CAPÍTULO 4
CONSTRUCCIÓN DEL SOFTWARE

ACRÓNIMOS

OMG	Grupo de Gestión de Objetos
UML	Lenguaje Unificado de Modelado

INTRODUCCIÓN

El término construcción del software hace referencia a la creación detallada de software operativo y significativo, por medio de una combinación de codificación, verificación, pruebas unitarias, pruebas de integración y depuración.

El Área de Conocimiento de la Construcción del Software está vinculada a todas las otras KAs (Áreas de Conocimiento), más fuertemente al Diseño del Software y a las Pruebas del Software. Esto se debe a que el proceso mismo de construcción del software cubre tanto el diseño significativo de software como las actividades de pruebas. También utiliza las salidas del diseño y proporciona una de las entradas para las pruebas, consistiendo estas actividades en el diseño y en las pruebas, y en este caso no en las KAs. Las fronteras detalladas entre el diseño, la construcción y las pruebas (si es que existen) varían dependiendo de los procesos de ciclo de vida del software utilizados en un proyecto.

A pesar de que se pueda realizar parte del diseño detallado antes de la construcción, mucho del trabajo del diseño se lleva a cabo durante la actividad misma de la construcción. Por lo que el KA de Construcción del Software está vinculado muy de cerca al KA de Diseño del Software.

Por medio de la construcción los ingenieros del software realizan tanto pruebas unitarias, como pruebas de integración de su trabajo. De tal manera que el KA de Construcción del Software está también vinculada de cerca al KA de Pruebas del Software.

La construcción del software, por lo general, produce el mayor número de elementos de configuración que se necesitan gestionar en un proyecto de software (archivos de código fuente, contenido, casos de pruebas, etc). De este modo, el KA de Construcción del Software también está vinculado de cerca al KA de Gestión de la Configuración del Software.

Dado que la construcción del software tiene una gran dependencia de las herramientas y de los métodos, y de que se trata probablemente del KA que más herramientas tiene y utiliza, está vinculada al KA de Herramientas y Métodos de la Ingeniería del Software.

Aunque la calidad del software es importante en todas las KAs, el código es el último entregable de un proyecto de software y, por tanto, la Calidad del Software está vinculada de cerca a la Construcción del Software.

Entre las Disciplinas Descritas de la Ingeniería del Software el KA de Construcción del Software es lo más parecido a la ciencia informática en su uso del conocimiento de algoritmos y de las prácticas detalladas de codificación, ambas son consideradas, con frecuencia, como pertenecientes al dominio de la ciencia informática. También está relacionada con la gestión del proyecto en la medida en que la gestión de la construcción pueda presentar retos considerables.

DESCOMPOSICIÓN DE LOS TEMAS DE CONSTRUCCIÓN DEL SOFTWARE

A continuación se presenta la descomposición del KA de la Construcción del Software junto con breves descripciones de los temas más importantes asociados a este. La figura 1 ofrece una representación gráfica de la descomposición de alto nivel de las divisiones de este KA.

1. Fundamentos de la Construcción del Software

Los fundamentos de la construcción del software incluyen:

- Minimizar la complejidad
- Anticiparse a los cambios
- Construir para verificar
- Estándares en la construcción

Los tres primeros conceptos se aplican tanto al diseño como a la construcción. Las siguientes secciones definen estos conceptos y describen cómo se aplican a la construcción.

1.1 Minimizar la complejidad

[Bec99; Ben00; Hun00; Ker99; Mag93; McC04]

El principal factor que hace que la gente utilice ordenadores consiste en la limitadísima capacidad que tiene para retener estructuras complejas e información en su memoria operativa, especialmente durante largos períodos de tiempo. Esto lleva a uno de los más fuertes impulsores de la construcción del software: minimizar la complejidad. La necesidad de reducir la complejidad se aplica esencialmente a todo aspecto de la construcción del software, y es de

crítica importancia para el proceso de verificación y pruebas de las construcciones del software.

En la construcción del software sólo se alcanza una reducida complejidad por medio del énfasis en la creación de código que sea simple y legible, y no tanto inteligente.

Se logra minimizar la complejidad mediante el uso de estándares, como se ve en el apartado 1.4 *Estándares de Construcción*, y mediante numerosas técnicas específicas que están resumidas en el apartado 3.3 *Codificación.* También se apoya en las técnicas de calidad enfocadas a la construcción resumidas en el apartado 3.5 Calidad de la Construcción.

1.2 Anticiparse a los cambios

[Ben00; Ker99; McC04]

La mayoría del software cambiará a lo largo del tiempo, y el anticiparse a los cambios dirige muchos aspectos de la construcción del software. El software es inevitablemente parte de los ambientes externos que cambian continuamente, y los cambios en esos ambientes externos afectan al software de diversos modos.

El anticiparse a los cambios se apoya en muchas técnicas específicas resumidas en el apartado 3.3 *Codificación.*

1.3 Construir para verificar

[Ben00; Hun00; Ker99; Mag93; McC04]

Construir para verificar significa construir software de tal manera que los ingenieros del software puedan sacar a relucir los fallos con facilidad al estar escribiendo el código, además de cuando realizan pruebas independientes y actividades operacionales. Las técnicas específicas que sirven de base para construir con vistas a verificar incluyen el seguimiento de estándares de codificación que permitan las revisiones del código, las pruebas unitarias, la organización del código que permita pruebas automáticas, y el uso restringido de estructuras de lenguaje que sean complejas o difíciles de entender, entre otras.

1.4 Estándares en la construcción

[IEEE12207-95; McC04]

Los estándares que afectan directamente a elementos de la construcción incluyen:

- Métodos de comunicación (por ejemplo, estándares para los formatos de los documentos y de los contenidos)
- Programación de lenguajes (por ejemplo, estándares de lenguaje para lenguajes como Java y C++)
- Plataformas (por ejemplo, estándares de interfaces del programador para llamadas al sistema operativo)
- Herramientas (por ejemplo, estándares diagramáticos para notaciones como UML (Lenguaje Unificado de Modelado))

Uso de estándares externos. Construir depende del uso de estándares externos para los lenguajes de construcción, las herramientas de construcción, las interfaces técnicas, y las interacciones entre la Construcción del Software y las otras KAs. Los estándares provienen de numerosas fuentes, incluyendo las especificaciones de interfaz del hardware y del software, tales como el Grupo de Gestión de Objetos (OMG) y las organizaciones internacionales tales como la IEEE o ISO.

Uso de estándares internos. Los estándares también pueden crearse partiendo de una base organizacional a un nivel corporativo o para su uso en proyectos específicos. Estos estándares permiten la coordinación de actividades de grupo, el minimizar la complejidad, el anticipar los cambios y el construir para verificar.

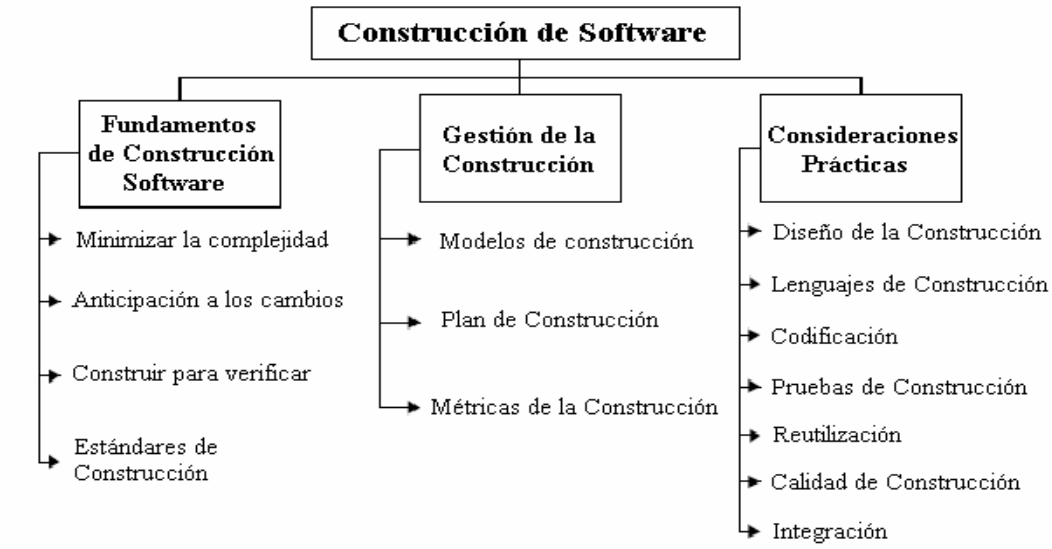

Figura 1 Descomposición de los temas del KA de Construcción de Software

2. Gestión de la Construcción

2.1 Modelos de Construcción [Bec99; McC04]

Se han creado numerosos modelos para el desarrollo del software, algunos de los cuales ponen más énfasis en la construcción que otros.

Algunos modelos son más lineales que otros desde el punto de vista de la construcción tales como los modelos en cascada y los del ciclo de vida de entregas por etapas. Estos modelos tratan la construcción como una actividad que sucede sólo después de que se haya completado un significativo trabajo con los prerrequisitos –incluyendo un trabajo detallado sobre los requisitos, un extensivo trabajo sobre el diseño y una planificación detallada. Los enfoques más lineales tienden a poner el énfasis en las actividades que preceden a la construcción (requisitos y diseño), y tienden a crear separaciones más marcadas entre las actividades. En estos modelos, la codificación sería el punto de mayor énfasis de la construcción.

Otros modelos son más iterativos, tales como el prototipado evolucionista, Programación Extrema y "Scrum". Estos enfoques tienden a tratar la construcción como una actividad que ocurre en estos momentos con otras actividades de desarrollo del software incluyendo los requisitos, el diseño y la planificación, o que se traslapa con ellas. Estos enfoques tienden a mezclar el diseño, la codificación y las actividades de pruebas, y con frecuencia tratan la combinación de actividades como una construcción.

Por consiguiente, lo que está considerado como "construcción" depende hasta cierto grado del modelo de ciclo de vida utilizado.

2.2 Planificación de la Construcción

[Bec99; McC04]

La elección de un método de construcción es un aspecto clave de la planificación de la actividad de construcción. La elección de un método de construcción afecta hasta dónde se realizan los prerrequisitos de construcción, el orden en el que se realizan, y el grado hasta el que se espera que se completen antes de que comience el trabajo de construcción.

El modo como se afronta la construcción afecta a la habilidad del proyecto para reducir la complejidad, anticipar cambios y construir para verificar. Cada uno de estos objetivos puede también afrontarse en los niveles de proceso, requisitos y diseño –pero también estarán influenciados por la elección de un método de construcción.

La planificación de la construcción también define el orden en el que se crean e integran, según el método elegido, los componentes, los procesos de gestión de la calidad del software, la asignación de tareas a ingenieros del software específicos y el resto de las tareas.

2.3 Medición de la Construcción

[McC04]

Se pueden medir numerosas actividades de construcción y artefactos, incluidos el código desarrollado, el código modificado, el código reutilizado, el código destruido, la complejidad del código, las estadísticas de la inspección del código, las tasas de rectificación de errores y de identificación de errores, y los horarios. Estas mediciones pueden ser útiles para propósitos de gestión de la construcción, asegurando la calidad durante la construcción, mejorando los procesos de construcción, amén de otras razones.

3. Consideraciones Prácticas

La construcción es una actividad en la cual el software se las tiene que ver con restricciones arbitrarias y caóticas del mundo real, y hacer exactamente lo que piden. Gracias a su proximidad a las restricciones del mundo real, la construcción está guiada por consideraciones prácticas más que otras KAs, y la ingeniería del software es quizás el área de construcción más artesanal.

3.1 Diseño de la Construcción

[Bec99; Ben00; Hun00; IEEE12207-95; Mag93; McC04]

Algunos proyectos asignan una mayor actividad de diseño a la construcción; otros a una fase que se centra explícitamente en el diseño. Independientemente de su asignación exacta, en el nivel de construcción también se trabaja algo el diseño detallado y ese trabajo de diseño tiende a estar dictaminado por restricciones inamovibles impuestas por un problema del mundo real que está siendo afrontado por el software.

Así como los obreros de una construcción que construyen una estructura física tienen que realizar modificaciones a pequeña escala para cubrir huecos no previstos en los planes del constructor, los obreros de la construcción del software tendrán que hacer modificaciones en una mayor o menor escala para revelar los detalles del diseño de software durante la construcción.

Los detalles de la actividad de diseño a nivel de la construcción son esencialmente los mismos que se describen en el KA del Diseño del Software, pero se aplican en una escala inferior.

3.2 Lenguajes de Construcción

[Hun00; McC04]

Los lenguajes de construcción incluyen todos los tipos de comunicación mediante los cuales un humano puede especificar una solución ejecutable para un problema de un ordenador.

El tipo más simple de lenguaje de construcción es un *lenguaje de configuración* en el que los ingenieros del software eligen de entre un conjunto limitado de opciones predefinidas para crear nuevas o típicas instalaciones del software. Los archivos de configuración basados en texto utilizados tanto en los sistemas operativos de Windows como de Unix son ejemplos de esto, y otro ejemplo son las listas de selección en forma de menú de algunos generadores de programas.

Los *lenguajes de herramientas* se utilizan para construir aplicaciones partiendo de las herramientas (conjuntos integrados de partes reutilizables específicas de las aplicaciones), y son más complejos que los lenguajes de configuración. Los lenguajes de herramientas pueden definirse explícitamente como lenguajes de programación de aplicaciones (por ejemplo, scripts), o pueden simplemente estar implícitos en el conjunto de interfaces de las herramientas.

Los *lenguajes de programación* son el tipo más flexible de lenguaje de construcción. También son los que menos información contienen acerca de las áreas específicas de la aplicación y los procesos de desarrollo, y por tanto requieren el mayor entrenamiento y destreza posibles para utilizarlo con eficacia.

Existen tres tipos generales de notación utilizados para los lenguajes de programación, a saber:

- Lingüísticos
- Formales
- Visuales

Las notaciones lingüísticas se distinguen en particular por la utilización de cadenas de texto del tipo palabra para representar construcciones complejas de software, y por la combinación en patrones de tales cadenas del tipo palabra que tienen una sintaxis del tipo sentencia. Utilizadas adecuadamente, cada una de estas cadenas debería tener una fuerte connotación ofreciendo un entendimiento intuitivo inmediato de lo que sucedería cuando se ejecutara la construcción del software subyacente.

Las notaciones formales se apoyan menos en los significados de las palabras y cadenas de texto intuitivos y de todos los días, y más en las definiciones respaldadas por definiciones precisas, sin ambigüedad, y formales (o matemáticas). Las notaciones de construcción formal y los métodos formales están en el corazón de la mayoría de las formas de programación de sistemas, donde la precisión, el comportamiento del tiempo, y la capacidad de realizar pruebas son más importantes que la facilidad de mapeo a un lenguaje natural. Las construcciones formales también utilizan modos de combinar símbolos definidos con precisión que evitan la ambigüedad de muchas construcciones del lenguaje natural.

Las notaciones visuales se apoyan bastante poco en las notaciones orientadas al texto tanto de la construcción lingüística como de la formal, y en cambio sí se apoyan en una interpretación visual directa y en la colocación de las entidades visuales que representan al software subyacente. La construcción visual tiende a estar un tanto limitada por la dificultad de hacer declaraciones "complejas" utilizando sólo el movimiento de entidades visuales en un despliegue. Sin embargo, también puede convertirse en un arma poderosa en los casos en donde la principal tarea de programación es simplemente construir y "ajustar" una interfaz visual a un programa, cuyo comportamiento detallado ha sido definido anteriormente.

3.3 Codificación

[Ben00; IEEE12207-95; McC04]

Las consideraciones siguientes se aplican a la actividad de construcción del código del software:

- Técnicas para crear código fuente comprensible, que incluye la asignación de nombres y el esquema del código fuente
- Utilización de clases, tipos enumerados, variables, constantes predefinidas, y otras entidades similares
- Utilización de estructuras de control
- Tratamiento de las condiciones de error –tanto lo errores planeados como las excepciones (la entrada de datos malos, por ejemplo)
- Prevención de brechas en la seguridad a nivel de código (el búfer o el índice de la matriz se desborda, por ejemplo)
- Utilización de recursos por medio del uso de mecanismos de exclusión y disciplina en el acceso serial a recursos reutilizables (incluyendo threads o bloqueos de bases de datos)
- Organización del código fuente (en declaraciones, rutinas, clases, paquetes u otras estructuras)
- Documentación del código
- Puesta a punto del código

3.4 Pruebas de Construcción

[Bec99; Hun00; Mag93; McC04]

Construir implica dos tipos de pruebas, que por lo general las realiza el mismo ingeniero del software que escribió el código:

- Pruebas unitarias
- Pruebas de integración

El propósito de las pruebas de construcción es reducir la brecha entre el tiempo en el que se introducen fallos en el código y el tiempo en el que se detectan esos fallos. En algunos casos, las pruebas de construcción se llevan a cabo después de la escritura del código. En otros casos, se pueden elaborar casos de pruebas antes de que se escriba el código.

Es típico de las pruebas de construcción el incluir un subconjunto de tipos de pruebas, que se describen en el KA de Pruebas del Software. Por ejemplo, no es típico de las pruebas de construcción el incluir las pruebas del sistema, las pruebas alfa, las pruebas beta, las pruebas de estrés, las pruebas de construcción, las pruebas de posibilidad de uso, u otros tipos de pruebas más especializadas.

Se han publicado dos estándares sobre dicho tema: IEEE Std 829-1998, *IEEE Standard for Software Test Documentation* and IEEE Std 1008-1987, *IEEE Standard for Software Unit Testing.*

Se pueden ver también los sub-temas correspondientes en el KA de Pruebas del Software: 2.1.1 Pruebas Unitarias y 2.1.2 Pruebas de Integración para un material de referencia más especializado.

3.5 Reutilización

[IEEE1517-99; Som05].

Tal y como se afirma en la introducción del (IEEE1517-99):

"El implementar la utilización del software conlleva algo más que crear y utilizar librerías de recursos. Requiere formalizar la práctica de la reutilización por medio de la integración de procesos y actividades de reutilización en el ciclo de vida del software." Sin embargo, la reutilización tiene suficiente importancia en la construcción del software como para dedicarle aquí un tema.

Las tareas relacionadas con la reutilización en la construcción del software durante su codificación y pruebas son:

- La selección de unidades, bases de datos, procedimientos de pruebas o datos de pruebas reutilizables.
- La evaluación de la posibilidad de reutilización del código o de las pruebas.
- Comunicar la información sobre reutilización realizada en el código nuevo, los procedimientos de pruebas o los datos de pruebas.

3.6 Calidad de la Construcción

[Bec99; Hun00; IEEE12207-95; Mag93; McC04]

Existen numerosas técnicas para garantizar la calidad del código mientras está siendo elaborado. Las técnicas más importantes utilizadas para la construcción incluyen:

- Las pruebas unitarias y las pruebas de integración (tal y como se describen en el punto 3.4 *Pruebas de Construcción*)
- El desarrollo de primero-haz-pruebas (ver también el KA de las Pruebas del Software, punto 2.2 *Objetivos de las Pruebas*)
- El código paso a paso
- Utilización de aserciones
- Depuración
- Revisiones Técnicas (ver también el KA de la Calidad del Software, sub-punto 2.3.2 *Revisiones Técnicas*)
- Análisis estático (IEEE1028) (ver también el KA de la Calidad del Software, punto 2.3 *Revisiones y Auditorias*)

La técnica o técnicas específicas elegidas dependen de la naturaleza del software que se está construyendo, así como del conjunto de habilidades de los ingenieros del software que llevan a cabo la construcción.

Las actividades de calidad de la construcción se distinguen de las otras actividades de calidad por su enfoque. Las actividades de calidad de la construcción se centran en el código y en los artefactos que están estrechamente relacionados con el código: diseños en pequeña escala –en oposición a otros artefactos que están menos directamente ligados al código, tales como requisitos, diseños de alto nivel y planes.

3.7 Integración

[Bec99; IEEE12207-95; McC04]

Una actividad clave durante la construcción es la integración de rutinas, clases, componentes y subsistemas construidos por separado. Además, un sistema particular del software podría necesitar ser integrado con otros sistemas de software o de hardware.

Los intereses relacionados con la integración de la construcción incluyen planificar la secuencia en la que se integrarán los componentes, crear andamiajes que soporten versiones provisionales del software, determinar el grado de pruebas y la calidad del trabajo realizado sobre los componentes antes de que sean integrados, y determinar los puntos en el proyecto en los que se prueban las versiones provisionales del software.

MATRIZ DE TEMAS VS. MATERIAL DE REFERENCIA

	[Bec99]	[Ben00]	[Hun00]	[IEEE 1517]	[IEEE 12207.0]	[Ker99]	[Mag93]	[Mcc04]	[Som05]
1. Fundamentos de Construcción de Software									
1.1 Minimizar la Complejidad	c17	c2, c3	c7,c8			c2, c3	c6	c2, c3, c7-c9, c24, c27, c28, c31, c32-c34	
1.2 Anticipación a Cambios		c11,c13-c14				c2, c9		c3-c5, c24, c31, c32, c34	
1.3 Construir para verificar		c4	c21, c23, c34, c43			c1, c5, c6	c2,c3, c5,c7	c8, c20-c23, c31-c34	
1.4 Estándares de Construcción					X			c4	
2. Gestión de la Construcción									
2.1 Modelos de Construcción	c10							c2, c3, c27, c29	
2.2 Plan de Construcción	c12, c15, c21							c3, c4,c21, c27-c29	
2.3 Métricas de la construcción								c25, c28	
3. Consideraciones Prácticas									
3.1 Diseño de la Construcción	c17	c18-c10, p175-6	c33		X		c6	c3, c5, c24	
3.2 Lenguajes de Construcción			c12, c14-c20					C4	
3.3 Codificación		c6-c10			X			c5-c19, c25-c26	
3.4 Pruebas de Construcción	c18		c34, c43		X		c4	c22, c23	
3.5 Reusabilidad				X					c14
3.6 Calidad de Construcción	c18		c18		X		c4, c6, c7	c8, c20-c25	
3.7 Integración	c16				X			c29	

REFERENCIAS RECOMENDADAS PARA LA CONSTRUCCIÓN DE SOFTWARE

[Bec99] K. Beck, *Extreme Programming Explained: Embrace Change*, Addison-Wesley, 1999, Chap. 10, 12, 15, 16-18, 21.

[Ben00a] J. Bentley, *Programming Pearls*, second ed., Addison-Wesley, 2000, Chap. 2-4, 6-11, 13, 14, pp. 175-176.

[Hun00] A. Hunt and D. Thomas, *The Pragmatic Programmer*, Addison-Wesley, 2000, Chap. 7, 8 12, 14-21, 23, 33, 34, 36-40, 42, 43.

[IEEE1517-99] IEEE Std 1517-1999, *IEEE Standard for Information Technology-Software Life Cycle Processes-Reuse Processes*, IEEE, 1999.

[IEEE12207.0-96] IEEE/EIA 12207.0-1996//ISO/IEC12207:1995, *Industry Implementation of Int. Std.ISO/IEC 12207:95, Standard for Information Technology-Software Life Cycle Processes*, IEEE, 1996.

[Ker99a] B.W. Kernighan and R. Pike, *The Practice of Programming*, Addison-Wesley, 1999, Chap. 2, 3, 5, 6, 9.

[Mag93] S. Maguire, *Writing Solid Code: Microsoft's Techniques for Developing Bug-Free C Software*, Microsoft Press, 1993, Chap. 2-7.

[McC04] S. McConnell, *Code Complete: A PracticalHandbook of Software Construction*, Microsoft Press, second ed., 2004.

[Som05] I. Sommerville, *Software Engineering*, seventh ed., Addison-Wesley, 2005.

APÉNDICE A. LISTA DE LECTURAS ADICIONALES.

(Bar98) T.T. Barker, *Writing Software Documentation: A Task-Oriented Approach*, Allyn & Bacon, 1998.
(Bec02) K. Beck, *Test-Driven Development: By Example*, Addison-Wesley, 2002.
(Fow99) M. Fowler and al., *Refactoring: Improving the Design of Existing Code*, Addison-Wesley, 1999.
(How02) M. Howard and D.C. Leblanc, *Writing Secure Code*, Microsoft Press, 2002.
(Hum97b) W.S. Humphrey, *Introduction to the Personal Software Process*, Addison-Wesley, 1997.
(Mey97) B. Meyer, *Object-Oriented Software Construction*, second ed., Prentice Hall, 1997, Chap. 6, 10, 11.
(Set96) R. Sethi, *Programming Languages: Concepts & Constructs*, second ed., Addison-Wesley, 1996, Parts II-V.

APÉNDICE B. LISTA DE ESTÁNDARES

(IEEE829-98) IEEE Std 829-1998, IEEE Standard for Software Test Documentation, IEEE, 1998.
(IEEE1008-87) IEEE Std 1008-1987 (R2003), IEEE Standard for Software Unit Testing, IEEE, 1987.
(IEEE1028-97) IEEE Std 1028-1997 (R2002), IEEE Standard for Software Reviews, IEEE, 1997.
(IEEE1517-99) IEEE Std 1517-1999, IEEE Standard for Information Technology-Software Life Cycle Processes-Reuse Processes, IEEE, 1999.
(IEEE12207.0-96) IEEE/EIA 12207.0-1996//ISO/IEC12207:1995, Industry Implementation of Int. Std.ISO/IEC 12207:95, Standard for Information Technology-Software Life Cycle Processes, IEEE, 1996.