CURSO DE INTRODUCCIÓN A LA INGENIERÍA DEL SOFTWARE

Laboratorio Nacional de Calidad del Software

NOTA DE EDICIÓN

Este curso ha sido desarrollado por el Laboratorio Nacional de Calidad del Software de INTECO. Esta primera versión ha sido editada en Junio del 2009.

AVISO LEGAL

- Las distintas normas ISO mencionadas han sido desarrolladas por la International Organization for Standardization.

Todas las demás marcas registradas que se mencionan, usan o citan en el presente curso son propiedad de los respectivos titulares.

INTECO cita estas marcas porque se consideran referentes en los temas que se tratan, buscando únicamente fines puramente divulgativos. En ningún momento INTECO busca con su mención el uso interesado de estas marcas ni manifestar cualquier participación y/o autoría de las mismas.

Nada de lo contenido en este documento debe ser entendido como concesión, por implicación o de otra forma, y cualquier licencia o derecho para las Marcas Registradas deben tener una autorización escrita de los terceros propietarios de la marca.

Por otro lado, INTECO renuncia expresamente a asumir cualquier responsabilidad relacionada con la publicación de las Marcas Registradas en este documento en cuanto al uso de ninguna en particular y se eximen de la responsabilidad de la utilización de dichas Marcas por terceros.

El carácter de todos los cursos editados por INTECO es únicamente formativo, buscando en todo momento facilitar a los lectores la comprensión, adaptación y divulgación de las disciplinas, metodologías, estándares y normas presentes en el ámbito de la calidad del software.

ÍNDICE

Escenario de apertura

La empresa COMPASS.SA se ha dado cuenta de la necesidad de adquirir una herramienta de gestión de RRHH, ya que su plantilla ha aumentado de forma considerable y el método que estaban usando hasta hora se ha visto desbordado.

La directiva se pone en contacto con la empresa proveedora de software con la que suelen trabajar.

Figura 1. Escenario de apertura I

Figura 2. Escenario de apertura II

A lo largo del curso vamos a ver cómo fue el piloto con la herramienta y los problemas con los que se encontró la empresa.

Introducción

Hoy en día, el software es una parte integral de la mayoría de los sistemas. Para ejecutar proyectos software de forma satisfactoria y construir productos de alta calidad, los profesionales del software necesitan entender las características únicas del software y el enfoque usado para desarrollar y mantener software.

Este curso permitirá entender qué es el software y cuáles son los objetivos y componentes de la ingeniería del software, así como entender los conceptos de ciclo de vida del software y metodología. Además, se verán los principales modelos de ciclo de vida del software y la diferenciación entre metodologías tradicionales y ágiles.

Software

Hoy en día, el software es una parte integral de la mayoría de los sistemas y es un importante diferenciador de negocio. En este apartado veremos qué es el software y cuáles son sus componentes.

El **software** son los programas y la documentación asociada tal como requisitos, modelos de diseño y manuales de usuario.

Los productos software pueden desarrollarse para un cliente particular o se pueden desarrollar para un mercado general. Por lo tanto, los productos software pueden ser:

- Genéricos: desarrollados para ser vendidos a un ámbito de clientes diferentes.
- Hechos a medida (personalizados): desarrollados para un cliente individual de acuerdo a su especificación.

Se puede crear software nuevo desarrollando nuevos programas, configurando sistemas de software genéricos o reutilizando software existente.

Atributos de un buen software

El software debe proporcionar la funcionalidad y el rendimiento requeridos a los usuarios y debe ser sostenible, fiable y aceptable.

- **Mantenibilidad** (capacidad de poder mantenerse): el software debe evolucionar para cumplir con las necesidades de cambio.
- **Fiabilidad**: el software deber ser digno de confianza.
- **Eficiencia**: el software no debe hacer un uso derrochador de los recursos del sistema.
- **Aceptabilidad**: el software debe ser aceptado por los usuarios para los que se diseñó. Esto significa que ha de ser entendible, usable y compatible con otros sistemas.

Componentes del software

El software se puede definir como el conjunto de tres componentes:

- **Programas** (instrucciones): los programas son conjuntos de instrucciones que proporcionan la funcionalidad deseada y el rendimiento cuando se ejecutan. Están escritos usando lenguajes específicos que los ordenadores pueden leer y ejecutar, tales como lenguaje ensamblador, Basic, FORTRAN, COBOL, C… Los programas también pueden ser generados usando generadores de programas.
- **Datos**: este componente incluye los datos necesarios para manejar y probar los programas y las estructuras requeridas para mantener y manipular estos datos. Los programas proporcionan la funcionalidad requerida manipulando datos. Usan datos para ejercer el control apropiado en lo que hacen. El mantenimiento y las pruebas de los programas también necesitan datos. El diseño del programa asume la disponibilidad de las estructuras de datos tales como bases de datos y archivos que contienen datos.
- **Documentos**: este componente describe la operación y uso del programa. Además de los programas y los datos, los usuarios necesitan también una explicación de cómo usar el programa. Documentos como manuales de usuario y de operación son necesarios para permitir a los usuarios operar con el sistema. Los documentos también son requeridos por las personas encargadas de mantener el software para entender el interior del software y modificarlo, en el caso en que sea necesario.

Figura 3. Componentes del software

Es importante contar con una definición exhaustiva del software ya que de otra manera se podrían olvidar algunos componentes. Una percepción común es que el software sólo consiste en programas, sin embargo, los programas no son los únicos componentes del software.

Se hizo un piloto con la herramienta que les proporcionó el proveedor con algunas de las personas del departamento de RRHH.

Figura 4. Evaluación uso de herramienta de RRHH I

Figura 5. Evaluación uso de herramienta de RRHH II

Características del software

Hoy en día, casi todo el mundo se ve afectado por el amplio uso del software, bien como usuario o bien como profesional encargado de construirlo. Los usuarios han de apreciar las ventajas de usar software, mientras que los profesionales necesitan entender las características únicas del software de forma que puedan construir software de alta calidad y realizar proyectos software de forma satisfactoria.

Para poder comprender lo que es el software (y consecuentemente la ingeniería del software), es importante examinar las características del software que lo diferencian de otras cosas que el hombre puede construir.

El software es un elemento lógico y se diferencia del hardware, un elemento físico, en sus características.

El software se desarrolla

No se fabrica en el sentido clásico. Aunque existen similitudes entre el desarrollo del software y la construcción del hardware, ambas actividades son fundamentalmente distintas.

Cada producto software es diferente porque se construye para cumplir los requisitos únicos de un cliente. Cada software necesita, por lo tanto, ser construido usando un enfoque de ingeniería.

Construir un producto software implica entender qué es necesario, diseñar el producto para que cumpla los requisitos, implementar el diseño usando un lenguaje de programación y comprobar que el producto cumple con los requisitos. Todas estas actividades se llevan a cabo mediante la ejecución de un proyecto software y requiere un equipo trabajando de una forma coordinada.

En el software, el recurso principal son las personas. No siempre es posible acelerar la construcción de software añadiendo personas porque el desarrollo de software requiere un esfuerzo en equipo. El equipo tiene que trabajar de forma coordinada y compartir un objetivo de proyecto común. Se necesita comunicación efectiva dentro del equipo. Un nuevo miembro del equipo no es inmediatamente productivo y necesita la iniciación adecuada al equipo y la formación para realizar el trabajo. Esto requiere una inversión de tiempo y

esfuerzo por parte de los miembros del equipo existentes y les puede distraer de su propio trabajo.

El software no se estropea

Los defectos no detectados harán que falle el programa durante las primeras etapas de su vida. Sin embargo, una vez que se corrigen (suponiendo que no se introducen nuevos errores) los fallos disminuyen.

El software no se estropea, pero se deteriora. Durante su vida, el software sufre cambios (mantenimiento). Conforme se hacen los cambios, es bastante probable que se introduzcan nuevos defectos, lo que hace que el software se vaya deteriorando debido a los cambios.

El software se construye a medida

Otro aspecto del software es que, debido a que la industria del software es nueva, el software se diferencia del hardware en el aspecto relacionado con el uso de componentes. Aunque la mayoría de la industria tiende a ensamblar componentes, en el caso del software, la mayoría **se construye a medida**. Aunque la reutilización y ensamblaje de componentes está aumentando, el software con frecuencia se construye de acuerdo a los requisitos específicos de un cliente.

Los componentes reutilizables se han creado para que el ingeniero pueda concentrarse en elementos verdaderamente innovadores de un diseño. El componente software debería diseñarse e implementarse para que pueda volver a ser reutilizado en muchos programas diferentes. Hoy en día, se ha extendido la visión de la reutilización para abarcar tanto algoritmos como estructuras de datos, permitiendo al ingeniero del software crear nuevas aplicaciones a partir de las partes reutilizables.

El hardware usa componentes estándar con funciones e interfaces bien definidas. La fase de diseño en el ciclo de vida de un producto hardware implica seleccionar los componentes disponibles más adecuados y decidir el enfoque para montarlos. Los componentes de hardware estándar son útiles porque conducen a:

- Reducir el coste y el tiempo de lanzamiento al mercado
- Buena calidad
- Ingeniería rápida

- Fácil mantenimiento
- Fácil mejora

Como la industria del hardware, la industria del software está intentando adoptar el mecanismo de reutilizar para hacer más fácil y más rápida la construcción. Las ventajas de la reutilización de software están siendo entendidas y apreciadas. Existen algunos elementos reutilizables a través de librerías de funciones y objetos reutilizables que combinan funciones y datos.

Mientras que la reutilización y el montaje basado en componentes se están incrementando, la mayoría del software continua siendo construido de forma personalizada, y los niveles de reutilización actuales están lejos de los que deberían ser. Además, la tarea de identificar componentes reutilizables potenciales es difícil porque cada producto software es único y distinto.

La industria del software tiene procesos bien definidos para la reutilización de componentes. Esto incluye procesos para la construcción de componentes, almacenamiento de los mismos en librerías de donde se pueden extraer para su reutilización.

Con el paso de los años, la industria del software espera crear componentes reutilizables específicos a dominios de aplicación particulares.

Aplicaciones del software

El software puede aplicarse en cualquier situación en la que se haya definido previamente un conjunto específico de pasos procedimentales, es decir, un algoritmo (excepciones notables a esta regla son el software de los sistemas expertos y de redes neuronales).

Algunas veces es difícil establecer categorías genéricas para las aplicaciones del software que sean significativas. Conforme aumenta la complejidad del software, es más difícil establecer compartimentos nítidamente separados. Las siguientes áreas del software indican la amplitud de las aplicaciones potenciales:

- **Software de sistemas**: engloba el conjunto de programas que han sido escritos para servir a otros programas. Algunos programas de sistemas (por ejemplo: compiladores, editores y utilidades de gestión de archivos) procesan estructuras de

información complejas pero determinadas. Otras aplicaciones de sistemas (por ejemplo: ciertos componentes del sistema operativo, utilidades de manejo de periféricos, procesadores de telecomunicaciones) procesan datos en gran medida indeterminados. En cualquier caso, el área del software de sistemas se caracteriza por una fuerte interacción con el hardware de la computadora; una gran utilización por múltiples usuarios; una operación concurrente que requiere una planificación, una compartición de recursos y una sofisticada gestión de procesos; unas estructuras de datos complejas y múltiples interfaces externas.

- **Software de tiempo real**: coordina/analiza/controla sucesos del mundo real conforme ocurren. Entre los elementos del software de tiempo real se incluyen: un componente de adquisición de datos que recolecta y da formato a la información recibida del entorno externo, un componente de análisis que transforma la información según lo requiera la aplicación, un componente de control/salida que responda al entorno externo y un componente de monitorización que coordina todos los demás componentes, de forma que pueda mantenerse el respuesta en tiempo real.

- **Software de gestión**: se ocupa del tratamiento de la información comercial y constituye la mayor de las áreas de aplicación del software. Los sistemas discretos (por ejemplo: nóminas, cuentas de haberes-débitos, inventarios, etc.) han evolucionado hacia el software de sistemas de información de gestión (SIG) que accede a una o más bases de datos que contienen información comercial. Las aplicaciones en esta área reestructuran los datos existentes para facilitar las operaciones comerciales o gestionar la toma de decisiones. Además de las tareas convencionales de procesamiento de datos, las aplicaciones de software de gestión también realizan cálculo interactivo (por ejemplo: el procesamiento de transacciones en puntos de venta).

- **Software de ingeniería y científico**: se caracteriza por los algoritmos de manejo de números. Las aplicaciones van desde la astronomía a la vulcanología, desde el análisis de la presión de los automotores a la dinámica orbital de las lanzaderas espaciales y desde la biología molecular a la fabricación automática. Sin embargo las nuevas aplicaciones del área de ingeniería/ciencia se han alejado de los algoritmos convencionales numéricos. El diseño asistido por computadora (CAD), la simulación

de sistemas y otras aplicaciones interactivas, han comenzado a coger características del software de tiempo real e incluso de software de sistemas.

- **Software empotrado**: los productos inteligentes se han convertido en algo común en casi todos los mercados de consumo e industriales. El software empotrado reside en memoria de sólo lectura y se utiliza para controlar productos y sistemas de los mercados industriales y de consumo. El software empotrado puede ejecutar funciones muy limitadas y curiosas (por ejemplo: el control de las teclas de un horno microondas) o suministrar una función significativa y con capacidad de control (por ejemplo: funciones digitales en un automóvil, tales como control de la gasolina, indicadores en el salpicadero, sistemas de frenado, etc.)
- **Software de computadoras personales**: el mercado del software de computadoras personales ha germinado en las pasadas décadas. El procesamiento de textos, las hojas de cálculo, los gráficos por computadora, multimedia, entretenimiento, gestión de bases de datos, aplicaciones financieras, de negocios y personales y redes o acceso a bases de datos externas son algunas de los cientos de aplicaciones.
- **Software basado en web**: las páginas web buscadas por un explorador son software que incorpora instrucciones ejecutables y datos.
- **Software de inteligencia artificial**: hace uso de algoritmos no numéricos para resolver problemas complejos para los que no son adecuados el cálculo o el análisis directo. Los sistemas expertos, también llamados sistemas basados en el conocimiento, reconocimiento de patrones (imágenes y voz), redes neuronales artificiales, prueba de teoremas y los juegos son representativos de las aplicaciones de esta categoría.

Ingeniería del Software

Como disciplina, la ingeniería el software ha progresado mucho en un corto periodo de tiempo. Actualmente, se construyen sistemas muy grandes en cuanto a tamaño y complejidad y el software está presente en casi todos los aspectos de la vida.

A pesar del rápido progreso, todavía existen grandes problemas para conseguir proporcionar a los clientes productos software de alta calidad en los plazos establecidos. Hay muchos desafíos que hay que tratar para progresar hacia un campo de la ingeniería más maduro que permita obtener productos de alta calidad.

En este apartado se tratará de dar una definición clara de lo que es la ingeniería del software, explicando cada uno de sus componentes.

Definición de Ingeniería del Software

La ingeniería del software es una disciplina de ingeniería preocupada por todos los aspectos de la producción de software desde las primeras etapas de especificación del sistema hasta el mantenimiento del sistema después de que éste se haya puesto en uso. Se preocupa de las teorías, métodos y herramientas para el desarrollo profesional de software. La ingeniería del software se preocupa del desarrollo de software rentable.

La ingeniería del software debería adoptar un enfoque sistemático y organizado para su trabajo y usar las herramientas y técnicas apropiadas dependiendo del problema a solucionar, las restricciones de desarrollo y los recursos disponibles.

La **ingeniería del software** es la aplicación de un enfoque sistemático, disciplinado y cuantificable para el desarrollo, operación y mantenimiento del software, que es la aplicación de la ingeniería del software (IEEE, 1990).

El enfoque sistemático, disciplinado y cuantificable es con frecuencia calificado de modelo de proceso de software (en el sentido general) o de proceso de desarrollo de software (en el sentido específico). El proceso de desarrollo de software específico consiste en un conjunto particular de **prácticas de desarrollo de software** que son realizadas por el ingeniero de software en un orden predeterminado.

Los ingenieros adoptan un enfoque sistemático y organizacional para su trabajo. Cuando se habla de prácticas de desarrollo de software se hace referencia a un requisito empleado para recomendar un enfoque disciplinado y uniforme del proceso de desarrollo de software, es decir, una actividad bien definida que contribuye a la satisfacción de los objetivos del proyecto; generalmente la salida de una práctica se convierte en la entrada de otra. Entre las prácticas de desarrollo de software se encuentran las siguientes (las prácticas pueden depender en base al proceso y la terminología asociada al mismo):

- Ingeniería de requisitos
- Análisis de sistemas
- Diseño/arquitectura a alto nivel
- Diseño a bajo nivel
- Codificación
- Integración
- Diseño y revisiones de código
- Pruebas
- Mantenimiento
- Gestión de proyectos
- Gestión de la configuración

La mayoría de las disciplinas reconocen algunas prácticas como mejores prácticas. Una mejor práctica es una práctica que, a través de la experiencia e investigación, se ha probado que lleva al resultado deseado fiablemente y se considera prudente y recomendable hacerla en una variedad de contextos.

Desafíos de la Ingeniería del software

Los desafíos clave con los que se enfrenta la ingeniería del software son:

- **Heterogeneidad**: desarrollando técnicas para construir software que puedan utilizar plataformas y entornos de ejecución heterogéneos.

- **Entrega**: desarrollando técnicas que lleven a una entrega de software más rápida.
- **Confianza**: desarrollando técnicas que demuestren que los usuarios pueden tener confianza en el software.

Capas de la Ingeniería del Software

El enfoque de ingeniería del software cuenta con un compromiso organizacional con la calidad porque no es posible incorporar la ingeniería del software en una organización que no está centrada en conseguir calidad.

La ingeniería del software es una tecnología multicapa. Se puede ver como un conjunto de componentes estratificados, que reposan sobre ese enfoque de calidad.

Figura 6. Capas de la ingeniería del software

Estos componentes que forman parte de la ingeniería del software son:

- **Procesos**: un marco de trabajo que ayuda al jefe de proyecto a controlar la gestión del proyecto y las actividades de ingeniería.
- **Métodos**: las actividades técnicas requeridas para la creación de productos de trabajo.
- **Herramientas**: la ayuda automatizada para los procesos y métodos.

Procesos

El fundamento de la ingeniería del software es la capa de proceso. El proceso define un marco de trabajo para un conjunto de áreas clave de proceso que se deben establecer para la entrega efectiva de la tecnología de la ingeniería del software.

La capa de proceso define el proceso que se usará para construir el software y las actividades y tareas que un jefe de proyecto tiene que gestionar. Por lo tanto, las áreas

claves del proceso forman la base del control de gestión de proyectos del software y establecen el contexto en el que se aplican los métodos técnicos, se obtienen productos de trabajo (modelos, documentos, datos, informes, formularios, etc.), se establecen hitos, se asegura la calidad y el cambio se gestiona adecuadamente. El proceso de la ingeniería del software es la unión que mantiene juntas las capas de tecnologías y que permite un desarrollo racional y oportuno de la ingeniería del software.

Se pueden ver todas las actividades, incluyendo las actividades técnicas, como parte del proceso. Además, cualquier recurso, incluyendo herramientas usadas para construir el software también encajan en el proceso. La capa de proceso es, por lo tanto, el fundamento de la ingeniería del software y da soporte a las capas de métodos y herramientas.

Todos los enfoques de la construcción de software tienen un proceso, pero en muchos casos, son ad hoc, invisibles y caóticos. Una buena ingeniería de software hace que el proceso de software sea más visible, predecible y más útil para aquellos que construyen software.

Métodos

La capa de métodos se centra en las actividades técnicas que se deben realizar para conseguir las tareas de ingeniería. Proporciona el “cómo” y cubre las actividades de ingeniería fundamentales.

Los métodos abarcan una gran gama de tareas que incluyen análisis de requisitos, diseño, construcción de programas, pruebas y mantenimiento. Los métodos de la ingeniería del software dependen de un conjunto de principios básicos que gobiernan cada una de las áreas de la tecnología e incluyen actividades de modelado y otras técnicas descriptivas.

La construcción de software implica una amplia colección de actividades técnicas. La capa de métodos contiene los métodos definidos para realizar esas actividades de forma eficiente. Se centra en cómo se han de realizar las actividades técnicas. Los personas involucradas usan los métodos para realizar las actividades de ingeniería fundamentales necesarias para construir el software.

Para varias actividades de proceso, la capa de métodos contiene el correspondiente conjunto de métodos técnicos para usar. Esto abarca un conjunto de reglas, los modos de representación gráficos o basados en texto, y las guías relacionadas para la evaluación de la calidad de la información representada.

Para definir la capa de métodos, es necesario seleccionar un método adecuado de un amplio rango de métodos disponibles.

Herramientas

La capa de herramientas proporciona soporte a las capas de proceso y métodos centrándose en el significado de la automatización de algunas de las actividades manuales. Las herramientas se pueden utilizar para automatizar las siguientes actividades:

- Actividades de gestión de proyectos.
- Métodos técnicos usados en la ingeniería del software.
- Soporte de sistemas general.
- Marcos de trabajo para otras herramientas.

La jefa de RRHH habla con un usuario de la herramienta.

Figura 7. Automatización de tareas

La automatización ayuda a eliminar el tedio del trabajo, reduce las posibilidades de errores, y hace más fácil usar buenas prácticas de ingeniería del software. Cuando se usan herramientas, la documentación se convierte en una parte integral del trabajo hecho, en vez de ser una actividad adicional. De ahí que la documentación no se tenga que realizar como

actividad adicional. Las herramientas se pueden utilizar para realizar actividades de gestión de proyecto así como para actividades técnicas.

Existen una gran variedad de herramientas para múltiples actividades. Entre ellas se pueden destacar las siguientes:

- Herramientas de gestión de proyectos.
- Herramientas de control de cambios.
- Herramientas de análisis y diseño.
- Herramientas de generación de código.
- Herramientas de pruebas.
- Herramientas de reingeniería.
- Herramientas de documentación.
- Herramientas de prototipos.

Estas herramientas soportan las capas de proceso y de métodos en varias actividades.

Ciclo de vida de desarrollo del software

El ciclo de vida es el conjunto de fases por las que pasa el sistema que se está desarrollando desde que nace la idea inicial hasta que el software es retirado o remplazado (muere). También se denomina a veces paradigma.

Entre las funciones que debe tener un ciclo de vida se pueden destacar:

- Determinar el orden de las fases del proceso de software.
- Establecer los criterios de transición para pasar de una fase a la siguiente.
- Definir las entradas y salidas de cada fase.
- Describir los estados por los que pasa el producto.
- Describir las actividades a realizar para transformar el producto.
- Definir un esquema que sirve como base para planificar, organizar, coordinar, desarrollar…

Un ciclo de vida para un proyecto se compone de fases sucesivas compuestas por tareas que se pueden planificar. Según el modelo de ciclo de vida, la sucesión de fases puede ampliarse con bucles de realimentación, de manera que lo que conceptualmente se considera una misma fase se pueda ejecutar más de una vez a lo largo de un proyecto, recibiendo en cada pasada de ejecución aportaciones a los resultados intermedios que se van produciendo (realimentación).

- Fases: una fase es un conjunto de actividades relacionadas con un objetivo en el desarrollo del proyecto. Se construye agrupando tareas (actividades elementales) que pueden compartir un tramo determinado del tiempo de vida de un proyecto. La agrupación temporal de tareas impone requisitos temporales correspondientes a la asignación de recursos (humanos, financieros o materiales).
- Entregables: son los productos intermedios que generan las fases. Pueden ser materiales o inmateriales (documentos, software). Los entregables permiten evaluar la marcha del proyecto mediante comprobaciones de su adecuación o no a los requisitos funcionales y de condiciones de realización previamente establecidos.

Las actividades genéricas del ciclo de vida del desarrollo del software son:

- Especificación: lo que el sistema debería hacer y sus restricciones de desarrollo.
- Desarrollo: producción del sistema software.
- Validación: comprobar que el sistema es lo que el cliente quiere.
- Evolución: cambiar el software en respuesta a las demandas de cambio.

Figura 8. Desarrollo a medida

Modelos de ciclo de vida del software

La ingeniería del software se vale de una serie de modelos que establecen y muestran las distintas etapas y estados por los que pasa un producto software, desde su concepción inicial, pasando por su desarrollo, puesta en marcha y posterior mantenimiento, hasta la retirada del producto. A estos modelos se les denomina “Modelos de ciclo de vida del software”. El primer modelo concebido fue el de Royce, más comúnmente conocido como “Cascada” o “Lineal Secuencial”. Este modelo establece que las diversas actividades que se van realizando al desarrollar un producto software, se suceden de forma lineal.

Los modelos de ciclo de vida del software describen las fases del ciclo de software y el orden en que se ejecutan las fases.

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 asociados 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.
- Provee un espacio de trabajo para la definición de un proceso detallado de desarrollo de software.

Las principales diferencias entre distintos modelos de ciclo de vida están en:

- El alcance del ciclo dependiendo de hasta dónde llegue el proyecto correspondiente. Un proyecto puede comprender un simple estudio de viabilidad del desarrollo de un producto, o su desarrollo completo o en el extremo, toda la historia del producto con su desarrollo, fabricación y modificaciones posteriores hasta su retirada del mercado.
- Las características (contenidos) de las fases en que dividen el ciclo. Esto puede depender del propio tema al que se refiere el proyecto, o de la organización.
- La estructura y la sucesión de las etapas, si hay realimentación entre ellas, y si tenemos libertad de repetirlas (iterar).

Modelo en cascada

Es un enfoque metodológico que ordena rigurosamente las etapas del ciclo de vida del software, de forma que el inicio de cada etapa debe esperar a la finalización de la inmediatamente anterior.

El modelo en cascada es un proceso de desarrollo secuencial, en el que el desarrollo se ve fluyendo hacia abajo (como una cascada) sobre las fases que componen el ciclo de vida.

La primera descripción formal del modelo en cascada se cree que fue en un artículo publicado en 1970 por Winston W. Royce, aunque Royce no usó el término cascada en este

artículo. Irónicamente, Royce estaba presentando este modelo como un ejemplo de modelo que no funcionaba, defectuoso.

En el modelo original de Royce, existían las siguientes fases:

1. Especificación de requisitos
2. Diseño
3. Construcción (Implementación o codificación)
4. Integración
5. Pruebas
6. Instalación
7. Mantenimiento

Para seguir el modelo en cascada, se avanza de una fase a la siguiente en una forma puramente secuencial.

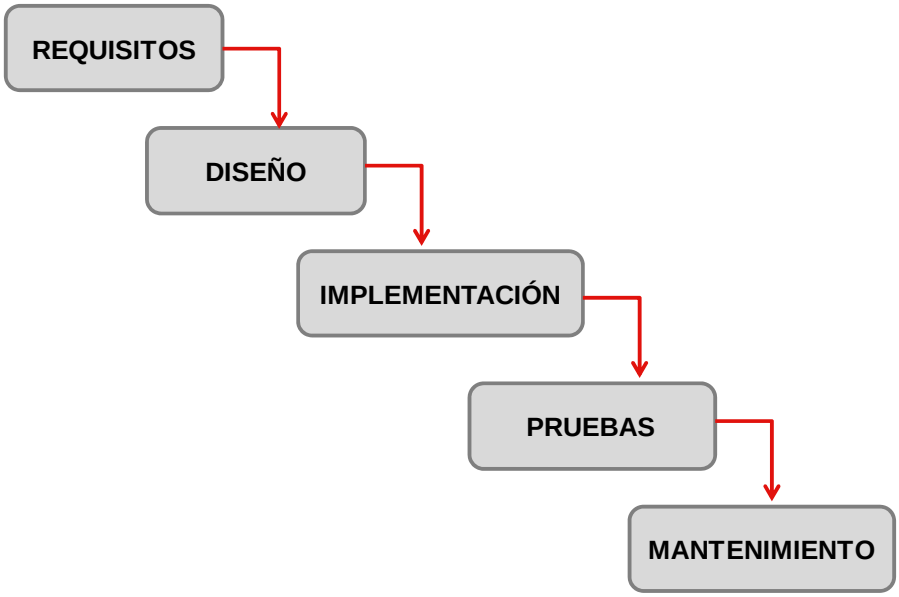

Figura 9. Modelo de ciclo de vida en cascada

Si bien ha sido ampliamente criticado desde el ámbito académico y la industria, sigue siendo el paradigma más seguido a día de hoy.

Modelo en V

El modelo en v se desarrolló para terminar con algunos de los problemas que se vieron utilizando el enfoque de cascada tradicional. Los defectos estaban siendo encontrados demasiado tarde en el ciclo de vida, ya que las pruebas no se introducían hasta el final del

proyecto. El modelo en v dice que las pruebas necesitan empezarse lo más pronto posible en el ciclo de vida. También muestra que las pruebas no son sólo una actividad basada en la ejecución. Hay una variedad de actividades que se han de realizar antes del fin de la fase de codificación. Estas actividades deberían ser llevadas a cabo en paralelo con las actividades de desarrollo, y los técnicos de pruebas necesitan trabajar con los desarrolladores y analistas de negocio de tal forma que puedan realizar estas actividades y tareas y producir una serie de entregables de pruebas.

El modelo en v es un proceso que representa la secuencia de pasos en el desarrollo del ciclo de vida de un proyecto. Describe las actividades y resultados que han de ser producidos durante el desarrollo del producto. La parte izquierda de la v representa la descomposición de los requisitos y la creación de las especificaciones del sistema. El lado derecho de la v representa la integración de partes y su verificación. V significa “Validación y Verificación”.

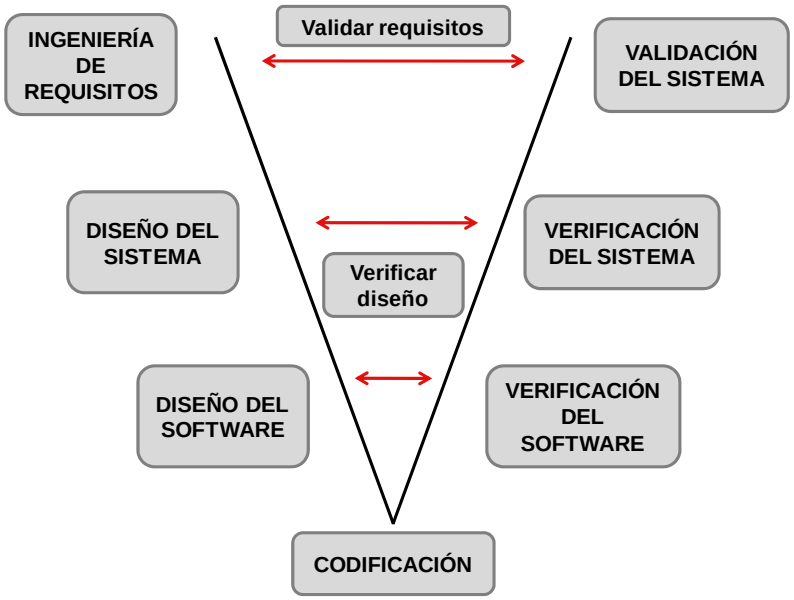

Figura 10. Modelo de ciclo de vida en V

Realmente las etapas individuales del proceso pueden ser casi las mismas que las del modelo en cascada. Sin embargo hay una gran diferencia. En vez de ir para abajo de una forma lineal las fases del proceso vuelven hacia arriba tras la fase de codificación, formando una v. La razón de esto es que para cada una de las fases de diseño se ha encontrado que hay un homólogo en las fases de pruebas que se correlacionan.

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.

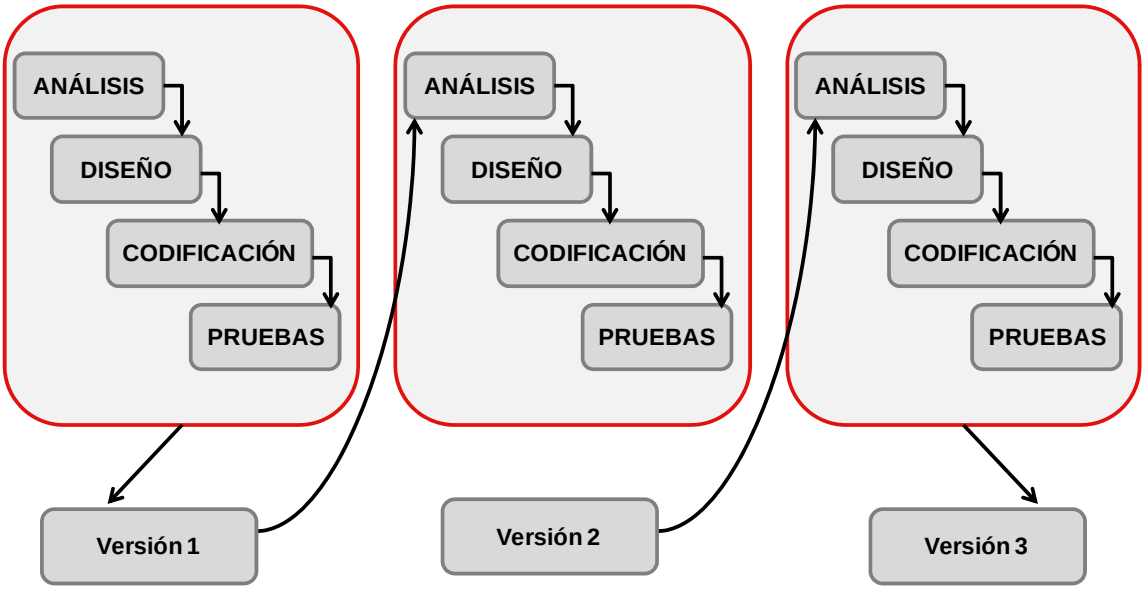

Figura 11. Modelo de ciclo de vida iterativo

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.

Modelo de desarrollo incremental

El modelo incremental combina elementos del modelo en cascada con la filosofía interactiva de construcción de prototipos. Se basa en la filosofía de construir incrementando las funcionalidades del programa. Este modelo aplica secuencias lineales de forma escalonada mientras progresa el tiempo en el calendario. Cada secuencia lineal produce un incremento del software.

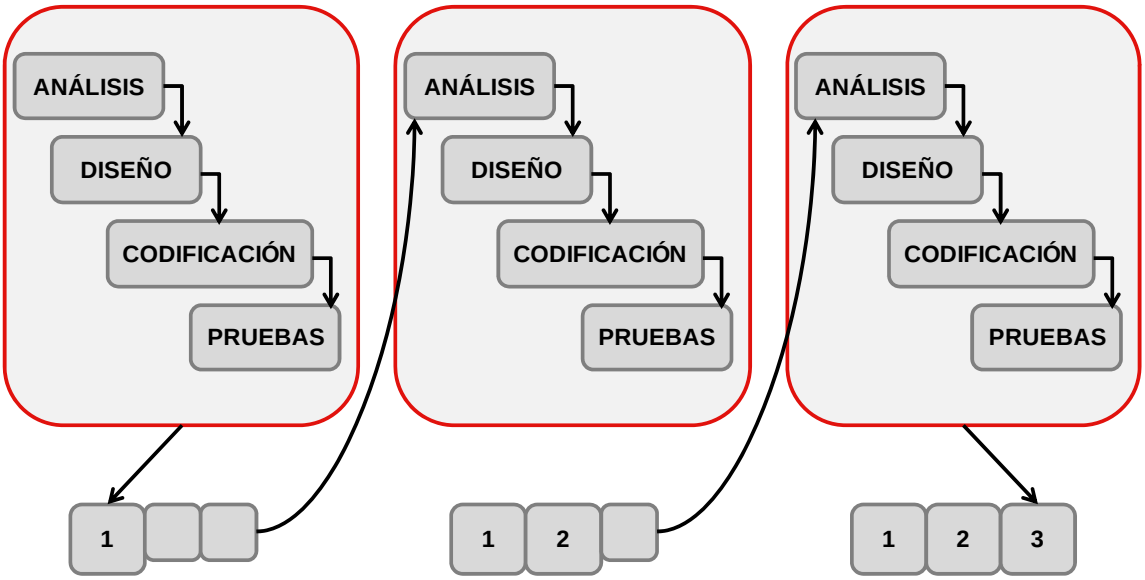

Figura 12. Modelo de ciclo de vida incremental

Cuando se utiliza un modelo incremental, el primer incremento es a menudo un producto esencial, sólo con los requisitos básicos. Este modelo se centra en la entrega de un producto operativo con cada incremento. Los primeros incrementos son versiones incompletas del producto final, pero proporcionan al usuario la funcionalidad que precisa y también una plataforma para la evaluación.

Modelo en espiral

El desarrollo en espiral es un modelo de ciclo de vida desarrollado por Barry Boehm en 1985, utilizado de forma generalizada en la ingeniería del software. Las actividades de este modelo se conforman en una espiral, cada bucle representa un conjunto de actividades. Las actividades no están fijadas a priori, sino que las siguientes se eligen en función del análisis de riesgos, comenzando por el bucle anterior.

Al ser un modelo de ciclo de vida orientado a la gestión de riesgos se dice que uno de los aspectos fundamentales de su éxito radica en que el equipo que lo aplique tenga la necesaria experiencia y habilidad para detectar y catalogar correctamente riesgos.

Tareas:

Para cada ciclo habrá cuatro actividades:

1. Determinar o fijar objetivos:

- Fijar también los productos definidos a obtener: requerimientos, especificación, manual de usuario.
- Fijar las restricciones.
- Identificar riesgos del proyecto y estrategias alternativas para evitarlos.
- Hay una cosa que solo se hace una vez: planificación inicial o previa

2. Análisis del riesgo:
    - Estudiar todos los riesgos potenciales y se seleccionan una o varias alternativas propuestas para reducir o eliminar los riesgos
3. Desarrollar, verificar y validar (probar):
    - Tareas de la actividad propia y de prueba.
    - Análisis de alternativas e identificación de resolución de riesgos.
    - Dependiendo del resultado de la evaluación de riesgos, se elige un modelo para el desarrollo, que puede ser cualquiera de los otros existentes, como formal, evolutivo, cascada, etc. Así, por ejemplo, si los riesgos de la interfaz de usuario son dominantes, un modelo de desarrollo apropiado podría ser la construcción de prototipos evolutivos.
4. Planificar:
    - Revisar todo lo que se ha llevado a cabo, evaluándolo y decidiendo si se continua con las fases siguientes y planificando la próxima actividad.

El proceso empieza en la posición central. Desde allí se mueve en el sentido de las agujas del reloj.

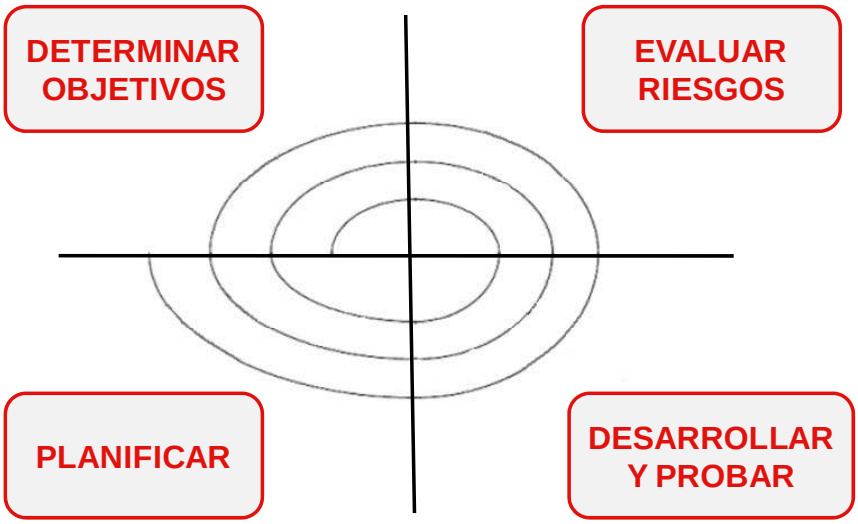

Figura 13. Ciclo de vida en espiral

Modelo de prototipos

El paradigma de construcción de prototipos comienza con la recolección de requisitos. El desarrollador y el cliente encuentran y definen los objetivos globales para el software, identifican los requisitos conocidos y las áreas del esquema en donde es obligatoria más definición. Entonces aparece un diseño rápido. El diseño rápido se centra en una representación de esos aspectos del software que serán visibles para el usuario/cliente. El diseño rápido lleva a la construcción de un prototipo. El prototipo lo evalúa el cliente/usuario y se utiliza para refinar los requisitos del software a desarrollar. La iteración ocurre cuando el prototipo se pone a punto para satisfacer las necesidades del cliente, permitiendo al mismo tiempo que el desarrollador comprenda mejor lo que se necesita hacer.

Figura 14. Modelo de ciclo de vida de prototipos

Comparativa de los modelos de ciclo de vida

A continuación se van a mostrar las ventajas e inconvenientes más significativas de cada uno de los modelos de ciclo de vida:

Tabla 1. Descripción de áreas de proceso de gestión de procesos

	VENTAJAS	INCONVENIENTES
Modelo en cascada	• Modelo en el que está todo bien organizado. • No se mezclan las fases. • Simple y fácil de llevar a la práctica. • Fácil de gestionar.	• Rara vez los proyectos siguen una secuencia lineal. • Difícil establecer todos los requisitos al principio. • Visibilidad del producto cuando está terminado.
Modelo en V	• Simple y fácil de llevar a la práctica. • En cada una de las fases hay entregables específicos.	• Tiene poca flexibilidad y ajustar el alcance es difícil y caro. • El modelo no proporciona caminos claros para

	• Desarrollo de planes de prueba en etapas tempranas del ciclo de vida. • Suele funcionar bien para proyectos pequeños donde los requisitos son entendidos fácilmente.	problemas encontrados durante las fases de pruebas.
Modelo incremental	• Se genera software operativo de forma rápida y en etapas tempranas del ciclo de vida del software. • Modelo más flexible, por lo que se reduce el coste en cambios de alcance y requisitos. • Es más fácil probar y depurar en una iteración más pequeña. • Es más fácil gestionar riesgos. • Cada iteración es un hito gestionado fácilmente.	• Se requiere mucha experiencia para definir los incrementos y distribuir en ellos las tareas de forma proporcionada. • Cada fase de una iteración es rígida y no se superpone con otras. • Todos los requisitos han de definirse al inicio.
Modelo iterativo	• No hace falta que los requisitos estén totalmente definidos desde el principio.	• Que los requisitos no estén definidos desde el principio también puede verse como un inconveniente ya que

	• Desarrollo en pequeños ciclos. • Es más fácil gestionar riesgos. • Cada iteración es un hito gestionado fácilmente.	pueden surgir problemas con la arquitectura.
Modelo de prototipos	• Visibilidad del producto desde el inicio del ciclo de vida con el primer prototipo • Permite introducir cambios en las iteraciones siguientes del ciclo. • Permite la realimentación continua del cliente.	• Puede ser un desarrollo lento.
Modelo en espiral	• Reduce riesgos del proyecto. • Incorpora objetivos de calidad. • Integra el desarrollo con el mantenimiento. • No es rígido ni estático. • Se produce software en etapas tempranas del ciclo de vida.	• Modelo que genera mucho trabajo adicional. • Exige un alto nivel de experiencia y cierta habilidad en los analistas de riesgos. • Modelo costoso.

ISO/IEC 12207

Esta norma establece un marco de referencia común para los procesos del ciclo de vida del software, con una terminología bien definida a la que puede hacer referencia la industria del software. Contiene procesos, actividades y tareas para aplicar durante la adquisición de un sistema que contiene software, un producto software puro o un servicio software, y durante el suministro, desarrollo, operación y mantenimiento de productos software. El software incluye la parte software del firmware.

Esta norma incluye también un proceso que puede emplearse para definir, controlar y mejorar los procesos del ciclo de vida del software.

La ISO 12207 define un **modelo de ciclo de vida** como un marco de referencia que contiene los procesos, actividades y tareas involucradas en el desarrollo, operación y mantenimiento de un producto software, y que abarca toda la vida del sistema, desde la definición de sus requisitos hasta el final del uso.

Esta norma agrupa las actividades que pueden llevarse a cabo durante el ciclo de vida del software en cinco procesos principales, ocho procesos de apoyo y cuatro procesos organizativos. Cada proceso del ciclo de vida está dividido en un conjunto de actividades; cada actividad se subdivide a su vez en un conjunto de tareas.

- Procesos principales del ciclo de vida: son cinco procesos que dan servicio a las partes principales durante el ciclo de vida del software. Una parte principal es la que inicia o lleva a cabo el desarrollo, operación y mantenimiento de productos software. Los procesos principales son:
    - Proceso de adquisición
    - Proceso de suministro
    - Proceso de desarrollo
    - Proceso de operación
    - Proceso de mantenimiento
- Procesos de apoyo al ciclo de vida: son procesos que apoyan a otros procesos como parte esencial de los mismos, con un propósito bien definido, y contribuyen al éxito y

calidad del proyecto software. Un proceso de apoyo se emplea y ejecuta por otro proceso según sus necesidades.

Los procesos de apoyo son:

- Proceso de documentación
- Proceso de gestión de la configuración
- Proceso de verificación
- Proceso de validación
- Proceso de revisiones conjuntas
- Proceso de auditoría
- Proceso de solución de problemas

- Procesos organizativos del ciclo de vida: se emplean por una organización para establecer e implementar una infraestructura construida por procesos y personal asociado al ciclo de vida, y para mejorar continuamente esta estructura y procesos.
  - Proceso de gestión
  - Proceso de infraestructura
  - Proceso de mejora
  - Proceso de formación

Metodologías de desarrollo de software

Un objetivo de décadas ha sido encontrar procesos y metodologías que sean sistemáticas, predecibles y repetibles, a fin de mejorar la productividad en el desarrollo y la calidad del producto software. Existen numerosas propuestas metodológicas que inciden en distintas dimensiones del proceso de desarrollo. Por una parte tenemos aquellas propuestas más **tradicionales** que se centran especialmente en el control del proceso, estableciendo rigurosamente las actividades involucradas, los artefactos que se deben producir, y las herramientas y notaciones que se usarán. Estas propuestas han demostrado ser efectivas y necesarias en un gran número de proyectos, pero también han presentado problemas en muchos otros. Una posible mejora es incluir en los procesos de desarrollo más actividades, más artefactos y más restricciones, basándose en los puntos débiles detectados. Sin embargo, el resultado final sería un proceso de desarrollo más complejo que puede incluso limitar la propia habilidad del equipo para llevar a cabo el proyecto. Otra aproximación es centrarse en otras dimensiones, como por ejemplo el factor humano o el producto software. Esta es la filosofía de las metodologías **ágiles**, las cuales dan mayor valor al individuo, a la colaboración con el cliente y al desarrollo incremental del software con iteraciones muy cortas. Este enfoque está mostrando su efectividad en proyectos con requisitos muy cambiantes y cuando se exige reducir drásticamente los tiempos de desarrollo pero manteniendo una alta calidad. Las metodologías ágiles están revolucionando la manera de producir software, y a la vez generando un amplio debate entre sus seguidores y quienes por escepticismo o convencimiento no las ven como alternativa para las metodologías tradicionales.

La evolución de la disciplina de ingeniería del software ha traído consigo propuestas diferentes para mejorar los resultados del proceso de construcción. Las metodologías tradicionales haciendo énfasis en la planificación y las metodologías ágiles haciendo énfasis en la adaptabilidad del proceso, delinean las principales propuestas presentes.

Definición de metodología

Una metodología es un conjunto integrado de técnicas y métodos que permite abordar de forma homogénea y abierta cada una de las actividades del ciclo de vida de un proyecto de desarrollo. Es un proceso de software detallado y completo.

Las metodologías se basan en una combinación de los modelos de proceso genéricos (cascada, incremental…). Definen artefactos, roles y actividades, junto con prácticas y técnicas recomendadas.

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

Una definición estándar de metodología puede ser el conjunto de métodos que se utilizan en una determinada actividad con el fin de formalizarla y optimizarla. Determina los pasos a seguir y cómo realizarlos para finalizar una tarea.

Si esto se aplica a la ingeniería del software, podemos destacar que una metodología:

- Optimiza el proceso y el producto software.
- Proporciona métodos que guían en la planificación y en el desarrollo del software.
- Define qué hacer, cómo y cuándo durante todo el desarrollo y mantenimiento de un proyecto.

Una metodología define una estrategia global para enfrentarse con el proyecto. Entre los elementos que forman parte de una metodología se pueden destacar:

- Fases: tareas a realizar en cada fase.
- Productos: E/S de cada fase, documentos.
- Procedimientos y herramientas: apoyo a la realización de cada tarea.
- Criterios de evaluación del proceso y del producto: permiten determinar si se han logrado los objetivos.

El marco de trabajo de una metodología de desarrollo de software consiste en:

- Una filosofía de desarrollo de software, con el enfoque o enfoques del proceso de desarrollo de software.
- Múltiples herramientas, modelos y métodos para ayudar en el proceso de desarrollo de software.

Estos marcos de trabajo están con frecuencia vinculados a algunos tipos de organizaciones, que se encargan del desarrollo, soporte de uso y promoción de la metodología. La metodología con frecuencia se documenta de alguna manera formal.

Ventajas del uso de una metodología

Son muchas las ventajas que puede aportar el uso de una metodología. A continuación se van a exponer algunas de ellas, clasificadas desde distintos puntos de vista.

Desde el punto de vista de gestión:

- Facilitar la tarea de planificación
- Facilitar la tarea del control y seguimiento de un proyecto
- Mejorar la relación coste/beneficio
- Optimizar el uso de los recursos disponibles
- Facilitar la evaluación de resultados y el cumplimiento de los objetivos
- Facilitar la comunicación efectiva entre usuarios y desarrolladores

Desde el punto de vista de los ingenieros del software:

- Ayudar a la comprensión del problema
- Optimizar el conjunto y cada una de las fases del proceso de desarrollo
- Facilitar el mantenimiento del producto final
- Permitir la reutilización de partes del producto

Desde el punto de vista del cliente o usuario:

- Garantizar un determinado nivel de calidad en el producto final
- Ofrecer confianza en los plazos de tiempo fijados en la definición del proyecto
- Definir el ciclo de vida que más se adecue a las condiciones y características del desarrollo

Metodologías tradicionales y ágiles

Desarrollar un buen software depende de un gran número de actividades y etapas, donde el impacto de elegir la metodología para un equipo en un determinado proyecto es trascendental para el éxito del producto.

Según la filosofía de desarrollo se pueden clasificar las metodologías en dos grupos. Las metodologías tradicionales, que se basan en una fuerte planificación durante todo el desarrollo, y las metodologías ágiles, en las que el desarrollo de software es incremental, cooperativo, sencillo y adaptado.

Metodologías tradicionales

Las metodologías tradicionales son denominadas, a veces, de forma peyorativa, como metodologías pesadas.

Centran su atención en llevar una documentación exhaustiva de todo el proyecto y en cumplir con un plan de proyecto, definido todo esto, en la fase inicial del desarrollo del proyecto.

Otra de las características importantes dentro de este enfoque, son los altos costes al implementar un cambio y la falta de flexibilidad en proyectos donde el entorno es volátil.

Las metodologías tradicionales (formales) se focalizan en la documentación, planificación y procesos (plantillas, técnicas de administración, revisiones, etc.)

Metodologías ágiles

Este enfoque nace como respuesta a los problemas que puedan ocasionar las metodologías tradicionales y se basa en dos aspectos fundamentales, retrasar las decisiones y la planificación adaptativa. Basan su fundamento en la adaptabilidad de los procesos de desarrollo.

Estas metodologías ponen de relevancia que la capacidad de respuesta a un cambio es más importante que el seguimiento estricto de un plan.

¿Metodologías ágiles o metodologías tradicionales?

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.

Tabla 2. Tabla comparativa entre metodologías tradicionales y desarrollo ágil

Metodologías ágiles	Metodologías tradicionales
Basadas en heurísticas provenientes de prácticas de producción de código	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo
Especialmente preparados para cambios durante el proyecto	Cierta resistencia a los cambios
Impuestas internamente (por el equipo)	Impuestas externamente
Proceso menos controlado, con pocos principios	Proceso mucho más controlado, con numerosas políticas/normas
No existe contrato tradicional o al menos	Existe un contrato prefijado

es bastante flexible	
El cliente es parte del equipo de desarrollo	El cliente interactúa con el equipo de desarrollo mediante reuniones
Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio	Grupos grandes y posiblemente distribuidos
Pocos artefactos	Más artefactos
Pocos roles	Más roles
Menos énfasis en la arquitectura del software	La arquitectura del software es esencial y se expresa mediante modelos

Escenario de clausura

La solución que se ha adoptado es la de que el proveedor desarrolle una aplicación a medida para la empresa. De esta forma la empresa podrá demandar los requisitos que satisfagan sus necesidades.

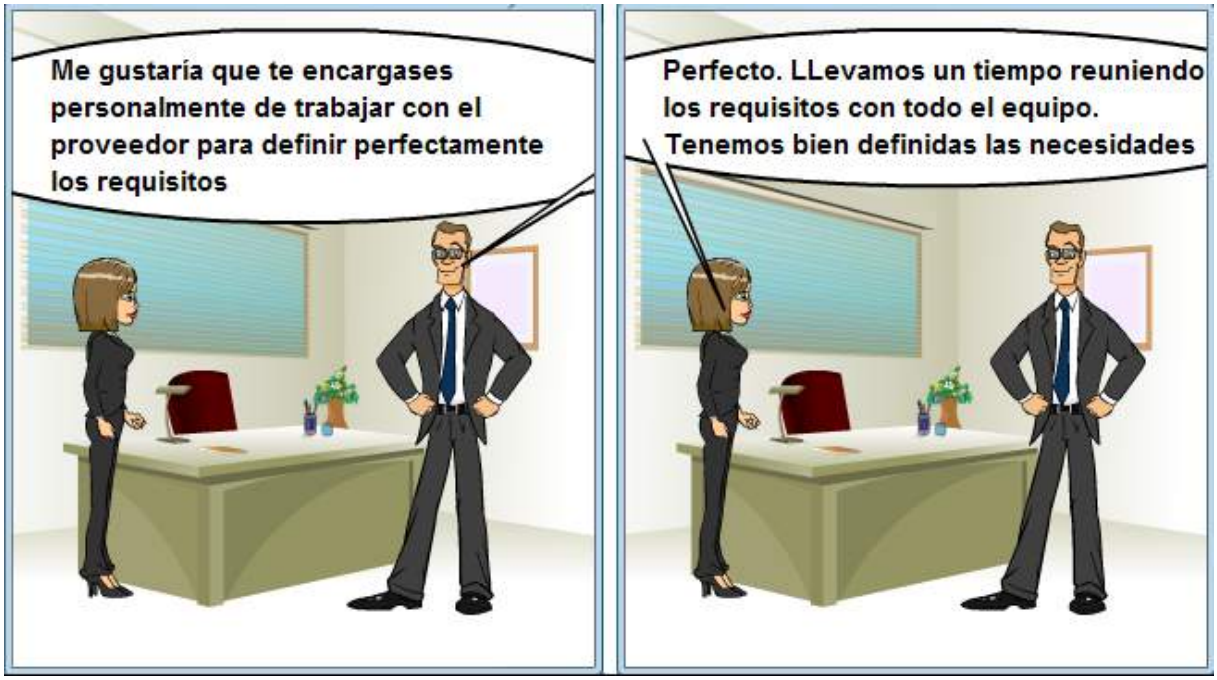

Figura 15. Escenario de clausura I

Figura 16. Escenario de clausura II

En unos meses el proveedor les suministra la herramienta que ha desarrollado para ellos.

Figura 17. Escenario de clausura III

Figura 18. Escenario de clausura IV

Enlaces

Agile Spain www.agile-spain.com

Alianza ágil www.agilealliance.org

IEEE www.ieee.org/portal/site

Manifiesto ágil www.agilemanifesto.org

Sitio web de la Organización Internacional para la Estandarización www.iso.org

Swebok www.swebok.org

Glosario

- **Ciclo de vida:** conjunto de fases por las que pasa el sistema que se está desarrollando desde que nace la idea inicial hasta que el software es retirado o remplazado.
- **Ingeniería del software:** disciplina de ingeniería preocupada por todos los aspectos de la producción de software desde las primeras etapas de especificación del sistema hasta el mantenimiento del sistema después de que éste se haya puesto en uso. Es la aplicación de un enfoque sistemático, disciplinado y cuantificable para el desarrollo, operación y mantenimiento del software, que es la aplicación de la ingeniería del software.
- **Metodología**: conjunto integrado de técnicas y métodos que permite abordar de forma homogénea y abierta cada una de las actividades del ciclo de vida de un proyecto de desarrollo.
- **Sistema experto**: programa o conjunto de programas informáticos concebidos para resolver problemas o situaciones de un modo similar al que utilizaría una persona experta.
- **Software**: son los programas y la documentación asociada tal como requisitos, modelos de diseño y manuales de usuario.