TECNOLOGICO NACIONAL CEICOM

CARRERA DE SISTEMAS INFORMÁTICOS

F

PRÁCTICA 01

INGENIERÍA DE SOFTWARE

Materia: Gestión de Software

Elaborado por: Fabrica Plaz Estevan

Docente: Ing. Baltazar Llusco Ever Jaime

Fecha: 14 de enero de 2019

Cochabamba – Bolivia

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

El software y la ingeniería de software

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 (el 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?

Éstas y muchas otras preguntas, denotan la preocupación sobre el software y la manera en que se desarrolla, preocupación que ha llevado a la adopción de la práctica de la ingeniería del software.

Definición de software

En la actualidad, la mayoría de profesionales y muchos usuarios tienen la fuerte sensación de que entienden el software.

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.

No hay duda de que podrían darse definiciones más completas.

Es importante examinar las características del software que lo hacen diferente de otros objetos que construyen los seres humanos. El software es el elemento de un sistema lógico y no de uno físico. Por tanto, tiene características que definen considerablemente de las del hardware:

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

Software y hardware, en sus 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.

El software no se “desgasta”.

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.

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” 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!

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” 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.

Otro aspecto del desgaste ilustra la diferencia entre el hardware y el software.

Cuando un componente del hardware se desgasta es sustituido por una refacción. En cambio, no hay refacciones para el software. Cada falla de éste indica un error en el diseño o en el proceso que tradujo el diseño a código ejecutable por la máquina. Entonces, las tareas de mantenimiento del software, que incluyen la satisfacción de peticiones de cambios, involucran una complejidad considerablemente mayor que el mantenimiento del hardware.

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 reconstruidos 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. Si quiere reducir el deterioro del software, tendrá que mejorar su diseño.

Tasa de fallas incrementada debido a efectos colaterales

Tasa de fallas

1. Cambio
2. Curva real
3. Curva idealizada
4. Curvas de falla del software

En realidad, los distintos participantes solicitan cambios desde el momento en que comienza el desarrollo y mucho antes de que se disponga de la primera versión. Natural del proceso de ingeniería. En el del software, es algo que apenas ha empezado 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.

Dominios de aplicación del software

Actualmente, hay siete grandes categorías de software de computadora que plantean retos continuos a los ingenieros de software:

Software de sistemas: conjunto de programas escritos para dar servicio a otros programas.

Determinado software de sistemas (compiladores, editores y herramientas para administrar archivos) procesa estructuras de informaciones complejas pero deterministas.

Otras aplicaciones de sistemas (por ejemplo, componentes de sistemas operativos, manejadores, software de redes, procesadores de telecomunicaciones) procesan sobre todo datos indeterminados. En cualquier caso, el área de software de sistemas se caracteriza por: gran interacción con el hardware de la computadora, uso intensivo por parte de usuarios múltiples, operación concurrente

que requiere la secuenciación, recursos compartidos y administración de un proceso sofisticado, estructuras complejas de datos e interfaces externas múltiples.

Software de aplicación: programas aislados que resuelven una necesidad específica de negocios. Las aplicaciones en esta área procesan datos comerciales o técnicos en una forma que facilita las operaciones de negocios o la toma de decisiones administrativas o técnicas. Además de las aplicaciones convencionales de procesamiento de datos, el software de aplicación se usa para controlar funciones de negocios en tiempo real (por ejemplo, procesamiento de transacciones en punto de venta, control de procesos de manufactura en tiempo real).

Software de ingeniería y ciencias: se ha caracterizado por algoritmos. Las aplicaciones van de la astronomía a la vulcanología, del análisis de tensiones en automóviles a la dinámica orbital del transbordador espacial, y de la biología molecular a la manufactura automatizada. Sin embargo, las aplicaciones modernas dentro del área de la ingeniería y las ciencias están abandonando los algoritmos numéricos convencionales.

El diseño asistido por computadora, la simulación de sistemas y otras aplicaciones interactivas, han comenzado a hacerse en tiempo real e incluso han tomado características del software de sistemas.

Software incrustado: reside dentro de un producto o sistema y se usa para implementar y controlar características y funciones para el usuario final y para el sistema en sí. El software incrustado ejecuta funciones limitadas y particulares (por ejemplo, control del tablero de un horno de microondas) o provee una capacidad significativa de funcionamiento y control.

El software es no determinista si no pueden predecirse el orden y momento en que ocurren éstos. (Funciones, digitales en un automóvil, como el control del combustible, del tablero de control y de los sistemas de frenado).

Software de línea de productos: El software de línea de productos se centra en algún mercado limitado y particular (por ejemplo, control del inventario de productos) o se dirige a mercados masivos de consumidores (procesamiento de textos, hojas de cálculo, gráficas por computadora, multimedios, entretenimiento, administración de base de datos y aplicaciones para finanzas personales o de negocios).

Aplicaciones web: llamadas “*webapps*”, esta categoría de software centrado en redes agrupa una amplia gama de aplicaciones. En su forma más sencilla, las *webapps* son poco más que un conjunto de archivos de hipertexto vinculados que presentan información con uso de texto y gráficas limitadas. Sin embargo, desde que surgió Web, las *webapps* están evolucionando hacia ambientes de cómputo sofisticados que no sólo proveen características aisladas, funciones de cómputo y contenido para el usuario final, sino que también están integradas con bases de datos corporativas y aplicaciones de negocios.

Software de inteligencia artificial: hace uso de algoritmos no numéricos para resolver problemas complejos que no son fáciles de tratar computacionalmente o con el análisis directo.

Las aplicaciones en esta área incluyen robótica, sistemas expertos, reconocimiento de patrones (imagen y voz), redes neurales artificiales, demostración de teoremas y juegos.

Son millones de ingenieros de software en todo el mundo los que trabajan duro en proyectos de software en una o más de estas categorías. En ciertos casos se elaboran sistemas nuevos, pero en muchos otros se corrigen, adaptan y mejoran aplicaciones ya existentes.

Computación en un mundo abierto: el crecimiento de las redes inalámbricas quizá lleve pronto a la computación verdaderamente ubicua y distribuida. El reto para los ingenieros de software será desarrollar software de sistemas y aplicación que permita a dispositivos móviles, computadoras personales y sistemas empresariales comunicarse a través de redes enormes.

Construcción de redes: la red mundial (World Wide Web) se está convirtiendo con rapidez tanto en un motor de computación como en un proveedor de contenido. El desafío para los ingenieros de software es hacer arquitecturas sencillas (por ejemplo, planeación financiera personal y aplicaciones sofisticadas que proporcionen un beneficio a mercados objetivo de usuarios finales en todo el mundo).

Fuente abierta: tendencia creciente que da como resultado la distribución de código fuente para aplicaciones de sistemas (por ejemplo, sistemas operativos, bases de datos y ambientes de desarrollo) de modo que mucha gente pueda contribuir a su desarrollo. El desafío para los ingenieros de software es elaborar código fuente que sea auto descriptivo, y también, lo que es más importante, desarrollar técnicas que permitirán tanto a los consumidores como a los desarrolladores saber cuáles son los cambios hechos y cómo se manifiestan dentro del software.

Software heredado

Cientos de miles de programas de cómputo caen en uno de los siete dominios amplios de aplicación que se estudiaron en la subsección anterior. Algunos de ellos son software muy nuevo, disponible Estos programas antiguos que es frecuente denominar software heredado han sido centro de atención y preocupación continuas desde la década de 1960. Dayani-Fard y sus colegas Day99 describen el software heredado de la siguiente manera:

Los sistemas de software heredado fueron desarrollados hace varias décadas y han sido modificados de manera continua para que satisfagan los cambios en los requerimientos de los negocios y plataformas de computación. La proliferación de tales sistemas es dolores de cabeza para las organizaciones grandes, a las que resulta costoso mantenerlos y riesgoso hacerlos evolucionar.

Liu y sus colegas Liu98] amplían esta descripción al hacer notar que “muchos sistemas heredados continúan siendo un apoyo para las funciones básicas del negocio y son ‘indispensables’ para éste”. Además, el software heredado se caracteriza por su longevidad e importancia crítica para el negocio.

Desafortunadamente, en ocasiones hay otra característica en el software heredado: mala calidad. Tienen diseños que no son susceptibles de extenderse, código confuso, documentación mala o inexistente, casos y resultados de pruebas que nunca se archivaron, una historia de los cambios mal administrada la lista es muy larga. La única respuesta razonable es: hacer nada, al menos hasta que el sistema heredado tenga un cambio significativo. Si el software heredado satisface las necesidades de sus usuarios y corre de manera confiable, entonces no falla ni necesita repararse. Sin embargo, conforme pase el tiempo será frecuente que los sistemas de software evolucionen por una o varias de las siguientes razones:

- El software debe adaptarse para que cumpla las necesidades de los nuevos ambientes del cómputo y de la tecnología.
- El software debe ser mejorado para implementar nuevos requerimientos del negocio.
- El software debe ampliarse para que sea operable con otros sistemas o bases de datos modernos.
- La arquitectura del software debe rediseñarse para hacerla viable dentro de un ambiente de redes.

Cuando ocurren estos modos de evolución, debe hacerse la reingeniería del sistema heredado para que sea viable en el futuro. La meta de la ingeniería de software moderna es “desarrollar metodologías que se basen en el concepto de evolución; es decir, el concepto de que los sistemas de software cambian continuamente, que los nuevos sistemas de software.

La naturaleza única de las *webapps*

En los primeros días de la Red Mundial (entre 1990 y 1995), los *sitios web* consistían en poco más que un conjunto de archivos de hipertexto vinculados que presentaban la información con el empleo de texto y gráficas limitadas. Al pasar el tiempo, el aumento de HTML por medio de herramientas de desarrollo (XML, Java) permitió a los ingenieros de la web brindar capacidad de cómputo junto con contenido de información. Habían nacido los *sistemas y aplicaciones basados en la web*6 (denominó a éstas en forma colectiva como *webapps*). En la actualidad, las *webapps* se han convertido en herramientas sofisticadas de cómputo que no sólo proporcionan funciones aisladas al usuario final, sino que también se han integrado con bases de datos corporativas y aplicaciones de negocios.

Powell Pow sugiere que los sistemas y aplicaciones basados en web “involucran una mezcla entre las publicaciones empresas y el desarrollo de software, entre la mercadotecnia y la computación, entre las comunicaciones internas y las relaciones exteriores, y entre el arte y la tecnología”. La gran mayoría de *webapps* presenta los siguientes atributos:

Uso intensivo de redes. Una *webapp* reside en una red y debe atender las necesidades de una comunidad diversa de clientes. La red permite acceso y comunicación mundiales (por ejemplo, internet) o tiene acceso y comunicación limitados (por ejemplo, una intranet corporativa).

Concurrencia. A la *webapp* puede acceder un gran número de usuarios a la vez. En muchos casos, los patrones de uso entre los usuarios finales varían mucho.

Carga impredecible. El número de usuarios de la *webapp* cambia en varios órdenes de magnitud de un día a otro. El lunes tal vez la utilicen cien personas, el jueves quizá 10 000 usen el sistema.

Rendimiento. Si un usuario de la *webapp* debe esperar demasiado (para entrar, para el procesamiento por parte del servidor, para el formado y despliegue del lado del cliente), él o ella quizá decidan irse a otra parte.

Disponibilidad. Aunque no es razonable esperar una disponibilidad de 100%, es frecuente que los usuarios de webapps populares demanden acceso las 24 horas de los 365 días del año. Los usuarios en Australia o Asia quizá demanden acceso en horas en las que las aplicaciones internas de software tradicionales en Norteamérica no estén en línea por razones de mantenimiento.

Orientadas a los datos. La función principal de muchas webapp es el uso de hipermedios para presentar al usuario final contenido en forma de texto, gráficas, audio y video.

Además, las webapps se utilizan en forma común para acceder a información que existe en bases de datos que no son parte integral del ambiente basado en web (por ejemplo, comercio electrónico o aplicaciones financieras).

En el contexto de este libro, el término *aplicación web* (*webapp*) agrupa todo, desde una simple página web que ayude al consumidor a calcular el pago del arrendamiento de un automóvil hasta un sitio web integral que proporcione servicios completos de viaje para gente de negocios y vacacionistas. En esta categoría se incluyen sitios web completos, funcionalidad especializada dentro

de sitios web y aplicaciones de procesamiento de información que residen en internet o en una intranet o extranet.

Contenido sensible. La calidad y naturaleza estética del contenido constituye un rasgo importante de la calidad de una webapp.

Evolución continúa. A diferencia del software de aplicación convencional que evoluciona a lo largo de una serie de etapas planeadas y separadas cronológicamente, las aplicaciones web evolucionan en forma continua. No es raro que ciertas *webapp* (específicamente su contenido) se actualicen minuto a minuto o que su contenido se calcule en cada solicitud.

Inmediatez. Aunque la inmediatez necesidad apremiante de que el software llegue con rapidez al mercado es una característica en muchos dominios de aplicación, es frecuente que las webapps tengan plazos de algunos días o semanas para llegar al mercado.

Seguridad. Debido a que las *webapps* se encuentran disponibles con el acceso a una red,es difícil o imposible limitar la población de usuarios finales que pueden acceder a la aplicación.

Con el fin de proteger el contenido sensible y brindar modos seguros de transmisión de los datos, deben implementarse medidas estrictas de seguridad a través de la infraestructura de apoyo de una webapp y dentro de la aplicación misma.

Estética. Parte innegable del atractivo de una webapp es su apariencia y percepción.

Cuando se ha diseñado una aplicación para comercializar o vender productos o ideas, la estética tiene tanto que ver con el éxito como el diseño técnico.

Ingeniería de software

Con objeto de elaborar software listo para enfrentar los retos del siglo XXI, el lector debe aceptar algunas realidades sencillas:

- El software se ha incrustado profundamente en casi todos los aspectos de nuestras vidas y, como consecuencia, el número de personas que tienen interés en las características y funciones que brinda una aplicación específica ha crecido en forma notable. Cuando ha de construirse una aplicación nueva o sistema incrustado que debe hacerse un esfuerzo concertado para entender el problema antes de desarrollar una aplicación de software.
- Los requerimientos de la tecnología de la información que demandan los individuos, negocios y gobiernos se hacen más complejos con cada año que pasa. En la actualidad, grandes equipos de personas crean programas de cómputo que antes eran elaborados por un solo individuo. El software sofisticado, que alguna vez se implementó en un ambiente de cómputo predecible y auto contenido, hoy en día se halla incrustado en el interior de todo, desde la electrónica de consumo hasta dispositivos médicos o sistemas de armamento. La complejidad de estos nuevos sistemas y productos basados en computadora demanda atención cuidadosa a las interacciones de todos los elementos del sistema. Se concluye que el diseño se ha vuelto una actividad crucial.

Con las herramientas modernas es posible producir páginas web sofisticadas en unas cuantas horas.

- Los individuos, negocios y gobiernos dependen cada vez más del software para tomar decisiones estratégicas y tácticas, así como para sus operaciones y control cotidianos. Si el software falla, las personas y empresas grandes pueden experimentar desde un inconveniente menor hasta fallas catastróficas. Se concluye que el software debe tener alta calidad.

- A medida que aumenta el valor percibido de una aplicación específica se incrementa la probabilidad de que su base de usuarios y longevidad también crezcan. Conforme se extienda su base de usuarios y el tiempo de uso, las demandas para adaptarla y mejorarla también crecerán. *Se concluye* que el software debe tener facilidad para recibir mantenimiento.

Estas realidades simples llevan a una conclusión: debe hacerse ingeniería con el software en todas sus formas y a través de todos sus dominios de aplicación.

Aunque cientos de autores han desarrollado definiciones personales de la ingeniería de software, la propuesta por Fritz Bauer Nau69 en la conferencia fundamental sobre el tema todavía sirve como base para el análisis:

La ingeniería de software es el establecimiento y uso de principios fundamentales de la ingeniería con objeto de desarrollar en forma económica software que sea confiable y que trabaje con eficiencia en máquinas reales.

Bauer proporciona una base. ¿Cuáles son los "principios fundamentales de la ingeniería" que pueden aplicarse al desarrollo del software de computadora? ¿Cómo se desarrolla software "en forma económica" y que sea "confiable"? ¿Qué se requiere para crear programas de cómputo que trabajen con "eficiencia", no en una sino en muchas "máquinas reales" diferentes? Éstas son las preguntas que siguen siendo un reto para los ingenieros de software.

El IEEE ha desarrollado una definición más completa, como sigue:

La ingeniería de software es: 1) La aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento de software; es decir, la aplicación de la ingeniería al software.

Aun así, el enfoque "sistemático, disciplinado y cuantificable" aplicado por un equipo de software podría ser algo burdo para otro. Se necesita disciplina, pero también adaptabilidad y agilidad.

La ingeniería de software es una tecnología con varias capas. Cualquier enfoque de ingeniería (incluso la de software) debe basarse en un compromiso organizacional con la calidad. La administración total de la calidad, Six Sigma y otras filosofías

El fundamento para la ingeniería de software es la capa *proceso*. El proceso de ingeniería de software es el aglutinante que une las capas de la tecnología y permite el desarrollo racional.

Tanto la calidad como la facilidad de recibir mantenimiento son resultado de un buen diseño. "Más que una disciplina o cuerpo de conocimientos, ingeniería es un verbo, una palabra de acción, una forma de abordar un problema."

La ingeniería de software incluye un proceso, métodos y herramientas para administrar y hacer ingeniería con el software.

El proceso define una estructura que debe establecerse para la obtención eficaz de tecnología de ingeniería de software. El proceso de software forma la base para el control de la administración de proyectos de software, y establece el contexto en el que se aplican métodos técnicos, se generan productos del trabajo (modelos, documentos, datos, reportes, formatos, etc.), se establecen puntos de referencia, se asegura la calidad y se administra el cambio de manera apropiada.

Los métodos de la ingeniería de software proporcionan la experiencia técnica para elaborar software. Incluyen un conjunto amplio de tareas, como comunicación, análisis de los requerimientos, modelación del diseño, construcción del programa, pruebas y apoyo. Los métodos de la ingeniería de

software se basan en un conjunto de principios fundamentales que gobiernan cada área de la tecnología e incluyen actividades de modelación y otras técnicas descriptivas.

Las herramientas de la ingeniería de software proporcionan un apoyo automatizado o semiautomatizado para el proceso y los métodos. Cuando se integran las herramientas de modo que la información creada por una pueda ser utilizada por otra, queda establecido un sistema llamado ingeniería de software asistido por computadora que apoya el desarrollo de software.

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.

Planeación. 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. El ingeniero de software crea 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.

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.

¿Cuáles son las cinco actividades estructurales del proceso?

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

Su objetivo es mejorar la calidad del sistema, desarrollar proyectos más manejables, hacer más predecibles las fechas de entrega y los costos, y guiar a los equipos de ingenieros de software cuando realizan el trabajo que se requiere para construir un sistema. Desafortunadamente, ha habido casos en los que estos objetivos no se han logrado. Si los modelos prescriptivos se aplican en forma dogmática y sin adaptación, pueden incrementar el nivel de burocracia asociada con el desarrollo de sistemas basados en computadora y crear inadvertidamente dificultades para todos los participantes.

Los modelos de proceso ágil ponen el énfasis en la "agilidad" del proyecto y siguen un conjunto de principios que conducen a un enfoque más informal (pero no menos efectivo, dicen sus defensores) del proceso de software. Por lo general, se dice que estos modelos del proceso son "ágiles" porque acentúan la maniobrabilidad y la adaptabilidad. Son apropiados para muchos tipos de proyectos y son útiles en particular cuando se hace ingeniería sobre aplicaciones web.

La práctica de la ingeniería de software

1. Entender el problema (comunicación y análisis).

2. Planear la solución (modelado y diseño del software).

3. Ejecutar el plan (generación del código).

4. Examinar la exactitud del resultado (probar y asegurar la calidad).

En el contexto de la ingeniería de software, estas etapas de sentido común conducen a una serie de preguntas esenciales.

Entender el problema. Desafortunadamente, entender no siempre es fácil. Es conveniente dedicar un poco de tiempo a responder algunas preguntas sencillas:

- ¿Quiénes tienen que ver con la solución del problema? Es decir, ¿quiénes son los participantes?
- ¿Cuáles son las incógnitas? ¿Cuáles datos, funciones y características se requieren para resolver el problema en forma apropiada?
- ¿Puede fraccionarse el problema? ¿Es posible representarlo con problemas más pequeños que sean más fáciles de entender?
- ¿Es posible representar gráficamente el problema? ¿Puede crearse un modelo de análisis?

Planear la solución•

¿Ha visto antes problemas similares? ¿Hay patrones reconocibles en una solución potencial? ¿Hay algún software existente que implemente los datos, funciones y características que se requieren?

- ¿Ha resuelto un problema similar? Si es así, ¿son reutilizables los elementos de la solución?
- ¿Pueden definirse problemas más pequeños? Si así fuera, ¿hay soluciones evidentes para éstos?

Podría decirse que el enfoque de Polya es simple sentido común. Es verdad. Pero es sorprendente la frecuencia con la que el sentido común es poco común en el mundo del software.

- ¿Es capaz de representar una solución en una forma que lleve a su implementación eficaz?

¿Es posible crear un modelo del diseño?

Ejecutar el plan. El diseño que creó sirve como un mapa de carreteras para el sistema que quiere construir. Puede haber desviaciones inesperadas y es posible que descubra un camino mejor a medida que avanza, pero el "plan" le permitirá proceder sin que se pierda.

- ¿Se ajusta la solución al plan? ¿El código fuente puede apegarse al modelo del diseño?
- ¿Es probable que cada parte componente de la solución sea correcta? ¿El diseño y código se han revisado o, mejor aún, se han hecho pruebas respecto de la corrección del algoritmo?

Examinar el resultado.

- ¿Puede probarse cada parte componente de la solución? ¿Se ha implementado una estrategia razonable para hacer pruebas?
- ¿La solución produce resultados que se apegan a los datos, funciones y características que se requieren? ¿El software se ha validado contra todos los requerimientos de los participantes?

No debiera sorprender que gran parte de este enfoque tenga que ver con el sentido común. En realidad, es razonable afirmar que un enfoque de sentido común para la ingeniería de software hará que nunca se extravíe.

Principios generales

David Hooker Hoo propuso siete principios que se centran en la práctica de la ingeniería de software como un todo. Se reproducen en los párrafos siguientes:

Primer principio: La razón de que exista todo

Un sistema de software existe por una razón: *dar valor a sus usuarios.* Todas las decisiones deben tomarse teniendo esto en mente. Antes de especificar un requerimiento del sistema, antes de notar la funcionalidad de una parte de él, antes de determinar las plataformas del hardware o desarrollar procesos, plantéese preguntas tales como: "¿Esto agrega valor real al sistema?" Si la respuesta es "no", entonces no lo haga. Todos los demás principios apoyan a éste.

Segundo principio: MSE (Mantenlo sencillo, estúpido…)

El diseño de software no es un proceso caprichoso. Hay muchos factores por considerar en cualquier actividad de diseño. Todo diseño debe ser tan simple como sea posible, pero no más. Reproducido con permiso del autor [Hoo96]. Hooker define algunos patrones para estos principios

Esto facilita conseguir un sistema que sea comprendido más fácilmente y que sea susceptible de recibir mantenimiento, lo que no quiere decir que en nombre de la simplicidad deban descartarse características o hasta rasgos internos. En realidad, los diseños más elegantes por lo general son los más simples. Simple tampoco significa "rápido y sucio". La verdad es que con frecuencia se requiere mucha reflexión y trabajo con iteraciones múltiples para poder simplificar.

La recompensa es un software más fácil de mantener y menos propenso al error.

Tercer principio: Mantener la visión

Una visión clara es esencial para el éxito de un proyecto de software. Sin ella, casi infaliblemente el proyecto terminará siendo un ser "con dos [o más mentes]". Sin integridad conceptual, un sistema está amenazado de convertirse en una urdimbre de diseños incompatibles unidos por tornillos del tipo equivocado Comprometer la visión de la arquitectura de un sistema de software debilita y, finalmente hará que colapsen incluso los sistemas bien diseñados.

Tener un arquitecto que pueda para mantener la visión y que obligue a su cumplimiento garantiza un proyecto de software muy exitoso.

Cuarto principio: Otros consumirán lo que usted produce

Rara vez se construye en el vacío un sistema de software con fortaleza industrial. En un modo u otro, alguien más lo usará, mantendrá, documentará o, de alguna forma, dependerá de su capacidad para entender el sistema. Así que siempre establezca especificaciones, diseñe e implemente con la seguridad de que alguien más tendrá que entender lo que usted haga. La audiencia para cualquier producto de desarrollo de software es potencialmente grande. Elabore especificaciones con la mirada puesta en los usuarios. Diseñe con los implementadores en mente. Codifique pensando en aquellos que deben dar mantenimiento y ampliar el sistema.

Alguien debe depurar el código que usted escriba, y eso lo hace usuario de su código. Hacer su trabajo más fácil agrega valor al sistema.

Quinto principio: Ábrase al futuro

Un sistema con larga vida útil tiene más valor. En los ambientes de cómputo actuales, donde las especificaciones cambian de un momento a otro y las plataformas de hardware quedan obsoletas con sólo unos meses de edad, es común que la vida útil del software se mida en meses y no en años. Sin embargo, los sistemas de software con verdadera “fortaleza industrial” deben durar mucho más tiempo. Para tener éxito en esto, los sistemas deben ser fáciles de adaptar a ésos y otros cambios. Los sistemas que lo logran son los que se diseñaron para ello desde el principio. *Nunca diseñe sobre algo* iniciado. Siempre pregunte: “¿qué pasa si?” y prepárese para todas las respuestas posibles mediante la creación de sistemas que resuelvan el problema general, no sólo uno específico.14 Es muy posible que esto lleve a volver a usar un sistema completo.

Sexto principio: Planee por anticipado la reutilización

La reutilización ahorra tiempo y esfuerzo.15 Al desarrollar un sistema de software, lograr un alto nivel de reutilización es quizá la meta más difícil de lograr. La reutilización del código y de los diseños se ha reconocido como uno de los mayores beneficios de usar tecnologías orientadas a objetos. Sin embargo, la recuperación de esta inversión no es automática. Para reforzar las posibilidades de la reutilización que da la programación orientada a objetos.

Si el software tiene valor, cambiará durante su vida útil. Por esa razón, debe construirse de forma que sea fácil darle mantenimiento.

Es peligroso llevar este consejo a los extremos. Diseñar para resolver “el problema general” en ocasiones requiere compromisos de rendimiento y puede volver ineficientes las soluciones específicas.

Aunque esto es verdad para aquellos que reutilizan software en proyectos futuros, volver a usar puede ser caro para quienes deben diseñar y elaborar componentes reutilizables. Los estudios indican que diseñar y construir componentes reutilizables llega a costar entre 25 y 200% más que el software buscado. En ciertos casos no se justifica la diferencia de costos. Convencional], se requiere reflexión y planeación. Hay muchas técnicas para incluir la reutilización en cada nivel del proceso de desarrollo del sistema. La planeación anticipada en busca de la reutilización disminuye el costo e incrementa el valor tanto de los componentes reutilizables como de los sistemas en los que se incorpora.

Séptimo principio: ¡Piense!

Este último principio es tal vez el que más se pasa por alto. Pensar en todo con claridad antes de emprender la acción casi siempre produce mejores resultados. Cuando se piensa en algo es más probable que se haga bien. Asimismo, también se gana conocimiento al pensar cómo volver a hacerlo bien. Si usted piensa en algo y aun así lo hace mal, eso se convierte en una experiencia valiosa. Un efecto colateral de pensar es aprender a reconocer cuando no se sabe algo, punto en el que se puede investigar la respuesta. Cuando en un sistema se han puesto pensamientos claros, el valor se manifiesta. La aplicación de los primeros seis principios requiere pensar con intensidad, por lo que las recompensas potenciales son enormes.

Si todo ingeniero y equipo de software tan sólo siguiera los siete principios de Hooker, se eliminarían muchas de las dificultades que se experimentan al construir sistemas complejos basados en computadora.

Mitos del software

Los mitos del software creencias erróneas sobre éste y sobre el proceso que se utiliza para obtenerlo se remontan a los primeros días de la computación. Los mitos tienen cierto número de atributos que los hacen insidiosos. Por ejemplo, parecen enunciados razonables de hechos (a veces contienen elementos de verdad), tienen una sensación intuitiva y es frecuente que los manifiesten profesionales experimentados que “conocen la historia”.

En la actualidad, la mayoría de profesionales de la ingeniería de software reconocen los mitos como lo que son: actitudes equivocadas que han ocasionado serios problemas a los administradores y a los trabajadores por igual. Sin embargo, las actitudes y hábitos antiguos son difíciles de modificar, y persisten algunos remanentes de los mitos del software.

Mitos de la administración. Los gerentes que tienen responsabilidades en el software, como los de otras disciplinas, con frecuencia se hallan bajo presión para cumplir el presupuesto, mantener la programación de actividades sin desvíos y mejorar la calidad. Así como la persona que se ahoga se agarra de un clavo ardiente, no es raro que un gerente de software sostenga la creencia en un mito del software si eso disminuye la presión a que está sujeto (incluso de manera temporal).

Mito: Tenemos un libro lleno de estándares y procedimientos para elaborar software.

Realidad: Tal vez exista el libro de estándares, pero ¿se utiliza? ¿Saben de su existencia los trabajadores del software? ¿Refleja la práctica moderna de la ingeniería de software? ¿Es completo? ¿Es adaptable? ¿Está dirigido a mejorar la entrega a tiempo y también se centra en la calidad? En muchos casos, la respuesta a todas estas preguntas es “no”.

Mito: Si nos atrasamos, podemos agregar más programadores y ponernos al corriente (en ocasiones, a esto se le llama “concepto de la horda de mongoles”).

Realidad: El desarrollo del software no es un proceso mecánico similar a la manufactura.

En palabras de Brooks Bro: “agregar personal a un proyecto de software atrasado lo atrasará más”. Al principio, esta afirmación parece ir contra la intuición. Sin embargo, a medida que se agregan personas, las que ya se encontraban trabajando deben dedicar tiempo para enseñar a los recién llegados, lo que disminuye la cantidad de tiempo dedicada al esfuerzo de desarrollo productivo. Pueden agregarse individuos, pero sólo en forma planeada y bien coordinada.

Mito: Si decido subcontratar el proyecto de software a un tercero, puedo descansar y dejar que esa compañía lo elabore.

Realidad: Si una organización no comprende cómo administrar y controlar proyectos de software internamente, de manera invariable tendrá dificultades cuando subcontrate proyectos de software.

Mitos del cliente. El cliente que requiere software de computadora puede ser la persona en el escritorio de al lado, un grupo técnico en el piso inferior, el departamento de mercadotecnia y ventas, o una compañía externa que solicita software mediante un contrato. En muchos casos, el cliente sostiene mitos sobre el software porque los gerentes y profesionales de éste hacen poco para corregir la mala información. Los mitos generan falsas expectativas (por parte del cliente) y, en última instancia, la insatisfacción con el desarrollador.

Mito: Para comenzar a escribir programas, es suficiente el enunciado general de los objetivos podremos entrar en detalles más adelante.

Realidad: Aunque no siempre es posible tener el enunciado exhaustivo y estable de los requerimientos, un "planteamiento de objetivos" ambiguo es una receta para el desastre. Los requerimientos que no son ambiguos (que por lo general se obtienen en forma iterativa) se desarrollan sólo por medio de una comunicación eficaz y continua entre el cliente y el desarrollador.

Mito: Los requerimientos del software cambian continuamente, pero el cambio se asimila con facilidad debido a que el software es flexible.

Realidad: Es verdad que los requerimientos del software cambian, pero el efecto que los cambios tienen varía según la época en la que se introducen. Cuando se solicitan al principio cambios en los requerimientos (antes de que haya comenzado el diseño o la elaboración de código), el efecto sobre el costo es relativamente pequeño.16 Sin embargo, conforme pasa el tiempo, el costo aumenta con rapidez: los recursos ya se han comprometido, se ha establecido la estructura del diseño y el cambio ocasiona perturbaciones que exigen recursos adicionales y modificaciones importantes del diseño.

Mitos del profesional. Los mitos que aún sostienen los trabajadores del software han sido alimentados por más de 50 años de cultura de programación. Durante los primeros días, la programación se veía como una forma del arte. Es difícil que mueran los hábitos y actitudes arraigados.

Mito: Una vez que escribimos el programa y hacemos que funcione, nuestro trabajo ha terminado.

Realidad: Alguien dijo alguna vez que "entre más pronto se comience a 'escribir el código', más tiempo tomará hacer que funcione". Los datos de la industria indican que entre 60 y 80% de todo el esfuerzo dedicado al software ocurrirá después de entregarlo al cliente por primera vez.

Mito: Hasta que no se haga "correr" el programa, no hay manera de evaluar su calidad.

Muchos ingenieros de software han adoptado un enfoque "ágil" que asimila los cambios en forma gradual y creciente, con lo que controlan su efecto y costo.

Realidad: Uno de los mecanismos más eficaces de asegurar la calidad del software puede aplicarse desde la concepción del proyecto: *la revisión técnica*. Las revisiones del software son un "filtro de la calidad" que se ha revelado más eficaz que las pruebas para encontrar ciertas clases de defectos de software.

Mito: El único producto del trabajo que se entrega en un proyecto exitoso es el programa que funciona.

Realidad: Un programa que funciona sólo es una parte de una configuración de software que incluye muchos elementos. Son varios los productos terminados (modelos, documentos, planes) que proporcionan la base de la ingeniería exitosa y, lo más importante, que guían el apoyo para el software.

Mito: La ingeniería de software hará que generemos documentación voluminosa e innecesaria, e invariablemente nos retrasará.

Realidad: La ingeniería de software no consiste en producir documentos. Se trata de crear un producto de calidad. La mejor calidad conduce a menos repeticiones, lo que da como resultado tiempos de entrega más cortos.

Muchos profesionales del software reconocen la falacia de los mitos mencionados. Es lamentable que las actitudes y métodos habituales nutran la administración y las prácticas técnicas deficientes, aun cuando la realidad dicta un enfoque mejor. El primer paso hacia la formulación de soluciones prácticas para la ingeniería de software es el reconocimiento de las realidades en este campo.