EL software

¿Qué es? El software de computadora es el producto que construyen los programadores profesionales y al que después le dan mantenimiento durante un largo tiempo. Incluye programas que se ejecutan en una computadora de cualquier tamaño y arquitectura, contenido que se presenta a medida que se ejecutan los programas de cómputo e información descriptiva tanto en una copia dura como en formatos virtuales que engloban virtualmente a cualesquiera medios electrónicos. La ingeniería de software está formada por un proceso, un conjunto de métodos (prácticas) y un arreglo de herramientas que permite a los profesionales elaborar software de cómputo de alta calidad.

¿Quién lo hace? Los ingenieros de software elaboran y dan mantenimiento al software, y virtualmente cada persona lo emplea en el mundo industrializado, ya sea en forma directa o indirecta.

¿Por qué es importante? El software es importante por- que afecta a casi todos los aspectos de nuestras vidas y ha invadido nuestro comercio, cultura y actividades cotidianas. La ingeniería de software es importante porque nos permite construir sistemas complejos en un tiempo razonable y con alta calidad.

¿Cuáles son los pasos? El software de computadora se construye del mismo modo que cualquier producto exitoso, con la aplicación de un proceso ágil y adaptable para obtener un resultado de mucha calidad, que satisfaga las necesidades de las personas que usarán el producto. En estos pasos se aplica el enfoque de la ingeniería de software.

¿Cuál es el producto final? Desde el punto de vista de un ingeniero de software, el producto final es el conjunto de programas, contenido (datos) y otros productos terminados que constituyen el software de computadora. Pero desde la perspectiva del usuario, el producto final es la información resultante que de algún modo hace mejor al mundo en el que vive.

¿Cómo me aseguro de que lo hice bien? Seleccione aquellas ideas que sean aplicables al software que usted hace y aplíquelas a su trabajo.

El software de computadora sigue siendo la tecnología más importante en la escena mundial. Y también es un ejemplo magnífico de la ley de las consecuencias inesperadas. Hace 50 años, nadie hubiera podido predecir que el software se convertiría en una tecnología indispensable para los negocios, ciencias e ingeniería, ni que permitiría la creación de tecnologías nuevas (por ejemplo, ingeniería genética y nanotecnología), la ampliación de tecnologías ya existentes (telecomunicaciones) y el cambio radical de tecnologías antiguas (la industria de la impresión); tampoco que el software sería la fuerza que impulsaría la revolución de las computadoras personales, que productos de software empacados se comprarían en los supermercados, que el software evolucionaría poco a poco de un producto a un servicio cuando compañías de software "sobre pedido" proporcionaran funcionalidad justo a tiempo a través de un navegador web, que una compañía de software sería más grande y tendría más influencia que casi todas las empresas de la era industrial, que una vasta red llamada Internet sería operada con software y evolucionaría y cambiaría todo, desde la investigación en bibliotecas y la compra de productos para el consumidor hasta el discurso político y los hábitos de encuentro de los adultos jóvenes (y no tan jóvenes).

LA NATURALEZA DEL SOFTWARE

En la actualidad, el software tiene un papel dual. Es un producto y al mismo tiempo es el vehículo para entregar un producto. En su forma de producto, brinda el potencial de cómputo incorporado en el hardware de cómputo o, con más amplitud, en una red de computadoras a las que se accede por medio de un hardware local. Ya sea que resida en un teléfono móvil u opere en el interior de una computadora central, el software es un transformador de información —produce, administra, adquiere, modifica, despliega o transmite información que puede ser tan simple como un solo bit o tan compleja como una presentación con multimedios generada a partir de datos obtenidos de decenas de fuentes independientes—. Como vehículo utilizado para distribuir el producto, el software actúa como la base para el control de la computadora (sistemas operativos), para la comunicación de información (redes) y para la creación y control de otros programas (herramientas y ambientes de software).

El software distribuye el producto más importante de nuestro tiempo: *información*. Transforma los datos personales (por ejemplo, las transacciones financieras de un individuo) de modo que puedan ser más útiles en un contexto local, administra la información de negocios para mejorar la competitividad, provee una vía para las redes mundiales de información (la internet) y brinda los medios para obtener información en todas sus formas.

En el último medio siglo, el papel del software de cómputo ha sufrido un cambio significativo. Las notables mejoras en el funcionamiento del hardware, los profundos cambios en las arquitecturas de computadora, el gran incremento en la memoria y capacidad de almacenamiento, y una amplia variedad de opciones de entradas y salidas exóticas han propiciado la existencia de sistemas basados en computadora más sofisticados y complejos. Cuando un sistema tiene éxito, la sofisticación y complejidad producen resultados deslumbrantes, pero también plantean problemas enormes para aquellos que deben construir sistemas complejos.

En la actualidad, la enorme industria del software se ha convertido en un factor dominante en las economías del mundo industrializado. Equipos de especialistas de software, cada uno centrado en una parte de la tecnología que se requiere para llegar a una aplicación compleja, han reemplazado al programador solitario de los primeros tiempos. A pesar de ello, las preguntas que se hacía aquel programador son las mismas que surgen cuando se construyen sistemas modernos basados en computadora:

- ¿Por qué se requiere tanto tiempo para terminar el software?
- ¿Por qué son tan altos los costos de desarrollo?
- ¿Por qué no podemos detectar todos los errores antes de entregar el software a nuestros clientes?
- ¿Por qué dedicamos tanto tiempo y esfuerzo a mantener los programas existentes?
- ¿Por qué seguimos con dificultades para medir el avance mientras se desarrolla y mantiene el software?

En la actualidad, la mayoría de los profesionales y muchos usuarios tienen la fuerte sensación de que entienden el software. Pero ¿es así?

La descripción que daría un libro de texto sobre software sería más o menos así:

El software es: 1) instrucciones (programas de cómputo) que cuando se ejecutan proporcionan las características, función y desempeño buscados; 2) estructuras de datos que permiten que los programas manipulen en forma adecuada la información, y 3) información descriptiva tanto en papel como en formas virtuales que describen la operación y uso de los programas.

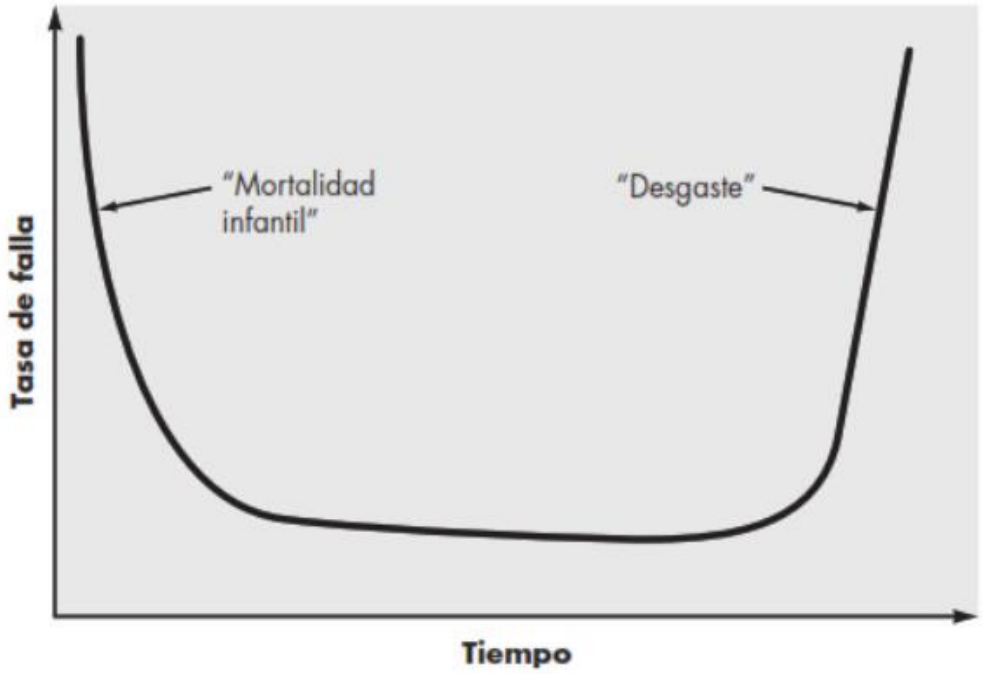
Pero es probable que una definición más formal no mejore de manera apreciable nuestra comprensión. Para asimilar lo anterior, es importante examinar las características del software que lo hacen diferente de otros objetos que construyen los seres humanos. El software es elemento de un sistema lógico y no de uno físico. Por tanto, tiene características que difieren considerablemente de las del hardware:

1. El software se desarrolla o modifica con intelecto; no se manufactura en el sentido clásico.

Aunque hay algunas similitudes entre el desarrollo de software y la fabricación de hardware, las dos actividades son diferentes en lo fundamental. En ambas, la alta calidad se logra a través de un buen diseño, pero la fase de manufactura del hardware introduce problemas de calidad que no existen (o que se corrigen con facilidad) en el software. Ambas actividades dependen de personas, pero la relación entre los individuos dedicados y el trabajo logrado es diferente por completo. Las dos actividades requieren la construcción de un “producto”, pero los enfoques son distintos. Los costos del software se concentran en la ingeniería. Esto significa que los proyectos de software no pueden administrarse como si fueran proyectos de manufactura.

2. El software no se “desgasta”.

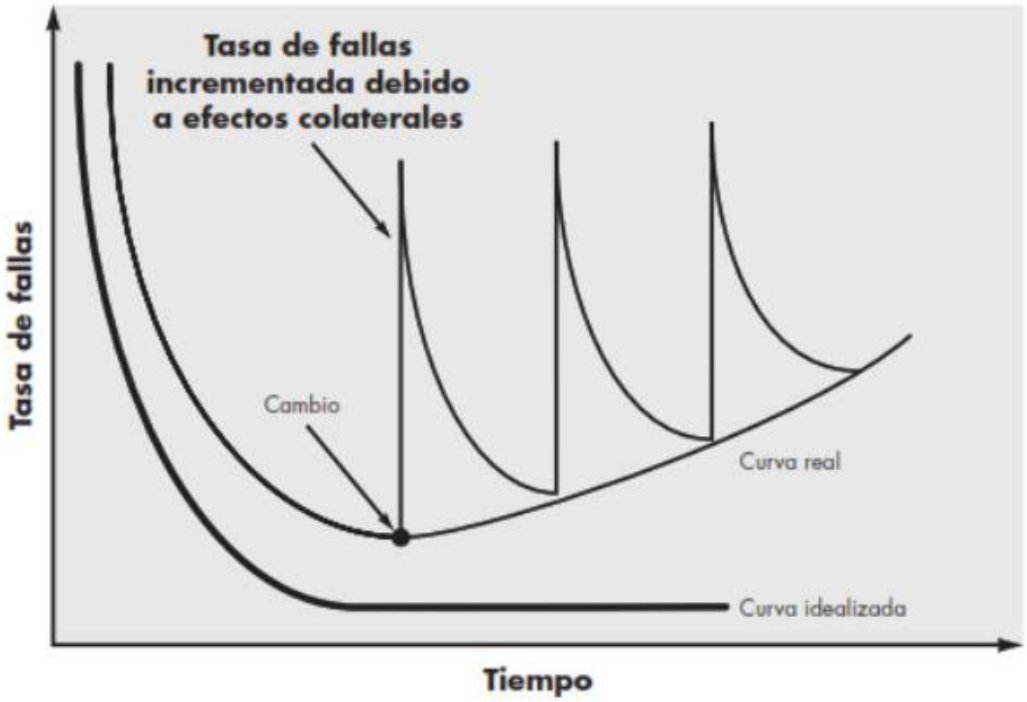
La figura 1 ilustra la tasa de falla del hardware como función del tiempo. La relación, que es frecuente llamar “curva de tina”, indica que el hardware presenta una tasa de fallas relativamente elevada en una etapa temprana de su vida (fallas que con frecuencia son atribuibles a defectos de diseño o manufactura); los defectos se corrigen y la tasa de fallas se abate a un nivel estable (muy bajo, por fortuna) durante cierto tiempo. No obstante, conforme pasa el tiempo, la tasa de fallas aumenta de nuevo a medida que los componentes del hardware resienten los efectos acumulativos de suciedad, vibración, abuso, temperaturas extremas y muchos otros inconvenientes ambientales. En pocas palabras, el hardware comienza a desgastarse.



Curva de fallas del hardware

El software no es susceptible a los problemas ambientales que hacen que el hardware se desgaste. Por tanto, en teoría, la curva de la tasa de fallas adopta la forma de la "curva idealizada" que se aprecia en la figura 2. Los defectos ocultos ocasionarán tasas elevadas de fallas al comienzo de la vida de un programa. Sin embargo, éstas se corrigen y la curva se aplana, como se indica. La curva idealizada es una gran simplificación de los modelos reales de las fallas del software. Aun así, la implicación está clara: el software no se desgasta, *¡pero sí se deteriora!*

Esta contradicción aparente se entiende mejor si se considera la curva real en la figura 2. Durante su vida, el software sufrirá cambios. Es probable que cuando éstos se realicen, se introduzcan errores que ocasionen que la curva de tasa de fallas tenga aumentos súbitos, como se ilustra en la "curva real" (véase la figura 2). Antes de que la curva vuelva a su tasa de fallas original de estado estable, surge la solicitud de otro cambio que hace que la curva se dispare otra vez. Poco a poco, el nivel mínimo de la tasa de fallas comienza a aumentar: el software se está deteriorando como consecuencia del cambio.



Curvas de falla del software

3. Aunque la industria se mueve hacia la construcción basada en componentes, la mayor parte del software se construye para un uso individualizado.

A medida que evoluciona una disciplina de ingeniería, se crea un conjunto de componentes estandarizados para el diseño. Los tornillos estándar y los circuitos integrados preconstruidos son sólo dos de los miles de componentes estándar que utilizan los ingenieros mecánicos y eléctricos conforme diseñan nuevos sistemas. Los componentes reutilizables han sido creados para que el ingeniero pueda concentrarse en los elementos verdaderamente innovadores de un diseño; es decir, en las partes de éste que representan algo nuevo. En el mundo del hardware, volver a usar componentes es una parte natural del proceso de ingeniería. En el del software, es algo que apenas ha empezado a hacerse a gran escala.

Un componente de software debe diseñarse e implementarse de modo que pueda volverse a usar en muchos programas diferentes. Los modernos componentes reutilizables incorporan tanto los datos como el procesamiento que se les aplica, lo que permite que el ingeniero de software cree nuevas aplicaciones a partir de partes susceptibles de volverse a usar. Por ejemplo, las actuales interfaces interactivas de usuario se construyen con componentes reutilizables que permiten la creación de ventanas gráficas, menús desplegables y una amplia variedad de mecanismos de interacción. Las estructuras de datos y el detalle de procesamiento que se requieren para construir la interfaz están contenidos en una librería de componentes reusables para tal fin.

Etapas del proceso de desarrollo de software

1. El proceso del software

Un ***proceso*** es un conjunto de actividades, acciones y tareas que se ejecutan cuando va a crearse algún producto del trabajo.

Una ***actividad*** busca lograr un objetivo amplio (por ejemplo, comunicación con los participantes) y se desarrolla sin importar el dominio de la aplicación, tamaño del proyecto, complejidad del esfuerzo o grado de rigor con el que se usará la ingeniería de software.

Una ***acción*** (diseño de la arquitectura) es un conjunto de tareas que producen un producto importante del trabajo (por ejemplo, un modelo del diseño de la arquitectura).

Una ***tarea*** se centra en un objetivo pequeño, pero bien definido (por ejemplo, realizar una prueba unitaria) que produce un resultado tangible.

En el contexto de la **ingeniería de software**, un proceso no es una prescripción rígida de cómo elaborar software de cómputo. Por el contrario, es un enfoque adaptable que permite que las personas que hacen el trabajo (el equipo de software) busquen y elijan el conjunto apropiado de acciones y tareas para el trabajo. Se busca siempre entregar el software en forma oportuna y con calidad suficiente para satisfacer a quienes patrocinaron su creación y a aquellos que lo usarán.

La *estructura del proceso* establece el fundamento para el proceso completo de la ingeniería de software por medio de la identificación de un número pequeño de actividades estructurales que sean aplicables a todos los proyectos de software, sin importar su tamaño o complejidad. Además, la estructura del proceso incluye un conjunto de actividades sombrilla que son aplicables a través de todo el proceso del software. Una estructura de proceso general para la ingeniería de software consta de cinco actividades:

Comunicación. Antes de que comience cualquier trabajo técnico, tiene importancia crítica comunicarse y colaborar con el cliente (y con otros participantes). Se busca entender los objetivos de los participantes respecto del proyecto, y reunir los requerimientos que ayuden a definir las características y funciones del software.

Planificación. Cualquier viaje complicado se simplifica si existe un mapa. Un proyecto de software es un viaje difícil, y la actividad de planeación crea un "mapa" que guía al equipo mientras viaja. El mapa —llamado plan del proyecto de software— define el trabajo de ingeniería de software al describir las tareas técnicas por realizar, los riesgos probables, los recursos que se requieren, los productos del trabajo que se obtendrán y una programación de las actividades.

Modelado. Ya sea usted diseñador de paisaje, constructor de puentes, ingeniero aeronáutico, carpintero o arquitecto, a diario trabaja con modelos. Crea un "bosquejo" del objeto por hacer a fin de entender el panorama general —cómo se verá arquitectónicamente, cómo ajustan entre sí las partes constituyentes y muchas características más—. Si se requiere, refina el bosquejo con más y más detalles en un esfuerzo por comprender mejor el problema y cómo resolverlo. Un ingeniero de software hace lo mismo al crear modelos a fin de entender mejor los requerimientos del software y el diseño que los satisfará.

Construcción. Esta actividad combina la generación de código (ya sea manual o automatizada) y las pruebas que se requieren para descubrir errores en éste.

Despliegue. El software (como entidad completa o como un incremento parcialmente terminado) se entrega al consumidor que lo evalúa y que le da retroalimentación, misma que se basa en dicha evaluación.

Estas cinco actividades estructurales genéricas se usan durante el desarrollo de programas pequeños y sencillos, en la creación de aplicaciones web grandes y en la ingeniería de sistemas enormes y complejos basados en computadoras. Los detalles del proceso de software serán distintos en cada caso, pero las actividades estructurales son las mismas.

Para muchos proyectos de software, las actividades estructurales se aplican en forma iterativa a medida que avanza el proyecto. Es decir, la comunicación, la planeación, el modelado, la construcción y el despliegue se ejecutan a través de cierto número de repeticiones del proyecto. Cada iteración produce un incremento del software que da a los participantes un subconjunto de características y funcionalidad generales del software.

Conforme se produce cada incremento, el software se hace más y más completo.

2. Actividades complementarias

Las actividades estructurales del proceso de ingeniería de software son complementadas por cierto número de *actividades sombrilla*. En general, las actividades sombrilla se aplican a lo largo de un proyecto de software y ayudan al equipo que lo lleva a cabo a administrar y controlar el avance, la calidad, el cambio y el riesgo. Es común que las actividades sombrilla sean las siguientes:

Seguimiento y control del proyecto de software: permite que el equipo de software evalúe el progreso comparándolo con el plan del proyecto y tome cualquier acción necesaria para apegarse a la programación de actividades.

Administración del riesgo: evalúa los riesgos que puedan afectar el resultado del proyecto o la calidad del producto.

Aseguramiento de la calidad del software: define y ejecuta las actividades requeridas para garantizar la calidad del software.

Revisiones técnicas: evalúa los productos del trabajo de la ingeniería de software a fin de descubrir y eliminar errores antes de que se propaguen a la siguiente actividad.

Medición: define y reúne mediciones del proceso, proyecto y producto para ayudar al equipo a entregar el software que satisfaga las necesidades de los participantes; puede usarse junto con todas las demás actividades estructurales y sombrilla.

Administración de la configuración del software: administra los efectos del cambio a lo largo del proceso del software.

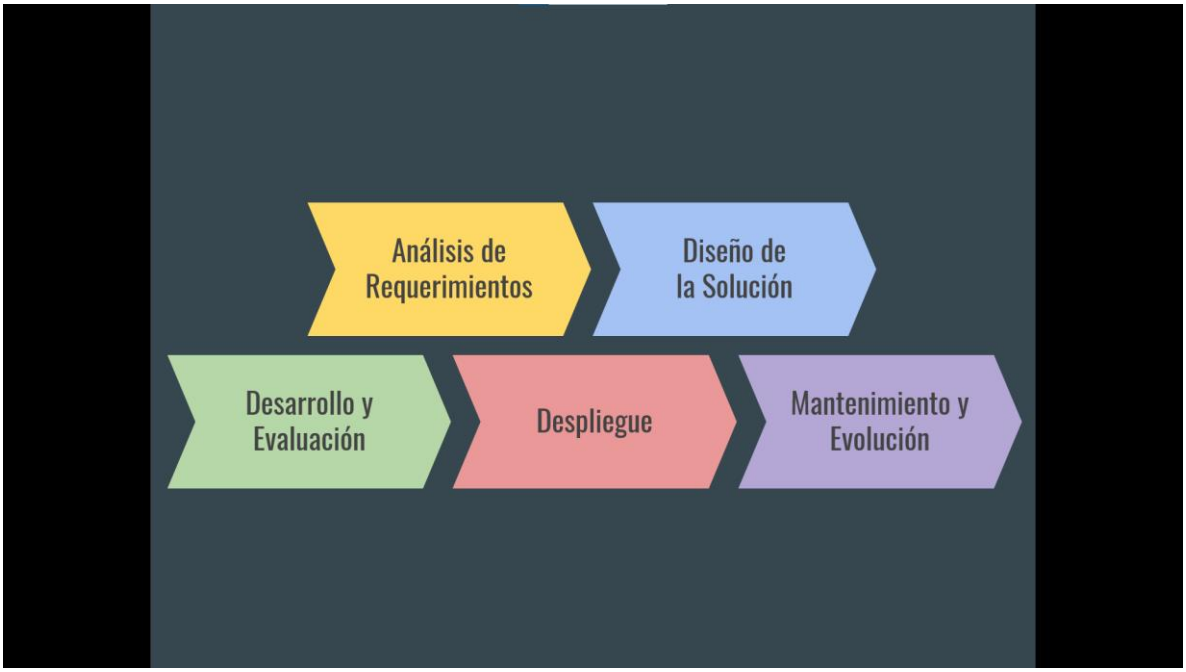
Administración de la reutilización: define criterios para volver a usar el producto del trabajo (incluso los componentes del software) y establece mecanismos para obtener componentes reutilizables.

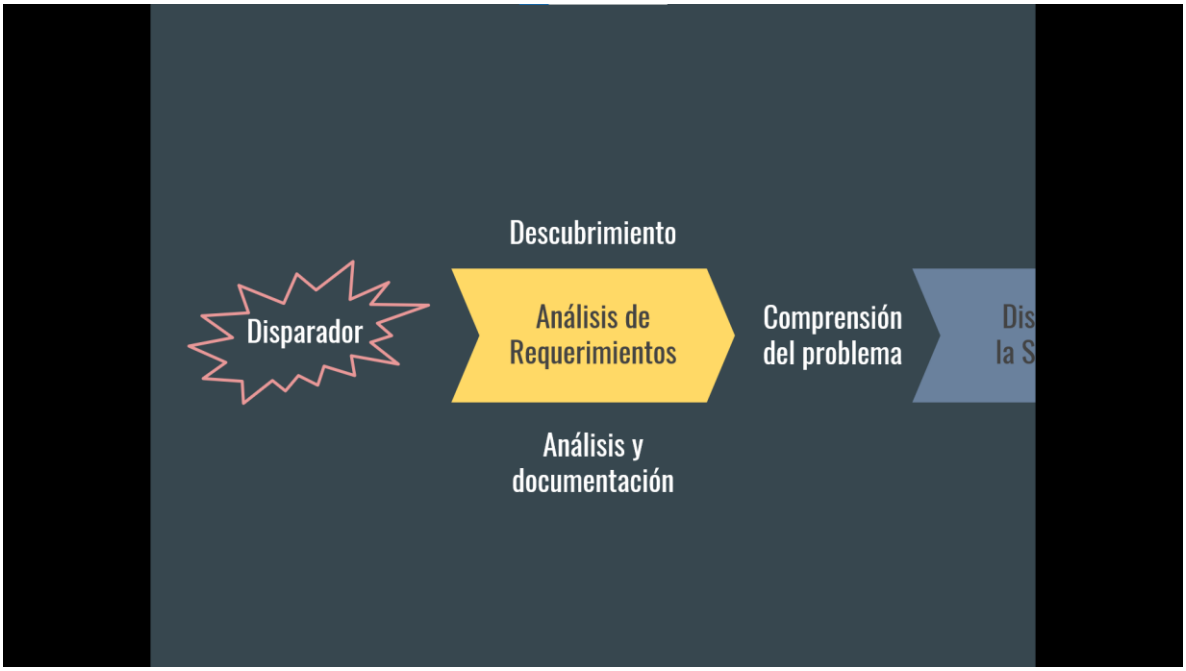
Preparación y producción del producto del trabajo: agrupa las actividades requeridas para crear productos del trabajo, tales como modelos, documentos, registros, formatos y listas.

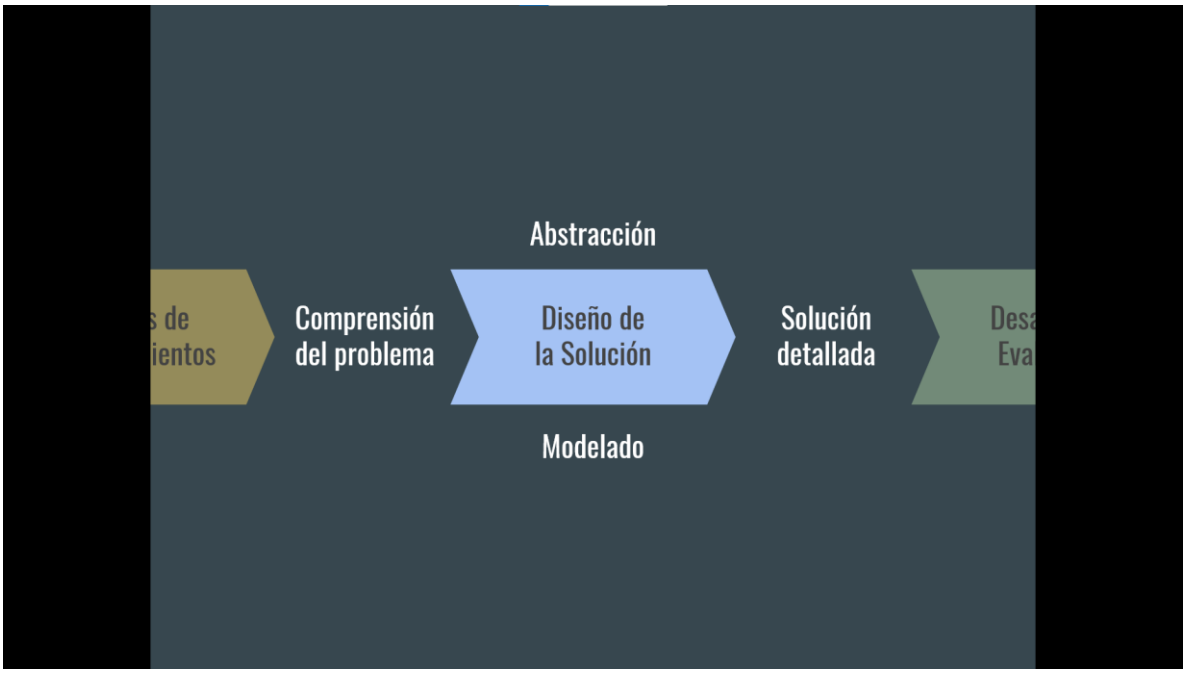
Ya se dijo en esta sección que el proceso de ingeniería de software no es una prescripción rígida que deba seguir en forma dogmática el equipo que lo crea. Al contrario, debe ser ágil y adaptable (al problema, al proyecto, al equipo y a la cultura organizacional). Por tanto, un proceso adoptado para un proyecto puede ser significativamente distinto de otro adoptado para otro proyecto. Entre las diferencias se encuentran las siguientes:

- Flujo general de las actividades, acciones y tareas, así como de las interdependencias entre ellas.
- Grado en el que las acciones y tareas están definidas dentro de cada actividad estructural
- Grado en el que se identifican y requieren los productos del trabajo
- Forma en la que se aplican las actividades de aseguramiento de la calidad
- Manera en la que se realizan las actividades de seguimiento y control del proyecto
- Grado general de detalle y rigor con el que se describe el proceso
- Grado con el que el cliente y otros participantes se involucran con el proyecto
- Nivel de autonomía que se da al equipo de software
- Grado con el que son prescritos la organización y los roles del equipo

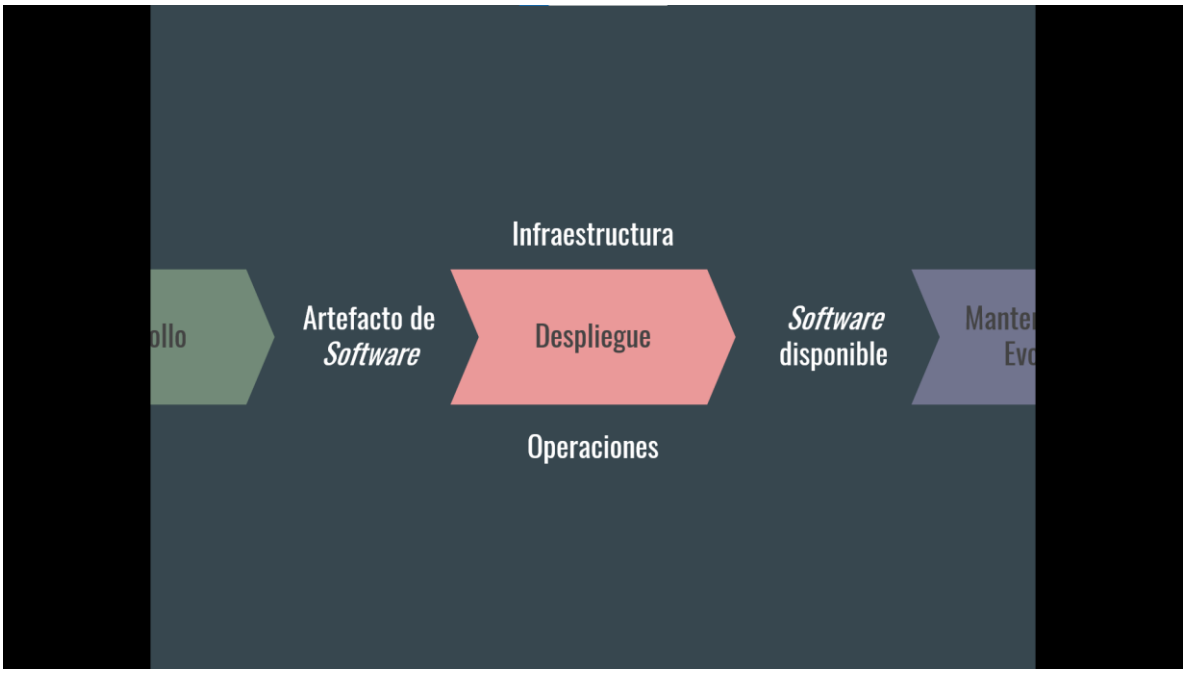
3. Material complementario.

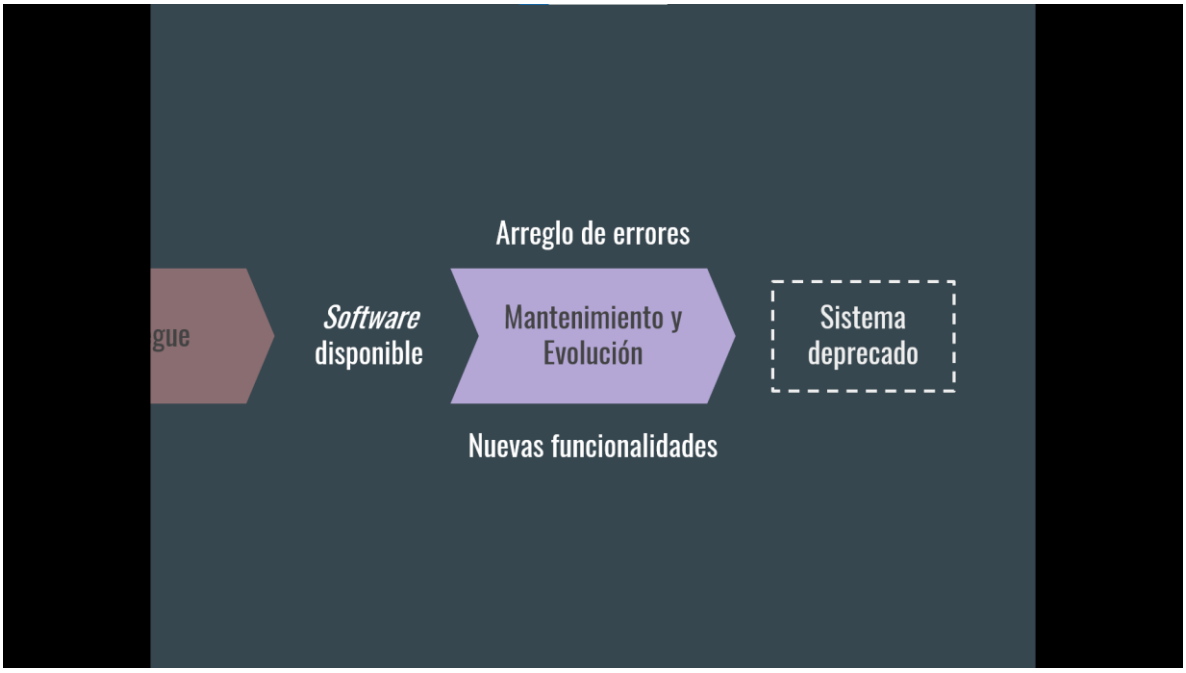





Abstracción
s de
ientos
Comprensión
del problema
Diseño de
la Solución
Solución
detallada
Desa
Eva
Modelado


Programación e
integración
de
ción
Solución
detallada
Desarrollo y
Evaluación
Artefacto de
Software
Des
Automatización de
pruebas


Infraestructura
ollo
Artefacto de
Software
Despliegue
Software
disponible
Manter
Evo
Operaciones


Arreglo de errores
gue
Software
disponible
Mantenimiento y
Evolución
Sistema
deprecado
Nuevas funcionalidades

Ejercicio 1 - Análisis de requerimientos

Descripción

Venta libros a través de Internet

Los clientes acceden a la información sobre los libros a través de la Web y realizan búsquedas por autor, título o ISBN. A medida que navegan por las distintas páginas, y encuentran algún libro que les interesa, lo incluyen en el carrito de la compra para efectuar al final el pedido correspondiente. Para realizar un pedido, un cliente debe estar previamente registrado como tal. Esto significa introducir una serie de datos personales (nombre y apellidos, dirección, localidad, código postal, país), datos de la tarjeta de crédito (tipo de tarjeta, número, fecha límite de validez) y sobre preferencias de envío (correo normal, expreso, internacional). Asociado a un pedido específico pueden introducirse opciones de empaquetado (estándar o regalo), tarjeta con mensaje adicional cuando es un regalo, o un nombre y dirección de otra persona a la que se le hace enviar un pedido.

Como es habitual en este tipo de aplicaciones, debería elegir un nombre de usuario y una clave como método de autentificación para efectuar las transacciones habituales con la librería. Cuando se han incluido en el carrito de la compra el conjunto de los libros deseados (cantidad, título y autor), se debe pasar al proceso de confirmar el pedido que debería requerir un paso previo de seguridad para garantizar que el cliente es quien dice ser. Una vez introducidos todos los datos adicionales, el cliente confirma el pedido que pasa a un estado de espera -90 minutos- durante el cual es posible modificar algunos de los ítems del pedido (eliminar o cambiar cantidad) pero no añadir nuevos ítems, para lo cual se debería crear un nuevo pedido. Una vez transcurridos los 90 minutos, los pedidos quedan confirmados definitivamente y no se pueden modificar ni anular. La empresa puede realizar envíos parciales en función de la disponibilidad de los ítems, pero sin modificar el costo total de envío debido a este fraccionamiento del pedido. A medida que se van rearmando los pedidos se envía un e-mail al cliente para confirmarle el pedido, lo mismo que al realizar el envío correspondiente.

Requerimientos

Esenciales

- El sistema deberá permitir registrar usuarios,
- El sistema deberá permitir autentificación (log in) de clientes,
- El sistema deberá permitir realizar ABM de libros,
- El sistema deberá permitir realizar búsquedas de libros por autor, título o ISBN,
- El sistema deberá permitir agregar un libro(ítem) al carrito de compra de un cliente,
- El sistema deberá permitir confirmar una compra por parte de un cliente,
- El sistema deberá permitir modificar un pedido ya confirmado, pero aún no enviado por parte de un cliente,
- El sistema deberá permitir realizar envíos parciales de pedidos,

Importantes

- El sistema podrá permitir distintas opciones de empaquetado para un pedido,

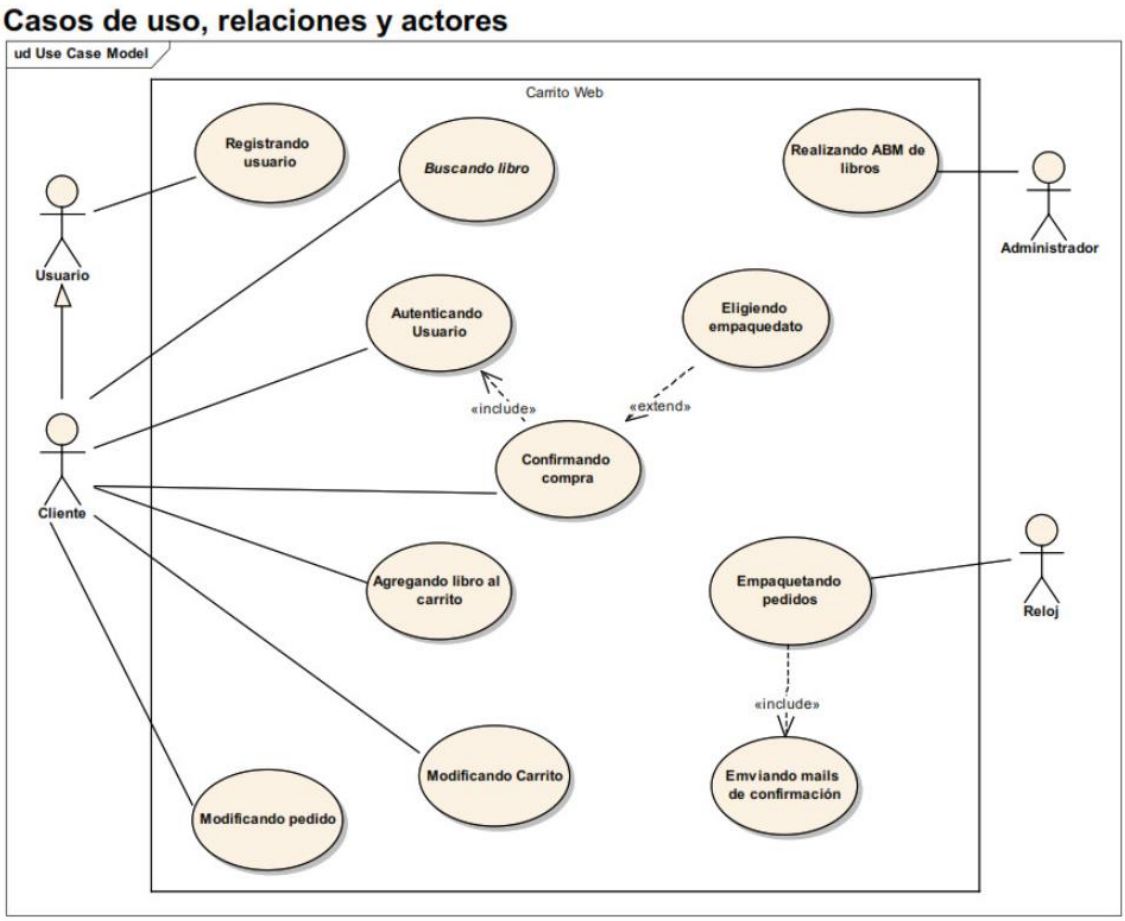
Deseables

- Es deseable que el sistema pueda enviar notificaciones vía mail de confirmación de pedidos a los clientes

Glosario

- *Usuario*: persona no logueada en el sistema.
- *Libro / ítem*: unidad de negocio de la empresa.
- *Carrito*: repositorio volátil de libros del cliente. En el momento que el usuario es autenticado en el sistema, dispone de un carrito vacío.
- *Cliente*: usuario registrado y autenticado en el sistema
- *Compra*: Confirmación de un carrito de compra.
- *Pedido*: compras no realizadas por clientes, aún no entregas.

Caso de uso, relación y actores



Modelos del Proceso

Introducción

Baetjer, Jr. [Bae98] comenta acerca del proceso del software.

Debido a que el software, como todo capital, es conocimiento incorporado y a que el conocimiento originalmente se halla disperso, tácito, latente e incompleto en gran medida, el desarrollo de software es un proceso de aprendizaje social. El proceso es un diálogo en el que el conocimiento que debe convertirse en software se reúne e incorpora en éste. El proceso genera interacción entre usuarios y diseñadores, entre usuarios y herramientas cambiantes, y entre diseñadores y herramientas en evolución [tecnología]. Es un proceso que se repite y en el que la herramienta que evoluciona sirve por sí misma como medio para la comunicación: con cada nueva ronda del diálogo se genera más conocimiento útil a partir de las personas involucradas.

En realidad, la elaboración de software de computadora es un proceso reiterativo de aprendizaje social, y el resultado, algo que Baetjer llamaría "capital de software", es la reunión de conocimiento recabado, depurado y organizado a medida que se realiza el proceso.

Pero desde el punto de vista técnico, **¿qué es exactamente un proceso del software?** En el contexto de este curso, se define proceso del software como **una estructura para las actividades, acciones y tareas que se requieren a fin de construir software de alta calidad.** ¿"Proceso" es sinónimo de "ingeniería de software"? La respuesta es "sí y no". Un proceso del software define el enfoque adoptado mientras se hace ingeniería sobre el software. Pero **la ingeniería de software también incluye tecnologías que enriquecen el proceso: métodos técnicos y herramientas automatizadas.**

Más importante aún, la ingeniería de software es llevada a cabo por personas creativas y preparadas que deben adaptar un proceso maduro de software a fin de que resulte apropiado para los productos que construyen y para las demandas de su mercado.

¿Qué es? Cuando se trabaja en la construcción de un producto o sistema, es importante ejecutar una serie de pasos predecibles —el mapa de carreteras que lo ayuda a obtener a tiempo un resultado de alta calidad—. El mapa que se sigue se llama "proceso del software".

¿Quién lo hace? Los ingenieros de software y sus gerentes adaptan el proceso a sus necesidades y luego lo siguen. Además, las personas que solicitaron el software tienen un papel en el proceso de definición, elaboración y prueba.

¿Por qué es importante? Porque da estabilidad, control y organización a una actividad que puede volverse caótica si se descontrola. Sin embargo, un enfoque moderno de ingeniería de software debe ser "ágil". Debe incluir sólo aquellas actividades, controles y productos del trabajo que sean apropiados para el equipo del proyecto y para el producto que se busca obtener.

¿Cuáles son los pasos? En un nivel detallado, el proceso que se adopte depende del software que se esté elaborando. Un proceso puede ser apropiado para crear software destinado a un sistema de control electrónico de un aeroplano, mientras que para la creación de un sitio web será necesario un proceso completamente distinto.

¿Cuál es el producto final? Desde el punto de vista de un ingeniero de software, los productos del trabajo son los programas, documentos y datos que se producen como consecuencia de las actividades y tareas definidas por el proceso.

¿Cómo me aseguro de que lo hice bien? Hay cierto número de mecanismos de evaluación del proceso del software que permiten que las organizaciones determinen la "madurez" de su proceso. Sin embargo, la calidad, oportunidad y viabilidad a largo plazo del producto que se elabora son los mejores indicadores de la eficacia del proceso que se utiliza.

UN MODELO GENERAL DE PROCESO

Ya se definió un proceso como la colección de actividades de trabajo, acciones y tareas que se realizan cuando va a crearse algún producto terminado. Cada una de las actividades, acciones y tareas se encuentra dentro de una estructura o modelo que define su relación tanto con el proceso como entre sí.

En la figura 2 se representa el proceso del software de manera esquemática. En dicha figura, cada actividad estructural está formada por un conjunto de acciones de ingeniería de software y cada una de éstas se encuentra definida por un conjunto de tareas que identifica las tareas del trabajo que deben realizarse, los productos del trabajo que se producirán, los puntos de aseguramiento de la calidad que se requieren y los puntos de referencia que se utilizarán para evaluar el avance.

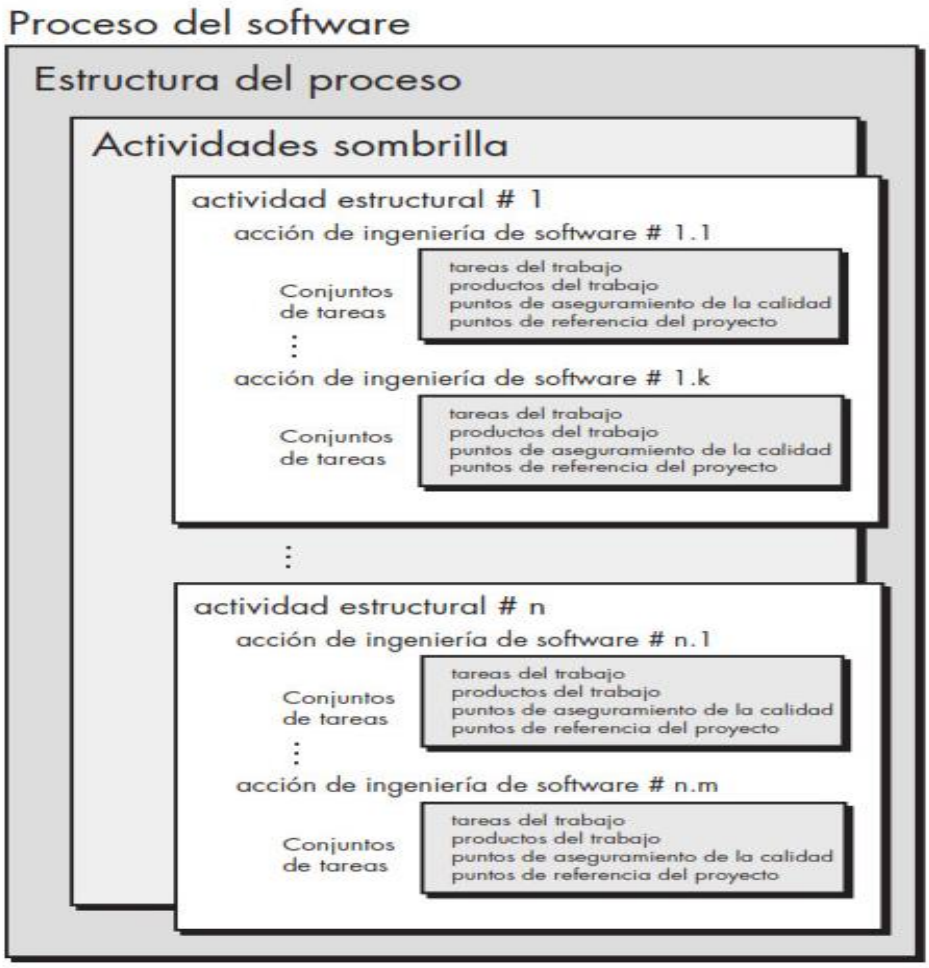
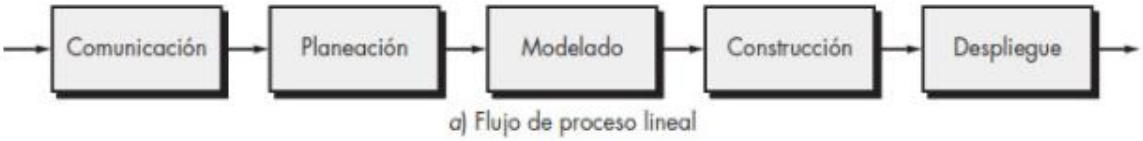


Figura 2. Estructura de un proceso del software

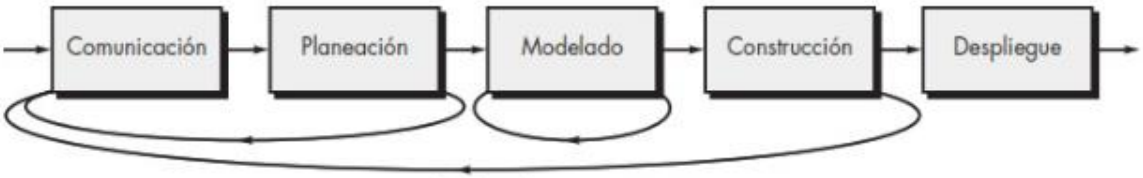
Como se dijo en la introducción, una estructura general para la ingeniería de software **define cinco actividades estructurales: comunicación, planeación, modelado, construcción y despliegue**. Además, a lo largo de todo el proceso se aplica un conjunto de actividades sombrilla: seguimiento y control del proyecto, administración de riesgos, aseguramiento de la calidad, administración de la configuración, revisiones técnicas, entre otras.

Aún no se menciona un aspecto importante del proceso del software —**llamado flujo del proceso**— y se describe la manera en que están organizadas las actividades estructurales, las acciones y tareas que ocurren dentro de cada una con respecto de la secuencia y el tiempo.

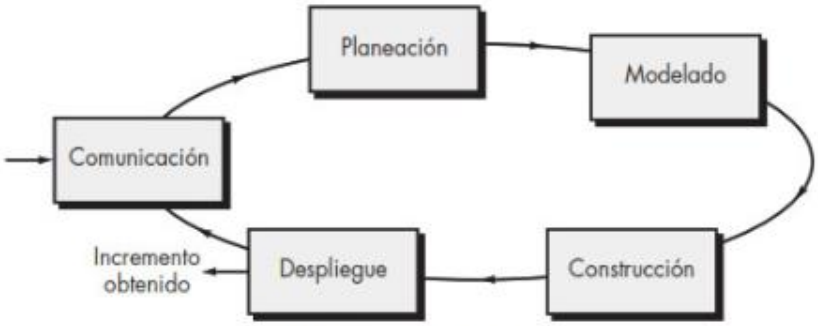
Un *flujo de proceso lineal* ejecuta cada una de las cinco actividades estructurales en secuencia, comenzando por la comunicación y terminando con el despliegue . Un *flujo de proceso iterativo* repite una o más de las actividades antes de pasar a la siguiente. Un *flujo de proceso evolutivo* realiza las actividades en forma "circular". A través de las cinco actividades, cada circuito lleva a una versión más completa del software. Un *flujo de proceso paralelo* ejecuta una o más actividades en paralelo con otras (por ejemplo, el modelado de un aspecto del software tal vez se ejecute en paralelo con la construcción de otro aspecto del software).



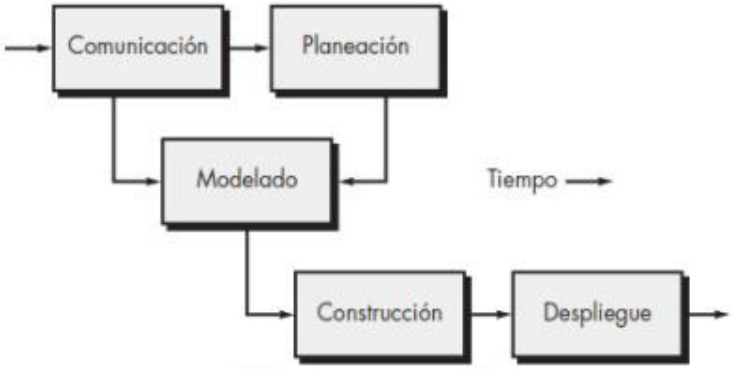
a) Flujo de proceso lineal



b) Flujo de proceso iterativo



c) Flujo de proceso evolutivo



d) Flujo de proceso paralelo

Definición de actividad estructural

Aunque ya se describieron las cinco actividades estructurales y se dio una definición básica de cada una, un equipo de software necesitará mucha más información antes de poder ejecutar de manera apropiada cualquiera de ellas como parte del proceso del software. Por tanto, surge una pregunta clave: *¿Qué acciones son apropiadas para una actividad estructural, dados la naturaleza del problema por resolver, las características de las personas que hacen el trabajo y los participantes que patrocinan el proyecto?*

Para un proyecto de software pequeño solicitado por una persona (en una ubicación remota) con requerimientos sencillos y directos, la actividad de comunicación tal vez no incluya algo más que una llamada telefónica con el participante apropiado. Entonces, la única acción necesaria es una conversación telefónica, y las tareas del trabajo (el conjunto de tareas) que engloba son las siguientes:

1. Hacer contacto con el participante por vía telefónica.
2. Analizar los requerimientos y tomar notas.
3. Organizar las notas por escrito en una formulación breve de los requerimientos.
4. Enviar correo electrónico al participante para que revise y apruebe.

Si el proyecto fuera considerablemente más complejo, con muchos participantes y cada uno con un distinto conjunto de requerimientos (a veces en conflicto), la actividad de comunicación puede tener seis acciones distintas: *concepción, indagación, elaboración, negociación, especificación y validación*. Cada una de estas acciones de la ingeniería del software tendrá muchas tareas de trabajo y un número grande de diferentes productos finales.

Punto Clave: Diferentes proyectos demandan diferentes conjuntos de tareas. El equipo de software elige el conjunto de tareas con base en las características del problema y el proyecto.

Identificación de un conjunto de tareas

Cada acción de la ingeniería de software (por ejemplo, obtención, asociada a la actividad de comunicación) se representa por cierto número de distintos conjuntos de tareas, cada uno de los cuales es una colección de tareas de trabajo de la ingeniería de software, relacionadas con productos del trabajo, puntos de aseguramiento de la calidad y puntos de referencia del proyecto. Debe escogerse el conjunto de tareas que se adapte mejor a las necesidades del proyecto y a las características del equipo. Esto implica que una acción de la ingeniería de software puede adaptarse a las necesidades específicas del proyecto de software y a las características del equipo del proyecto.

Patrones del proceso

Cada equipo de software se enfrenta a problemas conforme avanza en el proceso del software. Si se demostrara que existen soluciones fáciles para dichos problemas, sería útil para el equipo abordarlos y resolverlos rápidamente. Un patrón del proceso describe un problema relacionado con el proceso que se encuentra durante el trabajo de ingeniería de software, identifica el ambiente en el que surge el problema y sugiere una o más soluciones para el mismo. Dicho de manera general, un **patrón de proceso da un formato: un método consistente para describir soluciones del problema en el contexto del proceso del software**. Al combinar patrones, un equipo de software resuelve problemas y construye el proceso que mejor satisfaga las necesidades de un proyecto.

Los patrones se definen en cualquier nivel de abstracción. En ciertos casos, un patrón puede usarse para describir un problema (y su solución) asociado con un modelo completo del proceso (por ejemplo, hacer prototipos). En otras situaciones, los patrones se utilizan para describir un problema (y su solución) asociado con una actividad estructural (por ejemplo, **planeación**) o una acción dentro de una actividad estructural (estimación de proyectos).

Scoot Ambler ha propuesto un formato para describir un patrón del proceso:

Nombre del patrón. El patrón recibe un nombre significativo que lo describe en el contexto del proceso del software (por ejemplo, **Revisiones Técnicas**).

Tipo. Se especifica el tipo de patrón. Ambler sugiere tres tipos:

1. *Patrón de etapa*: define un problema asociado con una actividad estructural para el proceso. Como una actividad estructural incluye múltiples acciones y tareas del trabajo, un patrón de la etapa incorpora múltiples patrones de la tarea que son relevantes para la etapa (actividad estructural). Un ejemplo de patrón de etapa sería **Establecer Comunicación**. Este patrón incorporaría el patrón de tarea Recabar Requerimientos y otros más.
2. *Patrón de tarea*: define un problema asociado con una acción o tarea de trabajo de la ingeniería de software y que es relevante para el éxito de la práctica de ingeniería de software (por ejemplo, Recabar Requerimientos es un patrón de tarea).
3. *Patrón de fase*: define la secuencia de las actividades estructurales que ocurren dentro del proceso, aun cuando el flujo general de las actividades sea de naturaleza iterativa. Un ejemplo de patrón de fase es **Modelo Espiral** o **Prototipos**.

Contexto inicial. Describe las condiciones en las que se aplica el patrón. Antes de iniciar el patrón: 1) ¿Qué actividades organizacionales o relacionadas con el equipo han ocurrido? 2) ¿Cuál es el estado de entrada para el proceso? 3) ¿Qué información de ingeniería de software o del proyecto ya existe?

Problema. El problema específico que debe resolver el patrón.

Solución. Describe cómo implementar con éxito el patrón. Esta sección describe la forma en la que se modifica el estado inicial del proceso (que existe antes de implementar el patrón) como consecuencia de la iniciación del patrón. También describe cómo se transforma la información sobre la ingeniería de software o sobre el proyecto, disponible antes de que inicie el patrón, como consecuencia de la ejecución exitosa del patrón.

Contexto resultante. Describe las condiciones que resultarán una vez que se haya implementado con éxito el patrón: 1) ¿Qué actividades organizacionales o relacionadas con el equipo deben haber ocurrido? 2) ¿Cuál es el estado de salida del proceso? 3) ¿Qué información sobre la ingeniería de software o sobre el proyecto se ha desarrollado?

Los patrones de proceso dan un mecanismo efectivo para enfrentar problemas asociados con cualquier proceso del software. Los patrones permiten desarrollar una descripción jerárquica del proceso, que comienza en un nivel alto de abstracción (un patrón de fase). Después se mejora la descripción como un conjunto de patrones de etapa que describe las actividades estructurales y se mejora aún más en forma jerárquica en patrones de tarea más detallados para cada patrón de etapa. Una vez desarrollados los patrones de proceso, pueden reutilizarse para la definición de variantes del proceso, es decir, un equipo de software puede definir un modelo de proceso específico con el empleo de los patrones como bloques constituyentes del modelo del proceso.

Metodologías de desarrollo de software

Introducción

Las metodologías de desarrollo de software son un marco de trabajo eficiente que surgió en la década de los años 70 ya que ofrecían una respuesta a los problemas que surgían con los antiguos métodos de desarrollo, los cuales se enfocaban en la creación de software sin el control apropiado de las actividades del grupo de trabajo, lo que provocaba un producto lleno de deficiencias y problemas resultando en la insatisfacción del cliente, pues se le ofrecía un software que no cumplía con sus necesidades.

Las metodologías han logrado mejorar de manera significativa el producto de software por medio de fases o procesos efectivos que promueven la calidad; en el desarrollo de sistemas informáticos se hace imperativo una administración, planificación, seguimiento, control del grupo de trabajo, así como también procesos de recopilación y análisis de requisitos del sistema.

Una metodología de desarrollo de software se refiere al entorno que se usa para estructurar, planificar y controlar el proceso de desarrollo de un sistema de información.

Una gran variedad de metodologías se ha desarrollado a lo largo de los años, cada una de ellas con sus fortalezas y debilidades.

Una determinada metodología no es necesariamente aplicable a todo tipo de proyectos, más bien cada tipo de proyecto tiene una metodología a la que se adapta mejor.

Filosofía

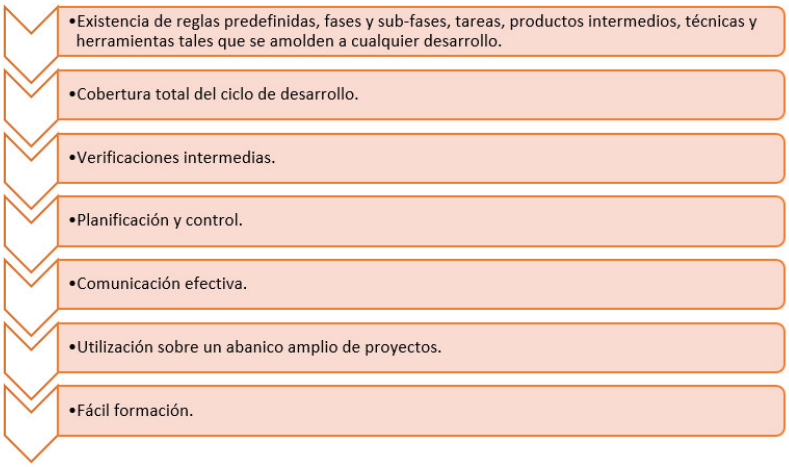
Una filosofía de desarrollo de software con una base de procesos de desarrollo de software.

Utiliza múltiples herramientas, modelos y métodos, para asistir en el proceso de desarrollo de software.

Suele estar promovida por algún tipo de organización ya sea esta pública o privada que es la que se encarga de promover esta metodología.

Define “quién debe hacer qué, cuándo y cómo debe hacerlo”, no existe una metodología de software universal. Las características de cada proyecto (equipo de desarrollo, recursos, etc.) exigen que el proceso sea configurable.

Características deseables



Metodología y ciclo de vida

No confundir Metodología y Ciclo de vida

Metodología

- Conjunto de procedimientos, técnicas, herramientas y un soporte documental que ayuda a los desarrolladores a realizar nuevo software.
- 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

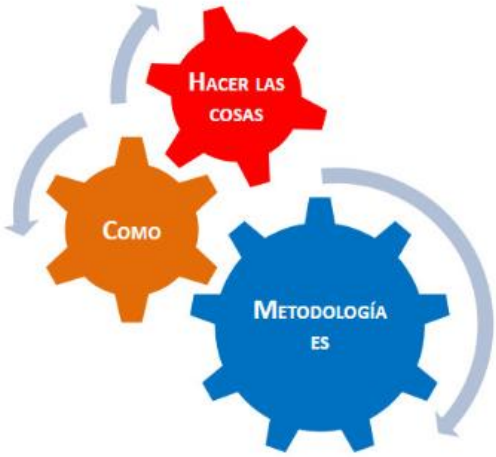
Ciclo de vida

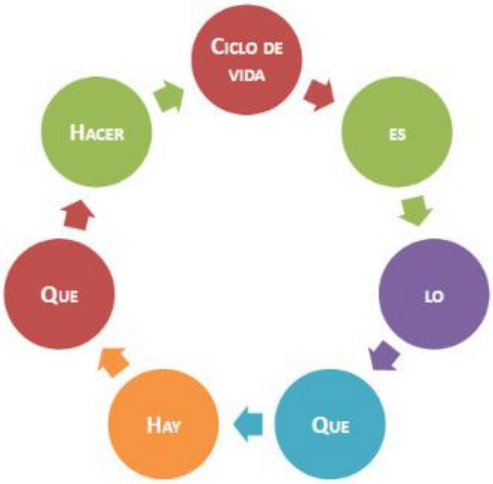
Conjunto de fases por las que pasa el sistema que se está desarrollando desde que nace la idea inicial hasta que el software es retirado o remplazado

- 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, entre otros.

En otras palabras





Tipos de metodologías

La comparación y clasificación de metodologías no es una tarea sencilla debido a la diversidad de propuestas y diferencias en el grado de detalle, la información disponible y alcance de cada una de ellas. Si se toma como criterio las notaciones utilizadas para especificar artefactos producidos en actividades de análisis y diseño, se pueden clasificar las metodologías en dos grupos:

- Metodologías Estructuradas
- Metodologías Orientadas a Objetos.

Metodologías Estructuradas

Orientada a Procesos	Orientada a Datos
Se basa en el modelo básico de entrada/salida de un Sistema. Está compuesta por: • Diagrama de flujo de datos (DFD) • Diccionario de datos • Especificaciones de procesos • La estructura de control del programa debe ser jerárquica y se debe derivar de la estructura de datos del programa	Son metodologías basadas en la información. Primero se definen las estructuras de datos y, a partir de éstos, se derivan los componentes procedimentales.

Metodologías No Estructuradas

Orientada a Objetos	Sistemas de Tiempo Real
La orientación a objetos unifica procesos y datos encapsulándolos en el concepto de objetos. Tiene dos enfoques distintos: • Revolucionario, puro u ortodoxo. Rompen con las metodologías tradicionales. Ejemplos: metodologías OOD de Booch, CRC/RDD de Wirfs-Brock. • Sintetista o evolutivo. Toman como base los sistemas estructurados y conforman elementos de uno y otro tipo. Ejemplos: metodología OMT de Rumbourgh.	Procesan información orientada al control más que a los datos. Se caracterizan por concurrencia, priorización de procesos, comunicación entre tareas y acceso simultáneo a datos comunes.

Por otra parte, si se considera su filosofía de desarrollo se pueden diferenciar en **metodologías tradicionales y ágiles.**

Metodologías

Tradicionales y Ágiles

- Adaptative Software Development
- Agile Modeling
- Agile Model Driven Development
- Agile Project Management
- Agile Unifi ed Process
- Crystal Methods
- Dynamic Systems development methods
- Feature driven development
- Internet Speed Development

- Lean development
- Pragmatic programming
- Scrum
- Test Driven Development
- XBreed
- Extreme Programming
- Win Win Spiral
- Evolutionary Project Management
- Story cards driven development
- Agile Unifi ed Process
- Open Unifi ed Process
- Canbam

Desarrollo ágil

Introducción

En 2001, Kent Beck y otros 16 notables desarrolladores de software, escritores y consultores (grupo conocido como la "Alianza Ágil") firmaron el "Manifiesto por el desarrollo ágil de software". En él se establecía lo siguiente:

Estamos descubriendo formas mejores de desarrollar software, por medio de hacerlo y de dar ayuda a otros para que lo hagan. Ese trabajo nos ha hecho valorar:

- *Los individuos y sus interacciones*, sobre los procesos y las herramientas.
- *El software que funciona*, más que la documentación exhaustiva.
- *La colaboración con el cliente*, y no tanto la negociación del contrato.
- Responder al cambio, mejor que apegarse a un plan.

Es decir, si bien son valiosos los conceptos que aparecen en segundo lugar, valoramos más los que aparecen en primer sitio.

Un manifiesto normalmente se asocia con un movimiento político emergente: ataca a la vieja guardia y sugiere un cambio revolucionario (se espera que para mejorar). En cierta forma, de eso es de lo que trata el desarrollo ágil.

Aunque las ideas subyacentes que lo guían han estado durante muchos años entre nosotros, ha sido en menos de dos décadas que cristalizaron en un "movimiento". Los métodos ágiles (En ocasiones se conoce a los métodos ágiles como métodos ligeros o métodos esbeltos.) se desarrollaron como un esfuerzo por superar las debilidades reales y percibidas de la ingeniería de software convencional. El desarrollo ágil proporciona beneficios importantes, pero no es aplicable a todos los proyectos, productos, personas y situaciones. No es la antítesis de la práctica de la ingeniería de software sólida y puede aplicarse como filosofía general para todo el trabajo de software.

La fluidez implica cambio, y el cambio es caro, en particular si es descontrolado o si se administra mal. Una de las características más atractivas del enfoque ágil es su capacidad de reducir los costos del cambio durante el proceso del software.

¿Significa esto que el reconocimiento de los retos planteados por las realidades modernas hace que sean descartables los valiosos principios, conceptos, métodos y herramientas de la ingeniería del software? No, en absoluto… Igual que todas las disciplinas de la ingeniería, la del software evoluciona en forma continua. Puede adaptarse con facilidad para que satisfaga los desafíos que surgen de la demanda de agilidad.

¿Qué es la agilidad?

qué es la agilidad en el contexto del trabajo de la ingeniería de software? Ivar Jacobson hace un análisis útil:

La *agilidad* se ha convertido en la palabra mágica de hoy para describir un proceso del software moderno. Todos son ágiles. Un equipo ágil es diestro y capaz de responder de manera apropiada a los cambios. El cambio es de lo que trata el software en gran medida. Hay cambios en el software que se construye, en los miembros del equipo, debidos a las nuevas tecnologías, de todas clases y que tienen un efecto en el producto que se elabora o en el proyecto que lo crea. Deben introducirse apoyos para el cambio en todo lo que se haga en el software; en ocasiones se hace porque es el alma y corazón de éste. Un equipo ágil reconoce que el software es desarrollado por individuos que trabajan en equipo, y que su capacidad, su habilidad para colaborar, es el fundamento para el éxito del proyecto.

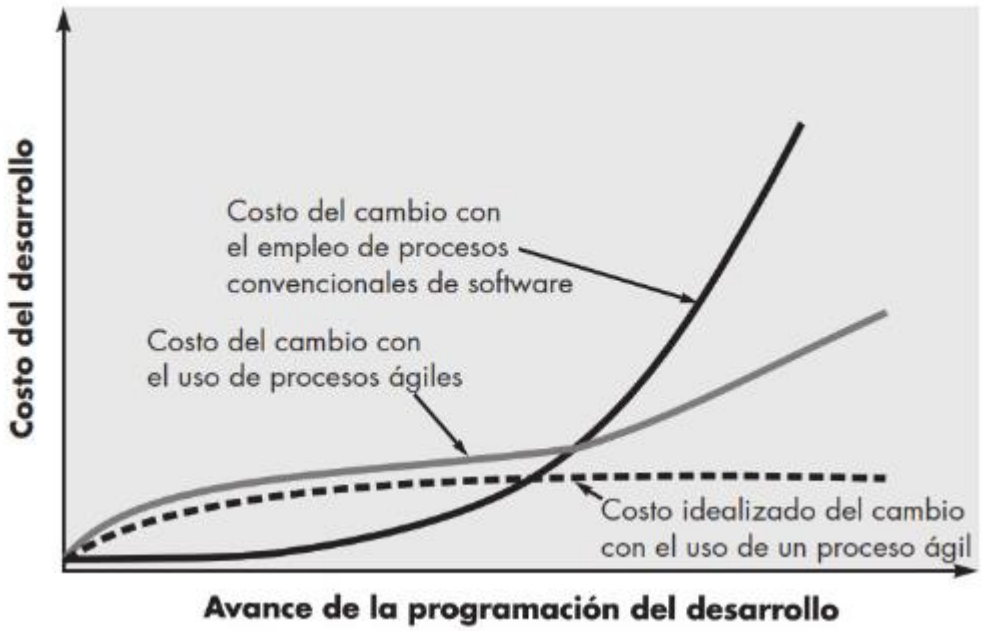
Pero la agilidad es algo más que una respuesta efectiva al cambio. También incluye la filosofía expuesta en el manifiesto citado al principio de este tema. Ésta recomienda las estructuras de equipo y las actitudes que hacen más fácil la comunicación (entre los miembros del equipo, tecnólogos y gente de negocios, entre los ingenieros de software y sus gerentes, etc.); pone el énfasis en la entrega rápida de software funcional y resta importancia a los productos intermedios del trabajo (lo que no siempre es bueno); adopta al cliente como parte del equipo de desarrollo y trabaja para eliminar la actitud de "nosotros y ellos" que todavía invade muchos proyectos de software; reconoce que la planeación en un mundo incierto tiene sus límites y que un plan de proyecto debe ser flexible.

La agilidad puede aplicarse a cualquier proceso del software. Sin embargo, para lograrlo es esencial que éste se diseñe en forma que permita al equipo del proyecto adaptar las tareas y hacerlas directas, ejecutar la planeación de manera que entienda la fluidez de un enfoque ágil del desarrollo, eliminar todos los productos del trabajo excepto los más esenciales y mantenerlos esbeltos, y poner el énfasis en una estrategia de entrega incremental que haga trabajar al software tan rápido como sea posible para el cliente, según el tipo de producto y el ambiente de operación.

La agilidad y el costo del cambio

La sabiduría convencional del desarrollo de software (apoyada por décadas de experiencia) señala que el costo se incrementa en forma no lineal a medida que el proyecto avanza (véase la figura , curva continua negra). Es relativamente fácil efectuar un cambio cuando el equipo de software reúne los requerimientos (al principio de un proyecto). El escenario de uso tal vez tenga que modificarse, la lista de funciones puede aumentar, o editarse una especificación escrita. Los costos de hacer que esto funcione son mínimos, y el tiempo requerido no perjudicará el resultado del proyecto. Pero, ¿Qué pasa una vez transcurridos algunos meses? El equipo está a la mitad de las pruebas de validación (algo que ocurre cuando el proyecto está relativamente avanzado) y un participante de importancia solicita que se haga un cambio funcional grande. El cambio requiere modificar el diseño de la arquitectura del software, el diseño y construcción de tres componentes nuevos, hacer cambios en otros cinco componentes, diseñar nuevas

pruebas, etc. Los costos aumentan con rapidez, y no son pocos el tiempo y el dinero requeridos para asegurar que se haga el cambio sin efectos colaterales no intencionados.



Los defensores de la agilidad afirman que un proceso ágil bien diseñado "aplana" el costo de la curva de cambio (véase la figura, curva continua gris claro), lo que permite que el equipo de software haga cambios en una fase tardía de un proyecto de software sin que haya un efecto notable en el costo y en el tiempo. El lector ya sabe que el proceso ágil incluye la entrega incremental. Cuando ésta se acopla con otras prácticas ágiles, como las pruebas unitarias continuas y la programación por parejas, el costo de hacer un cambio disminuye. Aunque hay debate sobre el grado en el que se aplana la curva de costo, existen evidencias que sugieren que es posible lograr una reducción significativa del costo.

¿Qué es un proceso ágil?

Cualquier proceso del software ágil se caracteriza por la forma en la que aborda cierto número de suposiciones clave acerca de la mayoría de los proyectos de software:

1. Es difícil predecir qué requerimientos de software persistirán y cuáles cambiarán. También es difícil pronosticar cómo cambiarán las prioridades del cliente a medida que avanza el proyecto.
2. Para muchos tipos de software, el diseño y la construcción están imbricados. Es decir, ambas actividades deben ejecutarse en forma simultánea, de modo que los modelos de diseño se prueben a medida que se crean. Es difícil predecir cuánto diseño se necesita antes de que se use la construcción para probar el diseño.
3. El análisis, el diseño, la construcción y las pruebas no son tan predecibles como nos gustaría (desde un punto de vista de planeación).

Dadas estas tres suposiciones, surge una pregunta importante: ¿Cómo crear un proceso que pueda manejar lo *impredecible*? La respuesta, como ya se dijo, está en la adaptabilidad del proceso (al cambio rápido del proyecto y a las condiciones técnicas). Por tanto, un proceso ágil debe ser *adaptable*.

Pero la adaptación continua logra muy poco si no hay avance. Entonces, un proceso de software ágil debe adaptarse incrementalmente. Para lograr la adaptación *incremental*, un equipo ágil requiere retroalimentación con el cliente (de modo que sea posible hacer las adaptaciones apropiadas). Un catalizador eficaz para la retroalimentación con el cliente es un prototipo operativo o una porción de un sistema operativo. Así, debe instituirse una estrategia de desarrollo incremental. Deben entregarse incrementos de software (prototipos ejecutables o porciones de un sistema operativo) en periodos cortos de tiempo, de modo que la adaptación vaya a ritmo con el cambio (impredecible). Este enfoque iterativo permite que el cliente evalúe en forma regular el incremento de software, dé la retroalimentación necesaria al equipo de software e influya en las adaptaciones del proceso que se realicen para aprovechar la retroalimentación.

Principios de agilidad

La Alianza Ágil define 12 principios de agilidad para aquellos que la quieran alcanzar:

1. La prioridad más alta es satisfacer al cliente a través de la entrega pronta y continua de software valioso.
2. Son bienvenidos los requerimientos cambiantes, aun en una etapa avanzada del desarrollo. Los procesos ágiles dominan el cambio para provecho de la ventaja competitiva del cliente.
3. Entregar con frecuencia software que funcione, de dos semanas a un par de meses, de preferencia lo más pronto que se pueda.
4. Las personas de negocios y los desarrolladores deben trabajar juntos, a diario y durante todo el proyecto.
5. Hay que desarrollar los proyectos con individuos motivados. Debe darse a éstos el ambiente y el apoyo que necesiten, y confiar en que harán el trabajo.
6. El método más eficiente y eficaz para transmitir información a los integrantes de un equipo de desarrollo, y entre éstos, es la conversación cara a cara.
7. La medida principal de avance es el software que funciona.
8. Los procesos ágiles promueven el desarrollo sostenible. Los patrocinadores, desarrolladores y usuarios deben poder mantener un ritmo constante en forma indefinida.
9. La atención continua a la excelencia técnica y el buen diseño mejora la agilidad.
10. Es esencial la simplicidad: el arte de maximizar la cantidad de trabajo no realizado.
11. Las mejores arquitecturas, requerimientos y diseños surgen de los equipos con organización propia.
12. El equipo reflexiona a intervalos regulares sobre cómo ser más eficaz, para después afinar y ajustar su comportamiento en consecuencia.

No todo modelo de proceso ágil aplica estos 12 principios con igual intensidad y algunos eligen ignorar (o al menos soslayar) la importancia de uno o más de ellos. Sin embargo, los principios definen un espíritu ágil que se mantiene en cada uno de los modelos de proceso que se verán más adelante.