MÓDULO 3 | METODOLOGÍA PARA EL DESARROLLO DE SOFTWARE

1. CONCEPTOS BÁSICOS

¿Qué es un Método?
Un Método se compone de diversos aspectos que nos permitirán conseguir una meta o lograr un objetivo. Se define más claramente como un conjunto de herramientas, las cuales utilizadas mediante las técnicas correctas, permiten la ejecución de procesos que nos llevarán a cumplir los objetivos que buscamos. Habitualmente son aplicables a una (o algunas) actividades del proceso de desarrollo, por ejemplo, suele hablarse de métodos de análisis y/o diseño.

> **MÉTODO** es un conjunto de herramientas, técnicas y procesos que facilitan la obtención de un objetivo.
>
> **METODOLOGÍA**, es un proceso detallado y completo a seguir sistemáticamente para idear, implementar y mantener un producto de software. Se basa en una combinación de los modelos de proceso genéricos.

¿Qué es una Metodología?
En el desarrollo de software, una metodología hace cierto énfasis al entorno en el cuál se plantea y estructura el desarrollo de un sistema. Existe una gran cantidad de metodologías de programación que se han utilizado desde los tiempos atrás y que con el paso del tiempo han ido evolucionando. Esto se debe principalmente a que no todos los sistemas de la información, son compatibles con todas las metodologías, pues el ciclo de vida del software puede ser variable. Por esta razón, es importante que dependiendo del tipo de software que se vaya a desarrollar, se identifique la metodología más idónea para el diseño de software.

¿Qué es una Metodología para Desarrollo de Software?
La metodología para el desarrollo de software es un modo sistemático de realizar, gestionar y administrar un proyecto para llevarlo a cabo con **altas posibilidades de éxito.** Esta sistematización nos indica cómo dividiremos un gran proyecto en módulos más pequeños, llamados etapas, y las acciones que correspondan en cada una de ellas, nos ayuda a definir entradas y salidas para cada una de las etapas y, sobre todo, normaliza el modo en que administraremos el proyecto.

Entonces, una metodología para el desarrollo de software son los procesos a seguir sistemáticamente para idear, implementar y mantener un producto de software desde que surge la necesidad del producto hasta que cumplimos el objetivo por el cual fue creado.

Podríamos decir que consiste principalmente en hacer uso de diversas herramientas, técnicas, métodos y modelos para el desarrollo. Regularmente este tipo de metodologías, deben estar documentadas, para que los programadores que estarán dentro de la planeación del proyecto, comprendan perfectamente la metodología y en algunos casos el ciclo de vida del software que se pretende seguir.

Aunque actualmente existe mucha variedad en metodologías de programación, todas están basadas en ciertos enfoques generalistas que se crearon hace muchos años, al principio de nuestra era tecnológica y son las que veremos a continuación.

Una metodología debe definir con precisión los objetos, roles y actividades involucrados, junto con prácticas y técnicas recomendadas, guías de adaptación de la metodología al proyecto, guías para uso de herramientas de apoyo, etc.

2. TIPOS DE METODOLOGÍAS

La comparación y/o clasificación de metodologías no es una tarea sencilla, debido a la diversidad de propuestas y diferencias en el grado de detalle, información disponible y alcance de cada una de ellas.

Si tomamos como criterio las anotaciones utilizadas para especificar actividades de análisis y diseño, podemos clasificar las metodologías en dos grupos: **Metodologías Estructuradas** y **Metodologías Orientadas a Objetos**.

Las Metodologías con mayor énfasis en la planificación y control del proyecto, en especificación precisa de requisitos y modelado, se denominan **Metodologías Tradicionales** (o Pesadas).

Las Metodologías, más orientadas a la generación de código con ciclos muy cortos de desarrollo, se dirigen a equipos de desarrollo pequeños, hacen hincapié en aspectos humanos asociados al trabajo en equipo e involucran activamente al cliente en el proceso, se denominan **Metodologías Ágiles**.

TIPOS DE METODOLOGÍAS:
- Estructuradas.
- Orientadas a Objetos.
- Tradicionales.
- Ágiles

METODOLOGÍAS ESTRUCTURADAS

Los **Métodos Estructurados** comenzaron a desarrollarse a fines de los 70's con la Programación Estructurada.

Luego aparecieron Técnicas:
- Para el Diseño (por ejemplo: el diagrama de Estructura).
- Para el Análisis (por ejemplo: Diagramas de Flujo de Datos).

Estas metodologías, son apropiadas en proyectos que utilizan para la implementación, lenguajes de 3ra y 4ta generación.

EJEMPLOS de metodologías estructuradas de ámbito gubernamental:
- MERISE (Francia).

- MÉTRICA (España).
- SSADM (Reino Unido).

EJEMPLOS de propuestas de métodos estructurados en el ámbito académico:

- Gane & Sarson
- Ward & Mellor
- Yourdon & DeMarco
- Information Engineering.

METODOLOGÍAS ORIENTADAS A OBJETOS

Las **Metodologías Orientadas a Objetos**, van unidas a la evolución de los lenguajes de programación orientada a objeto, los más representativos: a fines de los 60's SIMULA, a fines de los 70's Smalltalk-80, la primera versión de C++ por Bjarne Stroustrup en 1981 y actualmente Java o C# de Microsoft.

A fines de los 80's comenzaron a consolidarse algunos métodos Orientadas a Objeto.

En 1995 Booch y Rumbaugh proponen el Método Unificado, con la ambiciosa idea de conseguir una unificación de sus métodos y notaciones, que posteriormente se reorienta a un objetivo más modesto, para dar lugar al Unified Modeling Language (UML), la notación Orientada a Objetos más popular en la actualidad.

Algunas metodologías orientadas a objetos que utilizan la notación UML son:

- Rational Unified Process (RUP).
- OPEN.
- MÉTRICA (que también soporta la notación estructurada).

METODOLOGÍAS TRADICIONALES o CLASICAS

Las **Metodologías Tradicionales o Clásicas**, son aquellas que están guiadas por una fuerte planificación durante todo el proceso de desarrollo; llamadas también **Metodologías NO Ágiles**, donde se realiza una intensa etapa de análisis y diseño, antes de la construcción del sistema.

Todas las propuestas metodológicas antes indicadas pueden considerarse como metodologías tradicionales.

Ventajas

- Evaluación en cada fase que permite cambios de objetivos
- Funciona bien en proyectos de innovación.
- Es sencillo, ya que sigue los pasos intuitivos necesarios a la hora de desarrollar el software.
- Seguimiento detallado en cada una de las fases.

Desventajas

- La evaluación de riesgos es compleja
- Excesiva flexibilidad para algunos proyectos
- Estamos poniendo a nuestro cliente en una situación que puede ser muy incómoda para él. Nuestro cliente deberá ser capaz de describir y entender a un gran nivel de detalle para poder acordar un alcance del proyecto con él.

Te invitamos a ver el siguiente video:
#1. Que son las metodologias tradicionales en el desarrollo de Software
Por Cristian Henao – Duración: 7:13 min.
https://www.youtube.com/watch?v=i8CPD1dW88k

Cuáles son las Metodologías Tradicionales o Clásicas:

Todas las propuestas metodológicas antes indicadas pueden considerarse como metodologías tradicionales, y enunciaremos algunas más en las que nos enfocaremos en este curso:

- Cascada
- Prototipos
- Espiral
- V

METODOLOGÍAS ÁGILES

Por **Metodologías Ágiles** entendemos a aquellas metodologías de gestión que permiten adaptar la forma de trabajo al contexto y naturaleza de un proyecto, basándose en la flexibilidad y la inmediatez, y teniendo en cuenta las exigencias del mercado y los clientes. Los pilares fundamentales de las metodologías ágiles son el trabajo colaborativo y en equipo.

UN PROCESO ÁGIL ES:

- Incremental.
- Cooperativo.
- Sencillo.
- Adaptable.

Un proceso es ágil, cuando el desarrollo de software es;

- Incremental (entregas pequeñas de software, con ciclos rápidos).
- Cooperativo (cliente y desarrolladores trabajan juntos constantemente, con una cercana comunicación).
- Sencillo (el método en sí mismo es fácil de aprender y modificar, bien documentado).
- Adaptable (permite realizar cambios de último momento).

Te invitamos a ver el siguiente video:
#2. Que son las metodologias ágiles en el desarrollo de Software
Por Cristian Henao – Duración: 4:50 min.
https://www.youtube.com/watch?v=fHKsufzM7qQ

Beneficios de implementar metodologías ágiles

Gracias a la flexibilidad y capacidad de adaptación de las mismas, son muchos los beneficios de incorporar metodologías ágiles a la gestión de las organizaciones. Aquí te contamos los principales:

1. Reducción de costos.

2.Rapidez en la entrega de proyectos.
3. Trabajo en equipo y compromiso de todos los miembros del equipo de trabajo.
4. Mayor calidad en el trabajo y en el producto final (ya sea producto o servicio).

Cuáles son las Metodologías Ágiles:

- Extreme Programming
- Scrum
- Familia de Metodologías Crystal
- Feature Driven Development
- Proceso Unificado Rational, una configuración ágil
- Dynamic Systems Development Method
- Adaptive Software Development
- Open Source Software Development

Pausa con humor

DIFERENCIAS ENTRE METODOLOGÍAS TRADICIONALES Y AGILES

Te invitamos a ver el siguiente video:
Diferencias entre metodologías ágiles y el método tradicional
Por LíderdeProyecto.com - Duración: 6:59 min.
https://www.youtube.com/watch?v=n-N5dL5n-LU

En las metodologías tradicionales el principal problema es que nunca se logra planificar bien el esfuerzo requerido para seguir la metodología. Pero entonces, si logramos definir métricas que apoyen la estimación de las actividades de desarrollo, muchas prácticas de metodologías tradicionales podrían ser apropiadas. El no poder predecir siempre los resultados de cada proceso no significa que estemos frente a una disciplina de azar. Lo que significa es que estamos frente a la necesidad de adaptación de los procesos de desarrollo que son llevados por parte de los equipos que desarrollan software.
Tener metodologías diferentes para aplicar de acuerdo con el proyecto que se desarrolle resulta una idea interesante. Estas metodologías pueden involucrar prácticas tanto de metodologías ágiles como de metodologías tradicionales. De esta manera podríamos tener una metodología por cada proyecto, la problemática sería definir cada una de las prácticas, y en el momento preciso definir parámetros para saber cuál usar.
Es importante tener en cuenta que el uso de un método ágil no vale para cualquier proyecto. Sin embargo, una de las principales ventajas de los métodos ágiles es su peso inicialmente ligero y por eso las personas que no estén acostumbradas a seguir procesos encuentran estas metodologías bastante agradables.

En la tabla que se muestra a continuación aparece una comparativa entre estos dos grupos de metodologías.

Metodologías Tradicionales	Metodologías Ágiles
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
Existe un contrato prefijado	No existe contrato tradicional o al menos es bastante flexible
El cliente interactúa con el equipo de desarrollo mediante reuniones	El cliente es parte del equipo de desarrollo
Grupos grandes y posiblemente distribuidos	Grupos pequeños (< 10 integrantes) y trabajado en el mismo sitio
Más artefactos	Pocos artefactos
Más roles	Pocos roles
La arquitectura del software es esencial y se expresa mediante modelos	Menos énfasis en la arquitectura del software

3. LECTURAS COMPLEMENTARIAS

- Metodologías del Desarrollo de Software
  https://okhosting.com/blog/metodologias-del-desarrollo-de-software/

- UCA – BIBLIOTECA DIGITAL – Metodología de Desarrollo de Software, Maida, Esteban Gabriel y Pacienzia, Julián
  http://bibliotecadigital.uca.edu.ar/repositorio/tesis/metodologias-desarrollo-software.pdf

- UNAM – Metodologías y Procesos de Análisis de Software, Capítulo 2
  http://www.ptolomeo.unam.mx:8080/xmlui/bitstream/handle/132.248.52.100/175/A5%20Cap%C3%ADtulo%202.pdf?sequence=5

MÓDULO 3 | CICLO DE VIDA DEL DESARROLLO DE SOFTWARE (SDLC)

4. CONCEPTOS BÁSICOS

¿Qué es el Ciclo de vida de desarrollo de software o SDLC (Software Development Life Cycle)?
La **ISO** , International Organization for Standardization, en su norma **12207** define al **ciclo de vida de un software** como un marco de referencia que contiene las actividades y las tareas involucradas en el desarrollo, la explotación, y el mantenimiento de un producto de software, abarcando desde la definición hasta la finalización de su uso.

> El término **CICLO DE VIDA** del software describe el desarrollo de software, desde la fase inicial hasta la fase final.
>
> Permite que los errores se detecten lo antes posible. Permite a los desarrolladores concentrarse en la calidad del software, en los plazos de implementación y en los costos asociados.

El propósito es definir las distintas fases intermedias que se requieren para llevar a cabo el desarrollo de una aplicación, garantizando que el software cumpla con los requisitos del diseño. Esto se origina en el hecho de que es muy costoso rectificar los errores que se detectan tarde dentro de la fase de implementación.

El **CICLO DE VIDA** básico de un software, consta de las siguientes fases:

1. DEFINICIÓN DE NECESIDADES: definir el resultado del proyecto y su papel en la estrategia global.
2. ANÁLISIS DE LOS REQUISITOS Y SU VIABILIDAD: recopilar, examinar y formular los requisitos del cliente y examinar cualquier restricción que se pueda aplicar.
3. DISEÑO GENERAL y DISEÑO DETALLADO: definir los requisitos generales de la arquitectura de la aplicación y precisar a nivel de detalle cada subconjunto de la aplicación.
4. CODIFICACION: utilizar un lenguaje de programación para crear las funciones definidas durante la etapa de diseño.
5. PRUEBAS (TESTING): asegurar la calidad del código a través de diversas capas de pruebas :
   - ***prueba unitariao de componentes*** es la prueba individual de cada programa o subconjunto dentro de la aplicación para garantizar que cumplen con las especificaciones.
   - ***prueba integral o de sistema*** para garantizar que los diferentes módulos se integren en la aplicación. Ésta debe estar cuidadosamente documentada.
   - ***prueba de regresión*** para comprobar que los cambios sobre un componente no introducen un comportamiento no deseado o errores adicionales en otros componentes no modificados. Las pruebas de regresión se deben llevar a cabo cada vez que se hace un cambio en el sistema, tanto para corregir un error como para realizar una mejora.
   - ***prueba de usuario o de aceptación*** para validar requerimientos del cliente
6. VALIDACION: poner a disposición del cliente el producto terminado, a través de los siguientes pasos:
   - ***entrega y conformidad***
   - ***documentación***
   - ***capacitación***
   - ***implementación o puesta en producción***
7. MANTENIMIENTO y EVOLUCION: efectuar procedimientos correctivos (mantenimiento correctivo) y las actualizaciones secundarias del software (mantenimiento continuo) que llevan a una evolución hacia la mejora contínua del sistema.

El orden y la presencia de cada uno de estos procedimientos en el ciclo de vida de una aplicación dependen del tipo de modelo de ciclo de vida acordado entre el cliente y el equipo de desarrolladores.

5. MODELOS DE CICLO DE VIDA

Para facilitar una metodología de trabajo común entre el cliente y la compañía de software, los modelos de ciclo de vida se han actualizado para reflejar las etapas de desarrollo involucradas y la documentación requerida, de manera que cada etapa se valide antes de continuar con la siguiente etapa.

Al final de cada etapa se arreglan las revisiones de manera que se avance sobre objetivos ajustados a las necesidades del cliente.

Las principales diferencias entre distintos modelos de ciclo de vida están divididas en tres grandes visiones:

- El alcance del ciclo de vida
- La cualidad y cantidad de las etapas
- La estructura y la sucesión de las etapas

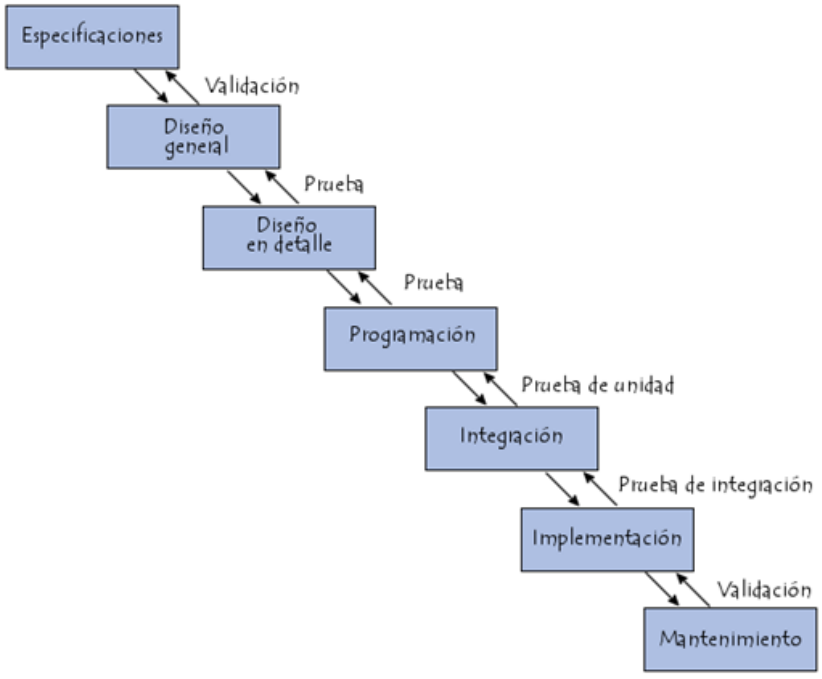
En los distintos modelos de ciclo de vida mencionaremos el **riesgo** que suponemos aceptar al elegirlo. Cuando hablamos de riesgo, nos referimos a la probabilidad que tendremos de volver a retomar una de las etapas anteriores, perdiendo tiempo, dinero y esfuerzo.

MODELOS DE CICLO DE VIDA
Lineal
En cascada
En "V"
Tipo Sashimi
En cascada con subproyectos
Iterativo
Por prototipos
Evolutivo
Incremental
En espiral
Orientado a objetos

A los fines de este curso , limitaremos el estudio a 5 modelos únicamente, que son los más frecuentemente utilizados en nuestros desarrollos. Sugerimos ahondar en el tema: *Capitulo 1: Ciclo de vida del software*: https://ingsw.pbworks.com/f/Ciclo+de+Vida+del+Software.pdf

MODELO EN CASCADA

El modelo de **CICLO DE VIDA EN CASCADA**, comenzó a diseñarse en 1966 y se terminó alrededor de 1970. Se define como una secuencia de fases en la que al final de cada una de ellas se reúne la documentación para garantizar que cumple las especificaciones y los requisitos antes de pasar a la fase siguiente:



Ventajas

El énfasis de la metodología en cascada se pone en la planificación de proyecto y, por tanto, antes de comenzar cualquier tipo de desarrollo es necesario que tanto la visión como el plan estén claros. Debido a que el método de cascada requiere una amplio esfuerzo de preparación previa, permite:

- Comenzar con el software con bastante rapidez.
- Estimar calendarios y presupuestos con mayor precisión.
- Lograr un nivel de satisfacción del cliente más elevado que otros enfoques, ya desde el principio.

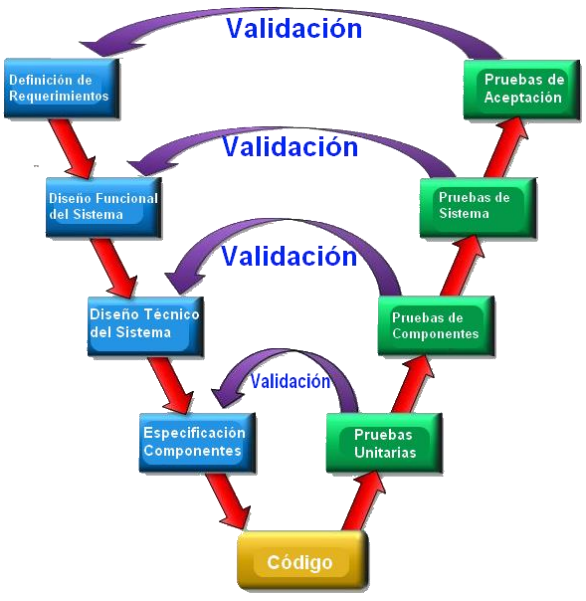
Desventajas

Este método es increíblemente rígido e inflexible, lo que plante inconvenientes como:

- Alterar el diseño del proyecto en cualquier etapa es muy complicado.
- Una vez que una fase se ha completado, es casi imposible de realizar cambios.
- Es absolutamente necesario reunir todos los requisitos iniciales.
- Resulta muy difícil responder a los problemas que puedan surgir, ya que tanto la retroalimentación, como las pruebas se retrasan hasta estadios muy tardíos del desarrollo de proyecto.
- Solucionar cualquier cuestión que se plantee requiere una cantidad sustancial de tiempo, esfuerzo y dinero.

MODELO EN "V"

El modelo de **CICLO DE VIDA EN "V"**, proviene del principio que establece que los procedimientos utilizados para probar si la aplicación cumple las especificaciones ya deben haberse creado en la fase de diseño.
Asocia un tipo de pruebas a cada producto de cada fase según su nivel de abstracción.



Por un lado sirve para indicar en qué fase de desarrollo se deben definir las pruebas correspondientes. Por otro sirve para saber a qué fase de desarrollo hay que volver si se encuentran fallos en las pruebas correspondientes.

Por lo tanto el modelo en V hace más explícita parte de las iteraciones y repeticiones de trabajo que están ocultas en el modelo en cascada. Mientras el foco del modelo en cascada se sitúa en los documentos y productos desarrollados, el modelo en V se centra en las actividades y la corrección.
Ventajas y desventajas del Modelo en "V"

Ventajas:
- La relación entre las etapas de desarrollo y los distintos tipos de pruebas facilitan la localización de fallos.
- Es un modelo sencillo y de fácil aprendizaje
- Hace explícito parte de la iteración y trabajo que hay que revisar
- Especifica bien los roles de los distintos tipos de pruebas a realizar
- Involucra al usuario en las pruebas

Desventajas:
- Es difícil que el cliente exponga explícitamente todos los requisitos
- El cliente debe tener paciencia pues obtendrá el producto al final del ciclo de vida
- Las pruebas pueden ser caras y, a veces, no lo suficientemente efectivas
- El producto final obtenido puede que no refleje todos los requisitos del usuario

MODELO ITERATIVO

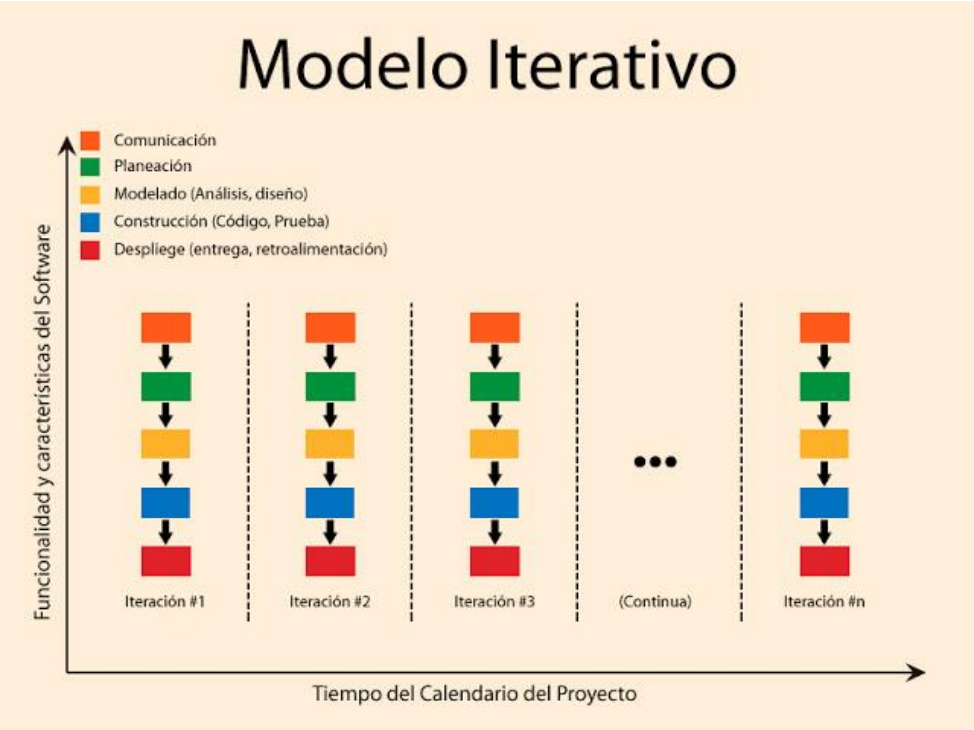
Es un modelo derivado del ciclo de vida en cascada. Este modelo busca reducir el riesgo que surge entre las necesidades del usuario y el producto final por malos entendidos durante la etapa de recogida de requisitos.

Consiste en la iteración de varios ciclos de vida en cascada. Al final de cada iteración se le entrega al cliente una versión mejorada o con mayores funcionalidades del producto. El cliente es quien después de cada iteración evalúa el producto y lo corrige o propone mejoras. Estas iteraciones se repetirán hasta obtener un producto que satisfaga las necesidades del cliente.:

Este modelo se suele utilizar en proyectos en los que los requisitos no están claros por parte del usuario, por lo que se hace necesaria la creación de distintos prototipos para presentarlos y conseguir la conformidad del cliente.

Ventajas
- no hace falta que los requisitos estén totalmente definidos al inicio del desarrollo, sino que se pueden ir refinando en cada una de las iteraciones.
- realizar el desarrollo en pequeños ciclos, lo que permite gestionar mejor los riesgos, y las entregas.

Desventajas
- Al no ser necesario tener los requisitos definidos desde el principio, pueden surgir problemas relacionados con la arquitectura.



MODELO POR PROTOTIPOS

Un ***prototipo*** es un modelo que podemos utilizar para generar y diseñar una actividad que nos permita crear un diseño rápido en la construcción de un software.

Cuando hablamos del modelo como tal, éste debe ser construido en poco tiempo, usando los programas adecuados y no se deben utilizar muchos recursos. El diseño rápido se centra en una representación de aquellos aspectos del software que serán visibles para el cliente o el usuario final. Este diseño conduce a la construcción de un prototipo, el cual es evaluado por el cliente para una retroalimentación; gracias a ésta se refinan los requisitos del software que se desarrollará. La interacción ocurre cuando el prototipo se ajusta para satisfacer las necesidades del cliente. Esto permite que al mismo tiempo el desarrollador entienda mejor lo que se debe hacer y el cliente vea resultados a corto plazo.

A pesar de que tal vez surjan problemas, la construcción de prototipos puede ser un paradigma efectivo para la ingeniería del software. La clave es definir las reglas del juego desde el principio; es decir, el cliente y el desarrollador se deben poner de acuerdo en:

- Que el prototipo se construya y sirva como un **mecanismo para la definición de requisitos**.
- Que el prototipo se **descarte**, al menos en parte.
- Que después se desarrolle el software real con un enfoque hacia la **calidad**.

Ventajas:

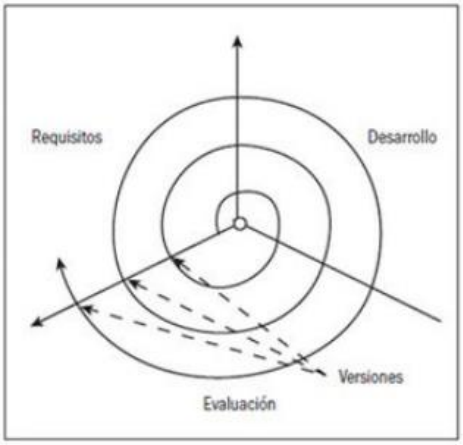
- Este modelo es útil cuando el cliente conoce los objetivos generales para el software, pero no identifica los requisitos detallados de entrada, procesamiento o salida.
- También ofrece un mejor enfoque cuando el responsable del desarrollo del software está inseguro de la eficacia de un algoritmo, de la adaptabilidad de un sistema operativo o de la forma que debería tomar la interacción humano-máquina.

Desventajas:

- Requiere participación activa del usuario, al menos, para evaluar el prototipo. Y mucho más involucramiento si queremos que participe en su creación.
- Una desventaja importante a tener en cuenta es la falta de experiencia que tienen muchos Analistas Funcionales en programación y en actividades de diseño de interfaces de usuario.

MODELO EVOLUTIVO

En este modelo los requerimientos del usuario pueden cambiar en cualquier momento.
Cuanto mayor es un proyecto, menor es la probabilidad de éxito (informe CHAOS). Obtener todos los requisitos al comienzo es prácticamente imposible, las necesidades de clientes y usuarios evolucionan durante el desarrollo.
Se focaliza básicamente en tres ciclos: **requisitos-desarrollo-evaluación**
El resultado de la evaluación permite evolucionar hacia la siguiente versión. El proceso se repite indefinidamente.



Ventajas:

*Este modelo puede ser cambiado en cualquier momento.
*Es muy útil cuando desconocemos la mayoría de las solicitudes iniciales o cuando los requerimientos no están completos.

Desventajas:

*Modelo evolutivo asume que las necesidades no son completamente conocidos al inicio del proyecto.
*El desarrollo de software en forma evolutiva requiere un especial cuidado en la manipulación de documentos, programas, etc. desarrollados para distintas versiones del software.