra se asegura de que se siguen los estándares de codificación (parte necesaria de XP) o de que el código para la historia satisfará la prueba unitaria desarrollada a fin de validar el código confrontándolo con la historia.

A medida que las parejas de programadores terminan su trabajo, el código que desarrollan se integra con el trabajo de los demás. En ciertos casos, esto lo lleva a cabo a diario un equipo de integración. En otros, las parejas de programadores tienen la responsabilidad de la integración. Esta estrategia de "integración continua" ayuda a evitar los problemas de compatibilidad e interfaces y brinda un ambiente de "prueba de humo" (pruebas que pretenden evaluar la calidad de un producto de software previo a una recepción formal) que ayuda a descubrir a tiempo los errores.

Pruebas. Ya se dijo que la creación de pruebas unitarias antes de que comience la codificación es un elemento clave del enfoque de XP. Las pruebas unitarias que se crean deben implementarse con el uso de una estructura que permita automatizarlas. Esto estimula una estrategia de pruebas de regresión, siempre que se modifique el código (lo que ocurre con frecuencia, dada la filosofía del rediseño en XP). A medida que se organizan las pruebas unitarias individuales en un "grupo de prueba universal", las pruebas de la integración y validación del sistema pueden efectuarse a diario. Esto da al equipo XP una indicación continua del avance y también lanza señales de alerta si las cosas marchan mal. Las *pruebas de aceptación* XP, también llamadas *pruebas del cliente*, son especificadas por el cliente y se centran en las características y funcionalidad generales del sistema que son visibles y revisables por parte del cliente. Las pruebas de aceptación se derivan de las historias de los usuarios que se han implementado como parte de la liberación del software.

La siguiente figura ilustra las actividades de la metodología de desarrollo XP.



Fuente: Ingeniería del Software: Un enfoque práctico Roger S. Pressman 7ª Edición McGraw Hill @2010. Página 62

- ## EJEMPLIFICACIÓN

En el desarrollo de un programa de contabilidad para una pyme puede usarse la metodología XP. Ella permite enfocarse en las necesidades de los empleados de la pyme a través de una comunicación continua, no generar documentación (salvo las tarjetas CRC) y actuar con rapidez y flexibilidad ante cambios en los requisitos.

- ## SITIOS DE INTERÉS

A continuación, se presenta el siguiente enlace como apoyo a los contenidos revisados.

Programación extrema

https://goo.gl/Poz2if

TEMA 16: "Lean Development (LD)"

También conocido como "Lean Programming", esta es una metodología ágil para el desarrollo de software orientado a conseguir exactamente lo que necesita el cliente. Es una evolución de metodologías japonesas aplicado al desarrollo de software y que está muy de moda entre los equipos de desarrollo en startups (equipo de trabajo técnico pero versátil, que permite la rotación de sus integrantes en varios puestos de trabajo dependiendo de las necesidades del software).

Principalmente se identifica por hacer uso de ciclos de evolución de software incrementales en los que se alejan las decisiones lo más posible hasta no tener retroalimentación por parte del cliente y con esto reaccionar de modo más flexible posible contra sus posibles necesidades cambiantes. Por esto mismo de provenir de una metodología japonesa de trabajo se fundamenta en tener un equipo muy capaz y comprometido al principio del aprendizaje continuo.

La siguiente figura ilustra los principios del desarrollo de software mediante LD.

Fuente: https://es.slideshare.net/diegoorlandoquispecondori/modelos-de-desarrollo-de-software-inf162-2017

- ***EJEMPLIFICACIÓN***

El Desarrollo Lean es una metodología fantástica para empresas que están desarrollando software B2C (Business-to-Consumer, del negocio al consumidor) orientado a tener éxito en el mercado. Las iteraciones rápidas, con alta interacción con el cliente o usuario, proveen los mecanismos de

retroalimentación suficientes, para que el desarrollo se ajuste fielmente a los requisitos, impidiendo largas iteraciones que puedan conducir a errores en la comprensión de lo que se necesita.

- ## SITIOS DE INTERÉS

A continuación, se presenta el siguiente enlace como apoyo a los contenidos revisados.

Metodologías de desarrollo ágil

https://goo.gl/9kzgeq

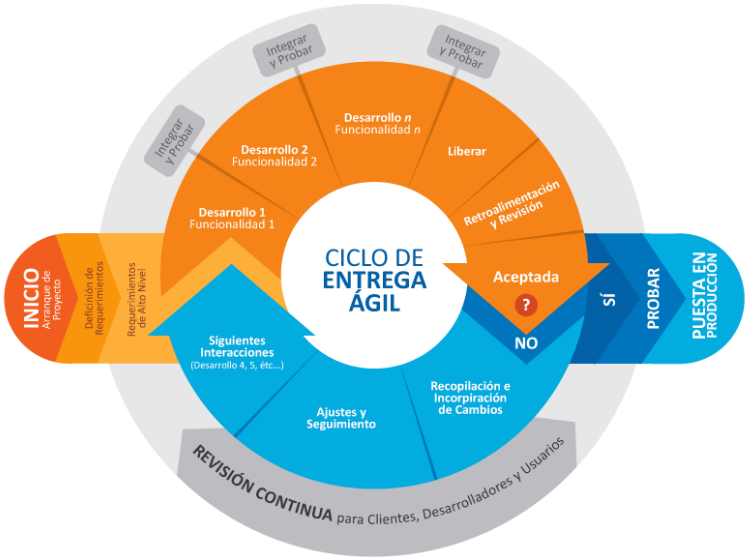
TEMA 17: "Crystal Clear"

Crystal es una metodología de desarrollo de software que más que una metodología se la considera una familia de metodologías, debido a que se subdivide en varios tipos de metodologías en función a la cantidad de persona que vayan a estar en el proyecto. Es una metodología que ha sido creada por Alistair Cockburn. La creó en base al análisis de distintos proyectos de desarrollo de software y a su propia experiencia.

Crystal Clear no es una metodología en si misma sino una familia de metodologías con un "código genético" común. El nombre Crystal deriva de la caracterización de los proyectos según 2 dimensiones, tamaño y complejidad (como en los minerales, color y dureza). CC puede ser usado en proyectos pequeños y como casi todos los otros métodos. CC consiste en valores, técnicas y procesos.

La familia de metodologías Crystal comparten con la XP una orientación humana, pero esta centralización en la gente se hace de una manera diferente. Alistair considera que las personas encuentran difícil seguir un proceso disciplinado, así que más que seguir la alta disciplina de la XP, Alistair explora la metodología menos disciplinada que aún podría tener éxito, intercambiando conscientemente productividad por facilidad de ejecución. Él considera que, aunque Crystal es menos productivo que la XP, más personas serán capaces de seguirlo.

La siguiente figura ilustra las fases del ciclo de CC.



Fuente: https://iswugaps2crystalclear.wordpress.com/

- ## EJEMPLIFICACIÓN

La aplicación de CC al proyecto Roble Alto permitió un desarrollo flexible del software para la administración del parque. La metodología fue especialmente apropiada, dado el gran número de participantes del proyecto, permitiendo canalizar los diferentes requerimientos de administración y logística del parque.

- ## SITIOS DE INTERÉS

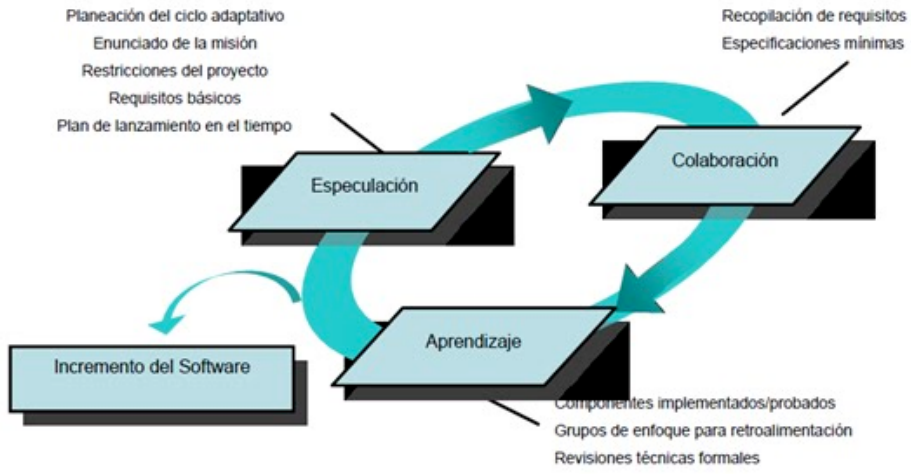
A continuación, se presenta el siguiente enlace como apoyo a los contenidos revisados.

Metodologías de desarrollo ágil

https://goo.gl/9kzgeq

Sus características son: iterativo, orientado a los componentes de software más que a las tareas y tolerante a los cambios. El ciclo de vida que propone tiene tres fases; especulación, colaboración y aprendizaje. En la primera se planifican las características del software. En la segunda se desarrollan las características y finalmente en la tercera fase se revisa su calidad, y se entrega al cliente. La revisión de los componentes sirve para aprender de los errores y volver a iniciar el ciclo de desarrollo.

La figura siguiente muestra las fases de ASD.



Fuente: http://desarrolloadaptativodesoftware.blogspot.cl/2011/06/desarrollo-adaptativo-de-software-das.html

- ## EJEMPLIFICACIÓN

Como ejemplo de algunos de los beneficios de aplicar ASD podemos mencionar:

1. Enfoque "Do It Wrong the First Time" – Hazlo mal la primera vez

Este enfoque parte de la premisa de que realizar un buen trabajo en base a predicciones en entornos complejos es **muy complicado**, y por lo tanto contempla la posibilidad de **aprender de los errores** en la fase de revisión de componentes, que permitirá comenzar nuevamente el **ciclo de desarrollo**, lo que aumentará ostensiblemente la **calidad** del software desarrollado.

2. Apunta hacia el desarrollo rápido de aplicaciones (RAD)

Esta metodología pone en valor al cliente involucrándole en el proceso, añadiendo un plus de usabilidad al producto final, sin olvidar tampoco la rapidez de desarrollo.

3. Al entender al entorno como cambiante, se adapta al mismo

Tal y como su propio nombre indica, es una metodología que favorece la **adaptabilidad del software**,

ya que su comportamiento va a ser continuamente mejorado en base a la información disponible de los posibles cambios que puedan suceder.

- ***SITIOS DE INTERÉS***

A continuación, se presenta el siguiente enlace como apoyo a los contenidos revisados.

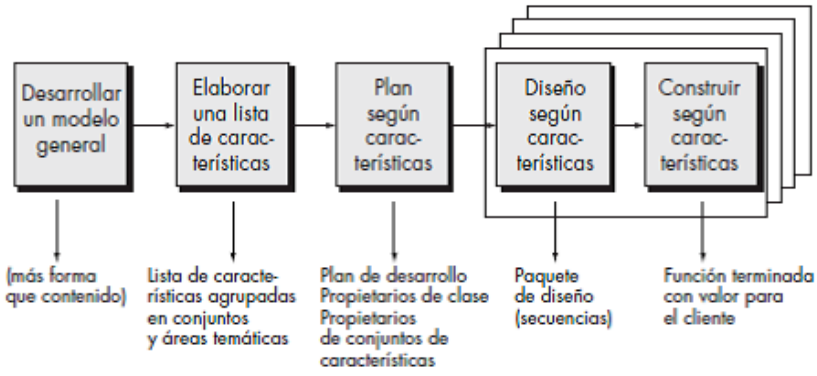
Desarrollo adaptativo del software

https://goo.gl/BgJ3Y8

TEMA 19: "Feature Driven Development (FDD)"

Feature Driven Development (FDD) es un framework o **metodología dirigida por modelos, y de iteraciones cortas.** FDD **define 5 procesos:** proceso 1 – Desarrollar el modelo general (Develop overall model), proceso 2 – Elaborar una lista de características (Build feature list), proceso 3 – Planificar según las características (Plan by feature), proceso 4 – Diseñar según las características (Design by feature) y proceso 5 – Construir (Build by feature). Los 3 primeros pueden considerarse la "iteración cero", aunque en FDD no le llaman así, y los consideran "procesos iníciales".

La siguiente figura ilustra los procesos involucrados en FDD.



Fuente: http://bustejose.blogspot.cl/2014/07/tema-3.html

EJEMPLIFICACIÓN

Como ejemplo de la aplicación de la metodología FDD mencionaremos el sistema SISFIUX. Para este sistema se detallaremos los tres primeros procesos:

1. Modelo del dominio general

De acuerdo con FDD el primer proceso está dedicado al modelo de dominio, por lo que fue necesario realizar una reunión con los expertos del dominio y el arquitecto en jefe. Los procesos que se llevan a cabo en el departamento de ingresos de la Universidad de Xalapa que necesitan ser automatizados y que se toman como base para la elaboración del modelo de dominio general son: Proyecciones financieras de ingresos, Reportes de ingresos y Reportes de Vales.

2. Lista de características

Una vez que se ha construido el modelo de dominio general, la siguiente fase es la de construir una lista de características, considerando el modelo de dominio se identifica que el sistema debe contar con los módulos principales tales como: proyecciones financieras, reportes de ingresos y reporte de vales. Para ello se construye la lista de características considerando áreas, subáreas y características.

3. Una vez construida la lista de características se estableció un plan por características que consiste en calendarizar la forma en la que serán construidas estas. Como parte de las adaptaciones que se realizaron al proceso de FDD se adoptó SCRUM como marco de trabajo para la administración del proyecto más adelante se detalla cómo se llevó acabo esta adaptación.

SITIOS DE INTERÉS

A continuación, se presenta el siguiente enlace como apoyo a los contenidos revisados.

SISFIUX: adaptación de Feature-driven Development para el desarrollo de un sistema financiero para una universidad

https://goo.gl/oKhcVW

TEMA 20: "Metodología ágil vs metodología tradicional"

Las metodologías ágiles cobran importancia cuando la comunidad de desarrollo de software se da cuenta que los proyectos guiados por las metodologías tradicionales se vuelven engorrosos cuando se aplican a software de negocios pequeños y medianos, y los costos que se incluyen son tan grandes que dominan el proceso de desarrollo; se invierte más tiempo en diseñar el software, que en el desarrollo y la prueba del mismo. El problema aumenta cuando cambian los requerimientos del software y resulta esencial la reelaboración y, en principio al menos, la especificación y el diseño deben modificarse con el software ya en fase de pruebas. Estos enfoques tradicionales incluyen costos operativos significativos en la planeación, el diseño y la documentación del software. Dichos gastos se justifican cuando debe coordinarse el trabajo de múltiples equipos de desarrollo, cuando el sistema es un software crítico y cuando numerosas personas intervendrán en el mantenimiento del software a lo largo de su vida.

La siguiente figura muestra una tabla comparativa de metodologías ágiles vs tradicionales

Diferencias entre Metodología Ágil y Metodología Tradicional

Metodologías Ágiles	Metodologías Tradicionales
Basadas en heurísticas provenientes de prácticas de producción de código	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo
Especialmente preparados para cambios durante el proyecto	Cierta resistencia a los cambios
Impuestas internamente (por el equipo)	Impuestas externamente
Proceso menos controlado, con pocos principios	Proceso mucho más controlado, con numerosas políticas/normas
No existe contrato tradicional o al menos es bastante flexible	Existe un contrato prefijado
El cliente es parte del equipo de desarrollo	El cliente interactúa con el equipo de desarrollo mediante reuniones
Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio	Grupos grandes y posiblemente distribuidos
Pocos artefactos	Mas artefactos
Pocos roles	Mas roles
Menos énfasis en la arquitectura del software	La arquitectura del software es esencial y se expresa mediante modelos

- ## EJEMPLIFICACIÓN

Los métodos ágiles han tenido mucho éxito para ciertos tipos de desarrollo de software:

1. Desarrollo del producto, donde una compañía de software elabora un producto pequeño o mediano para su venta.
2. Diseño de sistemas a la medida dentro de una organización, donde hay un claro compromiso del cliente por intervenir en el proceso de desarrollo, y donde no existen muchas reglas ni

regulaciones externas que afecten el software.

- ## SITIOS DE INTERÉS

A continuación, se presenta el siguiente enlace como apoyo a los contenidos revisados.

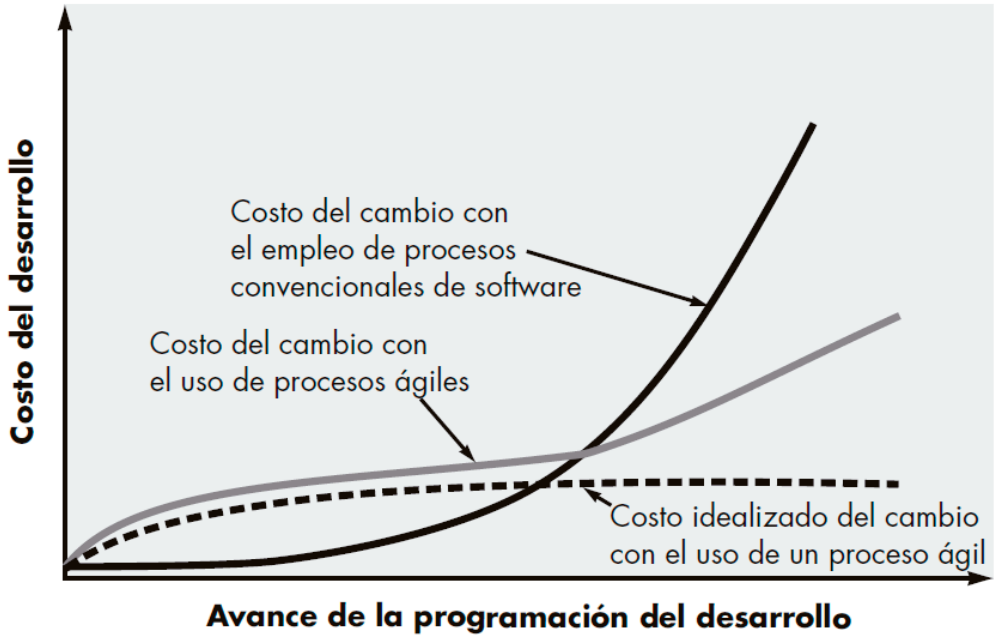
Metodologías de desarrollo ágiles

https://goo.gl/VhBq9y

TEMA 21: "Beneficios de aplicar metodologías ágiles"

El descontento con estas metodologías tradicionales condujo a algunos desarrolladores de software a proponer nuevos "métodos ágiles", los cuales permitieron que el equipo de desarrollo se enfocara en el software en lugar del diseño y la documentación. Los métodos ágiles se apoyan universalmente en el enfoque incremental para la especificación, el desarrollo y la entrega del software. Son más adecuados para el diseño de software en que los requerimientos del mismo cambian, por lo general, rápidamente durante el proceso de desarrollo. Tienen la intención de entregar con prontitud el software a los clientes, quienes entonces propondrán requerimientos nuevos y variados para incluir en posteriores iteraciones del software. Se dirigen a simplificar el proceso, al evitar, el trabajo con valor dudoso a largo plazo, y a eliminar documentación que quizá nunca se emplee. Una de las características más atractivas del enfoque ágil es su capacidad de reducir los costos del cambio durante el proceso del software.

Los defensores de la agilidad (por ejemplo, Beck) afirman que un proceso ágil bien diseñado "aplana" el costo de la curva de cambio lo que permite que el equipo de software haga cambios en una fase tardía de un proyecto de software sin que haya un efecto notable en el costo y en el tiempo. La siguiente figura ilustra este planteamiento.



Fuente: Ingeniería del Software: Un enfoque práctico Roger S. Pressman 7ª Edición McGraw Hill @2010. Página 57

- ## EJEMPLIFICACIÓN

El ambiente moderno de negocios que genera sistemas basados en computador y productos de software evoluciona rápida y constantemente. Las metodologías ágiles representan una alternativa razonable a las metodologías de software tradicional para ciertas clases de software y en algunos tipos de proyectos. Asimismo, se ha demostrado que concluye con rapidez sistemas exitosos.

- ## SITIOS DE INTERÉS

A continuación, se presenta el siguiente enlace como apoyo a los contenidos revisados.

5 beneficios de aplicar metodologías ágiles en el desarrollo de software

https://goo.gl/Ht9swv

TEMA 22: "El manifiesto ágil"

La filosofía detrás de las metodologías ágiles se refleja en el manifiesto ágil, que acordaron muchos de los desarrolladores líderes de estos métodos. Este manifiesto afirma que:
"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
Esto es, aunque exista valor en los objetos a la derecha, valoraremos más los de la izquierda"

- ### EJEMPLIFICACIÓN

Como ejemplo de la aplicación del manifiesto ágil, detallaremos los principios aplicables a las metodologías ágiles.

1. La prioridad más alta es satisfacer al cliente a través de la entrega pronta y continua de software valioso.
2. Son bienvenidos los requerimientos cambiantes, aun en una etapa avanzada del desarrollo. Los procesos ágiles dominan el cambio para provecho de la ventaja competitiva del cliente.
3. Entregar con frecuencia software que funcione, de dos semanas a un par de meses, de preferencia lo más pronto que se pueda.
4. Las personas de negocios y los desarrolladores deben trabajar juntos, a diario y durante todo el proyecto.
5. Hay que desarrollar los proyectos con individuos motivados. Debe darse a éstos el ambiente y el apoyo que necesiten, y confiar en que harán el trabajo.
6. El método más eficiente y eficaz para transmitir información a los integrantes de un equipo de desarrollo, y entre éstos, es la conversación cara a cara.
7. La medida principal de avance es el software que funciona.
8. Los procesos ágiles promueven el desarrollo sostenible. Los patrocinadores, desarrolladores y usuarios deben poder mantener un ritmo constante en forma indefinida.
9. La atención continua a la excelencia técnica y el buen diseño mejora la agilidad.
10. Es esencial la simplicidad: el arte de maximizar la cantidad de trabajo no realizado.
11. Las mejores arquitecturas, requerimientos y diseños surgen de los equipos con organización propia.

El equipo reflexiona a intervalos regulares sobre cómo ser más eficaz, para después afinar y ajustar su comportamiento en consecuencia.

- ## SITIOS DE INTERÉS

A continuación, se presenta el siguiente enlace como apoyo a los contenidos revisados.

El manifiesto ágil

https://goo.gl/anBvYc

TEMA 23: "¿Qué es SCRUM?"

Como cualquier otro proceso de desarrollo de software profesional, el desarrollo ágil tiene que administrarse de tal modo que se busque el mejor uso del tiempo y de los recursos disponibles para el equipo. Esto requiere un enfoque diferente a la administración del proyecto, que se adapte al desarrollo incremental y a las fortalezas particulares de las metodologías ágiles. Scrum es una metodología ágil general, pero su enfoque está en la administración iterativa del desarrollo, y no en aspectos técnicos específicos para metodologías de desarrollo de software ágil. La siguiente figura muestra el ciclo *sprint* de administración de Scrum.

Ciclo *sprint*

Fuente: Ingeniería del Software 7ª edición. Ian Somerville. Pág. 73

Principios y roles de Scrum

Los principios por los que se guía Scrum son: transparencia, coraje, respeto, foco y compromiso para realizar el mejor trabajo posible en todo momento. Entre las personas que participan de Scrum se reparten los siguientes roles:

- Stakeholders: es el público destinatario del desarrollo, las personas a las que beneficiará el producto-servicio producido. Intervienen en el momento de revisar cada sprint (Sprint Review).
- Product Owner (PO): por su cercanía con los Stakeholders, y su conocimiento de negocio, es la persona que se encarga de mantener actualizada y priorizada la pila de producto o backlog con los requisitos (historias de usuario).
- ScrumMaster (o Facilitador): es la persona que crea las condiciones favorables para que el equipo Scrum logre cumplir con el sprint; vela por el cumplimiento de las normas acordadas en la fase de inicio (inception), cuida a las personas que forman parte del proyecto y ayuda en la toma de decisiones y la resolución de conflictos.

- **Equipo de desarrollo:** equipo de personas multidisciplinar y auto organizado que asumen la responsabilidad de entregar el producto.

Equipo Scrum: equipo que gestiona la planificación de los sprint y está compuesto por el PO y el equipo de desarrollo.

- ## EJEMPLIFICACIÓN

La metodología ágil Scrum no indica el uso de prácticas específicas de programación. Esta metodología no prescribe el uso, como la programación en pares y el desarrollo de primera prueba. Por lo tanto, puede usarse con enfoques ágiles más técnicos, como XP, para ofrecer al proyecto un marco administrativo.

- ## SITIOS DE INTERÉS

A continuación, se presenta el siguiente enlace como apoyo a los contenidos revisados.

Proyectos ágiles

https://proyectosagiles.org/que-es-scrum/

TEMA 24: "Beneficios del SCRUM"

Scrum acentúa el uso de un conjunto de patrones de proceso del software que han demostrado ser eficaces para proyectos con plazos de entrega muy apretados, requerimientos cambiantes y negocios críticos. Cada uno de estos patrones de proceso define un grupo de acciones de desarrollo:

- *Retraso:* lista de prioridades de los requerimientos o características del proyecto que dan al cliente un valor del negocio. Es posible agregar en cualquier momento otros aspectos al retraso (ésta es la forma en la que se introducen los cambios). El gerente del proyecto evalúa el retraso y actualiza las prioridades según se requiera.
- *Sprints:* consiste en unidades de trabajo que se necesitan para alcanzar un requerimiento definido en el retraso que debe ajustarse en una caja de tiempo predefinida (lo común son 30 días). Durante el sprint no se introducen cambios (por ejemplo, aspectos del trabajo retrasado). Así, el sprint permite a los miembros del equipo trabajar en un ambiente de corto plazo, pero estable.
- *Reuniones Scrum:* son reuniones breves (de 15 minutos, por lo general) que el equipo Scrum efectúa a diario. Hay tres preguntas clave que se pide que respondan todos los miembros del equipo:
    - ¿Qué hiciste desde la última reunión del equipo?
    - ¿Qué obstáculos estás encontrando?
    - ¿Qué planeas hacer mientras llega la siguiente reunión del equipo?

Un líder del equipo, llamado *maestro Scrum*, dirige la junta y evalúa las respuestas de cada persona.

La junta Scrum ayuda al equipo a descubrir los problemas potenciales tan pronto como sea posible. Asimismo, estas juntas diarias llevan a la "socialización del conocimiento", con lo que se promueve una estructura de equipo con organización propia.

Demostraciones preliminares: entregar el incremento de software al cliente de modo que la funcionalidad que se haya implementado pueda demostrarse al cliente y éste pueda evaluarla. Es importante notar que las demostraciones preliminares no contienen toda la funcionalidad planeada, sino que éstas se entregarán dentro de la caja de tiempo establecida.

• EJEMPLIFICACIÓN

Los principales beneficios de Scrum pueden resumirse de la siguiente manera:

- **Entrega mensual (o quincenal) de resultados** (los requisitos más prioritarios en ese momento, ya completados) lo cual proporciona las siguientes ventajas:
    - Gestión regular de las expectativas del cliente y basada en resultados tangibles.
    - Resultados anticipados (*time to market*).
    - Flexibilidad y adaptación respecto a las necesidades del cliente, cambios en el mercado, etc.
    - Gestión sistemática del Retorno de Inversión (ROI).
    - Mitigación sistemática de los riesgos del proyecto.
- Productividad y calidad.
- Alineamiento entre el cliente y el equipo de desarrollo.
- Equipo motivado.

• SITIOS DE INTERÉS

A continuación, se presenta el siguiente enlace como apoyo a los contenidos revisados.

Beneficios de proyectos ágiles

https://proyectosagiles.org/beneficios-de-scrum/

TEMA 25: "Proceso SCRUM"

Existen tres fases en Scrum. La primera es la planeación del bosquejo, donde se establecen los objetivos generales del proyecto y el diseño de la arquitectura de software. A esto le sigue una serie de ciclos *sprint*, donde cada ciclo desarrolla un incremento del sistema. Finalmente, la fase de cierre del proyecto concluye el proyecto, completa la documentación requerida, como los marcos de ayuda del sistema y los manuales de usuario, y valora las lecciones aprendidas en el proyecto. La característica innovadora de Scrum es su fase central, a saber, los ciclos *sprint*. Un *sprint* de Scrum es una unidad de planeación en la que se valora el trabajo que se va a realizar, se seleccionan las particularidades por desarrollar y se implementa el software. Al final de un *sprint*, la funcionalidad completa se entrega a los participantes. Las características clave de este proceso son las siguientes:

1. Los *sprints* tienen longitud fija, por lo general de dos a cuatro semanas. Corresponden al desarrollo de una liberación del sistema en XP.
2. El punto de partida para la planeación es la cartera del producto, que es la lista de trabajo por realizar en el proyecto. Durante la fase de valoración del *sprint*, esto se revisa, y se asignan prioridades y riesgos. El cliente interviene estrechamente en este proceso y al comienzo de cada *sprint* puede introducir nuevos requerimientos o tareas.
3. La fase de selección incluye a todo el equipo del proyecto que trabaja con el cliente, con la finalidad de seleccionar las características y la funcionalidad a desarrollar durante el *sprint*.
4. Una vez acordado, el equipo se organiza para desarrollar el software. Con el objetivo de revisar el progreso y, si es necesario, volver a asignar prioridades al trabajo, se realizan reuniones diarias breves con todos los miembros del equipo. Durante esta etapa, el equipo se aísla del cliente y la organización, y todas las comunicaciones se canalizan a través del llamado "maestro de Scrum". El papel de este último es proteger al equipo de desarrollo de distracciones externas. La forma en que el trabajo se realiza depende del problema y del equipo. A diferencia de XP, Scrum no hace sugerencias específicas sobre cómo escribir requerimientos, desarrollar la primera prueba, etcétera. Sin embargo, dichas prácticas XP se usan cuando el equipo las considera adecuadas.

Al final del *sprint*, el trabajo hecho se revisa y se presenta a los participantes. Luego comienza el siguiente ciclo de *sprint*.

- ### EJEMPLIFICACIÓN

En la Web existen muchos ejemplos del uso exitoso del Scrum. En ellos se discute su uso exitoso en un entorno de desarrollo de software para telecomunicaciones y mencionan sus ventajas del modo siguiente:

1. El producto se desglosa en un conjunto de piezas manejables y comprensibles.
2. Los requerimientos inestables no retrasan el progreso.
3. Todo el equipo tiene conocimiento de todo y, en consecuencia, se mejora la comunicación entre el equipo.
4. Los clientes observan la entrega a tiempo de los incrementos y obtienen retroalimentación sobre cómo funciona el producto.

Se establece la confianza entre clientes y desarrolladores, a la vez que se crea una cultura positiva donde todos esperan el triunfo del proyecto.

- ## SITIOS DE INTERÉS

A continuación, se presenta el siguiente enlace como apoyo a los contenidos revisados.

Las 5 etapas en los “Sprints” de un desarrollo Scrum

https://goo.gl/yeHPhS

TEMA 26: “Requisitos para utilizar SCRUM”

El requisito central detrás de Scrum es que debe autorizarse a todo el equipo para tomar decisiones, de modo que se evita deliberadamente el término “administrador del proyecto”. En lugar de ello, el “maestro de Scrum” es el facilitador que ordena las reuniones diarias, rastrea el atraso del trabajo a realizar, registra las decisiones, mide el progreso del atraso, y se comunica con los clientes y administradores fuera del equipo. Todo el equipo asiste a las reuniones diarias, que en ocasiones son reuniones en las que los participantes no se sientan, para hacerlas breves y enfocadas. Durante la reunión, todos los miembros del equipo comparten información, describen sus avances desde la última reunión, los problemas que han surgido y los planes del día siguiente. Ello significa que todos en el equipo conocen lo que acontece y, si surgen problemas, replantean el trabajo en el corto plazo para enfrentarlo. Todos participan en esta planeación; no hay dirección descendente desde el maestro de Scrum.

- ## EJEMPLIFICACIÓN

Los siguientes puntos son de especial importancia para la implantación de una gestión ágil de proyectos como Scrum:

- Cultura de empresa basada en trabajo en equipo, delegación, creatividad y mejora continua.
- Compromiso del cliente en la dirección de los resultados del proyecto, gestión del ROI y disponibilidad para poder colaborar.
- Compromiso de la Dirección de la organización para resolver problemas endémicos y realizar cambios organizativos, formando equipos auto gestionados y multidisciplinares y fomentando una cultura de gestión basada en la colaboración y en la facilitación llevada a cabo por líderes al servicio del equipo.
- Compromiso conjunto y colaboración de los miembros del equipo.
- Relación entre proveedor y cliente basada en ganar-ganar, colaboración y transparencia.
- Facilidad para realizar cambios en el proyecto.
- Tamaño de cada equipo entre 5 y 9 personas (con técnicas específicas de planificación y coordinación cuando varios equipos trabajan en el mismo proyecto).
- Equipo trabajando en un mismo espacio común para maximizar la comunicación.
- Dedicación del equipo a tiempo completo.

- Estabilidad de los miembros del equipo

SITIOS DE INTERÉS

A continuación, se presenta el siguiente enlace como apoyo a los contenidos revisados.

Requisitos de Scrum

https://proyectosagiles.org/requisitos-de-scrum/

TEMA 27: "Artefactos del SCRUM"

El Backlog de Producto es un listado dinámico y públicamente visible para todos los involucrados en el proyecto. En él, el Dueño de Producto, mantiene una lista actualizada de requerimientos funcionales para el software. Esta lista, representa "qué es lo que se pretende" pero sin mencionar "cómo hacerlo", ya que esta será tarea del Scrum Team. El Backlog de Sprint es la recopilación sintética de ítems del Backlog de Producto, negociados entre el Dueño de Producto y el Scrum Team en la ceremonia de planificación, reunión que se realiza al comienzo del Sprint. Esta recopilación, que durante la planificación ha sido propuesta por el Dueño de Producto y ya negociada, es aquella que el Scrum Team se compromete a construir durante el Sprint en curso. El incremento de funcionalidad, es el que el equipo entrega al finalizar el Sprint. El mismo debe asemejarse a un "software funcionando", permitiendo implementarse operativamente sin restricciones en un ambiente productivo.

EJEMPLIFICACIÓN

Algunos ejemplos de artefactos de Scrum son:

- Product Vision Board
  - Visión
  - Grupos de usuarios
  - Necesidades
  - Funcionalidades
  - Beneficios
- Product Backlog Board
  - Técnica Personas
  - Restricciones
    - Lo que no nos deja dormir (Riesgos)
    - Sería genial que pasara (Oportunidades)
    - Prioridades (alcance, tiempo, costo, calidad, usabilidad, adaptabilidad, seguridad, etc.)

        - Incluido en el alcance
        - Fuera del alcance
    - Modelos /Diagramas
        - Despliegue
        - Procesos
        - Mockups
- User Story Map / Construcción del Release Plan
- Calculo del tiempo y costo de la construcción del producto, empleando una hoja de cálculo de mi autoría

Construcción del **Product Backlog** basado en requerimientos con historias de usuario

SITIOS DE INTERÉS

A continuación, se presenta el siguiente enlace como apoyo a los contenidos revisados.

Artefactos de Scrum

https://cristinaramosvega.com/z-los-artefactos-scrum/

TEMA 28: "SCRUM y etapas de un proyecto de software"

Las actividades que se llevan a cabo en Scrum son las siguientes:

1. Planificación de la iteración (Sprint Planning). El cliente presenta al equipo la lista de requisitos priorizada del producto o proyecto. El equipo examina la lista, pregunta al cliente las dudas que le surgen, añade más condiciones de satisfacción y selecciona los objetivos/requisitos más prioritarios que se compromete a completar en la iteración, de manera que puedan ser entregados si el cliente lo solicita. Se realiza en un Timebox de cómo máximo 4 horas.
2. Ejecución de la iteración (sprint). Un proyecto se ejecuta en bloques temporales cortos y fijos (iteraciones de un mes natural y hasta de dos semanas).
3. Reunión diaria de sincronización del equipo (Scrum Daily Meeting). Cada miembro del equipo inspecciona el trabajo que el resto está realizando.
4. Demostración de requisitos completados (Sprint Review). Reunión informal donde el equipo presenta al cliente los requisitos completados en la interacción.

Retrospectiva (Sprint Retrospective). El equipo analiza cómo ha sido su manera de trabajar durante la iteración, por qué está consiguiendo o no los objetivos a que se comprometió al inicio de la iteración y por qué el incremento de producto que acaba de demostrar al cliente era lo que él esperaba o no.

EJEMPLIFICACIÓN

Como resultado de una **demanda de los stakeholders surge la visión de un producto o servicio**. En este momento, **comienza la fase de inception o inicio del producto/servicio** para redactar los requisitos iniciales del desarrollo, designar al equipo Scrum, seleccionar la tecnología adecuada, acordar normas de funcionamiento, realizar la definición de producto terminado y, por último, construir una pila de producto o **Product Backlog** inicial.

Tal y como hemos explicado anteriormente, es tarea del Product Owner, en coordinación con el equipo Scrum, **ordenar y refinar constantemente los requisitos de cada Sprint Backlog** en función de las prioridades y el valor que aporta a los stakeholders. Una vez designados los roles, establecidas las normas y redactado el backlog, llega el **primer Sprint Planning**, que no es otra cosa que un encuentro para indicar qué historias de usuario o requisitos se incluirán en el Sprint Backlog (lista de historias de usuario que se desarrollarán durante el sprint). Es aconsejable que éstas se encuentren reducidas a su mínima expresión.

SITIOS DE INTERÉS

A continuación, se presenta el siguiente enlace como apoyo a los contenidos revisados.

Scrum, metodología ágil para desarrollo de proyectos (I)

https://goo.gl/jwxxkY

TEMA 29: "Herramientas de apoyo a las metodologías ágiles"

Las herramientas de apoyo tienen un beneficio para los equipos ágiles. Se deben favorecer el uso de herramientas que permiten el flujo rápido de entendimiento. Algunas de estas herramientas son sociales y comienzan incluso en la etapa de reclutamiento. Otras son tecnológicas y ayudan a que los equipos distribuidos simulen su presencia física. Muchas herramientas son físicas y permiten que las personas las manipulen en talleres". Prácticamente todos los modelos de proceso ágil son elementos clave en la contratación del personal adecuado (reclutamiento), la colaboración en equipo, la comunicación con los participantes y la administración indirecta. Las herramientas que se abocan a dichos aspectos son factores críticos para el éxito de la agilidad. Por ejemplo, una herramienta de reclutamiento tal vez sea el requerimiento de que un prospecto a miembro del equipo pase algunas horas programando en pareja con alguien que ya es integrante del equipo. El "ajuste" se evalúa de inmediato. Las "herramientas" de colaboración y comunicación por lo general son de baja tecnología e incorporan cualquier mecanismo (pizarrones, tableros, tarjetas y notas adheribles) que provea información y coordinación entre los desarrolladores ágiles.

EJEMPLIFICACIÓN

Por ejemplo, la comunicación activa puede lograse por medio de la dinámica del equipo (por ejemplo, la programación en parejas), mientras que la comunicación pasiva se consigue con "radiadores de información" (un tablero que muestre el estado general de los distintos componentes de un incremento). Las herramientas de administración de proyectos no ponen el énfasis en la gráfica de Gantt y la sustituyen con otras de valor agregado o gráficas de pruebas creadas *versus* pasadas; otras herramientas ágiles se utilizan para optimizar el ambiente en el que trabaja el equipo ágil (por ejemplo, áreas más eficientes para reunirse), mejoran la cultura del equipo por medio de cultivar las interacciones sociales (equipos con algo en común), dispositivos físicos (pizarrones electrónicos) y el mejoramiento del proceso (por ejemplo, la programación por parejas o la caja de tiempo).

SITIOS DE INTERÉS

A continuación, se presenta el siguiente enlace como apoyo a los contenidos revisados.

Gestión de proyectos Agiles y Scrum: una selección de plataformas útiles

https://goo.gl/779tTk

TEMA 30: "Historias de usuario"

Cuando los requerimientos *se expresan informalmente.* Las *historias de usuario* y las pruebas de aceptación son la única manifestación explícita, por ejemplo, en XP. Los críticos afirman que es frecuente que se necesite un modelo o especificación más formal para garantizar que se descubran las omisiones, inconsistencias y errores antes de que se construya el sistema. Los defensores contraatacan diciendo que la naturaleza cambiante de los requerimientos vuelve obsoletos esos modelos y especificaciones casi tan pronto como se desarrollan. Las historias de usuario siguen los principios básicos de las metodologías ágiles porque:

- Potencian la participación del equipo en la toma de decisiones
- Se crean y evolucionan a medida que el proyecto avanza
- Son peticiones concretas y pequeñas
- Contienen la información imprescindible
- Apoyan la colaboración y conversación entre los miembros del equipo

La historia de usuario consta de tres partes, estas son:

- Tarjeta: Una descripción escrita en lenguaje de negocio que sirve como identificación y recordatorio del requerimiento y ayuda a la planificación mediante la priorización.
- Conversación: El dialogo que ocurre entre los miembros del equipo y el *dueño del producto* permite aclarar dudas sobre la historia de usuario y sus detalles. Esta es la parte

más importante de la historia de usuario.

Confirmación: Que pruebas se llevarán a cabo para poder decir que la historia de usuario se ha completado con éxito. Esta parte puede añadirse a la conversación.

• EJEMPLIFICACIÓN

Dentro de los métodos usados en la construcción de la historia de usuario tenemos el método *Invest.* Algunas características de este método son:

- **Independiente**: cada historia de usuario pueda ser planificada e implementada en cualquier orden. No dependen unas de otras, si ocurre se deben dividir o combinar.
- **Negociable**: las historias deben ser negociables ya que sus detalles serán acordados por el cliente/usuario y el equipo durante la fase de "conversación".
- **Valor**: una historia de usuario tiene que ser valiosa para el cliente o el usuario.
- **Estimable**: una buena historia de usuario debe ser estimada con la precisión suficiente para ayudar al cliente o usuario a priorizar y planificar su implementación. Si no podemos estimarla debemos incidir en la fase de conversación o dividirla.
- **Pequeña**: pequeñas. Solemos hacerlas de tal modo que ocupen como máximo un sprint.
- **Testeable**: la historia de usuario debe poderse probar (se ha trabajado antes en la fase "confirmación" de la historia de usuario). Tanto el usuario como el equipo de desarrollo tienen que poder probarla para saber cuándo está finalizada.

• SITIOS DE INTERÉS

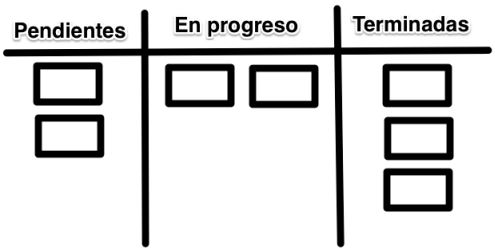
A continuación, se presenta el siguiente enlace como apoyo a los contenidos revisados.

Historias de usuario

https://goo.gl/SEZM6j

TEMA 31: "Tablero Kanban"

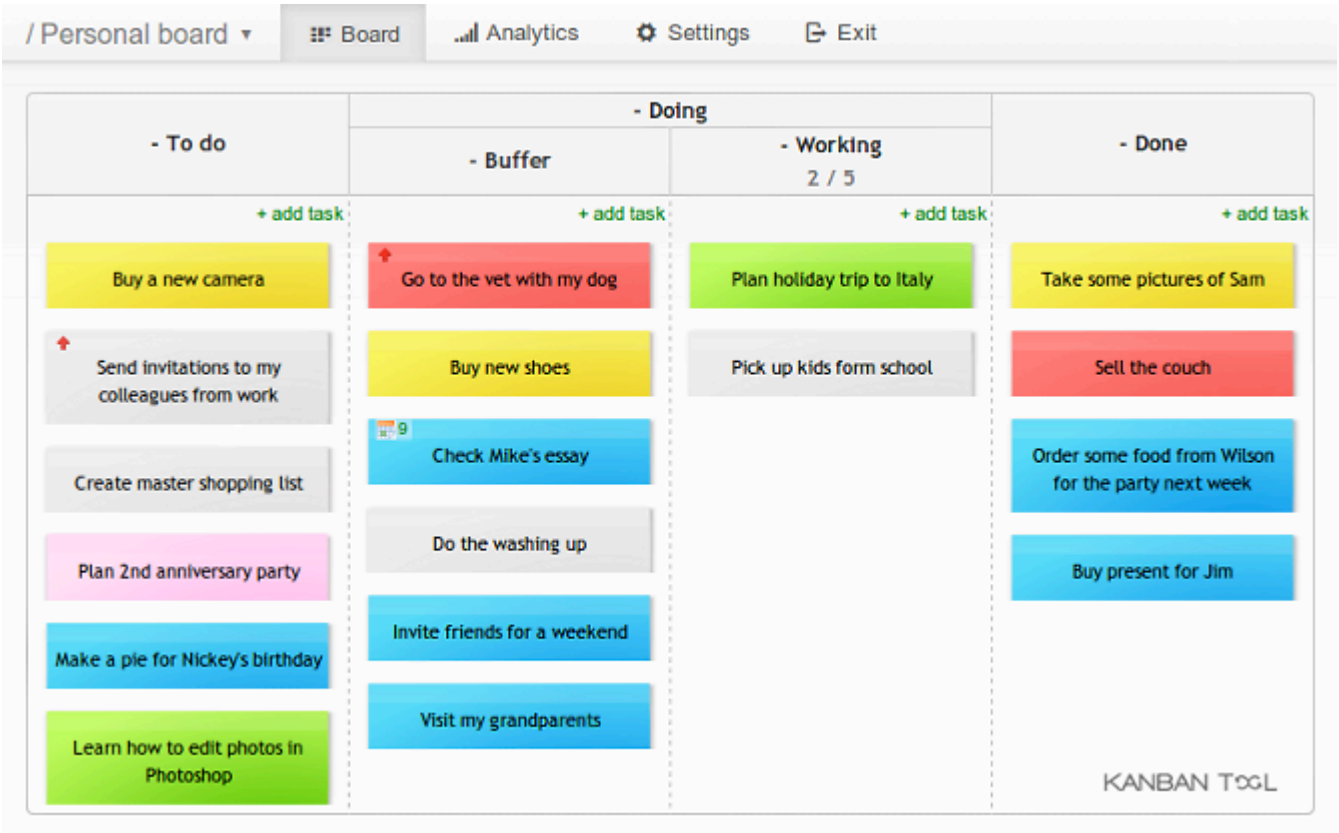
El concepto de **Kanban es tan sencillo como poderoso**. En un tablero dibujaremos tantas columnas como pasos por los que deba de pasar una tarea en nuestro flujo de trabajo. Por ejemplo, en desarrollo las columnas más comunes son ToDo (pendientes), que son las tareas pendientes de ser iniciadas; Doing (en progreso), que son aquellas que se están realizando; Testing, como indica su nombre son las tares que están en proceso de QA; y finalmente la columna de Done (terminadas), aquellas que están acabadas (el difícil concepto de Done Agile*).*



- ***EJEMPLIFICACIÓN***

Un tablero Kanban de equipo básico

Los tableros de equipo usan columnas de amortiguamiento para señalizar que las tareas están listas para moverse a la siguiente etapa del flujo de trabajo, en la mayoría de las veces por otros miembros del equipo. Esto contribuye a un flujo de trabajo más suave y a un gran ahorro de tiempo en comunicación.



Fuente: http://kanbantool.com/es/tablero-kanban-online

- ## SITIOS DE INTERÉS

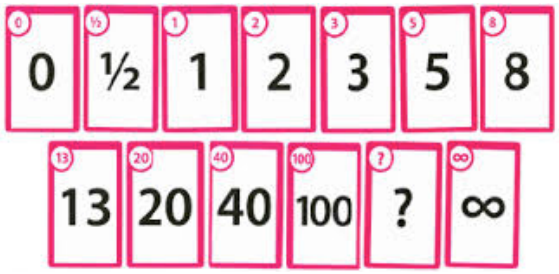
A continuación, se presenta el siguiente enlace como apoyo a los contenidos revisados.

Ejemplos de tableros kanban

https://goo.gl/LtHLZ5

TEMA 32: "Planning poker"

Planning poker o planificación poker es una técnica muy sencilla y eficaz, en la que todas las personas comprometidas en el desarrollo de un producto software opinan para estimar el esfuerzo de desarrollar cada una de las historias de usuario de un sprint. El **planning poker** se utiliza mucho en las planificaciones de sprint en Scrum, aunque realmente se puede utilizar en muchos otros contextos. Todos los miembros del equipo participan activamente, así que juntar al equipo para tomar decisiones es algo totalmente lógico y positivo para el proyecto. En esta técnica el equipo con una baraja de Poker modificada y se hacen rondas de estimación con ayuda de estas cartas.
Para una sesión de estimación ágil será necesario que cada participante tenga una baraja de Planning Poker. En cada baraja hay una pseudo-secuencia de Fibonacci modificada. Se recomienda solo las primeras cartas para **evitar estimar tareas demasiado grandes**. Así cada participante tendrá las siguientes cartas: 0,1/2, 1, 2, 3, 5, ? e infinito. El cero significa que la historia ya está hecha o no requiere ningún esfuerzo, el interrogante significa que nos falta información para estimar esa historia o tarea y el infinito es que es demasiado grande y hay que trocearla.



La dinámica de la técnica funciona de la siguiente manera:

- Todo el equipo "se encierra" para estimar, y todos conocen lo que se va a estimar. Si hay gente que no está al tanto de lo que se va a estimar la sesión debe comenzar con una explicación y una **sesión de preguntas** para despejar cualquier duda sobre las historias que se van a estimar.
- Una por una se leen y discuten las historias de usuario. Una vez todos tienen claro en qué consiste cada uno elige una carta en función del esfuerzo que prevé requerirá esa historia. No es posible seleccionar un valor no incluido en la baraja. **Solo estiman los que después desarrollan**.

- Si no hay consenso (lo normal con más de 2 participantes) se abre la discusión. No muy larga, por ejemplo, se puede hacer que explique su elección el que tiene la puntuación más baja y más alta. Se repite la estimación nuevamente en busca de consenso. Si no se consigue a la segunda se vuelve a discutir. A la tercera si no hay consenso se escoge o bien la media o bien el máximo (mejor el máximo).

Al final de la sesión el resultado es una **estimación consensuada y validada por todo el equipo** para cada una de las historias o tareas seleccionadas. Si se trabaja con Sprints lo habitual es estimar alto nivel cuando los elementos entran en el backlog, y nuevamente cuando se realiza el sprint planning o backlog grooming si se hace por separado para estimar las tareas a bajo nivel.

EJEMPLIFICACIÓN

En planning poker la secuencia de números no es lineal. Por ejemplo, no hay nada entre 40 y 100. ¿Por qué? Esto es para evitar una falsa sensación de exactitud para las estimaciones de tiempo más grandes. Si una historia se estima aproximadamente en 20 puntos, no es relevante discutir si deberían ser 20, 18 o 21. Todo lo que sabemos es que es una historia grande y es difícil de estimar. Así que 20 es nuestra idea aproximada.
¿Quieres estimaciones más detalladas? Divide la historia en historias más pequeñas y trata de estimar las historias pequeñas.
Y no, no se puede hacer trampas combinando un 5 y un 2 para hacer un 7. Tienes que escoger entre 5 y 8, no hay 7.
Algunas cartas especiales sobre las que hablar:

- 0 = “esta historia ya está hecha” o “esta historia es prácticamente nada, apenas unos minutos de trabajo”.
- ¿ = “no tengo ni la más remota de las ideas. Nada”.
- ∞ = “estoy demasiado cansado para pensar. Tomemos un descanso”.

SITIOS DE INTERÉS

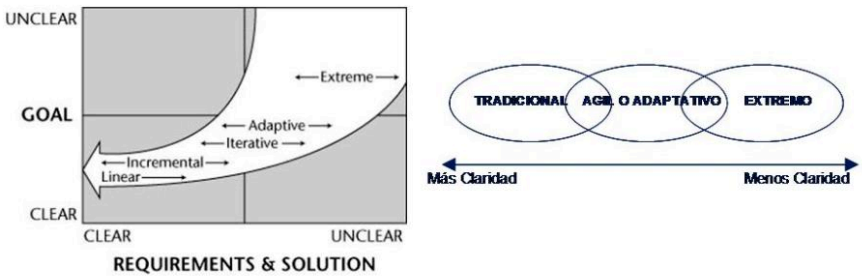
A continuación, se presenta el siguiente enlace como apoyo a los contenidos revisados.

Planning poker

https://goo.gl/Z3wRXC

TEMA 33: "Metodología a utilizar en función del proyecto de software"

De acuerdo a las necesidades de proyecto podemos mostrar (o graficar) que existen proyectos en los cuales los objetivos del negocio, los requerimientos, y la solución a implementarse están más o menos claros de entrada, por lo que se podría planificar todo al principio y sin mayores inconvenientes. En los otros extremos cuando no existe tanta claridad debería utilizarse una aproximación adaptativa o ágil y cuando hay mucha incertidumbre se recurre a la programación extrema (XP). Las metodologías ágiles proponen una planificación adaptativa utilizando un ciclo de vida de proyecto iterativo e incremental opuesto al tradicional que utilizaría el ciclo de vida cascada. Por otro lado, las metodologías ágiles tratan de bajar los decibeles a todo formalismo que no agrega valor, documentando lo menos posible.

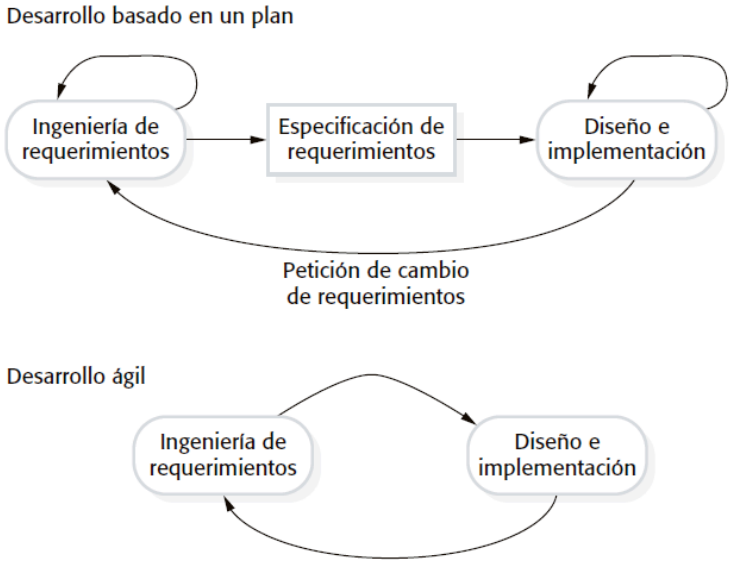


Fuente: https://pmqlinkedin.wordpress.com/about/metodologia-tradicional-o-agil/

Algunas metodologías ágiles vienen a triunfar donde la metodología "tradicional" había fracasado. Para otros, las metodologías ágiles son sólo una excusa para desarrollar software sin planes, diseño ni documentación. Ambos extremos son erróneos; por una parte, varias prácticas ágiles tienen nichos donde funcionan muy bien, pero hay otros ambientes donde pierden su efectividad o deben ser modificadas para lograr el objetivo. Tampoco es correcto que lo ágil se contraponga con la aplicación de mejores prácticas en gestión de proyectos.

- ### EJEMPLIFICACIÓN

Dependiendo del proyecto, en un enfoque basado en un plan (o tradicional), la iteración ocurre dentro de las actividades con documentos formales usados para comunicarse entre etapas del proceso. Por ejemplo, los requerimientos evolucionarán y, a final de cuentas, se producirá una especificación de aquéllos. Esto entonces es una entrada al proceso de diseño y la implementación. Por otra parte, en un enfoque ágil, la iteración ocurre a través de las actividades. Por lo tanto, los requerimientos y el diseño se desarrollan en conjunto, no por separado.



Fuente: Ingeniería del Software 7a edición. Ian Somerville. Pág. 63

- ## SITIOS DE INTERÉS

A continuación, se presenta el siguiente enlace como apoyo a los contenidos revisados.

Proyectos ágiles

https://goo.gl/oYxd5r