Ciclo de vida del software

Qué es el ciclo de vida del software

El ciclo de vida del desarrollo del software (también conocido como SDLC o *Systems Development Life Cycle*) contempla las fases necesarias para validar el desarrollo del software y así garantizar que este cumpla los requisitos para la aplicación y verificación de los procedimientos de desarrollo, asegurándose de que los métodos usados son apropiados.

Modelos de ciclos de vida del software

Modelo en cascada

Modelo repetitivo

Modelo en espiral

La crisis del software

- Planificaciones imprecisas que difícilmente se llegan a cumplir, superando los plazos de entrega en la mayoría de los proyectos ***software***.
- Los costes del proyecto suelen superar con creces el presupuesto inicial.
- Índices de productividad muy bajos.
- Clientes insatisfechos con las soluciones implementadas ya que no satisfacen sus necesidades.
- La calidad del ***software*** es inaceptable.
- El producto ***software*** desarrollado resulta muy difícil de mantener.
- El producto ***software*** no está integrado o ni siquiera alineado con el negocio de la compañía.
- El departamento de las Tecnologías de la Información (TI) suele verse como un freno al negocio.

Año	Entidad	Descripción	Nivel de gravedad
2015	Avión A400M	Un fallo en el ***software*** del ordenador que controlaba los motores fue la causa del accidente de que un avión A400M se estrellara en Sevilla y que provocara la muerte de cuatro tripulantes y heridas a otros dos.	Alta
2013	Juzgados de Madrid	Un fallo informático colapsa 46 juzgados de Madrid durante 12 días.	Media
2012	Gobierno de Canarias	Como consecuencia de un error informático, el gobierno canario pierde datos fundamentales acerca de la erupción volcánica de 'El Hierro'.	Media
2012	Gobierno de Navarra	Un fallo en el sistema informático de la red corporativa del Gobierno de Navarra interrumpe los servicios de Salud, Hacienda y Desarrollo Rural.	Media
2012	Aeropuerto de Málaga	Un fallo informático deja a oscuras la torre de control del aeropuerto de Málaga.	Alta
2011	Periódico 'El País'	El periódico 'El País' casi no sale a la calle un domingo por un fallo en el sistema informático y la ausencia de técnicos para poder solventarlo.	Media
2003	Vodafone España	La red de telefonía móvil de Vodafone en España sufre una caída de servicio dejando a más de 8 millones de usuarios sin cobertura.	Alta

Tabla 3: Errores informáticos más costosos de la Historia

Año	Entidad	Descripción	Nivel de gravedad
2010	Toyota	Toyota tuvo que retirar más de 400.000 de sus vehículos híbridos, debido a un problema ***software*** que provocaba un retraso en el sistema anti-bloqueo de frenos. Se estima que, entre sustituciones y demandas, el error le costó a Toyota unos 2,8 billones de Euros.	Alta
1999	NASA	Los ingenieros de la NASA perdieron el contacto con la sonda ***Mars Climate Orbiter*** en un intento que	Alta
1996	Agencia Espacial Europea (ESA)	El cohete Ariane 5 explotó debido a que un número real de 64 bits en coma flotante, relacionado con la velocidad, se convirtió en un entero de 16 bits. Las pérdidas se estiman en 400 millones de Euros.	Alta
1994	Intel	Un profesor de Matemáticas descubrió e informó acerca de un fallo en el procesador Pentium de Intel. La sustitución de chips costó a Intel uno 443 millones de Euros.	Alta
1988	Internet	Un estudiante de posgrado, Robert Tappan Morris, fue condenado por el primer ataque con 'gusanos' a gran escala en Internet. El coste de limpiar el desastre ocasionado por Robert se cifra en unos 100 millones de dólares.	Alta

Ética y responsabilidad profesional en la ingeniería del software

«

Sin una sólida educación ética específica de la profesión, el ingeniero se convierte en un mero instrumento técnico y despersonalizado en las manos de otros

»

8 principios básicos que deben seguir todos los profesionales de la ingeniería de software a la hora de ejercer su profesión

Público. Los ingenieros de software deberán actuar de manera consistente en bien del interés general

Cliente y contratista. Los ingenieros de software deberán actuar de tal modo que sea del mejor interés para sus clientes y contratistas, en coherencia con el interés general.

Producto. Los ingenieros de software deberán garantizar que sus productos y las modificaciones asociadas a ellos cumplen con el mayor número posible de los mejores estándares profesionales.

Juicio. Los ingenieros de software deberán mantener la integridad e independencia en su valoración profesional.

Gestión. Los gestores y líderes en ingeniería de software suscribirán y promoverán un enfoque ético en toda gestión de desarrollo y mantenimiento de software.

Profesión. Los ingenieros de software deberán realizar avances en lo que se refiere a integridad y reputación de la profesión, en coherencia con el interés general.

Compañeros. Los ingenieros de software serán justos y apoyarán a sus compañeros de profesión.

Persona. Los ingenieros de software deberán participar en el aprendizaje continuo de la práctica de su profesión y promoverán un comportamiento ético en la práctica de la misma.

actividades fundamentales para todos los procesos de desarrollo de software

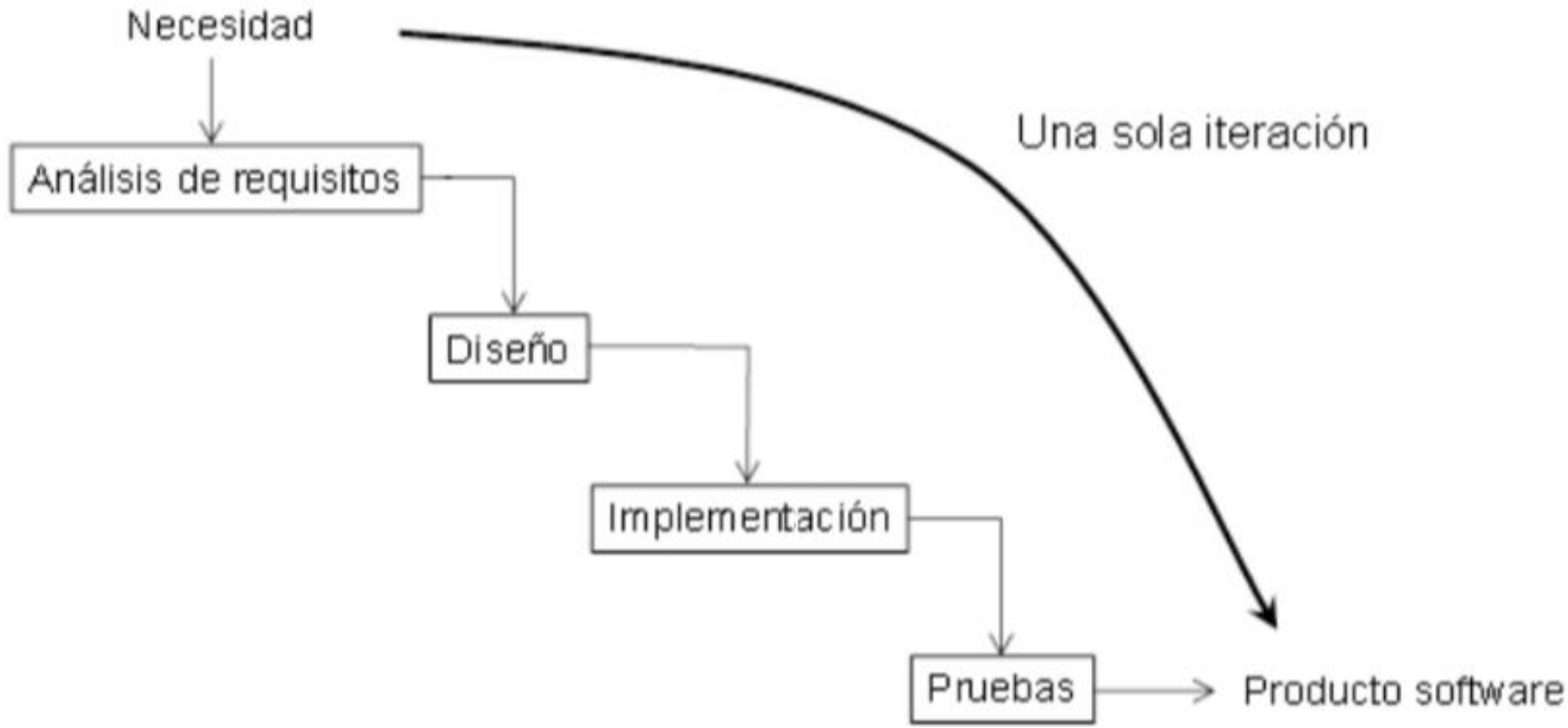
- **Especificación del *software***. Clientes e ingenieros definen el *software* a construir junto con sus restricciones.
- **Desarrollo del *software***. El *software* se diseña y se implementa, a la vez que se verifica que se construye de manera correcta.
- **Validación del *software***. El *software* se valida, para comprobar que el *software* es correcto, es decir, hace lo que el cliente había solicitado. La validación se llevará a cabo teniendo en cuenta los requisitos de usuario.
- **Evolución del *software***. El *software* se modifica para adaptarse a los cambios requeridos por el cliente o por el mercado.

Modelo de proceso software

Un modelo de proceso de desarrollo de software es una descripción simplificada de un proceso de desarrollo de software real.

Modelo en cascada

El modelo en cascada (***waterfall model***) fue el primer paradigma de proceso de desarrollo de ***software*** reconocido (ver Figura 2), y se deriva de otros procesos de ingeniería utilizados para la construcción de productos físicos. Este modelo toma las actividades comunes para todos los procesos de desarrollo de ***software*** (especificación, desarrollo, validación y evolución), y las representa como fases individuales secuenciales: especificación de requisitos, diseño de ***software***, implementación, pruebas, etc. El resultado de cada fase ha de ser aprobado (***signed off***). Una fase no puede comenzar hasta no haber finalizado (aprobado) la anterior. Desgraciadamente, este modelo de proceso resulta poco realista, ya que difícilmente se van a encontrar proyectos ***software*** reales que se ajusten a este tipo de modelo de proceso de manera tan rigurosa.

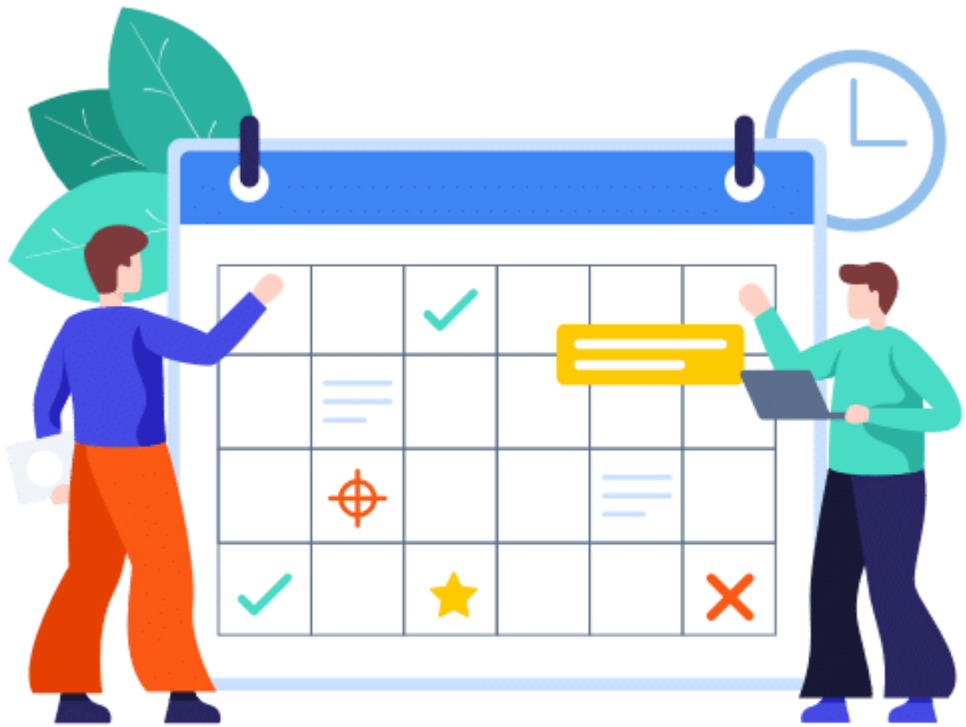

Necesidad
Una sola iteración
Análisis de requisitos
Diseño
Implementación
Pruebas
Producto software

Comunicación

Este es el momento en el que un cliente solicita un producto de software determinado. Nos contacta para plasmar sus necesidades concretas y presenta su solicitud de desarrollo de software.

Planificación y análisis

El desarrollo de software comienza con una fase inicial de planificación incluyendo un análisis de requisitos. Nos fijamos en los requisitos que piden los clientes para estudiar cuales están poco claros, incompletos, ambiguos o contradictorios. Se indaga en profundidad y se hacen demostraciones prácticas incluyendo a los usuarios clave. Los requisitos se agrupan en requisitos del usuario, requisitos funcionales y requisitos del sistema. La recolección de todos los requisitos se lleva a cabo: estudiando el software actual que tengan, entrevistando a usuarios y desarrolladores, consultando bases de datos o mediante cuestionarios.



Estudio de viabilidad

Después de la recolección de requisitos, se idea un plan para procesar el software. Se analiza que parte del software cubre los requisitos de cada usuario. Se investiga la viabilidad financiera y tecnológica. Se utilizan algoritmos para saber si el proyecto de software es factible o no.

Análisis del sistema

En este paso el equipo del proyecto asigna recursos y planifica el tiempo de duración del proyecto. Se buscan limitaciones del producto y se identifican los impactos del proyecto sobre toda la organización en su conjunto.

Diseño

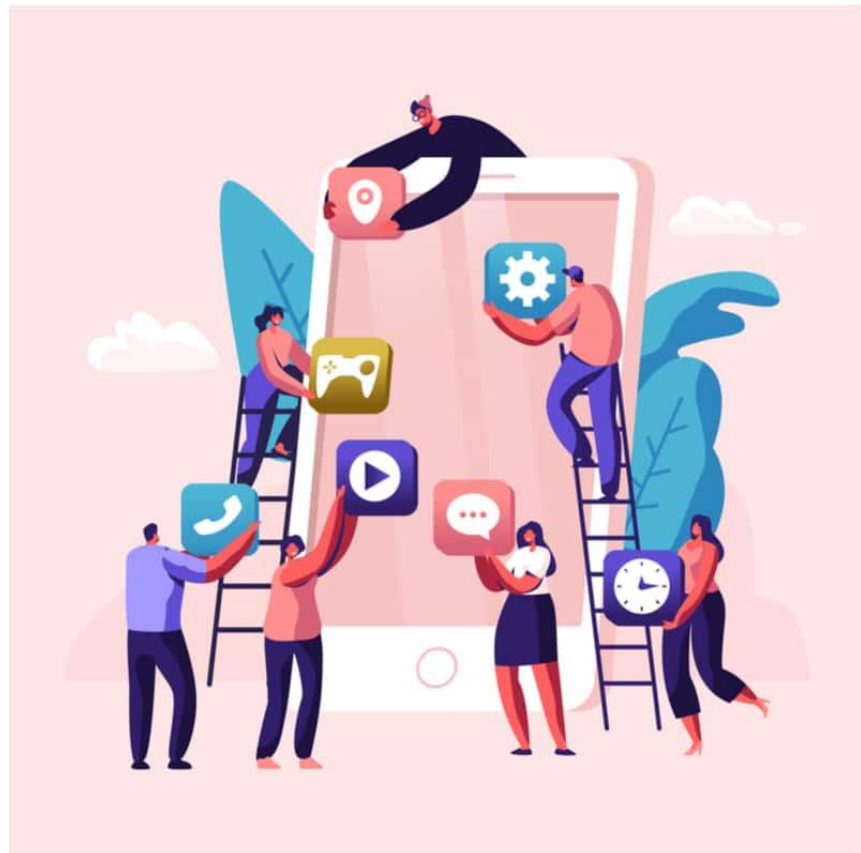
En esta fase ya se comienza a visualizar la solución con la ayuda de las anteriores fases. Se hace un diseño lógico y otro físico. Se crean metadatos, diagramas o pseudocódigos. La duración de esta fase varía de un proyecto a otro.

Codificación

Esta fase también denominada ‘fase de programación’ o ‘fase de desarrollo’ es en la que elige el lenguaje de programación más conveniente, y se desarrollan programas ejecutables y sin errores de manera eficiente. Nuestro enfoque es construir trozos de funcionalidad. Por lo tanto, entregar unidades de funcionalidad concisa. Al final de esta fase se puede obtener un PMV (Producto mínimo viable) o el software completamente desarrollado y listo para implementarse.

Integración

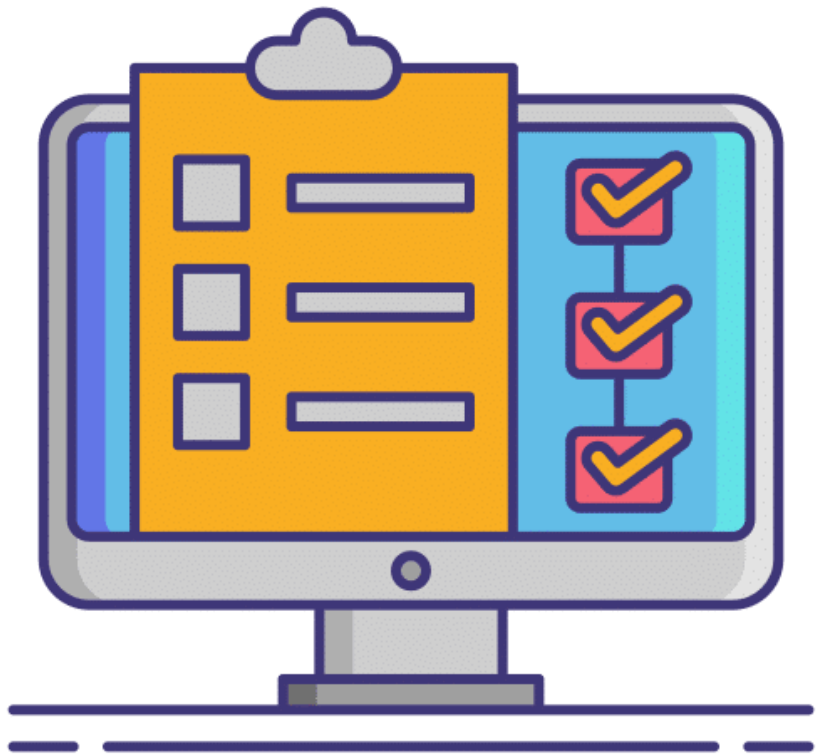
El Software puede necesitar estar integrado con bibliotecas, bases de datos o con otros programas. Esta fase del SDLC integra el software con las entidades del mundo exterior.



Pruebas

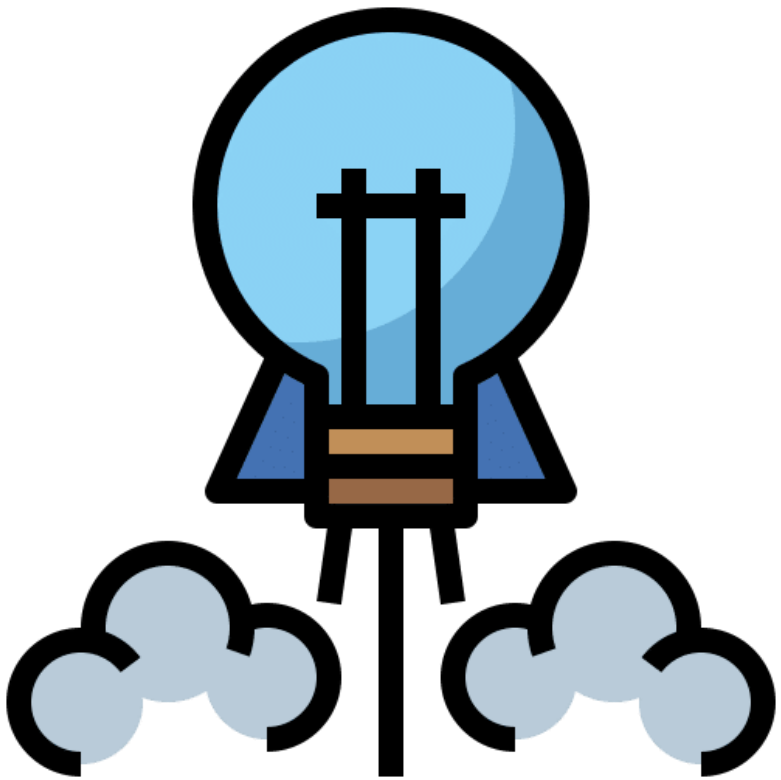
Esta fase junto con la fase de desarrollo entra en un ciclo continuo hasta que se completan el desarrollo y las pruebas. Probamos, probamos y luego volvemos a probar tanto como sea necesario hasta que la funcionalidad sea del 100%.

Además se hacen evaluaciones para evitar errores, incluyendo la evaluación de módulos, programas, productos, y finalmente evaluación con el cliente final. Encontrar errores y su arreglarlos a tiempo es la clave para conseguir un software confiable y eficiente.



Implementación

Aquí se instala el software, se evalúa la integración, la adaptabilidad, la portabilidad y se instalan las configuraciones posteriores necesarias.



Formación

Esta es la fase más interesante, ¡La formación! La adopción del usuario es muy importante y para ello ofrecemos capacitación inicial para cada usuario. Es importante comprobar el nivel de uso, la experiencia de usuario y resolver cualquier dificultad que pueda surgir a la hora de enfrentarse a un nuevo sistema o plataforma.

Mantenimiento y Funcionamiento

Por último, pero no menos importante el mantenimiento es uno de los elementos clave de éxito de cualquier proyecto. En esta fase se minimizan pequeños errores, se confirma el buen funcionamiento del software, su eficiencia y estabilidad. El proyecto ya está completado y necesitamos monitorear y mantener de forma continua para garantizar que el proyecto siga ejecutándose bien.

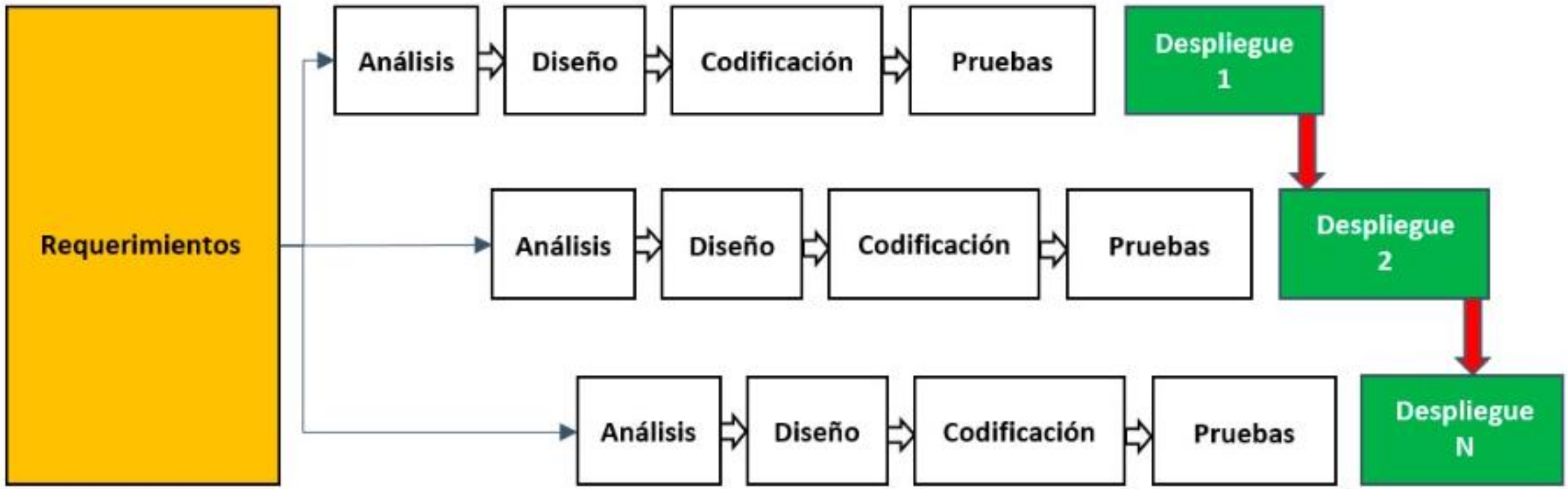
https://standards.ieee.org/ieee/12207/5672/

Si es necesario se dan nuevas formaciones, o se presta documentación sobre como operar y mantener el software en perfecto estado de funcionamiento.

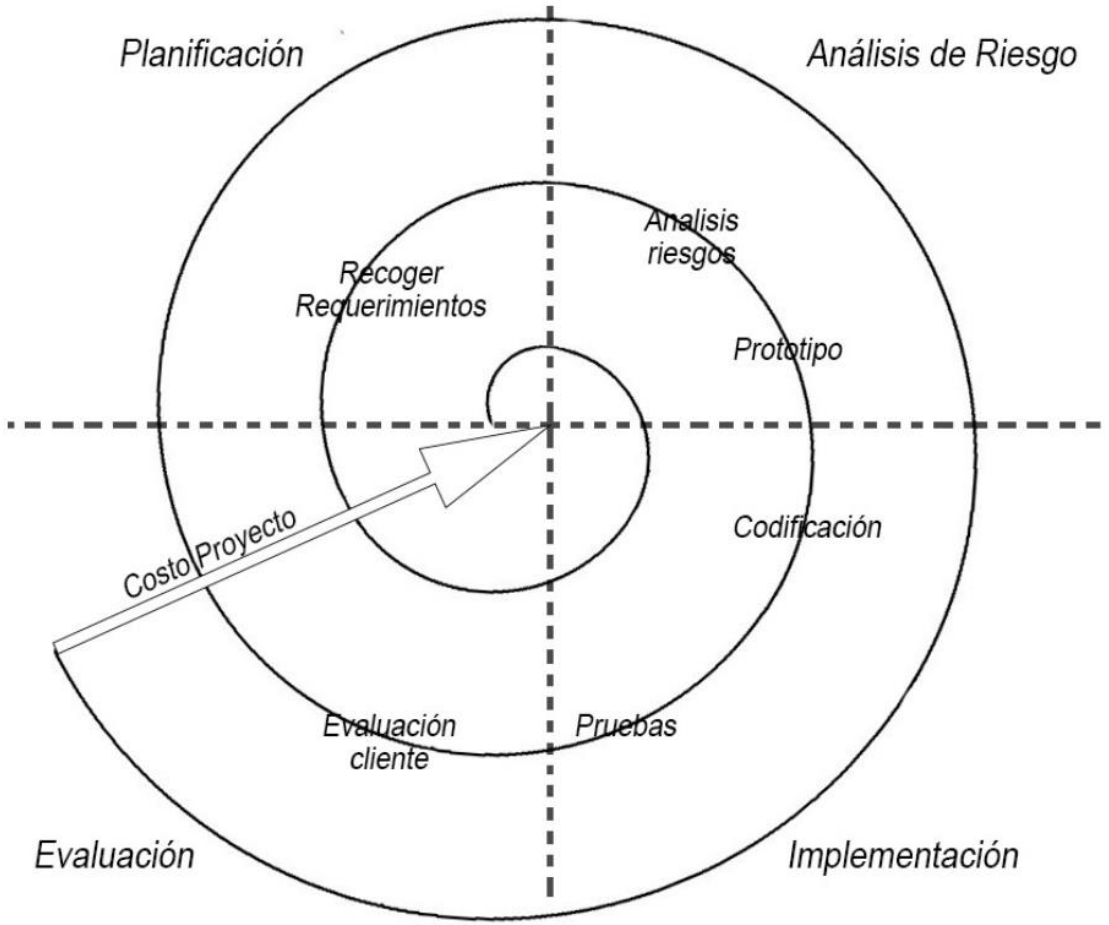
Se adaptan entornos del usuario o tecnológicos, dando mantenimiento al software, actualizando el código y configuración.

El software es efectivo cuando se usa de forma apropiada y por eso el mantenimiento y la mejora de los productos de software es crucial para poder corregir defectos que vayan surgiendo o para poder atender a los requisitos del software.

INCREMENTAL: Todos los requerimientos se dividen en varias compilaciones. Aquí se producen varios ciclos de desarrollo, por lo que podríamos llamar a este ciclo "multi cascada". Los ciclos se dividen en módulos más pequeños y más fáciles de gestionar. Cada módulo pasa por las fases de requerimientos, diseño, implementación y pruebas. Una versión del software se produce durante el primer módulo, teniendo un módulo activo y trabajando durante el ciclo de vida del software. Cada versión posterior del módulo añade funciones a la versión anterior.


Requerimientos
Análisis
Diseño
Codificación
Pruebas
Despliegue 1
Análisis
Diseño
Codificación
Pruebas
Despliegue 2
Análisis
Diseño
Codificación
Pruebas
Despliegue N

ESPIRAL: El modelo espiral es similar al modelo incremental, con mayor énfasis en el análisis de riesgo. El modelo espiral tiene cuatro fases: Planificación, Análisis de Riesgos, Implementación y Evaluación. Un proyecto pasa repetidamente por estas fases en iteraciones (llamadas espirales, que es de donde este modelo toma su nombre). En la espiral inicial, se recopilan los requisitos y se evalúa el riesgo. Cada espiral posterior se basa en la espiral base.



* Planificación: Los requisitos se recogen durante la fase de planificación.

* Análisis de riesgo: Se emprende un proceso para identificar el riesgo y soluciones alternativas. Al final de este análisis, se crea un prototipo. Si se encuentra algún riesgo durante el análisis, entonces se sugieren soluciones alternativas a implementar.

* Implementación: En esta fase se desarrolla el software, y al final de la mismo sus respectivas pruebas.

* Evaluación: Esta fase permite al cliente evaluar el proyecto terminado hasta esa fecha, antes de que el proyecto pase a la siguiente espiral.

MODELO V: Al igual que el modelo de cascada, el modelo V es una ruta secuencial de ejecución de procesos. Cada fase debe completarse antes de que comience la siguiente fase.

MODELO RAD (Rapid Application Development por sus siglas en inglés): Es un modelo incremental que se centra en desarrollar software en ciclos cortos de tiempo. En este modelo, los componentes o funciones se desarrollan en paralelo como si fueran mini proyectos. El desarrollo se empaqueta (compila) y se entrega como un prototipo. Esto puede dar rápidamente al cliente algo para ver y utilizar y para proporcionar comentarios y nuevos requisitos. Este modelo es un modelo "de alta velocidad" que adapta muchos pasos del modelo de cascada para alcanzar sus objetivos con un crecimiento rápido en sus ciclos.

MODELOS DE CICLO DE VIDA DE SOFTWARE: CUADRO COMPARATIVO			
Modelos	***Ventajas***	***Desventajas***	***Cuando usarlo***
Cascada	* Simple y sencillo de entender y usar. * Es fácil de manejar debido a su rigidez del modelo, cada fase tiene resultados específicos y un proceso de revisión. * En este modelo las fases son procesadas y completadas una a la vez. Las fases no se superponen. * Ideal para proyectos pequeños.	* Una vez que una aplicación está en la etapa de prueba, es muy difícil volver atrás y cambiar algo que no estaba pensado en la etapa de diseño. * Solo se programa una sola vez durante el ciclo de vida. * El riesgo y la incertidumbre son grandes. * No es un buen modelo para proyectos complejos y orientados a objetos. * Modelo deficiente para proyectos largos y en curso. * No es adecuado para los proyectos en los que los requisitos están sujetos a cambios.	* Este modelo se utiliza sólo cuando los requisitos son claros y fijos. * Cuando se implementará en tecnologías conocidas, es decir, no se innovará nada. * No hay requisitos ambiguos. * Los recursos están disponibles libremente. * Para proyectos cortos.
Incremental	* Genera tempranamente y rápidamente software durante el ciclo de vida. * Es más flexible, los costos de los cambios son menores. * Es más fácil probar y depurar durante una iteración menor. * En este modelo el cliente puede responder en cada despliegue. * Riesgo más fácil de manejar porque las piezas de riesgo son identificadas y manejadas durante su iteración.	* Necesita buena planificación y diseño. * Necesita una definición clara y completa de todo el sistema antes de que pueda modularse y empezar a construir de forma incremental. * El costo total es más alto que el modelo cascada.	* Este modelo puede utilizarse cuando los requisitos del sistema completo están claramente definidos y comprendidos, sin embargo, algunos detalles pueden evolucionar con el tiempo. * Cuando hay necesidad de tener una versión previa lista. * Se está utilizando una nueva tecnología (se va a innovar). * Cuando muchos recursos no están disponibles. * Cuando el riesgo es medio.
Espiral	* Las estimaciones (es decir, el presupuesto, el cronograma, etc.) se vuelven más realistas a medida que el trabajo avanza, porque tempranamente se van descubriendo problemas. * Es capaz de hacer frente a los cambios que generalmente implica el desarrollo de software. * Se puede empezar a trabajar en un prototipo caso al comienzo del ciclo.	* Limitantes en la reutilización y la personalización. * Se debe aplicar de forma diferente en cada aplicación. * Riesgo alto de no cumplir expectativas, presupuestos y horarios (si no se planea bien).	* Cuando los costos y la evaluación del riesgo son importantes. Para proyectos de mediano a alto riesgo. * Cuando el plazo del proyecto es incierto, debido a posibles cambios por prioridades económicas. * Los usuarios no están seguros de sus necesidades. * Los requisitos son complejos. Cuando se tratará con nueva línea de productos. * Se esperan cambios significativos (investigación y exploración).