Benemérita
Universidad Autónoma de Puebla

Facultad de Ciencias de la Computación

"Integración del modelo de capacidad de madurez de software (CMM-SW) en el desarrollo de aplicaciones con UML"

Tesis que presenta

Elier Mora Mastranzo

Para obtener el titulo de

Licenciado en Ciencias de la Computación

Director de la tesis

Dr. Abraham Sánchez López

Puebla, Pue. *Otoño 2008*

Contenido.

Introducción 6
Capítulo 1 8
Modelos y estándares de calidad de software 8
1.1 Modelos y Estándares de Calidad de Software. 9
1.1.1 ISO 9001 9
1.1.2 ISO 9126 11
1.1.3 ISO 15540 (SPICE , Software Process Improvement and Capability dEtermination) 11
1.1.4 BOOTSTRAP 12
1.1.5 MoProSoft 15
1.1.6 CMM (Capability Maturity Model) 17
1.2 Porque es importante usar Modelos y Estándares de Calidad de software. 19
1.3 Relación de UML en términos de los modelos y estándares de calidad de software. 20
Capítulo 2 23
Detalles Generales de CMM-SW 23
2.1 Historia 23
2.2 Características de CMM 26
2.3 Estructura 26
2.4 Niveles de Madurez 26
2.4.1 Nivel 1. Inicial 27
2.4.2 Nivel 2: Repetible 27
2.4.3 Nivel 3: Definido 28
2.4.4 Administrado 29
2.4.5 Optimizado 29
2.4.6 Áreas Clave del Proceso 30
2.4.6.1 KPA’s Nivel 2 31
2.4.6.2 KPA’s Nivel 3 31
2.4.6.3 KPA’s Nivel 4 32
2.4.6.4 KPA’s Nivel 5 32
2.4.7 Características de las KPA’s 32
2.4.7.1 Compromisos 33
2.4.7.2 Habilidades 34
2.4.7.3 Actividades 36
2.4.7.4 Medición y análisis 42
2.4.7.5 Verificaciones de implantación 43
Capítulo 3 45
Integración de CMMI con UML 45
3.1 UML 45
3.2 Vista General de UML 46
3.2.1 Bloques de construcción de UML 48
3.2.1.1 Elementos Estructurales 48
3.2.1.2 Relaciones 50
3.2.1.3 Diagramas 51
3.4 Proceso de desarrollo de software con UML y el modelo CMM 53
3.4.1 Modelado 55

3.4.1.1 Principios básicos del modelado 56
3.4.2 Proceso de desarrollo de software 56
Capítulo 4 58
Prototipo de Proceso de Desarrollo de Software 58
4.1 Etapas del Proceso 58
4.2 Indicadores de Desempeño 58
4.3 Descripción de la Fases del Proceso 59
4.3.1 Anteproyecto 59
4.3.1.1 Propósito 59
4.3.1.2 Documentos de Entrada 59
4.3.1.3 Actividades 59
4.3.1.4 Verificar 60
4.3.1.5 Criterios Finales 60
4.3.1.6 Entregables 60
4.3.2 Inicio de Proyecto 60
4.3.2.1 Propósito 60
4.3.2.2 Documentos de Entrada 61
4.3.2.3 Actividades 61
4.3.2.4 Verificar 61
4.3.2.5 Criterios Finales 62
4.3.2.6 Entregables 62
4.3.3 Análisis 62
4.3.3.1 Propósito 62
4.3.3.2 Documentos de Entrada 62
4.3.3.3 Actividades 63
4.3.3.4 Verificar 63
4.3.3.5 Criterios Finales 63
4.3.3.6 Entregables 64
4.3.4 Diseño 64
4.3.4.1 Propósito 64
4.3.4.2 Documentos de Entrada 64
4.3.4.3 Actividades 64
4.3.4.4 Verificar 65
4.3.4.5 Criterios Finales 65
4.3.4.6 Entregables 65
4.3.5 Construcción 66
4.3.5.1 Propósito 66
4.3.5.2 Documentos de Entrada 66
4.3.5.3 Actividades 66
4.3.5.4 Verificar 67
4.3.5.5 Criterios Finales 67
4.3.5.6 Entregables 68
4.3.6 Pruebas 68
4.3.6.1 Propósito 68
4.3.6.2 Documentos de Entrada 68
4.3.6.3 Actividades 68
4.3.6.4 Verificar 69
4.3.6.5 Criterios Finales 69
4.3.6.6 Entregables 69
4.3.7 Pase a Producción 70

4.3.7.1 Propósito 70
4.3.7.2 Documentos de Entrada 70
4.3.7.3 Actividades 70
4.3.7.4 Verificar 71
4.3.7.5 Criterios Finales 71
4.3.7.6 Entregables 71
4.3.8 Cierre de Proyecto 71
4.3.8.1 Propósito 71
4.3.8.2 Documentos de Entrada 71
4.3.8.3 Actividades 71
4.3.8.4 Verificar 72
4.3.8.5 Criterios Finales 72
4.3.8.6 Entregables 72
Capítulo 5 73
Conclusiones y trabajos futuros 73
Bibliografía ..... A

Introducción

La industria del software representa una actividad económica de suma importancia para todos los países del mundo en la actualidad. Ofrece múltiples fuentes de negocio, así como empleos y se perfila como la oportunidad más grande de los países en vías de desarrollo. Pero en los países latinoamericanos, la industria del software es incipiente e inmadura, lo cual conlleva a falta de competitividad que a su vez dificulta su crecimiento.

El sector informático se enfrenta a una serie de problemas como la dependencia tecnológica del país, el desconocimiento de la importancia que tiene el proceso de desarrollo de software sobre la calidad del producto y la construcción del software en forma artesanal, empírica y caótica. A raíz de esto el software desarrollado es de baja calidad, el tiempo de desarrollo es inapropiado, los costos no son competitivos, las actividades de operación y mantenimiento del software son difíciles y por supuesto existe incremento de la insatisfacción de los clientes y los usuarios finales.

Hablar de calidad del software implica la necesidad de contar con parámetros que permitan establecer los niveles mínimos que un producto de este tipo debe alcanzar para que se considere de calidad. El problema es que la mayoría de las características que definen al software no se pueden cuantificar fácilmente; generalmente, se establecen de forma cualitativa, lo que dificulta su medición, ya que se requiere establecer métricas que permitan evaluar cuantitativamente cada característica dependiendo del tipo de software que se pretende calificar.

Una estrategia primordial para resolver dichos problemas es desarrollar productos de calidad. La calidad de los productos esta íntimamente ligada a la calidad de los procesos utilizados para desarrollarlos. Entonces es evidente que para incrementar la calidad del producto de la empresa de desarrollo de software deben implementar proyectos para la mejora de sus procesos de software.

Asegurar la calidad a través del mejoramiento de los procesos de software es un paso que las empresas deben de dar como respuesta a dos situaciones: la primera por imagen, para poder exportar el software e ingresar y mantenerse en un mercado global; la segunda por necesidad, para poder hacer de sus proyectos unidades administrativas eficientes y eficaces.

Las empresas de software tienen problemas de madurez en sus procesos de desarrollo de software, en algunos casos existe un proceso de software pero nunca se aplica de la manera más adecuada y en otros casos no existe un proceso real conduciendo en ambos casos a modelos caóticos de operación que efectúa toda la empresa. Además estas empresas también planean asegurar la calidad de sus productos a través de la mejora de sus procesos acreditándose en modelos de calidad del ISO. Previo a la acreditación del ISO la preparación es muy larga y costosa, además sus modelos de mejoramiento, proceso y evaluación están estructurados para aplicarse a grandes empresas, esto nos lleva a que es muy complicado ser aplicado a pequeñas empresas debido a que un proyecto de mejora puede ser una gran inversión de dinero, tiempo y recursos.

Introducción

Según como afronten las organizaciones el desarrollo del software, éste puede comportarse como factor de riesgo o amenaza para el negocio; o por el contrario como una poderosa oportunidad de negocio. Todas las empresas quieren producir más rápido, mejor y con menores costos, y sin duda esto es posible porque la naturaleza del software no es origen de riesgos y problemas, sino una fuente de oportunidades.

La evolución hacia entornos de ingeniería del software requiere cambios severos en la organización, así como el convencimiento, implicación y empuje de los directivos de la misma. Pero sobre todo el diseño de un modelo de producción propio que sepa aprovechar la personalidad de la organización, y responder a las particularidades de su negocio.

El problema que pueden encontrar quienes deciden implantar métodos más eficientes es caer en la desorientación ante el abanico de modelos de calidad, de procesos y de técnicas de trabajo desplegado en la última década, o abrazar al primero que se presenta en la puerta de la organización como “solución” de eficiencia y calidad.

Por lo anterior las empresas deben de enriquecerse de información para proporcionar a su empresa las herramientas necesarias para motivarlas a mejorar sus procesos de desarrollo de software con el objetivo de facilitar el posicionamiento y la competitividad en el mercado nacional e internacional en un futuro muy cercano.

Por otra parte, UML ha llegado rápidamente a convertirse en la notación más popular para modelado orientado a objetos. Gracias a la definición de su metamodelo y a los mecanismos de extensión incluidos, UML ofrece una excelente oportunidad para establecer un marco común para la representación de especificaciones de requisitos, de desarrollo y de pruebas.

En esta tesis propondremos un prototipo de ingeniería de software, para el desarrollo de nuevos proyectos, haciendo uso de UML y CMM-SW (Capability Maturity Model - Software). Para realizar esta fusión tomaremos los procesos de CMM-SW, y determinaremos en cual de estos procesos pueden involucrarse los diagramas de UML.

En el capitulo 1 se presentaran de manera breve propuestas de modelos y estándares para el aseguramiento de la calidad de software, así como metodologías que son muy socorridas en el mercado de la industria del software. En el capitulo 2 hablaremos de CMM (Capability Maturity Model), desde su principio, su evolución y su organización de nieles de madurez. En el capitulo 3 trataremos a UML desde su concepción, una breve descripción de sus diagramas y su uso para el modelado de sistemas, y su integración en CMM. En el capitulo 4, se propone un proceso de ingeniería de software, para nuevos desarrollos, integrando CMM y los diagramas de UML para el modelado del producto y su mayor comprensión, para obtener un producto de calidad. Finalmente en el capitulo 5 se describirán las conclusiones y los trabajos futuros.

Capítulo 1

Modelos y estándares de calidad de software

El desarrollo de software no es una tarea fácil. Prueba de ello es que existen numerosas propuestas metodológicas que inciden en distintas dimensiones del proceso de desarrollo.

El aseguramiento de la calidad del software es uno de los temas más actuales e importantes dentro de la disciplina de la ingeniería de software, y curiosamente, es uno de los temas más olvidados en los programas de estudio universitarios. Cada vez más las organizaciones de software están empezando a reconocer la importancia de contar con ingenieros de software especializados en el área de calidad, y particularmente con conocimientos en áreas tales como administración de la calidad del software, administración de procesos de software, administración de proyectos, métricas de software, administración de la configuración, y pruebas de software.

Un estándar de ingeniería de software es una regla o base de comparación que se utiliza para medir aspectos del software tales como calidad, productividad, duración, esfuerzo, y costo.

Así, en 1984, el Depto. de Defensa (DoD) de los Estados Unidos establece al SEI (Software Engineering Institute) de la Universidad Carnegie Mellon como Centro de Investigación y Desarrollo financiado con la misión de liderar los avances para la mejora de la calidad de los sistemas dependientes del software. ISO plasma los estándares de calidad y desarrollo en 1987 con la norma ISO 9000, un conjunto. de estándares internacionales para sistemas de calidad; en particular ISO 9001 e ISO 9000-3 son aplicables al proceso software y a organizaciones de desarrollo software. La necesidad del DoD de determinar la capacidad de sus contratistas antes del contrato lleva al SEI, bajo la dirección de W. Humphrey al desarrollo de técnicas de evaluación y valoración de la capacidad que dan lugar al desarrollo y publicación, en agosto de 1991, del Capability Maturity Model (Modelo de Capacidad de Madurez) para software (CMM 1.0). Durante el mismo intervalo de tiempo, la Comisión Europea patrocino un proyecto llamado BOOTSTRAP, con el objetivo de acelerar la aplicación de técnicas de ingeniería del software a la industria del software europea. Por su parte, ISO e IEC crean un Joint Technical Commitee (JTC1) en Tecnologías de la Información, en junio de 1989. El proyecto SPICE (Software Process Improvement and Capability dEtermination) es una actividad del WG 10 del Subcomité 7 del ISO/IEC JTC1, un estándar internacional para procesos de desarrollo software que provee un marco de trabajo uniforme para gestión e ingeniería del software. El número de modelos y estándares ha seguido creciendo, dando lugar a un panorama ‘fangoso’ para una empresa que tuviera que escoger un modelo para la evaluación y mejora de su proceso de desarrollo

En la actualidad existen más de 250 diferentes estándares de ingeniería de software elaborados por diferentes organismos de estandarización, todos con diferentes grados de detalle, cobertura, y aplicabilidad. Generalmente, el propósito, el enfoque y el

nivel de adaptabilidad de estos estándares varia grandemente, lo que dificulta el proceso de selección de los estándares adecuados a una organización.

La Figura 1.1, reproducida del Software Productivity Consortioum, muestra parcialmente el estado actual de los principales estándares internacionales de calidad de software. Actualmente los principales estándares y normas de calidad son el CMM, el ISO 9000, y el ISO 15504. ¡Y faltan algunos como Trillium o BOOTSTRAP!

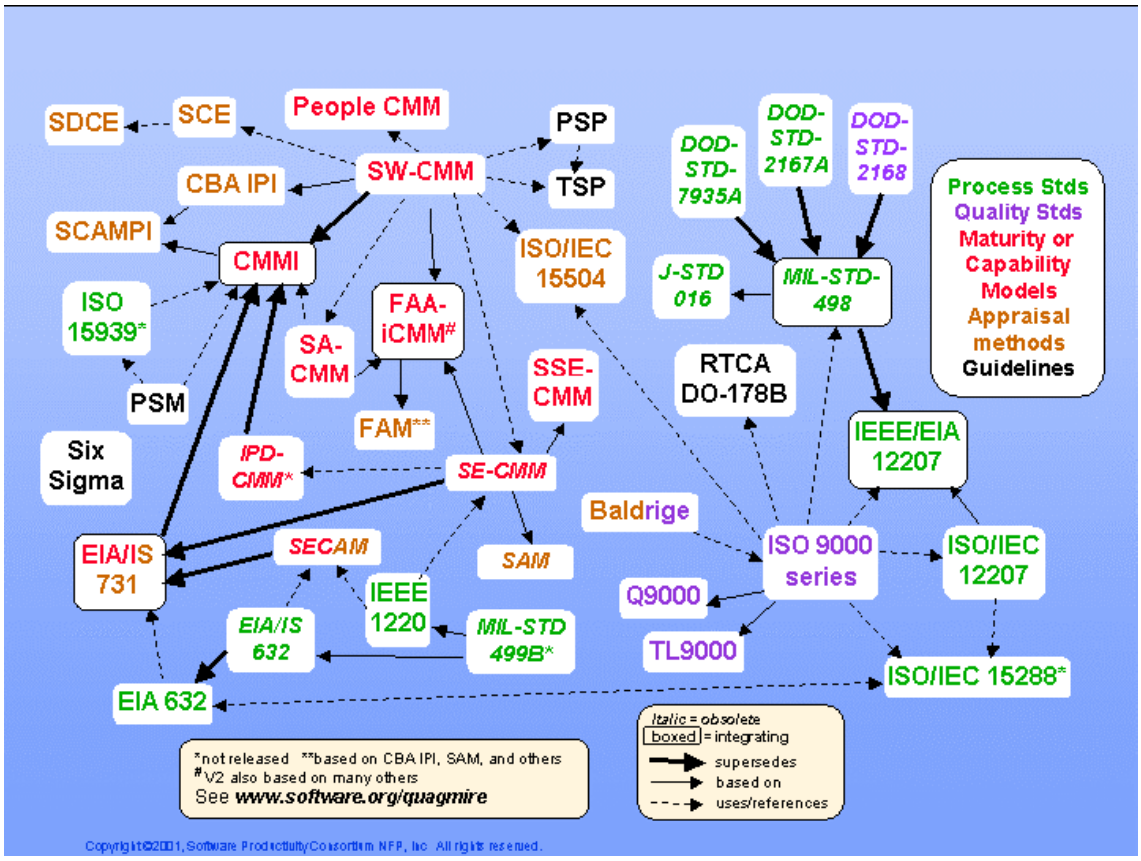


Figura 1.1. Los principales estándares internacionales de calidad de software.

De acuerdo con inspecciones y encuestas recientes en todo el mundo, el estándar ISO 9001 es el más popular en el mundo de la ingeniería del software, seguido de CMM e ISO/IEC 15504 (SPICE).

1.1 Modelos y Estándares de Calidad de Software.

1.1.1 ISO 9001

ISO 9001 es un conjunto estándares internacionales para sistemas de calidad. Diseñado para la gestión y aseguramiento de la calidad, especifica los requisitos básicos para el desarrollo, producción, instalación y servicio a nivel de sistema y a nivel de producto.

Su primera publicación fue en 1987, revisado en 1994 y actualizado en el año 2000 (con el compromiso de hacer una revisión cada 5 años). La primera versión de 1994 establecía un conjunto básico de requisitos para el establecimiento y

mantenimiento de la gestión del sistema y aseguramiento de la calidad de la ingeniería de software. Se concibe como una metodología de procesos basada en una lista de comprobaciones o requisitos a cumplir, umbral de calidad, valorado apto o no apto. Y esta simplicidad es lo que lo ha hecho mundialmente extendido.

La retroalimentación de los usuarios, el desarrollo de los modelos de evaluación y mejora continua y las críticas especializadas hacen que se requiera un estándar que:

- Emplee una aproximación de gestión basada en el proceso.
- Sea compatible con otros sistemas de gestión.
- Incluya requisitos papa la mejora continua del sistema de calidad.
- Coincida con las necesidades de los participantes externos.
- Sea amigable al usuario y al cliente.

La nueva familia de estándares es la siguiente:

- ISO 9000, Normas para la gestión y garantía de la calidad.
- ISO 9001, Modelo para la garantía de la calidad en diseño/desarrollo, producción, instalación y servicio.
- ISO 9002, Modelado para garantizar la calidad en producción y servicios.
- ISO 9003, Modelos para garantizar la calidad en inspección final y pruebas.
- ISO 9004, Elementos y gestión del sistema de calidad, Directrices para la mejora del rendimiento.
- ISO 9011, Directrices para la auditoria de los sistemas de gestión de la calidad y/o ambiental.

ISO 9001 e ISO 9004 se han desarrollado como un par coherente de normas, complementándose. Mientras ISO 9001 se centra en la eficacia del sistema de gestión de la calidad para dar cumplimiento a los requisitos del cliente, ISO 9004 se recomienda para organizaciones que persiguen la mejora continua, sin afán certificador.

El estándar se basa en un conjunto de Principios de Gestión de la Calidad: Enfoque al cliente, Liderazgo, Implicación de todo el personal, Enfoque a procesos, Enfoque del sistema hacia la gestión, Mejora continua, Enfoque objetivo hacia la toma de decisiones y Relaciones mutuamente beneficiosas con los proveedores.

Las cinco secciones en que se divide ISO 9001:2000 son:

1. QMS Sistema de Gestión de la Calidad (Requisitos generales y Requisitos de la documentación)
2. Responsabilidad de la Gestión (Compromiso de la dirección, Enfoque al cliente, Política de la calidad, Planificación,…).
3. Gestión de los Recursos (Provisión de recursos, Recursos humanos, Infraestructura, Ambiente de trabajo).
4. Realización del Producto (Planificación de la realización del producto, Procesos relacionados con los clientes, Diseño y desarrollo, Compras, Prestación del servicio).
5. Medición Análisis y Mejora (Generalidades, Supervisión y Medición, Control de servicio no-conforme, Análisis de datos, Mejora).

Cabe mencionar que el enfoque de ISO 9000 está en los procesos de “producción”, o sea de desarrollo en el caso de software. El fin de este tipo de estándar

es un mejor proceso, y no un mejor producto, excepto como consecuencia de mejores procesos.

1.1.2 ISO 9126

La ISO, bajo la norma ISO-9126, ha establecido un estándar internacional para la evaluación de la calidad de productos de software el cual fue publicado en 1992 con el nombre de "*Information tecnology – Software product evaluation: Quality characteristics and guidelines for their use*", en el cual se establecen las caracteristicas de calidad para productos de software.

El estándar ISO-9126 establece que cualquier componente de la calidad del software puede ser descrito en términos de una o más de seis caracteristicas básicas, las cuales son: funcionalidad, con fiabilidad y portabilidad; cada una de las cuales se detalla a través de un conjunto de subcaracterísticas que permiten profundizar en la evaluación de la calidad de productos de software. La tabla 1.1 muestra la pregunta central de atiende cada una de estas características.

Características	***Pregunta central***
Funcionalidad	¿Las funciones y propiedades satisfacen las necesidades explícitas e implícitas; esto es, el que . . .?
Confiablidad	¿Puede mantener el nivel de rendimiento, bajo ciertas condiciones y por cierto tiempo?
Usabilidad	¿El software es fácil de usar y de aprender?
Eficiencia	¿Es rápido y minimalista en cuanto al uso de recursos?
Mantenibilidad	¿Es fácil de modificar y verificar?
Portabilidad	¿Es fácil de transferir de un ambiente a otro?

Tabla 1.1. Caracteristicas de ISO-9126 y aspecto que atiende cada una.

1.1.3 ISO 15540 (SPICE , Software Process Improvement and Capability dEtermination)

ISO/IEC 15504 es un emergente estándar internacional de evaluación y determinación de la capacidad y mejora continua de procesos de ingeniería del software, con la filosofía de desarrollar un conjunto de medidas de capacidad estructuradas para todos los procesos del ciclo de vida y para todos los participantes. Es el resultado de un esfuerzo internacional de trabajo y colaboración y tiene la innovación, en comparación con otros modelos, del proceso paralelo de evaluación empírica del resultado.

En 1991 [49], ISO/IEC JTC1/SC7 aprueba un estudio para investigar la necesidad y los requisitos para un estándar de evaluación del proceso software, llegando a la conclusión (1992) de que había consenso internacional. El proceso de desarrollo y validación empírica (proyecto SPICE) se ha alargado diez años. En 1998 se publica la primera versión del estándar como Informe Técnico (en 1995 se publica como 'borrador'), evolucionando posteriormente hasta Estándar Internacional, con la realización de tres fases de pruebas, la Fase 1 (1995) con la idea de validar la decisiones de diseño y usabilidad del borrador, la Fase 2 (1996-1998) que a los objetivos

anteriores sumaba proveer de una guía de aplicación y revisar la consistencia, validez, adecuación, usabilidad y portabilidad de SPICE. La Fase 3 (hasta marzo de 2003, en que se cierra el proyecto SPICE) se realiza con la idea de aportar entradas y publicar el estándar ISO. Tras los Trials comienza la fase de Benchmarking (actual fase), con la idea de recolectar datos de los procesos de evaluación y analizarlos y comienza la publicación de partes del estándar.

La versión 1.0 inicialmente recogía treinta y cinco procesos agrupados en cinco categorías (Cliente-Proveedor, Ingeniería, Proyecto, Soporte y Organización). Sin embargo, la idea de expandir el ámbito de aplicación del estándar evitando restringirlo a un determinado ciclo de vida, la compatibilidad con ISO/IEC 12207 e ISO/IEC 15288 y con cualquier modelo posterior, permite la evolución del estándar para aceptar Modelos de Referencia de Procesos (PRM's) eliminando la inicial dimensión de procesos. La medida de capacidad (véase tabla 1.2), es aplicable a cualquier modelo de procesos plasmado en un PRM compatible con ISO 12207. Esto le confiere una infraestructura mucho más abierta, facilitando la compatibilidad.

Id.	Nivel de Capacidad	Atributos de Proceso y Descripción
CL[0]	Incompleto	El proceso no está implementado o falla en alcanzar su proposito. No es fácil identificar los productos o salidas de los procesos.
CL[1]	Realizado	El proposito del proceso se logra generalmente, aunque no sea rigurosamente planificado ni llevado a cabo. Hay productos identificables que testifican el alcance del proposito.
	PA.1.1	Realización del Proceso.
CL[2]	Gestionado	El proceso es gestionado y los entregables resultado de procedimientos específicos, planificados y seguidos, con requisitos de calidad, tiempo y recursos.
	PA.2.1 PA.2.2	Gestión de la Realización. Gestión de los Productos del trabajo.
CL[3]	Establecido	Un proceso realizado y gestionado usado un proceso definido, basado en un principios de buenas prácticas de ingeniería del software.
	PA.3.1 PA.3.2	Definición del Proceso. Despliegue del Proceso.
CL[4]	Predecible	El proceso definido es puesto consistentemente en práctica dentro de límites de control establecidos para alcanzar metas del proceso ya definidas. Entendimiento cuantitativo de la capacidad del proceso y habilidad mejorada de predecir y gestionar el rendimiento.
	PA.4.1 PA.4.2	Medición del Proceso. Control del Proceso.
CL[5]	En optimización	Realización del proceso optimizada en la busqueda de las necesidades actuales y futuras del negocio. Objetivos cuantitativos de eficiencia y efectividad se establecen en función de los objetivos de la organización. Optimización puede llevar a estudiar y adoptar ideas innovadoras o productos tecnológicos novedosos que incluyan y modifiquen el proceso definido.
	PA.5.1 PA.5.2	Innovación del Proceso. Optimización del proceso.

Tabla 1.2. Modelo de Capacidad de Procesos.

1.1.4 BOOTSTRAP

La metodología BOOTSTRAP ha sido desarrollada para asegurar la conformidad con el estándar del ISO emergente para evaluación de software y mejora (SPICE) y alinear la metodología con la ISO 12207 "Tecnología de Información – Procesos de Ciclo de Vida de sofware". La metodología de BOOTSTRAP puede ser

aplicada a pequeñas y medianas compañias de software o departamentos de software dentro de una organización grande.

Los metodología BOOTSTRAP tiene los siguientes objetivos:

- Proporcionar apoyo a la evaluación de capacidad de proceso entre las mejores practicas de una reconocida ingenieria de software.
- Incluir estandares de software reconocida internacionalmente como fuentes para identificar las mejores practicas.
- Apoyar la evaluación de como el estandar de referencia ha sido puesta en práctica en la organización.
- Asegurar la fiabilidad de la evaluación.
- Identificar, en la organización, procesos fuertes y debiles.
- Apoyar planificación de mejora con resultados convenientes y confiables.
- Apoyar el logro de los objetivos de la organización planeando acciones de mejora.
- Ayudar a aumentar la eficiencia de los proceso poniendo en practica la exigencias de estandares en la organizacion.

Las principales caracteristicas de BOOTSTRAP

El proceso de evaluación: el proceso de evaluación es parte de la mejora. Los resultados de evaluación proporcionan la entrada principal para el plan de acción de mejora y proporciona la reaccipon de las actividades de mejora puestas en práctica. Durante una evaluación BOOTSTRAP los procesos organizativos son evaluacdos para definir cada proceso. La evaluación de capacidad de proceso está basada en el modelo de proceso de BOOTSTRAP.

El modelo de proceso: el modelo de proceso de BOOTSTRAP define niveles de capacidad y procesos. La capacidad de procesos son badasos en los siguientes niveles de capacidad:

- Nivel 0: Proceso Incompleto
- Nivel 1: Proceso de Desempeño
- Nivel 2: Proceso Administrativo
- Nivel 3: Proceso Establecido
- Nivel 4: Proceso Previsible
- Nivel 5. Proceso de Optimización

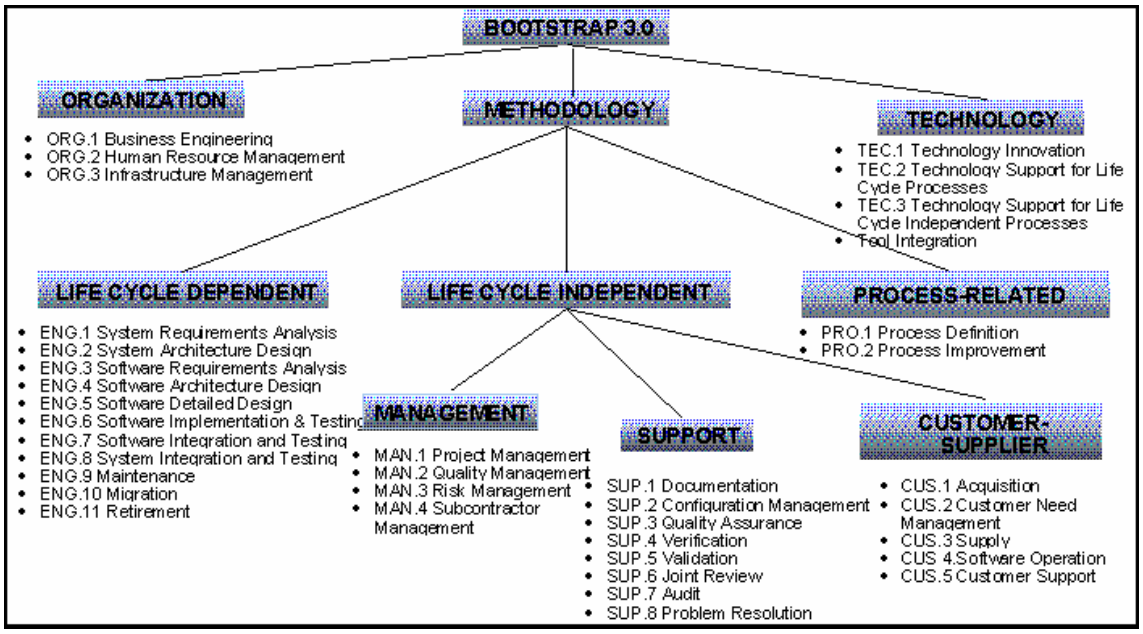


Figura 1.2. representa la estructura de árbol del modelo de proceso BOOTSTRAP.

Cuestionarios: una parte principal de la evaluación es reunir datos. La metodología de BOOTSTRAP proporciona dos cuestionarios, uno para juntra datos sobre la organización de desarrollo de software y el otro para juntar datos sobre el proyecto.

Tanteando, la posición y representación de resultados: los resultados de evaluación son la base para la planificación de mejora de procesos, pero este paple puede ocurrir sólo si los datos de evaluación son confiables y proporcionan una representación buena de la capacidad de la organización. La fiabilidad y la repetibilidad son obtenidas por:

- Asegurar que los asesores tienen el mismo background y usan el mismo acercamiento (este es garantizado por el proceso de acreditación del asesor de BOOTSTRAP), y
- Aplicando tanteo preciso y posición de reglas.

El BOOTSTRAP evalúa cada práctica basada por una escala valor cuatro (no, parcialmente, en gran parte y totalmente adecuado). El nivel de pacacidad es contado entonces en dos modos:

- Sobre una base algorítmica mostrando quartiles dentro de cada nivel;
- Aplicando las reglas de SPICE para sacar la posición del nivel de capacidad.

Los resultados de evaluación finales están listos con un instrumento y presentados como perfiles con quartiles y un perfil de SPICE. El perfil de capacidad es producido en organizativo y nivel de proyecto.

Pautas de mejora de proceso. Estas pautas apoyan la identificación de procesos que tienen el mayor impacto en el logro de los objetivos de la organización, entonces las prioridades de mejora son adjudicadas a procesos con capacidad baja e impacto alto. Los objetivos de mejora y las prioridades son evaluados en la luz de los riesgos de portencial a la organización en no conseguir sus objetivos. Este ayuda a prorporcionar una razón fundametal para emprender la dirección hacia el esfuerzo de mejora.

1.1.5 MoProSoft

Modelo de Procesos para la Industria del Software. Modelo para la mejora y evaluación de los procesos de desarrollo y mantenimiento de sistemas y productos de software. Desarrollado por la Asociación Mexicana para la Calidad en Ingeniería de Software a través de la Facultad de Ciencias de la Universidad Nacional Autónoma de México (UNAM) y a solicitud de la Secretaría de Economía para obtener una norma mexicana que resulte apropiada a las características de tamaño de la gran mayoría de empresas mexicanas de desarrollo y mantenimiento de software. Moprosoft es el nombre del modelo en la comunidad universitaria y profesional, y la norma técnica a la que da contenido es la NMX-059/01-NYCE-2005 que fue declarada Norma Mexicana el 15 de agosto de 2005 con la publicación de su declaratoria en el Diario de la Federación.

Moprosoft considera que los modelos de evaluación y mejora CMMI e ISO/IEC 15504 no resultan apropiados para empresas pequeñas y medianas de desarrollo y mantenimiento de software. Sobre las áreas de procesos de los niveles 2 y 3 del modelo SW-CMM e inspirándose en el marco de ISO/IEC 15504 se ha desarrollado este modelo.

La adopción del modelo permite elevar la capacidad de las organizaciones que desarrollan o mantienen software para ofrecer servicios con calidad y alcanzar niveles internacionales de competitividad. Es también aplicable en áreas internas de desarrollo de software de las empresas de diversos giros.

Características de MoProSoft

- Es específico para el desarrollo y mantenimiento de software.
- Es sencillo de entender y adoptar.
- Facilita el cumplimiento de los requisitos de otros modelos como ISO 9000:2000, CMM y CMMI.
- Se enfoca a procesos.
- Se le considera práctico en su aplicación, principalmente en organizaciones pequeñas, con bajos niveles de madurez.
- Comprende un documento de menos de 200 páginas que, al compararlo con otros modelos y estándares, lo hace bastante práctico.
- Resulta acorde con la estructura de las organizaciones mexicanas de la industria de software.
- Está orientado a mejorar los procesos, para contribuir a los objetivos de negocio, y no simplemente ser un marco de referencia o certificación.
- Tiene un bajo costo, tanto para su adopción como para su evaluación.

Moprosoft identifica los procesos empleados por las empresas de desarrollo y mantenimiento de software y los clasifica en tres categorías:

Categoría de alta dirección (DIR) Aborda las prácticas de la alta dirección relativas a la gestión del negocio. Proporciona alineación a los procesos de la categoría de gerencia (GER) y se retroalimenta de la información que éstos generan.

Categoría de Gerencia (GER) Aborda las prácticas de gestión de procesos, proyectos y recursos en función de las alineaciones establecidas a través de los procesos de alta dirección (DIR). Proporciona los elementos para el funcionamiento de los procesos de la siguiente categoría (Operación), recibe y evalúa la información que generan, y comunica los resultados a los procesos de alta dirección.

Categoría de Operación (OPE) Aborda las prácticas para los proyectos de desarrollo y mantenimiento de software. Los procesos de esta categoría realizan las actividades de acuerdo con los elementos proporcionados por los de gerencia, y remite a ésta la información y los productos generados.

MoProSoft es un modelo integrado donde las salidas de un proceso están claramente dirigidas como entradas a otros; las prácticas de planeación, seguimiento y evaluación se incluyeron en todos los procesos de gestión y administración; por su parte los objetivos, los indicadores, las mediciones y las metas cuantitativas fueron incorporados de manera congruente y práctica en todos los procesos; las verificaciones, validaciones y pruebas están incluidas de manera explícita dentro de las actividades de los procesos; y existe una base de conocimientos que resguarda todos los documentos y productos generados.

Categoría	Proceso	Propósito
Alta Dirección	Gestión de Negocio	Establecer la razón de ser de la organización, sus objetivos y las condiciones para lograrlos, para lo cual es necesario considerar las necesidades de los clientes, así como evaluar los resultados para poder proponer cambios que permitan la mejora continua. Adicionalmente habilita a la organización para responder a un ambiente de cambio y a sus miembros para trabajar en función de los objetivos establecidos.
Gestión	Gestión de Procesos	Establecer los procesos de la organización, en función de los procesos requeridos identificados en el Plan Estratégico. Así como definir, planificar e implantar las actividades de mejora en los mismos.
Gestión	Gestión de Proyectos	Asegurar que los proyectos contribuyan al cumplimiento de los objetivos y estrategias de la organización.
Gestión	Gestión de Recursos	Conseguir y dotar a la organización de los recursos humanos, infraestructura, ambiente de trabajo y proveedores, así como crear y mantener la Base de Conocimiento de la organización. La finalidad es apoyar el cumplimiento de los objetivos del Plan Estratégico de la organización. Las actividades de este proceso se apoyan en tres subprocesos: - Recursos humanos y ambiente de trabajo. - Bienes, servicios e infraestructura. - Conocimiento de la organización.
Operación	Administración de Proyectos Específicos	Establecer y llevar a cabo sistemáticamente las actividades que permitan cumplir con los objetivos de un proyecto en tiempo y costo esperados.

Operación	Desarrollo y Mantenimiento de Software	Realización sistemática de las actividades de análisis, diseño, construcción, integración y pruebas de productos de software nuevos o modificados cumpliendo con los requerimientos especificados.

Tabla 1.3. *Modelo de Procesos para la Industria de Software MoProSoft*, versión 1.3

La implantación de MoProSoft no demanda la incorporación de personal especializado en las empresas, únicamente requiere de una adecuada capacitación del personal existente.

1.1.6 CMM (Capability Maturity Model)

SEI (Software Engineering Institute), un auténtico peso pesado en la normalización de los procesos del software, desarrolló su línea de trabajo sobre el concepto de "madurez" de las organizaciones para producir software.

Por madurez se entiende la capacidad que tiene la organización para asegurar la calidad de sus proyectos (fecha, coste y funcionalidad), la homogeneidad (siempre y en todos sus proyectos) y la capacidad de aprendizaje de sus propia experiencia y su aplicación en la mejora continua.

Como resultado de estas líneas de trabajo, en 1990 publicó el modelo de madurez de la capacidad para el desarrollo de software (CMM-SW), que tras más de una década de existencia ha demostrado que en muchas organizaciones ha resultado eficaz. Este modelo crea una escala de cinco niveles para determinar la madurez de una organización.

- Nivel 1.- INICIAL

  Pertenecen a él las empresas que no basan el desarrollo en procesos definidos, y no aplican técnicas de gestión de proyectos. El modelo se refiere a entornos de producción caóticos con técnicas de programación heroica cuyos resultados no son predecibles y dependen exclusivamente de la valía de las personas.
- Nivel 2.- REPETIBLE.

  Define a las organizaciones que aplican técnicas de gestión de proyectos, aunque no dispongan de procesos definidos.
- Nivel 3.- DEFINIDO

  Pertenecen a él las organizaciones que disponen de procesos definidos con precisión y ejecutados de forma regular. Las empresas con este nivel de madurez examinan la experiencia de los proyectos que realizan y emplean las lecciones aprendidas para mejorar sus procesos.
- Nivel 4.- GESTIONADO

  En el cuarto nivel de madurez se sitúan las organizaciones que han depurado el análisis de los proyectos realizados, hasta institucionalizarlo como procesos que miden cuantitativamente la capacidad de los procesos de desarrollo, de forma que pueden predecir de forma cuantificable los resultados, y evaluar las mejoras con mediciones objetivas.
- Nivel 5.- OPTIMIZADO

Las organizaciones con un nivel 5 de madurez tienen definidos, y practican de forma institucionalizada, procesos de mejora continua que se nutren con la información cuantificada de los procesos del nivel 4.

CMM dirige su enfoque a la mejora de procesos en una organización, estudia los procesos de desarrollo y produce una evaluación de la madurez (indicador para medir la capacidad para construir un software de calidad) de la organización según una escala de cinco niveles (inicial, repetible, definido, dirigido y optimizado). Los modelos contienen los elementos esenciales de procesos efectivos para una o más disciplinas y describen el camino para evolucionar y mejorar desde procesos inmaduros a procesos disciplinados, maduros con calidad y eficiencia mejorada y probada.

Propiciado por su rápido éxito y por demanda de modelos en otros ámbitos, se publica una pléyade de modelos para otras disciplinas y funciones: People CMM (1995), Systems Engineering CMM (1995), Integrated Product Development (1996), Software Acquisition CMM, FAA-CMM, Trillium,…

Mientras algunas organizaciones encontraban estos modelos útiles, también encontraban que se solapaban sobremanera, que a veces eran contradictorios, escasamente limpios con interfaces ininteligibles, escasa estandarización y mezclando diferentes niveles de detalle.

El proyecto de integración de CMM o CMMI fue puesto en marcha para desarrollar un marco de trabajo simple para la mejora de procesos, para organizaciones que persiguen la mejora en todos los ámbitos y niveles de la empresa.

En la representación por etapas, se da un mapa predefinido, dividido en etapas (los niveles de madurez), para la mejora organizacional basada en procesos probados, agrupados y ordenados y sus relaciones asociadas. Cada nivel de madurez tiene un conjunto de áreas de proceso que indican donde una organización debería enfocar la mejora de su proceso. Cada área de proceso se describe en términos de prácticas que contribuyen a satisfacer sus objetivos. Las prácticas describen las actividades que más contribuyen a la implementación eficiente de un área de proceso; se aumenta el 'nivel de madurez' cuando se satisfacen los objetivos de todas las áreas de proceso de un determinado nivel de madurez.

N. de madurez de la organiz.	Centrado en	Areas de Proceso	Categoría
5. Optimizado	Mejora continua del proceso	-Análisis y resolución de causas de desviaciones. -Innovación y despliegue a toda la organización	Soporte G. Proceso
4. Gestionado cuantitativamente	Control cuantitativo del proceso	-Gestión cuantitativa de los proyectos. -Entendimiento cuantitativo del rendimiento de los procesos de la organización.	G. Proyecto G. Proceso
3. Definido	Proceso caracterizado por la organización y proactivo	-Desarrollo de los requisitos -Soluciones técnicas -Integración de productos -Verificación -Validación -Enfoque de procesos en organización -Definición de procesos en organización. -Entrenamiento y formación -Gestión integrada de proyectos -Gestión del riesgo	Ingeniería Ingeniería Ingeniería Ingeniería Ingeniería G. Proceso G. Proceso G. Proceso G. Proyecto G. Proyecto

		-Análisis y resolución de las decisiones -Entorno organizativo para la integración -Equipo para desarrollo integrado	Soporte Soporte G. Proyecto
2. Gestionado	Gestión básica del proyecto	-Gestión de requisitos -Planificación de proyectos -Monitorización y control de proyectos -Gestión de acuerdos con proveedores. -Medición y análisis -Aseguramiento de la calidad del producto y del proceso -Gestión de la configuración	Ingeniería G. Proyecto G. Proyecto G. Proyecto Soporte Soporte Soporte
1. Inicial	Proc. impredecible, control reactivo		

Tabla 1.4. Áreas en representación por etapas y en representación continúa.

En la representación continua, enfocamos la capacidad de cada área de proceso para establecer una línea a partir de la que medir la mejora individual, en cada área. Al igual que el modelo por etapas, el modelo continuo tiene áreas de proceso que contienen prácticas, pero éstas se organizan de manera que soportan el crecimiento y la mejora de un área de proceso individual.

Comparando ISO 9001 y la CMM parece ser que, más o menos, la certificación ISO 9001 es equivalente a estar en los niveles 2 ó 3 de la CMM, pero con la ventaja de que CMM es un modelo incremental más a largo plazo y con mejores resultados.

1.2 Porque es importante usar Modelos y Estándares de Calidad de software.

Dada la competitividad global que es mas fuerte cada día, es necesario que las micro, pequeñas y grandes empresas se preocupen en mejorar sus productos, para competir y tener mas éxito de su producto. Pero la calidad del producto no solo se mide al terminarlo. La complejidad de los problemas que hoy en día buscan una solución por software ha aumentado de manera considerable y con ello se pretende que el software sea capaz de dar una mejor solución a tales problemas. Pero este crecimiento ha sobrepasado la habilidad de desarrollar y mantener el software por parte de las organizaciones dedicadas a desarrollar o darle mantenimiento.

Ahora las empresas se enfrentan a una situación con dos problemas. La primera las organizaciones quieren ser capaces de desarrollar y entregar su software de manera confiable, segura, a tiempo y apegado al costo final acordado con el cliente. El segundo es la perspectiva del cliente, quien le interesa saber con mucha mayor certeza que todo lo anterior se cumplirá. Por tal motivo las organizaciones deben de adoptar una norma, estándar o modelo que pueda ayudarlas a conseguir la calidad de sus productos y con ello garantizarle al cliente que todo lo establecido por dicho software se llevara acabo y gracias a esto poder tener una mucha mejor competitividad en el mercado.

Aunque no solo la competitividad es la única razón por la cuál se debe de buscar la calidad en el software. Tienen las organizaciones de desarrollo de software que darle la importancia a cada programa que se desarrolla. Deben tomar conciencia y

responsabilidad de las consecuencias que un defecto en su producto final podría ocasionar. Algunos de estos defectos de software pueden ocasionar daños muy graves a los clientes y hasta perjudicar a personas físicamente.

El problema actual de los sistemas es que son más rápidos, complejos y automáticos. Una posible falla catastrófica aumenta a la par de las anteriores características y con ello el potencial del daño que ocasionaría al cliente. Así que las organizaciones dedicadas al desarrollo de software deben de distinguir entre simple y fácil. Un error simple no siempre es fácil de hallar, por lo tanto todos están involucrados en la calidad del producto final tanto la organización como el cliente.

El aspecto mucho más complicado es el factor económico, pues cada defecto representa un costo adicional. Un error identificado dentro de la fase de desarrollo es mucho mas barato de resolver que el mismo defecto en una fase posterior, y aun más caro sí este sale a la luz después de que el producto ya ha sido entregado. Por esa razón se debe tener un proceso de control de calidad para prever dichos defectos y hacer un entregable con calidad y la confianza de tener un mejor tratado de defectos y prever daños a los clientes.

Pero se debe tomar en cuenta que la implementación de algún modelo no solo involucra el seguir los puntos o requerimientos de cada una de ellos. El tener un proceso y mejores prácticas no sirven de nada sino se llevan a cabo. La norma por si misma no aportara avances sino existe un compromiso de toda la organización, desde la gerencia mas alta, hasta cada uno de los integrantes de la organización.

Las prácticas deben institucionalizarse, la gente debe tener la capacidad y ser responsable de seguir cada una de las prácticas que están definidas para toda la organización. Para poder ayudar a la gente a dar seguimiento a las prácticas correspondientes al proceso. Es necesario establecer que es lo que se va hacer, por quien y cuando, así como el ciclo de vida de los procesos.

Por lo anterior una organización de desarrollo de software debe adoptar algún estándar, norma o modelo, para tener una mejor competitividad, calidad y aseguramiento de sus productos finales y sobre todo la satisfacción completa del cliente.

1.3 Relación de UML en términos de los modelos y estándares de calidad de software.

UML son las siglas de Lenguaje de Modelado Unificado (Unified Modeling Language). UML es un lenguaje estándar que sirve para escribir los planos del software, puede utilizarse para visualizar, especificar, construir y documentar todos los artefactos que componen un sistema con gran cantidad de software. Se puede usarse para modelar desde sistemas de información hasta aplicaciones distribuidas basadas en Web, pasando por sistemas empotrados de tiempo real. UML es solamente un lenguaje por lo que es sólo una parte de un método de desarrollo software, es independiente del proceso aunque para que sea optimo debe usarse en un proceso dirigido por casos de uso, centrado en la arquitectura, iterativo e incremental.

UML es un lenguaje por que proporciona un vocabulario y las reglas para utilizarlo, además es un lenguaje de modelado lo que significa que el vocabulario y las reglas se utilizan para la representación conceptual y física del sistema.

UML es un lenguaje que nos ayuda a interpretar grandes sistemas mediante gráficos o mediante texto obteniendo modelos explícitos que ayudan a la comunicación durante el desarrollo ya que al ser estándar, los modelos podrán ser interpretados por personas que no participaron en su diseño (e incluso por herramientas) sin ninguna ambigüedad. En este contexto, UML sirve para *especificar*, modelos concretos, no ambiguos y completos.

El Lenguaje Unificado de Modelado prescribe un conjunto de notaciones y diagramas estándar para modelar sistemas orientados a objetos, y describe la semántica esencial de lo que estos diagramas y símbolos significan. Mientras que ha habido muchas notaciones y métodos usados para el diseño orientado a objetos, ahora los modeladores sólo tienen que aprender una única notación.

UML se puede usar para modelar distintos tipos de sistemas: sistemas de software, sistemas de hardware, y organizaciones del mundo real. UML ofrece nueve diagramas en los cuales modelar sistemas.

- Diagramas de Casos de Uso para modelar los procesos “business”.
- Diagramas de Secuencia para modelar el paso de mensajes entre objetos.
- Diagramas de Colaboración para modelar interacciones entre objetos.
- Diagramas de Estado para modelar el comportamiento de los objetos en el sistema.
- Diagramas de Actividad para modelar el comportamiento de los Casos de Uso, objetos u operaciones.
- Diagramas de Clases para modelar la estructura estática de las clases en el sistema.
- Diagramas de Objetos para modelar la estructura estática de los objetos en el sistema.
- Diagramas de Componentes para modelar componentes.
- Diagramas de Implementación para modelar la distribución del sistema.

UML proporciona la capacidad de modelar actividades de planificación de proyectos y de sus versiones, expresar requisitos y las pruebas sobre el sistema, representar todos sus detalles así como la propia arquitectura. Mediante estas capacidades se obtiene una documentación que es valida durante todo el ciclo de vida de un proyecto.

Aun así, UML no prescribe un proceso o método estándar para desarrollar un sistema. Para producir software que cumpla su propósito hay que obtener los requisitos del sistema, esto se consigue conociendo de una forma disciplinada a los usuarios y haciéndolos participar de manera activa para que no queden “cabos sueltos”. Para conseguir un software de calidad, que sea duradero y fácil de mantener hay que idear una sólida base arquitectónica que sea flexible al cambio. Para desarrollar software rápido y eficientemente, minimizando el trabajo de recodificación y evitando crear miles de líneas de código inútil hay que disponer, además de la gente y las herramientas necesarias, de un enfoque apropiado.

Para conseguir, que a la hora de desarrollar software de manera industrial se obtenga un producto de calidad, es completamente necesario seguir ciertas pautas y no abordar los problemas de manera somera, con el fin de obtener un modelo que represente lo suficientemente bien el problema que hemos de abordar. El modelado es la espina dorsal del desarrollo software de calidad. Se construyen modelos para poder comunicarnos con otros, para explicar el comportamiento del sistema a desarrollar, para comprender, nosotros mismos, mejor ese sistema, para controlar el riesgo y en definitiva para poder atacar problemas que sin el modelado su resolución seria imposible, tanto desde el punto de vista de los desarrolladores (no se pueden cumplir los plazos estimados, no se consigue ajustar los presupuestos...) como desde el punto de vista del cliente, el cual, si finalmente se le entrega el producto del desarrollo, se encontrará con infinidades de problemas, desde que no se cumplen las especificaciones hasta fallos que dejan inutilizado el sistema.

Por todas estas razones es inevitable el uso de modelos. Pero, ¿qué es un modelo?. La respuesta es bien sencilla, *un modelo es una simplificación de la realidad*. El modelo nos proporciona los planos de un sistema, desde los más generales, que proporcionan una visión general del sistema, hasta los más detallados. En un modelo se han de incluir los elementos que tengan más relevancia y omitir los que no son interesantes para el nivel de abstracción que se ha elegido. A través del modelado conseguimos cuatro objetivos:

1. Los modelos nos ayudan a visualizar cómo es o queremos que sea un sistema.
2. Los modelos nos permiten especificar la estructura o el comportamiento de un sistema.
3. Los modelos nos proporcionan plantillas que nos guían en la construcción de un sistema.
4. Los modelos documentan las decisiones que hemos adoptado.

En la realidad, no siempre se hace un modelado formal, la probabilidad de que exista un modelado formal para abordar un sistema es inversamente proporcional a la complejidad del mismo, esto es, cuanto más fácil sea un problema, menos tiempo se pasa modelándolo y esto es porque cuando hay de aportar una solución a un problema complejo el uso del modelado nos ayuda a comprenderlo, mientras que cuando tenemos un problema fácil el uso del modelado que hacemos se reduce a representar mentalmente el problema o, como mucho, a escribir unos cuantos garabatos sobre un papel.

La mayoría de estas organizaciones están adoptando e incorporando el UML como la notación orientada a objetos de sus metodologías. Algunos modeladores usarán un subconjunto de UML para modelar ’what they’re after’, por ejemplo simplemente el diagrama de clases, o solo los diagramas de clases y de secuencia con Casos de Uso. Otros usarán una suite más completa, incluyendo los diagramas de estado y actividad para modelar sistemas de tiempo real, y el diagrama de implementación para modelar sistemas distribuidos. Aun así, otros no estarán satisfechos con los diagramas ofrecidos por UML, y necesitarán extender UML con otros diagramas como modelos relacionales de datos y ’CRC cards’.

Capítulo 2

Detalles Generales de CMM-SW

2.1 Historia

A principios de la década de los 80's, una firma dedicada al estudio del mercado de Tecnologías de Información publica[1] un reporte sobre el éxito de los proyectos de desarrollo en la industria del software. El reporte, basado en encuestas hechas sobre proyectos de software, informaba los siguientes resultados estadísticos:

- El 30% de los proyectos se cancelaban
- El 54% de los proyectos excedían ampliamente los tiempos y costos estimados
- El 16% de los proyectos finalizaban exitosamente dentro del tiempo, el costo y la funcionalidad prevista.

En respuesta a la situación alarmante del momento, el Departamento de Defensa de los Estados Unidos propuso una solución a estos problemas. Como esta situación les parecía intolerable convocó un comité de expertos para que solucionase estos problemas, en el año 1983, dicho comité concluyó "Tienen que crear un instituto de la ingeniería del software, dedicado exclusivamente a los problemas del software, y a ayudar al Departamento de Defensa".

Convocaron un concurso público en el que dijeron: "Cualquiera que quiera enviar una solicitud tiene que explicar como van a resolver estos 4 problemas", se presentaron diversos estamentos y la Universidad Carnegie Mellon ganó el concurso en 1985, creando el SEI.

En 1991, el SEI publica el modelo CMM (Capability Maturity Model, en español Modelo de Capacidad de Madurez). El modelo está orientado a la mejora de los procesos relacionados con el desarrollo de software, para lo cual contempla las consideradas mejores prácticas de ingeniería de software y de gestión.

A partir de ese momento, el Departamento de Defensa (DoD) exige que sus proveedores estén acreditados en CMM, lo que impulsa a que el modelo tenga una amplia aceptación y se convierta en un estándar de facto dentro de la industria del software.

Como consiguiente los proveedores de DoD comenzaron a ver la mejora del proceso de software como una exigencia necesaria para hacer negocio con DoD y otras empresas, así como agencias del gobierno. Numerosas exigencias alrededor de la madures de procesos de software han sido requeridos por DoD y otras compañías a la comunidad de empresas dedicadas al ramo de la ingeniería de software.

El modelo CMM original define cinco niveles de madures dentro de los cuales se puede encontrar una organización:

[1] The Standish Group, http://www,standishgroup.com, pagina vigente al 30/01/2004

1. Nivel 1 – Inicial. el proceso de software es impredecible, sin control y reactivo. El éxito de los proyectos depende del talento de los individuos.
2. Nivel 2 – Repetible: existen procesos básicos de gestión los proyectos (costo, calendario, funcionalidad). Los procesos existen hacen que se pueda repetir éxitos en proyectos de similares características.
3. Nivel 3 – Definido: existe un proceso de software documentado y estandarizado dentro de la organización. Todos los proyectos utilizan una versión a medida del proceso.
4. Nivel 4 – Gestionado: la organización recolecta métricas del proceso de software y de los productos desarrollados. Tanto el proceso como los productos se entienden y controlan cuantitativamente.
5. Nivel 5 – Optimizado: existe una mejora continua del proceso del software, basada en realimentación cuantitativa del proceso y en la puesta en práctica de ideas t tecnologías innovadoras.

De acuerdo al modelo, el rendimiento en general de una organización mejora notablemente a medida que la misma incrementa su nivel de madurez. La figura 2.1 muestra de manera conceptual las mejoras en el rendimiento para casa nivel contemplando los factores tiempo y costo. Conclusiones similares se puedes extraer para otros factores como la funcionalidad y la calidad.

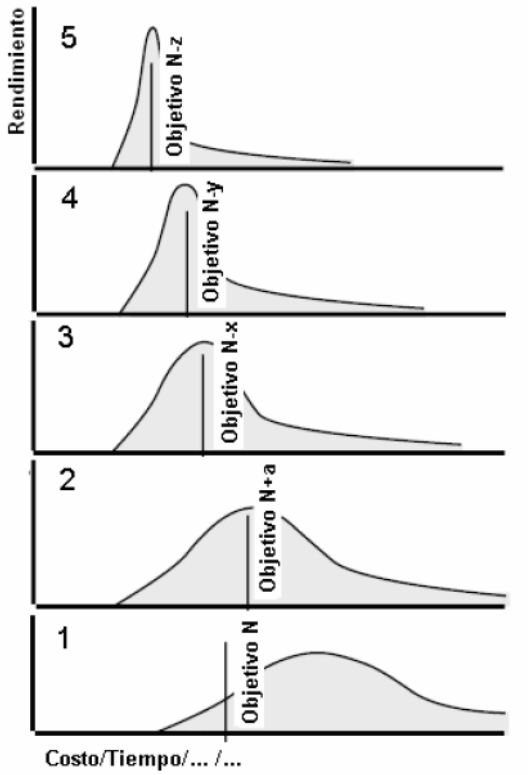


Figura 2.1. Variación del rendimiento con respecto a los objetivos fijados para los factores costo y tiempo, de acuerdo al nivel de CMM de la organización.

En la misma se puede observar lo siguiente:

- En las organizaciones que se encuentran en el nivel 1, los objetivos generalmente son ampliamente excedidos por la realidad.
- En las organizaciones que se encuentran en el nivel 2, se establecen objetivos más acordes a la realidad.
- En las organizaciones que se encuentran en los niveles 3, 4, y 5 existe una menos dispersión de la realidad con respecto a los objetivos, y el desempeño de mejora con cada nivel.

Algunas de las organizaciones que adoptaron el modelo fueron: Accenture, AT&T, Boeing, Ericsson, Fuji Seros, hewlett Packard, Hyundai, IBM, Motorota, Nasa, NCR, NEC, PriceWaterhouseCoopers, Samsung, Siemens, United Airlines, entre otras.

Luego del éxito alcanzado por CMM, el SEI desarrolló modelos similares para otras disciplinas, entre las cuales figurabas la ingeniería de sistemas (SE-CMM, Systems Engineering Capability Maturity Model), la adquisición de software (SA-CMM, Software Acquisition Capability Maturity Model), las personas (P-CMM, People Capability Maturity Model), y el desarrollo integrado de productos (IPD-CMM, Integrated Product Developer Capability Maturity Model).

A mediados de la decada de los 90's, el SEI decide unificar los modelos de ingeniería de software (SW-CMM, también conocido como CMM), de ingeniería de sistemas (SE-CMM) y de desarrollo integrado de productos (IPD-CMM), embarcándose en un esfuerzo que culmina en el año 2002 dando origen a una nueva versión llamada CMMI (Capability Maturity Model Integration).

CMMI brinda un marco con una estructura común para todas las disciplinas (ingeniería de software, ingeniería de sistemas, desarrollo integrado de productos, adquisición de productos) y agrega una nueva forma de representación además de la conocida representación por niveles y está orientada a medir la mejora en los procesos de manera individual en vez de hacerlo de manera conjunta como la representación por niveles. En paralelo con el desarrollo de CMMI, el SEI creo un método para la evaluación formal del modelo denominado SCAMPI (Standard CMMI Appraisal Method for Process Improvement).

Los resultados de una evaluación se obtienen mediante la aplicación de un conjunto de reglas de negocio aplicada a cada componente del modelo (prácticas, objetivos, áreas de proceso y niveles de madurez). Estas reglas hacen necesario utilizar herramientas, ya que el método de evaluación deja de ser una simple encuesta para convertirse en una evaluación detallada y casi matemática.

La situación actual con respecto a los modelos es la siguiente:

- El SEI ha iniciado la discontinuación gradual del modelo CMM original en diciembre del 2003, para finalizarla en el 2005.
- Las grandes organizaciones acreditadas en SW-CMM planean migrar a CMMI-SW.
- Muchas organizaciones pequeñas planean acreditarse en CMMI_SW, con el fin de poder acceder al mercado de las exportaciones.

- La preparación previa a la acreditación CMMI-SW es larga y costosa. Las organizaciones utilizan el concepto de “Evaluación interna” como paso preparatorio.
- Una “Evolución interna” es algo difícil de llevar a cado para las organizaciones recién iniciadas en el tema, y no existe un soporte adecuado de herramientas que le faciliten el cambio.

En resumen, el nuevo modelo trae un problema no trivial para las organizaciones, en lo referente a los costos y tiempos necesarios, para la preparación previa a su adopción o a una a acreditación. El problema se ve más acentuado en las organizaciones pequeñas, donde los recursos económicos, humanos y temporales suelen ser menores que en las grandes organizaciones.

2.2 Características de CMM

CMM provee una estructura conceptual para proporcionar el manejo y el desarrollo de productos de software en forma disciplinada y consistente. Se enfoca en la capacidad de las organizaciones de software para realizar productos de calidad de manera consistente y predecible. Con ello cubre prácticas de planeación, ingeniería, desarrollo de software y su mantenimiento, definiendo los niveles de madurez a las cuales debería de llegar una empresa para establecer una cultura de excelencia en ingeniería de software.

Otra característica importante de CMM es que se basa en gran parte de su efectividad en su educación. En todo el modelo frases como “de acuerdo a un proceso documentado” y siguiendo una “política organizacional escrita” son comunes entre los KPAs (Key Process Areas) de CMM. Siempre se hace énfasis en la necesidad de guardar información para su uso posterior en el proceso y el mejoramiento del mismo.

2.3 Estructura

La estructura del modelo de capacidad de madurez organiza sus prácticas complejas en unas cuantas categorías. Las divisiones principales del modelo son 5 niveles de madurez y cada nivel es descompuesto a su vez en un conjunto de áreas clave del proceso (KPA), las cuales están organizadas de acuerdo a sus características.

2.4 Niveles de Madurez

Un nivel de madurez es un sistema evolutivo y bien definido para alcanzar el proceso de madurez de software. Cada nivel de madurez tiene dentro de si mismo parámetros que permiten la mejora continua. Alcanzar un nivel dentro de la escala de CMM significa seguir en busca de mejores prácticas y a la vez mantener los logros alcanzados.

Los objetivos de cada nivel son varios:

- Definir un orden para medir la calidad de software gracias a la madurez de la compañía.
- Ayudar a la organización a ver que procesos debe de mejorar en forma gradual para alcanzar un nivel óptimo.
- Mantener un proceso bien documentado.
- Lograr un producto bien controlado, verificable, valido y medido.

Los niveles como anteriormente son: inicial, repetible, definido, administrado y óptimo que a continuación se describen en forma detallada.

2.4.1 Nivel 1. Inicial

Este nivel es el primer estado en la evolución de las organizaciones que desarrollan software. En este nivel se encuentran todas las empresas que no han logrado implementar las prácticas básicas de administración de proyectos e ingeniería de software definidas a partir algún nivel superior.

En este nivel es frecuente encontrar crisis dentro de la compañía. Esto se puede deber a diversos factores como comprometerse en los tiempos y alcances del proyecto, falta de comunicación en el equipo de desarrollo, etc. Cuando se presenta esta crisis, habitualmente el equipo deja de lado los procesos iniciales como el análisis y el diseño para dedicarse únicamente a codificar y si da tiempo realiza todas las pruebas necesarias para probar el software.

En estas empresas, el software es el producto del esfuerzo del talento individual de algunos de los miembros del equipo, como líderes de proyecto excepcionales o un efectivo equipo de desarrollo. Cada héroe crea su proceso con un estilo propio y suele ser necesario que la misma persona este involucrada en el siguiente proyecto para que pueda ser repetido. En muchas ocasiones los equipos realizan proyectos que cumplen los requerimientos necesarios sin entender claramente como lo lograron y otra lo logran pero exceden por mucho el presupuesto inicial o la calendarización para la entrega del producto final.

La administración tiene una participación en el proceso de este nivel, aunque generalmente es de forma ineficiente y puede llegar en muchas ocasiones caótica. Ellos ocupan una parte significativa de su tiempo en arreglar problemas y hablar con los clientes insatisfechos. Ante una situación de crisis permanente, se les hace difícil destinar recursos para definir o documentar los procesos, lo que lleva a un ciclo infinito. Normalmente no existe buena relación entre administrativos y desarrolladores debido a que hay muchos roces debido a la presión de la entrega o de los errores existentes.

En este nivel el proceso de software no está documentado y tiene un ambiente inestable en el desarrollo y mantenimiento. Existe una baja probabilidad de cumplir objetivos del proyecto y esto ocasiona problemas en tiempo, recurso y sobre todo en costos.

2.4.2 Nivel 2: Repetible

En este nivel se establecen prácticas básicas de administración de proyecto que permiten establecer control de requerimientos, calendarizaciones y costos. El equipo que desarrolla el proyecto puede aprovechar su experiencia e inversión en procesos que aplico en un anterior proyecto con mucho éxito.

Los proyectos repetibles llevan a cabo procesos definidos, medidos, entrenados y probados. Este nivel no garantiza que todos los proyectos dentro de la empresa tienen el mismo nivel de madurez. Algunos pueden estar aun en el nivel inicial.

Es importante que en este nivel la organización cuente con procesos formales de documentación. Y este proceso debe de ser comunicado a todos los niveles involucrados en el proyecto y continuamente mejorados por lo que debe de existir mucha precisión en la documentación de todos los procesos. Los análisis de proyectos anteriores proporcionan información importante acerca de todos los niveles de desarrollo. Esta información nos permite hacer compromisos inteligentes en cuanto a tiempos o costos, con ello tener una mejor calendarización.

Para nivel los líderes de proyecto toman más importancia que en el nivel anterior. Estos miembros del equipo dejan de encargarse de situaciones técnicas para dedicarse a administrar el proceso o los procesos involucrados en el desarrollo del producto. Ya pueden hacer un compromiso y seguir planes realistas basados en los resultados reproyectos similares. Deben de identificar los problemas que surjan durante el proyecto en el momento y darles solución o tomarlos en cuenta para futuros proyectos.

A partir de este nivel es importante llevar una implementación controlada de métricas y medidas de software ya que si no se hace aspa desde los primeros niveles, en los próximos será muy tardado acabo llevar una transición.

La repetición de éxitos anteriores es la base; los procesos efectivos son bien definidos, documentados, practicados y medidos pero aún pueden mejorar; además existen políticas para la administración de proyectos, sigue habiendo cajas negras pero ya son definidas y revisadas.

2.4.3 Nivel 3: Definido

En este nivel la organización ya debió haber definido un conjunto de procesos, metodologías y herramientas usados por todos los involucrados en el proceso, tanto administrativos como desarrolladores. Existen pautas y criterios definidos para un proceso estándar adaptadas a las necesidades y características propias de cada proyecto. Al llegar a este nivel todos los procesos son detallados y completos, esto da pie a que ya no exista dependencia de esfuerzos individuales, ya que todos los miembros de equipo conocen los procesos.

La línea que se sigue en este nivel es igual que en el nivel 2, pero aquí lo expande a todo el proceso, por ello hay un lenguaje común dentro de la empresa. La administración de un proyecto especifica en cada proceso que tareas se deben de llevar a cabo, así como quién o quienes deben de realizarlas, y con ello poder regular el tiempo

de entrega. La organización hace uso de sus mejores prácticas que tuvieron éxito en proyectos anteriores, y con ello estandarizar su proceso de desarrollo.

Se recomienda que dentro del equipo de trabajo exista gente dedicada al proceso de ingeniería de software, quienes se dedicaran de llevar a cabo actividades como la evaluación de los líderes o hacer revisiones del código con el fin de buscar mejoras en el proceso y saber como se desarrolla un producto dentro de la organización.

En resumen, el proceso es estándar, consistente y repetible a estas alturas. Esto se logra basándose en el entendimiento de las actividades, roles y responsabilidades en un proceso de software bien definido por cada integrante de la organización involucrado en el desarrollo del producto. Existen en este nivel métricas definidas para los productos y los servicios que ofrece la empresa. Además dicha organización debe de contar son un programa de capacitación para todos los miembros de su empresa, y sobre todos los involucrados en forma directa en el desarrollo.

2.4.4 Administrado

Alcanzar este nivel es muy complicados ya que no esta bien definido en la estructura de CMM. Para este nivel la empresa debe de medir la calidad del producto y del proceso de software. Ambas son seguidas en forma cuantitativa y se controlan mediante métricas detalladas. Y sobre todo en este nivel la capacidad de rendimiento del proceso es previsible.

La administración de proyectos en un papel esencial, en este nivel el proyecto se entiende como un conjunto de procesos la administración de los mismos y sus cambios. Todos los procedimientos y métricas utilizadas en los anteriores niveles se emplean aquí. El líder del proyecto dirige todos los procesos en forma paralela y los desarrollos hacia un objetivo bien definido. Así, también de encargarse de los procesos preventivos para corregir posibles errores antes de que estos se conviertan incontrolables.

En este nivel los procesos de software son muy predecibles. Esto ayuda a que cuando suceda algo no contemplado es fácilmente corregible debido al buen control del proceso. Esto ayuda mucho para que la organización sea capaz de proponerse metas cuantitativas para la calidad de sus productos y sus procesos de software. Con ello es posible medir la productividad y la calidad de sus procesos de software a través de todo el proyecto. Dichas mediciones permiten a la empresa detectar cuando las variaciones del rendimiento se salen de los rangos aceptables, de manera que se puedan tomar medidas correctivas o preventivas para asegurar la calidad del producto final.

Los procesos, a la par que el producto y servicios, son medidos y operan dentro de un límite cuantificable. En este nivel se cumplen con los planes y programas de mejora. Y sobre todo se hacen distinciones entre los procesos principales y los procesos de apoyo.

2.4.5 Optimizado

La principal característica aquí es que la empresa por completo lleva a cabo mejoras continuas en sus procesos, esto es en base a la retroalimentación cuantitativa y al ensayo de ideas y tecnologías innovadoras que existan en su momento. Todos estos cambios realizados en cualquier parte del proceso son vigilados y controlados con el sistema de métricas. Y con ello se asegura que durante todo el desarrollo no suelen suceder errores.

A esta altura la organización ya cuenta con los medios para identificar las debilidades y reforzar los procesos con el objetivo de prevenir defectos en la vida del desarrollo. Los datos relativos a la eficiencia de los procesos de software son valiosos para analizar el costo y el beneficio de usar nuevas tecnologías y de implementar cambios oportunos y adecuados a los procesos de software donde apliquen.

Los líderes analizan los defectos para determinar sus causas. Los procesos se evalúan para prevenir que los defectos conocidos vuelvan a ocurrir, asimismo estas prácticas aprendidas son difundidas a otros proyectos. Es por ello que es importante llevar a cabo retroalimentación en esta fase, de esto depende el control efectivo del proceso o procesos. Esta retroalimentación debe estar dirigida hacia un objetivo para no caer en la retroalimentación de errores desde un principio y seguir alimentando en forma incorrecta el proceso o procesos involucrado y causar todo lo contrario.

En resumen este nivel se dedica al mejoramiento continuo de los procesos a la par de su madurez, lo cual se da gracias al uso o implementación de nuevas tecnologías o métodos.

2.4.6 Áreas Clave del Proceso

Al conjunto de prácticas de software y el manejo específico de los niveles de madurez son llamadas áreas claves del proceso (Key Process Areas KPA). Cada una de éstas es descrita en términos de prácticas claves, las cuales describen las actividades o infraestructuras que contribuye de manera más efectiva a la implementación e institucionalización de la KPA.

Existen 18 KPA’s distribuidas a lo largo de los niveles 2 al 5 (El nivel 1 no tiene ninguna debido a las características que posee). Cada una de estas tiene distintas metas que cumplir.

Para cada nivel excepto como se menciono ya en el nivel 1, el CMM especifica algunas KPA’s que representan las áreas en las que una organización debe enfocarse si desea alcanzar algún nivel en particular. Cada una esta asociada con metas que representan los requerimientos a satisfacer por el proceso para esa KPA.

Las KPA’s para los niveles de madures existentes pueden ser usadoas para verificar la capacidad del proceso actual e identificar las áreas que necesitan ser reforzadas para alcanzar el siguiente nivel. En forma breve, los KPA’s de los distintos niveles se CMM son:

2.4.6.1 KPA's Nivel 2

Requirements Management (RM): Consiste en establecer un entendimiento común entre el cliente y el proyecto de software acerca de los requerimientos del usuario que serán cubiertos pro el proyecto de software. Mantener coherencia entre el cliente y los requerimientos que éste busca en su sistema de software.

Software Project Planning (SPP): Establece planes razonables para realizar la ingeniería de software y para administrar el proyecto de software. Busca diversos métodos para lograr un buen desempeño en la ingeniería de software y la administración del proyecto.

Software Project Tracking and Oversight (PTO): Se da la tarea de proporcionar una visibilidad adecuada del progreso actual, de manera que los administradores puedan tomar acciones efectivas cuando el desempeño del proyecto se desvía significativamente de lo planeado. Llevar un monitoreo del progreso actual del proyecto para que la administración puede tomar acciones acertadas cuando el desempeño del sistema se aleja de los planes iniciales.

Software Quality Assurance (SQA): Se encarga de proporcionar a la administración una visibilidad apropiada del proceso usado y los productos construidos durante el desarrollo de un proyecto de software. Tener una administración que tome en cuenta todo el proceso del proyecto y los productos desarrollados.

Software configuration Management (SCM): Establece y mantiene la integridad de los productos del proyecto de software a través del ciclo de vida del software del proyecto.

Software Subcontract management (SSM): Esta área es clave ya que tiene la tarea de selecciona a los subcontratistas de software calificados para el proyecto y administrarlos en forma eficiente.

2.4.6.2 KPA's Nivel 3

Organization Process Focus (OPF): Establece la responsabilidad organizacional de las actividades del proceso de software que mejoran la capacidad total del proceso de software de la organización. Fomenta un sentido de responsabilidad organizacional para generar actividades de proceso de software que pueden mejorar todo el proyecto.

Organization Process Definition (OPD): Desarrolla y mantiene un conjunto útil de activos de software que mejoren el desempeño de procesos entre proyectos y proporcionen una base de beneficios acumulables y de largo plazo para la organización.

Training Program (TP): Se encarga de desarrollar un programa de desarrollo de habilidades y conocimientos de individuos de tal manera que puedan desempeñar sus roles de forma efectiva y eficientemente.

Integrated Software Management (ISM): Sirve para integrar las actividades de ingeniería y administración del software en un proceso de software coherente y definido, que es adaptado del proceso estándar de software de la organización y de sus activos de procesos relacionados.

Software Product Engineering (SPE): Desempeña consistentemente un proceso de ingeniería bien definido que integra todas las actividades de ingeniería de software para producir, de manera efectiva y eficiente, productos correctos y consistentes de software.

Intergroup Coordination (IC): Establece los medios para que el grupo de ingeniería de software participe activamente con otros grupos de ingeniería, de tal manera que el proyecto tenga mayor capacidad de satisfacer las necesidades del cliente de manera efectiva y eficiente.

Peer Reviews (PR): Esta área elimina defectos de los productos de software en etapas tempranas y de manera eficiente. Un efecto colateral muy importante es desarrollar un mejor entendimiento de los productos de software y de los defectos que pueden prevenirse.

2.4.6.3 KPA's Nivel 4

Quantitative Process Management (QPM): Se encarga de controlar el desempeño del proceso del proyecto de softwa de manera cuantitativa. El desempeño del proceso representa los resultados actuales de seguir un proceso de software.

Software Quality Management (SQM): Desarrolla un entendimiento cuantitativo de la calidad de los productos de software de proyecto y alcanza metas de calidad específicas.

2.4.6.4 KPA's Nivel 5

Defect Prevention (DP): Identifica la cuasa de los defectos y prevenir su recurrencia.

Technology Change Management (TCM): Identifica nuevas tecnologías (es decir herramientas, métodos y procesos), que sean de beneficio y darles seguimiento dentro de la organización de manera ordenada.

Process Change Management (PCM): Mejora continuamente el proceso de software utilizado en la organización con la intención de mejorar la calidad del software, incrementar la productividad y disminuir el ciclo de tiempo para desarrollos de productos.

2.4.7 Características de las KPA's

2.4.7.1 Compromisos

Describen acciones que la organización debe realizar para asegurarse de que el proceso es establecido y de que perdurará. Típicamente involucra establecer políticas organizacionales y de liderazgo.

En el **nivel 2**, las políticas escritas que debe establecer la organización son las siguientes.

C2.1 Administrar los requerimientos del sistema cubiertos por el software.
C2.2 Planear y administrar un proyecto de software.
C2.3 Administrar el subcontrato de software.
C2.4 Implantar la Administración de la Calidad de Software.
C2.5 Implantar la Administración de la Configuración de Software.
C2.6 Designar un gerente del proyecto de software que tiene como responsabilidad negociar compromisos, desarrollar el plan del proyecto de desarrollo de software, de las actividades y resultados del proyecto de software
C2.7 Designar un gerente del subcontrato responsable de establecer y administrar el subconjunto de software.

En el **nivel 3** se establecen políticas organizacionales escritas para:

C3.1 Coordinar las actividades de desarrollo y mejoras del proceso de software a lo largo de la organización.
C3.2 Desarrollar y actualizar un proceso estándar de software y los activos de sus procesos relacionados.
C3.3 Alcanzar sus necesidades de capacitación.
C3.4 En este nivel se establecen políticas para los proyectos, las cuales requieren que el proyecto de software sea planeado y administrado utilizando el proceso estándar de software de la organización y los activos de procesos relacionados, se establecen para que el proyecto:
C3.5 Desempeñar actividades de ingeniería de software.
C3.6 Establecer equipos interdisciplinarios de ingeniería.
C3.7 Realizar revisiones entre colegas.
C3.8 A su vez, la gerencia tiene políticas que establecen patrocinar las actividades de los procesos de desarrollo y mejora del software y supervisar las actividades de la organización para el desarrollo y la mejora del proceso de software.

En el **nivel 4**, los compromisos que se establecen dicen que:

C4.1 El proyecto debe seguir una política organizacional escrita para medir y controlar cuantitativamente el desempeño del proceso definido de software, para administrar la calidad del software.
C4.2 la organización debe seguir una política escrita para analizar la capacidad del proceso estándar de software de la organización.

En el **nivel 5** tanto la organización como cada proyecto dentro de ella mantienen las siguientes políticas:

C5.1 Siguen una política escrita para las actividades de prevención de defectos.
C5.2 La organización sigue una política escrita para mejorar su capacidad tecnológica y para implantar mejoras en los procesos de software.
C5.3 La gerencia patrocina las actividades de la organización para la administración cambios tecnológicos y las de mejora del proceso de software.

2.4.7.2 Habilidades

Estas habilidades establecen y describen las condiciones que deben de existir en el proyecto y en la organización para implantar de manera competente los procesos de software. Esto involucra recursos, estructuras organizacionales y entrenamientos. En cada proyecto se establece la responsabilidad para analizar los requerimientos del sistema y dirigirlos hacia el software, u otros componentes del mismo.

En el **nivel 2**, se definen las siguientes habilidades:

H2.1 Los requerimientos son documentados; el plan de desarrollo para el proyecto de software tiene que ser documentado y aprobado.
H2.2 Los miembros del grupo de ingeniería de software y otros grupos relacionados reciben capacitación para realizar las actividades de administración de requerimientos.
H2.3 Existe una propuesta documentada y aprobada para el proyecto de software.
H2.4 Se asignan las responsabilidades para desarrollar el plan de desarrollo de software.
H2.5 Se proporcionan recursos y fondos adecuados para planear el proyecto de software para la realización de las actividades de Administración de la Configuración del Software y para seleccionar un subcontratista y para administrar el subcontrato; también para dar seguimiento al proyecto de software, realizar las actividades de aseguramiento de la calidad del software y para administrar los requerimientos.
H2.6 Los administradores de software, ingenieros de software y otros individuos involucrados en la planeación del proyecto reciben capacitación en procedimientos de estimación y planeación aplicables a las áreas en que son responsables, así como de aspectos técnicos del proyecto.
H2.7 El administrador del proyecto explícitamente asigna responsabilidades en los productos y actividades relacionadas con el software.
H2.8 Los ingenieros de software y otros individuos involucrados en la administración del subcontrato de software reciben orientación en los aspectos técnicos del subcontrato.
H2.9 Existe un grupo responsable de coordinar e implantar el aseguramiento de la calidad del software para el proyecto.
H2.10 Se capacita a los miembros del grupo de aseguramiento de calidad en los objetivos, procedimientos y métodos para ejecutar sus actividades.
H2.11 Existe un grupo responsable de la coordinación e implantación del Aseguramiento de Calidad de Software para cada proyecto.

En el **nivel 3** se establecen las siguientes habilidades:

H3.1 Existe un grupo responsable de las actividades de proceso de software de la organización quien recibe el entrenamiento requerido para el desempeño de las actividades que le corresponde y la forma en que deben adecuar el proceso estándar de software que le corresponde y la forma en que deben adecuar el proceso estándar de software de la organización y usar los activos del proceso relacionados; también recibe orientación de las actividades del proceso de software de la organización y sus roles en esas actividades.
H3.2 Existe un grupo responsable de satisfacer las necesidades de capacitación de la organización, el cual tiene las habilidades y los conocimientos necesarios para desempeñar sus actividades de capacitación.
H3.3 Los gerentes del software reciben la capacitación requerida para administrar los aspectos técnicos, administrativos y de personal del proyecto de software, basándose en el proceso de software definido.
H3.4 Los miembros del staff técnico de ingeniería de software reciben capacitación para desempeñar sus asignaciones técnicas y reciben orientación en las disciplinas de ingeniería de software relacionadas.
H3.5 El gerente de proyecto y todos los administradores de software reciben orientación en los aspectos técnicos del proyecto de software y para trabajar en equipo.
H3.6 Se proporcionan recursos y fondos adecuados para desarrollar y actualizar el proceso estándar y sus actividades de software de la organización y los activos de procesos relacionados, para implantar el programa de capacitación; para administrar el proyecto de software utilizando el proceso definido de software del proyecto, para desempeñar las tareas de ingeniería de software y para coordinar las actividades de ingeniería de software con otros grupos de ingeniería. Los recursos se administran también para que haya revisiones entre colegas de cada producto de software.
H3.7 Las herramientas de soporte utilizadas por los diferentes grupos de ingeniería son compatibles para permitir una coordinación y comunicación efectivas.
H3.8 todos los líderes de tareas de cada grupo de ingeniería reciben orientación en los procesos, métodos y estándares usados por otros grupos de ingeniería.
H3.9 Los coordinadores de las revisiones entre colegas reciben la capacitación necesaria en cuanto a los objetivos, principios y métodos de las revisiones entre colegas.
H3.10 Los revisores que participan en las revisiones entre colegas reciben la capacitación necesaria en cuanto a los objetivos, principios y métodos de las revisiones entre colegas.

En el **nivel 4**, el modelo establece las siguientes habilidades:

H4.1 En la organización, existe un grupo que es responsable de coordinar las actividades de administrar de forma cuantitativamente el proceso.
H4.2 Se proporcionan recursos y fondos apropiados para las actividades de administración cuantitativa del proceso y para administrar la calidad de los productos de software.
H4.3 Existe apoyo para recolectar, registrar y analizar datos para mediciones seleccionadas de procesos y productos.

H4.4 Los individuos que implantan o apoyan la administración cuantitativa del proceso reciben el entrenamiento requerido para el desempeño de estas actividades.
H4.5 Los miembros del grupo de ingeniería de software y otros grupos relacionados con el software reciben orientación acerca de los objetivos y el valor de la administración cuantitativa del proceso.

En el **nivel 5**, las habilidades descritas según el modelo CMM, son las siguientes:

H5.1 Existe un equipo a nivel organización que coordina las actividades de prevención de defectos.
H5.2 Se provee de recursos y fondos necesarios para las actividades de prevención de defectos a nivel organizacional y de proyectos, para establecer un grupo responsable de las actividades de administración de cambios tecnológicos y para proveer actividades de mejora en el proceso de software.
H5.3 Los miembros del grupo de ingeniería de software reciben capacitación para desempeñar sus actividades de prevención de defectos.
H5.4 Existe un grupo responsable de las actividades de administración de cambios tecnológicos el cual recibe entrenamientos necesario para realizar sus actividades de manera eficiente.
H5.5 Existe apoyo para recolectar y analizar la información necesaria para evaluar los cambios tecnológicos.
H5.6 La información apropiada de los procesos de software y el trabajo de esos productos, está disponible para apoyar los análisis realizados para evaluar y seleccionar cambios en la tecnología.
H5.7 Los administradores, staff técnico del software y la gerencia reciben entrenamiento acerca de las mejoras en el proceso de software-

2.4.7.3 Actividades

Estas actividades describen los roles y procedimientos necesarios para implantar un área clave del proceso. Típicamente involucra el establecimiento de planes y procedimientos, realización y seguimiento del trabajo y tomar acciones correctivas según sea necesario.

En el **nivel 2**, las actividades son:

A2.1 El grupo de ingeniería de software revisa los requerimientos establecidos antes de que sean incorporados en el proyecto de software y los usa como base para los planes, actividades y productos a desarrollar; éste grupo en la propuesta de equipo del proyecto.
A2.2 Los compromisos hechos hacia individuos o grupos externos a la organización con el proyecto de software, son revisados con la gerencia de acuerdo al procedimiento documentado.
A2.3 Es identificado o definido un ciclo de vida del software con etapas predefinidas de tamaño manejable.
A2.4 Es desarrollado, documentado y revisado un plan de desarrollo del proyecto de software de acuerdo al procedimiento documentado.
A2.5 Son identificados los productos de software que necesitan establecer y mantener control del proyecto de software.

A2.6 Estimados del tamaño del esfuerzo de los productos de software, costos, recursos críticos computacionales y horario son derivados de acuerdo al procedimiento documentado.
A2.7 Los riesgos de software asociados con el costo, horario, recurso y aspectos técnicos del proyecto, son identificados, asesorados y documentados.
A2.8 son preparados los planes de las facilidades de ingeniería del producto de software y herramientas de soporte.
A2.9 La información de planeación de software es registrada.
A2.10 Un plan de desarrollo de software documentado es usado para el seguimiento de las actividades de software y el estado de comunicación.
A2.11 Los compromisos del proyecto de software y sus cambios en individuaos y grupos externos a la organización, son revisados con la gerencia de acuerdo a un procedimiento documentado.
A2.12 Los cambios apropiados que afectan al proyecto de software son comunicados a los miembros del grupo de ingeniería de software y a otros grupos relacionados con el software.
A2.13 El esfuerzo, costo, recursos críticos computacionales, calendario, actividades técnicas de ingeniería de software del proyecto y sus riesgos asociados reciben seguimiento y acciones correctivas según sea necesario.
A2.14 La información actual de medición y replaneación es registrada.
A2.15 El grupo de ingeniería conduce revisiones internas periódicas para darle seguimiento a los progresos técnicos, planes, desempeño y problemas respecto al plan de desarrollo.
A2.16 Son conducidas revisiones formales para identificar las realizaciones y resultados del proyecto de software en un tiempo seleccionado de acuerdo al procedimiento documentado.
A2.17 El trabajo a ser subcontratado es definido y planeado de acuerdo al procedimiento documentado.
A2.18 Los subcontratistas de software son seleccionados basados en una evaluación de las habilidades de desempeño de realización del trabajo, de acuerdo al procedimiento documentado.
A2.19 Un plan de desarrollo de software de subcontratistas es revisado y aprobado por el contratista principal.
A2.20 Es usado un plan de desarrollo de contratistas que ha sido previamente documentado y aprobado para dar seguimiento a las actividades y estado de comunicación de software.
A2.21 Se llevan a cabo revisiones técnicas periódicas con los subcontratistas de software.
A2.22 El grupo de aseguramiento de Calidad de Software del contratista principal, monitorea las actividades de aseguramiento de calidad del subcontratista de acuerdo al procedimiento documentado.
A2.23 El contratista principal conduce una prueba de aceptación como parte de los entregables del subcontratista siguiendo el procedimiento establecido.
A2.24 El desempeño del subcontratista de software es evaluado en un periodo establecido y la evaluación es revisada con el subcontratista.
A2.25 Es preparado un plan de Aseguramiento de Calidad de Software para el proyecto de software siguiendo el procedimiento establecido.
A2.26 Las actividades del grupo de Aseguramiento de Calidad de Software son desempeñadas de acuerdo al plan definido y periódicamente reporta sus resultados al grupo de ingeniería.

A2.27 El grupo de Aseguramiento de Calidad de Software participa en la preparación y revisión del plan de desarrollo del proyecto de software, sus estándares y procedimientos; este grupo también se encarga de las revisiones de las actividades de ingeniería de software y auditorias para verificar el cumplimiento del producto.
A2.28 Las desviaciones identificadas en las actividades y productos de software se documentan y se manejan de acuerdo a un procedimiento documentado.
A2.29 El grupo de Aseguramiento de Calidad de Software lleva a cado revisiones periódicas de sus actividades y hallazgos con representantes de Aseguramiento de Calidad de Software del cliente.
A2.30 Es preparado un plan de administración de la configuración de software para cada proyecto de acuerdo al procedimiento establecido.
A2.31 Un plan de administración de la configuración del software documentado y aprobado es utilizado como base para realizar las actividades de administración de la configuración del software.
A2.32 Se inician, registran, revisan y aprueban los requerimientos y reportes de problemas de los asuntos/unidades de configuración, además de tener seguimiento de acuerdo al procedimiento.

En el **nivel 3**, las actividades a cumplir son:

A3.1 El proceso de software es definido periódicamente y planes de acciones son desarrollados para dirigirlos hacia los hallazgos de la valoración.
A3.2 La organización desarrolla y actualiza un plan para el desarrollo de su proceso software y para las actividades de mejora.
A3.3 Las actividades para el desarrollo y mejoría de los procesos de software de proyectos y de la organización son coordinadas a nivel de la organización.
A3.4 El uso de la base de datos del proceso de software de la organización se coordina a nivel organizacional.
A3.5 Se monitorean, evalúan y, cuando es apropiado, se transfieren a otras partes de la organización los nuevos procesos, métodos y herramientas de uso limitado en la organización.
A3.6 La capacitación para la organización y los proyectos respecto al proceso de software se coordinan a través de toda la organización.
A3.7 Los grupos involucrados en la implantación del proceso de software son informados de las actividades de la organización y los proyectos para el desarrollo y la mejora del proceso del software.
A3.8 El proceso estándar de software de la organización se desarrolla y actualiza de acuerdo a un procedimiento documentado y es documentado de acuerdo a los estándares organizacionales establecidos.
A3.9 Se documenta y actualiza la descripción de los ciclos de vida de software aprobados para ser usados en los proyectos.
A3.10 Se desarrollan y actualizan guías y criterios para adaptar el proceso estándar de software de la organización a proyectos específicos.
A3.11 Se establecen y actualiza una base de datos organizacional de proceso de software.
A3.12 Se establece y actualiza una biblioteca de documentación relacionada con el proceso de software.
A3.13 Cada proyecto de software desarrolla y actualiza un programa de capacitación que especifica sus necesidades de capacitación.

A3.14 El plan de capacitación de la organización se desarrolla y revisa de acuerdo con un procedimiento documentado.
A3.15 La capacitación para la organización se realiza de acuerdo con el plan de capacitación organizacional.
A3.16 Los cursos de capacitación preparados en la organización se desarrollan y actualizan de acuerdo con estándares organizacionales.
A3.17 Se establece y usa un procedimiento para la omisión de capacitación y es usado para determinar cuándo los individuos ya poseen los conocimientos y las habilidades requeridas para el ejecutar los roles asignados.
A3.18 Se conservan y actualizan los registros de la capacitación.
A3.19 El proceso de software definido para el proyecto se desarrolla adaptando el proceso estándar de software de la organización de acuerdo con un procedimiento documentado.
A3.20 Se revisa el proceso de software definido para cada proyecto de acuerdo con un procedimiento documentado.
A3.21 El plan de desarrollo de software del proyecto, que describe el uso del proceso de software definido para el proyecto, es desarrollado y revisado de acuerdo con un procedimiento adecuado.
A3.22 El proyecto de software es administrado de acuerdo con el proceso de software definido para el proyecto.
A3.23 La base de datos del proceso de software de la organización se usa para planear y estimar.
A3.24 El tamaño de los productos de software (o el tamaño de los cambios a los productos de software) se administra de acuerdo con un procedimiento documentado.
A3.25 El esfuerzo y los costos del proyecto de software se administran de acuerdo con un procedimiento adecuado.

A3.26 Los recursos de cómputo crípticos para el proyecto se administran de acuerdo con un procedimiento documentado.
A3.27 Las dependencias críticas y las rutas críticas del calendario de software del proyecto se administran de acuerdo con un procedimiento documentado.
A3.28 Se identifican, evalúan, documentan y administran los riesgos de software del proyecto de acuerdo con un procedimiento documentado.
A3.29 Se ejecutan revisiones periódicas del proyecto de software para determinar las acciones necesarias para llevar el desempeño y los resultados del proyecto de software en línea con las necesidades actuales y proyectadas del negocio, el cliente y los usuarios.
A3.30 Se integran métodos y herramientas apropiadas de ingeniería de software en el proceso de software definido para el proyecto.
A3.31 Los requerimientos de software se desarrollan, actualizan, documentación y verificación al analizar sistemáticamente los requerimientos proporcionados de acuerdo con el proceso de software definido para el proyecto.
A3.32 El diseño del software se desarrolla, actualiza, documenta y verifica de acuerdo con el proceso de software definido para el proyecto, para incluir los requerimientos de software y para formar un marco de referencia para la codificación.
A3.33 El código de software se desarrolla, actualiza, documenta y verifica de acuerdo con el proceso de software definido para le proyecto, para implantar los requerimientos de software y el diseño del software.

A3.34 Las pruebas de software se realizan de acuerdo con el proceso de software definido para el proyecto.

A3.35 Las pruebas de integración se planean y se realizan de acuerdo con el proceso de software definido para el proyecto.

A3.36 Las pruebas de sistema y de aceptación del software se planean y se realizan para demostrar que el software satisface sus requerimientos.

A3.37 La documentación que se usará para operar y actualizar el software se desarrolla y actualiza de acuerdo con el proceso de software definido para el proyecto.

A3.38 Los datos de los defectos identificados en las revisiones entre colegas y en las pruebas se colectan y analizan con el proceso de software definido para le proyecto.

A3.39 Se mantiene la consistencia entre los productos de software, incluyendo planes de software, descripciones de proceso, requerimientos proporcionados, requerimientos de software, diseño de software, codificación, planes de software y procedimientos de prueba.

A3.40 El grupo de ingeniería de software y otros de ingeniería participan con el cliente y los usuarios finales, según sea apropiado, para establecer los requerimientos del sistema.

A3.41 Representantes de grupo de ingeniería de software del proyecto trabajan con representantes de otros grupos de ingeniería para monitorear y coordinar las actividades técnicas y resolver asuntos técnicos.

A3.42 Se utiliza un plan documentado para comunicar los compromisos intergrupales y para coordinar y dar seguimiento al trabajo realizado.

A3.43 Se identifican, negocian y se les da seguimiento a las dependencias críticas entre grupos de ingeniería, de acuerdo con un procedimiento documentado.

A3.44 Los productos producidos como una entrada para otros grupos de ingeniería se revisan por representantes de los grupos que las reciben, para asegurar que los productos de trabajo satisfacen sus necesidades.

A3.45 Los asuntos intergrupales no resueltos por representantes individuales de los grupos de ingeniería del proyecto se manejan de acuerdo a un procedimiento documentado.

A3.46 Los representantes de los grupos de ingeniería del proyecto conducen revisiones e intercambios técnicos de manera periódica.

A3.47 Se planean las actividades de revisión entre colegan y se realizan de acuerdo con un procedimiento documentado; los datos de la conducción y los resultados de las revisiones entre colegas son registrados.

En el **nivel 4**, las actividades descritas son:

A4.1 El plan de proyecto de software para la administración cuantitativa del proceso se desarrolla de acuerdo con un procedimiento documentado.

A4.2 Las actividades de administración cuantitativa del proceso para el proyecto se realizan de acuerdo con el plan para el proyecto de administración cuantitativa del proceso.

A4.3 Se determina la estrategia para recolección de datos y el análisis cuantitativo a realizarse basados en el proceso de software definido para el proyecto.

A4.4 Los datos de mediciones usadas para controlar cuantitativamente el proceso de software del proyecto se recolecta a un procedimiento documentado.
A4.5 El proceso de software definido para el proyecto se analiza y se lleva bajo control cuantitativo de acuerdo con un procedimiento documentado.
A4.6 Se preparan y distribuyen reportes documentado los resultados de las actividades de administración cuantitativa del proceso del proyecto de software.
A4.7 Se establece y actualiza la base de capacitación del proceso para el proceso estándar de software de la organización, de acuerdo con un procedimiento documentado.
A4.8 El plan de calidad de software del proyecto es la base de las actividades del proyecto para administración de la calidad del software, es desarrollado y actualizado de acuerdo con un procedimiento documentado.
A4.9 Se definen, monitorean y revisan las metas cuantitativas de calidad del proyecto para os productos de software a través del ciclo de vida del software.
A4.10 La calidad de los productos de software del proyecto se mide, se analiza y se compara contra las metas cuantitativas de calidad de los productos, sobre la base de eventos.
A4.11 Se proporcionan adecuadamente a los subcontratistas que entregan productos de software del proyecto las metas cuantitativas de calidad del proyecto de software para los productos.

En el **nivel 5**, las actividades establecidas son:

A5.1 El proyecto de software desarrolla y mantiene un plan para sus actividades de prevención de defectos.
A5.2 Al inicio de una tarea de software, los miembros del equipo que realizan la tarea se reúnen para prepararse para llevar acabo sus actividades tanto de tareas como de prevención de defectos.
A5.3 Se llevan a cabo reuniones de análisis causal de acuerdo con un procedimiento documentado.
A5.4 Cada uno e los equipos asignados a coordinar las actividades de prevención de defectos se reúnen de manera periódica para revisar y coordinar propuestas de acción provenientes de reuniones de análisis causal.
A5.5 Datos de prevención de defectos se documentan y se les da seguimiento a través de los equipos que coordinan las actividades de prevención de defectos.
A5.6 Se incorporan las revisiones al proceso estándar y definido de software de la organización que resultan de las acciones de prevención de defectos, de acuerdo con un procedimiento documentado.
A5.7 Los miembros del grupo de ingeniería de software y otros grupos relacionados con el software reciben retroalimentación de la situación y los resultados de las actividades de prevención de defectos en la organización y en el proyecto, en forma periódica.
A5.8 La organización desarrolla y mantiene un plan para la administración del cambio tecnológico
A5.9 El grupo responsable de las actividades de administración del cambio tecnológico trabajo con los proyectos de software para identificar áreas de cambio tecnológico.
A5.10 Se mantiene informados a los gerentes de software y el staff técnico acerca de las nuevas tecnologías.

A5.11 El grupo responsable de la administración del cambio tecnológico de la organización sistemáticamente analiza el proceso estándar de software de la organización para identificar áreas que necesitan o pueden beneficiarse de las nuevas tecnologías.
A5.12 Las tecnologías para la organización y para los proyectos de software se adquieren y se seleccionan de acuerdo con un procedimiento definido.
A5.13 Cuando sea apropiado, se conducen esfuerzos piloto para mejorar la tecnología antes que una nueva tecnología se introduzca en la práctica normal.
A5.14 Las nuevas tecnologías se incorporan los proceso estándar de software de la organización de acuerdo con un procedimiento documentado.
A5.15 Las nuevas tecnologías se incorporar el proceso definido de software del proyecto de acuerdo con un procedimiento documentado.
A5.16 Se establece un programa de mejora del proceso de software, el cual faculta a los miembros de la organización a mejorar los procesos de la organización.
A5.17 El grupo responsable de las actividades del proceso de software de la organización (grupo de proceso de ingeniería de software) coordina las actividades de mejora del proceso de software.
A5.18 La organización desarrolla y actualiza un plan de mejora del proceso de software de acuerdo con un procedimiento documentado.
A5.19 Las actividades de mejora del proceso de software se realizan de acuerdo con el plan de mejora del proceso de software.
A5.20 Las propuestas de mejora del proceso de software se manejan de acuerdo con un procedimiento documentado.
A5.21 Los miembros de la organización participan activamente en equipos para desarrollar mejoras al proceso de software en áreas de procesos asignadas.
A5.22 Cuando sea apropiado, las mejoras al proceso de software se instalan como pilotos para determinar beneficios y efectividad antes de introducirse en la practica normal.
A5.23 Cuando se tome la decisión se transferir la mejora del proceso de software en la práctica normal, las mejoras son implantaciones de acuerdo con un procedimiento documentado.
A5.24 Se actualizan los registros de las actividades de mejora del proceso de software.
A5.25 Los gerentes y staff técnico de software reciben retroalimentación del estatus y resultados de actividades de mejora del proceso.

2.4.7.4 Medición y análisis

Describen las prácticas básicas de medición que son necesarias para determinar el estado relacionado con los procesos. Estas mediciones son utilizadas para controlar y mejorar los procesos:

En el **nivel 2**, se hacen y se usan mediciones para:

M2.1 Determinar el estado de las actividades para administrar los requerimientos y para la planeación del software.
M2.2 Determinar el estatus de las actividades de seguimiento y supervisión del software y para administrar el subcontrato de software.

M2.3 Determinar el estado de las actividades de aseguramiento de la calidad y de configuración del software.

En el **nivel 3** se realiza y utiliza mediciones para:

M3.1 Determinar el estado de los procesos de desarrollo, las actividades de mejora y de definición de procesos.
M3.2 Determinar el estado de las actividades del programa de capacitación, la calidad del programa de capacitación, la efectividad de las actividades de la administración integrada del software.
M3.3 Determinar la funcionalidad y calidad de los productos de software, la situación de las actividades de ingeniería de producto de software, de coordinación intergrupal y de las actividades de revisión entre colegas.

En el **nivel 4** las mediciones se usan para:

M4.1 Determinar la situación de las actividades de administración cuantitativa del proceso y la situación de las actividades de administración de la calidad del software.

En el **nivel 5** se establecen mediciones para:

M5.1 Determinar el estado de las actividades de prevención de defectos.
M5.2 Determinar la situación de las actividades de administración del cambio tecnológico de la organización.
M5.3 Determinar las actividades de mejoras del proceso de software.

2.4.7.5 Verificaciones de implantación

Esta etapa describe los pasos a seguir para asegurar que las actividades son realizadas de acuerdo a los procesos establecidos previamente. Involucra revisiones y auditorias de administración y aseguramiento de la calidad de software.

En el **nivel 2**, las verificaciones de implantación pueden ser de varios tipos y a diferentes niveles de la organización.

V2.1 Las actividades para administrar los requerimientos y para planear el proyecto, administrar el subconjunto de software, administración de la configuración, aseguramiento de calidad del software, seguimiento y supervisión del proyecto son revisadas periódicamente con la gerencia; estas actividades también son revisadas con el gerente del proyecto tanto periódicamente como por evento.
V2.2 El grupo de aseguramiento de calidad de software revisa y/o audita las actividades y entregables para administrar los requerimientos, administrar la configuración del software, planear los entregables, supervisar el proyecto, administrar el subcontrato de software y así reportar resultados del proyecto.
V3.3 El grupo de aseguramiento de calidad de software revisa periódicamente las actividades y entregables del proyecto de este grupo.

V2.4 El grupo de administración de la configuración de software audita las bases para verificar que siguen la documentación que definieron.

En el **nivel 3**, las verificaciones son:

V3.1 La gerencia superior revisa de forma periódica las actividades del proceso de desarrollo y mejora de software, del programa de capacitación, administración del proyecto de software, ingeniería del producto de software y las correspondientes a coordinación intergrupal.
V3.2 El grupo de aseguramiento de calidad de software revisa y/o audita las actividades de la organización y entregables para desarrollar y mantener un proceso de software estándar y relacionar los archivos de los procesos, las actividades y productos de coordinación intergrupal, administración del proyecto de software, revisión entre colegas, ingeniería del producto de software y reporta los resultados.
V3.3 Las actividades del programa de capacitación y sus entregables son revisadas y/o auditadas y los resultados de ésta son reportados.
V4.4 El programa de capacitación es evaluado independientemente y de manera periódica para asegurar que sea consistente y relevante respecto a las necesidades de la organización.
V3.5 El administrador del proyecto revisa tanto de forma periódica como por evento las actividades para administrar el proyecto, las actividades de ingeniería y las actividades de coordinación intergrupal.

En el **nivel 4,** las verificaciones son:

V4.1 Las actividades de administración cuantitativa del proceso y de administración de la calidad del software se revisan con la gerencia de forma periódica.
V4.2 Las actividades de administración cuantitativa del proceso y de administración de la calidad del software correspondiente al proyecto se revisan con el administrador del proyecto tanto de forma periódica como por evento.
V4.3 El grupo de aseguramiento de calidad revisa y/o audita las actividades y productos para la administración cuantitativa del proceso y administración de la calidad de software y reporta los resultados.

En el **nivel 5**, las verificaciones son:

V5.1 Las actividades de prevención de defectos, mejora del proceso de software y de administración del cambio tecnológico de la organización son revisadas con la gerencia de forma periódica.
V5.2 El administrador del proyecto hace revisiones tanto de forma periódica como por evento.
V5.3 El grupo de aseguramiento de calidad revisa y/o audita las actividades y productos de prevención de defectos, administración del cambio tecnológico y mejora del proceso de software para reportar éstos resultados.

A través de los cinco niveles, el proceso de capacidad interactúa con personas, tecnología y medidas mientras la organización madura, en la siguiente tabla se muestra la manera en que se relacionan.

Capítulo 3

Integración de CMMI con UML

3.1 UML

El Lenguaje Unificado de Modelado (UML) prescribe un conjunto de notaciones y diagramas estándares para modelar sistemas orientados a objetos, y describe la semántica esencial de lo que estos diagramas y símbolos significan. Mientras que ha habido muchas notaciones y métodos usados para el diseño orientado a objetos, ahora los modeladores sólo tienen que aprender una única notación.

UML es un lenguaje que nos ayuda a interpretar grandes sistemas mediante gráficos o mediante texto obteniendo modelos explícitos que ayudan a la comunicación durante el desarrollo ya que al ser estándar, los modelos podrán ser interpretados por personas que no participaron en su diseño (e incluso por herramientas) sin ninguna ambigüedad. En este contexto, UML sirve para *especificar*, modelos concretos, no ambiguos y completos.

UML proporciona la capacidad de modelar actividades de planificación de proyectos y de sus versiones, expresar requisitos y las pruebas sobre el sistema, representar todos sus detalles así como la propia arquitectura. Mediante estas capacidades se obtiene una documentación que es valida durante todo el ciclo de vida de un proyecto.

UML se puede usar para modelar distintos tipos de sistemas: sistemas de software, sistemas de hardware, y organizaciones del mundo real. UML ofrece nueve diagramas en los cuales modelar sistemas.

- Diagramas de Casos de Uso para modelar los procesos ’business’.
- Diagramas de Secuencia para modelar el paso de mensajes entre objetos.
- Diagramas de Colaboración para modelar interacciones entre objetos.
- Diagramas de Estado para modelar el comportamiento de los objetos en el sistema.
- Diagramas de Actividad para modelar el comportamiento de los Casos de Uso, objetos u operaciones.
- Diagramas de Clases para modelar la estructura estática de las clases en el sistema.
- Diagramas de Objetos para modelar la estructura estática de los objetos en el sistema.
- Diagramas de Componentes para modelar componentes.
- Diagramas de Implementación para modelar la distribución del sistema.

UML es una consolidación de muchas de las notaciones y conceptos más usadas orientados a objetos. Empezó como una consolidación del trabajo de Grady Booch, James Rumbaugh, e Ivar Jacobson, creadores de tres de las metodologías orientadas a objetos más populares.

Originalmente, UML fue concebido por Grady Booch, James Rumbaugh e Ivar Jacobson de Rational Software. Posteriormente obtuvo el apoyo de la industria vía el

consorcio de socios de UML, y fue presentado al Object Management Group (OMG) y aprobado por éste como estándar (17 de noviembre de 1997).

Debido a que UML evolucionó a partir de varios métodos orientados a objeto de segunda generación (en cuanto a nivel de notación), mayoritariamente se cree que sólo es aplicable a sistemas de software orientados a objeto cuando, realmente, UML no es simplemente un lenguaje para modelado orientado a objeto de tercera generación, sino un "lenguaje para modelado unificado" relativo a sistemas en general (ver Figura 3.1). Al ser un lenguaje de modelado y no un método, consiste en una notación principalmente gráfica, que cualquier método puede emplear para expresar su diseño.

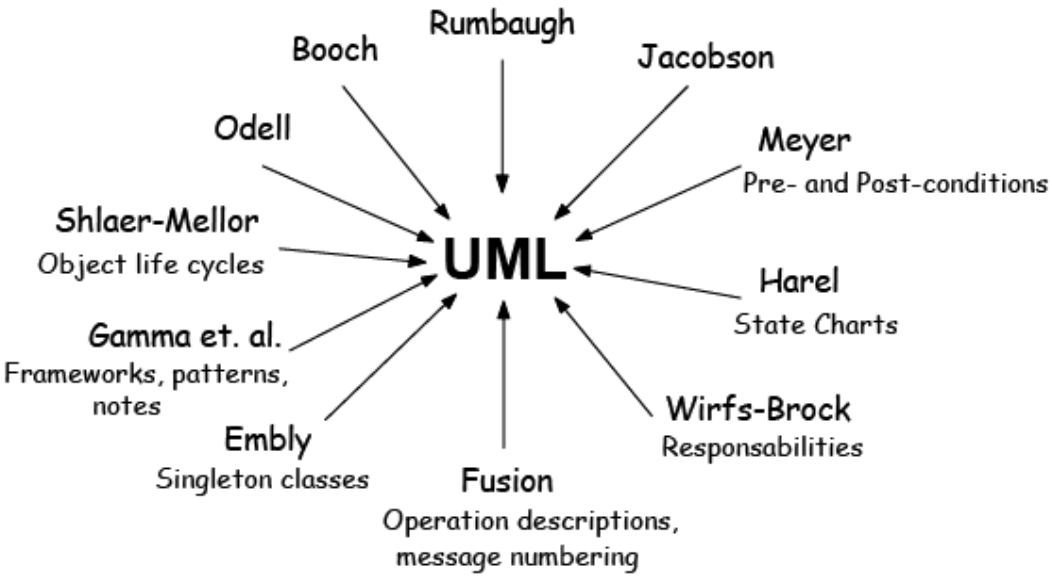


Fig. 3.1 UML como fusión de varios lenguajes de modelado

UML es un lenguaje de modelado para la especificación, visualización, construcción y documentación de sistemas.

3.2 Vista General de UML

Para conocer la estructura de UML, en la figura 3.2 vemos una vista general de todos sus componentes, esto hará que nos resulte más fácil la comprensión de cada uno de ellos.

El lenguaje UML se compone de tres elementos básicos, los bloques de construcción, las reglas y algunos mecanismos comunes. Estos elementos interaccionan entre sí para dar a UML el carácter de completitud y no-ambigüedad que antes comentábamos.

Los **bloques de construcción** se dividen en tres partes: **Elementos**, que son las abstracciones de primer nivel, **Relaciones**, que unen a los elementos entre sí, y los **Diagramas**, que son agrupaciones interesantes de elementos.

Existen cuatro tipos de elementos en UML, dependiendo del uso que se haga de ellos: *elementos estructurales, elementos de comportamiento, elementos de agrupación y elementos de anotación.* Las relaciones, a su vez se dividen para abarcar las posibles interacciones entre elementos que se nos pueden presentar a la hora de modelar usando UML, estas son: *relaciones de dependencia, relaciones de asociación, relaciones de generalización y relaciones de realización.* Se utilizan diferentes diagramas dependiendo de qué, nos interese representar en cada momento, para dar diferentes perspectivas de un mismo problema, para ajustar el nivel de detalle..., por esta razón UML soporta un gran numero de diagramas diferentes aunque, en la practica, sólo se utilicen un pequeño número de combinaciones.

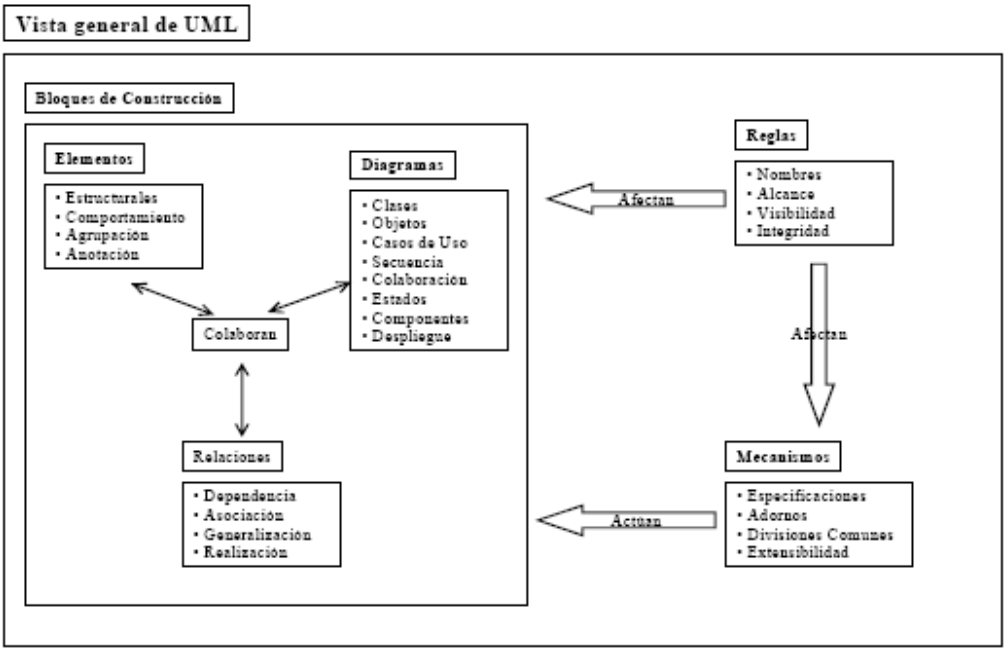


Fig. 3.2 Vista general de los elementos de UML.

UML proporciona un conjunto de reglas que dictan las pautas a la hora de realizar asociaciones entre objetos para poder obtener modelos bien formados, estas son reglas semánticas que afectan a los **nombres**, al **alcance** de dichos nombres, a la **visibilidad** de estos nombres por otros, a la **integridad** de unos elementos con otros y a la **ejecución**, o sea la vista dinámica del sistema.

UML proporciona una serie de mecanismos comunes que sirven para que cada persona o entidad adapte el lenguaje a sus necesidades, pero dentro de un marco ordenado y siguiendo unas ciertas reglas para que en el trasfondo de la adaptación no se pierda la semántica propia de UML. Dentro de estos mecanismos están las **especificaciones**, que proporcionan la explicación textual de la sintaxis y semántica de los bloques de construcción. Otro mecanismo es el de los **adornos** que sirven para conferir a los modelos de más semántica, los adornos son elementos secundarios ya que proporcionan más nivel de detalle, que quizá en un primer momento no sea conveniente descubrir. Las **divisiones comunes** permiten que los modelos se dividan al menos en un par de formas diferentes para facilitar la comprensión desde distintos puntos de vista, en primer lugar tenemos la división entre clase y objeto (clase es una abstracción y objeto es una manifestación de esa abstracción), en segundo lugar tenemos la división interfaz /

implementación donde la interfaz presenta un contrato (algo que se va a cumplir de una determinada manera) mientras que la implementación es la manera en que se cumple dicho contrato. Por ultimo, los **mecanismos de extensibilidad** que UML proporciona sirven para evitar posibles problemas que puedan surgir debido a la necesidad de poder representar ciertos matices, por esta razón UML incluye los *estereotipos*, para poder extender el vocabulario con nuevos bloques de construcción, los *valores etiquetados*, para extender las propiedades un bloque, y las *restricciones*, para extender la semántica.

De esta manera UML es un lenguaje estándar *"abierto-cerrado"* siendo posible extender el lenguaje de manera controlada.

3.2.1 Bloques de construcción de UML

A continuación se van a describir todos los elementos que componen los bloques estructurales de UML, así como su notación, para que nos sirva de introducción y se vaya generando un esquema conceptual sobre UML. En temas sucesivos se tratará con más profundidad cada uno de los bloques.

3.2.1.1 Elementos Estructurales

Los elementos estructurales en UML, en su mayoría, son las partes estáticas del modelo y representan cosas que son conceptuales o materiales.

Clases

A continuación se van a describir todos los elementos que componen los bloques estructurales de UML, así como su notación, para que nos sirva de introducción y se vaya generando un esquema conceptual sobre UML. En temas sucesivos se tratará con más profundidad cada uno de los bloques.

Interfaz

Una interfaz es una colección de operaciones que especifican un servicio de una determinada clase o componente. Una interfaz describe el comportamiento visible externamente de ese elemento, puede mostrar el comportamiento completo o sólo una parte del mismo. Una interfaz describe un conjunto de especificaciones de operaciones (o sea su signatura) pero nunca su implementación.

Colaboración

Define una interacción y es una sociedad de roles y otros elementos que colaboran para proporcionar un comportamiento cooperativo mayor que la suma de los comportamientos de sus elementos. Las colaboraciones tienen una dimensión tanto estructural como de comportamiento. Una misma clase puede participar en diferentes colaboraciones. Las colaboraciones representan la implementación de patrones que forman un sistema.

Casos de Uso

Un caso de uso es la descripción de un conjunto de acciones que un sistema ejecuta y que produce un determinado resultado que es de interés para un actor particular. Un caso de uso se utiliza para organizar los aspectos del comportamiento en un modelo. Un caso de uso es realizado por una colaboración.

Clase Activa

Es una clase cuyos objetos tienen uno o más procesos o hilos de ejecución por lo y tanto pueden dar lugar a actividades de control. Una clase activa es igual que una clase, excepto que sus objetos representan elementos cuyo comportamiento es concurrente con otros elementos.

Componentes

Un componente es una parte física y reemplazable de un sistema que conforma con un conjunto de interfaces y proporciona la implementación de dicho conjunto. Un componente representa típicamente el empaquetamiento físico de diferentes elementos lógicos, como clases, interfaces y colaboraciones.

Nodos

Un nodo es un elemento físico que existe en tiempo de ejecución y representa un recurso computacional que, por lo general, dispone de algo de memoria y, con frecuencia, de capacidad de procesamiento. Un conjunto de componentes puede residir en un nodo.

Estos siete elementos vistos son los elementos estructurales básicos que se pueden incluir en un modelo UML. Existen variaciones sobre estos elementos básicos, tales como actores, señales, utilidades (tipos de clases), procesos e hilos (tipos de clases activas) y aplicaciones, documentos, archivos, bibliotecas, páginas y tablas (tipos de componentes).

Elementos de comportamiento

Los elementos de comportamiento son las partes dinámicas de un modelo. Se podría decir que son los verbos de un modelo y representan el comportamiento en el tiempo y en el espacio. Los principales elementos son los dos que siguen.

Interacción

Es un comportamiento que comprende un conjunto de mensajes intercambiados entre un conjunto de objetos, dentro de un contexto particular para conseguir un propósito específico. Una interacción involucra otros muchos elementos, incluyendo mensajes, secuencias de acción (comportamiento invocado por un objeto) y enlaces (conexiones entre objetos). La representación de un mensaje es una flecha dirigida que normalmente con el nombre de la operación.

Maquinas de estados

Es un comportamiento que especifica las secuencias de estados por las que van pasando los objetos o las interacciones durante su vida en respuesta a eventos, junto con las respuestas a esos eventos. Una maquina de estados involucra otros elementos como son estados, transiciones (flujo de un estado a otro), eventos (que disparan una transición) y actividades (respuesta de una transición).

Elementos de agrupación

Forman la parte organizativa de los modelos UML. El principal elemento de agrupación es el **paquete**, que es un mecanismo de propósito general para organizar

elementos en grupos. Los elementos estructurales, los elementos de comportamiento, incluso los propios elementos de agrupación se pueden incluir en un paquete.

Un paquete es puramente conceptual (sólo existe en tiempo de desarrollo). Gráficamente se representa como una carpeta conteniendo normalmente su nombre y, a veces, su contenido.

Elementos de anotación

Los elementos de anotación son las partes explicativas de los modelos UML. Son comentarios que se pueden aplicar para describir, clasificar y hacer observaciones sobre cualquier elemento de un modelo. El tipo principal de anotación es la **nota** que simplemente es un símbolo para mostrar restricciones y comentarios junto a un elemento o un conjunto de elementos.

3.2.1.2 Relaciones

Existen cuatro tipos de relaciones entre los elementos de un modelo UML. *Dependencia*, *asociación*, *generalización* y *realización*, estas se describen a continuación:

Dependencia

Es una relación semántica entre dos elementos en la cual un cambio a un elemento (el elemento independiente) puede afectar a la semántica del otro elemento (elemento dependiente). Se representa como una línea discontinua, posiblemente dirigida, que a veces incluye una etiqueta.

Asociación

Es una relación estructural que describe un conjunto de enlaces, los cuales son conexiones entre objetos. La agregación es un tipo especial de asociación y representa una relación estructural entre un todo y sus partes. La asociación se representa con una línea continua, posiblemente dirigida, que a veces incluye una etiqueta.

Generalización

Es una relación de especialización / generalización en la cual los objetos del elemento especializado (el hijo) pueden sustituir a los objetos del elemento general (el padre). De esta forma, el hijo comparte la estructura y el comportamiento del padre. Gráficamente, la generalización se representa con una línea con punta de flecha vacía.

Realización

Es una relación semántica entre clasificadores, donde un clasificador especifica un contrato que otro clasificador garantiza que cumplirá. Se pueden encontrar relaciones de realización en dos sitios: entre interfaces y las clases y componentes que las realizan, y entre los casos de uso y las colaboraciones que los realizan. La realización se representa como una mezcla entre la generalización y la dependencia, esto es, una línea discontinua con una punta de flecha vacía.

3.2.1.3 Diagramas

EL UML está compuesto por diversos elementos gráficos que se combinan para conformar diagramas. Debido a que el UML es un lenguaje, cuenta con reglas para combinar tales elementos.

La finalidad de los diagramas es presentar diversas perspectivas de un sistema a las cuales se les conoce como *modelo*. El modelo UML de un sistema es similar a un modelo a escala de un edificio junto con la interpretación del artista. Es importante destacar que un modelo UML describe lo que supuestamente hará un sistema, pero no dice cómo implementar dicho sistema.

A continuación se describirán brevemente los diagramas más comunes del UML y los conceptos que representan. Recuerde que es posible generar híbridos de estos diagramas y que el UML otorga formas de organizarlos y extenderlos.

Diagrama de clases

Muestran un conjunto de clases, interfaces y colaboraciones, así como sus relaciones. Estos diagramas son los más comunes en el modelado de sistemas orientados a objetos y cubren la vista de diseño estática o la vista de procesos estática.

Piense en las cosas que le rodean. Es muy probable que muchas de esas cosas tengan atributos (propiedades) y que realicen determinadas acciones. Podríamos imaginar cada una de esas acciones como un conjunto de tareas. También se encontrará con que las cosas naturalmente se albergan en categorías. A tales categorías las llamaremos clases. Una *clase* es una categoría o grupo de cosas que tiene atributos y acciones similares.

El objetivo de pensar en las clases, así como sus atributos y acciones es para interactuar con nuestro complejo mundo, la mayoría del software moderno simula algún aspecto del mundo. Décadas de experiencia sugieren que es más sencillo desarrollar aplicaciones que simules algún aspecto del mundo cuando el software representa clases de cosas reales. Los diagramas de clase facilitan las representaciones a partir de las cuales los desarrolladores podrán trabajar. A su vez, estos diagramas colaboran en lo referente al análisis. Permiten al analista hablarle a los clientes en su propia terminología, lo cual hace posible que los clientes indiquen importantes detalles de los problemas que requieren ser resueltos.

Diagramas de objetos

Muestran un conjunto de objetos y sus relaciones, son como fotos instantáneas de los diagramas de clases y cubren la vista de diseño estática o la vista de procesos estática desde la perspectiva de casos reales o prototípicos.

Un objeto es una instancia de clase (una entidad que tiene valores específicos de los atributos y acciones).

Diagrama de casos de uso

Muestran un conjunto de objetos y sus relaciones, son como fotos instantáneas de los diagramas de clases y cubren la vista de diseño estática o la vista de procesos estática desde la perspectiva de casos reales o prototípicos.

Un caso de uso es una descripción de las acciones de un sistema desde el punto de vista del usuario. Para los desarrolladores del sistema, ésta es una herramienta valiosa, ya que es una técnica de aciertos y errores para obtener los requerimientos del sistema desde el punto de vista del usuario. Esto es importante si la finalidad es crear un sistema que pueda ser utilizado por la gente en general.

Diagrama de estados

Muestran una maquina de estados compuesta por estados, transiciones, eventos y actividades. Estos diagramas cubren la vista dinámica de un sistema y son muy importantes a la hora de modelar el comportamiento de una interfaz, clase o colaboración.

En cualquier momento, un objeto se encuentra en un estado en particular. Una persona puede ser recién nacida, infante, adolescente, joven o adulta. Un elevador se moverá hacia arriba, estará en estado de reposo o se moverá hacia abajo.

Diagramas de secuencias

Los diagramas de clases y los de objeto representan información estática. No obstante, en un sistema funcional los objetos interactúan entre spi, y tales interacciones suceden con el tiempo. El diagrama de secuencias UML muestra la mecánica de la interacción con base en tiempos.

El Diagrama de Secuencia es uno de los diagramas más efectivos para modelar interacción entre objetos en un sistema. Un diagrama de secuencia se modela para cada caso de uso. Mientras que el diagrama de caso de uso permite el modelado de una vista ’business’ del escenario, el diagrama de secuencia contiene detalles de implementación del escenario, incluyendo los objetos y clases que se usan para implementar el escenario, y mensajes pasados entre los objetos.

Un diagrama de secuencia muestra los objetos que intervienen en el escenario con líneas discontinuas verticales, y los mensajes pasados entre los objetos como vectores horizontales. Los mensajes se dibujan cronológicamente desde la parte superior del diagrama a la parte inferior; la distribución horizontal de los objetos es arbitraria.

Diagrama de Actividades

Son un tipo especial de diagramas de estados que se centra en mostrar el flujo de actividades dentro de un sistema. Los diagramas de actividades cubren la parte dinámica de un sistema y se utilizan para modelar el funcionamiento de un sistema resaltando el flujo de control entre objetos.

Las actividades que ocurren dentro de un caso de uso o dentro del comportamiento de un objeto se dan, normalmente, en secuencia. El Diagrama de Actividad es un diagrama de flujo del proceso multi-propósito que se usa para modelar el comportamiento del sistema. Los diagramas de actividad se pueden usar para modelar un Caso de Uso, o una clase, o un método complicado.

Un diagrama de actividad es parecido a un diagrama de flujo; la diferencia clave es que los diagramas de actividad pueden mostrar procesado paralelo (parallel processing). Esto es importante cuando se usan diagramas de actividad para modelar

procesos ’bussiness’ algunos de los cuales pueden actuar en paralelo, y para modelar varios hilos en los programas concurrentes.

Diagrama de colaboraciones

El Diagrama de Colaboración presenta una alternativa al diagrama de secuencia para modelar interacciones entre objetos en el sistema. Mientras que el diagrama de secuencia se centra en la secuencia cronológica del escenario que estamos modelando, el diagrama de colaboración se centra en estudiar todos los efectos de un objeto dado durante un escenario. Los objetos se conectan por medio de enlaces, cada enlace representa una instancia de una asociación entre las clases implicadas. El enlace muestra los mensajes enviados entre los objetos, el tipo de mensaje (sincrónico, asincrónico, simple, blanking, y ’time-out’), y la visibilidad de un objeto con respecto a los otros.

Los elementos de un sistema trabajan en conjunto para cumplir con los objetivos del sistema, y un lenguaje de modelado deberá contar con una forma de representar esto. El diagrama de colaboraciones UML, fue diseñado con este fin.

Diagrama de componentes

Muestra la organización y las dependencias entre un conjunto de componentes. Cubren la vista de la implementación estática y se relacionan con los diagramas de clases ya que en un componente suele tener una o más clases, interfaces o colaboraciones. El moderno desarrollo de software se realiza mediantes componentes, lo que es particularmente importante en los procesos de desarrollo en equipo.

Diagrama de distribución

Representan la configuración de los nodos de procesamiento en tiempo de ejecución y los componentes que residen en ellos. Muestran la vista de despliegue estática de una arquitectura y se relacionan con los componentes ya que, por lo común, los nodos contienen uno o más componentes.

El diagrama de distribución UML muestra la arquitectura física de un sistema informático. Puede representar los equipos y dispositivos, mostrar sus interconexiones y el software que se encontrará en cada máquina. Cada computadora está representada por un cubo y las interacciones entre las computadoras están representada por líneas que conecta a los cubos.

Como puede notarse, los diagramas del UML nos permiten examinar un sistema desde distintos puntos de vista. Es importante recalcar que en un modelo UML no es necesario que aparezcan todos los diagramas. De hecho, la mayoría de los modelos UML contiene un subconjunto de los diagramas que se han indicado.

3.4 Proceso de desarrollo de software con UML y el modelo CMM

El proceso de desarrollo de un sistema, sea o no para la Web, se enfrenta a diversos problemas. Dentro del proceso de solución de un problema, es el método subyacente el que sugiere cómo se utiliza el conocimiento para construir una solución. Esto incluye el sugerir qué diagramas se deben usar, y la perspectiva y el nivel de abstracción a emplear para trazar e interpretar dichos diagramas.

El ciclo de vida para UML consiste en una serie de ciclos cada uno de los cuales produce una nueva versión del producto, es decir, se basa en la evolución de prototipos ejecutables. Cada ciclo está compuesto por fases y cada una de estas fases está compuesta por un número de iteraciones.

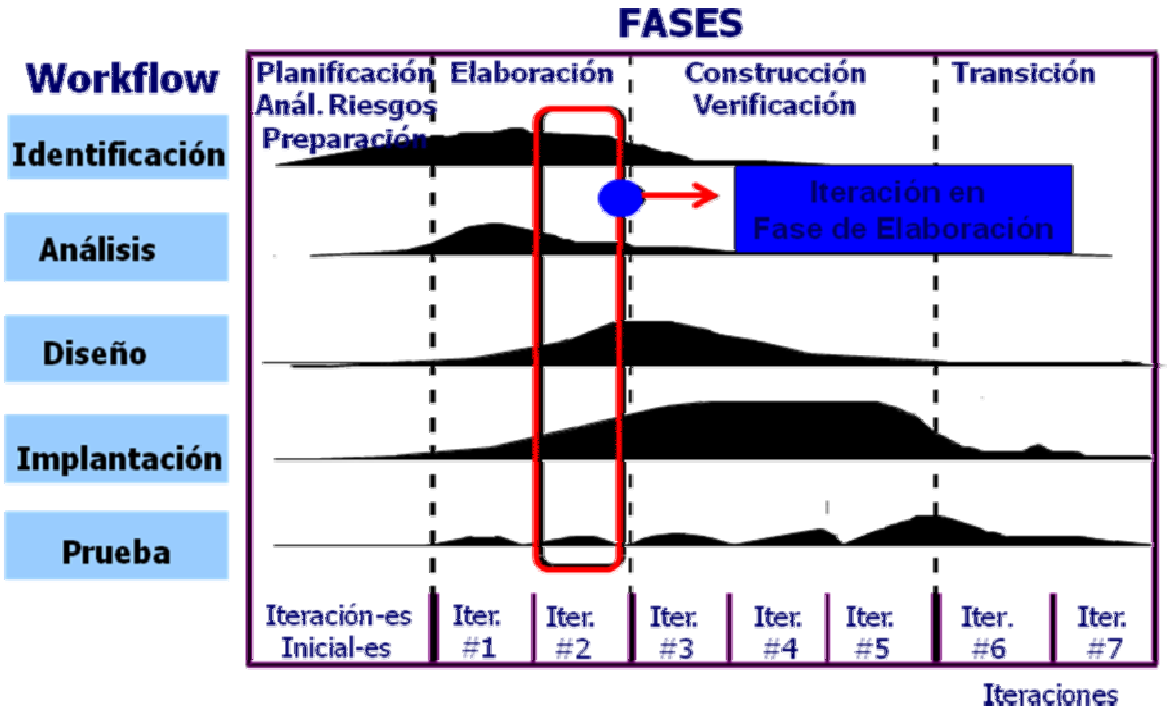


Fig. 3.3 Proceso de desarrollo.

Tal y como se refleja en la Figura 3.3, las fases son las siguientes:

Estudio de oportunidad: Se define la funcionalidad y capacidades del producto. *Elaboración:* Tanto la funcionalidad como el dominio del problema se estudian en profundidad, se define una arquitectura básica y se planifica el proyecto considerando los recursos disponibles. *Construcción:* El producto se desarrolla a través de iteraciones, y cada iteración involucra tareas de análisis, diseño e implementación. La arquitectura básica de las fases de estudio y análisis se refina de manera incremental conforme se construye (se permiten cambios en la estructura), se programa y prueba, y se documenta tanto el sistema construido como el manejo del mismo. *Transición:* El producto se entrega al usuario. También se incluyen tareas de marketing, instalación, configuración, soporte, mantenimiento, etc., así como de finalización de los manuales de usuario. Por supuesto, estas tareas se realizan en iteraciones. En cada iteración se reproduce el ciclo de vida en cascada (a menor escala) para acometer los objetivos que se establecen en función de la evaluación de las iteraciones precedentes. Cada iteración se basa en la construcción de un número reducido de escenarios que se centran primero en los riesgos más importantes y determinan las clases y las categorías a construir en dicha iteración. Cada iteración comprende:

- Planificar la iteración (estudio de riesgo).
- Análisis de los Casos de Uso y escenarios.
- Diseño de opciones arquitectónicas.

- Codificación y pruebas. La integración del nuevo código con el existente de iteraciones anteriores se hace gradualmente durante la construcción.
- Evaluación de la entrega ejecutable (evaluación del prototipo en función de las pruebas y de los criterios definidos).
- Preparación de la entrega (documentación e instalación del prototipo).

3.4.1 Modelado

Para producir software que cumple su propósito hay que obtener los requerimientos del sistema, esto se consigue conociendo de una forma disciplinada a los usuarios y haciéndolos participar de manera activa para que no queden "cabos sueltos". Para conseguir un software de calidad, que sea duradero y fácil de mantener hay que idear una sólida base arquitectónica que sea flexible al cambio. Para desarrollar software rápido y de manera eficiente, minimizando el trabajo de recodificación y evitando crear miles de líneas de código inútil hay que disponer, además de la gente y las herramientas necesarias, de un enfoque apropiado.

Para conseguir desarrollar de manera industrial se obtenga un producto de calidad, es completamente necesario seguir pautas y no abordar los problemas de manera somera, con el fin de obtener un modelo que represente lo suficientemente bien el problema que se ha de abordar. El modelado es la espina dorsal del desarrollo de software de calidad. Se construyen modelos para poder comunicarnos con otros para explicar el comportamiento del sistema a desarrollar, para comprender de una mejor manera nosotros mismos, para controlar el riesgo y en definitiva para poder atacar problemas que sin el modelado su resolución seria imposible, tanto desde el punto de vista de los desarrolladores (no se pueden cumplir los plazos estimados, no se consigue ajustar los presupuestos, entre otras actividades) como desde el punto de vista del cliente, el cual, si finalmente se le entrega el producto del desarrollo, se encontrará con infinidades de problemas, desde que no se cumplen las especificaciones hasta fallos que dejan inutilizados el sistema.

Cuando nos referimos al desarrollo de software en el ámbito industrial, no se pretende que la capacidad de modelar se reduzca a empresas que disponen de gran numero de empleados o empresa que han de abordar proyectos eminentemente grandiosos, si no que nos referimos a la capacidad de obtener un producto comercial (sea cual sea su coste o tamaño) que cumple lo que en la industria se suele denominar como ***calidad total*** y que además pueda reportar beneficios a corto o media plazo, evitando en ocasiones implantaciones casi eternas debido a la falta de previsión o al haber abordado los problemas muy a la ligera.

Por todas estas razones es inevitable el uso de modelos. Un modelo es una simplificación de la realidad. El modelo nos proporciona los planos de un sistema, desde los más generales, que proporcionan una visión general del sistema, hasta los más detallados. En el modelo se han de incluir los elementos que tengan más relevancia y omitir los que no son interesantes para el nivel de abstracción que se ha elegido. A través del modelado conseguimos cuatro objetivos:

- Los modelos nos ayudan a visualizar como es o queremos que sea un sistema.

- Los modelos nos permite especificar la estructura o el comportamiento de un sistema.
- Los modelos nos proporcionan plantillas que nos guían en la construcción de un sistema.
- Los modelos documentan las decisiones que hemos adoptado.

En la realidad, no siempre se hace un modelado formal, la probabilidad de que exista un modelado formal para abordar un sistema es inversamente proporcional a la complejidad del mismo , esto es, cuanto más fácil sea un problema complejo el uso del modelado nos ayuda a comprenderlo, mientras que cuando tenemos un problema fácil el uso del modelado que hacemos se reduce a representar mentalmente el problema o, como mucho, a escribir unos cuantos garabatos sobre un papel.

3.4.1.1 Principios básicos del modelado

Existen cuatro principios básicos, estos son frutos de la experiencia en todas las ramas de la ingeniería.

a) *La elección de que modelos se creen influye directamente sobre cómo se acomete el problema*. Hay que seleccionar el modelo adecuado para cada momento y dependiendo de que modelo se elija se obtendrán diferentes beneficios y costes. En la industria del software se ha comprobado que un modelado orientado a objetos proporciona unas arquitecturas más flexibles y readaptables que otros por ejemplo orientados a la funcionalidad o a los datos.
b) *Todo modelo puede ser expresado a diferentes niveles de precisión*. Es necesario seleccionar el nivel de detalle que se desea ya que en diferentes partes de un proyecto y en diferentes etapas se tendrán unas determinadas necesidades.
c) *Los mejores modelos están ligados a la realidad*. Lo principal es tener modelos que nos permitan representar la realidad lo más claramente posible, pero no sólo esto, tenemos que saber, exactamente cuando se partan de la realidad para no car en la ocultación de ningún detalle importante.
d) Un único modelo no es suficiente. Cualquier sistema que no sea trivial se afronta mejor desde pequeños modelos casi independientes, que los podamos construir y estudiar independientemente y que nos representa las partes más diferenciadas del sistema y sus interrelaciones.

3.4.2 Proceso de desarrollo de software

Actualmente existe una fuerte tendencia en la Ingeniería de Software de la definición y mejora de procesos de software. Se está relacionando la calidad de los productos de software con la capacidad y la madurez de los procesos que los genera. Los modelos de referencia disponibles para la industria de desarrollo de software, tal como CMM – SW, esta definido en términos de procesos. El problema común de estas propuestas es que acumulan gran volumen de información en forma casi puramente textual. Esto crea un costo importante para las empresas porque tienen que asimilar este conocimiento para luego interpretarlo en el contexto de su organización. Todos los modelos sugieren documentar los procesos.

El proceso de software es un conjunto de personas, estructuras de organización, reglas, políticas, actividades con procedimientos, componentes de software, metodologías y herramientas utilizadas o creadas específicamente para conceptualizar, desarrollar, ofrecer un servicio, innovar y extender un producto de software.

El proceso de software es una actividad muy compleja que involucra aspectos sociales, administrativos, técnicos y de producción. Los dos primeros, están reflejados en la definición de los roles y de sus responsabilidades, así como en las actividades de tipo planeación, provisión de recursos, seguimiento y verificación. Los aspectos técnicos y productivos se manifiestan vía la definición de ciclos de vida, metodologías, lenguajes de programación, herramientas y actividades de control o de aseguramiento de calidad.

En el siguiente capitulo se detallara el proceso de software, siguiendo los proceso de CMM-SW, y la integración de UML.

Capítulo 4

Prototipo de Proceso de Desarrollo de Software

El prototipo que se propone en este documento solo comprende la parte de nuevos proyectos (nuevos desarrollos), tomando en cuenta CMM-SW y UML. Para la parte de mantenimiento de software se requiere un proceso distinto al que se propone.

Las tareas de ingenierías de software deben ser definidas, integradas y ejecutadas consistentemente para generar productos de software de buena calidad.

El siguiente prototipo de proceso de desarrollo de software es sentar las bases para el desarrollo de nuevos proyectos de software de manera consistente, permitiendo brindar una empresa a su cliente soluciones costo-efectivas y de alta calidad.

El objetivo del proceso es documentar las actividades para el desarrollo de software en cada unas de las fases del proceso. Guiar a los involucrados en el flujo de cada uno de los procedimientos que se requieren para el desarrollo de software.

4.1 Etapas del Proceso

El proceso de software que se propone aquí consta de 8 fases:

- Anteproyecto
- Inicio de Proyecto
- Análisis
- Diseño
- Construcción
- Pruebas
- Pase a Producción
- Cierre de Proyecto

4.2 Indicadores de Desempeño

Los indicadores de desempeño o métricas que se aplicaran para medir el nivel de calidad del proceso de desarrollo son:

- Desviación en Esfuerzo
- Desviación en Cronograma
- Densidad de Defectos
- Eficacia de Revisión del Código
- Índice de Satisfacción al Cliente

4.3 Descripción de la Fases del Proceso

4.3.1 Anteproyecto

4.3.1.1 Propósito

En esta fase se realiza la valoración de los requerimientos que el cliente solicita, además de una parte primordial en todo proyecto, se lleva a cabo la negociación con el cliente del costo económico, el tiempo de entrega y el personal que se requiere para el desarrollo del proyecto, para ello se requiere se lleve a cabo una muy buena valoración de los requerimientos. Y finalmente si el cliente esta satisfecho con el costo y tiempo, el cliente autorizara la propuesta que se halla hecho.

4.3.1.2 Documentos de Entrada

Se recibe el Documento de Definición de Proyecto

4.3.1.3 Actividades

- Recibir el requerimiento de Proyecto y asignar responsable de valoración.

- Entender los requerimientos.

- Seleccionar y valorar la solución del requerimiento.

- Ejecutar decisión, análisis y resolución para seleccionar la solución.

- Identificar los riesgos asociados.

- Solicitar cotización de bienes y servicios.

- Valorar la solución elegida técnicamente.

- Ejecutar decisión, análisis y resolución para seleccionar la solución mejor.

- Negociar con el cliente.

4.3.1.4 Verificar

- Revisar las valoraciones
- Revisar el documento de definición de proyecto., y documento de requerimiento funcional por pares.
- Revisar el documento de requerimiento funcional, documento de definición de proyecto, y la propuesta definitiva de proyecto con el cliente.

4.3.1.5 Criterios Finales

- El cliente ha firmado el documento de requerimiento funcional.
- El cliente ha firmado el documento de definición de proyecto con criterio de aceptación.
- El plan de trabajo esta llenado parcialmente, solo hasta la fase de anteproyecto.
- El documento de definición de proyecto debe de contar con requerimientos de recursos, requerimientos de Software y Hardware.

4.3.1.6 Entregables

- Documento de Requerimiento Funcional.
- Documento de Definición de Proyecto.
- Plan de trabajo sólo las actividades de la fase de anteproyecto.
- Documento de Valoración de Proyecto.
- Documento de Valoración Técnica.
- Documento de Enfoque de Solución.
- Propuesta Definitiva de Proyecto.
- Solicitud de Cotización.
- Minutas y Acuerdos.
- Reporte de Decisión, Análisis y Resolución.

4.3.2 Inicio de Proyecto

4.3.2.1 Propósito

Obtener las herramientas requeridas de software y hardware, configurar el ambiente creado, desarrollar el plan de trabajo y compartirlo con las áreas involucradas para lograr el desarrollo exitoso del software. Y finalmente capacitar a los miembros del proyecto.

4.3.2.2 Documentos de Entrada

- Propuesta Definitiva de Proyecto, cuyo documento ha sido autorizado por el cliente.
- Se selecciona el líder de proyecto y analista.
- El Documento de Requerimiento Funcional, Documento de Definición de Proyecto y Documento de Valoración de Proyecto se encuentran disponibles.

4.3.2.3 Actividades

- Valoración Técnica.

- Preparar y compartir el Plan de Trabajo con todos los involucrados.
- Seleccionar el equipo de trabajo para la ejecución del proyecto.

- Identificar los riesgos asociados.

- Ejecutar el procedimiento de Reclutamiento y Selección de Personal, si se requiere.
- Crear y actualizar los documentos del Proyecto.

- Solicitar capacitación para el proyecto.
- Conseguir los requerimientos de hardware y software.

- Reunión de Kick Off.
- Seguimiento del estado del proyecto.

- Configurar las herramientas requeridas para la realización del proyecto.

4.3.2.4 Verificar

- Verificar el ambiente.

- Revisar el plan de trabajo y plan integrado del proyecto.

4.3.2.5 Criterios Finales

- El proyecto se ha iniciado
- Los requerimientos de personal, hardware, software y otros requerimientos se han cumplido.
- Se ha capacitado al equipo.
- Se ha creado el plan de trabajo y ha sido compartido con los involucrados en el proyecto.
- La reunión de Kick Off ha sido realizado.
- Se ha creado los controles documentales (versiones).

4.3.2.6 Entregables

- Plan de trabajo completo.
- Detección de Necesidades de Capacitación.
- Documento de Formalización de Proyecto.
- Documento de Requisición de Compra (si se requiere).
- Minutas y Acuerdos.
- Documento de Reunión de Kick Off
- Reporte de Seguimiento de Proyecto.
- Base de Datos de Riesgos (si se requiere).
- Documento de Seguimiento del Proveedor (si se requiere).

4.3.3 Análisis

4.3.3.1 Propósito

Analizar la información obtenida por parte del cliente sobre el proyecto, analizar las posibles contradicciones o conflictos que pueda tener, así como descomponer el proyecto en diversos partes, establecer los límites del mismo.

Finalizar las especificaciones del requerimiento del software, realizar el análisis de reutilización de anteriores proyectos, y preparar el plan de pruebas y casos de pruebas.

4.3.3.2 Documentos de Entrada

- Plan de trabajo.
- Documento de Requerimiento Funcional.
- Documento de Definición de Documento.
- Documento de Valoración de Proyecto

4.3.3.3 Actividades

- Hacer el análisis de impacto y Preparación de Matriz de Trazabilidad.
- Identificar los riesgos asociados.

- Valorar la solución elegida técnicamente.
- Generar la especificación de Requerimientos de Software
- Realizar Análisis de Reutilización.
- Ejecutar Decisión, Análisis y Resolución para seleccionar los componentes reutilizables.

- Obtener la firma de aceptación del cliente.

- Generar revisión por pares.
- Preparar Plan de Pruebas

- Preparar la especificación del diseño de pruebas.
- Actualizar el plan de trabajo.

4.3.3.4 Verificar

- Revisar los elementos reutilizables.
- Revisar la especificación de requerimientos de software.
- Revisar por pares la especificación de requerimientos de software, matriz de trazabilidad, casos de uso, plan de pruebas y análisis de impacto.

Estas actividades se realizan durante esta fase lo cual permite analizar los requerimientos solicitados por el cliente, valorando y ejecutando la revisión por pares a los productos de trabajo generados, eliminando los defectos para asegurar la calidad.

4.3.3.5 Criterios Finales

- El cliente ha firmado la especificación de requerimientos de software.
- La matriz de trazabilidad se ha creado.
- Los elementos reutilizables han sido identificados.
- Se encuentran disponibles el plan de pruebas y la especificación de Diseño de Pruebas.

4.3.3.6 Entregables

- Documento de Análisis de Impacto.
- Documento de Valoración Técnica.
- Documento de Especificación de Requerimientos de Software.
- Documento de Casos de Uso.
- Documento de Requerimientos de Seguridad.
- Matriz de Trazabilidad.
- Documento de Diseño de Conversión.
- Plan de Pruebas.
- Especificación de Diseño de Pruebas
- Necesidades de Ambiente
- Minutas y Acuerdos.
- Especificación de Escenarios de Rendimiento.

4.3.4 Diseño

4.3.4.1 Propósito

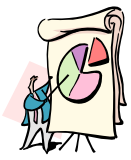
Diseñar la arquitectura de la aplicación solicitada por el cliente, realizando la descripción del sistema a partir de la especificación de diseño de los componentes de software con el nivel de detalle suficiente para guiar su construcción en descomposición del sistema.

Diseño de Componentes, también se realiza la preparación del plan de pruebas unitarias y casos de pruebas unitarias.

4.3.4.2 Documentos de Entrada

- Casos de Uso.
- Especificación de Requerimientos de Software firmado por el cliente.
- El Plan de Pruebas
- Especificación de diseño de Pruebas.

4.3.4.3 Actividades



- Preparar y/o actualizar los estándares para codificar.
- Ejecutar el procedimiento de mejoras de proceso de software.
- Desarrollar el prototipo, si se requiere.

- Definir la arquitectura de la solución.
- Identificar los riesgos asociados.
- Diseñar base de datos y archivos.

- Crear y/o actualizar plan de contingencia.
- Identificar los componentes.
- Actualizar el plan del trabajo.

- Coordinar uso de fuentes.
- Hacer análisis de reutilización.
- Ejecutar Decisión, Análisis y Resolución para decidir la mejor secuencia de integración.

- Planificar y especificar el diseño de las pruebas unitarias.
- Identificar los riesgos asociados.
- Actualizar el plan de trabajo.

- Actualizar la Matriz de Trazabilidad y el Análisis de Impacto.
- Identificar los riesgos asociados.
- Configurar las herramientas a utilizar para la fase de construcción.
- Actualizar el plan de trabajo.

4.3.4.4 Verificar

- Revisar las especificaciones de los componentes.
- Revisar por pares los documentos creados.
- Revisar el plan de contingencia.
- Revisar plan de uso de fuentes de código.
- Revisar los elementos reutilizables.

4.3.4.5 Criterios Finales

- El ambiente para la construcción ha sido configurado.
- La arquitectura de la aplicación esta disponible.
- Las especificaciones de los componentes (Base de datos, archivos, tablas, programas, reportes, bach, rutinas, etc.) están disponibles.
- Plan de contingencia creado/actualizado.
- Plan de pruebas unitarias, los casos de pruebas unitarias están preparados.
- Los estándares para la fase de construcción están disponibles.
- Plan de uso de Integración Secuencia de Integración están preparados.
- La Matriz de Trazabilidad ha sido actualizada.

4.3.4.6 Entregables

- Documento de Especificación y Desarrollo de Prototipo.
- Modelo General de Datos.
- Arquitectura de la Aplicación.
- Solicitud de Cambios para Tablas.
- Documento de Coordinación de Fuetes.

- Especificación de Componentes.
- Plan de Contingencia.
- Secuencia de Integración.
- Especificación de Diseño de Pruebas Unitarias.
- Análisis de Impacto.
- Matriz de Trazabilidad.
- Documento de Casos de Uso.
- Plan de Pruebas.
- Solicitud de Documentación y Validación del Modelo de Datos.
- Validación de Modelado de Datos.
- Documentación de estándares de programas/codificación.
- Plan de Trabajo Actualizado.
- Documento de Volumetría.
- Requerimientos de seguridad.
- Diseño de Conversión.
- Minutas y Acuerdos.
- Orden de Trabajo.
- Seguimiento de Fabrica de Software (creado/actualizado).
- Mapa Aplicativo (actualizado con los componentes de software creados / modificados)
- Base de Datos de Riesgos.

4.3.5 Construcción

4.3.5.1 Propósito

Construir el código fuente de los módulos o funciones en los lenguajes de programación requeridos que integraran al sistema y realizar las pruebas unitarias para asegurar el correcto funcionamiento de cada modulo o función que componen al sistema. Todo esto en el ambiente de desarrollo. Además de preparar los Manuales de Usuario, Manual de Mantenimiento y Materiales de Capacitación.

4.3.5.2 Documentos de Entrada

- Todos los documentos de diseño están disponibles.
- Planeación y Especificación de Diseño de Pruebas Unitarias están disponibles.

4.3.5.3 Actividades

- Definir las librerías para almacenar los componentes del sistema.
- Ejecutar la Fase de Registro.
- Obtener los recursos logísticos.
- Construir el código.

- Realizar la prueba unitaria y corregir los defectos.
- Elaborar la documentación para control de cambios y operaciones.
- Realizar la prueba unitaria y corregir los defectos.
- Elaborar la documentación para control de cambios y operaciones.

- Realizar la prueba de sistema y corregir los defectos.
- Elaborar los manuales de usuario y mantenimiento.
- Crear y actualizar los documentos de proyecto.

- Elaborar el material de capacitación y capacitar al cliente.
- Realizar logística de capacitación.
- Generar y realizar la promoción de la petición de instalación para un ambiente pre-producción.
- Definir las librerías para almacenar los componentes del sistema.

- Ejecutar la fase de registro.
- Ejecutar la fase de revisión.
- Ejecutar la fase de instalación / Liberación de Pruebas.
- Actualizar la matriz de trazabilidad.
- Actualizar el plan de trabajo.

4.3.5.4 Verificar

- Revisar el código, la matriz de trazabilidad y el análisis de impacto por pares.
- Revisar los resultados de las pruebas unitarias.
- Revisar el plan de capacitación para el cliente.
- Revisar por pares los documentos generados.

Estas actividades se realizan durante el desarrollo de la fase de Diseño, permitiendo desglosar los requerimientos solicitados para el proyecto, valorando y ejecutando la revisión por pares a los productos generados, identificando los defectos para asegurar la calidad.

4.3.5.5 Criterios Finales

- La librería para guardar los componentes de sistema esta definida.
- Se ha realizado las pruebas unitarias y se han corregido los defectos.
- El código esta disponible para pruebas.
- El ambiente para la calidad del producto ha sido creada.
- La documentación para control de cambios esta disponible.
- Se cuenta los procedimientos de operación.
- El manual se usuario y el manual de mantenimiento están creados / actualizados.
- Se cuenta con el VoBo de la capacitación del cliente.
- Se cuenta con el número de peticiones de instalación.
- La arquitectura de la aplicación esta disponible.
- El ambiente para la construcción ha sido configurado.
- Plan de uso de fuentes de código se ha coordinado.

4.3.5.6 Entregables

- Bitácora de resultados de revisión.
- Planeación y Especificación de Diseño de las Pruebas Unitarias.
- Evidencia de la Prueba Unitaria.
- Revisión de código interno y corrección de los defectos.
- Procedimientos de operación.
- Manual de usuario.
- Manual de mantenimiento.
- Plan de capacitación.
- Necesidades de ambientes.
- Diseño de conversión.
- Consolidación de defectos de conversión.
- Minutas y acuerdos.
- Orden de trabajo.
- Seguimiento fábrica de software.

4.3.6 Pruebas

4.3.6.1 Propósito

Realizar pruebas integrales y de rendimiento para garantizar la funcionalidad del sistema, verificando que se cumple con los requerimientos del cliente y así poder asegurar la calidad del producto antes de su pase a producción. Con esto contamos con la aceptación del cliente.

4.3.6.2 Documentos de Entrada

- Todos los documentos, el código y programas de construcción estas disponibles.
- Los ambientes para probar y los casos de pruebas están disponibles.
- El cliente esta informado y disponible para realizar la prueba de aceptación y la capacitación.

4.3.6.3 Actividades

- Ejecutar pruebas integrales.
- Evaluar y corregir los defectos encontrados en el ambiente.
- Ejecutar pruebas de rendimiento.

- Evaluar y corregir los defectos encontrados en el ambiente.

- Crear / Homologar el ambiente.
- Determinar el impacto del nuevo sistema.
- Evaluar y corregir los defectos encontrados en el ambiente.

- Crear / Homologar el ambiente.
- Actualizar la matriz de trazabilidad.
- Aceptación del producto por parte del cliente.
- Evaluar y corregir los defectos encontrados en el ambiente.

- Ejecutar el procedimiento de generación de Solicitud Cambio de Alcanze.
- Ejecutar la fase de instalación / liberación en pruebas.
- Actualizar los manuales de usuario y mantenimiento.
- Crear y actualizar los documentos de proyectos.
- Elaborar el material de capacitación y capacitar al cliente.
- Realizar logística de capacitación.
- Documentar las buenas prácticas y las lecciones aprendidas.
- Liberar al personal que sea posible.
- Actualizar la matriz de trazabilidad.
- Actualizar el plan de trabajo.

4.3.6.4 Verificar

- Revisar los resultados de las pruebas hechos en ambiente.
- Revisar el manual de usuario y el manual de mantenimiento.
- Revisar por pares los documentos mencionados en Evaluar y Corregir los Defectos Encontrados en el Ambiente.

Estas actividades se realizan durante el desarrollo de la fase de prueba, con esto se verifica el éxito de las pruebas integrales y de rendimiento para garantizar la funcionalidad del sistema, identificando los defectos para asegurar la calidad.

4.3.6.5 Criterios Finales

- Se han creado los defectos y el código al que se le realizó pruebas integrales, pruebas de rendimiento e impacto y la aceptación del cliente están disponibles para la puesta en producción.
- El manual de usuario, mantenimiento han sido liberados.
- Los clientes están capacitados en el nuevo sistema.
- Las lecciones aprendidas y las buenas prácticas son compartidas y puestas en el repositorio organizacional.

4.3.6.6 Entregables

- Bitácora de resultados de revisión.
- Reporte periódico de resultados de pruebas.
- Reportes de revisión de permanencia.
- Reporte de revisión de impacto.
- Evidencia de las pruebas.
- Acta de liberación de pruebas.

- Evaluación del curso.
- Código aceptado por el cliente.
- Materiales de capacitación.
- Requisición de personal interno y externo.
- Evaluación de personal para contratación y salida.
- Minutas y acuerdos.
- Plan de pruebas.
- Matriz de trazabilidad.
- Conversión de datos.
- Manual de usuario.
- Manual de Mantenimiento.
- Plan de capacitación.
- Orden de trabajo.
- Seguimiento fabrica de software.

4.3.7 Pase a Producción

4.3.7.1 Propósito

El sistema se pone en producción y se verifica su buen funcionamiento por medio de un monitoreo en el cual se da una garantía de que en caso de existir errores estos serán eliminados. Se realiza el traspaso del sistema al equipo de mantenimiento.

4.3.7.2 Documentos de Entrada

- El cliente es capacitado en el nuevo sistema o el los cambios realizados al sistema.
- Plan de puesta en producción fue comunicada al cliente.
- Se ha liberado el código, el manual de usuario y el manual de mantenimiento.

4.3.7.3 Actividades

- Solicitar la instalación en producción.
- Ejecutar la fase de programación.
- Ejecutar la fase de liberación / instalación en producción.
- Dar soporte de garantía.
- Ejecutar el procedimiento de codificación y construcción.
- Ejecutar la fase de liberación / instalación en producción.
- Ejecutar la fase de cierre.

- Ejecutar el proceso de gestión de incidencias.
- Ejecutar el proceso de gestión de cambios.
- Actualizar los manuales de usuario y mantenimiento.
- Crear y actualizar los documentos del proyecto.

- Asignar el responsable de mantenimiento.

4.3.7.4 Verificar

- Revisar los defectos reportados desde producción.
- Revisar el manual de mantenimiento y manual de usuario.

Estas actividades se realizan durante el desarrollo de la fase de pase a producción, con esto se monitorea el buen funcionamiento del sistema para garantizar la calidad del producto.

4.3.7.5 Criterios Finales

- Los sistemas están funcionando apropiadamente en el ambiente de producción.
- Se obtuvo la aceptación final del cliente.
- Los sistemas fueron traspasados al equipo de mantenimiento.

4.3.7.6 Entregables

- Acta de liberación.
- Manual de usuario.
- Manual de mantenimiento.
- Orden de trabajo
- Seguimiento de fábrica de software.

4.3.8 Cierre de Proyecto

4.3.8.1 Propósito

Se realiza el cierre de proyecto con el objetivo de liberar al personal y los requerimientos de software y hardware que fueron utilizados, se guardan las buenas prácticas y las lecciones aprendidas y se presenta un reporte de finalización al comité.

4.3.8.2 Documentos de Entrada

- Se obtuvo la aceptación final del cliente.
- Los sistemas fueron traspasados al equipo de mantenimiento.
- Todos los requerimientos contractuales fueron cerrados.

4.3.8.3 Actividades

- Solicitar a la oficina de proyectos cambiar el estatus del proyecto.
- Liberar a los recursos.

- Ejecutar bajas de recursos externos.
- Renovar y cambiar de recursos externos.
- Liberar el equipo de cómputo y otros requerimientos logísticos.
- Actualizar el documento de buenas prácticas y de lecciones aprendidas y guardar los elementos reutilizables en el repositorio organizacional.

- Formalización de cierre de proyectos.

4.3.8.4 Verificar

- Revisar las buenas prácticas y lecciones aprendidas, así como elementos reutilizables en otro proyecto. Con el fin de identificar los componentes reutilizables en el proyecto y se comparten las buenas practicas y lecciones aprendidas.

4.3.8.5 Criterios Finales

- Se obtuvo la percepción del cliente.
- Se ha liberado a todo el personal, hardware, software y otros requerimientos logísticos.
- Desempeño del proyecto ha estado evaluado por el comité.
- Las buenas prácticas, lecciones aprendidas y el reporte de cierre del proyecto ha sido puesto en el repositorio organizacional.

4.3.8.6 Entregables

- Percepción del cliente.
- Requisición de personal interno y externo.
- Presentación del cierre del proyecto.
- Repositorio de buenas prácticas y lecciones aprendidas.
- Evaluación de proveedores.
- Evaluación de personal para contratación y salida.
- Minutas y acuerdos.

Finalmente se realiza el seguimiento del proyecto a través de la generación de reportes e informe, que permiten conocer las desviaciones del proyecto con respecto a su planificación inicial en costo, esfuerzo y horas para poder aplicar acciones preventivas y/o correctivas.

Capítulo 5

Conclusiones y trabajos futuros

Los procesos bien definidos y maduros son la parte primordial, en el desarrollo de una aplicación. De estos procesos depende la implementación, las pruebas, integración y su mantenimiento del sistema. Con la ayuda de procesos maduros se lograra realizar un sistema dentro de un tiempo y un presupuesto estimado, por lo que la definición de procesos maduros impactara en el desarrollo, costo, funcionalidad y satisfacción del cliente del producto final.

Por tal razón en esta tesis se ha propuesto un proceso de desarrollo de software, que contempla todas las etapas para el desarrollo de nuevos proyectos, esta propuesta surge de la metodología de CMM-SW y con la ayuda de UML para el modelado del sistema. Además de proponer los documentos que son útiles en cada etapa, lo que ayuda a interpretar la funcionalidad del sistemas a los diferentes usuario involucrados en el desarrollo del sistema.

El prototipo de procesos de desarrollo de software que en esta tesis se establece, permite la creación de software con calidad, porque satisface los criterios de calidad, rastreabilidad, mantenibilidad y usabilidad. Esta propuesta favorece el poder estimar el costo y el tiempo que puede tener un sistemas en su desarrollo, esto con la ayuda de entender con la ayuda del cliente el problema a resolver y garantizar que los requerimientos funcionales recolectados sean implantados y que no sufran muchos cambios, ya que esto a futuro causa problemas en el desarrollo del sistema, en tiempos y costo del mismo.

El uso de un lenguaje estándar como UML disminuye la ambigüedad del diseño, cuyo uso básicamente esta en los diagramas de: casos de uso, clases, secuencias, colaboración, entre otras. Dado que con estos se puede indicar como implementar el sistema y proporcionar la arquitectura del sistema.

La mezcla de la metodología CMM-SW y UML puede llegar a ser una excelente opción para la creación de nuevos proyectos, obteniendo lo mejor de la metodología se podría generarse un espacio amplio de soluciones y con UML un mejor diseño del problema para tener al final un producto con alta calidad.

La aplicación del modelado con UML en un proyecto permitirá a una organización llevar a cabo proyectos con alta calidad y poder lograr certificarse en el nivel 3 de CMMI.

En trabajos futuros se propondrán métricas formales, que sustenten los resultados observados en los procesos de desarrollo de software propuesto. Incrementar, fortalecer, mejorar ciertos procesos que ayuden incrementar los criterios de calidad con la ayuda de las mejores prácticas que surjan en el uso de este prototipo.

Bibliografía

[1] A. degboyega Ojo y E. Estevez. “Object Oriented Analisys and Design with UML. Training Course”, Version 1.0, October 2005.

[2] Candace L. Conwell, R. Enright, Marcia A. Stutzman. “Capability Maturity Models Support of Modeling and Simulation Verification, Validation, and Accreditation. Proceedings”, *the 2000 Winter Simulation Conference*.

[3] Charles G. Menk III. “System Security Engineering Capability Maturity Model and Evaluations: Partners Within the Assurance Framework.” *Department of Defense*. June 1996.

[4] Christian Fehr Allgood. “The Claims Library Capability Maturity Model.” *Thesis submitted to the Faculty of the Virginia Polytechinc Institute*. June 11th, 2004.

[5] Dieg J. Bodas Sagi. “El CMM y la mejora continua del proceso de software.” Junio, 2003.

[6] Erol Biberoghu and Hisham Haddad. A Survey of Industrial Experiences with CMM and the Teaching of CMM Practices. CSIS Department, Kennesaw State University.

[7] Felix Bachmann, Len Bass, Charles Buhman, Santiago Comella-Dorba, Fred Long, John Robert, Robert Seacord, Kart Wallnau. Vlumen II: “Technical Concepts of Components-Based Software Engineering 2nd Edition. Technical Report”, *Carnegie Mellon Software Engineering Institute*, Pittsburgh, PA 15213-3890, May, 2000.

[8] J. Manuel Cueva Lovelle. “Calidad del Software”. *Conferencia, Grupo GIDIS*, Universidad de la Pampa, 21 de Octubre de 1999.

[9] F. Niessink, V. Clerc and H. van Vliet. “The IT Service Capability Maturity Model”. Version 0.4. June, 2004.

[10] F. Niessink, V. Clerc, T. Tijdink, and Hans van Vliet. “The IT Service Capability Maturity Model”. Version 1.0, June, 2005.

[11] F. Niessink. “IT Service CMM.” Version 1.0.2, January, 2003

[12] G. Sunyé, A. Le Guennec, and Jean M. Jézequed. “Desing Patterns Application in UML”.

[13] H. Oktaba y C. Alquicira Esquivel. “Modelo gráfico de la Administración de Configuraciones del SW-CMM nivel2”.

[14] H. Oktaba and C. Alquicira Esquivel. “Diagramas de Procesos: Una extensión de diagramas de actividades de UML para el modelado de procesos de software.” *UNAM*, México, D.F.

[15] H. Curtis. “Describing the Capability Maturity Model.” *Measure Special Edition* 2001 Capability Maturity Model.

[16] H. Saiedian and N. Carr. “Characterizing a Sotware Process Maturity Model for Small Organizations.” Department of Computer Science, University of Nebraska at Omaha, Nebraska, USA.

[17] Intel. People Capability Maturity Model, “How Intel IT Uses People CMM to Improve Workforce Practices”. *Intel Information Technology White Paper*. February, 2003.

[18] J.. Herbsleb, Dennis R. Goldenson. “A System Survey of CMM Experience and Results.” Software Engineering Institute, Carnegie Mellon University, Pittsburgh.

[19] J. Herbsleb, D. Zubrow, Dennis Goldenson, “Will Hayes and Mark Paulk. Software Quality and y the Capability Maturity Model.” Communications of the ACM. June 1997/Vol. 40 No. 6.

[20] J Earthy. “Usability Maturity Model: Processes.” *Trump* Version. August 1998.

[21] J Earthy. “Usability Maturity Model: Human Centredness Scale”. Version 1.2. European Usability Support Centres.

[22] Judith G. Brodman and Donna L. Johnson. “What Small Businesses and Small Organizations Say About the CMM. LOGOS International”, Inc. IEEE, 1994.

[23] Kathryn M. Dodson, EDS, Dr. Humbert F. Hofmann, General Motors, Gowri S. Ramani, Hewlett Packard, Deborah K. Yedlin, General Motors. “Adapting CMMI for Acquisition Organizations: A Preliminary Report. Carnegie Mellom Software Engineering Institute”, Pittsburg, PA 15213-3890, June 2006

[24] L. Fernández Sanz, Pedro J. Lara Bercial. “Mejora de la Calidad en Desarrollos Orientados a Objetos Utilizando Especificaciones UML para la Obtención y Precedencia de Casos de Prueba.” *Revista de Procesos Y Métricas de las Tecnologías de la Información*, VOL. 1, No 3. Diciembre 2004.

[25] Linda Ibrahim. “Using an Integrated Capability Maturity Model The FAA Experience.” Proceeding of the Tenth Annual Internacional Symposium of the Internacional Council on Systems Engineering (INCOSE), Minnesota, July 2000

[26] L. Rene Flores. “La Reusabilidad en la Ingeniería de los sistemas de información.” *Tesis. Facultad de Ciencias de la Computación*. Julio, 2004.

[27] Louis A. Poulin. “A Comparative Analisys of Process Maturity Level and Quality”. *IEEE* Canadian. Spring, 2006.

[28] M. Foegen, C. Raak, M. “Tandler. Capability Maturity Model Integration. Carnegie Mellon University”. Software Engineering Institute.

[29] Marck C. Paulk, Bill Curtis, Mary Beth Chrissis, and Charles V. Weber. "The Capability Maturity Model for Software."

[30] Mark C. Paulk, Charles V. Weber, and Mary Beth Chrissis, "The Capability Maturity Model for Software." Carnegie Mellon University, Software Engineering Institute, USA.

[31] Mark C. Paulk. "How ISO 9001 Compares with the CMM", *IEEE Software*. January 1995.

[32] Marck C. Paulk, Hill Curtis, Mary Beth Crisis and Charles V. Weber. "Capability Maturity Model for Software, Version 1.1", *Technical Report*, CMU/SEI-93-TR-024 ESC-TR-93-177, February, 1993.

[33] Mark C. Paulk, Charles V. Weber, Suzanne M. Garcia, Mary Beth Chrissis and Marilyn Bush. "Key Practices of the Capability Maturity Model, Version 1.1", *Technical Report*, CMU/SEI-93-TR-025 EC-TR-93-178, February, 1993.

[34] M. Grottke." Software Process Maturity Model Study". *IST-1999-55017*.

[35] N. McFarland. "Capability Maturity Model Integration. Introduction to CMMI" *coerce participant*, February, 2006

[36] P. Jalote. "Moving from ISO9000 to Higher Levels of the CMM. Department of Computer Science and Engineering", Indian Institute of Technology Karpur.

[37] P. Forradellas, G. Pantaleo y Dr Juan Rogers. "El modelo CMM/CMMI- Cómo garantizar el éxito del proceso de mejoras en las organizaciones, superando los conflictos y tensiones generados por su implementación. it-Mentor."

[38] P. Maller, C. Ochoa y J. Silva. "Agilizando el Proceso de Producción de Software en un Entorno CMM de Nivel 5".

[39] R. Baker. "The Corporate Policts of CMM Rating". *Communications of the ACM*. September 1996/ Vol. 39, No. 9.

[40] P Letelier, V Anaya. "Integrando Especificaciones Textuales y elementos de modelado UML en un Marco de Trabajo para Trazabilidad de Requisitos".

[41] R.. Oshana, Richers C. Linger. "Capability Maturity Model Software Development Using Cleanroon Software Engineering Principles – Results of an Industry Project". Proceeding of the 32nd Hawaii International Conference on System Sciences, 1999

[42] R. Oshana and Frank P. Coyle. "Implementing Cleanroom Software Engineering into a Mature CMM-Based Software Organization".

[43] Rolf W. Reitzig, John B. Miller, Raymond L. Kile. "Achieving Capability Maturity Model Integration (CMMI) Maturity Level 2 Using IBM Rational Software Solutions. Cognence".

[44] Stephen Mellor. “UML Distilled: From Difficulties to Assets. IEEE Software”, *Published by the IEEE Computer Society*, 2005.

[45] “Systems Security Engineering Capability Maturity Model SSE-CMM Model Description Document”, Version 3.0, June 15, 2003.

[46] T. Orci. “Capability Maturity Model for Extra Small Organizations Level 2”. Version 1.0, Octubre 2000.

[47] “The Capability Maturity Model Integration (CMMI) for Senior Managers. European Software Institute (ESI)”

[48] U. Arranz. “La importancia de la certificación: de ISO a SW CMM. Sociedad de la información”, abril 2004.

[49] Vogten, H., Verhooren, M., and Koper, R. “UML Diagrams for IMS Learning Desing”. September, 2002.

[50] Watts S. Humphrey. “Relating the Team Software Process (TSP) to the Capability Maturity Model for Software (SW-CMM)”. *Report. Cargnegie Mellon Software Engineering Institute*, Pittsburgh, PA 15213-3898. March 2003.

[51] X. Ferré Grau, María I. Sánchez. “Desarrollo Orientado a Objetos con UML”. Facultad de Informática. UPM.

[52] Zhou Zhiying, “CMM in Uncertain Environments”. Communications of the *ACM*, August 2003/ Vol, 46, No. 8.