



Universidad de las Ciencias  
Informáticas

**Título: Propuesta de arquitectura de software para el producto  
“Mis Mejores Cuentos”.**

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias  
Informáticas

**Autora:**

Lianet Ballester Jiménez

**Tutor:**

Ing. Yasmani Ceballos Izquierdo

**Co-Tutor:**

Ing. José Antonio Soto Pérez

La Habana, junio de 2011

“Año 53 de la Revolución”

# Declaración de autoría

Declaro ser autor de la presente tesis y se reconoce a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los \_\_\_\_\_ días del mes de \_\_\_\_\_ del año 2011.

\_\_\_\_\_


Lianet Ballester Jiménez

Tesista

\_\_\_\_\_

Yasmani Ceballos Izquierdo

Tutor



*“Nunca consideres el estudio como una obligación, sino como una oportunidad para penetrar en el bello y maravilloso mundo del saber”.*

*Albert Einstein*

# Agradecimientos

*A mis padres, las personas que más quiero en el mundo, y los que han hecho de mí lo que soy, a ellos que me han brindado su apoyo, confianza, amor, por estar siempre orgullosos de sus hijas, por educarme y guiarme siempre por el camino correcto. Gracias los quiero mucho.*

*A mi hermana por estar siempre junto a mí, y soportarme, sigue como vas Yuya. Te quiero.*

*A Adrián, por ser el mejor guía en mi carrera, gracias por apoyarme en estos cinco años, por estar siempre junto a mí, por tu paciencia, entrega, amor y dedicación que me han hecho ser mejor persona y a su familia que me acogieron como una más entre ellos.*

*A mi familia que siempre me ha brindado su confianza, por apoyarme en todo momento y confiar ciegamente en mí. Los quiero muchísimo a todos.*

*A mis viejos e inolvidables amigos Dayana, Evelio y Yaneiris por todo este tiempo de amistad, sus buenos consejos y ayuda en todo momento gracias por permitirme entrar en sus corazones. Nunca los olvidaré.*

*A mis amigas de la universidad Lisy (Tita), Anita por compartir conmigo y brindarme su amistad. Gracias.*

*A las chicas y chicos del apartamento, con los que he compartido los últimos 2 años gracias por todo.*

*A compañeros del 8105 por estar siempre dispuestos ayudarme y compartir juntos alegrías y tristezas.*

*A los profes Yasmani, Lisandra y José por su paciencia, entrega y por estar ahí siempre que los necesité.*

*A todas las personas que me quieren y que sin esperar nada de mí de una forma u otra siempre me han ayudado.*

*A la memoria de mis abuelos.*

*A mis padres por ser mi ejemplo e inspiración y quererme tanto.*

*A mi familia por ser el centro de mi existencia, porque sin ellos no hubiera podido cumplir este sueño.*

*A mi Tito por su dedicación y amor todos estos años.*

La informatización de cada esfera de la sociedad es una necesidad que existe hoy en el mundo. Actualmente el sistema de enseñanza preescolar cubano cuenta con una cantidad insuficiente de aplicaciones educativas para impartir conocimientos y lograr un mayor aprendizaje de los niños. Para contribuir al desarrollo del software educativo, en el centro de Tecnologías para la Formación (FORTES) de la Universidad de las Ciencias Informáticas (UCI), se desarrolla el proyecto Producción de Recursos Didácticos, encargado de crear el producto “Mis Mejores Cuentos” con el propósito de digitalizar los cuentos infantiles tradicionales para los niños. Por las características propias del sistema, fue necesario diseñar una arquitectura que respondiera a cualidades como la integrabilidad, la portabilidad, la mantenibilidad, entre otras. Algunas de las contribuciones más importantes para alcanzar estas cualidades arquitectónicas fueron el estudio del estado del arte de la arquitectura de software, la selección de estilos, patrones y tecnologías web, el desarrollo de los artefactos propuestos por la metodología utilizada, y la validación de los principales escenarios arquitectónicos. Como parte de la organización del proceso de desarrollo se construyeron una serie de vistas arquitectónicas, que ayudaron a la comprensión del sistema. El análisis de los resultados obtenidos corroboró que el sistema cumple en gran medida con los objetivos propuestos y responde a las restricciones impuestas. Como resultado del trabajo se obtuvo la propuesta arquitectónica para el producto “Mis Mejores Cuentos” y la evaluación de la misma.

## PALABRAS CLAVES

arquitectura de software, software educativo, estilos arquitectónicos, patrones arquitectónicos, tecnologías web.

Introducción.....	12
1.1 Introducción.....	16
1.2 Arquitectura de software .....	16
1.3 Estilos arquitectónicos .....	17
1.3.1 Estilos de flujo de datos .....	19
1.3.2 Estilos peer-to-peer .....	19
1.3.3 Estilo de llamada y retorno .....	19
1.4 Patrones arquitectónicos .....	20
1.4.1 Tres capas .....	20
1.4.2 Patrón cliente-servidor .....	21
1.4.3 Modelo - Vista - Controlador (MVC).....	21
1.4.4 Modelo Vista Controlador MM (MVCMM) .....	22
1.4.5 Modelo Vista Controlador Entidad (MVC-E) .....	23
1.5 Patrones de diseño .....	24
1.5.1 Patrones GRASP.....	25
1.5.2 Patrones GoF .....	27
1.6 Metodologías de desarrollo.....	28
1.6.1 Extreme Programming (XP) .....	29
1.6.2 Rational Unified Process (RUP).....	30
1.7 Lenguajes de modelado.....	32

1.7.1 UML 2.0 .....	32
1.7.2 Lenguaje para la modelación Orientada a Objetos de Aplicaciones Multimedia (OMMMA-L).....	32
1.7.3 Lenguaje para la Modelación de Aplicaciones Educativas (ApEM-L 1.5).....	33
1.8 Herramientas CASE.....	34
1.8.1 Rational Suite Rose 2003.....	34
1.8.2 Visual Paradigm 6.4 .....	34
1.9 Sistema de control de versiones.....	35
1.9.1 Subversion (SVN) .....	35
1.10 Cliente Rapsidsvn 0.12 .....	36
1.11 Entornos de Desarrollo Integrado (IDE): Aptana Studio 2.5.....	36
1.12 Servidor Apache 2.2 .....	37
1.13 Lenguajes de programación web .....	37
1.14 Tecnologías de desarrollo web.....	38
1.14.1 Sistema de intercambio de datos .....	38
1.14.2 AJAX.....	39
1.14.3 Framework de desarrollo.....	40
1.15 Conclusiones.....	42
Capítulo 2: Propuesta de arquitectura.....	43
2.1 Introducción.....	43
2.2 Descripción de la arquitectura .....	43
2.2.3 Propósito.....	43



2.2.4 Estructura del equipo de desarrollo .....	43
2.2.5 Estructura del sistema.....	44
2.3 Estándares de codificación .....	45
2.4 Objetivos y Restricciones arquitectónicas.....	47
2.5 Representación arquitectónica del producto .....	48
2.6 Sistema de vistas de ApEM-L.....	50
2.6.1 Vista de Presentación.....	51
2.6.2 Vista Estática.....	52
2.6.3 Vista de arquitectura.....	53
2.7 Conclusiones .....	55
Capítulo 3: Evaluación de la arquitectura.....	56
3.1 Introducción.....	56
3.2 Necesidad de evaluar la arquitectura .....	56
3.3 Etapas en que se evalúa una arquitectura .....	57
3.4 Atributos de calidad.....	57
3.5 Técnicas de evaluación de arquitecturas .....	60
3.5.1. Evaluación basada en escenarios .....	60
3.5.2. Evaluación basada en simulación.....	62
3.5.3. Evaluación basada en modelos matemáticos .....	62
3.5.4. Evaluación basada en experiencia .....	63
3.6 Métodos de evaluación de arquitecturas.....	63

3.7 Evaluación de la arquitectura definida .....	68
3.8 Conclusiones .....	71
Conclusiones Generales.....	72
Recomendaciones .....	73
Referencias Bibliográficas .....	74

# Índice de Figuras

Figura 1. Estructura del MVCMM .....	23
Figura 2. Variación del MVCMM para la arquitectura del software educativo cubano .....	24
Figura 3 Fases de la metodología XP .....	30
Figura 4. Fases e iteraciones de la metodología RUP .....	31
Figura 5. Bases de ApEM-L.....	33
Figura 6. Estructura del sistema.....	45
Figura 7. Propuesta de desarrollo de ApEM-L.....	49
Figura 8. Vista de Gestión del Modelo .....	50
Figura 9. DEP Vista Cuento.....	51
Figura 10. DEN del módulo Cuento.....	52
Figura 11. DC Vista Común.....	53
Figura 12. Diagrama de despliegue .....	54

# Índice de Tablas

Tabla 1. Equipo de Desarrollo .....	44
Tabla 2. Descripción de atributos de calidad observables vía ejecución .....	59
Tabla 3. Descripción de atributos de calidad no observables vía ejecución .....	60
Tabla 4. Pasos del método de evaluación ARID.....	68
Tabla 5. Escenario #1 .....	69
Tabla 6. Escenario #2 .....	69
Tabla 7. Escenario #3 .....	70
Tabla 8. Escenario #4 .....	70
Tabla 9. Escenario #5 .....	71

## Introducción

Uno de los pasos más importantes para el desarrollo de un software es definir su arquitectura. ¿Por qué es esto importante? Porque necesitamos una arquitectura para entender el sistema, organizar su desarrollo, plantear la reutilización del software y hacerlo evolucionar.

“Una arquitectura de software sirve como un puente entre los requerimientos del sistema y su actual implementación” (1). El objetivo principal de la arquitectura del software es aportar elementos que ayuden a la toma de decisiones y al mismo tiempo, proporcionar conceptos y un lenguaje común que permita la comunicación entre los equipos multidisciplinarios que participan en un proyecto.

Para el año 1992 aproximadamente (2) no era común que los proyectos tuvieran una arquitectura de software (en lo adelante, AS) documentada, pero por la necesidad de la organización del desarrollo de programas informáticos, la reutilización de diseño y códigos, y el mejor entendimiento del software, se hace necesario la realización de una descripción de la arquitectura al desarrollar un software. Con una incorrecta e inadecuada descripción de la AS un proyecto de desarrollo tiende a fracasar. La Universidad de las Ciencias Informáticas (UCI) está llamada a respaldar las necesidades de producción de software en las diversas ramas de la economía cubana y se abre camino en el desarrollo de productos para otros países que deseen comercializar con esta institución. En este sentido se desea lograr que el trabajo en los proyectos productivos sea satisfactorio, que los productos se obtengan con la calidad requerida y que cumplan con las expectativas de los clientes, favoreciendo que la industria cubana del software se convierta en un renglón fundamental de la economía del país, por lo que una correcta y precisa descripción de la arquitectura de estos productos es de vital importancia para su desarrollo.

Dentro de la infraestructura productiva de la UCI se encuentran los centros de producción especializados en distintos productos y servicios informáticos. Uno de ellos es el Centro de Tecnologías para la Formación (FORTES) que contribuye al desarrollo de aplicaciones educativas siguiendo una línea de desarrollo libre. El proyecto Producción de Recursos Didácticos pertenece a este centro y tiene como propósito crear el producto “Mis Mejores Cuentos” cuyo objetivo es digitalizar los cuentos infantiles clásicos para los niños de 3 a 5 años de edad. Para este producto se realizó una primera versión de la descripción de la AS que no incluyó un diseño de clases pues los miembros del proyecto no estaban

# Capítulo I

## Fundamentación Teórica

capacitados para ello; las librerías utilizadas para facilitar el trabajo con Javascript (Prototype 1.6.0.3 y Script.aculo.us 1.8.2) carecían de numerosas extensiones (plugins), además de que no se planteó con especificidad cómo utilizar el patrón arquitectónico Modelo Vista Controlador (MVC). Al no utilizar una programación orientada a objetos no es posible fomentar la reutilización del código que es uno de los pilares del desarrollo de software con alta calidad. Un diseño de clases adecuado permite agilizar este proceso además que facilita el trabajo en equipo y la creación de sistemas más complejos por lo que esta descripción inicial de AS no se corresponde con los modelos arquitectónicos para plataformas web y los nuevos paradigmas de programación utilizando herramientas libres.

La situación anteriormente expuesta conlleva al siguiente **problema a resolver**: ¿Cómo definir la organización estructural y funcional del software educativo “Mis Mejores Cuentos” que contribuya a lograr un producto multimedia sobre plataforma web con herramientas libres?

El **objeto de estudio** de la investigación es la arquitectura de software para plataformas web utilizando herramientas libres.

Se propone como **objetivo general** definir una arquitectura de software para el producto “Mis Mejores Cuentos” estableciendo los elementos necesarios para el desarrollo de un producto educativo sobre plataforma web con herramientas libres.

Del objetivo general se derivan los siguientes **objetivos específicos**:

- Elaborar el marco teórico de la investigación y definir la posición teórica del investigador.
- Describir detalladamente la arquitectura de software del sistema.
- Evaluar la arquitectura de software propuesta.

El **campo de acción** de la investigación está enmarcado en la arquitectura de software para el producto “Mis Mejores Cuentos”.

# Capítulo I

## Fundamentación Teórica

Al concluir este trabajo se espera como **posible resultado**: la descripción de la arquitectura de software para el producto “Mis Mejores Cuentos” lo que permitirá establecer una guía para la construcción y desarrollo de la aplicación.

Con esta investigación se defiende la siguiente idea: si se definen correctamente los elementos necesarios para la AS del producto “Mis Mejores Cuentos” centrada en los atributos de calidad establecidos se contribuirá a alcanzar un producto multimedia sobre plataforma web y con herramientas libres.

Para lograr el objetivo general se desarrollan las siguientes **tareas de investigación**:

- Análisis de los conceptos relacionados con la arquitectura de software y definiciones sobre la misma.
- Identificación de los patrones y estilos arquitectónicos que se emplearán durante el proceso de desarrollo.
- Selección de la metodología, herramientas y tecnologías a utilizar para el desarrollo del producto.
- Elaboración de la propuesta arquitectónica a utilizar en el desarrollo del sistema.
- Análisis de los métodos existentes para la evaluación de una arquitectura de software.
- Aplicación del método seleccionado para evaluar la arquitectura de software definida.

Para el cumplimiento de las tareas de investigación se utilizaron los métodos científicos que se describen a continuación.

### **Métodos teóricos**

**Analítico-sintético:** Se realizará un estudio detallado del objeto, logrando resumir y exponer los resultados obtenidos del análisis, así como precisar las características del modelo arquitectónico propuesto.

# Capítulo I

## Fundamentación Teórica

**Histórico-lógico:** Se utilizará para estudiar la evolución histórica y desarrollo del objeto de estudio de la investigación.

El presente documento se estructura en 3 capítulos, cada uno con su introducción y conclusiones:

**Capítulo 1: Fundamentación Teórica**, en este capítulo se define el marco teórico de la investigación, se realiza un estudio del estado del arte de la arquitectura de software y conceptos relacionados con la misma. Se describen los patrones y los estilos arquitectónicos utilizados en la actualidad, así como metodologías de desarrollo, lenguaje de modelado, herramientas y tecnología a utilizar.

**Capítulo 2: Propuesta de arquitectura**, en este capítulo se realiza la propuesta de solución al problema planteado en el diseño teórico de la investigación.

**Capítulo 3: Evaluación de la arquitectura**, en este capítulo se exponen los principales métodos utilizados para evaluar las arquitecturas. Se describe además el método seleccionado para aplicarlo en el producto “Mis Mejores Cuentos”, logrando así la validación de la arquitectura de software propuesta.



# Capítulo I

## Fundamentación Teórica

### Capítulo 1: Fundamentación teórica

#### 1.1 Introducción

La AS remonta sus antecedentes a la década del '60; su historia no ha sido tan continua como la del campo más amplio en el que se inscribe: la ingeniería de software (3). En el presente capítulo se introduce al lector en los principales términos asociados a este concepto, sus antecedentes, definiciones se detallan además los patrones y estilos arquitectónicos más utilizados. Por otro lado se investigan las herramientas, tecnologías y metodologías existentes para el desarrollo de software en la actualidad.

#### 1.2 Arquitectura de software

Los orígenes de la AS datan de 1968 cuando Edsger Dijkstra, de la Universidad Tecnológica de Eindhoven (Holanda) propuso que se estableciera una estructuración correcta de los sistemas de software antes de lanzarse a programar escribiendo código de cualquier manera (4). Pero no es hasta la década del '90 que alcanza mayor popularidad.

Definir la arquitectura es uno de los pasos más importantes en el desarrollo de software, debido a que esta permite que se planifique el esfuerzo del equipo de desarrollo, definiendo una serie de reglas y pautas que guiarán el avance de la aplicación, siguiendo una línea coherente y asegurando un por ciento importante de éxito.

El objetivo principal de la AS es aportar elementos que ayuden a la toma de decisiones y al mismo tiempo, proporcionar conceptos así como un lenguaje común que permita la comunicación entre los equipos que participen en un proyecto. Para conseguirlo, la AS construye abstracciones, materializándolas en forma de diagramas comentados (5).

Son muchas las instituciones y autores que han dado su definición de AS, a continuación se relacionan algunas.

# Capítulo I

## Fundamentación Teórica

La definición oficial que ofrece la IEEE Std 1471-2000, plantea (6): “La arquitectura de software es la organización fundamental de un sistema formada por sus componentes, las relaciones entre ellos y el contexto en el que se implantarán y los principios que sustentan su diseño y evolución.” Una definición muy reconocida es la de Clements (7): “La arquitectura de software es, a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema. La vista arquitectónica es una vista abstracta, aportando el más alto nivel de comprensión y la supresión o diferimiento del detalle inherente a la mayor parte de las abstracciones”. Otro concepto de AS es el siguiente: La arquitectura de software consiste en la estructura o sistema de estructuras, que comprenden los elementos de software, las propiedades externas visibles de esos elementos y la relación entre ellos (8). Constituye además un puente entre el requerimiento y el código, ocupando el lugar que en los gráficos antiguos se reservaba para el diseño (9).

A pesar de la abundancia de definiciones del campo de la AS, existe de forma general un acuerdo de que se refiere a la estructura a grandes rasgos del sistema, estructura consistente en componentes y relaciones entre ellos (8).

A partir de las definiciones anteriores se puede concluir que la arquitectura de software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución.

### **1.3 Estilos arquitectónicos**

En el texto fundacional de la AS, algunos autores (2) establecieron el razonamiento sobre estilos de arquitectura como uno de los aspectos fundamentales de la disciplina. Según estos autores: “Un estilo es un concepto descriptivo que define una forma de articulación u organización arquitectónica. El conjunto de los estilos cataloga las formas básicas posibles de estructuras de software, mientras que las formas complejas se articulan mediante composición de los estilos fundamentales”.

# Capítulo I

## Fundamentación Teórica

De igual forma, Shaw y Clements (10) definieron los estilos arquitectónicos como un conjunto de reglas de diseño que identifica las clases de componentes y conectores que se pueden utilizar para componer un sistema o subsistema. Los estilos conjugan los componentes, conectores, configuraciones y restricciones de un sistema. La descripción de un estilo se puede formular en lenguaje natural o en diagramas, pero lo mejor es hacerlo en un lenguaje de descripción arquitectónica o en lenguajes formales de especificación.

En 1999, Klein y Kazman (11) formularon una definición según la cual un estilo arquitectónico es una descripción del patrón de los datos y la interacción de control entre los componentes, ligada a una descripción informal de los beneficios e inconvenientes aparejados por el uso del estilo. Los estilos arquitectónicos, afirman, son artefactos de ingeniería importantes porque definen clases de diseño junto con las propiedades conocidas asociadas a ellos. Ofrecen evidencia basada en la experiencia sobre la forma en que se ha utilizado históricamente cada clase, junto con razonamiento cualitativo para explicar por qué cada clase tiene esas propiedades específicas.

Teniendo en cuenta todas estas definiciones es posible decir que un estilo arquitectónico define las reglas generales de la organización en términos de un patrón y las restricciones en la forma y la estructura de un grupo numeroso de sistemas de software. Son entidades que ocurren en un nivel sumamente abstracto y puramente arquitectónico.

Existe una gran variedad de estilos arquitectónicos dentro de los que se pueden mencionar (12):

- Estilos de flujo de datos.
- Estilos peer-to-peer.
- Estilos de llamada y retorno.
- Estilos centrados en datos.
- Estilos de código móvil.
- Estilos heterogéneos.

# Capítulo I

## Fundamentación Teórica

### 1.3.1 Estilos de flujo de datos

Esta familia de estilos enfatiza la reutilización y la modificabilidad. Es apropiada para los grupos de sistemas que implementan transformaciones de datos en pasos sucesivos. Algunos ejemplos de las mismas son la arquitectura de tubería-filtros y las que presentan procesos secuenciales en lotes (13).

### 1.3.2 Estilos peer-to-peer

Esta familia, también llamada de componentes independientes, enfatiza la modificabilidad por medio de la separación de las diversas partes que intervienen en la computación. Consiste por lo general en procesos independientes o entidades que se comunican a través de mensajes. Cada entidad puede enviar mensajes a otras entidades, pero no controlarlas directamente (13).

- Arquitectura basada en eventos.
- Arquitecturas Orientadas a Servicios (SOA).
- Arquitecturas basadas en recursos.

### 1.3.3 Estilo de llamada y retorno

Esta familia de estilos enfatiza la modificabilidad y la escalabilidad. Son los estilos más generalizados en sistemas de gran escala. Miembros de la familia son las arquitecturas de programa principal y subrutina, los sistemas basados en llamadas a procedimientos remotos, los sistemas orientados a objeto y los sistemas jerárquicos en capas. Los sistemas basados en llamada y retorno reflejan la estructura del lenguaje de programación (13).

- Modelo - Vista - Controlador (MVC).
- Arquitectura en capas.
- Arquitectura orientada a objetos.
- Arquitectura basada en componentes.

# Capítulo I

## Fundamentación Teórica

### 1.4 Patrones arquitectónicos

Los patrones de arquitectura expresan el esquema fundamental de organización para sistemas de software. (13) Proveen un conjunto de subsistemas predefinidos, especifican sus responsabilidades e incluyen reglas y guías para organizar las relaciones entre ellos. Ayudan a especificar la estructura fundamental de una aplicación.

#### 1.4.1 Tres capas

Este patrón en capas es definido como una organización jerárquica tal que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior (1).

**Capa de presentación:** Conocida como interfaz gráfica del usuario en el sistema, es la encargada de presentar la interfaz del sistema al usuario, comunicar la información y capturar la información del mismo. Esta capa se comunica únicamente con la capa de negocio.

**Capa de negocio:** Es el contenedor de procesos del sistema, donde se ejecutan los programas que dan soporte a las funcionalidades y reglas que deben cumplirse. Esta capa se comunica con la capa de presentación, para recibir las solicitudes y presentar los resultados, y con la capa de datos, para solicitar al gestor de base de datos para almacenar o recuperar datos de él.

**Capa de datos:** Es donde reside la información persistente del sistema. Está formada por uno o más gestores de bases de datos que realizan todo el almacenamiento de datos, reciben solicitudes de almacenamiento o recuperación de información desde la capa de negocio (14).

Las ventajas del estilo en capas son obvias: el estilo soporta un diseño basado en niveles de abstracción crecientes, lo cual a su vez permite a los implementadores la partición de un problema complejo en una secuencia de pasos incrementales; mejora el soporte del sistema al admitir muy naturalmente optimizaciones y refinamientos; y proporciona amplia reutilización ya que se pueden utilizar diferentes

# Capítulo I

## Fundamentación Teórica

implementaciones o versiones de una misma capa en la medida que soporten las mismas interfaces de cara a las capas adyacentes (13).

### 1.4.2 Patrón cliente-servidor

Está basado en el principio clásico de divide y vencerás donde el procesamiento se fracciona entre dos entidades fundamentales denominadas cliente y servidor. Surgió en la década de los '80 como respuesta a la evolución del hardware en términos de redes, planteando un modelo versátil, modular basado en mensajes que permite incrementar la flexibilidad, la escalabilidad y la inter - operación de los sistemas. El cliente se define como el proceso que requiere un servicio en particular y el servidor como el proceso que provee dicho servicio. Debido a que clientes y servidores son conceptos a nivel de software y no de hardware, una misma máquina puede actuar como cliente y servidor al mismo tiempo (15).

#### Ventajas de la arquitectura:

**Centralización del control:** La arquitectura centraliza el control de los accesos, recursos y la integridad de los datos en el servidor garantizando que una aplicación cliente que se encuentre con defecto o que no esté autorizada no pueda dañar los datos. Además de que permite ir aumentando el número de clientes y servidores paulatinamente y no necesariamente a la misma vez, quiere decir que se pueden agregar aplicaciones clientes y servidoras por separado a lo que se le llama **Escalabilidad** (16).

### 1.4.3 Modelo - Vista - Controlador (MVC)

El patrón MVC separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres partes diferentes (17):

- **Modelo:** El modelo administra el comportamiento y los datos del dominio de aplicación, responde a requerimientos de información sobre su estado (usualmente formulados desde la vista) y responde a instrucciones de cambiar el estado (habitualmente desde el controlador).
- **Vista:** Maneja la visualización de la información.

# Capítulo I

## Fundamentación Teórica

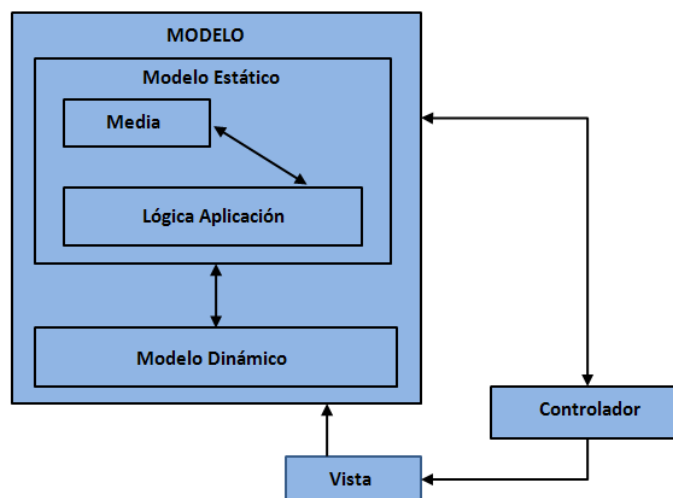
- Controlador: Interpreta las acciones del ratón y el teclado, informando al modelo y/o a la vista para que cambien según resulte apropiado.

Tanto la vista como el controlador dependen del modelo, el cual no depende de las otras clases. Esta separación permite construir y probar el modelo independientemente de la representación visual (13).

Dado que la vista se halla separada del modelo y no hay dependencia directa del modelo con respecto a la vista, la interfaz de usuario puede mostrar múltiples vistas de los mismos datos simultáneamente a esta ventaja se le denomina **soporte de vistas múltiples** por ejemplo, múltiples páginas de una aplicación Web pueden utilizar el mismo modelo de objetos, mostrado de maneras diferentes. Además de esta ventaja se encuentra la llamada **adaptación al cambio** en la que los requerimientos de interfaz de usuario tienden a cambiar con mayor rapidez que las reglas de negocio. Dado que el modelo no depende de las vistas, agregar nuevas opciones de presentación generalmente no afecta al modelo. Este patrón sentó las bases para especializaciones ulteriores, tales como Page Controller y Front Controller (13). MVC aporta una construcción de software mantenible, en la que se pueden localizar de forma ágil los errores, ofreciendo maneras sencillas para probar el correcto funcionamiento del sistema además que facilita el mantenimiento en caso de errores y el escalamiento de la aplicación en caso de ser requerido. Supone un diseño modular, y muy poco acoplado, favoreciendo la reutilización.

### 1.4.4 Modelo Vista Controlador MM (MVCMM)

Desde el surgimiento del MVC y con el paso de los años se han desarrollado diferentes variantes, dentro de las que se encuentra la modificada para Aplicaciones Multimedia, conocida como MVCMM. En esta variante se diversifica la clase modelo incorporando al esquema dos nuevos tipos de clases: la modelo dinámica y la modelo estática, la cual a su vez se subdivide en las clases media y lógica de la aplicación (18).



**Figura 1. Estructura del MVCMM**

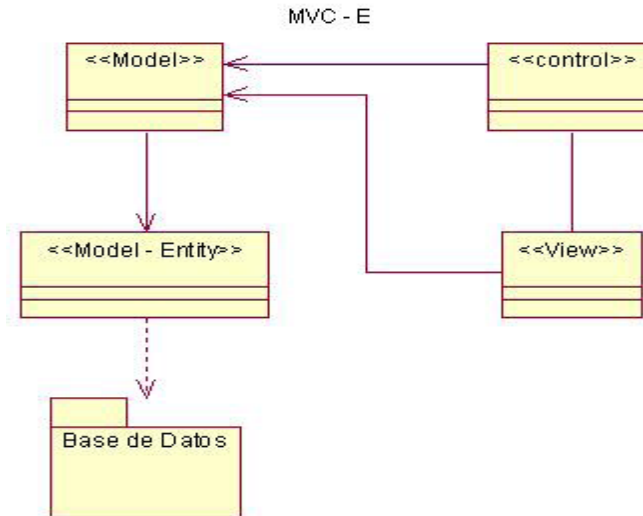
### 1.4.5 Modelo Vista Controlador Entidad (MVC-E)

El MVCMM para un mejor entendimiento y comprensión de aplicaciones educativas recibe un conjunto de transformaciones, dando como resultado, el patrón Modelo – Vista – Controlador – Entidad (MVC-E) como propuesta arquitectónica del software educativo donde se descargan las responsabilidades de la clase modelo concernientes al procesamiento y almacenamiento de la información persistente de las aplicaciones, a una nueva clase incorporada al modelo denominada Modelo Entidad, con dos tipos fundamentales, la clase Modelo – Entidad – Media y Modelo – Entidad - Persistente. La primera de estas con la responsabilidad de agrupar las clases que identifican las medias y su árbol de jerarquía en la aplicación y la segunda tienen como responsabilidad la gestión de la información persistente, que antes sobrecargaba a la clase Modelo del patrón MVC original. Esta variación a su vez sustenta las características actuales de los sistemas multimedia como son: comunicación con bases de datos, archivos XML, archivos JSON, o sistemas externos (18).



# Capítulo I

## Fundamentación Teórica



**Figura 2. Variación del MVCMM para la arquitectura del software educativo cubano**

Desde el punto de vista de que la arquitectura de la aplicación estará basada en los principios de la Programación Orientado a Objetos (POO). Los principales elementos de la arquitectura estarán centrados hacia los objetos y serán estos las unidades de modelado a partir de las clases que los definen interactuando a través de funciones y métodos. Por este motivo se selecciona el estilo arquitectónico de Llamada y Retorno y como patrón arquitectónico el Modelo Vista Controlador Entidad como variante de solución para las aplicaciones educativas cubanas.

### 1.5 Patrones de diseño

Son soluciones simples y elegantes a problemas específicos y comunes del diseño orientado a objetos. Estos centran su desarrollo en aspectos organizativos, sociales y económicos de un sistema. Con la selección de los patrones de diseño, el sistema tiende a ser más fácil de entender, mantener y ampliar, existen más oportunidades para reutilizar componentes en futuras aplicaciones (19).

A continuación se exponen algunos de los patrones de diseño que estarán presente en el modelado del software.

# Capítulo I

## Fundamentación Teórica

### 1.5.1 Patrones GRASP

GRASP es un acrónimo que significa General Responsibility Assignment Software Patterns (Patrones Generales de Software para Asignar Responsabilidades). Los patrones GRASP describen los principios fundamentales de diseño de objetos para la asignación de responsabilidades. Constituyen la base del cómo se diseñará el sistema y se aplican en los primeros momentos del diseño.

#### 1.5.1.1 Patrón Experto

Permite asignar una responsabilidad al experto en información. Nos indica que la responsabilidad de la creación de un objeto o la implementación de un método, debe recaer sobre la clase que conoce toda la información necesaria para crearlo. De este modo se obtendrá un diseño con mayor cohesión y así la información se mantendrá encapsulada. Entre sus ventajas están:

- Se conserva el encapsulamiento, ya que los objetos se valen de su propia información para hacer lo que se les pide.
- El comportamiento se distribuye entre las clases que cuentan con la información requerida, alentando con ello definiciones de clases sencillas y más cohesivas, las cuales son fáciles de comprender y mantener.

#### 1.5.1.2 Patrón Creador

Define quién debería ser el responsable de la creación de una nueva instancia de alguna clase. Guía la asignación de responsabilidades relacionadas con la creación de objetos. El propósito fundamental de este patrón es encontrar un creador que se debe conectar con el objeto producido en cualquier evento. Dentro de sus ventajas están:

- Guiar la asignación de responsabilidades relacionadas con la creación de objetos
- Se encuentra un creador que necesite conectarse al objeto creado en alguna situación, eligiéndolo como el creador, se favorece el bajo acoplamiento

# Capítulo I

## Fundamentación Teórica

### 1.5.1.3 Patrón Bajo acoplamiento

Este patrón define como dar soporte a una dependencia escasa y a un aumento de la reutilización, enfocándose en asignar una responsabilidad para mantener bajo acoplamiento. El bajo acoplamiento estimula la asignación de responsabilidades de forma tal que, la inclusión de estas no incremente el acoplamiento, creando clases más independientes y con mayor resistencia al impacto de los cambios, que aumentan la productividad y la posibilidad de reutilización. Es válido aclarar que este patrón no puede verse de forma independiente a los patrones Experto y Alta cohesión, sino más bien incluirse como otro de los principios del diseño que influyen de forma determinante a la hora de la asignación de responsabilidades.

#### Ventajas

- No afectan los cambios en otros componentes.
- Fácil de entender de manera aislada.
- Conveniente para reutilizar.

### 1.5.1.4 Patrón Alta cohesión

El propósito de este patrón es asignar una responsabilidad de manera que la cohesión permanezca alta; posibilitando mucha facilidad de mantenimiento, comprensión y uso. Brinda un alto grado de funcionalidad, combinada con una reducida cantidad de operaciones, también simplifica el mantenimiento y las mejoras a las clases que componen el software.

#### Ventajas

- Se incrementa la claridad y facilita la comprensión del diseño.
- Se simplifican el mantenimiento y las mejoras.
- Se soporta a menudo bajo acoplamiento.

# Capítulo I

## Fundamentación Teórica

### 1.5.1.5 Patrón Controlador

El patrón controlador es un patrón que sirve como intermediario entre una determinada interfaz y el algoritmo que la implementa, de tal forma que es el que recibe los datos del usuario y el que los envía a las distintas clases según el método llamado. Este patrón sugiere que la lógica de negocio debe estar separada de la capa de presentación, esto para aumentar la reutilización de código y a la vez tener un mayor control. Se recomienda dividir los eventos del sistema en el mayor número de controladores para poder aumentar la cohesión y disminuir el acoplamiento.

Además de estos patrones se utilizan otros más específicos acordes a la aplicación que se desea desarrollar, como son:

### 1.5.2 Patrones GoF

Gang-of-Four es el nombre con el que se conoce comúnmente a los autores del libro Design Patterns. Un patrón GoF es una descripción de clases y objetos que se comunican entre sí, adaptada para resolver un problema general de diseño en un contexto particular.

#### 1.5.2.1 Patrón Singleton

El Singleton es quizás el más sencillo de los patrones que se presentan en el catálogo de los patrones GoF. Es también uno de los patrones más conocidos y utilizados. Su propósito es asegurar que sólo exista una instancia de la clase. Cuando varios elementos distintos precisan referenciar a un mismo elemento y se desea asegurar que no hay más de una instancia de ese elemento, simplemente el patrón Singleton determina un punto de acceso global a esta clase o instancia garantizando resolver el problema.

#### 1.5.2.2 Patrón Facade

Su objetivo es proporcionar una interfaz unificada de alto nivel que, representando a todo un subsistema, facilite su uso. La “fachada” satisface a la mayoría de los clientes, sin ocultar las funciones de menor nivel a aquellos que necesiten acceder a ellas. Lo usamos cuando se pretende estructurar en capas el subsistema, para que cada capa tenga su propia fachada. Lo que nos permitirá usar las clases del

# Capítulo I

## Fundamentación Teórica

subsistema que necesitemos sin ningún obstáculo. De esta forma se puede elegir entre facilidad de uso y generalidad.

### 1.5.2.3 Patrón Observer

Define una dependencia de uno-a-muchos entre objetos, de forma que cuando un objeto cambia de estado se notifica y actualizan automáticamente todos los objetos. En la aplicación serán utilizados para actualizar determinados elementos que dependen directamente de los datos de otros, específicamente para actualizar la interfaz de usuario ante un cambio en los datos cargados desde los ficheros JSON.

### 1.6 Metodologías de desarrollo

Una metodología de desarrollo de software es un conjunto de pasos y procedimientos que deben seguirse para desarrollar software (20). Desarrollar un buen software depende de un sin número de actividades y etapas, donde el impacto de elegir la mejor metodología para un equipo en un determinado proyecto, es trascendental, para el éxito del producto.

A continuación se detallan los dos grandes enfoques de las metodologías, metodologías tradicionales y metodologías ágiles, las primeras recalcan el uso exhaustivo de documentación durante todo el ciclo del proyecto, mientras que las segundas ponen vital importancia en la capacidad de respuesta a los cambios además de mantener una buena relación con el cliente para llevar el éxito al proyecto.

Las metodologías tradicionales imponen una disciplina de trabajo sobre el proceso de desarrollo del software, con el fin de conseguir un software más eficiente. Para ello, se hace énfasis en la planificación total de todo el trabajo a realizar y una vez que está todo detallado, comienza el ciclo de desarrollo del producto. Se centran especialmente en el control del proceso, mediante una rigurosa definición de roles, actividades, artefactos, herramientas y notaciones para el modelado y documentación detallada. Entre las metodologías tradicionales o pesadas se encuentran: RUP (Rational Unified Process), MSF (Microsoft Solution Framework), entre otras (21).

# Capítulo I

## Fundamentación Teórica

Por otra parte, las metodologías ágiles proporcionan una serie de pautas y principios junto a técnicas pragmáticas que harán la entrega del proyecto menos complicada y más satisfactoria tanto para los clientes como para los equipos de entrega (21). Especialmente preparadas para cambios durante el proyecto y con un proceso menos controlado, con pocos principios. Entre las metodologías ágiles más reconocidas: XP (Extreme Programming), Scrum, y Crystal Clear.

Para poder seleccionar la metodología que se va a utilizar y cuál se adapta mejor a nuestro medio se analizarán XP y RUP. La primera por ser aplicable a proyectos de corto plazo con requisitos cambiantes que no genera gran documentación, y la segunda porque es un proceso en el que se define claramente quien, cómo, cuándo y qué debe hacerse en el proyecto. Además implementa las mejores prácticas de desarrollo de software.

### **1.6.1 Extreme Programming (XP)**

XP es una de las metodologías de desarrollo de software más exitosas en la actualidad, utilizada para proyectos de corto plazo. La metodología consiste en una programación rápida extrema, basándose en la simplicidad, la comunicación y el reciclado continuo de código. Cuya particularidad es tener como parte del equipo, al usuario final, pues es uno de los requisitos para llegar al éxito del proyecto. Esta es adecuada para proyectos de corto plazo con requisitos imprecisos y muy cambiantes, donde existe un alto riesgo técnico.

Los objetivos de XP son muy simples: la satisfacción del cliente, esta metodología trata de dar al cliente el software que él necesita y cuando lo necesita. El segundo objetivo es potenciar al máximo el trabajo en grupo. Tanto los jefes de proyecto, los clientes y desarrolladores, son parte del equipo y están involucrados en el desarrollo del software. Está compuesta por cuatro fases: planificación, diseño, codificación y prueba (22).

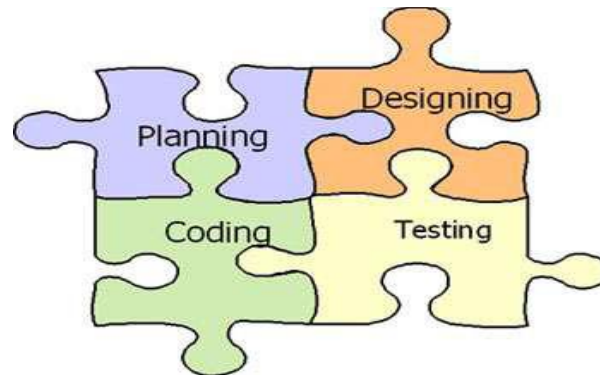


Figura 3 Fases de la metodología XP (22)

### 1.6.2 Rational Unified Process (RUP)

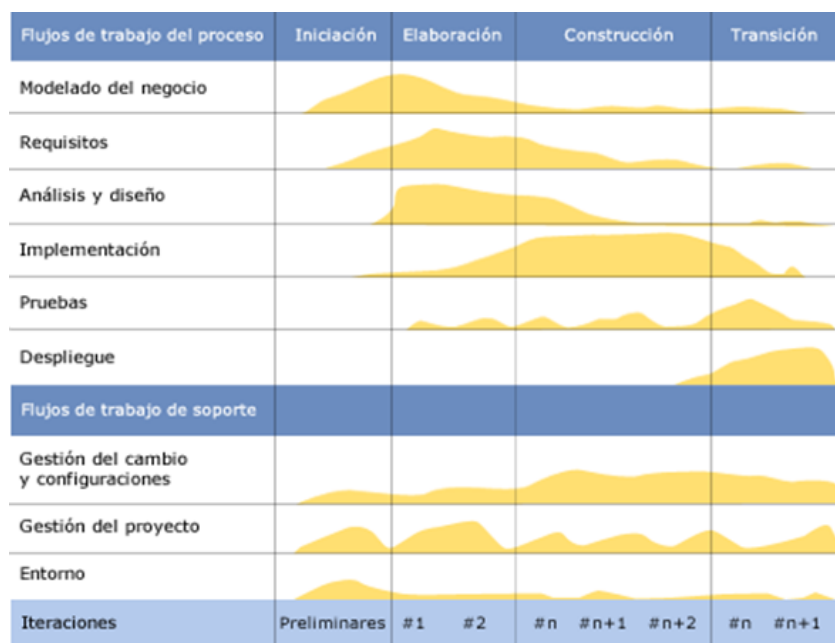
Es una metodología para la ingeniería de software que va más allá del análisis y diseño orientado a objetos para proporcionar una familia de técnicas que soportan el ciclo completo de desarrollo de software, cuyo objetivo es producir software de alta calidad, es decir, que cumpla con los requerimientos de los usuarios dentro de una planificación y presupuesto establecido. Es un proceso de desarrollo de software cuyo ciclo de vida se caracteriza por (23):

**Dirigido por casos de uso:** Esto significa que el proceso de desarrollo sigue una trayectoria que avanza a través de los flujos de trabajo generados por los casos de uso. Los casos de uso se especifican y diseñan al principio de cada iteración, y son la fuente a partir de la cual los ingenieros de prueba construyen sus casos de prueba. Estos describen la funcionalidad total del sistema.

**Centrado en la arquitectura:** Los casos de uso guían a la arquitectura del sistema y esta influye en la selección de los casos de uso. La arquitectura de un sistema es la organización o estructura de sus partes más relevantes. Una arquitectura ejecutable es una implementación parcial del sistema, construida para demostrar algunas funciones y propiedades. RUP establece refinamientos sucesivos de una arquitectura ejecutable, construida como un prototipo evolutivo.

## Fundamentación Teórica

**Iterativo e incremental:** RUP divide el proceso en cuatro fases, dentro de las cuales se realizan varias iteraciones en número variable según el proyecto y las cuales se definen según el nivel de madurez que alcanzan los productos que se van obteniendo con cada actividad ejecutada. La terminación de cada fase ocurre en el hito correspondiente a cada una, donde se evalúa que se hayan cumplido los objetivos de la fase en cuestión.



**Figura 4. Fases e iteraciones de la metodología RUP**

Una particularidad de esta metodología es que, en cada ciclo de iteración, se hace exigente el uso de artefactos, siendo por este motivo una de las metodologías más importantes para alcanzar un grado de certificación en el desarrollo del software.

RUP resulta ser la metodología más indicada para guiar el proceso de desarrollo de software pues esta metodología permite generar todos los modelos como principales artefactos en cada uno de los flujos de trabajo, garantizando además una arquitectura robusta desde las primeras etapas del ciclo de vida



# Capítulo I

## Fundamentación Teórica

además de la amplia documentación que brinda, factor clave para el proyecto pues le aporta requisitos de calidad, escalabilidad y organización al software a desarrollar.

### 1.7 Lenguajes de modelado

El proceso de desarrollo de software requiere además de una metodología un lenguaje de modelado, entre los más reconocidos se encuentran UML, OMMMA-L y ApEM-L que serán descritos a continuación.

#### 1.7.1 UML 2.0

El Lenguaje Unificado de Modelado (UML) es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software. Captura decisiones y conocimiento sobre los sistemas que se deben construir. Se usa para entender, diseñar, configurar, mantener y controlar la información sobre tales sistemas. Este fue desarrollado en el año 1996 por Ivar Jacobson, Grady Booch y James Rumbaugh, en la corporación “Rational Software” (24).

Consta de cinco áreas conceptuales que son: estructura estática, comportamiento dinámico, construcciones de implementación, organización del modelo y los mecanismos de extensión. A su vez, consta de ocho vistas: estática, de casos de uso, de implementación, de despliegue, de máquinas de estado, de actividad, de interacción y de gestión del modelo; para la modelación de los productos, a través de un conjunto de diagramas distribuidos por cada una de estas vistas (25).

#### 1.7.2 Lenguaje para la modelación Orientada a Objetos de Aplicaciones Multimedia (OMMMA-L)

Consta de cuatro vistas fundamentales en su modelación: vista lógica, vista de presentación espacial, vista de comportamiento temporal predefinido y vista de control interactivo. Modifica los diagramas originales de UML de: clases, secuencia y estado. Añade como parte de la vista de presentación espacial un nuevo diagrama: el diagrama de presentación, para la representación espacial de los elementos visuales del futuro software multimedia (24). Sin embargo no representa las modificaciones descriptivas y decorativas para un producto multimedia (26).

# Capítulo I

## Fundamentación Teórica

### 1.7.3 Lenguaje para la Modelación de Aplicaciones Educativas (ApEM-L 1.5)



**Figura 5. Bases de ApEM-L**

El Lenguaje para la Modelación de Aplicaciones Educativas y Multimedia (ApEM-L) se presenta como una extensión de UML que permite incorporar a este los elementos fundamentales del proceso productivo UCI, en ApEM-L se incorporan los elementos más significativos de OMMMA-L, para de esta forma lograr una extensión consistente para la modelación de aplicaciones educativas. ApEM-L no modifica la semántica del lenguaje base UML, sino que trabaja en estereotipos restrictivos, por lo que a su vez produce modificaciones descriptivas y decorativas en la representación de los componentes del lenguaje base (18).

Los conceptos y modelos de ApEM-L están agrupados en las siguientes áreas conceptuales: Presentación, Estructura lógica, Comportamiento dinámico y Gestión del modelo.

Se selecciona como lenguaje de modelado APEM-L 1.5 porque es un lenguaje orientado a la descripción de las vistas del software educativo, responde además a una mejor modelación de aplicaciones educativas ya que en él se incorporan nuevos diagramas y estereotipos que permiten una mejor comprensión del funcionamiento de aplicaciones con estas características.

# Capítulo I

## Fundamentación Teórica

### 1.8 Herramientas CASE

Las Herramientas CASE (Computer Aided Software Engineering, Ingeniería de Software Asistida por Ordenador) permiten efectuar la modelación de diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste de las mismas en términos de tiempo y de dinero (27).

#### 1.8.1 Rational Suite Rose 2003

Rational Rose es una herramienta con plataforma independiente que ayuda a la comunicación entre los miembros del equipo, a monitorear el tiempo de desarrollo y a entender el entorno de los sistemas. Utiliza la notación estándar UML, la cual posibilita a los arquitectos de software y desarrolladores visualizar el sistema completo utilizando un lenguaje común. Abarca todo el ciclo de vida de un proyecto: concepción y formalización del modelo, construcción de los componentes, transición a los usuarios y certificación de las distintas fases. Los diseñadores pueden tomar ventajas de esta herramienta, porque proporciona que la salida de una iteración sea la entrada de la próxima que está por venir. Puede generar código en Java, C++, Visual Basic, Ada, Corba y Oracle (28). Es software propietario por lo que queda descartado como herramienta de modelado a utilizar.

#### 1.8.2 Visual Paradigm 6.4

Visual Paradigm es una herramienta de modelado que está diseñada para un gran número de usuarios, incluyendo ingenieros de sistemas, analistas de sistemas, analistas de negocio, arquitectos de sistemas. Soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Permite construir todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. Una de las características más importantes de Visual Paradigm es que es multiplataforma (29).

Como herramienta CASE se selecciona Visual Paradigm 6.4 por las ventajas que posee dicha herramienta para el modelo de sistemas además por las facilidades que brinda al usuario a partir de una interfaz

# Capítulo I

## Fundamentación Teórica

amigable y fácil de utilizar. Una razón determinante en la selección de dicha herramienta lo constituyó la necesidad de un modelado rápido de la aplicación a desarrollar.

### 1.9 Sistema de control de versiones

Se llama control de versiones a la gestión de los diversos cambios que se realizan sobre los elementos de algún producto o una configuración del mismo. Los sistemas de control de versiones facilitan la administración de las distintas versiones de cada producto desarrollado, así como las posibles especializaciones realizadas (por ejemplo, para algún cliente específico) (30).

#### 1.9.1 Subversion (SVN)

Subversion, también conocido con las siglas SVN, es un software destinado a facilitar el trabajo en equipo sobre un conjunto de ficheros, promoviendo con esto la colaboración entre los miembros de un equipo de trabajo de una manera muy cómoda. Normalmente consiste en una copia maestra en un repositorio central, y un programa cliente con el que cada usuario sincroniza su copia local. Comparte los cambios sobre un mismo conjunto de ficheros. El repositorio guarda registros de los cambios realizados por cada usuario, y permite volver a un estado anterior en caso de necesidad (31).

Entre las ventajas que SVN brinda podemos encontrar:

- Brinda la posibilidad de realizar copias de seguridad centralizadas (solo el administrador debe preocuparse de realizar copias de seguridad en el repositorio).
- Las modificaciones (incluyendo cambios a varios archivos) son atómicas.
- La creación de ramas y etiquetas es una operación más eficiente.
- Maneja eficientemente archivos binarios.
- Proporciona el bloqueo de archivos: el usuario bloquea el fichero durante su edición, evitando el acceso concurrente de otros usuarios.
- Acceso remoto (es posible acceder remotamente al repositorio).

# Capítulo I

## Fundamentación Teórica

### 1.10 Cliente Rapidsvn 0.12

Proporciona una interfaz fácil de usar para Subversion, eficaz, sencillo para los principiantes, pero lo suficientemente flexible como para aumentar la productividad para los usuarios experimentados de Subversion, portable funciona en cualquier plataforma en la que Subversion y wxWidgets se puedan ejecutar: Linux, Windows, Mac OS / X, Solaris, etc (32).

### 1.11 Entornos de Desarrollo Integrado (IDE): Aptana Studio 2.5

Un Entorno de Desarrollo Integrado o, en inglés, Integrated Development Environment (IDE), es un entorno de programación que consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica GUI. Puede dedicarse en exclusiva a un solo lenguaje de programación o bien, utilizarse para varios. Pueden ser aplicaciones por sí solas o ser parte de aplicaciones existentes (33).

Aptana es un IDE para el desarrollo web de código abierto. Incorpora características completas, sincronización, administración de proyectos, funciones mediante plugins, posibilidad de realizar subidas y bajadas de ficheros a un servidor vía File Transfer Protocol FTP y Secure File Transfer Protocol SFTP, soporte para los sistemas operativos Microsoft Windows, Mac OS X y Linux en sus versiones/distribuciones más recientes (34). Posee una interfaz amigable y fácil de utilizar. Cuenta con un depurador de errores en tiempo real, y un útil índice de funciones. El programa brinda diversas utilidades que favorecen el desarrollo de distintas aplicaciones Web, en especial archivos HyperText Markup Language (HTML). Vigila y compila los cambios que se le realicen al proyecto en tiempo real. Permite comprobar la compatibilidad de las funciones con los diferentes navegadores, multiplataforma, sincronización con carpetas locales y remotas (35).

Es de gran importancia seleccionar de manera adecuada el ambiente de desarrollo. Para poder explotar todos sus beneficios y agilizar el desarrollo del software se escoge Aptana Studio 2.5 como IDE por ser una herramienta libre y por todas las ventajas descritas con anterioridad.

# Capítulo I

## Fundamentación Teórica

### 1.12 Servidor Apache 2.2

De acuerdo con una encuesta realizada por la empresa inglesa Netcraft, en septiembre de 2009, los servidores web más usados en la actualidad son Internet Information Services (IIS) y Apache (36).

Internet Information Services (IIS) es un conjunto de servicios para el sistema operativo Windows por lo que se descarta en esta investigación mientras que Apache es un servidor web de código abierto donde su configurabilidad, robustez, estabilidad y rapidez hacen que cada vez millones de servidores reiteren su confianza en este programa (37).

Apache está diseñado para ser un servidor web potente y flexible que funciona en la más amplia variedad de plataformas y entornos. Las diferentes plataformas y entornos, hacen que a menudo sean necesarias diferentes características o funcionalidades. Apache se ha adaptado siempre a una gran variedad de entornos a través de su diseño modular. Este diseño permite a los administradores de sitios web elegir que características van a ser incluidas en el servidor seleccionando que módulos se van a cargar, ya sea al compilar o al ejecutar el servidor. Ha alcanzado gran popularidad en ámbitos empresariales debido entre otras razones a (38):

- Constituye una tecnología gratuita de código fuente abierta.
- Permite personalizar la respuesta ante los posibles errores que se puedan dar en el servidor.
- Es posible configurar Apache para que ejecute un determinado script cuando ocurra un error en concreto.

### 1.13 Lenguajes de programación web

Desde el surgimiento de los lenguajes de programación han sido muchos los que han surgido, cada uno con sus características específicas. Dentro de estos lenguajes de programación se encuentra VBScript, Javascript entre otros; el primero es similar a los lenguajes Visual Basic que utilizan los desarrolladores de Windows y tiene, esencialmente, la misma estructura que Javascript. VBScript es el lenguaje de scripting desarrollado por Microsoft para páginas web con el que se pueden realizar efectos para el navegador

# Capítulo I

## Fundamentación Teórica

Internet Explorer por lo que en principio es una ventaja para programadores experimentados en este sistema, pero una desventaja para los demás (39). El segundo es un lenguaje de programación interpretado, por lo que no es necesario compilar los programas para ejecutarlos, los programas escritos en Javascript se pueden probar directamente en cualquier navegador sin necesidad de procesos intermedios. Javascript no guarda ninguna relación directa con el lenguaje de programación Java. Legalmente, Javascript es una marca registrada de la empresa Sun Microsystems. Es el lenguaje de script por defecto de la mayoría de los navegadores, es multiplataforma y el más popular de los lenguajes de script.

Para la implementación del sistema se utilizará Javascript como principal lenguaje, por ser libre, por sus características como lenguaje que soporta objetos y por las grandes potencialidades que brinda para manejar los eventos en el entorno web.

### **1.14 Tecnologías de desarrollo web**

Tecnología es el conjunto de conocimientos y técnicas, ordenados científicamente, que permiten diseñar y crear bienes o servicios que facilitan la adaptación al medio y satisfacen las necesidades de las personas esto es, un proceso combinado de pensamiento y acción con la finalidad de crear soluciones útiles. En el desarrollo web la utilización de conocimientos técnicos facilita el desarrollo de aplicaciones, cumpliendo con las necesidades existentes (40).

#### **1.14.1 Sistema de intercambio de datos**

Los sistemas gestores de base de datos son un conjunto de programas que permiten crear y mantener una base de datos, asegurando su integridad y confidencialidad aunque pueden provocar un incremento en el consumo de recursos de hardware y software, lo cual traería problemas para máquinas con pocas prestaciones. Una posible alternativa que se analiza en esta investigación son los formatos de intercambio de datos JSON y XML, los que pueden ser utilizados como un mecanismo ligero de almacenamiento de datos. Sin embargo es imposible que el navegador interactúe con los ficheros JSON y XML sin utilizar la tecnología AJAX.

# Capítulo I

## Fundamentación Teórica

### **XML 1.0**

Extensible Markup Language o Lenguaje Extensible de Marcas, es un conjunto de reglas que sirven para definir etiquetas semánticas y organizar un documento. Además XML es un metalenguaje que posibilita diseñar tu propio lenguaje de etiquetas (41).

### **JSON**

JSON (Javascript Object Notation) es un formato ligero de intercambio de datos utilizado comúnmente para el intercambio de información entre distintos sistemas. JSON es un formato de texto que es completamente independiente del lenguaje pero utiliza convenciones que son ampliamente conocidos por los programadores de la familia de lenguajes C, incluyendo C, C++, C#, Java, Javascript, Perl, Python, y muchos otros. Estas propiedades hacen que JSON sea un lenguaje ideal para el intercambio de datos (42).

### **1.14.2 AJAX**

AJAX, acrónimo de Asynchronous Javascript And XML (Javascript asíncrono y XML), es una técnica de desarrollo web para crear aplicaciones interactivas o Rich Internet Applications (RIA). Estas se ejecutan en el cliente, es decir, en el navegador de los usuarios y mantienen comunicación asíncrona con el servidor. De esta forma es posible realizar cambios sobre la misma página sin necesidad de recargarla. Esto significa aumentar la interactividad, velocidad y usabilidad de la misma.

AJAX es una combinación de cuatro tecnologías ya existentes:

- XHTML (o HTML) y hojas de estilos en cascada (CSS) para el diseño que acompaña a la información.
- Document Object Model (DOM) accedido con un lenguaje de scripting por parte del usuario, especialmente implementaciones ECMAScript como Javascript y JScript, para mostrar e interactuar dinámicamente con la información presentada.
- El objeto XMLHttpRequest para intercambiar datos asincrónicamente con el servidor web.



# Capítulo I

## Fundamentación Teórica

- XML es el formato usado comúnmente para la transferencia devuelta por el servidor, aunque cualquier formato puede funcionar, incluyendo HTML preformateado, texto plano, JSON y también EBML (43).

Teniendo en cuenta las características expuestas entre JSON y XML, se decide seleccionar a JSON. Primeramente los datos en formato JSON ocupan mucho menos espacio que en XML. Además cuando se obtiene una respuesta en formato XML se debe recorrer todo el XML, acción por lo general engorrosa y que requiere continuas llamadas recursivas incrementando la utilización de recursos de hardware. JSON representa mejor la estructura de los datos y requiere menos codificación y procesamiento.

### 1.14.3 Framework de desarrollo

Un framework define una arquitectura adaptada a las particularidades de un determinado dominio de aplicación, definiendo de forma abstracta una serie de componentes y sus interfaces, y estableciendo las reglas y mecanismos de interacción entre ellos (44). Estos pueden incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto. El objetivo fundamental de los framework es facilitar el desarrollo de software, permitiendo así minimizar el tiempo que se dedica para su desarrollo.

#### Prototype 1.6.1

Prototype es un framework de Javascript construido para elaborar de manera fácil aplicaciones web dinámicas. Prototype es una herramienta que brinda muchas funcionalidades a las aplicaciones web que lo utilizan. Simplifica gran parte del trabajo cuando se pretende desarrollar páginas altamente interactivas. (45).

#### Script.aculo.us 1.8.1

Script.aculo.us es una librería Javascript basada en Prototype que agrega efectos visuales dinámicos y una interface para elementos a través de DOM (45).

# Capítulo I

## Fundamentación Teórica

Script.aculo.us, es una gran librería, pero tiene demasiadas opciones y efectos que no se usan con frecuencia. Para ello se encuentran otras librerías con las que se pueden hacer cosas muy similares en mucho menos espacio.

### **Dojo 1.4.1**

Es un framework de código abierto para el trabajo con DHTML escrito en Javascript. Esta librería se compone de un conjunto de librerías al estilo Java. Se centra en abstraer al desarrollador de las complejidades del DHTML y de las discrepancias existentes entre navegadores, que hacen que el código Javascript a utilizar sea diferente (46).

### **JQuery 1.4.4**

JQuery es una librería programada en Javascript que trabaja con los elementos del DOM, maneja eventos, y crea animaciones sencillas. Además de esto permite ejecutar funciones sin necesidad de recargar la página, manipula las hojas de estilos, proporciona un conjunto de funciones para realizar animaciones con muy pocas líneas de código, se pueden añadir extensiones para aumentar su funcionalidad. JQuery es la librería que se ha presentado con más fuerza como alternativa a Prototype. JQuery comparte con Prototype muchas ideas e incluso dispone de funciones con el mismo nombre. Sin embargo, su diseño interno tiene algunas diferencias drásticas respecto a Prototype, sobre todo el "encadenamiento" de llamadas a métodos.

La librería JQuery en resumen aporta las siguientes ventajas (47):

- Ahorra muchas líneas de código.
- Hace transparente el soporte de nuestra aplicación para los navegadores principales.
- Provee de un mecanismo para la captura de eventos.
- Provee un conjunto de funciones para animar el contenido de la página de forma muy sencilla.

# Capítulo I

## Fundamentación Teórica

- Integra funciones para trabajar directamente con AJAX, por lo que como ya se ha dicho, permite realizar acciones dentro de un documento web sin la necesidad de recargar nuevamente toda la página.

Se utilizará JQuery como framework de desarrollo por su sistema expandible mediante plugins entre los que se encuentran el JQuery UI encargado de la implementación de elementos visuales comúnmente utilizados en aplicaciones web, por la estabilidad característica que acompaña a JQuery en todos los navegadores, por su popularidad entre los desarrolladores web de todo el mundo, y un grado de penetración en el mercado muy amplio, lo que hace suponer que es la mejor opción. Además es un producto serio, estable, bien documentado y con un gran equipo de desarrolladores a cargo de la mejora y actualización del mismo.

### **1.15 Conclusiones**

Como metodología de desarrollo se utilizará RUP que brinda una guía completa y apropiada para este tipo de aplicaciones.

La selección de Javascript como lenguaje de script y Aptana como IDE de desarrollo, facilitará en gran medida el desarrollo del producto.

AJAX como tecnología web aportará intercambio de información entre la aplicación y ficheros JSON.

## Capítulo 2: Propuesta de arquitectura

### 2.1 Introducción

La descripción de la arquitectura es una representación de alto nivel de la estructura de un sistema o aplicación que describe los componentes que lo integran, interacciones entre ellos, patrones que supervisan su composición así como las restricciones para aplicar dichos patrones (48).

En el presente capítulo se abordará la descripción de la arquitectura del producto “Mis Mejores Cuentos”, se detallarán los aspectos significativos para la arquitectura de este producto así como la organización de todos los componentes que forman parte de la misma.

### 2.2 Descripción de la arquitectura

#### 2.2.3 Propósito

Ofrecer una visión de la arquitectura del producto “Mis Mejores Cuentos” con el objetivo de representar las principales decisiones arquitectónicas dentro del sistema y explicar detalladamente la estructura que tendrá la aplicación, proporcionando una descripción arquitectónica comprensiva del sistema que servirá de guía (a partir de las diferentes vistas) para posteriores iteraciones.

#### 2.2.4 Estructura del equipo de desarrollo

El equipo de desarrollo estará conformado por

Miembros
Gerente de proyecto
Analistas
Desarrolladores
Grupo de calidad
Arquitecto
Arquitecto de Información
Administrador de Configuración y Gestión de cambio

# Capítulo II

## Propuesta de arquitectura

**Tabla 1. Equipo de Desarrollo**

### **Configuración de los puestos de trabajo por roles**

#### **Analista**

1. PC, con mouse y teclado
2. Instalación de Visual Paradigm 6.4
3. Instalación del paquete OpenOffice 3.3

#### **Desarrollador**

1. PC, con mouse y teclado
2. Instalación del IDE Aptana y Servidor web
3. Instalación del paquete OpenOffice 3.3

### **2.2.5 Estructura del sistema**

La aplicación estará organizada por carpetas según los tipos de archivos. El directorio de la aplicación se organizará de la siguiente forma: las páginas HTML estarán en una carpeta con nombre HTML, las hojas de estilos en cascada estarán en una carpeta con nombre CSS, los archivos y librerías Javascript en una carpeta con nombre JS, las medias se encontrarán en una carpeta con nombre MEDIA, en donde estará contenida una carpeta con nombre IMAGENES para los ficheros que sean imágenes y otra carpeta con nombre SONIDO para los que sean del tipo audio. Además existirá una carpeta con nombre DATOS, que contendrá los archivos JSON que almacenarán los datos que manejará el sistema.

## Propuesta de arquitectura

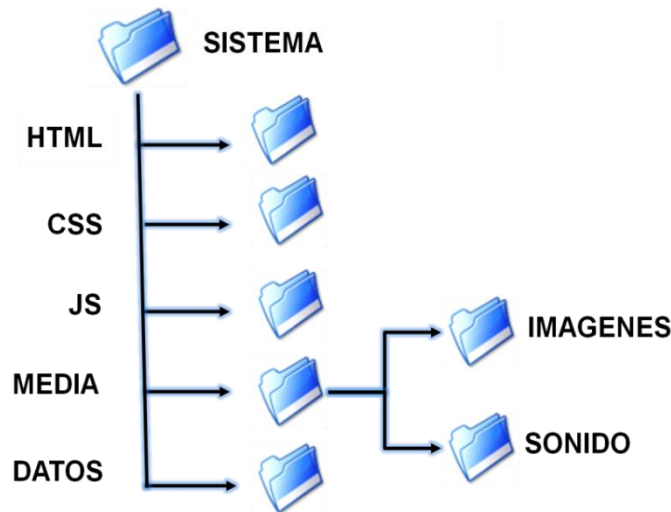


Figura 6. Estructura del sistema

### 2.3 Estándares de codificación

Los estándares de codificación ayudan a asegurar que el código tenga una alta calidad, menos errores, y pueda ser mantenido fácilmente. Están encaminados a solucionar los problemas con la legibilidad, predictibilidad y operaciones comunes bien conocidas en un nuevo ambiente. Regulan la nomenclatura en la cual se escriben, se diseñan y desarrollan los sistemas.

- La indentación debe estar compuesta por 4 espacios. Las tabulaciones no están permitidas.
- Los nombres de clases deben contener sólo caracteres alfanuméricos. Los números están permitidos en los nombres de clase, pero desaconsejados en la mayoría de los casos. El nombre de todas las clases estará antecedido del prefijo "js", Ej: "jsGame". Si el nombre de una clase está compuesto por más de una palabra, la primera letra de cada palabra debe aparecer en mayúsculas.

Ej: "jsPaintGame".

- Cualquier archivo que contenga código Javascript debe terminar con la extensión ".js", con la excepción de las páginas HTML. Los siguientes ejemplos muestran nombres de archivo admisibles.

`Class/Game.js`

`Class/Tool/Application.js`

## Propuesta de arquitectura

- Los nombres de archivo deben apuntar a nombres de clases como se describe arriba. Para cualquier otro archivo, sólo están permitidos caracteres alfanuméricos y guión bajo (\_). Los espacios en blanco están estrictamente prohibidos. Ej: "fichero\_json.json" está permitido.
- Los nombres de funciones deben contener únicamente caracteres alfanuméricos. Los guiones bajos (\_) no están permitidos. Los números están permitidos en los nombres de función pero no se aconseja en la mayoría de los casos. Deben empezar siempre con una letra minúscula. Cuando un nombre de función consiste en más de una palabra, la primera letra de cada nueva palabra debe estar en mayúsculas. Esto es llamado comúnmente como formato "camelCase". Por norma general, se recomienda la elocuencia para describir su propósito y comportamiento. Estos son ejemplos de nombres de funciones admisibles:

```
checkVersion()
```

```
paint()
```

```
stopNarrationSequence()
```

- Los métodos de acceso para las instancias, variables estáticas o atributos deben ir antepuestos con un "get" o un "set". Para el caso en que los métodos son declarados con el modificador "private" o "protected", el primer carácter debe ser un guión bajo (\_). Este es el único uso admisible de una barra baja en un nombre de método. Los métodos declarados como públicos no deberían contener nunca un guión bajo. Las funciones de alcance global (también llamadas "funciones flotantes") están permitidas pero desaconsejadas en la mayoría de los casos.
- Los nombres de variables deben contener caracteres alfanuméricos. Los guiones bajos (\_) no están permitidos. Los números están permitidos en los nombres de variable pero no se aconseja en la mayoría de los casos. Para las variables de instancia que son declaradas con el modificador "private" o "protected", el primer carácter de la variable debe ser un único guión bajo (\_). Este es el único caso admisible de un guión bajo en el nombre de una variable. Las variables declaradas como "public" no pueden empezar nunca con un guión bajo. Al igual que los nombres de funciones, los nombres de variables deben empezar siempre con una letra en minúscula y seguir la convención "camelCase".

(Ver Anexo 1)

# Capítulo II

## Propuesta de arquitectura

### 2.4 Objetivos y Restricciones arquitectónicas

Los objetivos y restricciones arquitectónicas vienen dados en gran medida por los requisitos no funcionales del sistema, que no son más que las propiedades o cualidades que el producto debe tener.

#### 2.4.1 Requerimientos No Funcionales

##### RNF 1 Requerimiento de software

Se requiere para el funcionamiento del sistema disponer de un servidor con el sistema operativo Linux y el servidor web Apache 2.2 instalado. Los usuarios del sistema deberán contar con un navegador o browser.

##### RNF 2 Requerimiento de hardware:

###### Estaciones de trabajo o PC Clientes

Periféricos: mouse, teclado

Tarjeta de red

Memoria RAM de 128 Mb (Mínimo)

###### Servidor de aplicaciones

Periféricos: mouse, teclado

Tarjeta de Red

512 MB de RAM (Mínimo)

##### RNF 3 Requerimiento de usabilidad

- El sistema deberá permitir el acceso a cualquier usuario. Los usuarios podrán acceder libremente al sistema.
- La información contenida en el sistema debe ser inédita o debe estar libre de derecho de autor.



# Capítulo II

## Propuesta de arquitectura

### **RNF 4 Requerimiento de soporte**

- El sistema será probado, instalado y configurado por los especialistas, que también se ocuparán de su mantenimiento.

### **RNF 5 Requerimiento de apariencia o interfaz externa**

- El diseño de la interfaz del sistema deberá ser sencillo, no estará sobrecargado de contenido, pero debe tener elementos visuales que capten la atención y que de forma metafórica representen una idea visual, para lograr un mayor entendimiento a los usuarios finales a los que va dirigido este producto niños entre 3 y 5 años.

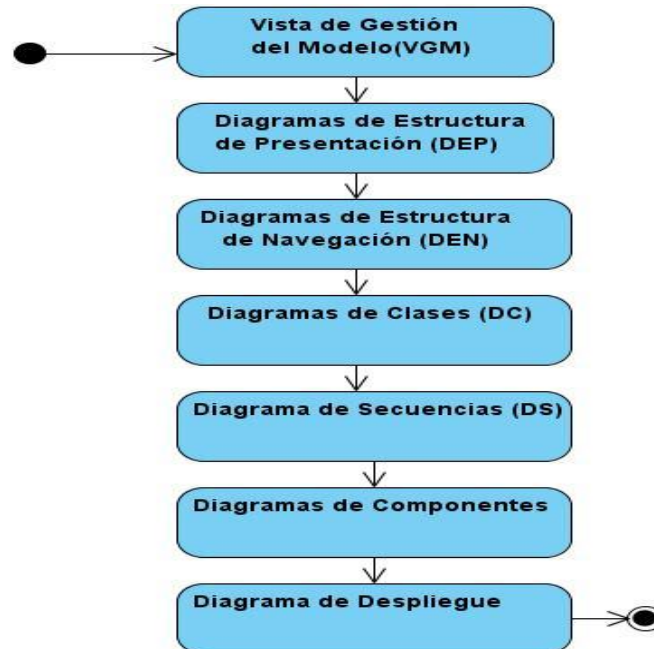
### **RNF 6 Restricciones en el diseño y la implementación**

- El sistema será implementado con el lenguaje de programación JavaScript haciendo uso de la librería JQuery.
- Se utilizarán como herramienta de modelado Visual Paradigm, y como IDE de desarrollo Aptana
- Las imágenes serán .jpeg o .gif.

### **2.5 Representación arquitectónica del producto**

En la siguiente figura se muestra la propuesta de desarrollo a seguir para la modelación de una aplicación educativa aplicando los diagramas de ApEM-L.

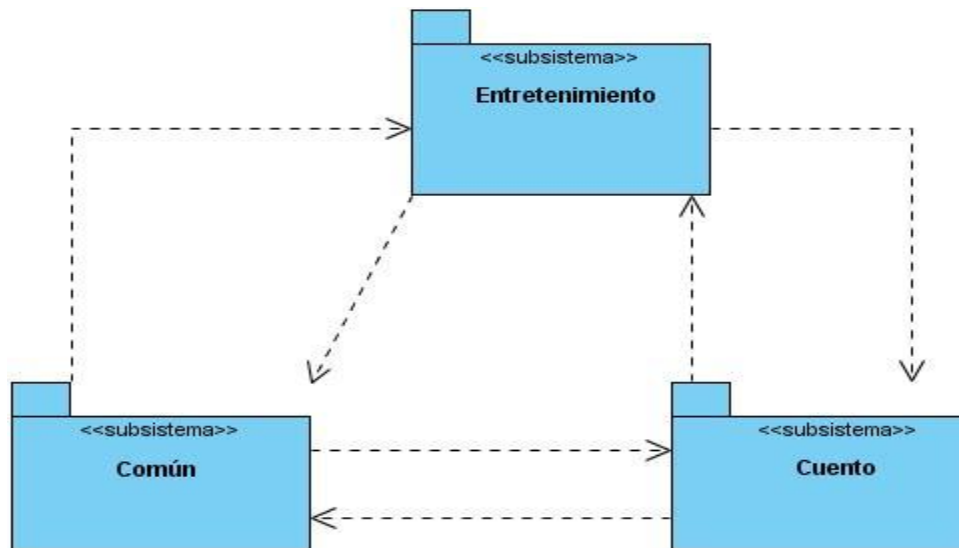
## Propuesta de arquitectura



**Figura 7. Propuesta de desarrollo de ApEM-L (49)**

El principio de organización que propone ApEM-L para la construcción de un sistema se basa en la división de la aplicación por subsistemas. Cada subsistema va a estar compuesto por una o más Vistas de Presentación. Las relaciones entre los subsistemas van a ser representados mediante la Vista de Gestión del Modelo (VGM) (18). Para la construcción del producto “Mis Mejores Cuentos” se han identificado 3 subsistemas arquitectónicamente significativos, que se muestran a continuación:

## Propuesta de arquitectura



**Figura 8. Vista de Gestión del Modelo**

A continuación se exponen cada uno de los subsistemas y las vistas que los componen, así como una breve explicación de la funcionalidad de estos.

**Cuento:** Es el módulo encargado de manejar los elementos más importantes de los cuentos, está compuesto por las siguientes vistas: Narración, Cuentos, Texto y Autor.

**Común:** Representa un conjunto de opciones comunes que permiten modelar de una manera más entendible el módulo y son de gran importancia a la hora de navegar por el mismo, está compuesto por la vista Común y la vista Idioma.

**Entretenimiento:** Constituido por los distintos tipos de juego que brinda la aplicación al usuario, está compuesto por las siguientes vistas: Entretenimiento, Dominó, Dibujo, Rompecabezas y Construcciones.

### 2.6 Sistema de vistas de ApEM-L

Tanto ApEM-L como UML; no establecen ninguna línea entre los diferentes conceptos y construcciones del lenguaje, pero por conveniencia del primero, este se ha dividido en varias vistas, modelando cada una

## Propuesta de arquitectura

de estas construcciones que representan un aspecto del sistema. La división ha sido sobre la base de las áreas conceptuales como se puede apreciar en el Anexo 2.

### 2.6.1 Vista de Presentación

La Vista Presentación ha sido incorporada completamente al lenguaje base UML, para permitir utilizar la semántica original de dicho lenguaje en la construcción de estructuras lógicas de presentación y navegación, incorporando un conjunto de estereotipos restrictivos y descriptivos para una mejor modelación, construyendo el diagrama de estructura de navegación (DEN) y el diagrama de estructura de presentación (DEP).

A continuación están descritos los diagramas arquitectónicamente significativos del sistema:

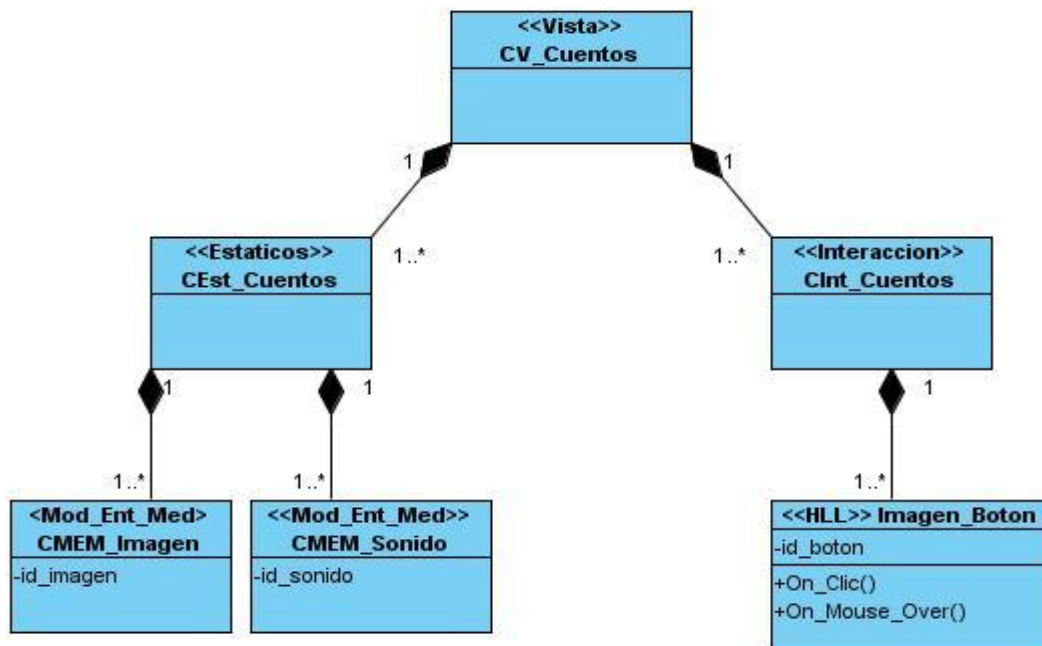


Figura 9. DEP Vista Cuento

Los Diagramas de Estructura de Presentación del resto de las vistas significativas pueden ser consultados en los Anexos 3 y 4.

## Propuesta de arquitectura

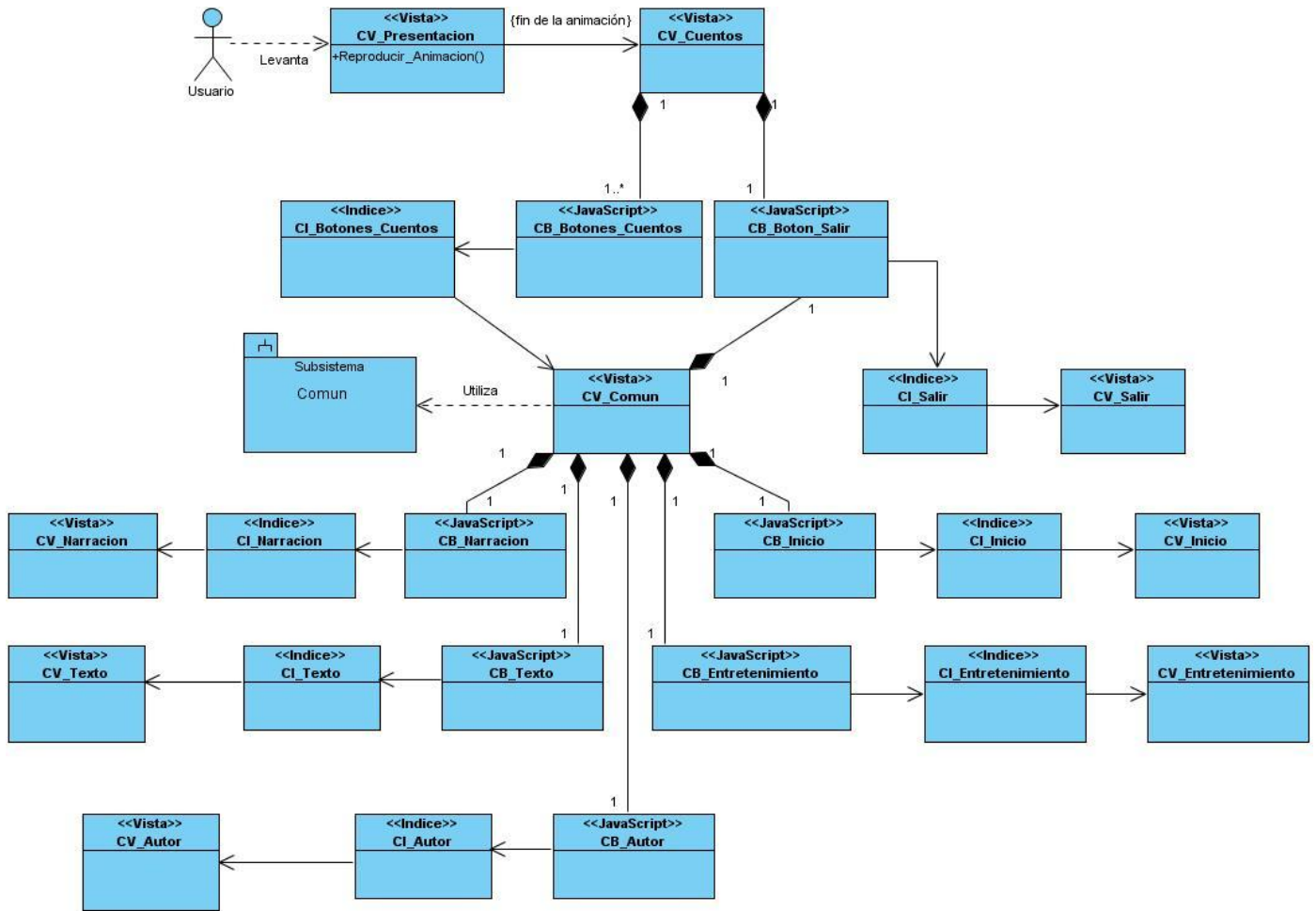


Figura 10. DEN del módulo Cuento

### 2.6.2 Vista Estática

La Vista Estática de ApEM-L está compuesta por el diagrama de clases, presentando algunas modificaciones para este tipo de diagrama, que consisten en dividirlo en dos grandes zonas, la de la izquierda dedicada al árbol jerárquico de las clases modelo entidad medias que representan los recursos mediáticos de la aplicación y en la zona de la derecha del diagrama, las clases que controlan la lógica del negocio de la aplicación propiamente dicha. La zona de la derecha se divide a su vez en 4 zonas, la primera dedicada a las clases vista, la contigua a esta y en el extremo superior derecho dedicada a las clases controladoras, inmediatamente debajo de esta sección, la destinada a las clases modelo, quedando

## Propuesta de arquitectura

una banda inferior derecha dedicada en su extremo derecho a las clases modelo entidad persistentes para el tratamiento de la información persistente de la aplicación; y en el extremo izquierdo las clases correspondientes al Lenguaje de Alto Nivel (HLL) (18). (Ver Anexo 5)

A continuación se muestran los diagramas de clases de las vistas más significativas del sistema.

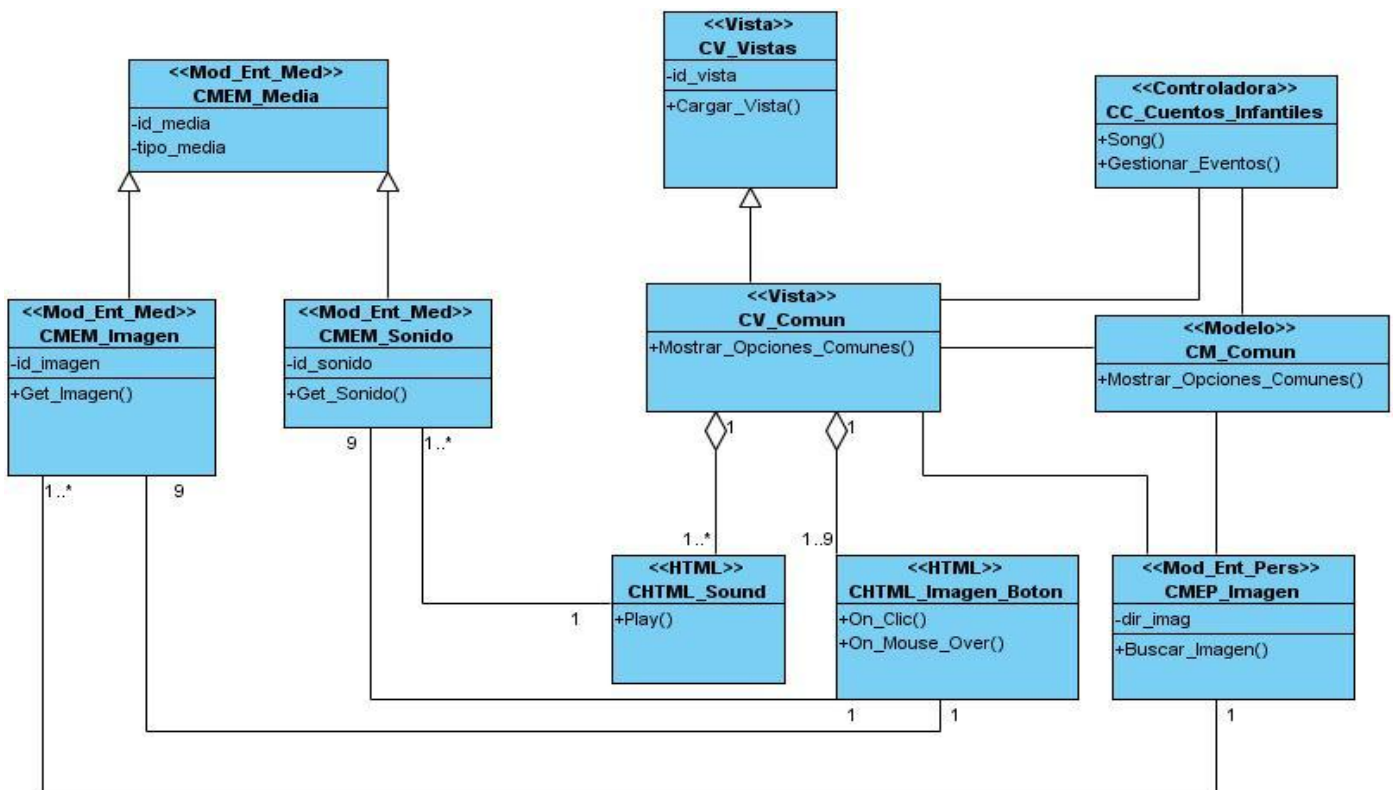


Figura 11. DC Vista Común

Los diagramas de clases del resto de las vistas pueden ser consultados en los Anexos 6 y 7.

### 2.6.3 Vista de arquitectura

La vista de arquitectura está compuesta por el diagrama de componentes y el diagrama de despliegue. El último de los mencionados no sufre cambios en ApEM-L, no así el de componentes donde se incorporan restricciones en los tipos de componentes. Al seguir la arquitectura propuesta por el patrón MVC-E,

## Propuesta de arquitectura

normalmente los componentes podrán ser organizados por paquetes, que identificarían unidades físicas de encapsulamiento del código (18).

### Diagrama de despliegue

Un diagrama de despliegue muestra las relaciones físicas entre los componentes hardware y software en el sistema. Es un conjunto de nodos unidos por conexiones de comunicación. Un nodo puede contener instancias de componentes software, objetos y procesos (50).

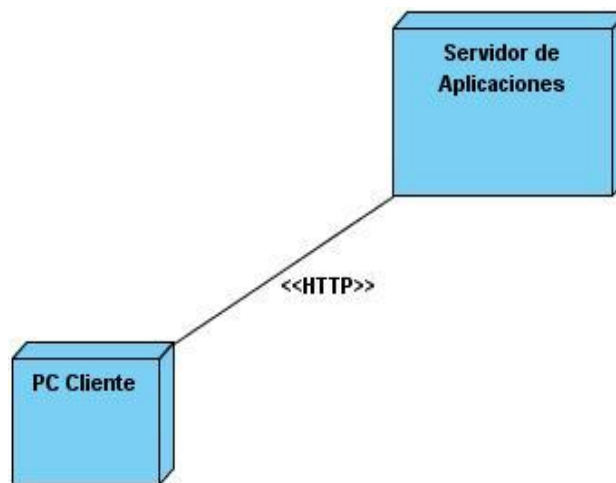


Figura 12. Diagrama de despliegue

**Nodo PC Cliente:** Este ordenador corresponde a la estación de los clientes. Las estaciones de los clientes deben poseer un navegador capaz de visualizar HTML.

**Nodo Servidor de Aplicaciones:** Nodo central que tiene instalado un servidor web y hospeda todos los componentes necesarios para el funcionamiento del producto.

### Diagrama de componentes

Los diagramas de componentes describen los elementos físicos del sistema y sus relaciones. Muestran las opciones de realización incluyendo código fuente, binario y ejecutable. Los componentes representan

## Propuesta de arquitectura

todos los tipos de elementos software que entran en la fabricación de aplicaciones informáticas. Pueden ser simples archivos, paquetes, bibliotecas cargadas dinámicamente (50).

ApEM-L extiende la semántica de UML para este tipo de diagrama al incorporar los elementos de organización en paquetes asociados al patrón arquitectónico MVC-E (Modelo – Vista – Controlador – Entidad) y sus relaciones de funcionamiento. (Ver Anexo 8)

### **2.7 Conclusiones**

En este capítulo se propuso un conjunto de estándares de codificación, que permite al producto “Mis Mejores Cuentos” contar con una mayor extensibilidad del software, mayor facilidad para hacer cambios, se podrá alcanzar un software reutilizable y con una mayor organización.

Se describieron los elementos arquitectónicamente significativos del producto, que facilita un punto de partida para un mejor entendimiento y escalabilidad de la aplicación.

La arquitectura obtenida como resultado de la propuesta presentada permite obtener un producto flexible y con alta reusabilidad de código.



## Capítulo 3: Evaluación de la arquitectura

### 3.1 Introducción

La calidad del software depende de los resultados de la evaluación de la arquitectura. Cuya función es determinante para la selección entre las distintas arquitecturas candidatas y la certificación de la factibilidad de la línea base, siendo el componente clave de las iteraciones arquitectónicas exitosas.

Evaluar una arquitectura de software sirve para prevenir todos los posibles problemas de un diseño que no cumple con los requerimientos de calidad y para saber que tan adecuada es la arquitectura de software definida para el sistema. De la evaluación de una arquitectura de software no se obtienen respuestas del tipo sí, no, bueno o malo, sino que explica cuáles son los puntos de riesgo del diseño evaluado, es decir, fortalezas y debilidades identificadas de la arquitectura de software.

En el presente capítulo se abordará la evaluación de la arquitectura propuesta como solución a la problemática existente en la presente investigación. Se describirán brevemente algunos de los métodos de evaluación de arquitectura, fundamentalmente características y pasos de aplicación de los mismos, así como los atributos de calidad en los que se centran estos. Finalmente se estimará el comportamiento del producto a desarrollar según los atributos de calidad que el mismo debe poseer.

### 3.2 Necesidad de evaluar la arquitectura

Uno de los factores que determina el éxito o fracaso de un sistema de software, es su arquitectura. El diseñar debidamente una arquitectura de software, garantiza que el sistema de software cumpla con uno o varios atributos de calidad. Si la arquitectura no se diseña de forma apropiada, el sistema de software resultante no logrará sus objetivos (51). Por lo regular, no se conoce sino hasta el final del desarrollo del sistema de software, si éste cumplió o no con los atributos de calidad que se especificaron en los requerimientos no funcionales. Dicho conocimiento tardío, implica tomar demasiados riesgos innecesarios, un ejemplo es, descubrir fallas en el sistema de software debido a que en la fase de diseño no se eligió apropiadamente una arquitectura. Para reducir tales riesgos, es recomendable llevar a cabo evaluaciones a la arquitectura.

## Evaluación de la arquitectura

### 3.3 Etapas en que se evalúa una arquitectura

La evaluación de la arquitectura se puede realizar en cualquier etapa del proceso de desarrollo, sobre todo cuando en el proyecto se comienzan a tomar decisiones que dependen de la arquitectura. Siempre se debe tener en cuenta que el costo de evaluar esa arquitectura debe ser menor que el costo de deshacer una decisión que dependa de dicha arquitectura. Existen dos variantes que agrupan todos los momentos en que se puede evaluar una arquitectura, estas son evaluación temprana y evaluación tardía (52).

**Evaluación temprana:** En esta variante no es necesario que la arquitectura esté completamente especificada para ser evaluada. Abarca desde las fases tempranas del desarrollo del proyecto hasta el final del mismo.

**Evaluación tardía:** En esta se establece la evaluación cuando la arquitectura está definida y el proyecto se encuentra implementado. A este nivel es muy importante la evaluación debido a que da una medida del cumplimiento de los atributos de calidad en el sistema y da una medida de cómo será su comportamiento.

### 3.4 Atributos de calidad

La calidad de software se define como el grado en el cual el software posee una combinación deseada de atributos. Tales atributos son requerimientos adicionales del sistema, que hacen referencia a características que éste debe satisfacer, diferentes a los requerimientos funcionales.

Estas características o atributos se conocen con el nombre de atributos de calidad, los cuales se definen como las propiedades de un servicio que presta el sistema a sus usuarios.

A grandes rasgos, Bass establece una clasificación de los atributos de calidad en dos categorías (53):

**Observables vía ejecución:** son atributos que se determinan del comportamiento del sistema en tiempo de ejecución. La descripción de algunos de estos atributos se presenta en la tabla 2.

## Evaluación de la arquitectura

Atributos de calidad	Descripción
Disponibilidad (Availability)	Es la medida de disponibilidad del sistema para el uso.
Confidencialidad (Confidentiality)	Es la ausencia de acceso no autorizado a la información.
Funcionalidad (Functionality)	Habilidad del sistema para realizar el trabajo para el cual fue concebido.
Desempeño (Performance)	<p>Es el grado en el cual un sistema o componente cumple con sus funciones designadas, dentro de ciertas restricciones dadas, como velocidad, exactitud o uso de memoria. Se refiere a capacidad de respuesta, ya sea el tiempo requerido para responder a aspectos específicos o el número de eventos procesados en un intervalo de tiempo.</p> <p>Se refiere además a la cantidad de comunicación e interacción existente entre los componentes del sistema.</p>
Confiabilidad (Reliability)	Es la medida de la habilidad de un sistema a mantenerse operativo a lo largo del tiempo.
Seguridad externa (Safety)	Ausencia de consecuencias catastróficas en el ambiente. Es la medida de ausencia de errores que generan pérdidas de información.
Seguridad interna (Security)	Es la medida de la habilidad del sistema para resistir a intentos de uso no autorizados y negación del servicio, mientras se sirve a usuarios legítimos.

## Evaluación de la arquitectura

**Tabla 2. Descripción de atributos de calidad observables vía ejecución**

**No observables vía ejecución:** son atributos que se establecen durante el desarrollo del sistema. La descripción de algunos de estos atributos se presenta en la tabla 3.

Atributos de calidad	Descripción
Configurabilidad (Configurability)	Posibilidad que se otorga a un usuario experto a realizar ciertos cambios al sistema.
Integrabilidad (Integrability)	Es la medida en que trabajan correctamente componentes del sistema que fueron desarrollados separadamente al ser integrados.
Integridad (Integrity)	Es la ausencia de alteraciones inapropiadas de la información.
Interoperabilidad (Interoperability)	Es la medida de la habilidad de que un grupo de partes del sistema trabajen con otro sistema. Es un tipo especial de integrabilidad.
Modificabilidad (Modifiability)	Es la habilidad de realizar cambios futuros al sistema.
Mantenibilidad (Maintainability)	Es la capacidad de someter a un sistema a reparaciones y evolución. Capacidad de modificar el sistema de manera rápida y a bajo costo.
Portabilidad (Portability)	Es la habilidad del sistema para ser ejecutado en diferentes ambientes de computación. Estos ambientes pueden ser hardware, software o una combinación de los dos.
Reusabilidad (Reusability)	Es la capacidad de diseñar un sistema de forma tal que su estructura o parte de sus componentes puedan ser reutilizados en futuras aplicaciones.
Escalabilidad	Es el grado con el que se pueden ampliar el diseño arquitectónico,

## Evaluación de la arquitectura

(Scalability)	de datos o procedimental.
Capacidad de Prueba (Testability)	Es la medida de la facilidad con la que el software, al ser sometido a una serie de pruebas, puede demostrar sus fallas. Es la probabilidad de que, asumiendo que tiene al menos una falla, el software fallará en su próxima ejecución de prueba.

**Tabla 3. Descripción de atributos de calidad no observables vía ejecución**

### 3.5 Técnicas de evaluación de arquitecturas

Las técnicas existentes en la actualidad para evaluar arquitecturas permiten hacer una evaluación cuantitativa sobre los atributos de calidad a nivel arquitectónico, pero se tienen pocos medios para predecir el máximo (o mínimo) teórico para las arquitecturas de software. Sin embargo, debido al costo de realizar este tipo de evaluación, en muchos casos los arquitectos de software evalúan cualitativamente, para decidir entre las alternativas de diseño. Bosch (54) propone diferentes técnicas de evaluación de arquitecturas de software: evaluación basada en escenarios, evaluación basada en simulación, evaluación basada en modelos matemáticos y evaluación basada en experiencia.

#### 3.5.1. Evaluación basada en escenarios

De acuerdo con Kazman (52) un escenario es una breve descripción de la interacción de alguno de los involucrados en el desarrollo del sistema con éste. Este consta de tres partes: el estímulo, el contexto y la respuesta.

**El estímulo:** es la parte del escenario que explica o describe lo que el involucrado en el desarrollo hace para iniciar la interacción con el sistema. Puede incluir ejecución de tareas, cambios en el sistema, ejecución de pruebas, configuración, etc.

**El contexto:** describe qué sucede en el sistema al momento del estímulo.

## Evaluación de la arquitectura

**La respuesta:** describe, a través de la arquitectura, cómo debería responder el sistema ante el estímulo. Este último elemento es el que permite establecer cuál es el atributo de calidad asociado. Los escenarios proveen un vehículo que permite concretar y entender atributos de calidad. Entre las ventajas de su uso están:

- Son simples de crear y entender.
- Son poco costosos y no requieren mucho entrenamiento.
- Son efectivos.

Actualmente las técnicas basadas en escenarios cuentan con dos instrumentos de evaluación relevantes: el Utility Tree propuesto por Kazman (52), y los Profiles, propuestos por Bosch (54).

### Utility Tree

Es un esquema en forma de árbol que presenta los atributos de calidad de un sistema de software, refinados hasta el establecimiento de escenarios que especifican con suficiente detalle el nivel de prioridad de cada uno. La intención del uso del Utility Tree es la identificación de los atributos de calidad más importantes para un proyecto particular. El Utility Tree contiene como nodo raíz la utilidad general del sistema. Los atributos de calidad asociados al mismo conforman el segundo nivel del árbol los cuales se refinan hasta la obtención de un escenario lo suficientemente concreto para ser analizado y otorgarle prioridad a cada atributo considerado. Cada atributo de calidad perteneciente al árbol contiene una serie de escenarios relacionados, y una escala de importancia y dificultad asociada, que será útil para efectos de la evaluación de la arquitectura.

### Perfiles (profiles)

Un perfil es un conjunto de escenarios, generalmente con alguna importancia relativa asociada a cada uno de ellos. El uso de perfiles permite hacer especificaciones más precisas del requerimiento para un atributo de calidad. Cada atributo de calidad tiene un perfil asociado. Algunos perfiles pueden ser usados para evaluar más de un atributo de calidad.

## Evaluación de la arquitectura

### 3.5.2. Evaluación basada en simulación

En el ámbito de las simulaciones, se encuentra la técnica de implementación de prototipos (prototyping). Esta técnica implementa una parte de la arquitectura de software y la ejecuta en el contexto del sistema. Es utilizada para evaluar requerimientos de calidad operacional, como desempeño y confiabilidad. Para su uso se necesita mayor información sobre el desarrollo y disponibilidad del hardware, y otras partes que constituyen el contexto del sistema de software. Se obtiene un resultado de evaluación con mayor exactitud (54). La evaluación basada en simulación utiliza una implementación de alto nivel de la arquitectura de software. La finalidad es evaluar el comportamiento de la arquitectura bajo diversas circunstancias. El proceso de evaluación basada en simulación sigue los siguientes pasos (54):

- Definición e implementación del contexto.
- Implementación de los componentes arquitectónicos.
- Implementación del perfil.
- Simulación del sistema e inicio del perfil.
- Predicción de atributos de calidad.

La exactitud de los resultados de la evaluación depende, a su vez, de la exactitud del perfil utilizado para evaluar el atributo de calidad y de la precisión con la que el contexto del sistema simula las condiciones del mundo real.

### 3.5.3. Evaluación basada en modelos matemáticos

Bosch establece que la evaluación basada en modelos matemáticos se utiliza para evaluar atributos de calidad operacionales. Permite una evaluación estática de los modelos de diseño arquitectónico, y se presentan como alternativa a la simulación, dado que evalúan el mismo tipo de atributos. Ambos enfoques pueden ser combinados, utilizando los resultados de uno como entrada para el otro. Los siguientes pasos son los que sigue esta técnica para la evaluación:

- Selección y adaptación del modelo matemático.
- Representación de la arquitectura en términos del modelo.
- Estimación de los datos de entrada requeridos.

## Evaluación de la arquitectura

- Predicción de atributos de calidad.

### 3.5.4. Evaluación basada en experiencia

Bosch establece que en muchas ocasiones los arquitectos e ingenieros de software otorgan valiosas ideas que resultan de utilidad para la evasión de decisiones erradas de diseño. Aunque todas estas experiencias se basan en evidencia anecdótica; es decir, basada en factores subjetivos como la intuición y la experiencia. Sin embargo, la mayoría de ellas puede ser justificada por una línea lógica de razonamiento, y puede ser la base de otros enfoques de evaluación (54). Existen dos tipos de evaluación basada en experiencia: la evaluación informal, que es realizada por los arquitectos de software durante el proceso de diseño, y la realizada por equipos externos de evaluación de arquitecturas.

### 3.6 Métodos de evaluación de arquitecturas

Hasta hace poco no existían métodos de utilidad general para evaluar arquitecturas de software. Si alguno existía, sus enfoques eran incompletos y no repetibles, lo que no brindaba mucha confianza. En virtud de esto, múltiples métodos de evaluación han sido propuestos (52). A continuación se explican algunos de los más importantes.

**SAAM** (Software Architecture Analysis Method) fue el primer método de evaluación basado en escenarios que surgió. Originalmente creado para el análisis de la modificabilidad de una arquitectura, pero en la práctica ha demostrado ser muy útil para evaluar de forma rápida distintos atributos de calidad, tales como modificabilidad, portabilidad, escalabilidad e integrabilidad.

Este método de evaluación se enfoca en la enumeración de un conjunto de escenarios que representan los cambios probables a los que estará sometido el sistema en el futuro. Como entrada principal, es necesaria alguna forma de descripción de la arquitectura a ser evaluada. Las salidas de la evaluación del método SAAM son las siguientes (52):

- Una proyección sobre la arquitectura de los escenarios que representan los cambios posibles ante los que puede estar expuesto el sistema.



# Evaluación de la arquitectura

- Entendimiento de la funcionalidad del sistema, e incluso una comparación de múltiples arquitecturas con respecto al nivel de funcionalidad que cada una soporta sin modificación.

SAAM puede ser utilizado para evaluar una o múltiples arquitecturas. Si se comparan dos o más se culmina el análisis con una tabla indicando las fortalezas y debilidades de cada una en cada escenario. Si se evalúa una sola, se culmina con un reporte señalando de los componentes computacionales donde la arquitectura no alcanza el nivel requerido. En ningún caso se emite un valor absoluto acerca de la “calidad arquitectónica”.

### **Architecture Trade-off Analysis Method (ATAM)**

Según Kazman el Método de Análisis de Acuerdos de Arquitectura (Architecture Trade-off Analysis Method, ATAM) está inspirado en tres áreas distintas: los estilos arquitectónicos, el análisis de atributos de calidad y el método de evaluación SAAM, explicado anteriormente. El nombre del método ATAM surge del hecho de que revela la forma en que una arquitectura específica satisface ciertos atributos de calidad, y provee una visión de cómo los atributos de calidad interactúan con otros; esto es, los tipos de acuerdos que se establecen entre ellos.

El propósito de ATAM es evaluar las consecuencias de las decisiones arquitectónicas sobre los atributos de calidad necesarios. Es un método de identificación de riesgos, o sea detecta las áreas de riesgos potenciales en la arquitectura de un sistema. Puede hacerse en una fase temprana en el ciclo de vida del desarrollo de software. Evalúa de una forma temprana los artefactos del diseño arquitectónico. No es necesario el análisis detallado sobre los atributos de calidad, sino una identificación de tendencias; no se trata de predecir con precisión el comportamiento de un atributo de calidad, ya que es imposible en una etapa temprana del diseño tener suficiente información. El interés está en conocer donde un atributo de interés es afectado por las decisiones del diseño arquitectónico.

Por tanto lo que se pretende hacer con ATAM a demás de mejorar la documentación, es registrar los posibles riesgos, los no riesgos, los puntos de sensibilidad y los puntos de desventajas que encontramos en el análisis de la arquitectura.

## Evaluación de la arquitectura

**Riesgos:** las decisiones de arquitectura que podría crear problemas en el futuro para algunos atributos de calidad.

**No riesgos:** las decisiones arquitectónicas que sean adecuadas al atributo de calidad que afectan.

**Desventaja:** las decisiones de arquitectura que tienen un efecto en más de un atributo de calidad.

**Puntos de sensibilidad:** una propiedad de uno o más componentes, y/o las relaciones entre componentes, fundamental para el logro de un determinado requisito de atributo de calidad.

El método de evaluación ATAM comprende nueve pasos, agrupados en cuatro fases que puede ser consultado en el Anexo 10.

### **ARID (Active Reviews for Intermediate Design)**

Es un método de bajo costo y gran beneficio, el mismo es conveniente cuando se posee un diseño parcialmente completo que se desea evaluar. Es un método que permite en las fases tempranas del diseño realizar una evaluación cualitativa basada en escenarios que permite analizar las debilidades en cuanto a riesgos de la arquitectura. Es un híbrido que posee lo mejor del ATAM combinado con el método Active Design Review (ADR) este último utilizado para la evaluación de diseños detallados de unidades del software como los componentes o módulos. Las preguntas giran en torno a la calidad y completitud de la documentación y la suficiencia, el ajuste y la conveniencia de los servicios que provee el diseño propuesto.

Tanto ADR como ATAM proveen características útiles para el problema de la evaluación de diseños preliminares. En el caso de ADR, los involucrados reciben documentación detallada y completan cuestionarios, cada uno por separado. En el caso de ATAM, está orientado a la evaluación de toda una arquitectura. De ADR, resulta conveniente la fidelidad de las respuestas que se obtienen de los involucrados en el desarrollo. Así mismo, la idea del uso de escenarios generados por los involucrados con el sistema es tomada del ATAM. De la combinación de ambas filosofías surge ARID, para efecto de la evaluación temprana de los diseños de una arquitectura de software. El ARID propone dos etapas

## Evaluación de la arquitectura

Actividades Previas y Revisión. A continuación se presentan los pasos del método de evaluación ARID, con una breve descripción de cada uno:

<b>Fase 1 Actividades Previas</b>	
1. Identificación de los encargados de la revisión	Los encargados de la revisión son los ingenieros de software que se espera que usen el diseño, y todos los involucrados en el diseño. En este punto, converge el concepto de encargado de revisión de ADR e involucrado del ATAM.
2. Preparar el informe de diseño	El diseñador prepara un informe que explica el diseño. Se incluyen ejemplos del uso del mismo para la resolución de problemas reales. Esto permite al facilitador anticipar el tipo de preguntas posibles, así como identificar áreas en las que la presentación puede ser mejorada.
3. Preparar los escenarios base	El diseñador y el facilitador preparan un conjunto de escenarios base. De forma similar a los escenarios del ATAM y el SAAM, se diseñan para ilustrar el concepto de escenario, que pueden o no ser utilizados para efectos de la evaluación.
4. Preparar los materiales	Se reproducen los materiales preparados para ser presentados en la segunda fase. Se establece la reunión, y los involucrados son invitados.
<b>Fase 2 Revisión</b>	
5. Presentación del ARID	Se explica los pasos del ARID a los participantes
6. Presentación del diseño	El líder del equipo de diseño realiza una presentación, con ejemplos incluidos. Se propone evitar preguntas que conciernen a la implementación o argumentación, así como alternativas de diseño. El objetivo es verificar que

## Evaluación de la arquitectura

	el diseño es conveniente.
7. Lluvia de ideas y establecimiento de prioridad de escenarios	Se establece una sesión para la lluvia de ideas sobre los escenarios y el establecimiento de prioridad de escenarios. Los involucrados proponen escenarios a ser usados en el diseño para resolver problemas que esperan encontrar. Luego, los escenarios son sometidos a votación, y se utilizan los que resultan ganadores para hacer pruebas sobre el diseño.
8. Aplicación de los escenarios	<p>Comenzando con el escenario que contó con más votos, el facilitador solicita pseudo-código que utiliza el diseño para proveer el servicio, y el diseñador no debe ayudar en esta tarea. Este paso continúa hasta que ocurra alguno de los siguientes eventos:</p> <ul style="list-style-type: none"> <li>• Se agota el tiempo destinado a la revisión.</li> <li>• Se han estudiado los escenarios de mayor prioridad.</li> <li>• El grupo se siente satisfecho con la conclusión alcanzada.</li> </ul> <p>Puede suceder que el diseño presentado sea conveniente con la exitosa aplicación de los escenarios, o por el contrario, no conveniente cuando el grupo encuentra problemas o deficiencias.</p>
9. Resumen	Al final, el facilitador recuenta la lista de puntos tratados, pide opiniones de los participantes sobre la eficiencia del ejercicio de revisión, y agradece por su participación.

## Evaluación de la arquitectura

Tabla 4. Pasos del método de evaluación ARID

### 3.7 Evaluación de la arquitectura definida

Para la evaluación de la arquitectura de software del producto “Mis Mejores Cuentos” se selecciona como método de evaluación ARID, que utiliza la técnica de evaluación basada en escenarios con el instrumento perfiles, donde se realiza un análisis de los principales escenarios dentro de la arquitectura del sistema. Este método permite realizar evaluaciones a diseños parciales en etapas tempranas del desarrollo, además aprovecha las mejores características de otros métodos, ya que es un híbrido entre ATAM y ADR. Para realizar la evaluación se seleccionaron los atributos de calidad más importantes en aquellos escenarios de acuerdo a un perfil en específico.

Debido a que en la presente investigación el encargado de llevar a cabo la evaluación de la arquitectura propuesta será el integrante del grupo de arquitectura que definió la misma, no se siguen estrictamente los pasos propuestos por el método ARID. El arquitecto del sistema tiene un dominio completo del diseño arquitectónico del mismo, y presenta conocimientos del método a aplicar para la evaluación, por lo que no se realizó la Fase1, pasando directamente a aplicarse los tres últimos pasos de la Fase 2.

Atributos de calidad seleccionados: Desempeño, Mantenibilidad, Integridad, Portabilidad y Disponibilidad

A continuación se exponen los escenarios seleccionados por su interés dentro de la arquitectura del sistema.

Escenario #1	Comunicación entre las capas de presentación y la lógica de negocio.
Atributo	Desempeño
Perfil	Rendimiento
Estímulo	Realizar solicitud de datos de la capa de lógica de negocio.
Respuesta	Muestra los datos sin interferir con la visualización o el comportamiento de la página.

## Evaluación de la arquitectura

Relación Atributo-Escenario
Las decisiones arquitectónicas que garantizan una respuesta satisfactoria del sistema ante una petición de datos de la capa lógica del negocio es la selección de la tecnología AJAX para dichas solicitudes las cuales son ejecutadas en segundo plano por parte del navegador. De esta forma es posible realizar cambios sobre las páginas sin necesidad de recargarlas, lo que significa aumentar la interactividad, velocidad y usabilidad del sistema. Estas facilidades que brinda AJAX permiten que el sistema gane en rendimiento mejorando su desempeño

**Tabla 5. Escenario #1**

Escenario #2	El sistema puede cambiar alguna de sus partes fácilmente.
Atributo de calidad	Mantenibilidad
Perfil	Mantenimiento
Estímulo	Modificar algunas de las partes del sistema.
Respuesta	Cambio de partes del sistema fácilmente.
Relación Atributo-Escenario	
Gracias a la implementación del patrón arquitectónico MVC-E se consigue un mantenimiento más sencillo del sistema, puesto que acciones como sustituir las vistas, agregar nuevas funcionalidades al controlador o cambiar los datos que almacena el sistema, no afecta a los elementos externos al cambio.	

**Tabla 6. Escenario #2**

Escenario #3	Los componentes del sistema que fueron desarrollados por separado deben de trabajar correctamente.
Atributo de calidad	Integrabilidad
Perfil	Integrabilidad
Estímulo	Se inicia la integración de todos los componentes para comprobar su funcionamiento en el sistema.

## Evaluación de la arquitectura

Respuesta	El sistema se integra de forma correcta.
Relación Atributo-Escenario	
Se garantiza una estructura e interfaz común en los módulos del sistema acorde a los usuarios a los que va dirigido el producto.	

**Tabla 7. Escenario #3**

Escenario # 4	Cambio de sistema operativo.
Atributo de calidad	Portabilidad
Perfil	Portabilidad
Estímulo	Migración de la aplicación a otro sistema operativo.
Respuesta	Cambiar el sistema operativo.
Relación Atributo-Escenario	
El sistema utiliza el lenguaje estándar de marcado HTML, lenguaje Javascript y emplea JSON como formato de intercambio de datos, por lo que el producto podrá ser montado en cualquier plataforma deseada, y podrá accederse al mismo mediante cualquier navegador web que utilice el usuario, sin tener en cuenta el sistema operativo que éste emplee.	

**Tabla 8. Escenario #4**

Escenario # 5	El sistema debe estar disponible para su uso.
Atributo de calidad	Disponibilidad
Perfil	Disponibilidad
Estímulo	Solicitud de disponibilidad de la aplicación.
Respuesta	El sistema está disponible.
Relación Atributo-Escenario	
El sistema permitirá que otras estaciones se conecten, siempre y cuando utilicen el nombre predefinido en el servidor.	

## Evaluación de la arquitectura

### Tabla 9. Escenario #5

Hacer un adecuado balance de estos atributos entre otros, es esencial para obtener un sistema que cumpla las expectativas de las distintas partes interesadas. A partir de todos los elementos que se han reflejado, se puede afirmar que la solución permite cumplir con los atributos Rendimiento, Mantenibilidad, Integrabilidad, Portabilidad y Disponibilidad. Teniendo en cuenta lo expuesto anteriormente, se define de manera general que el diseño arquitectónico evaluado de manera parcial, dentro de una etapa temprana del desarrollo, resulta aceptable.

### 3.8 Conclusiones

En este capítulo se expusieron las principales técnicas y métodos usados para evaluar una arquitectura de software. Tras el estudio de los métodos de evaluación se eligió para su utilización el método ARID, permitiendo así, tener una evaluación de la arquitectura en una etapa temprana del desarrollo del sistema y mostrando cómo se puede analizar el cumplimiento de los atributos de calidad. De esta forma se concluye que la arquitectura de software propuesta cumple con los requerimientos no funcionales previstos para la aplicación.



## Conclusiones Generales

Con el propósito de darle cumplimiento al objetivo general y a la problemática planteada en el presente trabajo, se han llevado a cabo satisfactoriamente cada una de las tareas que fueron trazadas al comienzo del mismo.

- La selección de la metodología RUP junto con el lenguaje de modelado ApEM-L para la propuesta presentada es correcta
- La propuesta presentada constituye la organización estructural y funcional del software educativo “Mis Mejores Cuentos” basado en atributos de calidad logrando un producto multimedia sobre plataforma web con herramientas libres
- La evaluación de la propuesta siguiendo el método ARID permitió demostrar que la arquitectura es aceptable de acuerdo a los indicadores de calidad evaluados y permite aplicarse a productos con características similares

## Recomendaciones

Al concluir el presente trabajo se recomiendan los elementos siguientes:

- Refinar la arquitectura propuesta, al incluirse el módulo administración
- Utilizar la arquitectura propuesta en productos multimedia para plataformas web que utilicen herramientas libres con características similares a las mencionadas en este trabajo
- Utilizar la documentación generada como estándar o guía en el desarrollo de proyectos con características similares

## Referencias Bibliográficas

1. **Garlan, David y Shaw, Mary.** *An Introduction to Software Architecture.* Pittsburgh : Canegie-Mellon University, 1994.
2. **Dewayne E, Perry y L. Wolf, Alexander.** “*Foundations for the study of software architecture*”. s.l. : ACM SIGSOFT, 1994.
3. **Lawrence, Shari.** *Ingeniería de software :Teoría y Práctica.* Madrid : Prentice-Hall.
4. **C.B, Reynoso.** *Introducción a la Arquitectura de Software.* 2004.
5. **López, Henry Mastrapa.** *Propuesta de Arquitectura del Sistema de Registro de visitante en la UCI.* Ciudad Habana : s.n., 2008.
6. **IEEE.** IEEE Standars Association. [En línea] IEEE Std 1471-200, 12 de 2 de 2009. [Citado el: 25 de noviembre de 2010.] [http://standards.ieee.org/reading/ieee/std\\_public/description/se/1471-2000\\_desc.html](http://standards.ieee.org/reading/ieee/std_public/description/se/1471-2000_desc.html).
7. **Clements, Paul.** *A Survey of Architecture Description Languages. Proceedings of the International Workshop on Software Specification and Design.* Alemania : s.n., 1996.
8. **Bass, Len, Clements, Paul y Kazman, Rick.** *Software Architecture in Practice.* s.l. : Reading, Addison-Wesley, 1998.
9. **Garlan, David.** *Architecture: A Roadmap. En Anthony Finkelstein: The future engineering.* s.l. : ACM, 2000.
10. **Shaw, Mary y Clements, Paul.** *A field guide to Boxology: Preliminary classification of architectural styles for software systems*”. s.l. : Carnegie Mellon, 1996.
11. **Klein, Mark y Kazman, Rick.** *Attribute-based architectural styles.* s.l. : Carnegie Mellon University, 1999.
12. *Arquitectura y Patrones de diseño., Ingeniería de Software II.* Habana, Cuba.: eva.uci.cu, 2009-2010. Semana: 3, Actividad No.5, Conferencia #2, Tema No. 1.
13. **Reynoso, Carlos y Kicillof, Nicolás.** *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* UNIVERSIDAD DE BUENOS AIRES : s.n., 2004.

# Referencias Bibliográficas

14. **Asencio, Reinier y Hinojosa, Jeanders.** *Definición de la arquitectura del sistema alasARBOGEN 2.0.* Ciudad Habana : Universidad de las Ciencias Informáticas, 2010.
15. slideshare.net. [En línea] 2010. [Citado el: 10 de diciembre de 2010.] <http://www.slideshare.net/jcampo/cliente-servidor-307243>.
16. **Díaz, Luis.** *Propuesta de arquitectura para el sistema de gestión automatizado de recursos humanos GESTAPro.* Ciudad Habana : UCI, 2009.
17. **Burbeck, Steve.** "Application programming in Smalltalk-80: How to use Model-View-Controller (MVC)". [En línea] University of Illinois in Urbana-Champaign, Smalltalk Archive. [Citado el: 3 de diciembre de 2010.] <http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html..>
18. **Ciudad Ricardo, Febe Ángel.** *ApEM – L como una nueva solución a la modelación de aplicaciones educativas multimedias en la UCI.* Ciudad de La Habana : s.n., 2007.
19. **Gracia, Joaquin.** IngenierosSoftware. [En línea] 27 de mayo de 2005. [Citado el: 5 de febrero de 2011.] <http://www.ingenierossoftware.com/analisisydiseno/patrones-diseno.php..>
20. Ciclo de vida del software. [En línea] 1998. [Citado el: 19 de noviembre de 2010.] <http://html.rincondelvago.com/ciclo-de-vida-del-software.html>.
21. **Brito Acuña, Kareenny.** Metodologías tradicionales y metodologías ágiles. [En línea] [Citado el: 16 de noviembre de 2010.] <http://www.eumed.net/libros/2009c/584/Metodologias%20tradicionales%20y%20metodologias%20agiles.htm>.
22. **Sánchez Mendoza, María A.** *Sánchez, María A. Mendoza. 2004. Metodologías De Desarrollo De Software..* Perú : s.n., 2004.
23. **Jacobson, Ivar, Booch, Grady y Rumbaugh, James.** *El Proceso Unificado de Desarrollo de Software.* La Habana : Félix Varela, 2004.
24. **Callaguazo Vega, Wilson Ramiro y Torres Núñez, Edwin Leonardo.** *El Lenguaje Unificado de Modelado (UML) es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software. Captura decisiones y conocimientos sobre los sistemas que se deben construir.* 2008.

# Referencias Bibliográficas

25. **Janqui Guzmán, Hermógenes.** Introducción al UML el Lenguaje Unificado del Modelado". [En línea] 2003. [Citado el: 15 de diciembre de 2010.] [http://usuarios.multimania.es/admi/uml/uml.htm#\\_Toc50913495](http://usuarios.multimania.es/admi/uml/uml.htm#_Toc50913495).
26. **Matos Pérez, Sulay.** *Análisis y diseño del módulo ejercicios de la colección Multisaber.* Ciudad Habana : Universidad de las Ciencias Informáticas, 2008.
27. **Gómez Landeros, Ruth Priscila.** *Herramientas Case.* s.l. : Universidad Veracruzana, 2007.
28. **Menendez, Rosa.** Rational Software Corporation. [En línea] 2007. [Citado el: 9 de enero de 2011.] <http://www.usmp.edu.pe/publicaciones/boletin/fia/info36/proyectos.html..>
29. Paradigm, Visual. 2007. Visual Paradigm for UML (ME) - (Paradigma Visual para UML (ME)) (Visual Paradigm for UML (ME)) 6.0. [En línea] 2007. [Citado el: 12 de noviembre de 2010.] [http://www.freedownloadmanager.org/es/downloads/Paradigma\\_Visual\\_para\\_UML\\_\(M%C3%8D\)\\_14720\\_p/](http://www.freedownloadmanager.org/es/downloads/Paradigma_Visual_para_UML_(M%C3%8D)_14720_p/).
30. **Carlos Leopoldo.** 7 sistemas de control de versiones Open Source. [En línea] 19 de septiembre de 2008. [Citado el: 16 de diciembre de 2010.] <http://techtastico.com/post/7-sistemas-control-versiones/>.
31. **Ramírez, Alejandro.** Ramírez, Alejandro. 2004. Sistema de Control de Versiones. [En línea] 22 de Marzo de 2004. [Citado el: 30 de Enero de 2009.]. *Sistema de Control de Versiones.* [En línea] 22 de marzo de 2004. [Citado el: 9 de Diciembre de 2010.] <http://polaris.dit.upm.es/~rubentb/docs/subversion/TutorialSubversion/index.html#N10047>.
32. Tigris.org.Open Source Software Engineering Tools. [En línea] 2001-2009. [Citado el: 24 de febrero de 2011.] <http://rapidsvn.tigris.org/>.
33. **García García, Virgilio.** Herramientas de desarrollo. [En línea] [Citado el: 15 de enero de 2011.] [http://petra.euitio.uniovi.es/~i1656157/doc/implementacion/Herramientas\\_de\\_desarrollo.pdf](http://petra.euitio.uniovi.es/~i1656157/doc/implementacion/Herramientas_de_desarrollo.pdf).
34. **Pérez Chirino, Yosbel.** *Herramienta para la gestión de torneos ajedrecísticos, versión 2.0.* La Habana : s.n., 2010.
35. **Santillana, Fabricio A.** *Entornos de programación Equiparación entre:Aptana Studio y HTMLPAD.* 2008. V. 2-718-1147.
36. **Sánchez, Emilio A., Letelier, Patricio y Conos, José H.** *Mejorando la gestión de Historias de Usuario en eXtreme programming.* Valencia : s.n.

# Referencias Bibliográficas

37. Una Introducción a APACHE. [En línea] 2010. [Citado el: 21 de enero de 2011.] [http://linux.ciberaula.com/articulo/linux\\_apache\\_intro/](http://linux.ciberaula.com/articulo/linux_apache_intro/).
38. **Mendoza, Alberto.** *Alberto Mendoza Garnache., Definición de la arquitectura de software del Grupo de Desarrollo para la Gestión de Equipos Médicos.* Ciudad Habana : s.n., 2009.
39. **Martinez, Ronaldo.** <http://www.programacion.com/>. [En línea] 2010. [Citado el: 14 de 01 de 2011.] [http://www.programacion.com/articulo/vbscript\\_54/2](http://www.programacion.com/articulo/vbscript_54/2).
40. Sí a la tecnología. [En línea] [Citado el: 26 de febrero de 2011.] <http://www.sialatecnologia.org/tecnologia.php>.
41. **Chávez García, Enrique.** Introducción a XML. [En línea] [Citado el: 16 de enero de 2010.] <http://www.maestrosdelweb.com/editorial/flashxml/>.
42. json.org. [En línea] [Citado el: 12 de enero de 2011.] <http://www.json.org/json-es.html>.
43. **La Red Martínez, David L y Valesani, Maria E.** Comparación de Frameworks de Desarrollo de Páginas Web con AJAX. [En línea] [Citado el: 13 de enero de 2011.] <http://www.sicuma.uma.es/sicuma/independientes/argentina08/LaRed-Eugenia/index.html>.
44. **Perera, Jose Raul.** *Aquitectura de software para sistema gestión de inventarios.* Ciudad Habana : Universidad de las Ciencias Informáticas, 2007.
45. **David Tavárez.** Comparación de Frameworks en Javascript. [En línea] [Citado el: 12 de enero de 2010.] <http://www.maestrosdelweb.com/editorial/comparacion-frameworks-javascript/>.
46. **Prieto, Julio César.** *Propuesta de arquitectura para el proyecto SIGESPRO.* Ciudad Habana : Universidad de las Ciencias Informáticas, 2009.
47. JavaScript Ya. [En línea] [Citado el: 16 de diciembre de 2010.] <http://www.javascriptya.com.ar/jquery/temarios/descripcion.php?cod=57&punto=1&inicio=0>.
48. **Moreno Navarro, Juan José.** Arquitecturas Software. [En línea] [Citado el: 26 de febrero de 2011.] [http://babel.ls.fi.upm.es/~fred/sbc/arquitecturas\\_sw.pdf](http://babel.ls.fi.upm.es/~fred/sbc/arquitecturas_sw.pdf).
49. **Ciudad Ricardo, Febe Ángel y Herrera Martínez, Yosnel.** *DISEÑO DE APLICACIONES EDUCATIVAS MULTIMEDIAS UTILIZANDO UNA NUEVA VERSIÓN DEL LENGUAJE DE MODELACIÓN ApEM – L.* 2009.

# Referencias Bibliográficas

50. Unified Modeling Language. Diagramas de UML. [En línea] 11 de agosto de 2009. [Citado el: 27 de febrero de 2011.] [http://www.gratisblog.com/uml833/i130373-modelado\\_de\\_implementacion.htm](http://www.gratisblog.com/uml833/i130373-modelado_de_implementacion.htm).
51. **Gómez, Omar**. EISEI. [En línea] [Citado el: 15 de abril de 2011.] <http://www.eisei.net.mx/>.
52. **Kazman, Rich, Clements, Paul y Mark, Klein**. *Evaluating Software Architectures: Methods and Case Studies*. s.l. : Addison Wesley, 2000.
53. **Bass, Len, Clements, Paul y Kazman, Rich**. *Software Architecture in practice*. s.l. : Addison-Wesley, 1998.
54. **Bosch, J**. *Design & Use of Software Architectures*. s.l. : Addison Wesley, 2000.