Comparativas de los modelos de ciclo de vida

Ana Alberta Canepa Sáenz
Christian Erika García González*

Resumen
Los modelos de ciclos de vida son instrucciones indispensables para la elaboración de un proyecto de calidad, económico y en el menor tiempo posible.

Introducción
Los modelos de ciclos de vida son un conjunto de instrucciones que se deben de seguir para la elaboración de un proyecto, aun cuando el cliente no tiene un conocimiento exacto de lo que desea. En esta parte el usuario especifica sus necesidades; el ingeniero en software, conforme le van dando los requerimiento, aporta las opciones que considere que irá mejor con su proyecto. Los ciclos de vida tienen ventajas y desventajas, no obstante, el propósito de todos estos es garantizar que el proyecto sea de calidad y que cumpla con las expectativas del cliente. Cada ciclo nos explica en qué sectores son ideales para ser utilizados, y el elegir entre uno de los modelos de ciclo de vida del software depende de las necesidades del usuario, como el uso, la economía y la complejidad.

Ciclo de vida del software
El término ciclo de vida del software describe el desarrollo de software, desde la fase inicial hasta la fase final. [5]

Un modelo de ciclo de vida de software es una vista de las actividades que ocurren durante el desarrollo de software. Intenta determinar el orden de las etapas involucradas y los criterios de transición asociadas entre estas etapas.

Un modelo de ciclo de vida del software:

- Describe las fases principales de desarrollo de software.
- Define las fases primarias esperadas de ser ejecutadas durante esas fases.
- Ayuda a administrar el progreso del desarrollo, y
- Provee un espacio de trabajo para la definición de un detallado proceso de desarrollo de software.

Así, los modelos por una parte suministran una guía para los ingenieros de software con el fin de ordenar las diversas actividades técnicas en el proyecto; por otra parte, suministran un marco para la administración del desarrollo y el mantenimiento, en el sentido en que permiten estimar recursos, definir puntos de control intermedios, monitorear el avance, etcétera. [6]

Modelo lineal
Es el más sencillo de todos los modelos. Consiste en descomponer la actividad global del proyecto en etapas separadas, que son realizadas de manera lineal, es decir, cada etapa se realiza una sola vez, a continuación de la etapa anterior y antes de la etapa siguiente. Con un ciclo de vida lineal es muy fácil dividir las tareas, y prever los tiempos (sumando linealmente los de cada etapa). Las actividades de cada una de las etapas mencionadas deben ser independientes entre sí, es decir, que es condición primordial que no haya retroalimentación entre ellas, aunque sí pueden admitirse ciertos supuestos de realimentación correctiva. Desde el punto de vista de la gestión, requiere también que se conozca desde el primer momento, con excesiva rigidez, lo que va a ocurrir en cada una de las distintas etapas antes de comenzarla. Esto último minimiza también las posibilidades de errores durante la codificación y reduce al mínimo la necesidad de requerir información del cliente o del usuario.

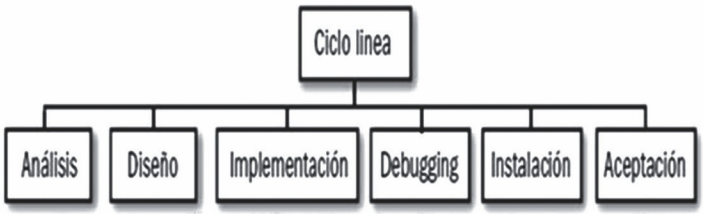

Figura 1. La sencilles del ciclo de vida lineal es la razón por la cual es el más elegido en el desarrollo de proyectos pequeños.

*Ana Alberta Canepa Sáenz, profesora de tiempo completo en la Dependencia Ciencias de la Información de la Universidad Autónoma del Carmen.
Christian Erika García González, estudiante de Licenciatura en Informática, de la dependencia Ciencia de la Información de la Universidad Autónoma del Carmen.

Se destaca como ventaja la sencillez de su gestión y administración tanto económica como temporal, ya que se acomoda perfectamente a proyectos internos de una empresa para programas muy pequeños de ABM (sistemas que realizan Altas, Bajas y Modificaciones sobre un conjunto de datos). Tiene como desventaja que no es apto para Desarrollos que superen mínimamente requerimientos de retroalimentación entre etapas, es decir, es muy costoso retomar una etapa anterior al detectar alguna falla.

Es válido tomar este ciclo de vida cuando algún sector pequeño de una empresa necesita llevar un registro de datos acumulativos, sin necesidad de realizar procesos sobre ellos más que una consulta simple. Es decir, una aplicación que se dedique exclusivamente a almacenar datos, sea una base de datos o un archivo plano. Debido a que la realización de las etapas es muy simple y el código muy sencillo. [7]

Ciclo de vida cascada pura

En un modelo en cascada, un proyecto progresa a través de una secuencia ordenada de pasos partiendo del concepto inicial del software hasta la prueba del sistema. El proyecto realiza una revisión al final de cada etapa para determinar si está preparado para pasar a la siguiente etapa. Cuando la revisión determina que el proyecto no está listo para pasar a la siguiente etapa, permanece en la etapa actual hasta que esté preparado.

El modelo en cascada pura se utiliza correctamente para ciclos de productos en los que se tiene una definición estable del producto, y también cuando se está trabajando con metodologías técnicas conocidas. En estos casos, el modelo en cascada ayuda a localizar errores en las primeras etapas del proyecto a un bajo coste. [2]

Es un modelo rígido, poco flexible, y con muchas restricciones. Aunque fue uno de los primeros, y sirvió de base para el resto de los modelos de ciclo de vida.

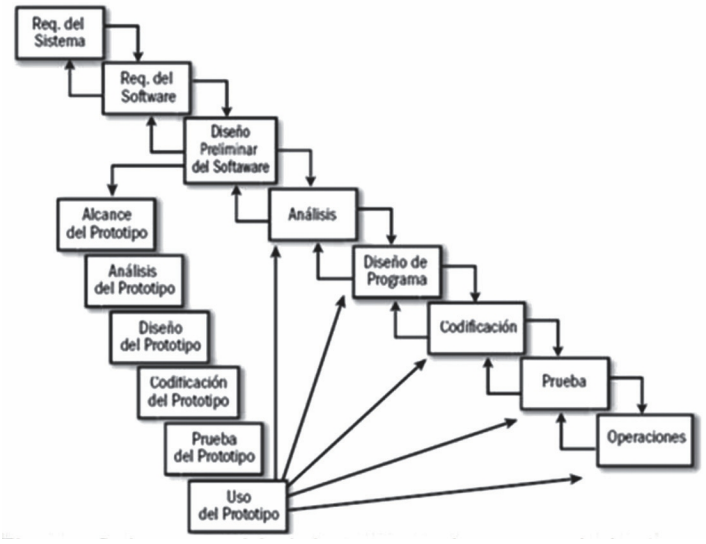

Figura 2. La nececidad de conocer los requerimientos al principio del proyecto es primordial al elegir este modelo de ciclo de vida a pesar de permitir las iteraciones.

Una de sus ventajas, además de su planificación sencilla, es la de proveer un producto con un elevado grado de calidad sin necesidad de un personal altamente calificado. Se pueden considerar como inconvenientes: la necesidad de contar con todos los requerimientos (o la mayoría) al comienzo del proyecto, y, si se han cometido errores y no se detectan en la etapa inmediata siguiente, es costoso y difícil volver atrás para realizar la corrección posterior.

Se evidencia que es un modelo puramente teórico, ya que el usuario rara vez mantiene los requerimientos iniciales y existen muchas posibilidades de que debamos retomar alguna etapa anterior. [7]

Las desventajas del modelo en cascada pura se centra en la dificultad para especificar claramente los requerimientos al comienzo del proyecto, antes de que se realice ningún trabajo de diseño y antes de escribir ningún código. [2]

Ciclo de vida en V

Este ciclo fue diseñado por Alan Davis, y contiene las mismas etapas que el ciclo de vida en cascada puro. A diferencia de aquél, a éste se le agregaron dos subetapas de retroalimentación entre las etapas de análisis y mantenimiento, y entre las de diseño y debugging[1].

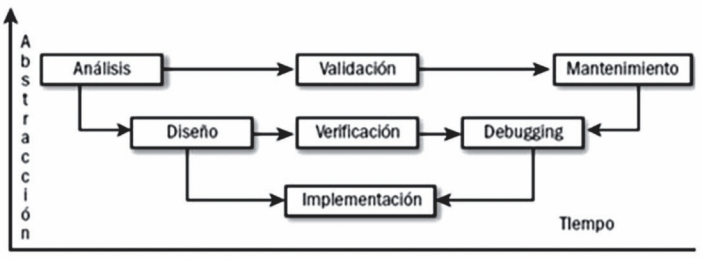

Figura 3. Este modelo nos ofrece mayor garantía de corrección al terminar el proyecto.

Las ventajas y desventajas de este modelo son las mismas del ciclo anterior, con el agregado de los controles cruzados entre etapas para lograr una mayor corrección. [7]

Ciclo de vida tipo Sashimi

Este ciclo de vida es parecido al ciclo de vida en cascada puro, con la diferencia de que en el ciclo de vida en cascada no se pueden solapar las etapas, y en éste sí. Esto suele, en muchos casos, aumentar su eficiencia ya que la retroalimentación entre etapas se encuentra implícitamente en el modelo

[1] Aplicación o herramienta que permite la ejecución controlada de un programa o un código, para seguir cada instrucción ejecutada y localizar así *bugs* o errores (proceso de depuración), códigos de protección, etcétera. [8]

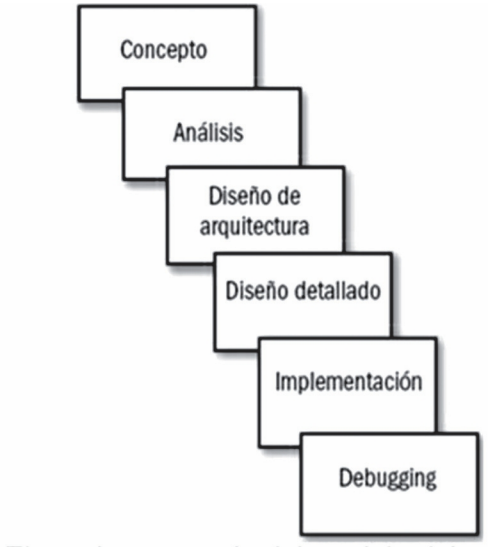

Figura 4. El nombre procede del modelo del estilo japonés de presentar el pescado crudo cortado, en que los cortes se solapan entre si.

Se hace notar como ventajas la ganancia de calidad en lo que respecta al producto final, la falta de necesidad de una documentación detallada (el ahorro proviene por el solapado de las etapas). Sus desventajas también se refieren al solapamiento de las etapas: es muy difícil gestionar el comienzo y fin de cada etapa y los problemas de comunicación, si aparecen, generan inconsistencias en el proyecto. [7]

Ciclo de vida en cascada con sub-proyectos

Sigue el modelo de ciclo de vida en cascada con subproyectos. Cada una de las cascadas se divide en subetapas independientes que se pueden desarrollar en paralelo.

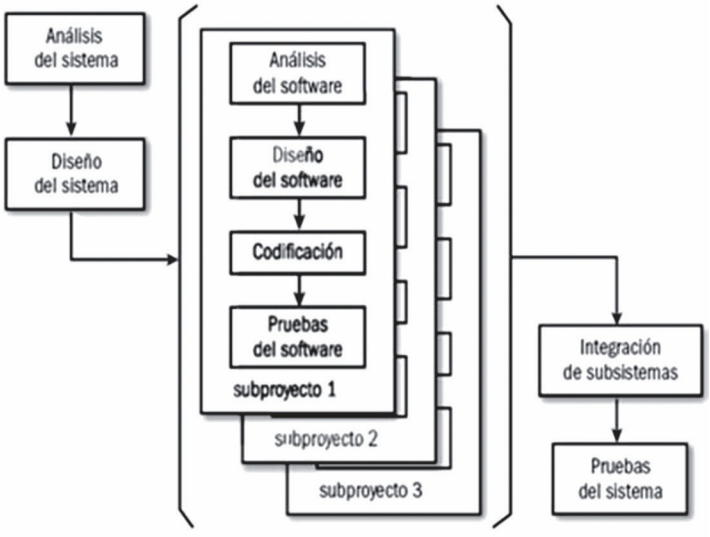

Figura 5. El modelo de ciclo de vida ideal cuando se cuenta con un panel de programadores numerosos.

La ventaja es que se puede tener más gente trabajando al mismo tiempo, pero la desventaja es que pueden surgir dependencias entre las distintas subetapas que detengan el proyecto temporalmente si no es gestionado de manera correcta. [7]

Ciclo de vida iterativo

También derivado del ciclo de vida en cascada puro, este modelo busca reducir el riesgo que surge entre las necesidades del usuario y el producto final por malos entendidos durante la etapa de solicitud de requerimientos.

Es 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 luego de cada iteración, evalúa el producto y lo corrige o propone mejoras. Estas iteraciones se repetirán hasta obtener un producto que satisfaga al cliente.

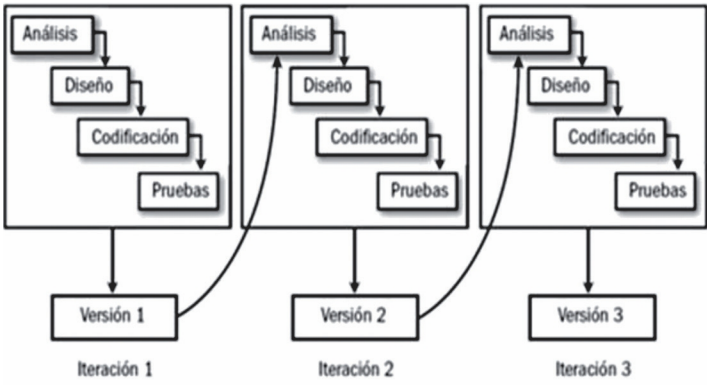

Figura 6. Es un modelo ideal para seguir cuando el usuario necesita entregas rápidas aunque el proyecto no esté terminado.

Se suele utilizar en proyectos en los que los requerimientos no están claros de parte del usuario, por lo que se hace necesaria la creación de distintos prototipos para presentarlos y conseguir la conformidad del cliente. [7]

Ciclo de vida por prototipos

La construcción de prototipos es un proceso que facilita al ingeniero de software el desarrollo de la aplicación. El prototipo suele tomar una de las tres formas siguientes:

- Un modelo en papel o en computadora que describe la interacción hombre-máquina, de forma que facilite al usuario la comprensión de su funcionamiento. Por ejemplo, si el sistema a construir es un cajero automático, se puede hacer un programa que simule la interacción del usuario con el cajero sin que el programa esté conectado a ninguna base de datos real ni se despache dinero. De esta manera el cliente puede hacerse a la idea de cómo va a funcionar el sistema al final sin tener que construirlo, y así discutirlo con el ingeniero de software.
- Un modelo que implementa una función requerida importante. El mismo caso que anteriormente pero sin centrarse en la interacción hombre-máquina. Por ejemplo, el modelo podría simular todos los pasos a seguir internamente en el sistema en el acceso a la base de datos de

clientes cuando se quiere obtener dinero del cajero, pero sin que realmente se trate de una base de datos real ni de un cliente del banco.

• Un programa real que se adecue en parte al software que se desea desarrollar. Por ejemplo, se puede disponer de una aplicación relacionada con un "cajero automático", que al presentarla al cliente, permita al analista identificar las necesidades del cliente y por lo tanto los requisitos del software a construir.

El prototipo sirve como mecanismo para identificar los requisitos del software, y su construcción suele llevar las siguientes etapas:

- Recolección de requisitos.
- Diseño rápido.
- Construcción de prototipo.
- Evaluación de prototipo.
- Refinamiento del prototipo.
- Producto. [3]

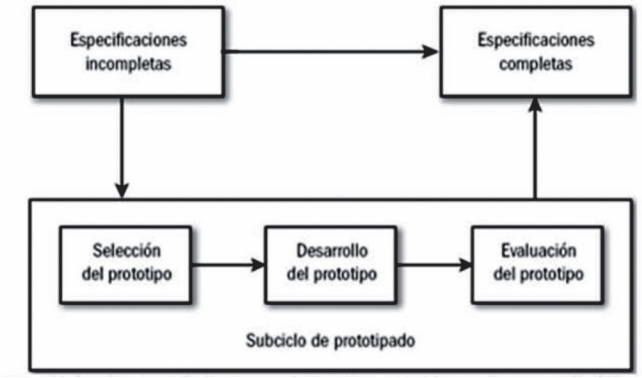

Figura 7. Este modelo nos permite suavisar la trancisión entre los requerimientos iniciales y finales que surgen en la creación de un proyecto con grandes innovaciones.

Se utiliza mayoritariamente en desarrollos de productos con innovaciones importantes, o en el uso de tecnologías nuevas o poco probadas, en las que la incertidumbre sobre los resultados a obtener, o la ignorancia sobre el comportamiento, impiden iniciar un proyecto secuencial.

La ventaja de este ciclo se basa en que es el único apto para desarrollos en los que no se conoce a priori sus especificaciones o la tecnología a utilizar. Como contrapartida, por este desconocimiento, tiene la desventaja de ser altamente costoso y difícil para la administración temporal. [7]

Ciclo de vida evolutivo

Este modelo acepta que los requerimientos del usuario pueden cambiar en cualquier momento. La práctica nos demuestra que obtener todos los requerimientos al comienzo del proyecto es extremadamente difícil, no sólo por la dificultad del usuario de transmitir su idea, sino porque estos requerimientos evolucionan durante el desarrollo y de esta manera, surgen nuevos requerimientos a cumplir. El modelo de ciclo de vida evolutivo afronta este problema mediante una iteración de ciclos requerimientos–desarrollo–evaluación.

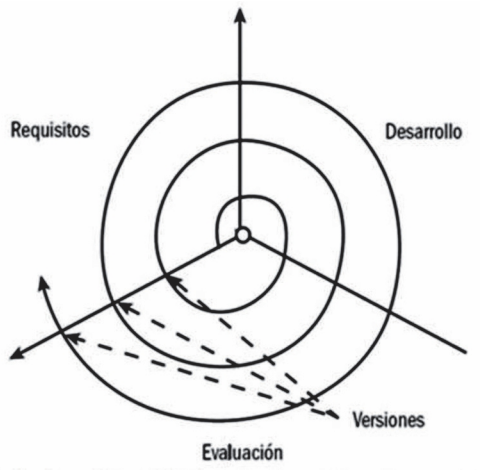

Figura 8. Luego de cada desarrollo obtenemos una nueva versión del producto.

Resulta ser un modelo muy útil cuando desconocemos la mayoría de los requerimientos iniciales, o estos requerimientos no están completos. [7]

Ciclo de vida incremental

El modelo incremental combina elementos del modelo lineal secuencial (aplicados repetidamente) con la filosofía interactiva de construcción de prototipos. El modelo incremental aplica secuencias lineales de forma escalonada mientras progresa el tiempo en el calendario. Cada secuencia lineal produce un incremento de software. Por ejemplo, el software de tratamiento del texto desarrollado con el paradigma incremental podría extraer funciones de gestión de archivos básicos y de producción de documentos en el primer incremento; funciones de edición más sofisticadas y de producción de documentos en el segundo incremento; corrección ortográfica y gramatical en el tercero; y una función avanzada de esquema de página en el cuarto. Se debería tener en cuenta que el flujo del proceso de cualquier incremento puede incorporar el paradigma de construcción de prototipos. "el modelo incremental entrega software en partes pequeñas, pero utilizables, llamadas *incrementos*. En general cada incremento se constituye sobre aquél que ya ha sido entregado. [1]

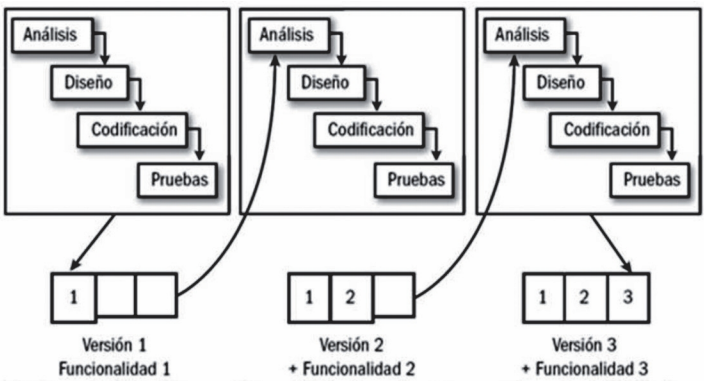

Figura 9. Una forma de reducir los riesgos es ir construyendo partes del sistema adoptando este modelo.

El modelo de ciclo de vida incremental nos genera algunos beneficios tales como los que se describen a continuación:

- Construir un sistema pequeño siempre es menos riesgoso que construir un sistema grande.
- Como desarrollamos independientemente las funcionalidades, es más fácil relevar los requerimientos del usuario.
- Si se detecta un error grave, sólo desechamos la última iteración.
- No es necesario disponer de los requerimientos de todas las funcionalidades en el comienzo del proyecto y además facilita la labor del desarrollo con la conocida filosofía de divide & conqueror. [7]

Ciclo de vida en espiral

El modelo en espiral, propuesto originalmente por Boehm [BOE88], es un modelo de proceso de software evolutivo que conjuga la naturaleza interactiva de construcción de los prototipos con los aspectos controlados y sistemáticos del modelo lineal secuencial. Proporciona el potencial para el desarrollo rápido de versiones incrementales del software. En el modelo espiral, el software se desarrolla en una serie de versiones incrementales. Durante las primeras interacciones, la versión incremental podría ser un modelo en papel o un prototipo. Durante las últimas iteraciones, se producen versiones cada vez más completas del sistema diseñado.

El modelo en espiral se divide en un número de actividades de marco de trabajo, también llamadas región de tareas. Generalmente, existen entre tres y seis regiones de tareas. La figura 10 representa un modelo en espiral que contiene seis regiones de tareas:

- Comunicación con el cliente: las tareas requeridas para establecer comunicación entre el desarrollador y el cliente.
- Planificación: Las tareas requeridas para definir recursos, el tiempo y otra información relacionadas con el proyecto.
- Ingeniería: Las tareas requeridas para construir una o más representaciones de la aplicación.
- Construcción y acción: Las tareas requeridas para construir, probar, instalar y proporcionar soporte al usuario (ejemplo: documentación y práctica).
- Evaluación de cliente: las tareas requeridas para obtener la relación del cliente según la evaluación de las representaciones del software creadas durante la etapa de ingeniería e implementada durante la etapa de instalación. [1]

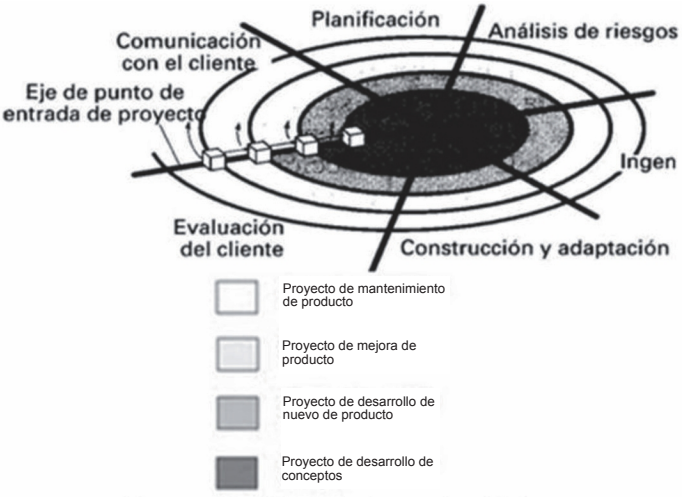

Figura 10. Un modelo espiral típico.

La ventaja más notoria de este modelo de desarrollo de software es que puede comenzarse el proyecto con un alto grado de incertidumbre, se entiende también como ventaja el bajo riesgo de retraso en caso de detección de errores, ya que se puede solucionar en la próxima rama del espiral.

Algunas de las desventajas son: el costo temporal que suma cada vuelta del espiral, la dificultad para evaluar los riesgos y la necesidad de la presencia o la comunicación continúa con el cliente o usuario.

Se observa que es un modelo adecuado para grandes proyectos internos de una empresa, en donde no es posible contar con todos los requerimientos desde el comienzo y el usuario está en nuestro mismo ambiente laboral. [7]

Ciclo de vida orientado a objetos

El ciclo de vida del software orientado a objetos, aunque mantiene casi las mismas fases generales que el ciclo de vida clásico, distribuye las fases de forma distinta, según el modelo iterativo e incremental, y teniendo en cuenta otros aspectos nuevos, como el análisis y la gestión de riesgos, el control de calidad y la obtención de modelos consistentes. [4]

Esta técnica fue presentada en la década del 90, tal vez como una de las mejores metodologías a seguir para la creación de productos software. Puede considerarse como un modelo pleno a seguir, como así también una alternativa dentro de los modelos anteriores.

Al igual que la filosofía del paradigma de la programación orientada a objetos, en esta metodología cada funcionalidad, o requerimiento solicitado por el usuario, es considerado un objeto. Los objetos están representados por un conjunto de propiedades, a los cuales denominamos atributos, por otra parte, al comportamiento que tendrán estos objetos los denominamos métodos.

Vemos que tanto la filosofía de esta metodología, los términos utilizados en ella y sus fines, coinciden con la idea de obtener un concepto de objeto sobre casos de la vida real.

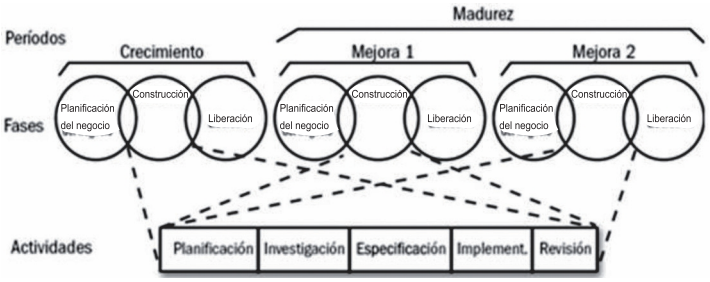

Figura 11. Un modelo muy versátil, tanto para pequeños como para grandes proyectos.

La característica principal de este modelo es la abstracción de los requerimientos de usuario, por lo que este modelo es mucho más flexible que los restantes, que son rígidos en requerimientos y definición, soportando mejor la incertidumbre que los anteriores, aunque sin garantizar la ausencia de riesgos. La abstracción es lo que nos permite analizar y desarrollar las características esenciales de un objeto (requerimiento), despreocupándonos de las menos relevantes. Favorece la reducción de la complejidad del problema que deseamos abordar y permite el perfeccionamiento del producto. [7]

Cuadro comparativo de los diferentes ciclos de vida

Tabla1. Ventajas y desventajas de los ciclos de vida.

Ciclo de vida	Ventajas	Desventajas
Lineal	Administración, económica	Costoso si se detecta una falla
Cascada puro	Planificación sencilla, producto de calidad, sin necesitar personal calificado.	Necesita todos los requerimientos, costoso y difícil volver atrás para realizar la corrección.
En V	Planificación sencilla, producto de calidad, sin necesitar personal calificado.	Necesita todos los requerimientos, costoso y difícil volver atrás para realizar la corrección.
Sashimi	Producto de calidad sin documentacion detallada.	Difícil de gestionar, al finalizar las etapas, problema de comunicación, inconsistencia en el proyecto.
Cascada con subproyecto	Puede ver a varias personas trabajando al mismo tiempo.	Dependencia entre las subetapas, detener el proyecto si no se administra bien.
Iterativo	Se va creando al gusto del cliente	Tardado
Prototipo	Desarrollos en los que no se conoce si prioridad, sus especificaciones o la tecnología a utilizar.	Altamente costoso y difícil para la administración temporal.
Evolutivo	Desconocemos la mayoría de los requerimientos iniciales o no están completos los requerimientos.	El usuario no conoce de informática, nos pueden pedir modificaciones o hacer nuevas solicitudes.
Incremental	Construir un sistema pequeño, dearrolla independientemente las funcionalidades, no es necesario disponer de los requerimientos	No es para todo tipo de aplicaciones
Espiral	Comenzar el proyecto, aunque no se conoscan las especificaciones, bajo riesgo de retraso	Costoso por cada vuelta, difícil evaluar los riesgos, necesita la presencia continua del cliente.
Orientado a objeto	Flexible	Conocimiento en lenguajes de programación.

Tabla 2. Comparación de los ciclos de vida en la elavoración de proyectos.

Ciclo de vida	Fácil	Complicado
Lineal	✓	
Cascada puro		✓
En V		✓
Sashimi		✓
Cascada con subproyecto		✓
Iterativo	✓	✓
Prototipo	✓	✓
Evolutivo	✓	
Incremental	✓	
Espiral	✓	
Orientado a objeto	✓	

Tabla 3. Comparación de las ventajas y desventajas los ciclos de vida.

Ciclo de vida	Admón.		Económico		Requerimientos		Flexible		Req. personal calificado		Producto de calidad	
	SI	NO	SI	NO	SI	NO	SI	NO	SI	NO	SI	NO
Lineal	✓		✓			✓				✓	✓	
Cascada puro	✓		✓		✓			✓		✓	✓	
En V	✓		✓		✓			✓		✓	✓	
Sashimi						✓			✓		✓	
Cascada con subproyecto	✓							✓	✓		✓	
Iterativo								✓	✓		✓	
Prototipo		✓		✓		✓		✓	✓		✓	
Evolutivo						✓						
Incremental						✓		✓	✓		✓	
Espiral				✓		✓		✓	✓		✓	
Orientado a objeto							✓		✓		✓	

Tabla 4. Tipos de proyectos donde se pueden utilizar los ciclos de vida.

Ciclo de vida	Tipo de Proyectos		
	Pequeños	Medianos	Grandes
Lineal	✓		
Cascada puro			
En V			
Sashimi			
Cascada con subproyecto	✓	✓	✓
Iterativo	✓	✓	✓
Prototipo			
Evolutivo	✓	✓	✓
Incremental			
Espiral			✓
Orientado a objeto	✓	✓	✓

Tabla 5. Donde se utilizan los ciclos de vida.

Ciclo de vida	Usos	
	Compartir recursos	Almacenar
Lineal		✓
Cascada puro		
En V		
Sashimi	✓	
Cascada con subproyecto	✓	✓
Iterativo		
Prototipo		
Evolutivo	✓	✓
Incremental		
Espiral		
Orientado a objeto	✓	✓

Tabla 6. Tipos de migraciones que soportan los ciclos de vida.

Ciclo de vida	Migrar	
	Aplicaciones	Base de datos
Lineal		
Cascada puro		
En V		✓
Sashimi		
Cascada con subproyecto	✓	✓
Iterativo	✓	✓
Prototipo		
Evolutivo	✓	✓
Incremental		
Espiral		
Orientado a objeto	✓	✓

En las tablas comparativas anteriores, se muestra de forma más precisas, los usos, ventajas, desventajas y la facilidad de elaboración de los diversos ciclos de vidas, uno son mejores que otro, pero no se puede escoger cual sería el mejor, ya que, cada uno se adapta a las necesidades de cliente, usuario o proyecto.

Conclusión

Por lo tanto, como hemos leído en lo anterior, los modelos ciclos de vida son una parte muy importante de la ingeniería de software, ya que es donde se concentra toda la información importante de un proyecto, y que todas las etapas van tomada de la mano ya que si una falla todo lo demás estaría mal y sería una pérdida enorme de tiempo y sobretodo de dinero, es por ello que en cada etapa van surgiendo prototipos para asegurarse de que haga lo que el cliente o usuaria quiere o necesite, cada modelo de ciclo de vida se adapta según las necesidades de cliente.

Bibliografía

1. S. Pressman, Roger, *Ingeniería de Software Un Enfoque Práctico*, Editorial McGraw Hill, 5ta Edición, (año).

2. McConnell, Steve, *Desarrollo y Gestión de Proyectos Informáticos*, Editorial McGraw Hill, 1a Edición.

3. Alonso Amo, Fernando, ínez Normand, Loïc; Pérez, Francisco Javier, *Introducción a la Ingeniería del Software*, Editorial Delta, 1ª edición, Año 2005.

4. Salvador Sánchez, José, *Ingeniería de Proyectos Informáticos: Actividades y Procedimientos*, Editorial Universitat Jaume I, 1ª edición, Año 2003.

5. *Jeff, 16 de octubre de 2008*. http://es.kioskea.net/contents/genie-logiciel/cycle-de-vie.php3

6. Definicion de un modelo de ciclo de vida, http://rguerrero334.blogspot.es/img/Def.Modelo_de_Ciclo_de_Vida.pdf

7. *lectores@mpediciones.com, Implementación y Debugging, Ciclo de Vida del Software* http://www.cepeu.edu.py/LIBROS_ELECTRONICOS_3/lpcu097%20-%2001.pdf

8. Definición de Debugger, ALEGSA Santa Fe, Argentina, 1998 – 2011. http://www.alegsa.com.ar/Dic/debugger.php