**Proceso para el desarrollo de software**

Es una estructura aplicada al desarrollo de [software](http://es.wikipedia.org/wiki/Software). Hay varios modelos a seguir para el establecimiento de un proceso para el desarrollo de software, cada uno de los cuales describe un enfoque diferente para diferentes actividades que tienen lugar durante el proceso. Algunos autores consideran un modelo de ciclo de vida un término más general que un determinado proceso para el desarrollo de software. Por ejemplo, hay varios procesos de desarrollo de software específicos que se ajustan a un modelo de ciclo de vida de espiral.

**Índice**

[2 Actividades del desarrollo de software](http://es.wikipedia.org/wiki/Proceso_para_el_desarrollo_de_software#Actividades_del_desarrollo_de_software)

[2.1 Planificación](http://es.wikipedia.org/wiki/Proceso_para_el_desarrollo_de_software#Planificaci.C3.B3n)

[2.2 Implementación, pruebas y documentación](http://es.wikipedia.org/wiki/Proceso_para_el_desarrollo_de_software#Implementaci.C3.B3n.2C_pruebas_y_documentaci.C3.B3n)

[2.3 Despliegue y mantenimiento](http://es.wikipedia.org/wiki/Proceso_para_el_desarrollo_de_software#Despliegue_y_mantenimiento)

[3 Modelos de desarrollo de software](http://es.wikipedia.org/wiki/Proceso_para_el_desarrollo_de_software#Modelos_de_desarrollo_de_software)

[3.1 Modelo de cascada](http://es.wikipedia.org/wiki/Proceso_para_el_desarrollo_de_software#Modelo_de_cascada)

[3.2 Modelo de espiral](http://es.wikipedia.org/wiki/Proceso_para_el_desarrollo_de_software#Modelo_de_espiral)

[3.3 Desarrollo iterativo e incremental](http://es.wikipedia.org/wiki/Proceso_para_el_desarrollo_de_software#Desarrollo_iterativo_e_incremental)

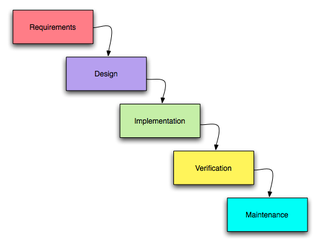
[3.4 Desarrollo ágil](http://es.wikipedia.org/wiki/Proceso_para_el_desarrollo_de_software#Desarrollo_.C3.A1gil)

[3.5 Codificación y corrección](http://es.wikipedia.org/wiki/Proceso_para_el_desarrollo_de_software#Codificaci.C3.B3n_y_correcci.C3.B3n)

[4 Modelos de mejora de procesos](http://es.wikipedia.org/wiki/Proceso_para_el_desarrollo_de_software#Modelos_de_mejora_de_procesos)

[5 Métodos formales](http://es.wikipedia.org/wiki/Proceso_para_el_desarrollo_de_software#M.C3.A9todos_formales)

**Actividades del desarrollo de software**

[](http://commons.wikimedia.org/wiki/File:Waterfall_model.png)

Actividades del proceso de desarrollo de software, representados [en cascada](http://es.wikipedia.org/wiki/Desarrollo_en_cascada).

**Planificación**

La importante tarea a la hora de crear un producto de software es obtener los [requisitos](http://es.wikipedia.org/wiki/Requisito_%28sistemas%29) o el [análisis de los requisitos](http://es.wikipedia.org/wiki/Ingenier%C3%ADa_de_requisitos). Los clientes suelen tener una idea más bien abstracta del resultado final, pero no sobre las funciones que debería cumplir el software.

Una vez que se hayan recopilado los requisitos del cliente, se debe realizar un análisis del ámbito del desarrollo. Este documento se conoce como especificación funcional.

**Implementación, pruebas y documentación**

La [implementación](http://es.wikipedia.org/wiki/Implementaci%C3%B3n) es parte del proceso en el que los [ingenieros de software](http://es.wikipedia.org/wiki/Ingenier%C3%ADa_del_software) [programan](http://es.wikipedia.org/wiki/Programaci%C3%B3n) el código para el proyecto.

Las [pruebas de software](http://es.wikipedia.org/wiki/Pruebas_de_software) son parte esencial del proceso de desarrollo del software. Esta parte del proceso tiene la función de detectar los [errores de software](http://es.wikipedia.org/wiki/Error_de_software) lo antes posible.

La [documentación](http://es.wikipedia.org/w/index.php?title=Documentaci%C3%B3n_del_software&action=edit&redlink=1) del diseño interno del software con el objetivo de facilitar su mejora y su mantenimiento se realiza a lo largo del proyecto. Esto puede incluir la documentación de un [API](http://es.wikipedia.org/wiki/API), tanto interior como exterior.

**Despliegue y mantenimiento**

El [despliegue](http://es.wikipedia.org/w/index.php?title=Despliegue_de_software&action=edit&redlink=1) comienza cuando el código ha sido suficientemente probado, ha sido aprobado para su [liberación](http://es.wikipedia.org/w/index.php?title=Liberaci%C3%B3n_del_software&action=edit&redlink=1) y ha sido distribuido en el entorno de producción.

[Entrenamiento y soporte para el software](http://es.wikipedia.org/wiki/Soporte_t%C3%A9cnico) es de suma importancia y algo que muchos desarrolladores de software descuidan. Los usuarios, por naturaleza, se oponen al cambio porque conlleva una cierta inseguridad, es por ello que es fundamental instruir de forma adecuada a los futuros usuarios del software.

El [mantenimiento](http://es.wikipedia.org/wiki/Mantenimiento_de_software) y mejora del software de un software con problemas recientemente desplegado puede requerir más tiempo que el desarrollo inicial del software. Es posible que haya que incorporar código que no se ajusta al diseño original con el objetivo de solucionar un problema o ampliar la funcionalidad para un cliente. Si los costes de mantenimiento son muy elevados puede que sea oportuno rediseñar el sistema para poder contener los costes de mantenimiento.

**Modelos de desarrollo de software**

Hay varios modelos para perfilar el proceso de desarrollo, cada uno de las cuales cuenta con pros y contras. El proyecto debería escoger el más apropiado para sus necesidades. En ocasiones puede que una combinación de varios modelos sea apropiado.

**Modelo de cascada**

Artículo principal: [Desarrollo en cascada](http://es.wikipedia.org/wiki/Desarrollo_en_cascada).

El modelo de cascada muestra un proceso donde los desarrolladores han de seguir las siguientes fases de forma sucesiva:

1. [Especificación de requisitos](http://es.wikipedia.org/wiki/Ingenier%C3%ADa_de_requisitos)
2. [Diseño del software](http://es.wikipedia.org/w/index.php?title=Dise%C3%B1o_del_software&action=edit&redlink=1)
3. [Construcción o Implementación del software](http://es.wikipedia.org/w/index.php?title=Construcci%C3%B3n_o_Implementaci%C3%B3n_del_software&action=edit&redlink=1)
4. [Integración](http://es.wikipedia.org/w/index.php?title=Integraci%C3%B3n_de_sistemas&action=edit&redlink=1)
5. [Pruebas](http://es.wikipedia.org/wiki/Pruebas_de_software) (o [validación](http://es.wikipedia.org/w/index.php?title=Validaci%C3%B3n&action=edit&redlink=1))
6. [Despliegue](http://es.wikipedia.org/w/index.php?title=Despliegue_de_software&action=edit&redlink=1) (o [instalación](http://es.wikipedia.org/wiki/Instalaci%C3%B3n_de_software))
7. [Mantenimiento](http://es.wikipedia.org/wiki/Mantenimiento_de_software)

Siguiendo el modelo de cascada de forma estricta, sólo cuando se finaliza una fase, comienza la otra. En ocasiones se realiza una revisión antes de iniciar la siguiente fase, lo que permite la posibilidad de cambios (lo que puede incluir un proceso de control formal de cambio). Las revisiones también se utilizan para asegurar que la fase anterior ha sido totalmente finalizada; los criterios para completar una fase se conocen frecuentemente con el término inglés "gate" (puerta). Este modelo desaconseja revisitar y revisar fases que ya se han completado. Esta falta de flexibilidad en un modelo de cascada puro ha sido fuente de crítica de los defensores de modelos más flexibles.

**Modelo de espiral**

Artículo principal: [Desarrollo en espiral](http://es.wikipedia.org/wiki/Desarrollo_en_espiral).

La principal características del modelo en espiral es la gestión de riesgos de forma periódica en el ciclo de desarrollo. Este modelo fue creado en 1988 por [Barry Boehm](http://es.wikipedia.org/wiki/Barry_Boehm), combinando algunos aspectos clave de las metodologías del modelo de cascada y del [desarrollo rápido de aplicaciones](http://es.wikipedia.org/wiki/Desarrollo_r%C3%A1pido_de_aplicaciones), pero dando énfasis en un área que para muchos no jugó el papel que requiere en otros modelos: un análisis iterativo y concienzudo de los riesgos, especialmente en el caso de sistema complejos de gran escala.

La espiral se visualiza como un proceso que pasa a través de algunas iteraciones con el diagrama de los cuatro cuadrantes representativos de las siguientes actividades:

1. crear planes con el propósito de identificar los objetivos del software,seleccionados para implementar el programa y clarificar las restricciones en el desarrollo del software;
2. Análisis de riesgos: una evaluación analítica de programas seleccionados, para evaluar como identificar y eliminar el riesgo;
3. la implementación del proyecto: implementación del desarrollo del software y su pertinente verificación;

Modelo de espiral con énfasis en los riesgos, haciendo hincapié en las condiciones de las opciones y limitaciones para facilitar la reutilización de software, la calidad del software puede ayudar como una meta propia en la integración en el desarrollo del producto. Sin embargo, el modelo en espiral tiene algunas limitaciones, entre las que destacan:

1. El énfasis se sitúa en el análisis de riesgo, y por lo tanto requiere de clientes que acepten este análisis y actúen en consecuencia. Para ello es necesaria confianza en los desarrolladores así como la predisposición a gastar más para solventar los temas, por lo cual este modelo se utiliza frecuentemente en desarrollo interno de software a gran escala.
2. Si la implementación del riesgo de análisis afectará de forma esencial los beneficios del proyecto, no debería utilizarse este modelo.
3. Los desarrolladores de software han de buscar de forma explícita riesgos y analizarlos de forma exhaustiva para que este modelo funcione.

La primera fase es la búsqueda de un plan para conseguir los objetivos con las limitaciones del proyecto para así buscar y eliminar todos los riesgos potenciales por medio de un cuidadoso análisis, y si fuera necesario incluyendo la fabricación de un prototipo. Si es imposible descartar algunos riesgos, el cliente ha de decidir si es conveniente terminar el proyecto o seguir adelante ignorando los riesgos. Por último, se evalúan los resultados y se inicia el diseño de la siguiente fase.

**Desarrollo iterativo e incremental**

El desarrollo iterativo recomienda la construcción de secciones reducidas de software que irán ganando en tamaño para facilitar así la detección de problemas de importancia antes de que sea demasiado tarde. Los procesos iterativos pueden ayudar a desvelar metas del diseño en el caso de clientes que no saben cómo definir lo que quieren.[1](http://es.wikipedia.org/wiki/Proceso_para_el_desarrollo_de_software#cite_note-1)

**Desarrollo ágil**

Artículo principal: [Desarrollo ágil de software](http://es.wikipedia.org/wiki/Desarrollo_%C3%A1gil_de_software).

El desarrollo ágil de software utiliza un desarrollo iterativo como base para abogar por un punto de vista más ligero y más centrado en las personas que en el caso de las soluciones tradicionales. Los procesos ágiles utilizan retroalimentación en lugar de planificación, como principal mecanismo de control. La retroalimentación se canaliza por medio de pruebas periódicas y frecuentes versiones del software.

Hay muchas variantes de los procesos ágiles:

* En el caso de la [programación extrema](http://es.wikipedia.org/wiki/Programaci%C3%B3n_extrema) (XP), las fases se realizan en pasos muy cortos (o "continuos") con respecto al anterior. El primer paso (intencionalmente incompleto) por los pasos puede ocurrir en un día o en una semana, en lugar de los meses o años de cada paso completo en el modelo en cascada. En primer lugar, se crean pruebas automatizadas para proveer metas concretas al desarrollo. Después se programa el código, que será completo cuando todas las pruebas se superan sin errores, y los desarrolladores ya no sabrían como mejorar el conjunto de pruebas necesario. El diseño y la arquitectura emergen a partir de la [refactorización](http://es.wikipedia.org/wiki/Refactorizaci%C3%B3n) del código, y se da después de programar. El diseño lo realizan los propios desarrolladores del código. El sistema, incompleto, pero funcional se despliega para su demostración a los usuarios (al menos uno de los cuales pertenece al equipo de desarrollo). Llegado este punto, los profesionales comienzan a escribir las pruebas para la siguiente parte del sistema de más importancia.

**Codificación y corrección**

El desarrollo de codificación y corrección (en inglés "Code and fix") es, más que una estrategia predeterminada, el resultado de una falta de experiencia o presión que se ejerce sobre los desarrolladores para cumplir con una fecha de entrega.[2](http://es.wikipedia.org/wiki/Proceso_para_el_desarrollo_de_software#cite_note-2) Sin dedicar tiempo de forma explícita para el diseño, los programadores comienzan de forma inmediata a producir [código](http://es.wikipedia.org/wiki/C%C3%B3digo_fuente). Antes o después comienza la fase de [pruebas de software](http://es.wikipedia.org/wiki/Pruebas_de_software) (a menudo de forma tardía) y los inevitables [errores](http://es.wikipedia.org/wiki/Error_de_software) que se encuentran han de eliminarse antes de poder entregar el software.

**Modelos de mejora de procesos**

Capability Maturity Model Integration

El [Capability Maturity Model Integration](http://es.wikipedia.org/wiki/Capability_Maturity_Model_Integration) (CMMI), en español «Integración de Modelos de Madurez de Capacidades» es uno de los modelos líderes basados en mejores prácticas. Son evaluaciones independientes las que confirman el grado con el que una organización siguen sus propios procesos, que no evalúa la calidad de los procesos o del software que se produce. CMMI ha reemplazado a [CMM](http://es.wikipedia.org/w/index.php?title=Capability_Maturity_Model&action=edit&redlink=1) y tiene un ámbito global, no sólo en procesos destinados al desarrollo del software.

ISO 9000

[ISO 9000](http://es.wikipedia.org/wiki/ISO_9000) describe estándares para un proceso organizado formalmente para resultar en un producto y los métodos de gestión y monitoreo del progreso. Aunque este estándar se creó inicialmente para el sector de producción, los estándares de ISO 9000 también se han aplicado al desarrollo del software. Al igual que CMMI, que una organización está certificada con el ISO 9000 no garantiza la calidad del resultado final, sólo confirma que se ha seguido los procesos establecidos.

ISO 15504

[ISO 15504](http://es.wikipedia.org/wiki/ISO_15504), también conocido como Software Process Improvement Capability Determination (SPICE), en español «Determinación de la Capacidad de Mejora del Proceso de Software» es un marco para la evaluación de procesos de software. Este estándar tiene como objetivo un modelo claro para poder comparar procesos. SPICE se utiliza como en el caso de CMMI. Modela procesos para gestionar, controlar, guiar y monitorear el desarrollo del software. Este modelo se utiliza entonces para medir lo que una organización o proyecto hace durante el desarrollo del software. Esta información se analiza para identificar puntos débiles y definir acciones para subsanarlos. También identifica puntos fuertes que pueden adoptarse en el resto de la organización.

**Métodos formales**

Los métodos formales son soluciones matemáticas para resolver problemas de software y hardware a nivel de requisitos, especificación y diseño. Ejemplos de métodos formales incluyen el [Método B](http://es.wikipedia.org/w/index.php?title=M%C3%A9todo_B&action=edit&redlink=1), la [red de Petri](http://es.wikipedia.org/wiki/Red_de_Petri), la [demostración automática de teoremas](http://es.wikipedia.org/wiki/Demostraci%C3%B3n_autom%C3%A1tica_de_teoremas), [RAISE](http://es.wikipedia.org/wiki/RAISE) y el [VDM](http://es.wikipedia.org/wiki/Vienna_Development_Method). Hay varias notaciones de especificaciones formales, tales como el [lenguaje Z](http://es.wikipedia.org/wiki/Lenguaje_Z). Más generalmente, se puede utilizar la [teoría de autómatas](http://es.wikipedia.org/wiki/Teor%C3%ADa_de_aut%C3%B3matas) para aumentar y validar el comportamiento de la aplicación diseñando un sistema de [autómata finito](http://es.wikipedia.org/wiki/Aut%C3%B3mata_finito).

Las metodologías basadas en los autómatas finitos permiten especificación de software ejecutable y evitar la creación convencional de código.

Los métodos formales se suelen aplicar en software de aviación, especialmente si es software de seguridad crítico. Los estándares de aseguramiento del software de seguridad, tales como [DO178B](http://es.wikipedia.org/wiki/DO178B) demandan métodos formales en el nivel más alto de categorización (Nivel A).

La formalización del desarrollo de software está ganando en fuerza poco a poco, en otros ámbitos, con la aplicación del [lenguaje de especificación OCL2.0](http://es.wikipedia.org/wiki/Lenguaje_de_especificaci%C3%B3n_OCL2.0) (y especializaciones tales como [Java Modeling Language](http://es.wikipedia.org/wiki/Java_Modeling_Language)) y particularmente con [Model-driven Architecture](http://es.wikipedia.org/wiki/Model-driven_Architecture), que permite la ejecución de diseños, incluso especificaciones.

Otra tendencia que está surgiendo en el desarrollo de software es la redacción de especificaciones en algún tipo de lógica (normalmente una variación de FOL), para acto seguido ejecutar esa lógica como si se tratase de un programa. El lenguaje [OWL](http://es.wikipedia.org/wiki/OWL), basado en lógica descriptiva, es un buen ejemplo. También se está trabajando en enlazar un idioma natural de forma automática con lógica, lógica que puede ejecutarse. Ejemplo en este campo es el [Attempto Controlled English](http://es.wikipedia.org/wiki/Attempto_Controlled_English), una lógica de negocios de Internet, que no busca controlar el vocabulario o la sintaxis. Una características de los sistemas que apoyan el vínculo bidireccional inglés-lógica y ejecución directa de la lógica es que pueden explicar sus resultados en inglés en un nivel de negocios o científico.