CAPÍTULO

1

EL SOFTWARE Y LA INGENIERÍA DE SOFTWARE

Tenía la apariencia clásica de un alto ejecutivo de una compañía importante de software —a la mitad de los 40, con las sienes comenzando a encanecer, esbelto y atlético, con ojos que penetraban al observador mientras hablaba—. Pero lo que dijo me dejó anonadado. "El software ha *muerto*".

Pestañeé con sorpresa y sonreí. "Bromeas, ¿verdad? El mundo es dirigido con software y tu empresa se ha beneficiado mucho de ello. ¡No ha muerto! Está vivo y en desarrollo."

Movió su cabeza de manera enfática. "No, está muerto... al menos como lo conocimos."

Me apoyé en el escritorio. "Continúa."

Habló al tiempo que golpeaba en la mesa con énfasis. "El concepto antiguo del software —lo compras, lo posees y tu trabajo consiste en administrarlo— está llegando a su fin. Hoy día, con Web 2.0 y la computación ubicua cada vez más fuerte, vamos a ver una generación de software por completo diferente. Se distribuirá por internet y se verá exactamente como si estuviera instalado en el equipo de cómputo de cada usuario... pero se encontrará en un servidor remoto."

Tuve que estar de acuerdo. "Entonces, tu vida será mucho más sencilla. Tus muchachos no tendrán que preocuparse por las cinco diferentes versiones de la misma App que utilizan decenas de miles de usuarios."

Sonrió. "Absolutamente. Sólo la versión más reciente estará en nuestros servidores. Cuando hagamos un cambio o corrección, actualizaremos funcionalidad y contenido a cada usuario. Todos lo tendrán en forma instantánea..."

Hice una mueca. "Pero si cometes un error, todos lo tendrán también instantáneamente".

Él se rió entre dientes. "Es verdad, por eso estamos redoblando nuestros esfuerzos para hacer una ingeniería de software aún mejor. El problema es que tenemos que hacerlo 'rápido' porque el mercado se ha acelerado en cada área de aplicación."

UNA MIRADA RÁPIDA

¿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 de 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 porque 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? Lea el resto de este libro, seleccione aquellas ideas que sean aplicables al software que usted hace y aplíquelas a su trabajo.

Me recargué en la espalda y coloqué mis manos en mi nuca. "Ya sabes lo que se dice... puedes tenerlo rápido o bien hecho o barato. Escoge dos de estas características..."

"Elijo rápido y bien hecho", dijo mientras comenzaba a levantarse.

También me incorporé. "Entonces realmente necesitas ingeniería de software."

"Ya lo sé", dijo mientras salía. "El problema es que tenemos que llegar a convencer a otra generación más de técnicos de que así es..."

¿Está muerto *realmente* el software? Si lo estuviera, usted no estaría leyendo este libro...

Cita:

"Las ideas y los descubrimientos tecnológicos son los motores que impulsan el crecimiento económico."

Wall Street Journal

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).

Nadie pudo prever que habría software incrustado en sistemas de toda clase: de transporte, médicos, de telecomunicaciones, militares, industriales, de entretenimiento, en máquinas de oficina... la lista es casi infinita. Y si usted cree en la ley de las consecuencias inesperadas, hay muchos efectos que aún no podemos predecir.

Nadie podía anticipar que millones de programas de computadora tendrían que ser corregidos, adaptados y mejorados a medida que transcurriera el tiempo. Ni que la carga de ejecutar estas actividades de "mantenimiento" absorbería más personas y recursos que todo el trabajo aplicado a la creación de software nuevo.

Conforme ha aumentado la importancia del software, la comunidad de programadores ha tratado continuamente de desarrollar tecnologías que hagan más fácil, rápida y barata la elaboración de programas de cómputo de alta calidad. Algunas de estas tecnologías se dirigen a un dominio específico de aplicaciones (por ejemplo, diseño e implantación de un sitio web), otras se centran en un dominio tecnológico (sistemas orientados a objetos o programación orientada a aspectos), otros más tienen una base amplia (sistemas operativos, como Linux). Sin embargo, todavía falta por desarrollarse una tecnología de software que haga todo esto, y hay pocas probabilidades de que surja una en el futuro. A pesar de ello, las personas basan sus trabajos, confort, seguridad, diversiones, decisiones y sus propias vidas en software de computadora. Más vale que esté bien hecho.

Este libro presenta una estructura que puede ser utilizada por aquellos que hacen software de cómputo —personas que deben hacerlo bien—. La estructura incluye un proceso, un conjunto de métodos y unas herramientas que llamamos *ingeniería de software*.

1.1 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

PUNTO CLAVE

El software es tanto un producto como un vehículo para entregar un producto.

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.

Cita:

"El software es un lugar donde se siembran sueños y se cosechan pesadillas, una ciénega abstracta y mística en la que terribles demonios luchan contra panaceas mágicas, un mundo de hombres lobo y balas de plata."

Brad J. Cox

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:[1]

- ¿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.

1.1.1 Definición de software

En la actualidad, la mayoría de 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:

? ¿Cómo se define *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 progra-

1 En un excelente libro de ensayos sobre el negocio del software, Tom DeMarco [DeM95] defiende el punto de vista contrario. Dice: "En lugar de preguntar por qué el software cuesta tanto, necesitamos comenzar a preguntar: ¿Qué hemos hecho para hacer posible que el software actual cueste tan poco? La respuesta a esa pregunta nos ayudará a continuar el extraordinario nivel de logro que siempre ha distinguido a la industria del software."

mas 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.

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.*

PUNTO CLAVE
El software se modifica con intelecto, no se manufactura.

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 (véase el capítulo 24). 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".*

PUNTO CLAVE
El software no se desgasta, pero sí se deteriora.

La figura 1.1 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*.

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 1.2. Los defectos ocultos ocasionarán ta-

FIGURA 1.1
Curva de fallas del hardware

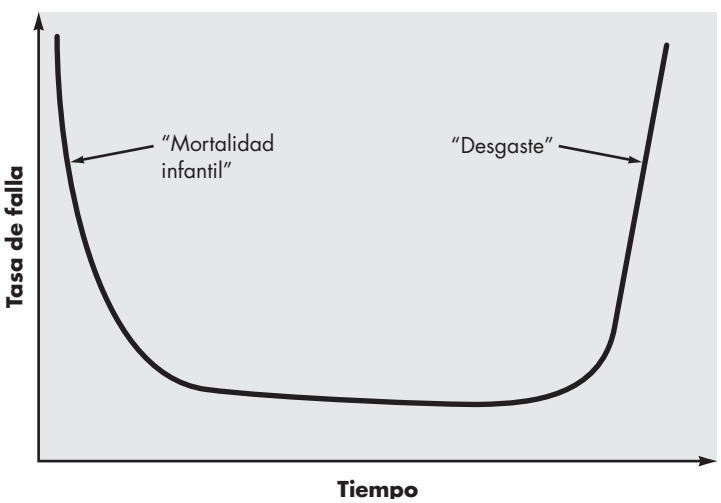
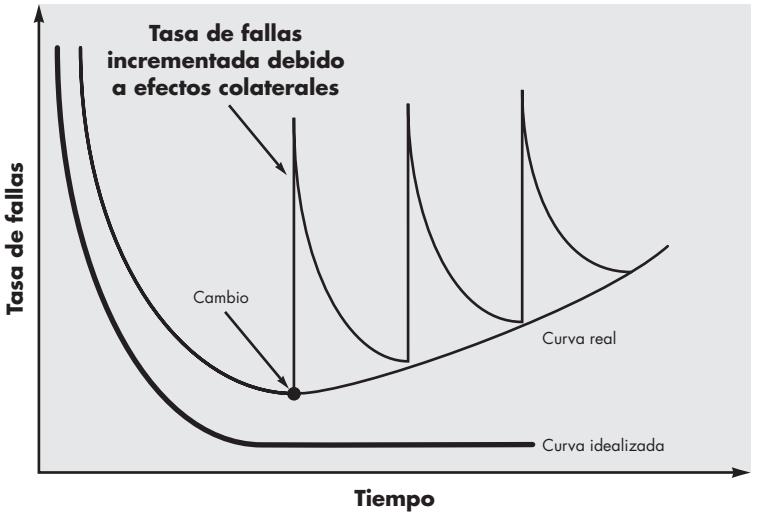


FIGURA 1.2
Curvas de falla del software



CONSEJO

Si quiere reducir el deterioro del software, tendrá que mejorar su diseño (capítulos 8 a 13).

PUNTO CLAVE

Los métodos de la ingeniería de software llevan a reducir la magnitud de los picos y de la pendiente de la curva real en la figura 1.2.

sas 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 1.2. Durante su vida,[2] 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 1.2). 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.

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.

Cita:

"Las ideas son los ladrillos con los que se construyen las ideas."

Jason Zebehazy

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

2 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 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.[3] 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.

1.1.2 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 (por ejemplo, compiladores, editores y herramientas para administrar archivos) procesa estructuras de información complejas pero deterministas.[4] 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.

WebRef

En la dirección **shareware.cnet.com** se encuentra una de las librerías más completas de software compartido y libre.

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 "devoradores de números". 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

3 El desarrollo basado en componentes se estudia en el capítulo 10.

4 El software es *determinista* si es posible predecir el orden y momento de las entradas, el procesamiento y las salidas. 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: es diseñado para proporcionar una capacidad específica para uso de muchos consumidores diferentes. 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 2.0, 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.

Cita:

"No hay computadora que tenga sentido común."

Marvin Minsky

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. No es raro que una joven ingeniera de software trabaje en un programa de mayor edad que la de ella… Las generaciones pasadas de los trabajadores del software dejaron un legado en cada una de las categorías mencionadas. Por fortuna, la herencia que dejará la actual generación aligerará la carga de los futuros ingenieros de software. Aun así, nuevos desafíos (capítulo 31) han aparecido en el horizonte.

Computación en un mundo abierto: el rápido 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 autodescriptivo, 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.

Cita:

"No siempre puedes predecir, pero siempre puedes prepararte."

Anónimo

Es indudable que cada uno de estos nuevos retos obedecerá a la ley de las consecuencias imprevistas y tendrá efectos (para hombres de negocios, ingenieros de software y usuarios finales) que hoy no pueden predecirse. Sin embargo, los ingenieros de software pueden prepararse de-

sarrollando un proceso que sea ágil y suficientemente adaptable para que acepte los cambios profundos en la tecnología y las reglas de los negocios que seguramente surgirán en la década siguiente.

1.1.3 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 para ciertos individuos, industria y gobierno. Pero otros programas son más viejos, en ciertos casos *muy* viejos.

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 manera siguiente:

> 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 causa de 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.

¿Qué hago si encuentro un sistema heredado de mala calidad?

Desafortunadamente, en ocasiones hay otra característica presente en el software heredado: *mala calidad*.[5] Hay veces en las que los sistemas heredados 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. A pesar de esto, dichos sistemas dan apoyo a las "funciones básicas del negocio y son indispensables para éste". ¿Qué hacer?

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:

¿Qué tipos de cambios se hacen a los sistemas heredados?

- 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.

Todo ingeniero de software debe reconocer que el cambio es natural. No trate de evitarlo.

Cuando ocurren estos modos de evolución, debe hacerse la reingeniería del sistema heredado (capítulo 29) 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 se

5 En este caso, la calidad se juzga con base en el pensamiento moderno de la ingeniería de software, criterio algo injusto, ya que algunos conceptos y principios de la ingeniería de software moderna tal vez no hayan sido bien entendidos en la época en que se desarrolló el software heredado.

desarrollan a partir de los antiguos y [...] que todo debe operar entre sí y cooperar con cada uno de los demás" [Day99].

1.2 LA NATURALEZA ÚNICA DE LAS *WEBAPPS*

Cita:

"Cuando veamos cualquier tipo de estabilización, la web se habrá convertido en algo completamente diferente."

Louis Monier

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.

Como se dijo en la sección 1.1.2, las *webapps* son una de varias categorías distintas de software. No obstante, podría argumentarse que las *webapps* son diferentes. Powell [Pow98] sugiere que los sistemas y aplicaciones basados en web "involucran una mezcla entre las publicaciones impresas 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:

¿Qué característica diferencia las *webapps* de otro software?

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).

---

6 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 continua. 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.[7]

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.

Podría argumentarse que otras categorías de aplicaciones estudiadas en la sección 1.1.2 muestran algunos de los atributos mencionados. Sin embargo, las *webapps* casi siempre poseen todos ellos.

1.3 INGENIERÍA DE SOFTWARE

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

PUNTO CLAVE
Entender el problema antes de dar una solución.

- 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[8] ha crecido en forma notable. Cuando ha de construirse una aplicación nueva o sistema incrustado, deben escucharse muchas opiniones. Y en ocasiones parece que cada una de ellas tiene una idea un poco distinta de cuáles características y funciones debiera tener el software. *Se concluye que debe hacerse un esfuerzo concertado para entender el problema antes de desarrollar una aplicación de software.*

PUNTO CLAVE
El diseño es una actividad crucial de la ingeniería 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 autocontenido, 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.*

7 Con las herramientas modernas es posible producir páginas web sofisticadas en unas cuantas horas.
8 En una parte posterior de este libro, llamaré a estas personas "participantes".

PUNTO CLAVE

Tanto la calidad como la facilidad de recibir mantenimiento son resultado de un buen diseño.

- 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.* Y esto conduce al tema de este libro: *la ingeniería de software*.

Cita:

"Más que una disciplina o cuerpo de conocimientos, ingeniería es un verbo, una palabra de acción, una forma de abordar un problema."

Scott Whitmir

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.

El lector se sentirá tentado de ampliar esta definición.[9] Dice poco sobre los aspectos técnicos de la calidad del software; no habla directamente de la necesidad de satisfacer a los consumidores ni de entregar el producto a tiempo; omite mencionar la importancia de la medición y la metrología; no establece la importancia de un proceso eficaz. No obstante, la definición de 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.

? ¿Cómo se define la ingeniería de software?

El IEEE [IEEE93a] 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. 2) El estudio de enfoques según el punto 1.

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.

PUNTO CLAVE

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

La ingeniería de software es una tecnología con varias capas. Como se aprecia en la figura 1.3, 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 similares[10] alimentan la cultura de mejora continua, y es esta cultura la que lleva en última instancia al desarrollo de enfoques cada vez más eficaces de la ingeniería de software. El fundamento en el que se apoya la ingeniería de software es el compromiso con la calidad.

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 y

9 Consulte muchas otras definiciones en www.answers.com/topic/software-engineering#wp-_note-13.

10 En el capítulo 14 y toda la parte 3 del libro se estudia la administración de la calidad y los enfoques relacionados con ésta.

FIGURA 1.3
Capas de la ingeniería de software

WebRef
CrossTalk es un periódico que da información práctica sobre procesos, métodos y herramientas. Se encuentra en **www.stsc.hill.af.mil**

oportuno del software de cómputo. 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.

1.4 EL PROCESO DEL SOFTWARE

? ¿Cuáles son los elementos de un proceso de software?

Cita:
"Un proceso define quién hace *qué*, *cuándo* y *cómo*, para alcanzar cierto objetivo."
Ivar Jacobson, Grady Booch y James Rumbaugh

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:

¿Cuáles son las cinco actividades estructurales del proceso?

Comunicación. Antes de que comience cualquier trabajo técnico, tiene importancia crítica comunicarse y colaborar con el cliente (y con otros participantes).[11] 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. 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.

Cita:

"Einstein afirmaba que debía haber una explicación sencilla de la naturaleza porque Dios no es caprichoso o arbitrario. Al ingeniero de software no lo conforta una fe parecida. Gran parte de la complejidad que debe doblegar es de origen arbitrario."

Fred Brooks

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.

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:

PUNTO CLAVE

Las actividades sombrilla ocurren a lo largo del proceso de software y se centran sobre todo en la administración, el seguimiento y el control del proyecto.

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.

11 Un *participante* es cualquier persona que tenga algo que ver en el resultado exitoso del proyecto —gerentes del negocio, usuarios finales, ingenieros de software, personal de apoyo, etc.—. Rob Thomset dice en broma que "un participante es una persona que blande una estaca grande y aguda [...] Si no vez más lejos que los participantes, ya sabes dónde terminará la estaca". (N. del T.: Esta nota es un juego de palabras: *stake* significa estaca y también *parte*, y *stakeholder* es el que blande una estaca, pero también un *participante*.)

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.

Cada una de estas actividades sombrilla se analiza en detalle más adelante.

PUNTO CLAVE
La adaptación del proceso de software es esencial para el éxito del proyecto.

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:

? ¿Qué diferencias existen entre los modelos del proceso?

- 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

Cita:

"Siento que una receta es sólo un tema que una cocinera inteligente ejecuta con una variación en cada ocasión."

Madame Benoit

En la parte 1 de este libro, se examinará el proceso de software con mucho detalle. Los *modelos de proceso prescriptivo* (capítulo 2) enfatizan la definición, la identificación y la aplicación detalladas de las actividades y tareas del proceso. 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.

? ¿Qué caracteriza a un proceso "ágil"?

Los *modelos de proceso ágil* (capítulo 3) 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.

1.5 LA PRÁCTICA DE LA INGENIERÍA DE SOFTWARE

WebRef

En la dirección **www.literateprogramming.com** se encuentran varias citas provocativas sobre la práctica de la ingeniería de software.

En la sección 1.4 se introdujo un modelo general de proceso de software compuesto de un conjunto de actividades que establecen una estructura para la práctica de la ingeniería de software. Las actividades estructurales generales —**comunicación, planeación, modelado, construcción** y **despliegue**— y las actividades sombrilla establecen el esqueleto de la arquitectura para el trabajo de ingeniería de software. Pero, ¿cómo entra aquí la práctica de la ingeniería de software? En las secciones que siguen, el lector obtendrá la comprensión básica de los conceptos y principios generales que se aplican a las actividades estructurales.[12]

1.5.1 La esencia de la práctica

CONSEJO

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.

En un libro clásico, *How to Solve It*, escrito antes de que existieran las computadoras modernas, George Polya [Pol45] describió la esencia de la solución de problemas y, en consecuencia, la esencia de 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 [adaptado de Pol45]:

Entender el problema. En ocasiones es difícil de admitir, pero la mayor parte de nosotros adoptamos una actitud de orgullo desmedido cuando se nos presenta un problema. Escuchamos por unos segundos y después pensamos: *Claro, sí, entiendo, resolvamos esto.* 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?

Cita:

"En la solución de cualquier problema hay un grano de descubrimiento."

George Polya

Planear la solución. Ahora entiende el problema (o es lo que piensa) y no puede esperar para escribir el código. Antes de hacerlo, cálmese un poco y haga un pequeño diseño:

- *¿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?

12 El lector debería volver a consultar las secciones de este capítulo a medida que en el libro se describan en específico los métodos y las actividades sombrilla de la ingeniería de 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. No se puede estar seguro de que la solución sea perfecta, pero sí de que se ha diseñado un número suficiente de pruebas para descubrir tantos errores como sea posible.

- *¿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.

1.5.2 Principios generales

El diccionario define la palabra *principio* como "una ley importante o suposición que subyace y se requiere en un sistema de pensamiento". En este libro se analizarán principios en muchos niveles distintos de abstracción. Algunos se centran en la ingeniería de software como un todo, otros consideran una actividad estructural general específica (por ejemplo, **comunicación**), y otros más se centran en acciones de la ingeniería de software (por ejemplo, diseño de la arquitectura) o en tareas técnicas (escribir un escenario para el uso). Sin importar su nivel de enfoque, los principios lo ayudarán a establecer un conjunto de herramientas mentales para una práctica sólida de la ingeniería de software. Ésa es la razón de que sean importantes.

David Hooker [Hoo96] propuso siete principios que se centran en la práctica de la ingeniería de software como un todo. Se reproducen en los párrafos siguientes:[13]

Antes de comenzar un proyecto de software, asegúrese de que el software tenga un propósito para el negocio y que los usuarios perciben valor en él.

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.*

13 Reproducido con permiso del autor [Hoo96]. Hooker define algunos patrones para estos principios en http://c2.com/cgi/wiki?SevenPrinciplesOfSoftwareDevelopment.

Cita:

"Hay cierta majestad en la sencillez, que es con mucho todo lo que adorna al ingenio."

Papa Alejandro (1688-1744)

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*

PUNTO CLAVE

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

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 [o la

14 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.

15 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.

1.6 MITOS DEL SOFTWARE

Cita:

"En ausencia de estándares significativos, una industria nueva como la del software depende sólo del folklore."

Tom DeMarco

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.

WebRef

La Software Project Managers Network (Red de Gerentes de Proyectos de Software), en **www.spmn.com**, lo ayuda a eliminar éstos y otros mitos.

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. ¿No le dará a mi personal todo lo que necesita saber?*

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 [Bro95]: "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.

Trabaje muy duro para entender qué es lo que tiene que hacer antes de empezar. Quizás no pueda desarrollarlo a detalle, pero entre más sepa, menor será el riesgo que tome.

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.

Siempre que piense que no hay tiempo para la ingeniería de software, pregúntese: "¿tendremos tiempo de hacerlo otra vez?".

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.*

16 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. Los métodos ágiles se estudian en el capítulo 3.

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 (descritas en el capítulo 15) 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.

1.7 CÓMO COMIENZA TODO

Todo proyecto de software se desencadena por alguna necesidad de negocios: la de corregir un defecto en una aplicación existente, la de adaptar un "sistema heredado" a un ambiente de negocios cambiante, la de ampliar las funciones y características de una aplicación ya existente o la necesidad de crear un producto, servicio o sistema nuevo.

Al comenzar un proyecto de software, es frecuente que las necesidades del negocio se expresen de manera informal como parte de una simple conversación. La plática que se presenta en el recuadro que sigue es muy común.

CASASEGURA[17]

Cómo se inicia un proyecto

La escena: Sala de juntas en CPI Corporation, empresa (ficticia) que manufactura productos de consumo para uso doméstico y comercial.

Participantes: Mal Golden, alto directivo de desarrollo de productos; Lisa Pérez, gerente comercial; Lee Warren, gerente de ingeniería; Joe Camalleri, VP ejecutivo, desarrollo de negocios.

La conversación:

Joe: Oye, Lee, ¿qué es eso que oí acerca de que tu gente va a desarrollar no sé qué? ¿Una caja inalámbrica universal general?

Lee: Es sensacional… más o menos del tamaño de una caja de cerillos pequeña… podemos conectarla a sensores de todo tipo, una cámara digital… a cualquier cosa. Usa el protocolo 802.11g inalámbrico. Permite el acceso a la salida de dispositivos sin cables. Pensamos que llevará a toda una nueva generación de productos.

Joe: ¿Estás de acuerdo, Mal?

Mal: Sí. En realidad, con las ventas tan planas que hemos tenido este año necesitamos algo nuevo. Lisa y yo hemos hecho algo de investigación del mercado y pensamos que tenemos una línea de productos que podría ser algo grande.

17 El proyecto *CasaSegura* se usará en todo el libro para ilustrar los entretelones de un equipo de proyecto que elabora un producto de software. La compañía, el proyecto y las personas son ficticias, pero las situaciones y problemas son reales.

Joe: ¿Cuán grande… tanto como el renglón de utilidades?

Mal (que evita el compromiso directo): Cuéntale nuestra idea, Lisa.

Lisa: Es toda una nueva generación que hemos llamado "productos para la administración del hogar". Le dimos el nombre de *CasaSegura*. Usan la nueva interfaz inalámbrica, proporcionan a los dueños de viviendas o pequeños negocios un sistema controlado por su PC —seguridad del hogar, vigilancia, control de aparatos y equipos—, tú sabes, apaga el aire acondicionado cuando sales de casa, esa clase de cosas.

Lee (dando un brinco): La oficina de ingeniería hizo un estudio de factibilidad técnica de esta idea, Joe. Es algo realizable con un costo bajo de manufactura. La mayor parte del hardware es de línea. Queda pendiente el software, pero no es algo que no podamos hacer.

Joe: Interesante. Pero pregunté sobre las utilidades.

Mal: Las PC han penetrado a 70 por ciento de los hogares de Estados Unidos. Si lo vendemos en el precio correcto, podría ser una aplicación sensacional. Nadie tiene nuestra caja inalámbrica… somos dueños. Nos adelantaremos dos años a la competencia. ¿Las ganancias? Quizá tanto como 30 a 40 millones de dólares en el segundo año.

Joe (sonriente): Llevemos esto al siguiente nivel. Estoy interesado.

Con excepción de una referencia casual, el software no se mencionó en la conversación. Y, sin embargo, es lo que hará triunfar o fracasar la línea de productos *CasaSegura*. El esfuerzo de ingeniería tendrá éxito sólo si también lo tiene el software de *CasaSegura*. El mercado aceptará el producto sólo si el software incrustado en éste satisface las necesidades del cliente (aún no establecidas). En muchos de los capítulos siguientes continuaremos el avance de la ingeniería del software en *CasaSegura*.

1.8 Resumen

El software es un elemento clave en la evolución de sistemas y productos basados en computadoras, y una de las tecnologías más importantes en todo el mundo. En los últimos 50 años, el software ha pasado de ser la solución de un problema especializado y herramienta de análisis de la información a una industria en sí misma. No obstante, aún hay problemas para desarrollar software de alta calidad a tiempo y dentro del presupuesto asignado.

El software —programas, datos e información descriptiva— se dirige a una gama amplia de tecnología y campos de aplicación. El software heredado sigue planteando retos especiales a quienes deben darle mantenimiento.

Los sistemas y aplicaciones basados en web han evolucionado de simples conjuntos de contenido de información a sistemas sofisticados que presentan una funcionalidad compleja y contenido en multimedios. Aunque dichas *webapps* tienen características y requerimientos únicos, son software.

La ingeniería de software incluye procesos, métodos y herramientas que permiten elaborar a tiempo y con calidad sistemas complejos basados en computadoras. El proceso de software incorpora cinco actividades estructurales: comunicación, planeación, modelado, construcción y despliegue que son aplicables a todos los proyectos de software. La práctica de la ingeniería de software es una actividad para resolver problemas, que sigue un conjunto de principios fundamentales.

Muchos mitos del software todavía hacen que administradores y trabajadores se equivoquen, aun cuando ha aumentado nuestro conocimiento colectivo del software y las tecnologías requeridas para elaborarlo. Conforme el lector aprenda más sobre ingeniería de software, comenzará a entender por qué deben rebatirse estos mitos cada vez que surjan.

Problemas y puntos por evaluar

1.1. Dé al menos cinco ejemplos de la forma en que se aplica la ley de las consecuencias imprevistas al software de cómputo.

1.2. Diga algunos ejemplos (tanto positivos como negativos) que indiquen el efecto del software en nuestra sociedad.

1.3. Desarrolle sus propias respuestas a las cinco preguntas planteadas al principio de la sección 1.1. Analícelas con sus compañeros estudiantes.

1.4. Muchas aplicaciones modernas cambian con frecuencia, antes de que se presenten al usuario final y después de que la primera versión ha entrado en uso. Sugiera algunos modos de elaborar software para detener el deterioro que produce el cambio.

1.5. Considere las siete categorías de software presentadas en la sección 1.1.2. ¿Piensa que puede aplicarse a cada una el mismo enfoque de ingeniería de software? Explique su respuesta.

1.6. La figura 1.3 muestra las tres capas de la ingeniería de software arriba de otra llamada "compromiso con la calidad". Esto implica un programa de calidad organizacional como el enfoque de la administración total de la calidad. Haga un poco de investigación y desarrolle los lineamientos de los elementos clave de un programa para la administración de la calidad.

1.7. ¿Es aplicable la ingeniería de software cuando se elaboran *webapps*? Si es así, ¿cómo puede modificarse para que asimile las características únicas de éstas?

1.8. A medida que el software gana ubicuidad, los riesgos para el público (debidos a programas defectuosos) se convierten en motivo de preocupación significativa. Desarrolle un escenario catastrófico pero realista en el que la falla de un programa de cómputo pudiera ocasionar un gran daño (económico o humano).

1.9. Describa con sus propias palabras una estructura de proceso. Cuando se dice que las actividades estructurales son aplicables a todos los proyectos, ¿significa que se realizan las mismas tareas en todos los proyectos sin que importe su tamaño y complejidad? Explique su respuesta.

1.10. Las actividades sombrilla ocurren a través de todo el proceso del software. ¿Piensa usted que son aplicables por igual a través del proceso, o que algunas se concentran en una o más actividades estructurales?

1.11. Agregue dos mitos adicionales a la lista presentada en la sección 1.6. También diga la realidad que acompaña al mito.

Lecturas adicionales y fuentes de información[18]

Hay literalmente miles de libros escritos sobre software de cómputo. La gran mayoría analiza lenguajes de programación o aplicaciones de software, pero algunos estudian al software en sí mismo. Pressman y Herron (*Software Shock*, Dorset House, 1991) presentaron un estudio temprano (dirigido a las personas comunes) sobre el software y la forma en la que lo elaboran los profesionales. El libro de Negroponte que se convirtió en un éxito de ventas (*Being Digital*, Alfred A. Knopf, Inc., 1995) describe el panorama de la computación y su efecto general en el siglo XXI. DeMarco (*Why Does Software Cost So Much?*, Dorset House, 1995) ha producido varios ensayos amenos y profundos sobre el software y el proceso con el que se elabora.

Minasi (*The Software Conspiracy: Why Software Companies Put out Faulty Products, How They Can Hurt You, and What You Can Do*, McGraw-Hill, 2000) afirma que el "flagelo moderno" de los errores en el software puede eliminarse y sugiere formas de lograrlo. Compaine (*Digital Divide: Facing A Crisis or Creating a Myth*, MIT Press, 2001) asegura que la "división" entre aquellos que tienen acceso a recursos de la información (por ejemplo, la web) y los que no lo tienen se está estrechando conforme avanzamos en la primera década de este siglo. Los libros escritos por Greenfield (*Everyware: The Dawning Age of Ubiquitous Computing*, New Riders Publishing, 2006) y Loke (*Context-Aware Pervasive Systems: Architectures for a New Breed of Applications*, Auerbach, 2006) introducen el concepto de software de "mundo abierto" y predicen un ambiente inalámbrico en el que el software deba adaptarse a los requerimientos que surjan en tiempo real.

---

18 La sección de "Lecturas adicionales y fuentes de información" que se presenta al final de cada capítulo expone un panorama breve de fuentes impresas que ayudan a aumentar la comprensión de los principales temas presentados. El autor ha creado un sitio web para apoyar al libro *Ingeniería de software: enfoque del profesional* en www.mhhe.com/compsci/pressman. Entre los muchos temas que se abordan en dicho sitio, se encuentran desde los recursos de la ingeniería de software capítulo por capítulo hasta información basada en web que complementa el material presentado. Como parte de esos recursos se halla un vínculo hacia Amazon.com para cada libro citado en esta sección.

El estado actual de la ingeniería y del proceso de software se determina mejor a partir de publicaciones tales como *IEEE Software, IEEE Computer, CrossTalk* y *IEEE Transactions on Software Engineering*. Publicaciones periódicas como *Application Development Trends* y *Cutter IT Journal* con frecuencia contienen artículos sobre temas de ingeniería de software. La disciplina se "resume" cada año en *Proceeding of the International Conference on Software Engineering*, patrocinada por IEEE y ACM, y se analiza a profundidad en revistas tales como *ACM Transactions on Software Engineering and Methodology, ACM Software Engineering Notes* y *Annals of Software Engineering*. Hay decenas de miles de sitios web dedicados a la ingeniería y al proceso de software.

En años recientes se han publicado muchos libros que abordan el proceso y la ingeniería de software. Algunos presentan un panorama de todo el proceso, mientras otros profundizan en algunos temas importantes y omiten otros. Entre los más populares (¡además del que tiene usted en sus manos!) se encuentran los siguientes:

Abran, A., and J. Moore, *SWEBOK: Guide to the Software Engineering Body of Knowledge,* IEEE, 2002.

Andersson, E., et al., *Software Engineering for Internet Applications,* The MIT Press, 2006.

Christensen, M., and R. Thayer, *A Project Manager's Guide to Software Engineering Best Practices,* IEEE-CS Press (Wiley), 2002.

Glass, R., *Fact and Fallacies of Software Engineering,* Addison-Wesley, 2002.

Jacobson, I., *Object-Oriented Software Engineering: A Use Case Driven Approach,* 2d ed., Addison-Wesley, 2008.

Jalote, P., *An Integrated Approach to Software Engineering,* Springer, 2006.

Pfleeger, S., *Software Engineering: Theory and Practice,* 3d ed., Prentice-Hall, 2005.

Schach, S., *Object-Oriented and Classical Software Engineering,* 7th ed., McGraw-Hill, 2006.

Sommerville, I., *Software Engineering,* 8th ed., Addison-Wesley, 2006.

Tsui, F., and O. Karam, *Essentials of Software Engineering,* Jones & Bartlett Publishers, 2006.

En las últimas décadas, son muchos los estándares para la ingeniería de software que han sido publicados por IEEE, ISO y sus organizaciones. Moore (*The Road Map to Software Engineering: A Standards-Based Guide*, Wiley-IEEE Computer Society Press, 2006) proporciona una revisión útil de los estándares relevantes y la forma en la que se aplican a proyectos reales.

En internet se encuentra disponible una amplia variedad de fuentes acerca de la ingeniería y el proceso de software. Una lista actualizada de referencias en la Red Mundial que son útiles para el proceso de software se encuentra en el sitio web del libro, en la dirección **www.mhhe.com/engcs/compsci/pressman/professional/olc/ser.htm**.