Capítulo 5

Ingeniería de software

El desarrollo de sistemas de información se caracteriza por ser una difícil empresa. El grado de dificultad se incrementa cuando el software producido tiene características educativas. Ante esto, se verifica que el desarrollo de software educativo integra tanto los aspectos ergonómicos (i.e. aquellos estudiados en la ingeniería de usabilidad) y los aspectos educativos (i.e. aquellos estudiados por las teorías educativas), como los aspectos funcionales (i.e. aquellos estudiados por la ingeniería de software).

De esta manera, se añade a la presente tesis la ingeniería de software en donde se investigan, desde el enfoque basado en procesos[1], los siguientes aspectos:

- El análisis y diseño de modelos conceptuales de flujos de trabajo.
- Su estructura de organización.

[1]De acuerdo con Hammer y Champy (1993), la mayoría de las personas de negocios no usan el enfoque de análisis y diseño basado en proceso, sino que usan enfoques orientado a tareas, a trabajo, a personas y a estructuras.

- La gestión y control de calidad.

En este capítulo, se presentan los conceptos fundamentales para el entendimiento de los vínculos existentes entre la ingeniería de software y el modelo de test propuesto (véase Capítulo 7). El panorama presentado parte de la definición de software y sus características, pasando por los modelos y estándares internacionales de proceso de software, hasta la definición de modelos de proceso de test.

5.1 El software y su ingeniería

El software es un sistema informático compuesto por un conjunto de instrucciones que, cuándo se ejecutan en un dispositivo físico (i.e. el hardware) produce resultados de acuerdo con los objetivos y función principal predeterminada. Dicho conjunto de instrucciones está organizado en estructuras de datos[2] que permiten la manipulación de la información. Según Pressman (1995), el ingenio del "desarrollador" (o del equipo de desarrollo) es el único aspecto que limita el diseño (i.e. organización y complejidad) de una estructura de datos, aunque existen algunas estructuras de datos clásicas (e.g. escalar, vector secuencial, listas encadenadas y árbol jerárquica) que pueden ser usadas para la construcción de estructuras más complejas.

La estructura de datos asociada al tipo de hardware usado influye decisivamente en el desempeño del software. Así pues, durante el diseño y programación de un software es importante seleccionar el tipo de estructura de datos más adecuado respecto al tipo de procesamiento previsto.

Usando el concepto abstracto del software (i.e. parte lógica del sistema "computacional"[3]) como punto de partida, se presentan algunas características que distinguen el software del hardware y ofrecen una buena

[2]De acuerdo con Pressman (1995, p. 432-433),

> "*la estructura de datos es una representación de la relación lógica entre elementos de datos individuales. (...) La estructura de datos determina la organización, los métodos de acceso, el grado de asociatividad y las alternativas de procesamiento de informaciones*".

[3]Se define un sistema "computacional" como un entorno de trabajo, estudio, entretenimiento u otra actividad, el cual está compuesto por tres elementos interrelacionados: el software (i.e. la parte lógica), el hardware (i.e. la parte física) y el peopleware (i.e. la parte humana).

compresión de su significado (Pressman, 1995):

- Su desarrollo o proyecto no se basa en manufactura clásica sino que en la ingeniería.
- Considerando que un determinado software está implantado, éste no se deteriora con el tiempo, la suciedad, las temperatura altas y la vibración, como es el caso del hardware. Por otra parte, la evolución del hardware impone cambios de paradigmas en la filosofía de diseño del software, de manera que se identifica la implementación de nuevas versiones con el propósito de adaptarlas a las nuevas tecnologías.
- La mayoría de los sistemas informáticos son desarrollados a medida (i.e. se concibe el software como una unidad completa). Sin embargo, actualmente se observa una tendencia que consiste en diseñar y implementar componentes de software que puedan ser usados según la necesidad. En la literatura, se pueden encontrar diversos estudios sobre el tema (e.g. Veryard (1997) y García y Fuente (2000)).

En este sentido, se desarrolla un software con el objetivo de facilitar todas aquellas tareas en dónde se verifica la necesidad de procesamiento de grandes cantidades de informaciones. Esto implica la determinación de áreas de aplicación de software (e.g. software básico, de tiempo real, comercial, científico y de ingeniería, educativo, etc.), lo que, según Pressman (1995), es una tarea difícil. Dentro el contexto de la presente investigación, se trata con el software educativo representado las aplicaciones multimedia usadas en educación y formación a distancia.

Zahran (1998) argumenta que el desarrollo de software, más que un intento, es una empresa caracterizada por un proceso continuado de desafío. Además, la influencia del software dentro del mundo de los negocios y de la vida cotidiana de las personas se vuelve crítica debido no sólo a los problemas conceptuales y de diseño, sino también a los problemas técnicos y económicos generados durante el proyecto de software.

Las propuestas académicas y de la industria para reducción de estos problemas como por ejemplo las herramientas CASE[4] y las metodologías de desarrollo, han permitido que el software evolucionara provocando, a partir de las necesidades creadas, la aparición de una nueva disciplina: la ingeniería de software.

Dicha disciplina, incluyendo la ingeniería de usabilidad (véase Capítulo 4), define estándares para el desarrollo de software de calidad basados en diversos indicadores como, por ejemplo, la corrección, la eficacia, la eficiencia, la fiabilidad, la facilidad de comprensión, de uso y de mantenimiento, la interoperabilidad, la portabilidad, la reusabilidad y la robustez.

De acuerdo con Naur y Randall (1968), la ingeniería de software es

> "*el establecimiento y uso de sólidos principios de ingeniería con el propósito de obtener económicamente un software que sea fiable y que funcione eficientemente en máquina reales*".

Humphrey (1989) presenta una definición similar resaltando, de manera

[4]La sigla CASE es acrónimo de *Computer-Aided Software Engineering*. Esta herramienta consiste en un sistema compuesto por software, hardware y una base de datos que ofrece soporte a la ingeniería de software, permitiendo el análisis estructurado, la implementación y test de sistemas informáticos.

general, la calidad del software. Usando la definición citada anteriormente como punto de partida, Totland y Conradi (1995) comentan que el área de modelado del proceso de software (*software process modeling*) puede ser considerado como un subdominio de la ingeniería de software.

El desarrollo formal de software se caracteriza por la definición de los requerimientos de la aplicación, la especificación de los objetivos, el diseño iterativo, los procesos continuados de test y la implementación del software. Para cada una de estas actividades se diseñan procesos (y subprocesos) que serán implementados a través de un lenguaje de programación.

Considerando la perspectiva del diseño centrado en el usuario (véase Secciones 4.4 y 4.6, páginas 101 y 122 respectivamente), el estudio y la evolución de los métodos y técnicas de control y gestión de la calidad han sido determinantes para la creación de la ingeniería de usabilidad.

Ésta se caracteriza por ser uno de los macro-procesos de la ingeniería de software. En este sentido, Sulaiman (1996) presenta una propuesta de integrar la usabilidad al ciclo de vida de producción de software a través de la inserción de métodos de medición de calidad de software y en las fases iniciales de proyecto.

Se realiza la integración a partir de la definición de los procesos de software. Por lo tanto, se presentan en las próximas secciones la contextualización y conceptualización necesaria del enfoque de diseño y desarrollo de software orientado a procesos. De esta manera, se concentra el estudio en las actividades de mejora de procesos de software y procesos de test.

5.1.1 Proceso de software

Para lograr el entendimiento de proceso de software, es necesario, en primer lugar, presentar una definición de **proceso**. En la literatura, se puede encontrar una amplia variedad de definiciones de la palabra "proceso". Por consiguiente, se observa que dentro de esta variedad existen distintas interpretaciones, sean por extensión, cobertura o orientación (Zahran, 1998).

Hammer y Champy (1993, p. 35) presentan una definición adecuada para el ámbito de la presente tesis debido a su carácter general. Según los autores, un proceso es

> "*una colección de actividades que toman uno o más tipos de entradas y crea una salida que es de valor para cliente*".

En el contexto de la presente investigación, la colección de actividades se caracteriza por todos los procedimientos que componen el ciclo de vida de software (i.e. procesos de concepción, diseño, desarrollo, mantenimiento, documentación, control de calidad, test, gestión, entrenamiento, etc.) desde las fases iniciales hasta su uso por el cliente (i.e. el usuario final). Cabe resaltar que un software se caracteriza por tres grandes actividades: la entrada de datos, su procesamiento y la salida de las informaciones procesadas.

De acuerdo con Zahran (1998), un proceso está compuesto por tres aspectos. El primer aspecto es la definición del proceso que generalmente está representado por un documento que especifica las actividades y procedimientos para el proceso. El segundo aspecto se refiere a la transferencia del conocimiento del proceso para aquellos que lo van a ejecutar. Finalmente, el

tercer aspecto son los resultados obtenidos del proceso después de su ejecución. Según el autor, el enfoque orientado a proceso permite:

- La sincronización y armonía entre las actividades desempeñadas por cada individuo y los objetivos comunes del equipo.
- La garantía de la consistencia de las actividades y sus resultados.
- El uso de técnicas objetivas de medición del desempeño de cada individuo respecto a las actividades asignadas.
- Debido a la reducción de la dependencia en cada individuo, se consigue que el equipo pueda repetir un determinado proceso, aunque individuos ajenos sean asignados a la actividad.

Por otra parte, según Curtis, Kellner y Over (1992), las perspectivas en la representación del proceso son la funcional, la de conducta, la de la organización y la de la información. Usando la definición de proceso y sus consideraciones como punto de partida, Boehm (1988, p. 61) argumenta que

> "*las funciones primarias de un modelo de proceso de software son determinar el orden de las fases involucradas en el desarrollo y evolución de software y establecer los criterios de transición para avanzar de una fase a la próxima*".

La ingeniería del software comprende un conjunto estructurado de pasos que representan los "paradigmas de la ingeniería del software". Estos paradigmas se basan en la naturaleza del proyecto y de la aplicación, en los métodos y las herramientas que serán usados en el proyecto, los controles y los productos o servicios desarrollados (Pressman, 1995).

5.2 Ciclo vida de software

En la literatura, se pueden encontrar varios modelos de ciclo de vida de desarrollo de software. Todos ellos se basan en diversos aspectos tales como la planificación, el análisis, el diseño e implementación de software. Cada uno de estos métodos están asociados a los marcos y paradigmas tecnológicos de su entorno y época.

A continuación, se presenta una revisión de los modelos de ciclo de vida de desarrollo de software.

5.2.1 Modelo codificar-y-fijar

Boehm (1988) comenta que en la época inicial del desarrollo de software, se ha usado el modelo codificar-y-fijar (*code-and-fix*). Éste contiene dos pasos:

- Escribir algún código.
- Fijar los problemas en el código.

De esta manera, inicialmente se implementaba algún código y, a continuación, se pensaba sobre los requerimientos, el diseño, los test y el mantenimiento. Las principales dificultades del modelo consisten en la estructuración insuficiente de código, la carencia de correspondencia con las necesidades del usuario y el coste excesivo debido a la deficiente preparación de los test y modificaciones.

Ante esto, se observa la necesidad de reorganización del modelo en etapas, incorporando elementos de planificación, coordinación y control.

5.2.2 Modelo de etapas

A partir de la necesidad de construir grandes sistemas de software (e.g. *Semi-Automated Ground Environment* - SAGE), el modelo de etapas ha sido desarrollado con el propósito de facilitar la mejora de los problemas involucrados en los proyectos de software (Benington, 1956). El modelo determina que el software será desarrollado en etapas consecutivas:

- Plan operativo.
- Especificación operativa.
- Especificación de codificación.
- Codificación o implementación.
- Test de parámetros.
- Test de integración o montaje.
- Registro (*shakedown*).
- Evaluación del sistema.

Además, el modelo enfatiza la documentación resultante de cada una de las etapas, de manera que formaliza los procedimientos de planificación y de control. No obstante, se identifica no sólo una laguna entre el equipo de desarrollo, caracterizado básicamente por la figura del programador, y el usuario del software, sino también la necesidad de rehacer todo el sistema cuando se detectan problemas en su instalación.

5.2.3 Modelo en cascada

Este modelo, también conocido como ciclo de vida clásico (Pressman, 1995) ha sido inicialmente presentado por Royce (1970) y se caracteriza por un refinamiento del modelo de etapas, en el cual se realza los ciclos de retroalimentación entre las etapas con el objetivo de minimizar el coste de retrabajo del proyecto. El modelo incorpora el "prototipaje" al ciclo de vida de desarrollo de software. En la Figura 5.1, se presentan la etapas del modelo.

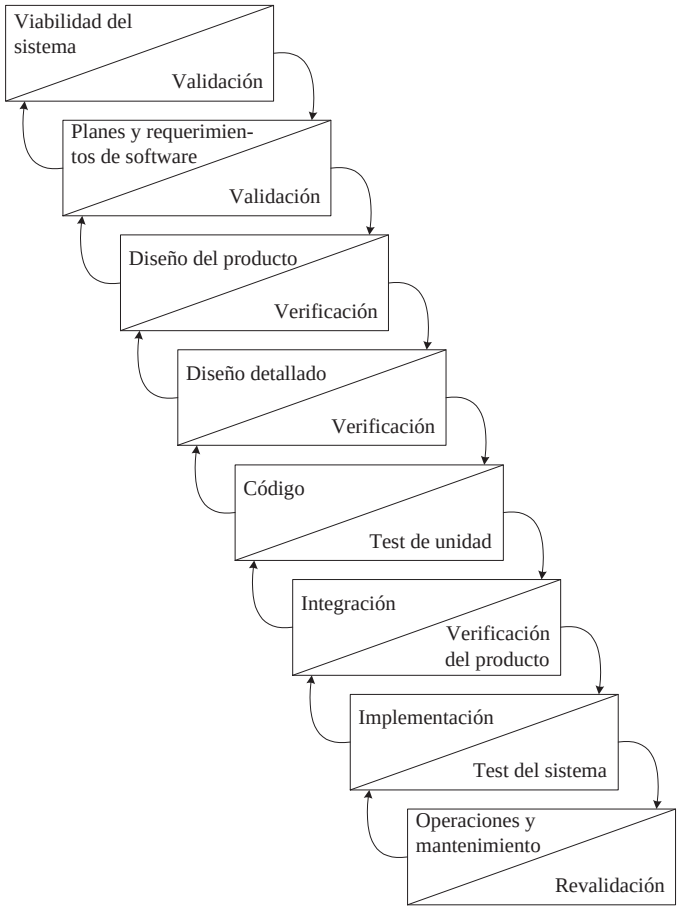

Figura 5.1: El modelo en cascada del proceso de software. Fuente: Boehm (1988)

Aunque los ciclos de retroalimentación permitan realizar extensas revisiones y refinamientos, se identifican algunas dificultades causadas por el énfasis en la elaboración de documentos completos como criterios de finalización para las etapas de requerimientos y diseño. Por otra parte, se observa la problemática de su aplicación a sistemas interactivos, debido, una vez más, al distanciamiento entre los usuarios y el equipo de desarrollo (Boehm, 1988; Sommerville, 1996).

5.2.4 Modelo de desarrollo orientado a prototipos

Este modelo consiste en un procedimiento que permite al equipo de desarrollo diseñar y analizar una aplicación que representa el sistema que será implementado (McCracken y Jackson, 1982). Dicha aplicación, llamada prototipo, está compuesta por los componentes que se desean evaluar (i.e. las funciones principales). Las etapas del modelo son:

- Investigación preliminar.
- Colecta y refinamiento de los requerimientos y proyecto rápido:
  - Análisis y especificación del prototipo.
  - Diseño y construcción del prototipo.
  - Evaluación del prototipo por el cliente.
  - Refinamiento del prototipo.
- Diseño técnico.
- Programación y test.

- Operación y mantenimiento.

De acuerdo con Pressman (1995), los problemas identificados en el modelo de desarrollo orientado a prototipos consisten, por una parte, en que "se ve lo que parece ser" (i.e. lo que el usuario examina es una representación del sistema con funcionalidades restringidas), por tanto, no se considera la calidad global del software, ni sus aspectos de mantenimiento. Por otra parte, el equipo de desarrollo hace concesiones de implementación basadas en el uso de sistemas operativos y lenguajes de programación impropios, sin importar la inadecuación posterior del producto.

5.2.5 Modelo de desarrollo evolutivo

De acuerdo con Boehm (1988), las etapas del modelo de desarrollo evolutivo

> "*consisten en expandir los incrementos de un producto de software operativo, siendo las direcciones de evolución determinadas por la experiencia operativa*".

Se identifica, por tanto, la importancia en la obtención de un sistema de producción flexible y expansible, permitiendo la adaptación de los cambios de requerimientos que no han sido planeados. Se asocia este modelo a un lenguaje de cuarta generación. En este sentido, Sommerville (1996) comenta que este modelo es el más apropiado para el desarrollo de sistemas interactivos y de sistemas de inteligencia artificial. Las etapas básicas del modelo son:

- Especificación inicial.
- Desarrollo del producto o servicio.

- Implementación.
- Uso.
- Evaluación.
- Nuevas versiones.
- Re-especificación.

Sin embargo, se observan algunas dificultades técnicas como, por ejemplo, la problemática de la integración de aplicaciones independientes, los casos de "esclerosis de información" debido a los trabajos temporales alrededor de algunas deficiencias del software y el reemplazo de un nuevo software en un gran sistema existente (Boehm, 1988).

5.2.6 Modelo de transformación

El modelo de transformación ha sido propuesto como una solución a las dificultades de código del modelo de desarrollo evolutivo y del modelo codificar-y-fijar analizado dentro del modelo en cascada. Desde la óptica de Boehm (1988, p. 63) el modelo

> "*asume la existencia de una capacidad de convertir automáticamente una especificación formal de un producto de software en un programa que satisfaga la especificación*".

Según el autor, los pasos de este modelo son:

- Especificación formal del mejor entendimiento inicial del producto deseado.

- Transformación automática de la especificación en código.
- Un ciclo iterativo, si necesario, para mejorar el desempeño del código resultante, lo que ofrece una guía de optimización al sistema de transformación.
- Ejercicio o evaluación del producto resultante.
- Un ciclo interactivo externo para ajustar las especificaciones basadas en el resultado de la experiencia operativa, y para rederivar, reoptimizar, y ejercitar o evaluar el producto de software ajustado.

Las dificultades del modelo se caracterizaban por la carencia generalizada de capacidades de transformación automáticas sólo disponibles para pequeños productos, la posibilidad de evolución no planeada y los problemas de mantenimiento de base de conocimiento.

5.2.7 Modelo espiral

Boehm (1988) presenta el modelo espiral del proceso de software (véase Figura 5.2), el cual ha evolucionado basándose en los diversos refinamientos del modelo en cascada y del desarrollo orientado a prototipos. El modelo espiral se caracteriza por un conjunto de ciclos progresivos, en los cuales se identifican los objetivos de cada parte del producto que está siendo desarrollado, las propuestas alternativas de implementación y las restricciones impuestas por dichas alternativas al software.

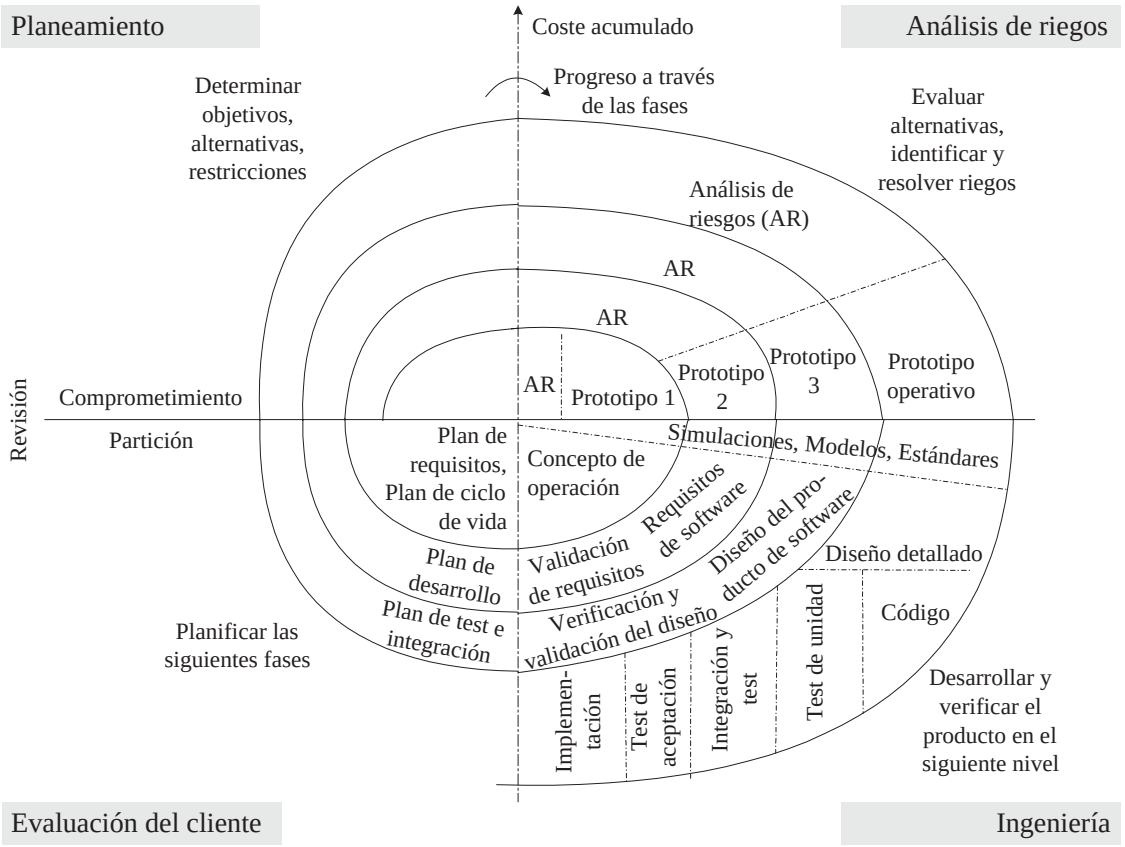

Figura 5.2: El modelo espiral del proceso de software. Fuente: Boehm (1988)

El modelo se divide en cuatro cuadrantes que representan las actividades principales para el ciclo de vida de desarrollo de software, resumidos por Pressman (1995) en:

- Planeamiento.
- Análisis de riegos.
- Ingeniería.
- Evaluación del cliente.

De acuerdo con Boehm (1988, p. 65) el ciclo de vida propuesto por el modelo se basa en cuatro preguntas fundamentales:

1. "*¿Cómo empieza la espiral?*"
2. "*¿Cómo uno rompe los ciclos de la espiral cuando es apropiado terminar un proyecto apenas iniciado?*"
3. "*¿Por qué la espiral se acaba bruscamente?*"
4. "*¿Qué pasa con la mejora (o el mantenimiento) del software?*"

Debido a la reducción de riesgos determinada por el método evolutivo del modelo, éste representa un enfoque más apropiado para el desarrollo de grandes, complejos y ambiciosos sistemas de información. Además, el modelo espiral posee un nivel alto de capacidades de entorno de soporte de software y es flexible, lo que permite acomodar diversas alternativas técnicas y objetivos de usuario.

5.3 Modelos y estándares internacionales de proceso de software

Como se ha comentado, la presente investigación se basa en el análisis de las interrelaciones entre los aspectos ergonómicos y el aprendizaje del usuario de aplicaciones multimedia en entornos de formación a distancia. Para ello, se definen en las Seccione 5.1 y 5.2 de este capítulo, los fundamentos teóricos sobre la ingeniería de software.

En consecuencia, se puede decir que a través de los modelos y estándares de evaluación de proceso de software se podrá establecer estrategias para la definición de criterios de evaluación de usabilidad de sistemas interactivos multimedia.

5.3.1 Capability Maturity Model for Software

Paulk, Curtis, Chrissis y Weber (1993) definen el modelo de madurez de capacidad (*Capability Maturity Model* - CMM) como un modelo que establece los niveles por los cuales las organizaciones de software hacen evolucionar sus definiciones, implementaciones, mediciones, controles y mejorías de sus procesos de software.

Además, el CMM permite la definición del grado de madurez de las prácticas de gestión y de ingeniería de software de dichas organizaciones. De esta manera, se puede determinar cómo madura una determinada organización y cuales son las acciones de mejora prioritarias para sus procesos de software.

El enfoque inicial del CMM ha sido el proceso de software. Sin embargo, se puede encontrar aplicaciones del modelo en otros campos como por ejemplo CMM para personas (*People CMM*), CMM para ingeniería de sistemas (*Systems Engineering CMM*), CMM para la gestión de productos integrados (*Integrated Product Management CMM*) y otros (Zahran, 1998).

5.3.1.1 Estructura del CMM

El esquema del modelo CMM está compuesto por cinco niveles de madurez de acuerdo con la capacidad del proceso de software y definidos por los objetivos de los procesos que, cuando satisfechos, permiten evolucionar al próximo nivel ya que uno o más componentes importantes del proceso de software han sido estabilizados (Paulk, Curtis, Chrissis y Weber, 1993). De esta manera, los niveles de madurez ayudan a las organizaciones a definir prioridades para sus esfuerzos de mejora. En la Figura 5.3, se presenta los cinco niveles de madurez del proceso de software.

- Nivel 1 - Inicial: se caracteriza como *ad hoc* o caótico. Pocos procesos son definidos.
- Nivel 2 - "Repetible" o repetición: se caracteriza como disciplinado. Se establecen procesos básicos de gestión.
- Nivel 3 - Definido: se caracteriza como estándar y consistente.
- Nivel 4 - Gestionado: se caracteriza como predicable. Hay un preocupación en la medición detallada de la calidad del proceso de software y del producto.
- Nivel 5 - Optimización: se caracteriza como mejora continua a partir de la realimentación (*feedback*) cuantitativa.

Cada nivel de madurez tiene una estructura interna (véase Figura 5.4) compuesta por los siguientes componentes:

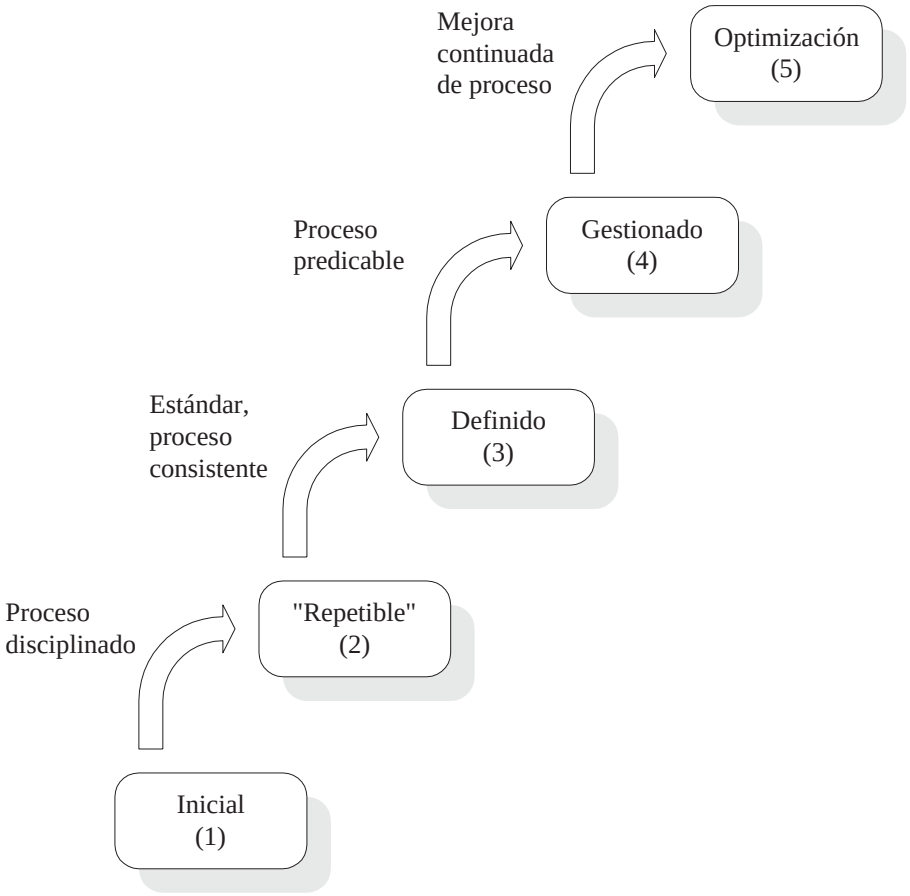

Figura 5.3: Los niveles de madurez del proceso de software del CMM. Fuente: Paulk, Curtis, Chrissis y Weber (1993)

- Nivel de madurez: Representa un indicador evolutivo que permite alcanzar la madurez del proceso de software.

- Áreas de proceso clave (*Key Process Areas* - KPA): Son las subestructuras de cada nivel que indican las áreas a las que una organización debería dirigir su atención con el propósito de mejorar su proceso de software. Se asigna cada conjunto de KPA a un nivel de madurez (excepto al nivel uno), como se muestra en la Figura 5.5. Zahran (1998) comenta que el CMM no detalla todas las áreas de proceso involucradas en el desarrollo y mantenimiento de software, sino que se enfoca en las áreas clave que contribuyen en la mayoría de las capacidades de proceso.

- Objetivos: Este componente delimita las KPA a través de la definición de su alcance, límites y intenciones. Los objetivos, por lo tanto, determinan las restricciones que deben ser superadas por la organización para que ésta pueda alcanzar mejores niveles de madurez.

- Características comunes: Son atributos que indican si la implementación e institucionalización de una KPA son eficaces, repetidas y duraderas. Paulk, Curtis, Chrissis y Weber (1993) presentan cinco características comunes:

  - El compromiso en desempeñar las acciones que garantizan el establecimiento del proceso.
  - La habilidad en desempeñar dichas acciones.
  - Las actividades desempeñadas para implementar las KPA.

  - La medición y el análisis del estado y eficacia de las actividades desempeñadas.
  - La implementación para la verificación de las actividades desempeñadas respecto a los procesos establecidos.

- Prácticas clave: Éstas describen la infraestructura y actividades que contribuyen para la implementación e institucionalización efectiva de la KPA.

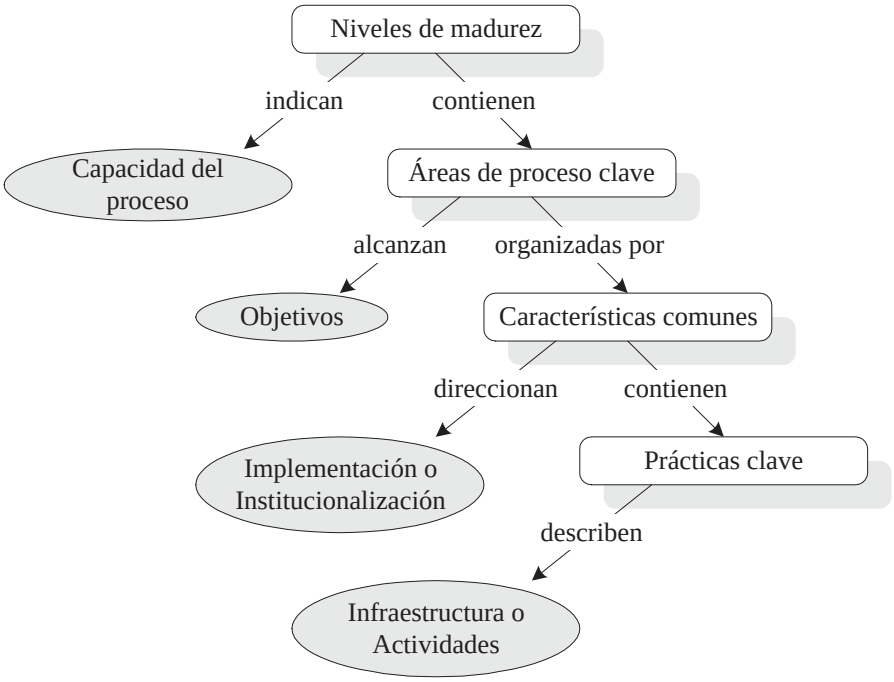

Figura 5.4: Estructura interna del CMM. Fuente: Paulk, Curtis, Chrissis y Weber (1993)

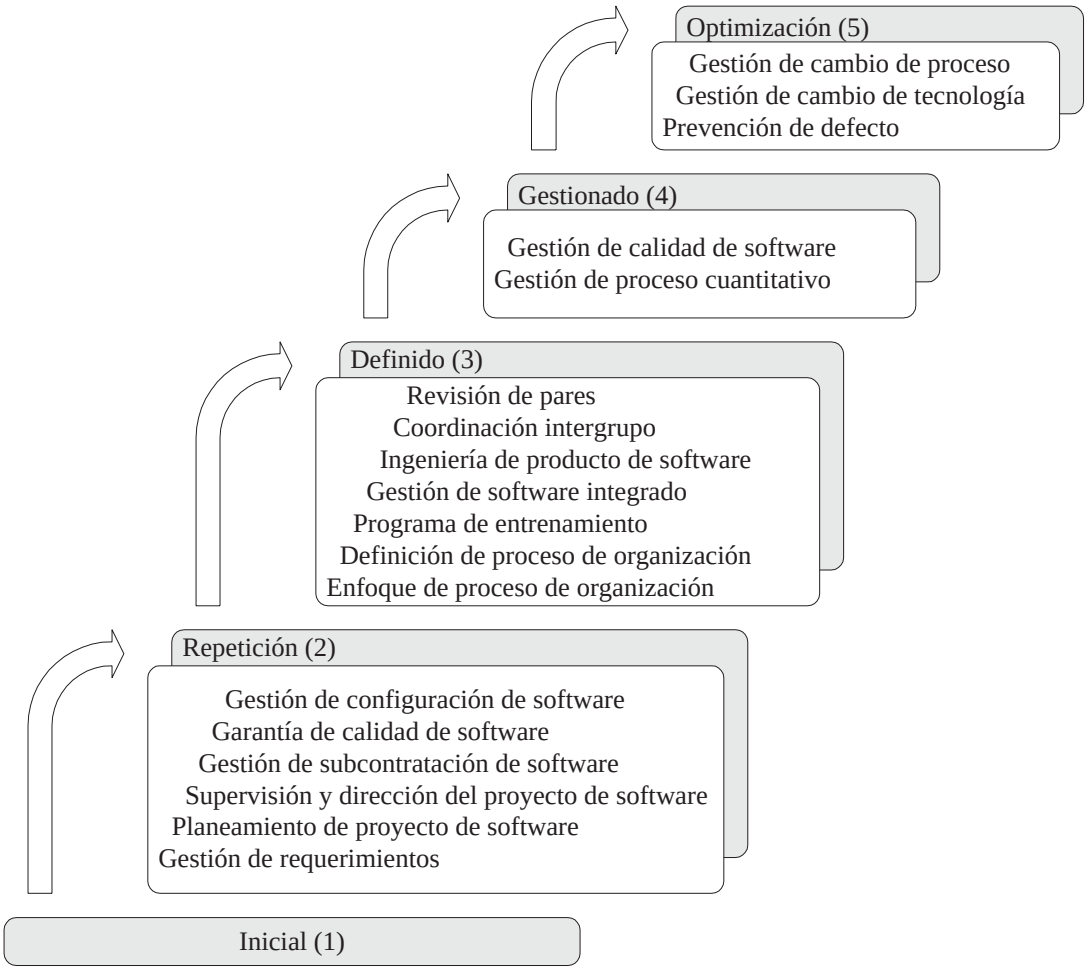

Figura 5.5: Las KPA representadas en cada uno de los cinco niveles de madurez del proceso de software del CMM. Fuente: Paulk, Curtis, Chrissis y Weber (1993)

De acuerdo con Zahran (1998), el CMM es un mapa para la definición de pasos (i.e. un esquema de evaluación) que una organización necesita realizar para moverse desde procesos caóticos a procesos de mejora continua. Las evaluaciones basadas en CMM utilizan el cuestionario del *Software Engineering Institute* (SEI) como un instrumento de evaluación. Su enfoque se dirige a temas relacionados con procesos y sus cuestiones se basan en los objetivos de las KPA del CMM.

5.3.2 Estándar ISO/IEC 15504

La Organización Internacional para Estandarización (*International Organization for Standardization*) y la Comisión Electrotécnica Internacional (*International Electrotechnical Commission*) han presentado el estándar ISO/IEC 15504 (1998), el cual se manifiesta a partir del consenso internacional en la definición de un estándar de dominio público para la evaluación de procesos de software.

El desarrollo del estándar ISO/IEC 15504 se establece a partir del proyeto SPICE[5] que tiene como objetivo garantizar una ruta de desarrollo rápida y solicitar las opiniones de los expertos más importantes del mundo (Zahran, 1998).

5.3.2.1 Estructura del ISO/IEC 15504

El estándar ISO/IEC 15504 consiste en dos dimensiones: la dimensión de proceso y la dimensión de capacidad.

[5]Software Process Improvement and Capability dEtermination.

- Dimensión de proceso: se caracteriza por los propósitos de proceso (i.e. objetivos de medición esenciales de un proceso) y el resultado esperado del proceso (i.e. la indicación de su finalización exitosa) (Zahran, 1998). Para esta dimensión son atribuidas cinco categorías:
  - Categoría de proceso cliente-proveedor.
  - Categoría de proceso de ingeniería.
  - Categoría de proceso de soporte.
  - Categoría de proceso de gestión.
  - Categoría de proceso de organización.
- Dimensión de capacidad: Esta dimensión consiste en un conjunto de atributos interrelacionados que ofrecen los indicadores de medición necesaria para gestionar un proceso y mejorar la capacidad de desempeñar un proceso. Esta dimensión compuesta de seis niveles tiene características evolutivas similares a las del CMM (véase Figura 5.6):
  - Nivel 0 - Incompleto: se caracteriza por un incumplimiento general para lograr el propósito del proceso;
  - Nivel 1 - Proceso desempeñado: se caracteriza por el logro de manera general del propósito del proceso;
  - Nivel 2 - Proceso gestionado: se caracteriza por la identificación de la calidad aceptable con definición de tiempos y recursos. Los productos del trabajo están de acuerdo con los estándares especificados y los requerimientos;

- Nivel 3 - Proceso consolidado: se caracteriza por la gestión y el desempeño del proceso usando el proceso estándar basado en principios estables de ingeniería de software;
- Nivel 4 - Proceso predecible: se caracteriza por la consistencia de su desempeño en la práctica con límites de control definidos para alcanzar sus objetivos de proceso definido;
- Nivel 5 - Proceso optimizado: se caracteriza por la optimización del desempeño del proceso para encontrar las necesidades de negocio actuales y futuras, y por el grado de repetición que el proceso alcanza encontrando sus objetivos de negocio definidos.

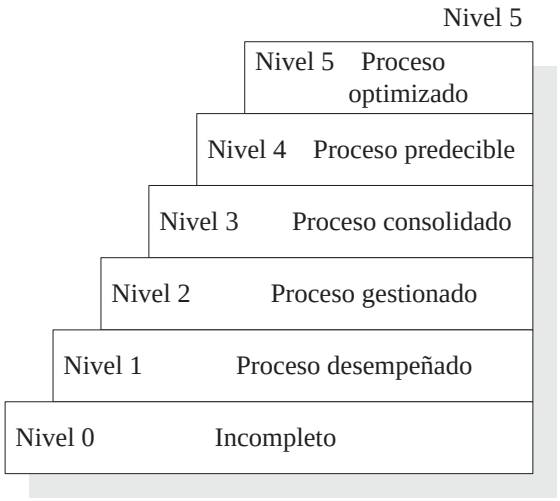

Figura 5.6: Los niveles de capacidad de proceso según ISO/IEC 15504. Fuente: Zahran (1998)

Como se ha comenta, el ISO/IEC 15504 (1998) es un estándar internacional para la evaluación de proceso de software que ofrece beneficios a los desarrolladores, proveedores y clientes (i.e. las organizaciones que adquieren

el software). En este sentido, se observa la necesidad de diseñar un modelo de test para aplicaciones multimedia que también ofrezca beneficios similares al ámbito educativo.

5.3.3 Bootstrap

El *Bootstrap* es una metodología de evaluación que mejora la capacidad de los procesos de desarrollo de software. Además, el modelo Bootstrap describe el proceso de evaluación, determina donde se encuentra una organización respecto a los niveles de madurez, identifica los puntos fuertes y debilidades de la organización, y ofrece un guía para el proceso de mejora (Kuvaja, Bicego, Haase, Koch, Krzanik, Cacchia, Lancelotti, Maiocchi, Messnarz, Saukkonen, Schynoll y Similä, 1993).

Este método es uno de los resultados del proyecto ESPRIT 5441 apoyado por la Comunidad Europea. El Bootstrap ha considerado como punto de partida varios estándares y metodologías como por ejemplo CMM, ISO/IEC 15504 y los estándares de ingeniería de software (*Software Engineering Standards* - SES) de la Agencia Espacial Europea (*European Space Agency* - ESA). El modelo Bootstrap se basa en la tríada Organización, Metodología y Tecnología (OMT) presentada en la Figura 5.7.

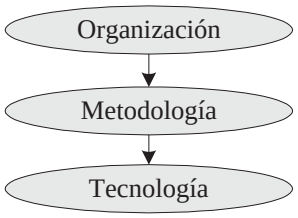

Figura 5.7: La tríada Bootstrap. Fuente: Zahran (1998)

5.3.3.1 Arquitectura del modelo de proceso Bootstrap

Usando la tríada presentada en la Figura 5.7 como punto de partida, se define una arquitectura en forma de árbol que identifica, según Zahran (1998), las categorías de proceso, las áreas de proceso, los procesos y las mejores prácticas.

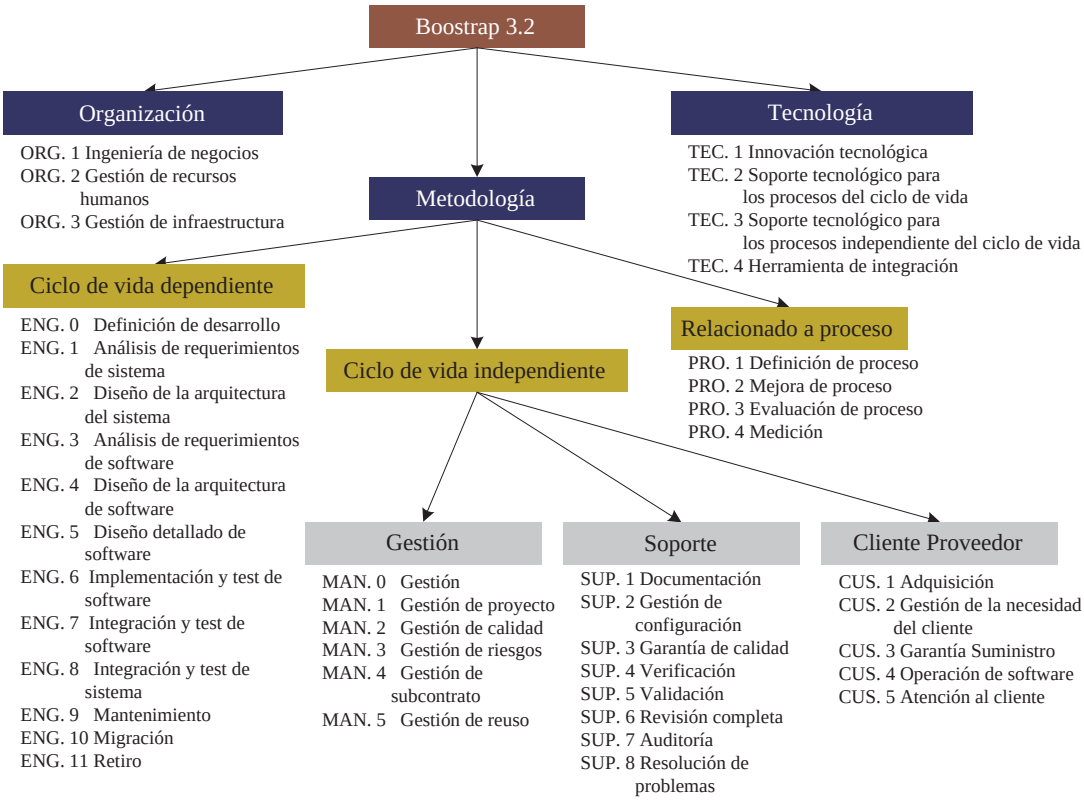

Figura 5.8: Arquitectura de proceso Bootstrap versión 3.2. Fuente: http://www.bootstrap-institute.com

En general, se utiliza el proceso de evaluación del método Bootstrap para medir el estado actual de la práctica de desarrollo de software dentro de una organización.

La evaluación se basa en un cuestionario que está formado por listas de

verificación (*checklists*) de acuerdo con unos atributos clave. De esta manera, se calcula la media agregada por atributos clave y consecuentemente el nivel de madurez, basándose en los cinco niveles de madurez del CMM.

El Bootstrap cubre la unidad de producción de software enfocando sus actividades no sólo en la evaluación sino también en el planeamiento de acción.

5.4 Proceso de test

Como se ha comentado, los procesos de test se realizan tanto dentro del ámbito de la ingeniería de software (e.g. test de verificación y validación), como dentro del ámbito de la ergonomía (e.g. test de usabilidad). En este sentido, es necesario realizar un estudio sobre la formación del proceso de test en la ingeniería de software.

Kit (1995, p. 3) define el proceso de test de software como

> "*los medios a través de los cuales se integran las personas, los métodos, las mediciones, las herramientas y los equipos con el objetivo de evaluar un producto de software*".

A partir de de esta definición, el autor presenta seis elementos esenciales para el proceso de test de software:

1. La calidad del proceso de test (y, consecuentemente, la calidad del software producido) determina el éxito del esfuerzo de test.

2. La prevención de la migración de defectos a través de la aplicación de

técnicas de test en las fases iniciales del ciclo de vida de desarrollo de software.

3. El uso inmediato de herramientas de test de software.

4. Una persona debe responsabilizarse de la mejora de procesos de test.

5. La realización de test (*testing*) es una disciplina profesional que requiere personas entrenadas y calificadas.

6. Cultivar una actitud positiva de eliminación de defectos en el equipo de test.

Dentro de este contexto, se observan avances continuados en la búsqueda de la calidad de software desde las perspectivas de la gestión (e.g. el estudio de Birk y Rombach (2001) sobre la gestión de calidad de software orientado a procesos), de la realización de test (e.g. el análisis de requerimientos funcionales y no-funcionales de procesos de test por Keese (2001)) y de la construcción de heramientas para la realización de test (e.g. Bromnick (2001)).

Usando estas consideraciones com punto de partida, se presentan, a continuación, las actividades que permiten tanto verificar y validar los software, como evaluar su aceptabilidad y la satisfacción del usuario.

5.4.1 Test de verificación, validación y usabilidad

El test de verificación es una de las fases del ciclo de vida de los procedimientos de test. Según Kit (1995) el principal objetivo es detectar la mayoría

de errores posibles. En este sentido, Schulmeyer y Mackenzie (2000) comentan que la verificación permite garantizar la consistencia entre los productos desarrollados (i.e. los prototipos y la versión final del software) y sus requerimientos predeterminados en la fase de especificación, es decir, se necesita saber si se está desarrollando de manera correcta el producto.

Por otra parte, según Schulmeyer y Mackenzie (2000) el test de validación permite garantizar que el software final satisfaga los requerimientos del sistema, es decir, se necesita saber si se desarrolla el producto correcto.

Finalmente están los test de usabilidad que son un procedimiento de análisis que considera criterios de evaluación previamente seleccionados (véase Capítulo 4, página 102). En este tipo de test se verifica el grado de eficiencia y eficacia con que un usuario desempeña sus tareas bajo los requerimientos y restricciones de un producto o servicio y su entorno, evaluando la aceptabilidad y satisfacción del usuario.

Para una mejor compresión de los tipos de test de verificación, validación y usabilidad, se presenta en la Figura 5.9 el modelo "U en punto" (*Dotted-U Model*) propuesto por *Software Development Technologies* (SDT)[6]. Según Kit (1995), este modelo integra el ciclo de vida de desarrollo y el ciclo de test.

[6]http://www.sdtcorp.com/trntest.htm

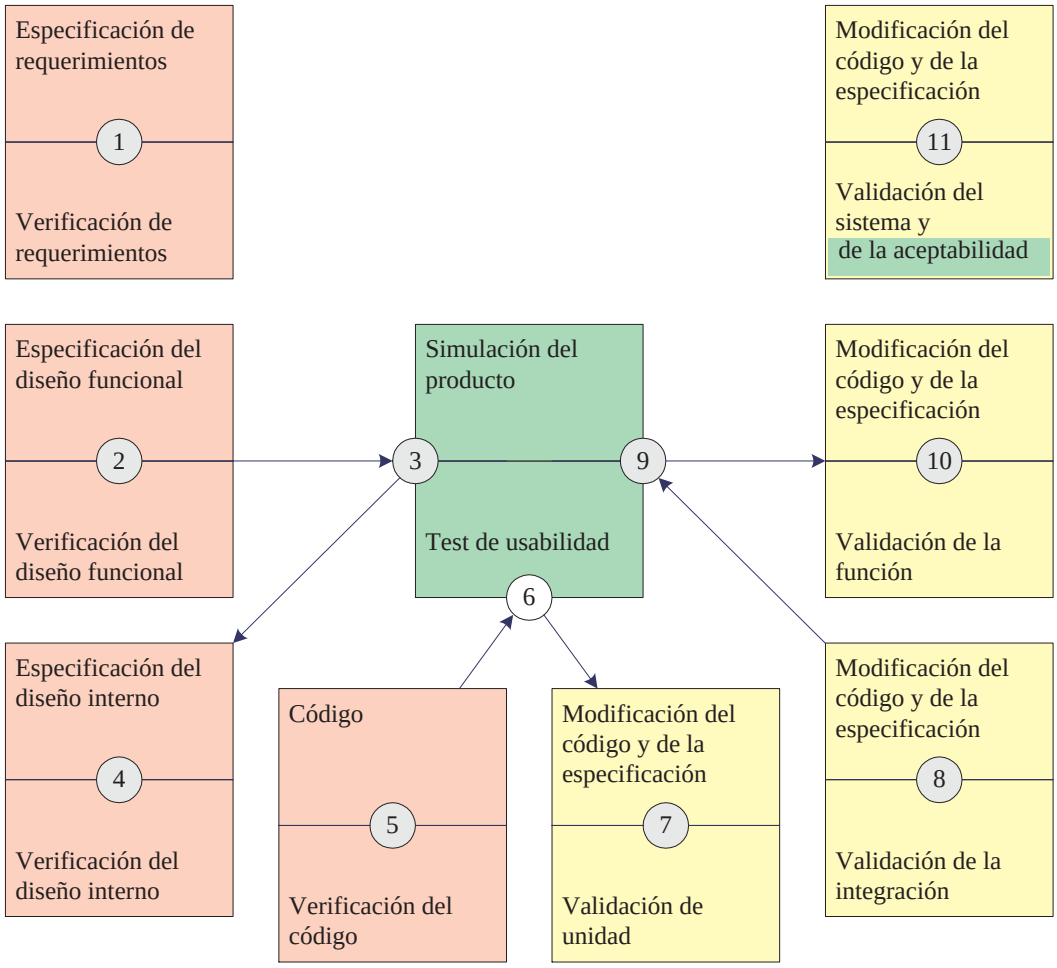

Figura 5.9: Adaptación del Modelo U en punto SDT. Fuente: http://www.sdtcorp.com/trntest.htm y Kit (1995)

En la Figura 5.9, se puede examinar el alcance de los tres métodos de test:

1. La verificación del software se caracteriza por las etapas representadas en rojo (i.e. la verificación de requerimientos, de diseño y de código).

2. La validación consiste en las etapas amarillas (i.e. se validan el software o sus componentes respecto a sus funciones predefinidas). Aunque que Kit (1995) considere el test de usabilidad una actividad de validación, debido a su importancia para la presente tesis se modifica su enfoque definiéndola como una tercera actividad.

3. Los test de usabilidad, representados por la etapa verde, examinan la aceptabilidad y satisfacción del usuario respecto al software.

5.4.2 Importancia de los test

La necesidad de garantizar la alta calidad de los sistemas informáticos (i.e. software) ha aumentado las horas de trabajo de los procedimientos de test respecto a los de análisis, diseño y programación. En los últimos años, diversos autores han publicado documentos en los cuales se constata la preocupación en transformar los procedimientos de test (i.e. un proceso dentro del ciclo de vida de desarrollo de software) en estudios más profundizados.

Recientemente, Shepard, Lamb y Kelly (2001) han comentado sobre la importancia de las técnicas de verificación y validación de software como parte de una disciplina importante de ingeniería de software, las cuales tienen el objetivo de formar desarrolladores de software más efectivos. Los autores

apuntan la dificultad de la inserción de dichas técnicas en los currículos de ingeniería de software, aunque el panorama ha cambiado en los últimos años. Por ejemplo, Adamsone, Borzovs, Gills, Linde y Plume (2000) comentan que el curriculum de cursos en informática en Latvia hace más de una década que consideran asignaturas sobre test de software, presentando con el mismo grado de importancia las actividades de test y las de análisis, diseño y programación.

No obstante y aunque parezca contradictorio, el ideal sería que el tiempo dedicado a las actividades de verificación y validación en la industria del software se redujera como resultado de mejores prácticas de desarrollo de software (Shepard, Lamb y Kelly, 2001).

5.4.3 Proceso de test en los modelos estándares

El estándar ISO/IEC 15504 define un esquema integrado para el desarrollo y aplicación de test a lo largo del ciclo de vida del software, así como el modelo Bootstrap. Por otra parte, estos modelos aportan referencia técnica a los test y cómo éstos dan soporte a los demás procesos.

Además el estándar ISO/IEC 15504 y el modelo Bootstrap, bajo un punto de vista operacional, mantienen un entendimiento común con los clientes, ya que el producto o el proceso está de acuerdo con sus requerimientos. Sin embargo, no se identifica una especialización sistemática de los procesos de test (i.e. los test de verificación, validación y usabilidad).

En el caso del CMM, no se contempla adecuadamente los aspectos relacionados con los test en las áreas de proceso clave (*key process areas*). Tam-

poco se identifican las prácticas de test como una herramienta o mecanismo de mejora del proceso. De esta manera, en el CMM no han sido identificados algunos conceptos, como por ejemplo el de madurez de los test.

5.5 Modelos específicos de proceso de test

De acuerdo con Gelperin y Hetzel (1988), la idea propuesta por diversos investigadores y profesionales de cómo transformar la tarea de test en un proceso de costes efectivos, ha crecido equitativamente respecto a la importancia de dicha tarea.

La evolución de los modelos específicos de proceso de test aumenta satisfactoriamente la calidad de dichos procesos y consecuentemente, los coloca en una posición relevante dentro de la ingeniería de software.

Como se ha comentado, los modelos y estándares de evaluación de proceso de software ofrecen herramientas para establecer niveles de madurez de desarrollo y mantenimiento del software. Sin embargo, ha sido constatado que estos modelos y estándares no dirigen adecuadamente sus enfoques a los procesos de test. En este sentido, se presentan, a continuación, dos modelos de evaluación y mejora de procesos de test:

- Modelo de madurez del test (*Testing Maturity Model* - TMM)
- Modelo de mejora del proceso de test (*Test Process Improvement* - TPI)

5.5.1 Modelo de madurez del test - TMM

Considerando que el test es un componente crítico en el proceso de desarrollo de software, Burnstein, Suwannasart y Carlson (1996a) y Burnstein, Suwannasart y Carlson (1996b) han propuesto el Modelo de madurez del test (*Testing Maturity Model* - TMM) cuyo objetivo es conducir las evaluaciones y mejoras de los procesos de test en las organizaciones que desarrollan software.

El modelo TMM es un modelo de evaluación de proceso de test, guiado por el conjunto básico de conceptos del CMM y compuesto por dos componentes princinpales: un conjunto de niveles de madurez y una modelo de evaluación (Burnstein, Homyen, Grom y Carlson, 1998). Así pues, el TMM prescribe la posición que un determinado proceso de test ocupa en la jerarquía de madurez del test (véase Figura 5.10).

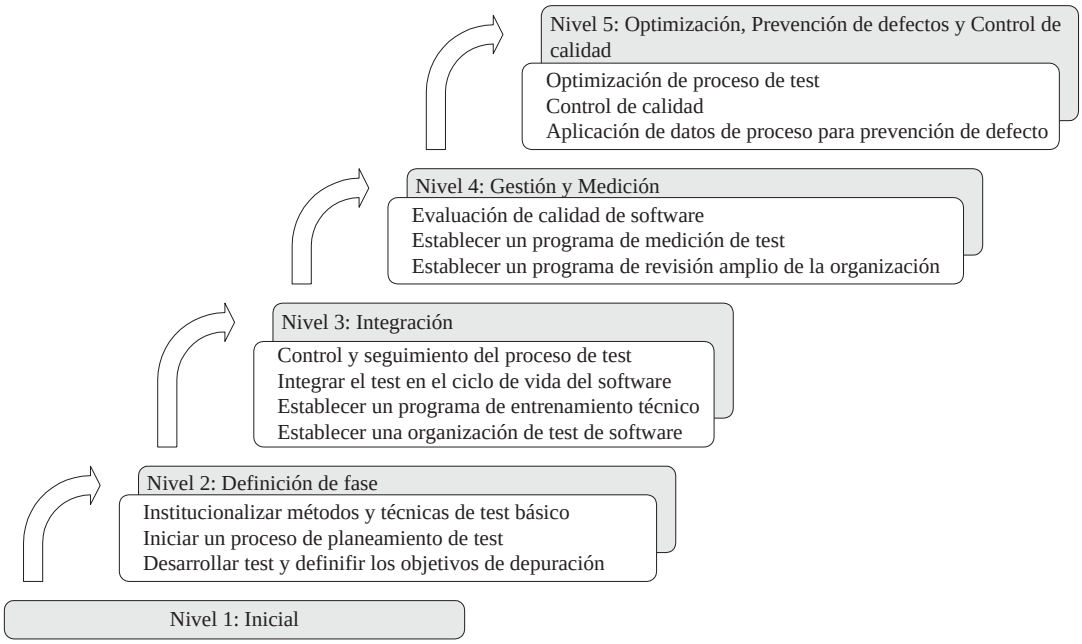

Figura 5.10: Los niveles de madurez del TMM y sus objetivos. Fuente: Burnstein, Suwannasart y Carlson (1996a, 1996b)

Los niveles de madurez del proceso de test son:

- Nivel 1 - Inicial: El test se caracteriza como un proceso caótico. El objetivo del test es mostrar que el software funciona y existe carencia de recursos, herramientas y de un equipo apropiadamente entrenado.
- Nivel 2 - Definición de fase: Se caracteriza por la utilización de métodos y técnicas básicas de test que identificarán si el software está de acuerdo con sus especificaciones.
- Nivel 3 - Integración: Se caracteriza por la identificación de test en todo el ciclo de vida del software. El test es una actividad profesional que realiza un papel de revisión importante (formal o informal) en el control de calidad.
- Nivel 4 - Gestión y Medición: El test es un proceso medido y cuantificado. La usabilidad es uno de los atributos de calidad utilizado en el test del software.
- Nivel 5 - Optimización, Prevención de defectos y Control de calidad: Se caracteriza por mecanismos precisos para que el test pueda ser mejorado continuamente. En este nivel se utilizan procedimientos para seleccionar y evaluar herramientas de test.

Se describe cada nivel considerando los objetivos de la organización y la capacidad de test. Además, la estructura de cada nivel (excepto el nivel uno) consiste en un conjunto de objetivos de madurez soportado por sub-objetivos (o metas) de madurez que son alcanzados por las actividades, tareas y responsabilidades (ATR). Por una parte, las actividades y tareas representan

las acciones que deben ser desempeñadas en cada nivel con el propósito de mejorar la capacidad del test y, por otra, se asignan responsabilidades, para las actividades y tareas, a los gestores, a los desarrolladores y "testeadores" (o revisores) y a los usuarios y clientes (Burnstein, Homyen, Grom y Carlson, 1998). En la Figura 5.11, se presenta la estructura de los niveles de madurez del modelo TMM.

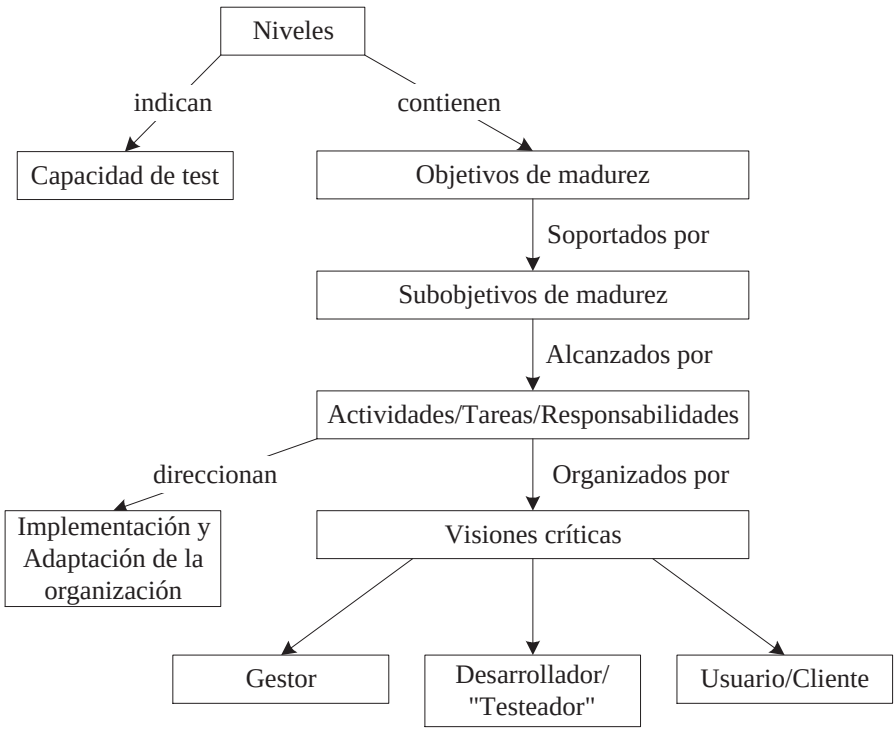

Figura 5.11: La estructura del modelo de madurez del proceso de test. Fuente: Burnstein, Suwannasart y Carlson (1996a, 1996b)

De acuerdo con Burnstein, Homyen, Grom y Carlson (1998), el segundo componente, el modelo de evaluación, consiste en un método que permite a la organización evaluar y mejorar sus procesos de test, basándose en un cuestionario de evaluación, un procedimiento de evaluación y el entrenamiento y

criterios de selección del equipo de evaluación.

Los objetivos de este componente son: proveer un esquema de evaluación de proceso de test de software y un fundamento para la mejora de dichos procesos a través de análisis de datos y planeamiento de acciones, así como contribuir con la ingeniería de proceso de software.

5.5.2 Mejora del proceso de test - TPI

Koomen y Pol (1999) presentan el modelo de mejora del proceso de test (*Test Process Improvement* - TPI), en cual ofrece soporte a la mejora del proceso de test a través de guías prácticas que serán utilizadas para evaluar el nivel de madurez de los test de una organización. Además, este esquema determina los puntos fuertes y débiles de un proceso de test, de manera que ayuda a definir pasos controlados y graduales de mejora.

El modelo TPI se basa en la metodología de test estructurado TMap (Pol y Veenendaal, 1998), a partir de diversas recomendaciones de estructuración de proceso de test organizadas en cuatro componentes: el ciclo de vida (C), las técnicas (T), la infraestructura (I) y la organización (O)(véase Figura 5.12).

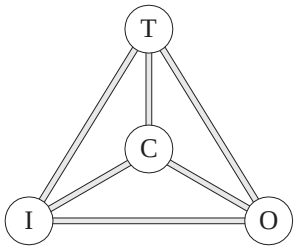

Figura 5.12: Los componentes de soporte para enfoques estructurados de test. Fuente: Pol y Veenendaal (1998)

El modelo TPI, presentado en la Figura 5.13, consiste en cinco componentes:

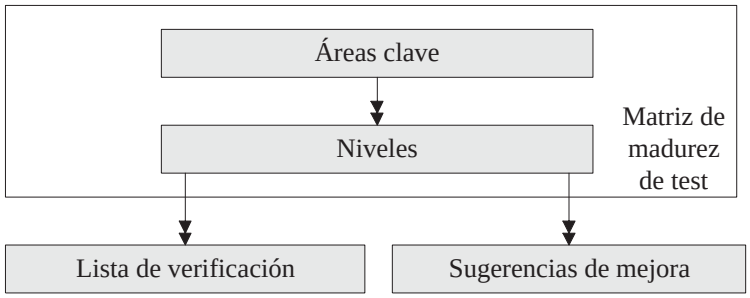

Figura 5.13: El esquema general del modelo TPI. Fuente: Koomen and Pol (1999)

- Las áreas clave: Constituyen la base para mejorar y estructurar el proceso de test. Las áreas clave comprenden tanto los test de sistema y de aceptación (i.e. test de alto nivel) como los test de unidad y de integración (i.e. test de bajo nivel). El modelo TPI tiene viente áreas clave:
  - Estrategias de test
  - Modelo de ciclo de vida
  - Momento de compromiso
  - Estimación y planeamiento
  - Técnicas de especificación de test
  - Técnicas de test estático
  - Métrica
  - Herramientas de test
  - Entorno de test
  - Entorno de oficina
  - Compromiso y motivación
  - Funciones de test y entrenamiento
  - Alcance de la metodología
  - Comunicación
  - Producción de informes
  - Gestión de defectos

  - Gestión de elementos de test
  - Evaluación
  - Gestión del proceso de test
  - Test de bajo nivel

- Los niveles de madurez: La organización de las áreas clave dentro de un proceso de test determina la madurez de dicho proceso (Koomen y Pol, 1999). Por consiguiente, los niveles de madurez indican el estados (e.g. A, B, C, etc.) de cada área clave, debido a que ellos conllevan el cumplimiento de determinados requerimientos para las áreas clave. Como promedio, existen tres niveles para cada área.

- La lista de verificación: Es un instrumento de medición objetivo, basado en preguntas, que permite determinar el nivel de madurez de cada área clave.

- La matriz de madurez del test: Representa un mapa de interrelaciones entre las áreas clave y sus niveles de madurez en una escala de 1 a 13 distribuida en tres categorías:

  - Controlada (1 al 5).
  - Eficiente (6 al 10).
  - Optimizada (11 al 13)

- Las sugerencias de mejora: Los resultados de los análisis de la matriz de madurez de test proponen acciones de mejora en función de los criterios que se deseen alcanzar.

El proceso de evaluación del modelo TPI se basa en la utilización de encuestas y documentación para determinar el estado de cada proceso de

test. En el procedimiento se consideran las características de cada área clave en los diferentes niveles, de manera que, a través del uso de los puntos de verificación (*checkpoints*), sea posible determinar los puntos fuertes y débiles del proceso de test, permitiendo establecer una posición relativa en la matriz de madurez del test (*Test Maturity Matrix*). Además, las sugerencias de mejora son una herramienta que ayuda a alcanzar un determinado nivel de madurez, ya que establece comentarios e sugerencias para el proceso de test (Koomen y Pol, 1999).

De acuerdo con Koomen y Pol (1999) el modelo TPI ofrece el soporte necesario para mejorar el proceso de test a través de la obtención de rápida de información acerca de la situación actual de dichos procesos. De esta manera, se considera el modelo como una herramienta de estructuración de acciones de mejora del proceso de test.

Como se ha visto en este capítulo, la ingeniería de software es una disciplina que involucra diversas metodologías, métodos, técnicas y herramientas con el propósito de desarrollar sistemas informáticos de alta calidad.

Considerando el contexto de la presente investigación (i.e. la interrelación entre la usabilidad del software educativo y el aprendizaje del usuario), se han presentado los conceptos más importantes sobre la ingeniería de software y sobre los modelos y estándares internacionales que auxilian en las actividades de evaluación de proceso de software y de test. Por lo tanto, el estudio teórico presentado ha guiado la preparación del modelo propuesto (véase Capítulo 7).