# Modelos de procesos de software

Universidad Nacional de Trujillo

Informática

Trujillo – Perú

Mg. Salazar Campos, Juan Orlando

1er Autor

Sánchez Gómez, Willians Eduardo  
927700436

wsanchezg@unitru.edu.pe

2do Autor

Zerpa Castro Patrick  
956771973

pzerpa@unitru.edu.pe

**Palabras clave**. Ciclo de vida del software.

1. **Modelos de procesos de software.**

También conocido como “Ciclo de vida del software”, es una representación simplificada de un proceso. Las principales diferencias entre los distintos modelos de ciclo de vida son:

* El alcance del ciclo de vida.
* La cualidad y la cantidad de etapas en que dividiremos el ciclo de vida.
* La estructura y la sucesión de las etapas.
  1. **Modelo en Cascada.**

Es el primer modelo publicado sobre el proceso de desarrollo de software, su nombre se debe al paso de una fase a otra simulando una cascada. “Éste modelo toma las actividades fundamentales del proceso de especificación, desarrollo, validación y evolución y, luego, los representa como fases separadas del proceso, tal como especificación de requerimientos, diseño de software, implementación, pruebas, etc.” (Sommerville, 2011, p. 29).

Las principales etapas del modelo en cascada reflejan directamente las actividades fundamentales del desarrollo:

*Análisis y definición de requerimientos:* Los servicios, las restricciones y las metas del sistema se establecen mediante consulta a los usuarios del sistema. Luego, se definen con detalle y sirven como una especificación del sistema.

*Diseño del sistema y del software:* El proceso de diseño de sistemas asigna los requerimientos, para sistemas de hardware o de software, al establecer una arquitectura de sistema global. El diseño del software implica identificar y describir las abstracciones fundamentales del sistema de software y sus relaciones.

*Implementación y prueba de unidad:* Durante esta etapa, el diseño de software se realiza como un conjunto de programas o unidades del programa. La prueba de unidad consiste en verificar que cada unidad cumpla con su especificación.

*Integración y prueba de sistema:* Las unidades del programa o los programas individuales se integran y prueban como un sistema completo para asegurarse de que se cumplan los requerimientos de software. Después de probarlo, se libera el sistema de software al cliente.

*Operación y mantenimiento:* Por lo general (aunque no necesariamente), ésta es la fase más larga del ciclo de vida, donde el sistema se instala y se pone en práctica. El mantenimiento incluye corregir los errores que no se detectaron en etapas anteriores del ciclo de vida, mejorar la implementación de las unidades del sistema e incrementar los servicios del sistema conforme se descubren nuevos requerimientos.

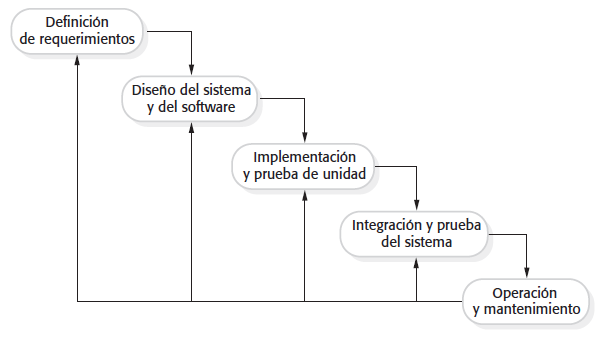


Figura 2. Modelo en cascada

* 1. **Modelo en V.**

Este modelo fue diseñado por Alan Davis, es una variante de la representación del modelo en cascada. A medida que el equipo de software avanza hacia abajo desde el lado izquierdo de la V, los requerimientos básicos del problema mejoran hacia representaciones técnicas cada vez más detalladas del problema y de su solución. Una vez que se ha generado el código, el equipo sube por el lado derecho de la V, y en esencia ejecuta una serie de pruebas (acciones para asegurar la calidad) que validan cada uno de los modelos creados cuando el equipo fue hacia abajo por el lado izquierdo. En realidad, no hay diferencias fundamentales entre el modelo en cascada y el modelo en V. Este último proporciona una forma de visualizar el modo de aplicación de las acciones de verificación y validación al trabajo de ingeniería inicial.

Pressman (2010) afirma “El modelo en V ilustra la forma en la que se asocian las acciones de verificación y validación con las primeras acciones de ingeniería” (p. 34).

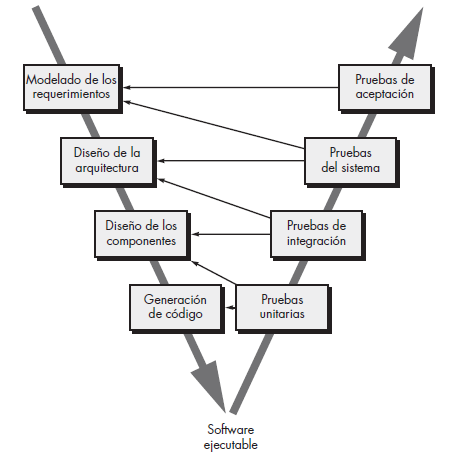


Figura 3. Modelo en V

* 1. **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.

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

Podemos adoptar el modelo mencionado en aplicaciones medianas a grandes, en las que el usuario o cliente final no necesita todas las funcionalidades desde el principio del proyecto. (Implementación y debugging,2006, p.27)

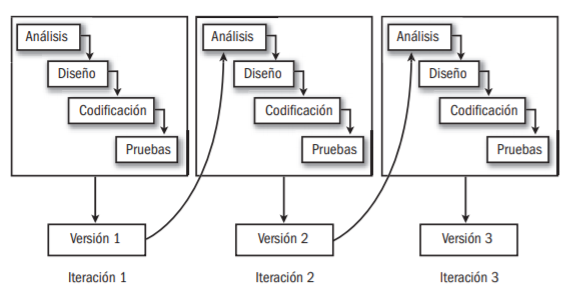


Figura 4. Ciclo de vida iterativo

* 1. **Desarrollo incremental**

“El desarrollo incremental se basa en la idea de diseñar una implementación inicial, exponer ésta al comentario del usuario, y luego desarrollarla en sus diversas versiones hasta producir un sistema adecuado” (Sommerville, 2011, p. 32). Las actividades de especificación, desarrollo y validación están entrelazadas en vez de separadas, con rápida retroalimentación a través de las actividades.

Cada incremento o versión del sistema incorpora algunas de las funciones que necesita el cliente. Por lo general, los primeros incrementos del sistema incluyen la función más importante o la más urgente. Esto significa que el cliente puede evaluar el desarrollo del sistema en una etapa relativamente temprana, para constatar si se entrega lo que se requiere. En caso contrario, sólo el incremento actual debe cambiarse y, posiblemente, definir una nueva función para incrementos posteriores.

Beneficios del desarrollo incremental:

* Se reduce el costo de adaptar los requerimientos cambiantes del cliente.
* Es más sencillo obtener retroalimentación del cliente sobre el trabajo de desarrollo que se realizó.
* Es posible que sea más rápida la entrega e implementación de software útil al cliente, aun si no se ha incluido toda la funcionalidad.

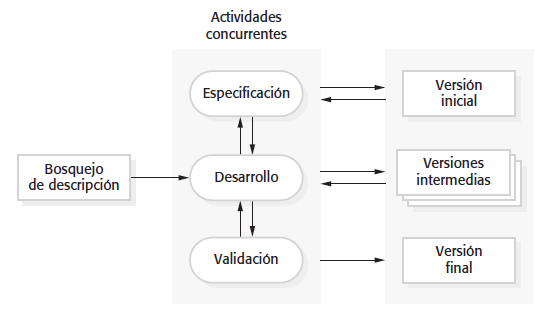


Figura 5. Desarrollo incremental

* 1. **Ingeniería de software orientada a la reutilización:**

En la mayoría de los proyectos de software hay cierta reutilización de software. Sucede con frecuencia de manera informal, cuando las personas que trabajan en el proyecto conocen diseños o códigos que son similares a lo que se requiere. Los buscan, los modifican según se necesite y los incorporan en sus sistemas. “Los enfoques orientados a la reutilización se apoyan en una gran base de componentes de software reutilizable y en la integración de marcos para la composición de dichos componentes” (Sommerville, 2011, p. 35).

La etapa inicial de especificación de requerimientos y la etapa de validación se comparan con otros procesos de software en un proceso orientado a la reutilización, las etapas intermedias son diferentes. Dichas etapas son:

*Análisis de componentes:* Dada la especificación de requerimientos, se realiza una búsqueda de componentes para implementar dicha especificación. Por lo general, no hay coincidencia exacta y los componentes que se usan proporcionan sólo parte de la funcionalidad requerida.

*Modificación de requerimientos:* Durante esta etapa se analizan los requerimientos usando información de los componentes descubiertos. Luego se modifican para reflejar los componentes disponibles. Donde las modificaciones son imposibles, puede regresarse a la actividad de análisis de componentes para buscar soluciones alternativas.

*Diseño de sistema con reutilización:* Durante esta fase se diseña el marco conceptual del sistema o se reutiliza un marco conceptual existente. Los creadores toman en cuenta los componentes que se reutilizan y organizan el marco de referencia para atenderlo. Es posible que deba diseñarse algo de software nuevo, si no están disponibles los componentes reutilizables.

*Desarrollo e integración:* Se diseña el software que no puede procurarse de manera externa, y se integran los componentes y los sistemas COTS para crear el nuevo sistema. La integración del sistema, en este modelo, puede ser parte del proceso de desarrollo, en vez de una actividad independiente.

La ingeniería de software orientada a la reutilización tiene la clara ventaja de reducir la cantidad de software a desarrollar y, por lo tanto, la de disminuir costos y riesgos; por lo general, también conduce a entregas más rápidas del software. Sin embargo, son inevitables los compromisos de requerimientos y esto conduciría hacia un sistema que no cubra las necesidades reales de los usuarios. Más aún, se pierde algo de control sobre la evolución del sistema, conforme las nuevas versiones de los componentes reutilizables no estén bajo el control de la organización que los usa.

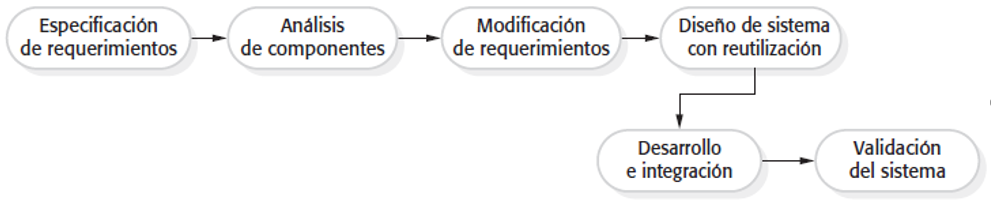


Figura 6. Ingeniería de software orientada a la reutilización

* 1. **Ciclo de vida orientado a objetos**

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.

No es correcto suponer que este modelo sólo es útil cuando se escoge para la implementación un lenguaje con orientación a objetos. Se puede utilizar independientemente del lenguaje elegido

Como mencionamos, es un modelo muy versátil, y por ser uno de los últimos en aparecer, aprendió mucho de los anteriores. Las aplicaciones que podemos incluir como ejemplo para su uso van desde programas de monitoreo de procesos, grandes sistemas de transacciones sobre base de datos, hasta procesamiento por lotes (Implementación y debugging,2006, p.33)

Ventajas:

* La calidad del producto resultante es alta.
* Permite trabajar con personal poco cualificado.
* La planificación es sencilla.

Desventajas:

* + - La necesidad de tener todos los requisitos al principio.
    - Si se comete un error en la fase de análisis no lo descubrimos hasta la entrega, con el consiguiente gasto inútil de recursos.

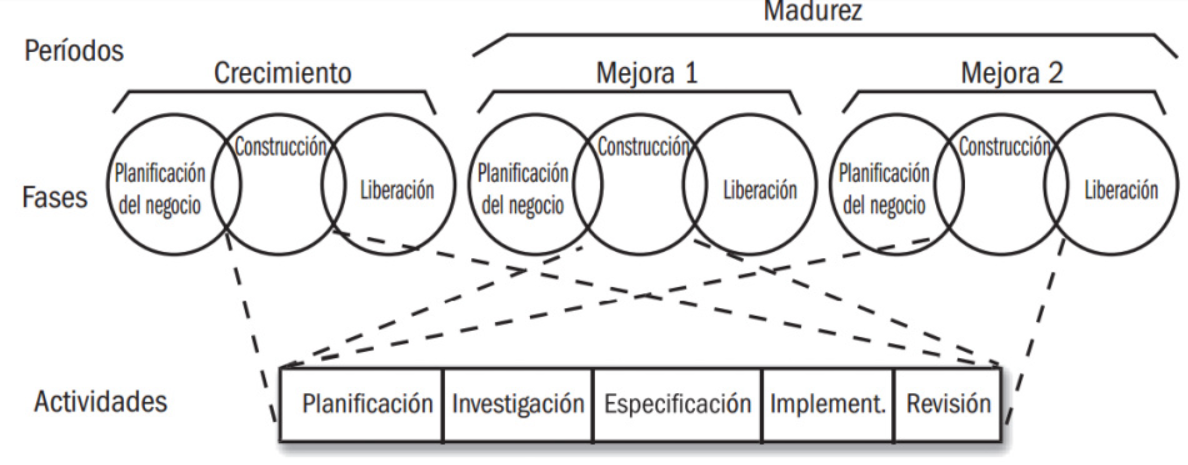


Figura 8. Ciclo de vida orientado a objetos

1. **Conclusiones.**

Después de leer los diferentes ciclos de vida se concluye que cada uno de estos tienen un uso diferente, siendo así de utilidad en diferentes aplicativos o proyectos

1. **Referencias.**
   1. Bibliografía.

Pressman R. S. (2010), *Ingeniería del software. Un enfoque práctico*, México: McGraw-Hill.

Sommerville, I. (2011), *Ingeniería de software*, México: Pearson Education, Inc.

Dante Cantone (2006), *Implementación y debugging* : *Creative Andina Corp.*

* 1. Webgrafía.

Apellidos y Nombres, Titulo, [medio informático], Año de publicación, Institución, Disponible en: <http://www.direccion>\_web [Fecha de consulta].