Procesos de ingeniería de software

Software engineering processes

Héctor Arturo Flórez Fernández*

Fecha de recepción: 5 de abril de 2009
Fecha de aceptación: 15 de mayo de 2009

Resumen

Existen diferentes procesos en el tema ingeniería de software, que tienen como objetivo presentar diferentes técnicas que consisten en la combinación de procedimientos que permiten guiar el diseño y el desarrollo de sistemas de software a un producto final de calidad. Algunos de esos procesos son: modelo secuencial, modelo en espiral, modelo win win, rational unified process (RUP), extreme programming (XP), método delphi, métrica versión 3, modelo de madurez de capacidad (CMM), proceso personal de software (PSP) y proceso en equipo de software (TSP).

Palabras clave: Ingeniería de Software, RUP (Rational Unified Process), XP (eXtreme Programming), CMM (Capability Maturity Model), PSP (Personal Software Process), TSP (Team Software Process).

* Ingeniero Electrónico de la Universidad El Bosque de Bogotá, Ingeniero de Sistemas de la Universidad El Bosque de Bogotá, Especialista en Alta Gerencia de la Universidad Militar Nueva Granada de Bogotá, magíster en Ciencias de Información y las Comunicaciones de la Universidad Distrital Francisco José de Caldas. Docente Investigador Fundación Universitaria Konrad Lorenz, docente Universidad Distrital Francisco José de Caldas. Adscrito al grupo de investigación Promente de la Fundación Universitaria Konrad Lorenz. Correo electrónico: hectorarturo@yahoo.com.

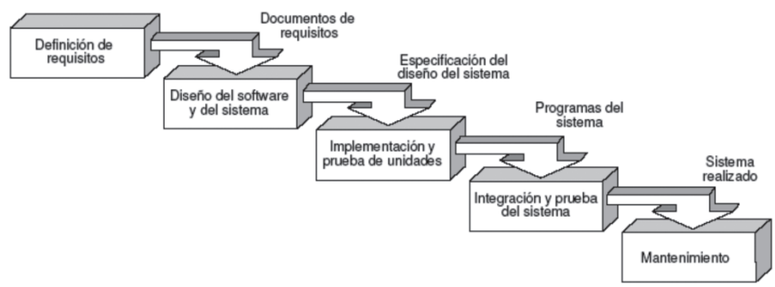
Abstract
There are different processes in software engineering, that have the goal presenting different techniques that consist in the combination of processors that allow the guide in the design and development of software systems to a final product whit quality. Some of these processes are sequential model, spiral model, win win model, rational unified process (RUP), extreme programming (XP) delphi method, metric version 3, capability maturity model (CMM), personal software process (PSP) and team software process (TSP)

Key words: Software Engineering, RUP (Rational Unified Process), XP (eXtreme Programming), CMM (Capability Maturity Model), PSP (Personal Software Process), TSP (Team Software Process)

Modelo lineal secuencial

También llamado "ciclo de vida clásico" o "modelo en cascada", sugiere un enfoque[1] sistemático, secuencial para el desarrollo del software que empieza con el establecimiento de requisitos y pasa a las fases de análisis, diseño, codificación, pruebas y mantenimiento[1].

Figura 1. El ciclo de vida del modelo lineal secuencial [3]



Ingeniería y modelado de sistemas e información

Establece los requisitos para todos los elementos del sistema y le asigna al software algún subgrupo de estos requisitos. Teniendo en cuenta el sistema como tal y la empresa desde los puntos de vista estratégico y de negocio.

1 El modelo original propuesto por Winston Royce hacía provisiones para "ciclos de realimentación", la gran mayoría lo aplica como si fuera estrictamente lineal.

Análisis de los requisitos del software

En esta parte el ingeniero intenta comprender la naturaleza de los programas que han de construirse, así como el dominio de la aplicación.

Diseño

En esta fase se traducen los requisitos a una representación que pueda ser evaluada previamente antes de empezar la fase de codificación.

Generación de código

Se traduce lo diseñado en la fase anterior a un lenguaje que pueda ser procesado por la máquina.

Pruebas

Cuando el código se ha generado es el momento de empezar a realizar las pruebas del programa, centrado en los procesos lógicos internos y en los procesos externos funcionales para asegurar que las entradas producen los resultados requeridos.

Mantenimiento

El software puede necesitar cambios, debido a varias razones: errores, el entorno o mejoras sugeridas por el cliente.

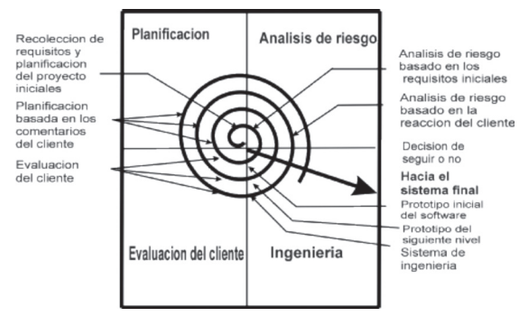
Debilidades del modelo en cascada

- Nada está hecho hasta que todo esté terminado.
- Se pueden presentar problemas debido a falta de información en la fase de levantamiento de requisitos.
- El progreso ordenado no es realista: durante el proceso de desarrollo pueden cambiar las variables de entorno del usuario que pueden afectar de manera potencial y los requisitos planteados inicialmente.
- La eliminación de fallas suele ser extremadamente difícil durante las últimas etapas de prueba del sistema.

Modelo en espiral

El modelo espiral, propuesto originalmente por B. Boehm, es un modelo de proceso de software evolutivo que conjuga la naturaleza iterativa de construcción de prototipos con los aspectos controlados y sistemáticos del modelo en cascada [1], añadiendo un nuevo elemento: el análisis de riesgo [2]. El modelo, representado mediante la espiral de la figura 2, define cuatro actividades principales:

Figura 2. Modelo espiral [2]



- Planificación. Determinación de objetivos, alternativas y restricciones.
- Análisis de riesgo. Análisis de alternativas e identificación/resolución de riesgos.
- Ingeniería. Desarrollo del producto del "siguiente nivel".
- Evaluación del cliente. Valorización de los resultados de la ingeniería.

Durante la primera vuelta alrededor de la espiral se definen los objetivos, las alternati-

vas y las restricciones, y se analizan e identifican los riesgos. Si el análisis de riesgo indica que hay una incertidumbre en los requisitos, se puede usar la creación de prototipos en el cuadrante de ingeniería para dar asistencia, tanto al encargado de desarrollo como al cliente. Éste evalúa el trabajo de ingeniería (cuadrante de evaluación de cliente) y sugiere modificaciones. Sobre la base de los comentarios del cliente se produce la siguiente fase de planificación y de análisis de riesgo; en cada bucle alrededor de la espiral, la culminación del análisis de riesgo resulta en una decisión de "seguir o no seguir".

Con cada iteración alrededor de la espiral (comenzando en el centro y siguiendo hacia el exterior), se construyen sucesivas versiones del software, cada vez más completa y, al final, al propio sistema operacional.

Actualmente, el paradigma del modelo en espiral para la ingeniería de software es el enfoque más realista para el desarrollo de software y de sistemas a gran escala. Utiliza un enfoque evolutivo para la ingeniería de software, puesto que le permite al desarrollador y al cliente entender y reaccionar conforme a los riesgos en cada nivel evolutivo. De igual forma, utiliza la creación de prototipos como un mecanismo de reducción de riesgo, pero, lo que es más importante permite a quien lo desarrolla aplicar el enfoque de creación de prototipos en cualquier etapa de la evolución de prototipos [2].

Modelo en espiral (win-win)

El modelo en espiral tratado anteriormente sugiere una actividad del marco de trabajo que aborda la comunicación con el cliente. El objetivo de esta actividad es mostrar los requisitos del cliente. En un contexto ideal, el desarrollador simplemente le pregunta al cliente lo que necesita y el cliente proporciona detalles suficientes para continuar. Desgraciadamente, esto raramente ocurre. En realidad, el cliente y el desarrollador entran en un proceso de negociación, en el cual el cliente puede ser preguntado para sopesar la funcionalidad, el rendimiento y otros productos o características del sistema frente al coste y al tiempo de comercialización.

Las mejores negociaciones se esfuerzan en obtener "gana-gana". Esto es, el cliente gana obteniendo el producto o sistema que satisface la mayor parte de sus necesidades y el desarrollador gana trabajando para conseguir presupuestos y lograr una fecha de entrega realista.

El modelo en espiral *win-win* define un conjunto de actividades de negociación al principio de cada paso alrededor de la espiral. Además del énfasis realizado en la negociación inicial, el modelo en espiral *win-win* introduce tres hitos en el proceso, llamados puntos de fijación, que ayudan a establecer la completitud de un ciclo alrededor de la espiral y proporcionan hitos de decisión antes de continuar el proyecto de software.

En esencia, los puntos de fijación representan tres visiones diferentes del progreso mientras que el proyecto recorre la espiral. El primer punto de fijación, llamado objetivos del ciclo de vida, define un conjunto de objetivos para cada actividad principal de ingeniería de software. El segundo punto de fijación, llamado arquitectura del ciclo de vida, establece los objetivos que se deben conocer, mientras que se define la arquitectura del software y el sistema. La capacidad operativa inicial es el tercer punto de fijación y representa un conjunto de objetivos asociados a la preparación del software para la instalación y distribución [1].

RUP (Rational Unified Process)

RUP (Rational Unified Process) es una metodología que permite construir software

en tiempo y precio justos. RUP utiliza UML (Unified Modeling Language), para especificar, visualizar, construir y documentar sistemas de software. También representa un conjunto de las mejores prácticas de ingeniería que han probado ser exitosas en el modelo de sistemas, reduciendo la complejidad y el diseño de los procesos [7].

Un proceso de Ingeniería de Software requiere herramientas que apoyen todas las actividades del ciclo de vida de los sistemas. Como respuesta a ello Rational Software Corp. pone a disposición herramientas que están diseñadas para hacer más eficiente la aplicación de metodologías como el RUP, de esa forma se permite la creación de un ambiente de desarrollo óptimo. El conjunto de herramientas de Rational está basado en seis principios fundamentales para el desarrollo de software, denominadas las seis mejores prácticas que son: administración de los requerimientos, desarrollar en forma iterativa, modelar visualmente, pruebas continuas, arquitecturas basadas en componentes y control de cambios [7].

Administrar los requerimientos

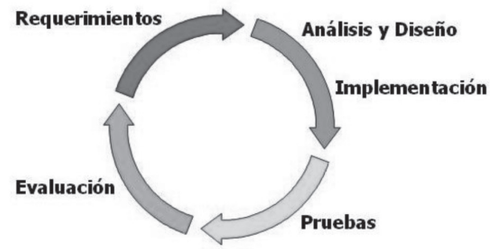
El desafío de administrar los requerimientos de un sistema de software es que son dinámicos, es decir, pueden cambiar durante la vida de un proyecto. Identificar los verdaderos requerimientos de un sistema es delicado, debido a que hay que satisfacer objetivos económicos y técnicos en un proceso continuo.

Desarrollar software iterativamente

El enfoque iterativo facilita la implementación de nuevos cambios tácticos de requerimientos y características del cronograma. Con este enfoque se fortalece tempranamente la identificación de los riesgos del proyecto, cuando aún es posible atacarlos y reaccionar a tiempo y eficientemente. De cada iteración debe resultar una nueva versión ejecutable para verificar claramente que todos los productos generados actúan en forma predecible y repetible [6].

Las iteraciones que se deben realizar en la elaboración de un proyecto se describen en la siguiente figura.

Figura 3. Desarrollo interactivo



Modelar software visualmente

Hacer modelos es importante, porque ayuda al equipo de desarrollo a visualizar, especificar, construir y documentar la estructura y comportamiento de la arquitectura de un sistema de software. Si, además, se utiliza un lenguaje de modelación estándar, los distintos miembros del equipo de desarrollo pueden comunicar sus decisiones sin ambigüedades. Las herramientas de modelación visual facilitan la administración de los modelos. Permiten presentar el modelo en distintos niveles, ocultando los detalles. En resumen, mejora la capacidad del equipo para administrar la complejidad del software. [7]

Verificar continuamente la calidad del software

Encontrar y reparar un problema de software después de la implementación, puede resul-

tar demasiado costoso. Por esta razón es importante evaluar continuamente la calidad de un sistema con respecto a su funcionalidad, confiabilidad y *performance*. La actividad fundamental que involucra esta práctica es la revisión, la cual permite encontrar las fallas antes de la puesta en producción de un sistema. Si, además se utiliza la práctica de desarrollar software iterativamente, se está revisando la versión en cada iteración, logrando así un proceso de evaluación continuo y cuantitativo [7].

Utilizar arquitecturas basadas en componentes

El desarrollo basado en componentes es un enfoque importante de arquitectura de software, porque permite la reutilización o adaptación de componentes existentes de miles de fuentes comercialmente disponibles. Un componente de software se puede definir como una pieza no trivial de software, un módulo, un paquete o un subsistema que completa una función clara [7].

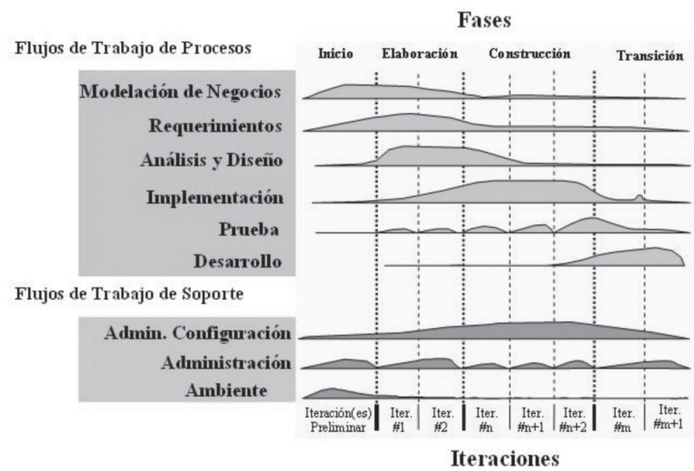
Controlar los cambios del software

Normalmente, un proyecto de sistemas, implica múltiples desarrolladores organizados en diferentes equipos de trabajo. Posiblemente, estos equipos de trabajo se encuentran en diferentes sitios, trabajando en conjunto y en múltiples iteraciones, para lograr nuevas versiones, productos y plataformas. Coordinar las actividades y los productos que van generando los desarrolladores implica establecer procesos repetibles para administrar los cambios al software y otros artefactos del desarrollo. Esta coordinación permite una mejor asignación de los recursos, basada en las prioridades y en los riesgos del proyecto y administrar activamente los cambios a través de las iteraciones [7].

Fases para el diseño y desarrollo

Existen cuatro fases en los procesos RUP: inicio, elaboración, construcción y transición. Estas fases representan el énfasis de las actividades dentro de cada iteración [5].

Figura 4. Fases de RUP [5]



Inicio

En esta fase, la meta de las iteraciones es ayudar al equipo de proyecto a decidir cuáles serán los objetivos verdaderos del proyecto [5].

Elaboración

En esta fase se establece una comprensión firme del problema que se solucionará, se establece la fundación arquitectónica del software, se apoya un plan detallado de iteraciones subsecuentes se refina el proceso y se elimina los altos riesgos. Las iteraciones producidas en esta fase están, en el promedio, perceptiblemente menos disponible que las producciones en la fase de inicio. Cada iteración debe agregar nuevas características al cuerpo del software [5].

Construcción

Las iteraciones en la fase de la construcción no son muy diferentes de las iteraciones de la fase de la elaboración. Cada iteración agrega características al software. Durante esta fase, se espera que las descripciones del caso del uso se estabilizarán hasta cierto punto, sin embargo, en muchos dominios del proyecto continuarán cambiando a través del curso de la vida del proyecto [5].

Transición

En esta fase, las iteraciones continúan agregando características al software. Sin embargo, esas características agregan a un sistema que los usuarios están utilizando activamente. Los artefactos producidos en esta fase son los mismos que los producidos en la fase de construcción. El equipo simplemente mejora el sistema hacia los objetivos que fueron fijados en el final de la fase del inicio [5].

Extreme Programing (XP)

Es una de las metodologías de desarrollo de software más exitosas en la actualidad, utilizadas para proyectos de corto plazo, corto equipo y cuyo plazo de entrega era ayer. La metodología consiste en una programación rápida o extrema, cuya particularidad es tener como parte del equipo, al usuario final, pues es uno de los requisitos para llegar al éxito del proyecto.

Características de XP, la metodología se basa en:

- Pruebas unitarias: se basa en las pruebas realizadas a los principales procesos, de tal manera que adelantándonos en algo hacia el futuro, podamos hacer pruebas de las fallas que pudieran ocurrir. Es como si nos adelantáramos a obtener los posibles errores.
- Refabricación: se basa en la reutilización de código, para lo cual se crean patrones o modelos estándares, siendo más flexible al cambio.
- Programación en pares: una particularidad de esta metodología es que propone la programación en pares, la cual consiste en que dos desarrolladores participen en un proyecto en una misma estación de trabajo. Cada miembro lleva a cabo la acción que el otro no está haciendo en ese momento. Es como el chofer y el copiloto: mientras uno conduce, el otro consulta el mapa.

El desarrollo bajo XP tiene características que lo distinguen claramente de otras metodologías:

- Los diseñadores y programadores se comunican efectivamente con el cliente y entre ellos mismos.
- Los diseños del software se mantienen sencillos y libres de complejidad o pretensiones excesivas.

Se obtiene retroalimentación de usuarios y clientes desde el primer día, gracias a las ba-

terías de pruebas El software es liberado en entregas frecuentes tan pronto como sea posible. Los cambios se implementan rápidamente tal y como fueron sugeridos. Las metas en características, tiempos y costos son reajustadas permanentemente en función del avance real obtenido.

Con estas características no es sorprendente que XP sea la metodología más apropiada para un entorno caracterizado por requerimientos cambiantes, originados por un mercado fluctuante y los propios avances de la tecnología y los negocios.

Cambio de paradigma

El cliente típico de servicios de desarrollo de software y páginas Web tiene mucha dificultad para ofrecer desde el inicio información precisa y detallada del sistema que necesita. Esto es normal y la explicación radica en muchos factores, entre los principales, la falta de prototipos y las sucesivas oportunidades para mejorar el sistema que el cliente encuentra mientras éste va tomando un forma más tangible.

El desarrollo de un entregable de características fijas a un costo fijo crea más problemas de los que pretende solucionar. A pesar de ser el paradigma comercial más usado en la actualidad, es un modelo que desperdicia demasiadas oportunidades de entregar como resultado un software no sólo más útil, sino también mejor diseñado y más fácil de mantener. Además, la dificultad natural en establecer las características exactas de un producto que inicialmente es sólo un concepto que genera diferencias, muchas veces insalvables entre clientes y desarrolladores que originan la ruptura de las relaciones comerciales, casi siempre, ni bien el proyecto es finalmente entregado y en casos incluso antes. La solución radica en un manejo más eficiente de los cambios en los requerimientos y un fuerte enfoque en las pruebas de aceptación de cada etapa.

Gestión de cambios

El cliente está en el pleno derecho de hacer cuantos cambios necesite al proyecto, si con ello consigue un mejor resultado final. Para lograr esto, desarrolladores y clientes deben trabajar en conjunto y muy de cerca desde el primer día y las metas en términos de características, tiempos y costos deben ser reajustadas permanentemente.

Un desarrollador nunca debe eliminar características con el afán de disminuir tiempos o costos sin la aprobación del cliente. Un cliente no puede esperar cambiar reiteradamente los requerimientos de un sistema si ya los ha probado y aceptado sin incurrir en gastos adicionales.

Gestión de costos

XP crea transparencia y un clima de agilidad en la relación entre desarrolladores y clientes. El costo de hora/hombre por cada tipo de recurso es conocido y acordado desde el principio. Un proyecto de varios meses es dividido en pequeños proyectos de pocas semanas de duración y las metas y cronogramas se van ajustando en tiempo real, de acuerdo con el nivel de avance y las dificultades reales que ofrece el proyecto aceptadas en forma conjunta por desarrolladores y clientes.

El hecho de tener que aceptar resultados de poca calidad o que no solucionan realmente los problemas de las organizaciones perjudica enormemente a los clientes. De igual forma, tener que enfrentar una gran cantidad de cambios y ajustes y regresar a etapas ya realizadas por cambios en los requerimientos o

los criterios de aceptación perjudica enormemente a los desarrolladores, si es que no van a recibir una compensación económica por el esfuerzo adicional requerido.

Se debe trabajar un máximo de 40 horas por semana. No se trabajan horas extras en dos semanas seguidas. Si esto ocurre, probablemente, está ocurriendo un problema que se debe corregir. El trabajo extra desmotiva al equipo. Los proyectos que requieren trabajo extra para intentar cumplir con los plazos suelen ser entregados con retraso. En lugar de esto, se puede realizar el juego de la planificación para cambiar el ámbito del proyecto o la fecha de entrega.

Metodo Delphi

El método Delphi consiste en una serie de interrogaciones repetidas, usualmente por medio de cuestionarios a un grupo de individuos, cuyas opiniones o juicios son de interés. Después de un interrogatorio inicial a cada individuo, cada subsiguiente interrogatorio es acompañado por información considerando las respuestas precedentes, usualmente presentadas anónimamente. De esta manera, el individuo es alentado a reconsiderar, de ser apropiado, o cambiar su respuesta previa a la luz de las respuestas de otros miembros del grupo. Después de dos o tres vueltas, la posición del grupo es determinada por promedio [10]

Generalidades

El método Delphi es una técnica para la resolución de problemas, toma de decisiones y predicciones de escenarios desarrollado por la corporación Rand de la inteligencia militar de los Estados Unidos. Fue desarrollado durante la Guerra Fría y, al parecer, toma su nombre de la localización del mítico oráculo griego, Delfos.

Este método hace uso de un panel de expertos que de forma asincrónica –no se reúnen, intercambian opiniones a través de correo electrónico u otro medio impersonal–, son seleccionados en función de las áreas de conocimiento requeridas por el problema. Lo anterior implica la noción de que individuos bien informados, haciendo uso de su experiencia e introspección, están mejor equipados para predecir el futuro que las aproximaciones teóricas o la extrapolación de tendencias [11]. Este método asume que a través de repetidos cuestionarios, los expertos se acercarán a un punto en común que será la mejor respuesta, lo cual es definido por indicadores estadísticos.

Etapas del proceso

Este método incluye las siguientes etapas [12]:

- Formación de un equipo para asumir y monitorear un tema específico.
- Selección de uno o más paneles para participar en el ejercicio. Frecuentemente, los panelistas son expertos en el área por ser investigada.
- Desarrollo del primer cuestionario Delphi.
- Prueba del cuestionario para asegurar el uso de palabras correctas, es decir, por ejemplo, para evitar divagar o caer en ambigüedades.
- Transmisión del primer cuestionario a los panelistas.
- Análisis de la primera serie de respuestas.
- Preparación del segundo cuestionario Delphi y sus posibles pruebas.
- Transmisión de la segunda ronda de cuestionarios a los panelistas.
- Análisis de la segunda ronda de respuestas. Los pasos del 7 al 9 son repetidos hasta tanto se alcance la estabilidad necesaria en los resultados.

- Preparación de un reporte por el equipo de análisis para presentar las conclusiones del ejercicio.

Métrica versión 3

Estructura de Métrica versión 3

Ha tomado como referencia el modelo de ciclo de vida propuesto en la norma ISO 12207, distinguiendo procesos principales (planificación, desarrollo y mantenimiento) e interfaces (gestión de proyectos, aseguramiento de la calidad, seguridad y gestión de la configuración), cuyo objetivo es dar soporte al proceso en los aspectos organizativos. Ha sido concebida para abarcar el desarrollo completo de sistemas de información, sea cual sea su complejidad y magnitud, mediante un enfoque orientado al proceso.

La metodología descompone cada uno de los procesos en actividades y éstas, a su vez, en tareas. Para cada tarea se describe su contenido haciendo referencia a sus principales acciones, productos, técnicas, prácticas y participantes.

Procesos principales de Métrica 3

Los procesos de la estructura principal de Métrica 3 son los siguientes:

- Planificación de sistemas de información.
- Desarrollo de sistemas de información.
- Mantenimiento de sistemas de información.

Proceso de planificación de sistemas de información

Su objetivo es proporcionar un marco estratégico de referencia para los sistemas de información de un determinado ámbito de la organización. La perspectiva del plan debe ser estratégica y operativa, no tecnológica. En este proceso participan, por un lado, los responsables de los procesos de la organización con una visión estratégica y, por otro, los profesionales de ingeniería de software [14].

Como productos finales del proceso se obtienen:

- Catálogo de requisitos de PSI.

Arquitectura de información, que se compone de:

- Modelo de información.
- Modelo de sistemas de información.
- Arquitectura tecnológica.
- Plan de Proyectos.
- Plan de mantenimiento del PSI.

Proceso de desarrollo de sistemas de información

Contiene todas las actividades y tareas que se deben llevar a cabo para desarrollar un sistema, cubriendo desde el análisis de requisitos hasta la instalación del software. Este proceso es el más importante de los identificados en el ciclo de vida de un sistema y se relaciona con todos los demás.

Se ha subdivido en cinco procesos:

1. Estudio de Viabilidad del Sistema (EVS). El propósito de este proceso es analizar un conjunto concreto de necesidades, con la idea de proponer una solución a corto plazo. Los resultados del EVS constituirán la base para tomar la decisión de seguir adelante o abandonar.
2. Análisis del Sistema de Información (ASI). El propósito de este proceso es conseguir la especificación detallada del sistema de información, a través de un catálogo de requisitos y una serie de modelos que cubran las necesidades de información de los usuarios para los que se desarrollará el sistema de información y que serán la entrada para el proceso del diseño del sistema de información. Se elabora el pro-

ducto especificación de requisitos software. Se inicia la especificación del plan de pruebas, que se completará en el DSI. La participación activa de los usuarios es una condición imprescindible para el análisis del sistema de información.

3. Diseño del Sistema de Información (DSI). El propósito del DSI es obtener la definición de la arquitectura del sistema y del entorno tecnológico que le va a dar soporte, junto con la especificación detallada de los componentes del sistema de información. A partir de dicha información, se generan todas las especificaciones de construcción relativas al propio sistema, así como la especificación técnica del plan de pruebas, la definición de los requisitos de implantación y el diseño de los procedimientos de migración y carga inicial, cuando proceda. En el diseño de arquitectura del sistema participarán activamente los responsables de sistemas y explotación.
4. "Construcción del Sistema de Información (CSI). Tiene como objetivo final la construcción y prueba de los distintos componentes del sistema de información, a partir del conjunto de especificaciones lógicas y físicas de éste-obtenido en el DSI. Se desarrollan los procedimientos de operación y seguridad y se elaboran los manuales de usuario final y de explotación, como parte del proceso de desarrollo del proyecto Para ello, se recoge la información relativa al producto obtenido del proceso de diseño, se prepara el entorno de producción, se genera el código de cada uno de los componentes del sistema de información y se realizan las pruebas unitarias de cada uno de ellos y las pruebas de integración entre subsistemas".
5. "Implantación y Aceptación del Sistema (IAS). Tiene como objetivo principal la entrega y aceptación del sistema en su totalidad; un segundo objetivo es llevar a cabo las actividades oportunas para el paso a producción del sistema. En este proceso se elabora el plan de mantenimiento del sistema. También se establece el acuerdo de nivel de servicio"[14].

Modelo de madurez de capacidad (CMM)

Al final de los años ochenta y a comienzos de los noventa el SEI desarrolló el modelo de madurez de capacidad CMM que capturó las mejores prácticas de las organizaciones para el desarrollo de software. El CMM es un framework construido sobre las mejores prácticas de las organizaciones. La mejora del proceso ha demostrado aumentar el producto y la calidad del servicio, mientras que las organizaciones la aplican para alcanzar sus objetivos del negocio.

El CMM define cinco niveles de madurez de software basados sobre las áreas de procesos que soportan a las organizaciones.

- Nivel 1. Inicial. Describe a una organización con procesos indefinidos e inmaduros.
- Nivel 2. Repetible. Describe a una organización que requiere gestión en planeación, seguimiento de proyectos, subcontratación, aseguramiento de la calidad y configuración en software.
- Nivel 3. Definido. Define a una organización que adopta procesos de organización focalizados, definidos, programas de entrenamiento, manejo de software integrado, productos de ingeniería y coordinación grupal.
- Nivel 4. Gestionable. Describe organizaciones que tienen análisis, medición y prevención de defectos sobre sus procesos y productos de software.

- Nivel 5. Optimizado. Organizaciones con innovaciones tecnológicas y procesos de manejo al cambio.

Los niveles 2, 3, 4 y 5 describen organizaciones con sucesivos niveles mejores de procesos de madurez de software. La primera meta de las organizaciones es alcanzar el nivel 3 de madurez.

El modelo CMM tiene las siguientes desventajas:

- Está soportado sobre el modelo en cascada.
- No hace énfasis sobre los procesos de arquitectura y diseño.
- Tiene métodos "policivos" en las revisiones, inspecciones y en el aseguramiento de calidad.
- Hay una sobre documentación, ya que se considera que entre más detalles es mejor.
- Hay conflictos entre el modelo CMM y el ISO9001.
- Proliferación de modelos: modelos SE-CMM, desarrollo del producto integrado IDP-CMM, adquisición de software SA-CMM y recursos humanos People-CMM.

Como respuesta a lo anterior, el SEI desarrolló la integración de modelo de madurez de capacidad CMMI.

Proceso personal de software (PSP)

Watts Humphrey, junto con el equipo del SEI, deciden aplicar los principios subyacentes del CMM a las prácticas del desarrollo de software enfocado al desarrollador. El resultado de este esfuerzo es el proceso personal de software (PSP), diseñado para ser un proceso de nivel 5 CMM, para los desarrolladores de software.

El 70% de los costos de desarrollo en un proyecto, lo constituyen las habilidades y los hábitos del trabajo de los ingenieros, los cuales determinan en gran parte el resultado del desarrollo de software. El proceso personal de software PSP (el cómo) puede ser utilizado por los ingenieros como una guía a un acercamiento disciplinado y estructurado al desarrollo de software.

El PSP enseña a los ingenieros lo siguiente:

- Cómo manejar la calidad de sus proyectos.
- Hacer la cosas simples para dar soluciones.
- A mejorar tiempos de estimación y planeación.
- Reducir los defectos de los productos.

El PSP puede aplicar a muchas partes del desarrollo de software, incluyendo:

- Desarrollo de programas.
- Definición de requisitos.
- Estructura de la documentación.
- Pruebas del sistema.
- Mantenimiento de los sistemas.
- Desarrollo de sistemas de software grandes.

Este proceso se centró en una persona y se olvidó que en los proceso de desarrollo intervienen más de una persona.

Proceso de software del equipo (TSP)

Pronto llegó a ser obvio que mientras los resultados con el PSP eran excelentes, era casi imposible mantener la práctica en ambientes de desarrollo colaborativos. Entonces, Watts Humphrey desarrolla el proceso de software de equipo TSP para la unidad operacional más pequeña de las organizaciones de software, el equipo del proyecto. TSP fue diseñado para ser un proceso de nivel 5 CMM, para los equipos del proyecto.

El proceso del software del equipo (TSP), junto con el proceso personal del software PSP, ayuda al ingeniero de alto rendimiento a:

- Aseguramiento de la calidad de los productos de software.
- Crear productos de software seguros.
- Mejora el proceso de gerencia en una organización.

Los grupos de la ingeniería que utilizan el TSP aplican conceptos integrados del desarrollo de sistemas orientados al software, llevando al equipo a:

- Establecer metas.
- Definir papeles del equipo.
- Determinación de riesgos.
- Producir un plan del equipo.

Los equipos pueden estar integrados por sólo desarrolladores o equipos integrados que participan en la obtención del producto. Los equipos pueden estar integrados de 3 a 20 personas.

Conclusiones

El modelo en Cascada posee grandes inconvenientes, en particular, cuando un cambio de requisitos se realiza al final del proceso debido a la rigidez que tiene este modelo. Un mal levantamiento requerimientos en la parte inicial del proceso conllevará a un fracaso. En los modelos en espiral, el proceso de software es evolutivo e incluye riesgos que deben ser tenidos en cuenta desde el inicio de éste, lo que hacen de estos modelos una opción más acorde.

RUP tiene una característica importante explicada en un modelo denominado las seis mejores prácticas. Este modelo le permite a un grupo de trabajo desarrollar un proyecto teniendo en cuenta los requerimientos, el desarrollo iterativo, el diseño bajo un lenguaje de modelado visual, un control calidad, una arquitectura basada en componentes y un control de cambios del software.

La metodología XP es exitosa porque enfatiza la satisfacción del cliente y promueve el trabajo en equipo. Las actividades improductivas han sido eliminadas para reducir costos y frustraciones. Esta metodología ha sido diseñada para solucionar el eterno problema del desarrollo de software por encargo, es decir, entregar el resultado que el cliente necesita a tiempo.

La metodología Métrica versión 3 proporciona un conjunto de métodos y técnicas que le permite a los diseñadores de sistemas de información desarrollar los procesos del ciclo de vida de un proyecto informático. A fin de mejorar la productividad y asegurar la calidad de los productos resultantes, la metodología está soportada por herramientas disponibles en el mercado que automatizan en mayor o menor grado su utilización. En cualquier caso, no todos los productos resultantes de cada tarea son susceptibles de obtenerse de forma automatizada.

El método Delphi permite obtener lo máximo del conocimiento, experiencia y habilidades de un grupo entero de personas, en el cual cada uno cuenta con un conocimiento y una perspectiva única y diferente. Es especialmente bueno para la planeación estratégica de negocios en medio de la incertidumbre y ambientes de movimiento y cambio rápido. Sin embargo, es asincrónico, pues reduce la retroalimentación directa que ofrece un grupo de trabajo. En ocasiones, los desarrollos futuros no son siempre predichos por los expertos en consensos iterativos, sino que en ocasiones se obtienen mejores resultados con panelistas “no expertos”. Los participantes pueden ser expertos buenos, pero pobres estimadores. Los expertos pueden tender a considerar el futuro aisladamente de otros eventos.

Referencias bibliográficas

[1] R. Pressman. *Ingeniería del Software: un enfoque práctico*. 5ª Edición. Mc Graw-Hill 2001.

[2] http://www.itlp.edu.mx/publica/tutoriales/analisis

[3] http://juanfc.lcc.uma.es/EDU/PM/1.IngSoft.pdf

[4] B. Bruegas, A Dutoit. *Ingeniería de software orientada a objetos*. Ed. Prentice-Hall. 2002.

[5] http://www.objectmentor.com/resources/articles/RUPvsXP.pdf

[6] http://www.dcs.ed.ac.uk/teaching/cs2/online/Lectures/CS2Ah/SoftEng/se06-slides.PDF

[7] http://www.stc-online.org/cd-rom/1999/slides/EBestPra.pdf

[8] P. Abrahamsson, O. Salo, J. Ronkainen, J. Warsta. *Agile software development methods Review and analysis*". VTT Publications. 2002.

[9] Beck, K.. "*Extreme Programming Explained. Embrace Change*", Pearson Education, 1999. Traducido al español como: "*Una explicación de la programación extrema. Aceptar el cambio*. Addison Wesley. 2000.

[10] http://pespmc1.vub.ac.be/ASC/DELPHI_METHO.html

[11] http://www.ryerson.ca/~mjoppe/ResearchProcess/841TheDelphiMethod.htm

[12] http://www.iit.edu/~it/delphi.html

[13] http://www.corporate-partnering.com/info/delphi-method-business-planning.htm

[14] Metodología de Planificación y Desarrollo de Sistemas de Información. Guías de Referencia, de Técnicas y del Usuario. http://www.csi.map.es/csi/metrica3/index.html.

[15] A. Durán y B. Bernárd. *Metodología para la Elicitación de Requisitos de Sistemas Software Versión 2.1*.

[16] Desarrollos Mecame S.L. Madrid (España). Técnicas y Prácticas. http://www.desarrollos-mecame.com/formacion/Ingenieria-software/Ingenieria%20software_archivos/tecnicas.pdf