

Calidad y Testeo del Software

Mientras en los grandes centros tecnológicos del mundo es una prioridad desde hace varios años, el concepto de calidad en el software es prácticamente desconocido por un número importante de empresas españolas. Aprender a hacer bien las cosas lleva tiempo, pero es una necesidad si se quiere desarrollar software correcto de forma eficiente. Un proceso de software dirigido por estándares de calidad, soportado por herramientas integradas de gestión automática, integrado con un buen proceso de testeo y realizado por personal capacitado garantiza la construcción de productos consistentes con los requisitos de clientes cumpliendo restricciones de tiempo y presupuesto. Este artículo presenta el concepto de calidad sobre un conjunto de procesos interrelacionados de ingeniería y gestión del software que cooperan dentro del ciclo de vida de un software para construir un producto de software de calidad. El ITI está actualmente involucrado en la definición de metodologías propias de evaluación de la calidad de procesos y productos de software, entre ellos, de métodos de testeo de software para certificar la calidad de los productos finales.

Introducción

El desarrollo actual de software continúa siendo muy propenso a errores. Un gran número de proyectos termina con grandes retrasos, excediendo sustancialmente presupuestos y recursos planificados. Es frecuente encontrar a desarrolladores trabajando desorganizadamente bajo fuertes condiciones de estrés, dentro de un proceso de software con pobre o nula calidad.

Según un informe reciente del Instituto de Estudios Laborales titulado "Nuevas formas de organización del trabajo y productividad: la visión de la Comisión Europea" (Computerworld, num. 952, pág. 27, diciembre de 2002), España tiene una productividad de un 23% y un 8% inferior a la de los EEUU y la media de Europea, respectivamente. Entre otras causas, el informe destaca la no existencia de modelos de colaboración (en favor del modelo de confrontación), la ausencia de una cultura organizativa centrada en las personas, la falta de confianza en el capital humano, y la no orientación de los funcionamientos empresariales hacia la calidad.

Mientras empresas informáticas de Estados Unidos, Japón, India y parte de Europa llevan años consolidando la calidad en el software como el único camino para desarrollar software correcto en tiempos y presupuestos competitivos, el concepto de "calidad en el software" en España es aún prácticamente desconocido por un número importante de empresas. Es usual la institucionalización de malas prácticas y una carencia notable de profesionales con formación adecuada para cambiar esta situación. Estas son las conclusiones de otro estudio reciente (Computerworld, pág. 2, septiembre de 2001) cuyos resultados señalan que aproximadamente el 30% de las empresas de software españolas no siguen ningún procedimiento de calidad, un 60% basan sus desarrollos en el modelo ISO 9001, y sólo un 3% basa su funcionamiento en las metodologías del CMM. A este último modelo (CMM) se le reconoce con respecto al primero (ISO 9001) mayor rigor y especificidad en la definición de los procesos internos del desarrollo de software, así como el ser una metodología de mejora progresiva. Por otra parte, el Consejo Superior de Informática del Ministerio de Administraciones Públicas ha definido una metodología de planificación, desarrollo y mantenimiento de sistemas de información que se llama Métrica, actualmente en su versión 3, que está siendo implantado por múltiples organismos tanto públicos como privados.

El objetivo de este artículo es presentar el concepto de calidad en el software como parte del proceso de ciclo de vida del software, haciendo énfasis en una breve caracterización de este concepto en áreas y técnicas de ingeniería y gestión del software.

Qué es calidad y gestión de la calidad

La Organización Internacional de Estándares (ISO, por sus siglas en inglés), que ejerce un rol importante en uniformar definiciones, ha publicado varios estándares relacionados con calidad en general y, en particular, con calidad en el software. Estándares como ISO 8402, 9000, 14598 definen calidad del software como la capacidad de un conjunto de características de un producto, sistema o proceso para satisfacer requisitos de clientes y otras partes interesadas.

El estándar de gestión de la calidad ISO 9000 es actualmente sinónimo de calidad y de buenas prácticas. La teoría detrás de este estándar es que una organización bien gestionada con un proceso de ingeniería bien definido es más probable que construya productos consistentes con los requisitos del cliente cumpliendo restricciones de tiempo y presupuesto, que organizaciones pobremente gestionadas sin un proceso definido. Dentro de la familia ISO 9000, la norma ISO 9000-3 es específica para desarrollo de software y su mantenimiento. La gestión de la calidad del software dentro de este contexto es definida como todas las acciones coordinadas para dirigir y controlar una organización con respecto a calidad del software.

Gestión de la calidad: 4 pilares del desarrollo

La gestión de la calidad del software actúa sobre 4 pilares que componen el proceso de desarrollo de software:

- procesos de ciclo de vida
- técnicas (¿cómo?)
- organización (¿quién?)
- infraestructura (¿con qué?)

Este artículo se centrará únicamente en los dos primeros pilares, procesos y técnicas, que son los que tienen una vinculación más directa con la calidad del producto final. La organización se basa en las personas, en su formación o especialización, y en cómo se

organizan para desarrollar un proyecto. La infraestructura, por su parte, son las instalaciones, equipamiento, servidores, medios de comunicación, de los que se dispone para el desarrollo de software.

Procesos del ciclo de vida

El ciclo de vida de un software es el período de tiempo que comienza con la concepción de la idea de un software y que termina con la vida útil del mismo. Durante este período de tiempo cooperan un conjunto de procesos interrelacionados, denominados procesos del ciclo de vida, con el objetivo de construir un producto de software de calidad. Los modelos y estándares internacionales como ISO 12207, IEEE 1074 y CMMI identifican procesos que componen el ciclo de vida de un software. Tomando como base estos estándares, a continuación se identifican las siguientes áreas de procesos:

- **Procesos primarios de ingeniería:** son las actividades primarias del ciclo de vida, aquellas incluidas en las disciplinas técnicas. Independientemente del modelo de ciclo de vida seleccionado (e.g. cascada, espiral, V, W), siempre será necesario el análisis de requisitos, diseño, implementación, validación y verificación, y mantenimiento.
- **Procesos de gestión de proyectos:** cubre las actividades de estimación, planificación del proyecto y asignación de recursos, medición del progreso, seguimiento y control del proyecto, gestión de riesgos y gestión de las relaciones con los clientes.
- **Procesos de aseguramiento de la calidad:** son actividades sistemáticas y planificadas, necesarias para dirigir y controlar los procesos del ciclo de vida con el objetivo de proporcionar suficiente confianza de que el proceso y los productos del desarrollo satisfacen aceptablemente estándares de calidad. Estas actividades ejercen, por tanto, una función de watchdog, controlando todos los procesos del ciclo de vida de software.

La siguiente tabla ilustra la organización de procesos en las diferentes áreas:

ÁREAS DE PROCESOS			
PROCESOS	Ingeniería	Gestión	Calidad
	análisis de requisitos	estimación	prevención
	diseño	planificación	detección y corrección
	implementación	medición	evaluación y mejora
	validación y verificación	control y seguimiento	
	mantenimiento	gestión de riesgos	
		relaciones con clientes	

En las secciones que siguen se hace una caracterización breve de cada proceso, haciendo énfasis en aquellos aspectos que definen la calidad del mismo, y que condicionan, en última instancia, la calidad global del ciclo de vida del software.

Procesos primarios de ingeniería

1. Gestión de Requisitos

La Gestión de Requisitos es el proceso de captura de requisitos, su especificación en un formato bien definido, el uso de prácticas de comunicación (prototipos, entrevistas) para refinar la comprensión de lo que quiere el cliente, la revisión periódica de la consistencia entre

requisitos y otros contenidos del desarrollo (diseño, código, manual de usuarios), y la gestión de cambios en los requisitos durante todo el proyecto.

Una gestión insuficiente de requisitos es una de las causas más frecuentes de que los proyectos se retrasen, sobrepasen sus presupuestos o tengan menos funcionalidad de la esperada. El éxito en la gestión de requisitos depende del conocimiento y la aplicación apropiada de diferentes fundamentos, por ejemplo, metodologías de análisis de requisitos, modelos de representación, prácticas de comunicación, metodología de gestión de cambios en los requisitos, técnicas de verificación y validación de la completitud y corrección de los requisitos y de su consistencia con otros productos del software.

Un gestión correcta y completa de requisitos debe permitir su uso como base para estimar, planificar, diseñar, implementar y verificar y validar el software.

2. Diseño

El Diseño es el proceso de definición de la arquitectura del sistema, de las estructuras de datos y de los algoritmos a emplear, antes de realizar la construcción del software. Algunos fundamentos que garantizan diseños robustos son el conocimiento de estilos (estructurado, OO) y conceptos (modularidad, abstracción) básicos de diseño, algoritmos y estructuras de datos primarias, esquemas típicos de arquitecturas, herramientas de diseño, entre otros.

Los ciclos de vida modernos de software prestan especial atención al diseño de arquitectura, cuya solución suele ser una tarea prioritaria. Organizaciones preocupadas por la calidad de su proceso de software documentan soluciones genéricas de diseño en función del dominio de aplicación a resolver, e incluyen experiencias previas de la aplicación de estas soluciones.

3. Implementación

Cuando se llega a la implementación dentro de un proceso correcto de software, la mayoría del trabajo creativo ya ha sido realizado. En este sentido, la implementación se considera una tarea de bajo nivel. Es decir, prácticas pobres de diseño pueden forzar la reescritura de gran parte del sistema, no siendo necesariamente así en el caso de usar prácticas pobres de codificación. Sin embargo, estas malas prácticas pueden provocar errores sutiles cuya detección y corrección puede costar días o semanas. Por lo tanto, una organización que haga de la calidad una prioridad no debe desconocer ciertos fundamentos de construcción del software, por ejemplo, prácticas correctas y uniformes de codificación, directrices para el uso de tipos de datos, reglas para empaquetar código en módulos, clases o ficheros, prácticas de testeo de unidad y de depuración, estrategias de integración, etc.

La estandarización de las prácticas de implementación de un software simplifican notablemente los esfuerzos de trabajo en grupo, en especial, aquellos orientados al mantenimiento del propio software o al reuso de código en futuros proyectos por personas diferentes.

4. Mantenimiento

De acuerdo a IEEE 1219, el mantenimiento de software es el conjunto de actividades de modificación de un producto de software después de entregado, para corregir fallos, mejorar su rendimiento u otros atributos, o adaptar el producto a un entorno modificado.

Una vez comienzan a operar con el sistema, los usuarios pueden encontrar errores y aspectos que quieran mejorar, los mantenedores

realizan los cambios, después de lo cual los usuarios vuelven a usarlos y a proporcionar nueva información de mejora. Este ciclo de mantenimiento extiende la vida del producto de software. En muchos casos, el mantenimiento es el proceso más largo del ciclo de vida.

El mantenimiento de software es difícil de realizar y gestionar. Sin embargo, este proceso se simplifica notablemente si los procesos primarios previos de ingeniería han sido correctamente realizados y documentados.

5. Verificación y Validación

Como proceso de validación y de verificación (V&V) se entiende cualquier actividad orientada a determinar si los objetivos se han cumplido o no. Más específicamente:

- Verificación comprueba la consistencia del software con respecto a especificaciones y requisitos; es decir, responde a ¿se ha construido correctamente el software?
- Validación comprueba si lo que se ha especificado (e implementado) es lo que el usuario realmente desea; es decir, responde a ¿se ha construido el software correcto?

Las tareas de V&V no solo se aplican a productos de software, sino también a otros productos resultantes del proceso del desarrollo. Las primeras tareas de V&V al análisis y a la especificación de requisitos, por ejemplo, comprobando que el proyecto es viable, que las especificaciones documentadas son completas, correctas, precisas, legibles, evaluables, y que, en general, responden a las expectativas del cliente. La V&V del diseño debe garantizar que los requisitos no están incompletos o incorrectamente diseñados. En el caso de la implementación y codificación, la V&V de software es comúnmente conocida como testeo de software.

Existen muchas definiciones incorrectas del testeo de software que conducen a una inadecuada aplicación de este proceso, por ejemplo, “el testeo demuestra que no hay errores”, o “el testeo demuestra que un programa funciona correctamente”. Según Edsgar Dijkstra “el testeo puede demostrar la presencia de errores, no su ausencia”. Por lo tanto, se realiza test al software para detectar errores que, una vez corregidos, mejoran la calidad o fiabilidad del mismo. Existen distintos tipos de testeo en función de la unidad de software a la que se aplique y del objetivo que se persigue, por ejemplo, el testeo de unidad, de integración, de sistema y de aceptación.

Finalmente, las actividades de V&V son también necesarias durante la operación y el mantenimiento del software. Cuando se realiza un cambio en el software, se debe examinar el impacto del cambio sobre el sistema y considerar qué actividades de V&V es necesario repetir para garantizar, al menos, la misma calidad en el software antes del cambio.

Procesos primarios de gestión

Los fundamentos de gestión consisten en estimar el tamaño del proyecto de software a desarrollar y los recursos (tiempo, personas, medios) necesarios para su construcción, definir y gestionar riesgos e incertidumbres asociados al desarrollo, planificar el proceso de desarrollo asignando recursos a las tareas en función de las estimaciones y riesgos analizados, y finalmente controlar y dar seguimiento al progreso del plan y al uso de los recursos planificados. En proyectos complejos con riesgos importantes, es frecuente realizar re-estimaciones y refinar planificaciones según se va avanzando en el proyecto y se van aclarando incertidumbres iniciales.

1. Estimación

El proceso de estimación puede definirse a partir de tres pasos básicos: primero, estimar el tamaño del proyecto a partir de un análisis preliminar de requisitos; luego estimar el esfuerzo total (en unidades de tiempo) que requiere el desarrollo de un proyecto de tal tamaño; por último, estimar el tiempo de desarrollo del proyecto en función del esfuerzo estimado y del personal con el que se cuente para su realización.

La diferencia entre un procedimiento de calidad y otro improvisado es que el primero define metodologías para hacer estimaciones objetivas y contrastadas dando lugar a estimaciones precisas, mientras que en el segundo las estimaciones son resultados de análisis subjetivos y no contrastados conduciendo a resultados vagos, casi siempre, muy optimistas.

2. Gestión de Riesgos

Usualmente, cuando realizamos el análisis de un proyecto, aparecen incertidumbres sobre su comprensión, sobre el método de solución, sobre las herramientas de solución, entre otras. De no atender prioritariamente estos aspectos inciertos, conocidos formalmente como riesgos, se convertirán en fuentes potenciales de errores en nuestro proceso.

Una de las líneas esenciales de la gestión moderna de software es la gestión dinámica de riesgos. Este proceso periódico consiste en identificar y analizar cada riesgo, estimar su probabilidad de ocurrencia y su posible impacto en el cronograma, y definir un plan de gestión del mismo, el cual es un grupo de acciones orientadas a prevenir el riesgo o a corregir sus consecuencias, en función del proceso que resulte menos costoso. Una gestión global incluye además el mantenimiento de listas actualizadas de riesgos ordenados por peligrosidad, de forma que nos sea posible centrarnos en aquellas incertidumbres potencialmente más destructivas.

Un procedimiento de calidad para el desarrollo de software debe incluir una metodología de gestión de riesgos, así como un registro de riesgos y errores frecuentes en la organización que ayuden a evitar omisiones importantes.

3. Planificación

La planificación consta de dos partes: la división del proyecto en tareas y la asignación de recursos a tareas, es decir, ordenar las tareas en el tiempo, asignándoles recursos humanos y materiales para su realización. El tiempo asignado a una tarea depende de múltiples factores: tamaño y complejidad de la tarea (productos de la estimación), grado de conocimiento o de incertidumbre que tenemos sobre ella (análisis de riesgos), y de la preparación y experiencia del personal que debe realizarla.

En proyectos con riesgos importantes, el tiempo de desarrollo no suele ser cerrado, sino en forma de rango dependiendo de los riesgos presentes. Su posible presentación a clientes debe acompañarse de un documento que relacione incertidumbres con el rango. Estos proyectos deben ser periódicamente re-estimados y su planificación refinada, tareas que deben ser también planificadas.

Es recomendable dentro de un procedimiento de calidad la existencia de una metodología con directrices para realizar planes de desarrollo, relacionada con las metodologías de elaboración de estimaciones y de gestión de riesgos.

4. Control y Seguimiento

Las actividades de control y seguimiento consisten en verificar que el progreso del proyecto se ajusta al plan y a los estándares, es decir, que se están cumpliendo los plazos, costos, y los objetivos de calidad. En otras palabras, el control y seguimiento es un conjunto de actividades de validación y verificación del proceso de desarrollo. Idealmente, estas actividades deben aportar absoluta visibilidad del progreso del desarrollo.

Algunas de estas actividades son revisiones y auditorías técnicas, revisiones de hitos, reportes de estado, realizar mediciones (tiempo, presupuesto) y comparar con estimados, etc.

Las tareas de control y seguimiento deben ser también planificadas. Sin ellas no es posible gestionar un proyecto ni sus riesgos, y no hay forma de saber si los planes se están cumpliendo o no. Un control efectivo permite detectar anticipadamente problemas en el cronograma, cuando aún hay tiempo suficiente para actuar sobre él.

5. Medición de Estadísticos

Una de las claves del progreso a largo plazo de una organización de software es la medición de datos para analizar la calidad del software y la productividad. Aparte de las típicas mediciones sobre costos y tiempos en proyectos, recolectar datos históricos sobre cuán largo es un programa (en líneas de código) o un análisis de requisitos (en número y complejidad de requisitos), nos creará bases objetivas para realizar futuras estimaciones en nuevos proyectos que suelen ser generalmente mejores que el instinto puro.

Procesos más sofisticados colectan mediciones sobre los cambios (errores, mejoras o nuevos requisitos) entre sucesivas versiones, por ejemplo, del documento de análisis de requisitos o de cualquier producto de software. Estas mediciones sobre el número y naturaleza de los cambios permiten conocer más objetivamente el nivel de estabilidad o madurez del producto objeto de medición, el grado de flexibilidad ante cambios, entre otras características.

El procedimiento de calidad de desarrollo de software de una organización, debe definir qué mediciones realizar, con qué objetivo y periodicidad, y cómo van a ser colectadas. Es usual disponer de un software que soporte la recolección automática o semiautomática de estas mediciones, y su uso de acuerdo a los fines para los que han sido definidas.

6. Gestión de Relaciones con los Clientes

El conocimiento y aplicación de buenas prácticas en las relaciones con clientes producen beneficios directos para el desarrollo de un software. Buenas relaciones con los clientes disminuyen el tiempo real y percibido de desarrollo, pues eliminan fuentes importantes de errores y riesgos para el proyecto, y propician una cooperación más activa y comprometida por parte de clientes y usuarios. Estas prácticas se extienden por múltiples áreas de la ingeniería y la gestión, por ejemplo, definir y gestionar riesgos asociados con los clientes, emplear prácticas activas de comunicación para ayudar a clientes a comprender lo que quieren, involucrar a clientes y usuarios en actividades de control del progreso del proyecto, emplear modelos incrementales de ciclos de vida que proporcionen al cliente señales periódicas y tangibles de progreso, entre otras.

Como en los casos anteriores, la organización debe documentar la política de gestión de las relaciones con clientes.

Procesos de Aseguramiento de la Calidad del Software

El aseguramiento de la calidad del software (ACS) consiste en controlar que los productos y procesos del desarrollo de software cumplen estándares de completitud y calidad. Como se ha comentado, ACS cumple el rol de watchdog de los procesos del ciclo de vida del software.

Existen dos formas de obtener software de calidad. La primera es prevenir la falta de calidad, definiendo normas, estándares, métodos y técnicas apropiadas durante los procesos del ciclo de vida. La segunda es detectar y corregir la falta de calidad (e.g. errores en el código, en el diseño, en manuales de usuarios, o código complejo mal documentado) a través de la evaluación de procesos, mejoramiento de procesos, revisiones y, por supuesto, testeado de software.

Las actividades de aseguramiento de la calidad deben ser planificadas, con sus correspondientes asignaciones de recursos humanos y materiales. O sea, asegurar la calidad cuesta dinero. Sin embargo, la falta de calidad también tiene un precio. Joseph Juran, uno de los más notables teóricos de la economía de la calidad, propuso en 1951 el análisis de costos relacionados con la calidad en su libro *Quality Control Handbook*. Juran distingue 3 tipos de costos de aseguramiento de la calidad:

- Costos de prevención: costos de actividades específicamente diseñadas para prevenir una calidad pobre.
- Costos de detección: costos de actividades orientadas a encontrar problemas de calidad.
- Costos de fallos: costos derivados de una calidad pobre, por ejemplo, el costo de corregir errores y el costo de atender quejas de

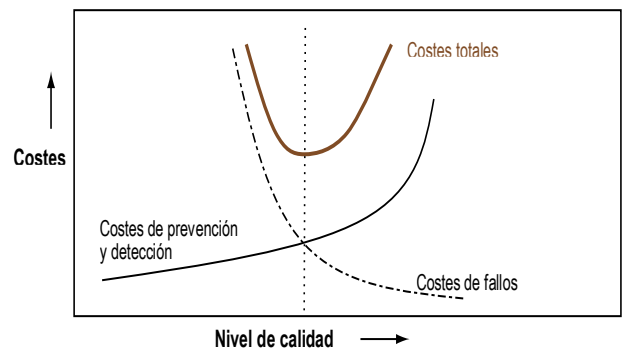


Figura 1: Relación entre costes (Juran).

usuarios, entre otros.

La relación entre estos costos es ilustrada por Juran en la Figura 1. Juran indica que “el costo de las actividades de aseguramiento de la calidad necesarias para alcanzar niveles de calidad altos crece geoméricamente según nos acercamos a la perfección”. Perseguir la perfección, por tanto, no es rentable porque un pequeño incremento en calidad requerirá una gran inversión. Las inversiones en aseguramiento de la calidad deben hacerse mientras el costo total de prevenir y detectar problemas sea menor que el costo de corregirlos.

1. Prácticas de prevención

Los estándares son uno de los medios más efectivos para garantizar la calidad del software. Prácticamente para cada producto a elaborar (manual de usuario, interfaz, código, análisis de requisitos, etc.) o proceso a realizar (análisis de riesgos, diseño, planificación, etc.) deben existir estándares o normas organizacionales que definan

directrices sobre cómo hacerlo. Los estándares tienen dos beneficios principales: i) evitan improvisaciones, olvidos y errores al definir qué hacer y cómo hacerlo y ii) proponen una manera uniforme de hacer que facilitan comparaciones entre proyectos y colaboraciones entre equipos de trabajo diferentes.

Otro grupo de técnicas orientado a prevenir errores y omisiones es el de métodos formales, que hace referencia a una variedad de técnicas de modelación matemáticas aplicables al diseño de sistemas informáticos. Los métodos formales pueden ser usados para especificar y modelar el comportamiento de un sistema y para verificar matemáticamente que el diseño y la implementación del sistema satisfacen sus especificaciones. Estas técnicas pueden ser aplicadas prácticamente a todos los niveles del ciclo de vida del software, por ejemplo, un lenguaje de especificaciones formales para escribir requisitos (VDM, OCL), proceso de transformación de requisitos en código ejecutable que garantice que el código satisface las propiedades especificadas, probar las propiedades de las especificaciones a través de técnicas automáticas como verificación de modelos y prueba de teoremas, formalismos para derivar casos de pruebas a partir de las especificaciones de software, entre otras.

Los métodos formales no son una estrategia de "todo o nada". Aplicar métodos formales solo a las partes más críticas de un sistema es una estrategia útil y muy efectiva. La verificación formal completa debe aplicarse únicamente en sistemas críticos que requieran la máxima fiabilidad.

2. Prácticas de detección y corrección

La práctica más conocida de detección de errores es el testeo de software. Aparte de ser un proceso primario de ingeniería (Verificación y Validación) para asegurar que las especificaciones y necesidades del usuario final se satisfacen, el testeo de software también pertenece a las actividades de detección de la gestión de la calidad pues ellas pueden detectar fallos de calidad.

Las actividades de testeo pueden clasificarse en estáticas o dinámicas. Las técnicas de testeo estático detectan errores sin ejecutar el programa, por ejemplo, inspecciones o recorridos de código son técnicas que consisten en detectar errores a través de la lectura de código. El testeo dinámico, por su parte, implica la ejecución de programas.

A su vez, las técnicas dinámicas pueden subdividirse en dos estrategias generales: testeo de comportamiento (caja negra, basado en datos, entrada/salida, basado en requisitos), en la que el tester es completamente ajeno al código fuente del programa, y está únicamente interesado en casos en los que el programa no se comporta como se espera, y el testeo estructurado (caja blanca, basado en lógica, basado en código,) en la que el tester examina la estructura interna del programa con el objetivo de derivar casos de test.

La derivación de casos de test es, independiente de la estrategia de testeo utilizada, su parte más importante y difícil. Existen múltiples técnicas para este fin que varían desde la aplicación informal de heurísticas simples (testeo según la experiencia, testeo de ciclo de datos, testeo de combinación de datos) hasta la derivación formal utilizando modelos como grafos de flujo de datos o de control.

3. Evaluación y mejora de proceso

La mejora de procesos de software (SPI, de Software Process Improvement) se orienta a reducir costos y riesgos de los procesos,

acortar el tiempo del proceso de desarrollo, y a incrementar la calidad del producto. Existen múltiples métodos, y técnicas que pueden ser usadas para determinar la efectividad de un proceso y para definir las correspondientes acciones de mejora. Estos modelos se dividen en dos estrategias principales: enfoque top-down, por ejemplo, CMMI, SPICE y BOOTSTRAP, que se basan fundamentalmente en evaluación y en modelos, y enfoque bottom-up, por ejemplo, GQM, QIP y AMI, los cuales aplican fundamentalmente mediciones como guías básicas de mejora.

Los modelos de madurez de proceso de desarrollo de software, como los antes mencionados, no han tratado adecuadamente el proceso de testeo. ¿Qué es exactamente un proceso maduro de testeo? ¿Cómo se debe organizar y poner en marcha la mejora de un proceso de testeo? ¿Cómo se debe incorporar a la organización de una empresa? Para responder a estas preguntas existen modelos especializados para medir la madurez y mejorar el proceso de testeo, por ejemplo, TIM (Test Improvement Model), TOM (Test Organisation Maturity Model), TPI (Test Process Improvement Model) y TMM (Testing Maturity Model).

Servicios de Calidad del Software

El ITI está involucrado en la definición de una metodología propia de evaluación de la calidad del proceso de software basada en el modelo CMM (Capability Maturity Model), y en el modelo de gestión descrito en el libro Software Project Management: A Unified Framework de Walker Royce. Como consecuencia de una evaluación satisfactoria, el ITI certificará con un sello propio un nivel de calidad en el proceso de software de una organización. Complementariamente se elaborará un informe con la caracterización del estado actual del proceso, sugerencias y recomendaciones de mejoras, y conclusiones finales. La metodología persigue juzgar la efectividad del proceso de software de una organización e identificar aquellas áreas susceptibles de ser mejoradas. La propia metodología pretende ser la herramienta que describa el camino a seguir para incrementar gradualmente la madurez del proceso de software.

El ITI también ha desarrollado servicios de testeo de software que ofrecen a las empresas la posibilidad de adquirir información sobre:

- La eficacia de su proceso de testeo. Estos servicios se dirigirán a la evaluación y el asesoramiento del proceso de testeo de software para poder definir pasos de mejora graduales y controlados. Estos servicios proporcionarán una vista independiente de donde está y a dónde va la empresa.
- La calidad de su propio software o software externo que desean comprar para el uso interno. Estos servicios estarán dirigidos al testeo y evaluación de productos finales de software.

Para la ejecución de todas estas actividades, el ITI cuenta con personal con certificado ISEB que utiliza métodos ampliamente aceptados y probados como TPI, TMM, TMAP, y estándares internacionales mencionados arriba, todo lo cual es una garantía de calidad. Sin embargo, ITI no trata estos métodos como dogmas sino administra, controla y adapta estos métodos por medio de investigación constante con orientación práctica.

Autores: Ramón Mollineda, Tanja Vos
Para más información sobre Calidad y Testing:
scq@iti.upv.es