Universidad de las Ciencias Informáticas

Facultad 5

"Proceso de Diseño Arquitectónico para Software de Realidad Virtual"

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

Autor(as):

Odalys Juantorena Portuondo

Maricarmen Torres Concepción

Tutor(a):

Ing. Yusleidy Guelmes León

Ciudad de la Habana

Junio, 2009

"Todos y cada uno de nosotros paga puntualmente su cuota de sacrificio consciente de recibir el premio en la satisfacción del deber cumplido, conscientes de avanzar con todos hacia el Hombre Nuevo que se vislumbra en el horizonte."

Ernesto Guevara de la Serna

Datos de Contacto

Tutor(a):

Nombre y Apellidos: Yusleidy Guelmes León
Institución: UCI
Título: Ingeniera Informática
Categoría Docente: Asistente
Año de Graduación: 2005
E-mail: yguelmes@uci.cu
Actividades y cargos desempeñados: Profesora de la Facultad 5 de la asignatura Ingeniería de Software. Actualmente investiga en la temática de Arquitectura de Software y métodos para su evaluación. Ha tutorado varios Trabajos de diploma vinculados al Proceso de Desarrollo de Software y relacionadas con temas de Realidad Virtual.

Dedicatoria

A mis padres, por su infinito amor.

A mi abuela Blanca, por ser la guía de toda la familia y por ser especial para mí.

A mi toda mi familia materna, en especial a mi tía Mari por su apoyo incondicional durante estos 5 años.

Maricarmen

A mis padres por su apoyo incondicional.

A mi hermano por ser tan especial conmigo.

A toda mi familia y mis seres queridos por no dudar nunca que este día llegaría.

Odalys

Agradecimientos

De Maricarmen:

A la Revolución por permitirme realizar este sueño.

A la persona que me ha guiado por este camino, quien es además mi ejemplo y mi razón de existir, por ser incondicional y amiga en momentos difíciles, a la persona que más amo en el mundo: mi madre.

A mi papá y a Nora por su cariño y apoyo.

A Mima por su dedicación, amor y preocupación.

A mi tía Mari por ser mi segunda madre, por su cariño y por estar siempre cuando la necesito.

A toda mi familia materna por ser tan unida, por apoyarme en estos 5 años de carrera, por su amor y por haber sabido guiarme en la vida.

A Javy por ser la persona con la que comparto mi vida, por estar a mi lado cada día apoyándome, dándome fuerzas, por su amor, su bondad y su apoyo.

A mis suegros Violeta y Carlos por ser las personas mas maravillosas del mundo.

A todos los compañeros de trabajo de mi mamá, en especial a Elizabeth y Alina por su cariño y por ayudarme siempre en todo lo que necesito.

A mi compañera de tesis por estos largos meses de empeño y sacrificio, por haber aprendido a lidiar con mi carácter, por su preocupación y dedicación.

A mi tutora por toda la ayuda que nos ha brindado en la realización del trabajo, por contribuir a nuestra formación, por su paciencia y delicadeza.

A todas las personas que de una forma u otra han contribuido a perfeccionar este trabajo, por sus consejos y sugerencias, en especial al líder y la arquitecta del proyecto Laboratorios de Realidad Aumentada: Ernesto y Mileidis.

A mis amigas de la infancia: Ana Belen, Ana Elvy, Dayi y Sonya por ser especiales para mi y porque junto a ellas he pasado momentos inolvidables.

A todos mis amigos de la UCI, en especial a Marilin, Ale, Ana Ivis, Joannis y Aron, por ser incondicionales, por brindarme su amistad, su cariño, por ayudarme en momentos difíciles y porque junto a ellos he pasado los mejores momentos en la Universidad.

A todos los quiero infinitamente y gracias por hacer que cada uno de mis días sea mucho mas alegre.

De Odalys:

A mi mamá un agradecimiento muy especial por ser lo mas grande y lindo que tengo en la vida, por ser más que mi madre mi amiga, por su ternura, comprensión, su infinito amor y sobre todas las cosas, por la inmensa confianza que siempre ha depositado en mí.

A mi papá por ser el mejor padre del mundo, mi fuente de inspiración, para que siempre se sienta orgulloso de su niña ingeniera.

A mi hermano, por quererme tanto, por estar, por ser especial conmigo.

A mi abuela porque nunca dudó de mis capacidades para hacer realidad este gran sueño.

A mi tía la loca como la llamo cariñosamente, por ser mi segunda madre, por ayudarme siempre a seguir adelante, por sus consejos.

A toda mi familia por estar presentes en los momentos en que los he necesitado, gracias por el amor que siempre me han dado.

A Glenda y a su familia por quererme y compartir conmigo estos 5 años.

A Pablo y Ariagna, que más que amigos son mis hermanos.

A mi compañera de tesis, porque sin su esfuerzo y determinación me hubiese resultado imposible desarrollar este trabajo.

A mi tutora por su colaboración, por su ayuda y apoyo, sobre todo en los momentos en que creía que no lo lograría.

A todos los profesores que contribuyeron en mi desarrollo profesional y en la realización de este trabajo.

A todos mis compañeros de grupo y de universidad porque me hicieron sonreír y aportaron granitos de felicidad a mi estancia en esta escuela, nunca los olvidaré.

A todos los quiero mucho, gracias por existir.

Resumen

La presente investigación surge a partir de la necesidad de definir el Proceso de Diseño Arquitectónico para la construcción de Software de Realidad Virtual (RV) en la Facultad 5 de la Universidad de las Ciencias Informáticas (UCI), la misma tiene como propósito fundamental elevar la calidad del diseño de más alto nivel en estos proyectos.

Para elaborar la propuesta fue necesario realizar un estudio del Proceso de Diseño Arquitectónico planteado en diferentes Metodologías de Desarrollo de Software, tales como el Proceso Unificado para el desarrollo de software (RUP) y Microsoft Solution Framework Ágil (MSF Ágil). Se realizó además un análisis del proceso en los proyectos que pertenecen al polo de RV de la Facultad 5 a través de entrevistas a los Arquitectos de Software, Diseñadores y Programadores de estos proyectos, con el objetivo fundamental de encontrar las principales deficiencias en el proceso empleado actualmente y obtener sus opiniones para la perfección de un proceso futuro.

Posteriormente se elaboró el Proceso de Diseño Arquitectónico para Software de RV, el cual está basado en procesos ágiles. El mismo consta de tres actividades y un conjunto de prácticas asociadas, las cuales pueden ser aplicables según las características del proyecto. Estas prácticas se clasifican en: generales, específicas y de entrada al proceso.

Finalmente se utilizó la solución propuesta para diseñar la arquitectura del software Kinetic, el cual se desarrolla en el polo de RV de la Facultad 5, asegurando de esta forma la utilidad y confiabilidad del proceso.

Palabras clave: arquitectura de software, diseño arquitectónico, proceso, realidad virtual.

Índice

Introducción

El cambio tecnológico em el campo de la informática provoca que esta ciencia sea cada vez mucho más dinámica e introduce cambios que llevan al reemplazo de productos, procesos, diseños, técnicas, etc.

La desintegración del campo socialista de Europa del Este, la URSS y el cruel bloqueo impuesto por los gobiernos de Estados Unidos a Cuba provocó que la entrada de las nuevas tecnologías al país se viera frenada y que la existente se tornara caduca. El gobierno cubano ha invertido grandes recursos en el proceso de informatización de la sociedad para lograr revertir esta situación.

La Universidad de las Ciencias Informáticas es un programa de la Batalla de Ideas surgido en el año 2002, donde se forma el capital humano especializado, investigando y produciendo software y servicios informáticos para la sociedad cubana y para el mundo, cuya misión es producir software y servicios informáticos a partir de la vinculación estudio trabajo como modelo de formación. Consta de 10 facultades especializadas en distintos perfiles informáticos.

En la Facultad 5 se desarrollan numerosos proyectos de RV. Este término surgido recientemente encierra amplias aplicaciones que van encaminadas a simular objetos y procesos del mundo real. Entre sus principales aplicaciones se encuentran los simuladores: de auto, de vuelos, quirúrgicos, entre otros. Se puede mencionar también los videojuegos, los cuales empleados de forma racional y centrados en la formación, incluyen de igual manera una importante modalidad de la RV.

Estos proyectos son de gran importancia económica y social para el país, por lo que se hace necesario enfrentar una serie de deficiencias para que los productos finales posean la calidad necesaria.

En la actualidad existen muchas metodologías para el desarrollo de software, desde métodos muy *pesados* y burocráticos, métodos ajustables al proyecto y a las condiciones de desarrollo, hasta métodos *ligeros* que surgen como respuesta a los excesos formales de otros métodos en cuanto a planificación, documentación, jerarquía de roles, excesiva utilización de patrones, etc.

Están las llamadas metodologías tradicionales, las cuáles se caracterizan por estar basadas en normas provenientes de estándares seguidos por el entorno de desarrollo, ser un proceso más controlado, con un contrato prefijado, donde el cliente interactúa con el equipo de desarrollo mediante

reuniones. Estas se caracterizan además por poseer cierta resistencia a los cambios. En las metodologías tradicionales la Arquitectura de Software es esencial y se expresa mediante modelos.

Otro tipo de metodología son las conocidas como metodologías ágiles, las cuales están basadas en heurísticas provenientes de prácticas de producción de código, sus procesos son mucho menos controlados, no existe contrato tradicional o al menos es bastante flexible, el cliente es parte del equipo de desarrollo, se utilizan cuando el equipo de desarrollo es pequeño, puesto que poseen pocos roles y pocos artefactos. Las metodologías ágiles hacen menos énfasis en la Arquitectura de Software que las tradicionales.

El avance de la informática ha propiciado además el término Arquitectura de Software, el cual se refiere al diseño de más alto nivel de la estructura de un sistema, programa o aplicación que involucra los elementos más significativos del sistema y está influenciada entre otros factores por las plataformas, software, sistemas operativos, manejadores de bases de datos, protocolos, consideraciones de desarrollo y requisitos no funcionales.

El diseño arquitectónico es tratado de forma particular por cada una de las Metodologías de Desarrollo de Software. Los procesos de diseño arquitectónico planteados en estas metodologías no han sido pensados específicamente para el desarrollo de software de RV.

Situación Problémica

Actualmente la Facultad 5 no cuenta con una guía detallada que explique los pasos a seguir para lograr una buena arquitectura en este tipo de software.

El arquitecto de software es el responsable de mantener la integridad conceptual del sistema y su deber fundamental es asegurarse de que éste cumpla con los requisitos de calidad especificados.

En el marco de los proyectos de RV no es muy conocido el alcance de la responsabilidad del rol del arquitecto de software. Por lo general estos proyectos cuentan con arquitectos de software que poseen poca o ninguna experiencia en este rol, lo cual conlleva a que el proceso de diseñar la arquitectura en muchas ocasiones no se realice o se haga de manera muy empírica y se obtengan diseños de poca calidad que no satisfacen las necesidades de actividades posteriores como el diseño detallado y la implementación.

En muchas ocasiones no se toman decisiones arquitectónicas adecuadas en etapas tempranas del desarrollo que eviten riesgos asociados a los frameworks y componentes de software que se utilizan, patrones de diseño, etc.

Por tanto el **problema científico** es el siguiente**:**

¿Cómo elaborar el proceso de diseño arquitectónico ajustado a las características de los software de Realidad Virtual?

Objeto de estudio:

Proceso de Diseño de Arquitectura de Software.

Con el fin de resolver el problema científico se trazó el siguiente **objetivo general:**

Elaborar un proceso de diseño arquitectónico para Software de Realidad Virtual que contribuya a elevar la calidad del diseño de alto nivel del software y en consecuencia del producto final.

Campo de Acción:

Proceso de Diseño Arquitectónico para Software de Realidad Virtual.

Para el total cumplimiento del objetivo general se trazan las siguientes **tareas investigativas:**

1. Revisión de la bibliografía existente y análisis del estado del arte sobre Metodologías de Desarrollo de Software, para analizar cómo realizan el proceso de diseño arquitectónico.
2. Revisión de la bibliografía existente y análisis del estado del arte sobre Arquitectura de Software, para estudiar en qué estado se encuentra la misma y cuáles son sus principales tendencias.
3. Establecimiento de una comparación entre los procesos de Diseño Arquitectónico de las metodologías analizadas, para tomar las ventajas de cada uno e integrarlas en la propuesta de solución.
4. Obtención de información acerca del proceso de diseño de la arquitectura en los proyectos de Realidad Virtual que pertenecen a la Facultad 5, para determinar las necesidades actuales de los mismos y fundamentar la propuesta de solución.

5. Análisis de los documentos de Arquitectura y Especificación de los Requisitos de cada proyecto de Realidad Virtual que se realizan en la Facultad 5, para detectar las principales deficiencias de los mismos.

6. Procesamiento de la información obtenida, para agrupar los resultados que permitan obtener estadísticas de las principales deficiencias que posee el proceso de diseño arquitectónico en los Proyectos de Realidad Virtual de la Facultad 5.

7. Definición de un proceso para diseñar la arquitectura de los software de Realidad Virtual, para de esta forma elevar la calidad del Diseño Arquitectónico de los mismos.

8. Validación del proceso propuesto aplicando el mismo en un proyecto perteneciente al polo de Realidad Virtual de la Facultad 5, para de esta forma asegurar su confiabilidad.

Para la realización de estas tareas se utilizarán los siguientes **métodos de investigación:**

Métodos Teóricos:

Analítico – Sintético: Para facilitar su estudio mediante la descomposición en sus principales partes y seguidamente resumir los distintos enfoques que le dan las Metodologías de Desarrollo de Software al proceso de Diseño Arquitectónico.

Inductivo – Deductivo: Partir del estudio de los específicos y distintos procesos de Diseño Arquitectónico planteados en las Metodologías de Desarrollo analizadas, para obtener las ideas generales que permitan determinar qué características de cada diseño se pueden reutilizar en la propuesta de solución concreta que se va a elaborar.

Métodos Empíricos:

Entrevista: Realizar entrevistas a los arquitectos de software, diseñadores y programadores de los proyectos de Realidad Virtual de la Facultad 5 con el objetivo de obtener información sobre el Diseño Arquitectónico empleado en éstos.

Análisis Estadístico: A través de la Estadística Descriptiva para la representación e interpretación de los datos obtenidos a partir de las entrevistas realizadas en los proyectos de Realidad Virtual de Facultad 5.

Encuesta: Realizar encuestas a los involucrados en la validación del proceso (arquitecto, líder, diseñadores, especialistas de calidad del proyecto, entre otros) sobre la efectividad de los resultados obtenidos luego de aplicar la propuesta a un proyecto de Realidad Virtual de la Facultad 5.

Este trabajo está estructurado en tres capítulos:

Capítulo 1: Fundamentación Teórica

En este capítulo se define el marco teórico y el modelo teórico de la investigación y se realiza un estudio del arte de las Metodologías de Desarrollo de Software, de la Arquitectura de Software y del proceso de diseñar la arquitectura en cada una de estas metodologías.

Se definen además los parámetros del Diseño Arquitectónico a evaluar en los proyectos de Realidad Virtual de la facultad 5.

Capítulo 2: Propuesta de Solución

En este capítulo se realiza un análisis del estado actual del proceso de Diseño Arquitectónico en los proyectos de Realidad Virtual de la Facultad 5.

Se describe cada uno de los elementos que componen el Proceso de Diseño Arquitectónico para software de Realidad Virtual.

Capítulo 3: Validación de la Solución

En este capítulo se analizan los resultados obtenidos luego de la utilización del proceso propuesto para diseñar la arquitectura en un proyecto que pertenece al polo de Realidad Virtual de la Facultad 5. Se realizan además encuestas a los involucrados en la validación del proceso.

CAPÍTULO 1

Fundamentación Teórica

1.1 Introducción

En el presente capítulo se realiza un análisis del estado del arte de las Metodologías de Desarrollo de Software, de la Arquitectura de Software y del proceso de diseño arquitectónico en las metodologías que se consideran más relevantes.

1.2 Proceso

Cada autor proporciona su propia definición de *proceso*, por ejemplo, Evan J. R. y Lindsay W. en su libro "Administración y Control de la Calidad" definen un proceso como: "Una secuencia de actividades que tienen la finalidad de lograr algún resultado, generalmente crear un valor agregado para el cliente". (Evans & Lindsay, 2000)

Otra definición interesante de *proceso* es la que refiere: "un proceso implica el uso de los recursos de una organización, para obtener algo de valor. Así, ningún producto puede fabricarse y ningún servicio puede suministrarse sin un proceso, y ningún proceso puede existir sin un producto o servicio". (Krajewki & Ritzman, 2000)

El Diccionario de la Real Academia Española define *proceso* como: "acción o sucesión de acciones continuas regulares, que ocurren o se llevan a cabo de una forma definida, y que llevan al cumplimiento de algún resultado; una operación continua o una serie de operaciones".

A manera de resumen se puede decir que un *proceso* es un conjunto de tareas o actividades las cuales se encuentran interrelacionadas entre sí y que poseen un orden lógico, donde a partir de un conjunto de entradas, que pueden ser: información, materiales o salidas de otros procesos, se obtienen un conjunto de salidas como por ejemplo: productos o información con un valor añadido.

Algunos ejemplos de procesos pueden ser los de producción de bienes, entrega de productos o servicios, procesos biotecnológicos, educativos, etc., así como los Procesos de Desarrollo de Software.

1.2.1 Proceso de Desarrollo de Software

"Un proceso define *quién* está haciendo *qué*, *cuándo* y *cómo* alcanzar un determinado objetivo. En la Ingeniería de Software el objetivo es construir un producto de software o mejorar uno existente. Un proceso efectivo proporciona normas para el desarrollo eficiente de software de calidad. Captura y presenta las mejores prácticas que el estado actual de la tecnología permite". (Jacobson, Booch, & Rumbaugh, 1999)

El proceso de Ingeniería de Software se define como "un conjunto de etapas parcialmente ordenadas con la intención de lograr un objetivo, en este caso, la obtención de un producto de software de calidad". (Jacobson, 1998)

… "un proceso de software se define como un marco de trabajo de las tareas que se requieren para construir software de alta calidad"…(Pressman R. S., 2002)

Según Pressman existe un estrecho vínculo entre proceso de desarrollo de software e Ingeniería de Software, aunque no se deben confundir estos conceptos, puesto que …"un proceso de software define el enfoque que se toma cuando el software es tratado por la ingeniería, pero la ingeniería del software también comprende las tecnologías que tiene el proceso, métodos técnicos y herramientas automatizadas"… (Pressman R. S., 2002)

Figura 1: Capas de la Ingeniería de Software según Pressman

1.2.1.1 Subprocesos del Proceso de Desarrollo de Software

Cada proceso de desarrollo de software está estructurado a su vez por diversos subprocesos. Estos subprocesos pueden ejecutarse de forma secuencial o paralela y permiten a los equipos de desarrollo de software realizar diversas actividades de forma organizada.

Subprocesos que propone RUP

El Proceso Unificado de Desarrollo (RUP) es el producto final de tres décadas de desarrollo y uso práctico, desde el proceso Objectory (primera publicación en 1987), pasando por el proceso Objectory de Rational (publicado en 1997) hasta el Proceso Unificado de Rational (publicado en 1998). Su desarrollo ha recibido influencia de muchas fuentes, entre ellas el método Ericsson y el método Rational. (Jacobson, Booch, & Rumbaugh, 1999)

RUP se caracteriza por ser un proceso:

- ✓ Dirigido por Casos de Uso

Un caso de uso es un fragmento de funcionalidad del sistema que proporciona al usuario un resultado importante. Los casos de uso son una herramienta para especificar los requisitos del sistema, para guiar además su diseño, implementación y prueba, es decir guían el proceso de desarrollo. (Jacobson, Booch, & Rumbaugh, 1999)

El proceso de desarrollo sigue un hilo, es decir, avanza a través de una serie de flujos de trabajo que parten de los casos de uso. (Jacobson, Booch, & Rumbaugh, 1999)

- ✓ Centrado en la arquitectura

La arquitectura muestra la visión común del sistema, describe los elementos del modelo que son más importantes para su construcción. RUP se desarrolla en iteraciones comenzando por los casos de uso significativos para la arquitectura.

Debe existir interacción entre los casos de uso y la arquitectura, debido a que los casos de uso deben encajar en la arquitectura y por otro lado la arquitectura debe permitir el desarrollo de todos los casos de uso requeridos. (Jacobson, Booch, & Rumbaugh, 1999)

- ✓ Iterativo e Incremental

El trabajo se divide en partes más pequeñas, cada una de estas partes es una iteración que resulta en un incremento. Las iteraciones hacen referencia a pasos en el flujo de trabajo y los incrementos al crecimiento del producto. (Jacobson, Booch, & Rumbaugh, 1999)

RUP llama flujos de trabajo a los subprocesos que lo componen, estos son: Modelo de Negocio, Requisitos, Análisis y Diseño, Implementación, Prueba, Despliegue, Administración de Proyecto, Administración de Configuración y Cambios y Ambiente.

Se centrará la atención en el proceso de diseño:

En el diseño se modela el sistema y se encuentra su forma para que soporte los requisitos. Es un modelo físico y mucho más formal que el análisis. Una entrada esencial en el diseño es el resultado del análisis. (Jacobson, Booch, & Rumbaugh, 1999)

El diseño es el centro de atención durante la fase de elaboración y el comienzo de las iteraciones de construcción. Esto contribuye a una arquitectura estable y sólida y a crear un plano del modelo de implementación. (Jacobson, Booch, & Rumbaugh, 1999)

En concreto, los propósitos del diseño son: (Jacobson, Booch, & Rumbaugh, 1999)

- ✓ Adquirir una comprensión en profundidad de los aspectos relacionados con los Requisitos No Funcionales (RNF) y restricciones relacionadas con los lenguajes de programación, componentes reutilizables, sistemas operativos, tecnologías de distribución y concurrencias, tecnologías de interfaz de usuario y tecnologías de gestión de transacciones.
- ✓ Crear una entrada apropiada y un punto de partida para actividades de implementación subsiguientes, capturando los requisitos o subsistemas individuales, interfaces y clases.
- ✓ Descomponer los trabajos de implementación en partes más manejables que puedan ser llevadas a cabo por diferentes equipos de desarrollo, teniendo en cuenta la posible concurrencia.
- ✓ Crear una abstracción sin costuras, en el sentido de que la implementación es un refinamiento directo del diseño que rellena lo existente sin cambiar la estructura.

Durante este proceso se generan los artefactos: Modelo de Diseño, Realización de Casos de Uso – Diseño, Descripción de la Arquitectura, Modelo de Despliegue. (Jacobson, Booch, & Rumbaugh, 1999)

Subprocesos en XP

La Programación Extrema (XP) define las siguientes fases durante su ciclo de desarrollo: Exploración, Planificación de la entrega, Iteraciones, Producción, Mantenimiento y Muerte del Proyecto. (ISSI, 2003)

XP define además una serie de actividades a desarrollar, algunas de estas son: Codificar, Hacer Pruebas, Escuchar, Diseñar, Planificar y Refactorizar. (ISSI, 2003)

XP hace énfasis en el diseño simple. Se debe diseñar la solución más simple que pueda funcionar. (ISSI, 2003)

Procesos dentro de la Ingeniería de Software según Pressman

Según Pressman todo trabajo que se asocia a la Ingeniería de Software se puede dividir en tres fases genéricas, con independencia del área de aplicación, tamaño o complejidad del proyecto: (Pressman R. S., 2002)

- ✓ La fase de *definición*, la cual se centra en el *qué*, es decir, en las funcionalidades del sistema, rendimiento que se desea, restricciones de diseño, etc.
- ✓ La fase de *desarrollo* se centra en el *cómo*, o sea, cómo han de diseñarse las estructuras de datos, cómo ha de implementarse una función, etc.
- ✓ Por último la fase de *mantenimiento*, la cual se centra en el *cambio*, es decir, corrección de errores, adaptaciones requeridas a medida que evoluciona el entorno del software, mejoras, etc.

Pressman plantea además que la Ingeniería de Software aparece como consecuencia de un proceso denominado Ingeniería de Sistemas. Los subprocesos de la Ingeniería de Sistemas son: Ingeniería de Proceso de Negocio, Ingeniería de Producto, Ingeniería de Requisitos, Modelo de Análisis, Diseño Arquitectónico, Diseño de la Interfaz de Usuario, Diseño a Nivel de Componentes y Pruebas. (Pressman R. S., 2002)

Pressman señala a su vez que las salidas del Diseño Arquitectónico deben ser: descripción del procesamiento de cada módulo, descripción de la interfaz por cada módulo, definición de las estructuras de datos generales y locales, y restricciones del diseño. (Pressman R. S., 2002)

Hasta aquí se han abordado los conceptos de proceso y proceso de desarrollo de software, centrando en los subprocesos de diseño de software, los cuales son de interés en esta investigación, y se profundizará en ellos en los siguientes epígrafes.

A continuación se analizan los niveles de abstracción del diseño de software y luego se centra en el diseño de alto nivel: o bien conocido como Arquitectura de Software.

1.3 Niveles de Abstracción del Diseño de Software

Para el desarrollo de un sistema por lo general se consideran distintos niveles de abstracción. El nivel superior de abstracción plantea una solución global y utiliza un lenguaje en el entorno del problema. En los niveles inferiores se enfoca más hacia los procesos y la implementación, de esta forma se llega a la descripción utilizando un lenguaje orientado a la implementación. Cada paso de un proceso de la Ingeniería del Software es un refinamiento del nivel de abstracción de la solución.

1.4 Arquitectura de Software

"La arquitectura de software es una disciplina reciente" (Szyperski, 2000). Ninguna definición de Arquitectura de Software es respaldada unánimemente por la totalidad de los arquitectos.

Una definición reconocida es la de Paul Clements: …"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 la 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"… (Clements, 1996)

Cuando los arquitectos se refieren a un componente no es precisamente a los componentes de software, sino a los componentes de arquitectura. Un componente es un elemento o entidad, a la que los arquitectos prefieren llamar "componente" antes que "objeto". Los componentes de la arquitectura o "elementos" pueden ser dinámicos o estáticos estos son elementos ya sea de procesamiento, de datos o de conexión, mientras que los componentes de software se refiere a subsistemas, clases, interfaces, etc. (Reinoso, 2004)

David Garlan establece que la Arquitectura de Software constituye un puente entre el requerimiento y el código.

Mary Shaw y David Garlan, sugieren que la Arquitectura del Software sea un nivel del diseño referido a las ediciones…"más allá de las estructuras de los algoritmos y de datos del cómputo; diseñar y especificar la estructura del sistema total emerge como nueva clase de problema. Las ediciones estructurales incluyen la organización gruesa y la estructura global del control; los protocolos para la comunicación, la sincronización, y el acceso a los datos; asignación de la funcionalidad para diseñar elementos; distribución física; composición de los elementos del diseño; escalamiento y funcionamiento; y selección entre alternativas del diseño"… (Shaw & Garlan, 1996)

La IEEE 1471-2000, define la Arquitectura de Software como:

…"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"… (IEEE-1471, 2000)

La IEEE plantea además que la Arquitectura no se inscribe en ninguna metodología de desarrollo, sino que es una disciplina que se desarrolla durante todo el ciclo de desarrollo de software.

Independientemente de las discrepancias entre las diversas definiciones, es común entre todos los autores el concepto de la arquitectura como un punto de vista que concierne a un alto nivel de abstracción.

Para resumir se puede decir que la Arquitectura de Software es el diseño de más alto nivel de la estructura de un sistema que aporta una visión abstracta posponiendo el detalle de cada uno de los módulos definidos a pasos posteriores del diseño. Define estilos o combinación de estilos para una solución, se concentra en los requisitos no funcionales y es esencial para el éxito o fracaso de un proyecto.

1.4.1 Frameworks

Un Framework o Marco de trabajo es una estructura de soporte que define una arquitectura adaptada a las particularidades de un determinado dominio de aplicación, mediante la cual otro proyecto de software puede ser organizado y desarrollado, definiendo de forma abstracta una serie de componentes y sus interfaces.

1.4.2 Vistas de la Arquitectura de Software

La Arquitectura de Software describe diversos aspectos del software. Estos aspectos son descritos de una forma mucho más comprensible si se utilizan los llamados modelos o vistas.

La idea de contemplar un sistema complejo desde múltiples puntos de vista no es nativa de la arquitectura de software contemporánea, sino que se origina por lo menos en la década de 1970, en el trabajo de Douglas Ross sobre análisis estructurado. (Ross., 1977)

Según la IEEE 1471-2000 una vista es la representación concreta de un sistema en particular desde una perspectiva unitaria. Una vista también se compone de modelos, aunque posee también atributos adicionales. (IEEE-1471, 2000)

No hay un límite necesario para el número de vistas posibles, ni un procedimiento formal para establecer lo que una vista debe o no abstraer. El estándar IEEE-1471 no delimita el número posible de vistas, debido a que se estima que no puede haber acuerdo en ello, pero señala lineamientos para su constitución y considera que un punto de vista es a una vista como una clase es a un objeto. (IEEE-1471, 2000)

Las vistas o modelos de una arquitectura se expresan utilizando uno o varios lenguajes. Se puede decir que el lenguaje más sencillo es el natural, pero actualmente existen otros como por ejemplo: los diagramas de estados y los de flujos de datos, los cuales son apropiados solamente para un modelo o vista. Hoy en día el lenguaje que más se utiliza para la representación de todas las vistas es el UML (Lenguaje Unificado de Modelado).

Se puede decir además que tanto los marcos arquitectónicos como las metodologías de modelado acostumbran ordenar las diferentes perspectivas de una arquitectura en términos de vistas:(Reinoso, 2004)

El marco de referencia arquitectónico de The Open Group (TOGAF) reconoce cuatro componentes principales, uno de los cuales es un framework de alto nivel que a su vez define cuatro vistas:

- ✓ Arquitectura de Negocios
- ✓ Arquitectura de Datos/Información
- ✓ Arquitectura de Aplicación
- ✓ Arquitectura Tecnológica

En 1995 Philippe Kruchten propuso su célebre modelo “4+1”, vinculado al Proceso Unificado de Desarrollo (RUP), que define cuatro vistas diferentes de la Arquitectura de Software: (Kruchten, 1995)

- ✓ Vista Lógica
- ✓ Vista de Proceso
- ✓ Vista Física
- ✓ Vista de Desarrollo
- ✓ Vista de Casos de Uso

En su introducción a UML (1.3), Grady Booch, James Rumbaugh e Ivar Jacobson han formulado un esquema de cinco vistas interrelacionadas que conforman la arquitectura de software. Los autores proporcionan un esquema de cinco vistas posibles de la arquitectura de un sistema: (Rumbaugh, Jacobson, & Booch, 1999)

- ✓ Vista de Casos de Uso
- ✓ Vista de Diseño
- ✓ Vista de Procesos
- ✓ Vista de Implementación
- ✓ Vista de Despliegue

La recomendación IEEE 1471-2000 procura establecer una base común para la descripción de Arquitecturas de Software, e implementa para ello tres términos básicos: arquitectura, vista y punto de vista. (IEEE-1471, 2000)

La estrategia de arquitectura de Microsoft define, en consonancia con las conceptualizaciones más generalizadas, cuatro vistas, ocasionalmente llamadas también arquitecturas: (Platt, 2002)

- ✓ Negocios
- ✓ Aplicación
- ✓ Información
- ✓ Tecnología

1.4.3 Estilos Arquitectónicos

Los Estilos Arquitectónicos de Software son Arquitecturas de Software comunes, que definen desde un vocabulario de tipos de componentes y conectores hasta el conjunto de restricciones sobre cómo combinar esos componentes y conectores.

Los estilos de arquitectura se definen como las 4 C (Wolf & Perry, 1992):

- ✓ Componentes
- ✓ Conectores
- ✓ Configuraciones
- ✓ Restricciones

Algunos de los principales estilos arquitectónicos que se utilizan en la actualidad están divididos por Clases de Estilos las que engloban una serie de estilos arquitectónicos específicos (Shaw & Garlan, 1996):

- ✓ Estilos de Flujo de datos
  - Tuberías y Filtros

- ✓ Estilos centrados en datos
    - Arquitecturas de Pizarra o repositorio
- ✓ Estilos de Llamada y Retorno
    - Modelo – Vista – Controlador (MVC)
    - Arquitectura en Capas
    - Arquitectura Orientada a Objetos
    - Arquitectura basada en Componentes
- ✓ Estilo de Código Móvil
    - Arquitectura de Máquinas Virtuales
- ✓ Estilos Peer – To – Peer
    - Arquitectura basadas en Eventos
    - Arquitecturas Orientas a Servicios (SOA)

1.5 Descripción Arquitectónica

La Descripción Arquitectónica depende de mucho más que la simple descripción de plataformas o frameworks de trabajo, sino que depende de los componentes (elementos), con aserción de propiedades, interfaces e implementaciones, los conectores, las configuraciones o sistemas de abstracción y encapsulamiento, los requisitos no funcionales, las restricciones, los estilos, la evolución y hasta las herramientas de verificación.

Existen varios tipos de lenguajes utilizados para describir la arquitectura de un sistema, aquí solo se analizan dos tipos esenciales.

1.5.1 Lenguajes de Descripción Arquitectónica (ADLs)

Los lenguajes de descripción de arquitecturas surgieron aproximadamente a principios de la década de 1990, en contemporaneidad con el Lenguaje Unificado de Modelado. Su función fundamental es la representación visual de la arquitectura de un sistema. Los ADLs permiten modelar una arquitectura mucho antes que se lleve a cabo la programación de las aplicaciones que la componen, analizar su adecuación, determinar sus puntos críticos y eventualmente simular su comportamiento. (Reinoso, 2004)

Entre los principales ADLs se encuentran:

- ✓ ACME
- ✓ Armani

- ✓ Jacal
- ✓ CHAM

1.5.2 Lenguaje Unificado de Modelado (UML)

El Lenguaje Unificado de Modelado (UML) es una notación para especificar, visualizar y documentar sistemas de software desde la perspectiva orientada a objetos.

El Lenguaje Unificado de Modelado permite a los creadores de sistemas generar diseños que capturen sus ideas en una forma convencional y fácil de comprender para comunicarlas a otras personas. (Schmuller, 2000)

UML es ya un estándar de la industria, pero no sólo de la industria del software, sino, en general, de cualquier industria que requiera la construcción de modelos como condición previa para el diseño y posterior construcción de prototipos. (Rumbaugh, Jacobson, & Booch, 1999)

Diagramas de UML

La finalidad de los diagramas es presentar diversas perspectivas de un sistema, a las cuales se les conoce como modelo. El modelo UML de un sistema es similar a un modelo a escala de un edificio junto con la interpretación del artista del edificio. Es importante destacar que un modelo UML describe lo que supuestamente hará un sistema, pero no dice como implementar dicho sistema. (Schmuller, 2000)

Los diagramas de UML se pueden dividir en:

- ✓ Diagramas Estáticos
  - Diagramas de Caso de Uso
  - Diagramas de Clase
  - Diagramas De Objeto
- ✓ Diagramas de Comportamiento
  - Diagramas de Actividad
  - Diagramas de Colaboración
  - Diagramas de Secuencia
- ✓ Diagramas de Implementación
  - Diagramas de Componentes
  - Diagramas de Despliegue

1.6 El Arquitecto de Software

El arquitecto crea la arquitectura junto con otros desarrolladores. Trabajan para conseguir un sistema que tendrá un alto rendimiento y una alta calidad, y será completamente funcional, verificable, amigable para el usuario, fiable, de alta disponibilidad, preciso, extensible, tolerante a cambios, robusto, portable, confiable, seguro y económico. (Jacobson, Booch, & Rumbaugh, 1999)

Un arquitecto calificado se consigue mediante dos tipos de aptitudes, una es el conocimiento del dominio en el que trabaja, el otro es el conocimiento del desarrollo de software, incluso debe ser capaz de escribir código, puesto que debe comunicar la arquitectura a los desarrolladores, coordinar sus esfuerzos y obtener su retroalimentación. El arquitecto ocupa un puesto difícil en la organización de desarrollo. (Jacobson, Booch, & Rumbaugh, 1999)

El arquitecto ideal debe ser una persona de letras, matemático, al corriente de estudios históricos, un estudiante diligente de la filosofía, conocido de la música, no ignorante de medicina, entendido en las respuestas de consultoría jurídica, al corriente de astronomía y de cálculos astronómicos. (Ayuda Rational, 2003)

Resumiendo, el arquitecto de software debe tener visión, y una profundidad de la experiencia que permite tomar decisiones rápidamente y hacer el juicio adecuado, crítico en ausencia de la información completa. (Ayuda Rational, 2003)

1.7 Diseño Arquitectónico

Según (Zavalar, 2008) el diseño de software se realiza a tres niveles: conceptual, lógico y físico.

El diseño conceptual se considera como un análisis de actividades y consiste en la solución de negocios para el usuario y se expresa con los casos de uso.

El diseño lógico traduce los escenarios de uso creados en el diseño conceptual en un conjunto de objetos de negocio y sus servicios. El diseño lógico se convierte en parte de la especificación funcional que se usa en el diseño físico. El diseño lógico es independiente de la tecnología. El diseño lógico refina, organiza y detalla la solución de negocios y define formalmente las reglas y políticas específicas de negocios.

El diseño físico traduce el diseño lógico en una solución implementable y económica.

1.7.1 Diseño Evolutivo

Esencialmente, evolutivo significa que el diseño del sistema crece conforme se implanta el sistema. El diseño es parte del proceso de programación y conforme el programa evoluciona el diseño cambia. (programacionextrema.org, 2002)

Una de las dificultades del diseño evolutivo es que es muy difícil determinar si el diseño está realmente ocurriendo. El peligro de entremezclar diseño con programación es que puede haber programación sin diseño, ésta es la situación donde el Diseño Evolutivo diverge y falla. (programacionextrema.org, 2002)

1.7.2 Diseño Planeado

En el diseño planeado los diseñadores piensan los grandes problemas con anticipación. No necesitan programar porque no están construyendo el software, sólo lo están planeando, así que pueden usar técnicas de diseño como el UML que deja de lado algunos de los detalles de la programación y permite a los diseñadores trabajar a un nivel más abstracto. Una vez hecho el diseño, pueden pasarlo a otro grupo (o a otra compañía) que lo construya; debido a que los diseñadores están pensando en una escala mayor, pueden evitar la serie de decisiones tácticas que llevan a la entropía del software. Los programadores pueden seguir la dirección del diseño y, dado que siguen el diseño al pie de la letra, tener un sistema bien construido. (programacionextrema.org, 2002)

1.8 Metodologías de Desarrollo de Software

Se define una Metodología de Desarrollo de Software como el conjunto de pasos y procedimientos que deben seguirse para desarrollar un software.

Una Metodología de Desarrollo de Software muestra cómo dividir un proyecto en etapas y qué tareas se deben llevar a cabo en cada una de estas etapas, muestra además qué técnicas y herramientas se deben emplear y de qué forma se controla y gestiona el proyecto.

En la actualidad existen diversas Metodologías de Desarrollo de Software, el problema fundamental consiste en analizar cuál es la que más se ajusta a un software en específico, teniendo en cuenta que una mal elección de la metodología de desarrollo puede conllevar a un software de mala calidad final y a obtener clientes insatisfechos con su producto final.

La evolución de la disciplina de Ingeniería de Software ha traído consigo diferentes tendencias para mejorar los resultados del proceso de construcción. Las metodologías tradicionales haciendo énfasis

en la planeación, y las metodologías ágiles en la adaptabilidad del proceso, delinean las principales propuestas presentes en la literatura.

El desarrollo tradicional de software se basa en procesos predefinidos, los cuales deben contar con documentación muy precisa, y una detallada planificación inicial.

Mediante una rigurosa definición de actividades, artefactos y roles. Este esquema "tradicional" para abordar el desarrollo de software ha demostrado ser efectivo en proyectos de gran envergadura donde por lo general se exige un alto grado de ceremonia en el proceso. (ISSI, 2003)

Este enfoque no resulta ser el más adecuado para muchos de los proyectos actuales donde el entorno del sistema es muy cambiante, y en donde se exige reducir drásticamente los tiempos de desarrollo pero manteniendo una alta calidad. (ISSI, 2003)

En la actualidad existen diversos proyectos donde el entorno es volátil y no se conocen los requisitos con exactitud desde la fase de inicio, por lo que las metodologías tradicionales no son las más convenientes a utilizar, puesto que no están pensadas para trabajar con incertidumbre.

En consecuencia a esto surgen las metodologías ágiles, también conocidas como metodologías livianas.

El desarrollo ágil de software se basa en procesos ágiles que se enfocan en los clientes y en los resultados. Estos procesos están preparados para realizar cambios en los requisitos en cualquier etapa de desarrollo, y por lo general no exigen documentación extensiva.

Entre las ventajas fundamentales que ofrecen las metodologías agiles están: la capacidad de respuesta a cambios de requisitos a lo largo del desarrollo, la entrega continua y en plazos breves de software funcional, trabajo conjunto entre el cliente y el equipo de desarrollo, importancia de la simplicidad, eliminado el trabajo innecesario, atención continua a la excelencia técnica y al buen diseño y mejora continua de los procesos y del equipo de desarrollo.

Generalmente los métodos ágiles son criticados y tratados como "indisciplinados" por la falta de documentación técnica.

1.9 Diseño Arquitectónico y Metodologías de Desarrollo de Software

Cada Metodología de Desarrollo de Software define el proceso de diseño arquitectónico, pero partiendo de los principios de tantas y diversas metodologías es muy difícil sacar una visión unificada del mismo.

A continuación se analizan algunas metodologías de desarrollo, las cuales se consideran son las más utilizadas, en cuanto a su Diseño Arquitectónico y su enfoque a la Arquitectura de Software.

1.9.1 Proceso Unificado de Software y Arquitectura de Software.

El Proceso Unificado de Desarrollo de Software plantea que la arquitectura involucra los elementos más significativos del sistema y está influenciada entre otros factores por plataformas, software, sistemas operativos, manejadores de bases de datos, protocolos, consideraciones de desarrollo como sistemas heredados y requisitos no funcionales. Se representa mediante varias vistas que se centran en aspectos concretos. (Ayuda Rational, 2003)

La arquitectura se desarrolla de forma iterativa durante la fase de elaboración, pasando por los requisitos, el análisis, el diseño, la implementación y las pruebas. Se utilizan los casos de uso significativos para la arquitectura y un conjunto de otras entradas para implementar la línea base de la arquitectura. (Jacobson, Booch, & Rumbaugh, 1999)

En el Proceso Unificado de Desarrollo la Arquitectura de Software se representa mediante 4+1 Vistas: vista de casos de uso, lógica, implementación, proceso y despliegue.

Proceso de Diseño Arquitectónico en RUP

La siguiente figura muestra la secuencia de actividades que propone RUP para el desarrollo del Flujo de Trabajo (FT) Análisis y Diseño, en el cual se realiza el trabajo de mayor peso en cuanto a diseño arquitectónico:

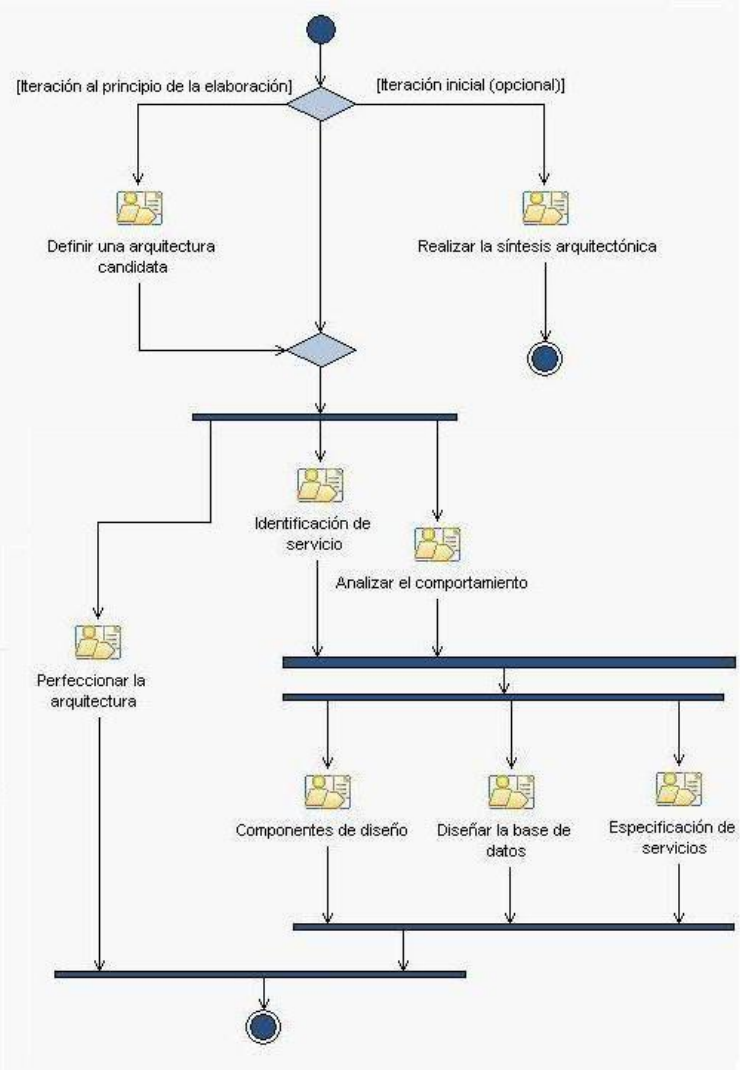


Figura 2: Actividades del FT Análisis y Diseño en la metodología RUP

Definir una arquitectura Candidata:

Según (Ayuda Rational, 2003) para definir la Arquitectura Candidata del Sistema es necesario realizar las siguientes actividades:

Actividad	Rol Responsable	Breve Descripción
Definir el Contexto del sistema	Analista de Sistemas	En esta tarea se define cómo crear un diagrama de contexto del sistema que muestre la colaboración de nivel superior entre el sistema y sus actores.

Análisis de la Arquitectura	Arquitecto de Software	Esta tarea se centra en la definición de una arquitectura candidata y en la restricción de las técnicas de arquitectura que se van a utilizar en el sistema. Los pasos a seguir para la realización de la misma son: Desarrollar una visión general de la arquitectura, Inspeccionar los activos disponibles, Definir la organización de alto nivel de los subsistemas, Identificar abstracciones claves, Identificar iteraciones estereotipadas, Desarrollar una visión general del desarrollo, Identificar mecanismos de análisis y Revisar los resultados.
Análisis de CU	Diseñador	En esta tarea se identifican las clases que ejecutan el flujo de sucesos de cada caso de uso. Se distribuye además el comportamiento de cada caso de uso en estas clases, utilizando realizaciones de caso de uso del análisis.
Análisis de la Operación	Diseñador	En esta tarea el diseñador inicia la transformación de una descripción del comportamiento a nivel de sistema en una estructura de sistema sin detallar (y en sus interacciones asociadas) y comportamiento a nivel de subsistemas.
Identificar Patrones de Seguridad	Arquitecto de Seguridad	El objetivo de esta tarea es permitir que los arquitectos identifiquen patrones de seguridad de alto nivel adecuados para conectar los elementos arquitectónicos en respuesta a los requisitos y las políticas de seguridad.

Tabla 1: Actividad Definir una Arquitectura Candidata del FT Análisis y Diseño propuesto por la metodología RUP.

Refinar la Arquitectura

Según (Ayuda Rational, 2003) para Refinar la Arquitectura del Sistema es necesario realizar las siguientes actividades:

Actividad	Rol Responsable	Breve Descripción
Identificar mecanismo de diseño	Arquitecto de Software	En esta tarea se describe cómo perfeccionar los mecanismos de análisis en mecanismos de diseño.
Identificar elementos de diseño	Arquitecto de Software	En esta tarea se explica cómo se identifican los subsistemas, las clases, las interfaces, los sucesos y las señales.
Incorporar	Arquitecto de Software	En esta tarea se describe cómo evolucionar y perfeccionar el modelo de

elementos de diseño existentes		diseño, identificando oportunidades de reutilización y revirtiendo la ingeniería de los componentes y las bases de datos.
Estructurar el modelo de Implementación	Arquitecto de Software	En esta tarea se describe cómo establecer la estructura de los elementos de implementación basándose en las responsabilidades asignadas de los subsistemas de implementación y su contenido.
Describir la arquitectura de tiempo de ejecución	Arquitecto de Software	Esta tarea define una arquitectura del proceso para el sistema en términos de clases activas e instancias, y la relación de éstas con los procesos y las hebras del sistema operativo.
Describir la distribución	Arquitecto de Software	En esta tarea se define la arquitectura de despliegue de un sistema distribuido en términos de nodos físicos y sus interconexiones.
Análisis de la Operación	Diseñador	En esta tarea el diseñador inicia la transformación de una descripción del comportamiento a nivel de sistema en una estructura de sistema sin detallar (y en sus interacciones asociadas) y comportamiento a nivel de subsistemas.
Revisar la arquitectura	Revisor Técnico	En esta tarea se define cuándo y cómo se realiza la revisión de una arquitectura y cómo se revisan los resultados.

Tabla 2: Actividad Refinar la Arquitectura del FT Análisis y Diseño propuesto por la metodología RUP.

Analizar el comportamiento

Según (Ayuda Rational, 2003) para Analizar el Comportamiento del Sistema es necesario realizar las siguientes actividades:

Actividad	Rol Responsable	Breve Descripción
Identificar elementos de diseño	Arquitecto de Software	En esta tarea se explica cómo se identifican los subsistemas, las clases, las interfaces, los sucesos y las señales.
Análisis de CU	Diseñador	En esta tarea se identifican las clases que ejecutan el flujo de sucesos de cada caso de uso. Se distribuye además el comportamiento de cada caso de uso en estas clases, utilizando realizaciones de caso de uso del análisis.
Análisis de la	Diseñador	En esta tarea el diseñador inicia la transformación de una descripción del comportamiento a nivel de sistema en una estructura de sistema sin detallar

Operación		(y en sus interacciones asociadas) y comportamiento a nivel de subsistemas.
Diseñar la Interfaz de usuario	Diseñador de Interfaz de usuario	Producir un diseño de la interfaz de usuario que dé soporte al razonamiento y la mejora de la utilización.
Prototipo de Interfaz de usuario	Diseñador de Interfaz de usuario	Crear un prototipo de la interfaz de usuario del sistema en un intento de validar el diseño de interfaz de usuario con los requisitos funcionales y de utilización.
Revisar el diseño	Revisor técnico	En esta tarea se define cómo se realiza la revisión de un diseño y cómo se revisan los resultados.

Tabla 3: Actividad Analizar el comportamiento del FT Análisis y Diseño propuesto por la metodología RUP.

Realizar la Síntesis Arquitectónica

Según (Ayuda Rational, 2003) para realizar la Síntesis Arquitectónica es necesario realizar las siguientes actividades:

Actividad	Rol Responsable	Breve Descripción
Definir el Contexto del sistema	Analista de Sistemas	En esta tarea se define cómo crear un diagrama de contexto del sistema que muestre la colaboración de nivel superior entre el sistema y sus actores.
Análisis de la Arquitectura	Arquitecto de Software	Esta tarea se centra en la definición de una arquitectura candidata y en la restricción de las técnicas de arquitectura que se van a utilizar en el sistema. Los pasos a seguir para la realización de la misma son: Desarrollar una visión general de la arquitectura, Inspeccionar los activos disponibles, Definir la organización de alto nivel de los subsistemas, Identificar abstracciones claves, Identificar iteraciones estereotipadas, Desarrollar una visión general del desarrollo, Identificar mecanismos de análisis y Revisar los resultados.
Construir Arquitectura de prueba de concepto	Arquitecto de Software	En esta tarea se define cómo desarrollar una arquitectura de prueba de concepto basada en el perfil de riesgos y los requisitos de arquitectura existentes.

Valorar la viabilidad de la arquitectura de prueba de concepto	Arquitecto de Software	En esta tarea se define cómo evaluar una arquitectura de prueba de concepto según los riesgos y los requisitos de arquitectura.

Tabla 4: Actividad Realizar la síntesis Arquitectónica del FT Análisis y Diseño propuesto por la metodología RUP.

1.9.2 Programación Extrema (XP) y Diseño Arquitectónico

La Programación Extrema es una Metodología Ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en el desarrollo de software promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en la retroalimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. XP se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico. (ISSI, 2003)

Ideada por Kent Beck, la programación extrema es un proceso de creación de software diferente al convencional. En palabras de Beck: “XP es una metodología ligera, eficiente, con bajo riesgo, flexible, predecible y divertida para desarrollar software”.

Para algunas personas que sólo han tenido un contacto breve con la Programación Extrema, pareciera que la XP convoca a la muerte del diseño de software. La XP involucra mucho diseño, pero lo hace de una manera diferente a la de los procesos de software establecidos.

La XP ha rejuvenecido la noción del diseño evolutivo con prácticas que permiten a la evolución ser una estrategia de diseño viable. También brinda nuevos retos y habilidades pues los diseñadores necesitan aprender cómo hacer diseño simple, cómo usar refactorización para mantener el diseño limpio y cómo usar patrones en un estilo evolutivo. (programacionextrema.org, 2002)

XP y el Diseño Evolutivo

La XP es controversial por muchas razones, pero una de las banderas rojas clave de la XP es que aboga por el diseño evolutivo en lugar del diseño planeado. (programacionextrema.org, 2002)

La curva del cambio dice que mientras más avanza el proyecto, es exponencialmente más caro hacer cambios. La curva exponencial del cambio significa que el diseño evolutivo no puede funcionar. También explica por qué el diseño planeado debe ser hecho cuidadosamente porque cualquier error encara la misma exponenciación. (programacionextrema.org, 2002)

La suposición fundamental de la XP es que es posible aplanar la curva del cambio lo suficiente como para hacer que funcione el diseño evolutivo. Este aplanamiento es a la vez permitido y explotado por la XP. (programacionextrema.org, 2002)

La **refactorización** en XP es una actividad constante de reestructuración del código con el objetivo de remover duplicación de código, mejorar su legibilidad, simplificarlo y hacerlo más flexible para facilitar los posteriores cambios. Se mejora la estructura interna del código sin alterar su comportamiento externo. (programacionextrema.org, 2002)

Según (programacionextrema.org, 2002) **el diseño XP busca las siguientes habilidades**:

- ✓ Un deseo constante de mantener el código tan claro y simple como sea posible.
- ✓ Habilidades de refactorización, de modo que confiadamente se puedan hacer mejoras cuando se vea la necesidad.
- ✓ Un buen conocimiento de patrones: no sólo las soluciones sino también apreciar cuando usarlos y cuando evolucionar hacia ellos.
- ✓ Saber cómo comunicar el diseño a las personas que necesitan entenderlo, usando código, diagramas y sobre todo conversación.

El valor de la Simplicidad

Dos de los más grandes gritos de batalla de la XP son los eslogans “Haz la cosa más simple que pueda funcionar” y “No lo vas a necesitar” (conocido como YAGNI por sus siglas en inglés). Ambos son manifestaciones de la práctica XP del diseño simple. (programacionextrema.org, 2002)

La manera en que usualmente se describe YAGNI, es que no deberías añadir hoy código que sólo será usado por alguna característica que será necesaria mañana. En principio esto suena simple. El problema viene con cosas tales como marcos (frameworks), componentes reusables, y diseño flexible. Tales cosas son complicadas de construir. (programacionextrema.org, 2002)

Sin embargo el consejo de la XP es que no construyas componentes flexibles y frameworks para el primer caso que necesite esa funcionalidad. Deja crecer esas estructuras como se necesiten. (programacionextrema.org, 2002)

La Arquitectura de Software en XP

Los críticos de la XP afirman que esta metodología ignora la arquitectura, que la ruta es ir hacia la codificación rápida y confiar en que la refactorización resolverá todos los problemas de diseño, lo que constituye una debilidad de la metodología. (programacionextrema.org, 2002)

Se considera que el diseño de más alto nivel del software es esencial para asegurar el éxito del mismo, lo cual contradice esta idea de XP. Si no se define una arquitectura en una etapa temprana de desarrollo, esto puede provocar la obtención de diseños detallados de poca calidad, sobrecarga de trabajo para los diseñadores y programadores, que el sistema no soporte los requisitos no funcionales, insatisfacción por parte de los clientes, así como el fracaso del proyecto.

1.9.3 Administración de Riesgos de MSF Ágil

Microsoft Solution Framework (MSF) no es como tal una metodología, sino prácticas que ajustándose al contexto del proyecto serán más o menos recomendables de aplicar. (Spaces, 2005)

Microsoft ha recopilado las mejores prácticas empleadas por sus grupos de producto, organización interna de tecnología, sus clientes y socios de negocios en todo el mundo y ha reunido los factores de éxito que son comunes a todas ellas, integrándolos en modelos reutilizables con etapas y logros medibles que pueden guiar las decisiones tecnológicas en un contexto de negocios. Estos modelos conforman el Microsoft Solutions Framework. (Arias & Martínez, 2008)

Según (Zavalar, 2008)…"más que una metodología, Microsoft Solutions Framework (MSF) es una serie de modelos flexibles interrelacionados que guían a una organización sobre como ensamblar los recursos, el personal y las técnicas necesarias para asegurar que su infraestructura tecnológica y sus soluciones cumplan los objetivos de negocio. MSF mantiene una relación clara entre los objetivos de negocio y las implementaciones tecnológicas"…

Por otra parte MSF Ágil ofrece una guía sobre cómo organizar personas y proyectos para planificar, desarrollar y desplegar soluciones tecnológicas de software de una forma orientada al cambio. Es un proceso ágil y disciplinado de desarrollo de software y constituye un marco de trabajo extensible y adaptable. (Arias & Martínez, 2008)

A diferencia de algunas metodologías tradicionales MSF Ágil es un proceso iterativo que incrementalmente va aproximando entregables de mayor fidelidad. Es un proceso que no está dirigido únicamente por escenarios, sino también por requisitos de calidad de servicio y en lugar de centrarse en la arquitectura como lo hace RUP, asigna mayor relevancia al Análisis de Riesgos. (Arias & Martínez, 2008)

Los seis pasos que conforman el proceso de Administración de Riesgos que propone MSF Ágil son los siguientes:

Identificación de riesgos: Los riesgos se identifican y se ponen al descubierto. Esto debe realizarse lo antes posible y repetirse con frecuencia a lo largo de todo el ciclo de vida del proyecto.

Análisis y asignación de prioridades: Se transforman las cifras y los datos de los riesgos detectados en información que el equipo puede utilizar para tomar decisiones relacionadas con la asignación de prioridades. Al establecer la prioridad de los riesgos el equipo puede confirmar los recursos del proyecto para administrar los riesgos más importantes.

Planeamiento y Programación de riesgos: Se toma la información obtenida tras el análisis de riesgos y se utiliza para trazar estrategias, planes y acciones. La programación de riesgos garantiza que estos planes se aprueban y se incorporan en la infraestructura. La programación de riesgos es el punto de conexión explícito entre el planeamiento de riesgos y el planeamiento de proyectos.

Seguimiento y elaboración de informes de los riesgos: Supervisa el estado de los riesgos y el progreso de sus planes de acción. El seguimiento de riesgos también incluye en la supervisión de probabilidades, impactos, exposiciones y otras medidas de riesgo para los cambios que pudiesen alterar los planes de prioridades o de riesgos y las características, los recursos o la programación del proyecto.

El informe de los riesgos garantiza que el equipo esté al corriente del estado de los riesgos del proyecto y de los planes para administrarlos.

Control de riesgos: Es el proceso que ejecuta los planes de acción de riesgos y sus informes de estado asociados. Incluye la iniciación de las solicitudes de control de cambios del proyecto si los cambios en el estado o los planes de los riesgos pueden alterar las características, los recursos o la programación del proyecto.

Aprendizaje de riesgos: Formaliza las lecciones aprendidas, los elementos y herramientas relevantes del proyecto y plasma toda esta información en un formato reutilizable para el equipo o la empresa.

1.9.4 FDD: Un proceso de Diseño y Elaboración

FDD con sus siglas en inglés Feature Driven Development (Desarrollo basado en funcionalidades) es un enfoque ágil para el desarrollo de sistemas. Dicho enfoque no hace énfasis en la obtención de los requisitos sino en cómo se realizan las fases de diseño y construcción. Sin embargo, fue diseñado para trabajar con otras actividades de desarrollo de software y no requiere la utilización de ningún modelo de proceso específico. Además, hace énfasis en aspectos de calidad durante todo el proceso e incluye un monitoreo permanente del avance del proyecto. (Calabria, 2003)

Se basa en un proceso iterativo con iteraciones cortas que producen un software funcional que el cliente y la dirección de la empresa pueden ver y monitorear. (Calabria, 2003)

Consiste en cinco fases secuenciales, las cuales son: (Calabria, 2003)

- ✓ Develop an Overall Model (Desarrollo de un modelo global)
- ✓ Build a Feature[1] List (Construcción de una lista de funcionalidades)
- ✓ Plan by Feature (Planeación por funcionalidad)
- ✓ Design by Feature (Diseño por funcionalidad)
- ✓ Build by Feature (Construcción por funcionalidad)

Planeación por Funcionalidad

En base a las features list[2] de la etapa anterior, el Líder de Proyecto, el Líder de Desarrollo y el Chief Programmer[3] establecen hitos y diseñan un cronograma de diseño y construcción. (Calabria, 2003)

Diseño por Funcionalidad

El responsable de programación toma la próxima funcionalidad a ser diseñada, identifica las clases involucradas y contacta los Class Owner[4] correspondientes. Luego el equipo, trabaja en la realización

[1] Feature: Pequeñas funcionalidades que el cliente quiere.

[2] Feature List: Lista que agrupa toda la funcionalidad del sistema.

[3] Chief Programmer: Responsable de la programación.

del diagrama de secuencia correspondiente. El Class Owner hace una descripción de la clase y sus métodos. (Calabria, 2003)

Entrada: Como entrada se necesitan los documentos desarrollados en la fase anterior. (Calabria, 2003)
Salida: Diagrama de secuencia detallado, Diagrama de clases actualizado, Descripción de clases y métodos, Notas del equipo que consideren significativas para el diseño. (Calabria, 2003)

Se enfatiza la realización de un diseño ligero, pero conciso, que exprese los puntos críticos que deben ser comprendidos. Eventualmente, resultará conveniente la realización de revisiones de los diseños propuestos. (Calabria, 2003)

1.10 Conclusiones parciales del proceso de diseño arquitectónico en las metodologías: RUP, XP, FDD y MSF Ágil.

La siguiente tabla muestra las características fundamentales del diseño arquitectónico de las metodologías explicadas anteriormente:

Metodología	Enfoque	Diseño	Arquitectura
RUP	tradicional	planeado	Centrado en la arquitectura, la cual se representa a través de 4 +1 vistas: vista de casos de uso, lógica, procesos, despliegue e implementación.
XP	ágil	evolutivo	Ignora la arquitectura, su ruta es ir hacia la codificación rápida y confiar en que la refactorización resolverá todos los problemas de diseño.
MSF Ágil	ágil	mixto(planeado, evolutivo)	Asigna mayor relevancia a la Administración de Riesgos.
FDD	ágil	evolutivo	Ignora la arquitectura, puesto que se centra en construir pequeños sistemas funcionales de forma iterativa y no logra ver el sistema de forma global.

Tabla 5: Resumen de las Metodologías de Desarrollo en cuanto a diseño arquitectónico

A modo de resumen se puede decir que existen tres tendencias en cuanto al diseño de software: diseño planeado, evolutivo y mixto.

Tanto el diseño planeado como el evolutivo presentan ventajas y desventajas.

[4]Class Owner: Trabaja bajo la guía del Chief Programmer en las tareas de diseño, codificación, pruebas y documentación.

El diseño planeado permite a los diseñadores trabajar a un alto nivel de abstracción y obtener una visión global del sistema a construir desde etapas tempranas de desarrollo. Este diseño detallado constituye una entrada fundamental en el proceso de construcción del software.

Sin embargo, no siempre es posible conocer todas las funcionalidades que debe realizar el software en etapas iniciales, por lo que en ocasiones los diseños planificados con anterioridad no satisfacen las nuevas necesidades que van surgiendo.

Por otra parte con el diseño evolutivo la especificación puede desarrollarse de forma creciente y los usuarios y desarrolladores logran un mejor entendimiento del sistema. Este tipo de diseño además cumple con las necesidades inmediatas del cliente, aunque en ocasiones presenta algunos problemas, como por ejemplo:

- ✓ Sistemas pobremente estructurados: Los cambios continuos pueden ser perjudiciales para la estructura del software haciendo costoso el mantenimiento.
- ✓ Se requieren técnicas y herramientas: Para el rápido desarrollo se necesitan herramientas que pueden ser incompatibles con otras o que poca gente sabe utilizar.

También se puede decir que estas tendencias no son totalmente excluyentes entre sí.

El enfoque mixto da la posibilidad de minimizar las desventajas de ambos tomando lo mejor de cada uno y evitando rechazar totalmente los aspectos positivos de los dos enfoques.

La propuesta de solución se centrará en el diseño mixto.

Por otra parte se puede decir que se hace necesario definir un proceso de diseño arquitectónico que lleve a la par la administración de riegos que propone MSF Ágil, así como establecer la arquitectura del sistema como centro para el desarrollo del mismo tal y como plantea RUP, de esta forma se obtendrá una visión común del sistema y los cimientos del mismo que son necesarios como base para comprenderlo, desarrollarlo y producirlo económicamente, y a la misma vez realizar ininterrumpidamente valoraciones de riesgos que incidan en la toma de decisiones durante el ciclo de vida del proyecto.

1.11 Software de Realidad Virtual

El desarrollo de la informática en los últimos tiempos ha traído consigo, el surgimiento y desarrollo de un nuevo término “Realidad Virtual”, iniciada en los programas de entrenamiento militares, simuladores

de vuelo, centros de investigación y académicos, ha pasado a los programas más variados en el ámbito profesional y doméstico. Hoy se empieza a aplicar en ingeniería, medicina, arquitectura, educación, juegos, entre otros campos.

En la actualidad es difícil dar una definición concreta de Realidad Virtual, existen muchas definiciones, quizás tantas como investigadores relacionados con esta área, esto se debe en gran medida a su reciente y rápida evolución. No resultaría extraño que el concepto de Realidad Virtual suela ser relativo para diferentes investigadores del tema en diferentes situaciones.

Una definición bastante completa es la de (Aukstakalnis, 1992) "La realidad virtual es un camino que tienen los humanos para visualizar, manipular e interactuar con computadoras y con información extremadamente compleja".

A modo de resumen se puede decir que un software de RV es un software que permite al usuario operar en un entorno gráfico 3D con el uso, o no, de dispositivos de hardware especiales diferentes a la PC tradicional (la que contiene teclado, monitor, mouse y bocinas para interactuar con el usuario).

1.12 Valoración de las propuestas realizadas en trabajos de diploma de la Facultad 5

En la Facultad 5 se han desarrollado 4 trabajos de diploma los cuales aportan aspectos positivos a la producción de software de RV, por tanto se hace necesario realizar un análisis de los mismos.

1.12.1 Propuesta de estrategia ágil para el desarrollo de videojuegos.

(Bauta Gómez & Castillo Barbosa, 2007) en su trabajo de diploma plantean que para el desarrollo de videojuegos las metodologías ágiles brindan mayores beneficios que las tradicionales, debido a que en las mismas se valora al individuo y las interacciones del equipo de desarrollo más que al proceso y las herramientas, que además tienen en cuenta que desarrollar software funciona más que conseguir una buena documentación y que responder a los cambios resulta más efectivo que seguir estrictamente un plan.

(Bauta Gómez & Castillo Barbosa, 2007) concluyen que la metodología MSF ágil es la que más se ajusta a las características del grupo de desarrollo de juegos virtuales de la Facultad 5 de la UCI debido a que la misma cabe en el marco de las soluciones de Microsoft como un sistema integrado de la dirección de procesos que abraza metodologías ágiles y formales y proporciona un espacio para poner una solución en ejecución, modificada para requisitos particulares de una amplia variedad de proyectos.

Consideraciones: Teniendo en cuenta que no existe proceso de desarrollo de software que se adecúe a todo tipo de proyecto, se considera que realmente MSF ágil es una metodología que se puede utilizar para el desarrollo de software de RV. Coincidiendo con lo planteado anteriormente por (Bauta Gómez & Castillo Barbosa, 2007), MSF ágil se compone de las mejores características de las metodologías ágiles y tradicionales por lo que el diseño mixto que la misma propone, satisface en gran medida las necesidades del proceso de diseño arquitectónico para software de RV.

1.12.2 Adaptación de Microsoft Solutions Framework Agile (MSF Ágil) al proceso de desarrollo de videojuegos.

Según (Arias & Martínez, 2008) MSF Ágil constituye el método más adaptable a proyectos de desarrollo de videojuegos, debido a que esta metodología recopila las mejores prácticas de los procesos de desarrollo tradicionales y ágiles. En este trabajo de diploma se exponen las causas del por qué es más conveniente la utilización de la metodología MSF Ágil que RUP en el proyecto Juegos de Consola.

A continuación se realiza una valoración de los argumentos planteados que aporten aspectos positivos a la propuesta de solución:

Documentación excesiva vs. Conocimiento implícito

(Arias & Martínez, 2008) plantea que cuando un equipo de desarrollo es pequeño, cohesionado y todo el personal trabaja en un mismo local, hay que tener claro que mucha documentación no es la forma más efectiva para alcanzar el objetivo. Un equipo con estas características es posible que necesite poco más que una enunciación de la visión, un listado de escenarios y una frecuente comunicación cara a cara con el cliente. Plantean que la clave es escribir documentación exclusivamente para su audiencia, enfocada a los propósitos específicos del proyecto y luego dejar de escribir documentación. De esta forma acumular el esfuerzo y recursos para emplearlos en la codificación.

Consideraciones: Se coincide con lo expuesto anteriormente, puesto que realmente se hace necesario evitar la documentación innecesaria dentro de equipos de desarrollo pequeños y enfocar esta fundamentalmente a los consumidores finales, aunque no se puede dejar de tener en cuenta que existe documentación que no debe faltar, como por ejemplo las especificaciones de RF y RNF, debido a que la elaboración de estos documentos evita discrepancias entre el equipo de desarrollo y los clientes.

Estructura jerárquica vs. Modelo de equipo que propone MSF Ágil

Según (Arias & Martínez, 2008) para el desarrollo de videojuegos se necesitan orientaciones más que reglas, debido a que esta rigidez y control en la organización del equipo provoca la resistencia al cambio. El modelo de equipo que propone MSF Ágil comparte flexiblemente los roles y las responsabilidades, de forma que todos los roles tienen la misma importancia dentro del equipo. La información fluye fácilmente entre los miembros del equipo, se promueve la combinación de diferentes puntos de vista y alienta la agilidad para hacer frente a nuevos cambios involucrando a todo el equipo en las decisiones fundamentales, por esto los individuos se ven incentivados a participar activamente en los proyectos al sentirlos como propios.

Consideraciones: Se considera que aunque es muy beneficioso que un equipo de desarrollo se encuentre integrado y todos los roles deben sentir el proyecto como propio, siempre es necesario que existan personas o roles que controlen el trabajo del resto del equipo, puesto que no siempre se debe confiar en que todo se realizará según lo planificado.

Centrado en la arquitectura vs. Mayor administración de riesgos

Según (Arias & Martínez, 2008) si un proyecto necesita responder rápidamente a la presión competitiva de negocio o la usabilidad, como es el caso de los videojuegos, no se puede diseñar perfectamente ni completamente desde el inicio, es por esto que se hace el diseño sobre la marcha del mismo, teniendo en cuenta nuevos requisitos. MSF Ágil se enfoca principalmente en la mitigación de los riesgos en cada uno de los grupos de trabajo del modelo de equipo, con el fin de lograr una arquitectura robusta. La administración de riesgos es una disciplina básica dentro de MSF Ágil, reconoce que los cambios y la incertidumbre que estos generan son aspectos inherentes del ciclo de vida de un proyecto.

Consideraciones: Se considera que es importante plantear una arquitectura base que permita obtener una visión global del sistema a construir desde etapas tempranas de desarrollo del software aunque la misma debe ser diseñada de tal forma que soporte posibles cambios futuros de los RF. También se deben administrar los riesgos más importantes que incidan en la toma de decisiones significativas.

1.12.3 Determinación de los roles, responsabilidades y conocimientos necesarios para el Proceso de Producción de Software de RV

En este trabajo de diploma (Pita & Álvarez, 2007) definen los roles necesarios para desarrollar un software de RV, así como sus responsabilidades correspondientes. Los roles que intervienen en el proceso de diseño son:

Guionista: Se encarga de hacer el guión de una tarea en cuestión.
Modelador: Modela las escenas, modelos y objetos necesarios para la tarea.
Diseñador gráfico: Define el diseño gráfico de la aplicación, dibuja las escenas, modelos y objetos necesarios para la tarea.
Diseñador de Base de datos: Diseñar la base de datos y garantizar la integridad referencial, responsable de las actualizaciones, correcciones y mantenimiento de la Base de Datos del Sistema.
Animador: Es el encargado de hacer las animaciones de las tareas (animaciones de cualquier vértice que se mueva).
Sonidista: Responsable de la edición y realización de los sonidos.
Editor: Es el encargado de unir todos los fragmentos de videos en una sola línea y de mezclarlos con el ambiente sonoro.
Diseñador de efectos especiales: Encargado de realizar todos los efectos especiales necesarios en la tarea.
Arquitecto de Software

- Comprueba la viabilidad del concepto de la prueba arquitectónica.
- Formula la prueba arquitectónica.
- Incluye elementos de diseño existentes.
- Identifica mecanismos de diseño.
- Identifica elementos del diseño.
- Encargado de priorizar casos de uso.
- Responsable de la descripción de la arquitectura (vista del modelo de casos de uso).
- Responsable del análisis, diseño e implementación de la arquitectura.
- Responsable de la realización del modelo de análisis.
- Responsable del modelo de diseño, modelo de despliegue y modelo de implementación

Consideraciones: Se considera que los roles propuestos se adecúan a las características específicas de los software de RV de manera general, aunque, se hace necesario llegar a un consenso entre los roles planteados en la metodología que se va a utilizar y las necesidades reales del proyecto.

1.12.4 Propuesta de Proceso de Software para sistemas de RV

En el trabajo de diploma (Trujillo, 2007) define el proceso de desarrollo para sofware de RV constituido por 30 actividades con el objetivo de lograr un trabajo organizado. En las actividades propuestas se establecieron los responsables, participantes, la misión, la entrada y la posible salida en forma de artefactos.

Las actividades 23, 24 y 25 son las relacionadas con los procesos de análisis y diseño.

Actividad 23: Análisis y Diseño.
Responsable: Analista Principal.
Participantes: Analista Principal, Diseñador, Arquitecto principal, Diseñador de Base de Datos, Jefe de Proyecto
Misión: Se realiza el análisis para investigar el problema al cual se le planteará una solución lógica en el diseño.
Entradas:

- Descripción de Casos de Uso.

Salidas:

- Propuesta de la Arquitectura.
- Diseño de interfaz de usuario.
- Documentos de realización de Casos de Uso.
- Diagrama de Clases (Análisis y diseño).
- Modelo de datos.
- Diagrama de despliegue.
- Documento final de arquitectura.

Actividad 24: Revisión técnica formal (interna).
Responsable: Jefe de Calidad.
Participantes: Jefe de Calidad, Analista Principal, Diseñador, Arquitecto principal, Diseñador de Base de Datos, Jefe de Proyecto.

Misión: Revisión interna con los encargados con la actividad de análisis y diseño y el Jefe del grupo de Calidad para comprobar la calidad de la misma.

Entradas:

- Propuesta de la Arquitectura.
- Diseño de interfaz de usuario.
- Documentos de realización de Casos de Uso.
- Diagrama de Clases.
- Modelo de datos.
- Diagrama de despliegue.
- Documento final de arquitectura.
- Glosario de Términos

Salidas:

- Glosario de Términos.
- Acta de la reunión.
- Documento de no conformidades.

Actividad 25: Revisión con el cliente.

Responsable: Jefe de Proyecto.

Participantes: Jefe de Proyecto, Analista principal, Jefe del grupo de Calidad.

Misión: Revisión con el cliente de los documentos y artefactos generados por el análisis y diseño

Entradas:

- Propuesta de la Arquitectura.
- Diseño de interfaz de usuario.
- Documentos de realización de Casos de Uso.
- Diagrama de Clases.
- Modelo de datos.
- Diagrama de despliegue.
- Documento final de arquitectura.
- Glosario de Términos.

Salidas:

- Glosario de Términos.
- Documento de no conformidades (Registros).

- Acta de aceptación del cliente.

Consideraciones: El proceso de diseño de este trabajo se encuentra bastante completo aunque se considera que para utilizar el mismo para el desarrollo de un proyecto específico habría que valorar si los roles propuestos realmente se adaptan a las necesidades existentes en el mismo y sería bueno también valorar la reducción de la documentación interna del proyecto.

1.13 Proyectos de Realidad Virtual de la Facultad 5

La Universidad de las Ciencias Informáticas, es una universidad productiva. La producción de software y servicios informáticos se basa en la integración de los procesos de formación, investigación y producción en torno a una temática para convertirla en una rama productiva. La misma está conformada por 10 facultades.

La facultad 5 cuenta con 2 polos productivos dedicados al desarrollo de software, el polo de Hardware y Automática y el de Realidad Virtual, este último cuenta con 14 proyectos (Ver Anexo 1).

Se realiza un estudio de los proyectos pertenecientes al polo de Realidad Virtual centrado en los siguientes objetivos:

- ✓ Determinar cómo se realiza hoy el proceso de Diseño de la Arquitectura.
- ✓ Determinar si existe algún aspecto del proceso actual que pudiera afectar los parámetros de costo, tiempo de desarrollo y calidad de los productos (software) resultantes.

Para alcanzar estos objetivos se diseñaron un conjunto de herramientas:

Entrevistas: a los Arquitectos de Software, Diseñadores y Programadores de cada proyecto. (Ver Anexo 2)

Análisis de documentos: Documentos de Arquitectura y Especificación de Requisitos de cada proyecto. (Ver Anexo 3)

1.14 Conclusiones Parciales

A partir de los objetivos propuestos para este capítulo definidos en la introducción se puede decir que:

- ✓ Se realizó un estudio de las Metodologías de Desarrollo de Software, centrando en el proceso de Diseño Arquitectónico.

- ✓ Se explicaron además los distintos niveles de abstracción del diseño resaltando la Arquitectura del Software como el diseño de más alto nivel.
- ✓ Se analizó el proceso de Diseño Arquitectónico de las Metodologías de Desarrollo de Software que se consideran son las más utilizadas (RUP, XP, FDD y MSF Ágil).
- ✓ Se establecieron una serie de herramientas y parámetros para analizar el estado actual del Diseño Arquitectónico de los proyectos de RV que se desarrollan en la facultad 5.

Concluyendo que:

- ✓ Los principales enfoques de las Metodologías de Desarrollo de Software son tradicional y ágil.
- ✓ No existe una definición estandarizada del concepto Arquitectura de Software.
- ✓ Cada Metodología de Desarrollo de Software enfoca la Arquitectura de una forma diferente.
- ✓ Las Metodologías de Desarrollo de Software tienden a organizar esta arquitectura en términos de Vistas.
- ✓ Estas Vistas son representadas utilizando Lenguajes de Descripción Arquitectónica, mediante modelos o diagramas.
- ✓ El Lenguaje más utilizado para representar la descripción arquitectónica es el UML.
- ✓ Las metodologías ágiles hacen menos énfasis en la Arquitectura de Software que las tradicionales.
- ✓ Las Metodologías de Desarrollo (RUP, XP, FDD y MSF Ágil), se centran en tres tipos de diseño: planeado, evolutivo, y mixto.
- ✓ El diseño mixto engloba las ventajas del diseño planeado y evolutivo.
- ✓ Se considera que el diseño mixto es el que más se ajusta para diseñar la arquitectura de los software de RV.

- ✓ Se hace necesario un proceso de diseño arquitectónico que combine de forma moderada la Administración de Riesgos que propone MSF Ágil, así como establecer la arquitectura del sistema como centro para el desarrollo del mismo tal y como plantea RUP.

CAPÍTULO 2

Propuesta de Solución

2.1 Introducción

En el presente capítulo se realiza un análisis del estado actual de los proyectos de RV que se realizan en la Facultad 5 en cuanto a su Diseño Arquitectónico (metodologías utilizadas, entradas y salidas del diseño, etc.)

Se describen además cada uno de los elementos que componen el Proceso de Diseño Arquitectónico para Software de RV (actividades, entradas, salidas, etc.)

2.2 Análisis del proceso de Diseño Arquitectónico en los Proyectos de RV que pertenecen a la Facultad 5

Con el objetivo fundamental de obtener información acerca del estado actual del proceso de diseño arquitectónico en los proyectos que pertenecen al polo de RV de la Facultad 5 en la Universidad de las Ciencias Informáticas, se hace necesario diseñar un conjunto de herramientas mediante las cuales se podrán determinar tanto deficiencias como aspectos positivos del mismo. Este análisis contribuirá en gran medida a la propuesta de Proceso de Diseño Arquitectónico para Software de RV, debido a que la misma se elaborará teniendo en cuenta los problemas existentes en el proceso actual, a los cuales se pretende dar solución y de esta forma elevar la calidad del mismo.

Para dar cumplimiento al objetivo trazado anteriormente se realizó un estudio preliminar de los 14 proyectos (100%) que pertenecen al polo de RV de la Facultad 5 y se pudo detectar lo siguiente:

- ✓ Los proyectos: Meñique y Escenarios Virtuales son proyectos de Diseño y Animación que no requieren de los procesos tradicionales de la Ingeniería de Software, los mismos están vinculados a la producción de diseño gráfico.
- ✓ Los proyectos: Simulador de Tiro, Laboratorios de Realidad Aumentada y Laboratorios Virtuales, son muy recientes y no han comenzado el trabajo en cuanto a diseño arquitectónico, por tanto no aportan datos significativos a esta investigación y se decidió no incluirlos en la muestra estudiada.

- ✓ El proyecto Simulador de Autos no se encuentra produciendo en estos momentos. Todo el personal que lo desarrollaba fue cambiado y el personal actual se encuentra en una fase inicial de preparación. Además no cuenta con documentación significativa aparte del código fuente. Por todo esto se decidió excluirlo de la muestra.

En resumen, se analizan los 8 proyectos restantes, constituyendo un 57% de la población estudiada.

2.3 Entrevistas a los Arquitectos de Software

Se realizó una entrevista a los Arquitectos de Software de cada proyecto de RV (Ver preguntas de la entrevista: Anexo 2). Los objetivos propuestos con la misma, así como las preguntas que tributan a cada objetivo y la especificación de las correspondientes respuestas en los casos necesarios se pueden observar en los Anexo 3.

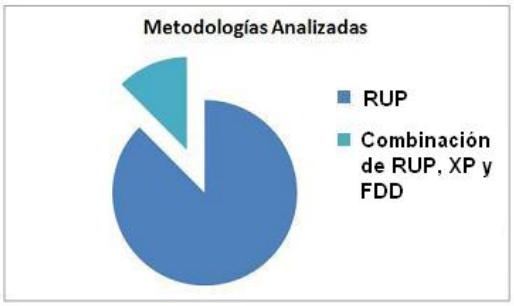
2.3.1 Resultados de las Entrevistas realizadas a los Arquitectos de Software

Luego de la aplicación de los instrumentos descritos anteriormente (Ver respuestas de los Arquitectos de Software: Anexo 5), y el procesamiento de la información obtenida se pudo llegar a las conclusiones siguientes:

- ✓ En un 50% de los proyectos analizados los Arquitectos de software no desempeñan las actividades definidas para su rol y estas responsabilidades son asumidas por los líderes de proyecto. En estos casos las entrevistas se les realizaron a los líderes de proyecto. (Ver tabla de Otras Observaciones: Anexo 8)

- ✓ En la totalidad de los proyectos analizados es necesario que se especifique la Arquitectura de Software.

- ✓ La siguiente gráfica muestra las Metodologías de Desarrollo de Software utilizadas en los proyectos analizados, donde se puede observar que para un 87.5% se utiliza el Proceso Unificado de Desarrollo, esto se debe principalmente a la falta de un estudio para encontrar la metodología que más se adecúe a este tipo de software.



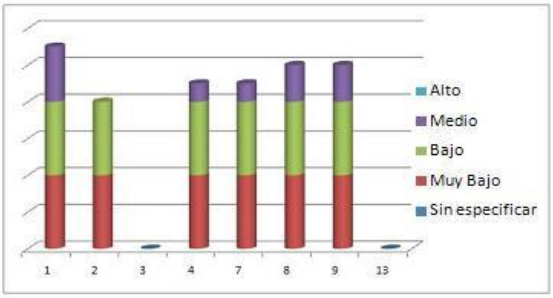
Gráfica 1: Metodologías de Desarrollo utilizadas en los proyectos de RV analizados.

Como resultado del intercambio sostenido con los miembros de los proyectos analizados, se detectó además que la mayor parte de estos proyectos requieren ser terminados en períodos relativamente cortos de tiempo para la complejidad de su desarrollo y además teniendo en cuenta que no se debe afectar la calidad del producto final. Por este motivo la mayor parte de los arquitectos de estos proyectos plantean que para este tipo de software el proceso de desarrollo propuesto por RUP en ocasiones se torna muy largo y genera documentación excesiva lo cual afecta el tiempo de desarrollo del proyecto. El proceso de diseñar la arquitectura es uno de los más afectados por estos problemas.

Se considera que la principal causa del rechazo que existe en los proyectos de RV de la Facultad 5 hacia la metodología RUP es que los arquitectos de software y líderes de proyecto intentan aplicarla con todos sus pasos y actividades, sin tener en cuenta que esta metodología no ha sido diseñada específicamente para este tipo de software. Teniendo en cuenta que no existe aún la metodología universal, cualquier metodología o framework de proceso para desarrollo de software necesita ser adaptado a las condiciones específicas del proyecto y de su personal. Por tanto, si se va a utilizar RUP en un proyecto de este tipo, se debe realizar un ajuste del mismo para aprovechar al máximo las características que aporten aspectos positivos al proceso de desarrollo de software de RV.

- ✓ Se puede decir además que en la totalidad de los proyectos analizados los artefactos utilizados como entrada al diseño son las Especificaciones de Requisitos Funcionales (RF) y No Funcionales (RNF). Como artefacto de salida solamente se genera el Documento de la Arquitectura.

- ✓ La siguiente gráfica muestra el nivel de especificación de la Arquitectura de Software en los proyectos analizados, donde se puede observar que para un 62.5% de los mismos el nivel de detalle es medio, para un 12.5% es bajo y para un 25% la Arquitectura de Software no se especifica formalmente.



Gráfica 2: Nivel de especificación de la Arquitectura en los proyectos de RV analizados.

- ✓ Se concluye además que en el 75% de los proyectos se diseña la arquitectura de software mientras se escribe el código o se realiza esta actividad una vez concluida la programación del software, esto sucede principalmente por una de las siguientes causas:
  - Se propone una arquitectura inicial que finalmente no satisface las necesidades del software, lo que provoca cambios de última hora cuando se dispone de corto tiempo y la solución más rápida es programar y después realizar la ingeniería.
  - Los líderes de proyecto consideran más oportuno realizar la programación y dejar las formalidades para más tarde.
  - Los requisitos son muy cambiantes por lo que es difícil tener una visión global del sistema completo desde el inicio. La metodología RUP no tiene en cuenta estos aspectos por lo que el proceso de diseño se torna complicado y se decide pasar directamente al código e ir diseñando sobre la marcha o una vez concluida la programación.
- ✓ Se puede decir además que el 62.5% de los Arquitectos de Software de estos proyectos posee poca experiencia y escasos conocimientos del alcance de la responsabilidad del rol que desempeñan.

2.4 Entrevistas realizadas a los Diseñadores

Se realizó una entrevista a los Diseñadores de cada proyecto de RV (Ver preguntas de la entrevista: Anexo 2). Los objetivos propuestos con la misma, así como las preguntas que tributan a cada objetivo y la especificación de las correspondientes respuestas en los casos necesarios se pueden observar en los Anexo 3.

2.4.1 Resultados de las Entrevistas realizadas a los Diseñadores

Luego de la aplicación de los instrumentos descritos anteriormente (Ver respuestas de los Diseñadores: Anexo 6), y el procesamiento de la información obtenida se pudo llegar a las conclusiones siguientes:

- ✓ La siguiente gráfica muestra la información de entrada utilizada para comenzar el diseño en los proyectos analizados, donde se puede observar que se utilizan los RF en un 75% de los mismos, el 25% restante realiza el diseño a partir de la combinación de los RF y los guiones técnicos.

Gráfica 3: Información de entrada para comenzar el diseño en los proyectos de RV analizados.

- ✓ Se puede decir que los diseñadores de estos proyectos consideran útil el Documento de Arquitectura así como el nivel de especificación del mismo, aunque un 37.5% de ellos aboga porque este documento se ajuste a las características específicas del tipo de software que se desarrolla.

- ✓ Sin embargo, el 87.5% de estos diseñadores no trabaja a partir del Documento de Arquitectura, debido a uno de los siguientes factores:

  - El documento no está listo cuando se va a comenzar a diseñar.
  - El diseño se realiza a partir del código fuente y el documento se elabora más tarde como formalidad para entregar el producto.

2.5 Entrevistas realizadas a los Programadores

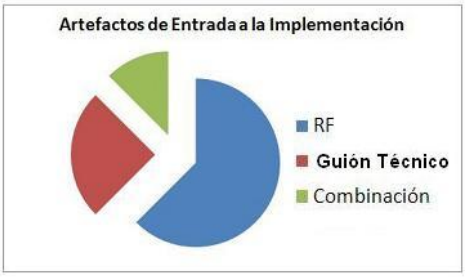
Se realizó una entrevista a los Programadores de cada proyecto de RV (Ver preguntas de la entrevista: Anexo 2). Los objetivos propuestos con la misma, así como las preguntas que tributan a cada objetivo

y la especificación de las correspondientes respuestas en los casos necesarios se pueden observar en los Anexo 3.

2.5.1 Resultados de las Entrevistas realizadas a los Programadores

Luego de la aplicación de los instrumentos descritos anteriormente (Ver respuestas de los Programadores: Anexo 7), y el procesamiento de la información obtenida se pudo llegar a las conclusiones siguientes:

- ✓ La siguiente gráfica muestra la información de entrada utilizada para comenzar la implementación en los proyectos analizados, donde se puede observar que se utilizan las especificaciones de requisitos en un 62.5%, los guiones técnicos en un 25% y la combinación de ambos en el 12.5% restante.



Gráfica 4: Información de entrada para comenzar la implementación en los proyectos de RV analizados.

- ✓ Se puede decir además que la totalidad de los programadores de estos proyectos consideran útil el Documento de Arquitectura así como el nivel de especificación del mismo, puesto que les facilita el trabajo en gran medida.

- ✓ Sin embargo, el 87.5% de los programadores de los proyectos analizados no trabaja a partir del Documento de Arquitectura y no conocen de la existencia del mimo en su proyecto, esto se debe principalmente a la tendencia prácticamente generalizada que existe actualmente en estos proyectos de realizar la programación del software antes que el diseño de alto nivel del mismo, lo que trae como consecuencia sobrecarga de trabajo para este rol e inconformidades en los productos finales.

2.6 Conclusiones de las Entrevistas Realizadas

Para resumir los resultados de las entrevistas realizadas, tanto a arquitectos de software, como a diseñadores y programadores se puede decir que se hace necesario que en este tipo de software se realice el diseño arquitectónico siguiendo un proceso que contribuya a entregar soluciones de software de alta calidad, a tiempo y dentro del presupuesto, y que a la vez posea las siguientes características:

- ✓ Tener en cuenta los posibles cambios futuros de los requisitos del software.
- ✓ Debe ser un proceso relativamente corto que genere una documentación concreta y a la vez útil para todo el equipo de desarrollo.
- ✓ Debe permitir un nivel alto de especificación de la arquitectura del software que no afecte el tiempo de desarrollo del proyecto y a la vez contribuya a facilitar el trabajo de los diseñadores y programadores.

Se hace necesario además que el rol de Arquitecto de Software sea desempeñado por una persona que posea los conocimientos y la experiencia necesaria para desarrollar las actividades del proceso de diseño arquitectónico, de esta forma los líderes de proyecto no se verán comprometidos con estas tareas y el proceso se desarrollará con mayor calidad.

2.7 Análisis de Documentos

En el Documento de Arquitectura de Software se describen cada una de las vistas de la Arquitectura de un sistema, se explican y fundamentan los elementos arquitectónicos seleccionados, se describen las restricciones de diseño e implementación, así como los patrones de diseño utilizados, entre otras cuestiones según el software a desarrollar en particular.

Por otra parte en el Documento de Especificación de Requisitos se especifican cada una de las funcionalidades que va a poseer el software, así como las necesidades de los stakeholders con respecto al rendimiento, escalabilidad, mantenimiento, seguridad, entre otros. Es por este motivo que este documento es vital como entrada al proceso de diseño arquitectónico.

Teniendo en cuenta la importancia de ambos documentos se realizó una recopilación de los mismos en los proyectos que pertenecen al polo de RV de la Facultad 5 con el objetivo general de determinar el nivel de detalle y especificación del diseño de alto nivel en los mismos, así como la correcta redacción de los requisitos, fundamentalmente los no funcionales claves para la arquitectura. (Ver

objetivos específicos del análisis de documentos así como los parámetros que tributan a cada objetivo: Anexo 4)

2.7.1 Resultados obtenidos del análisis de documentos

Luego de la recopilación de los Documentos de Arquitectura de Software y Especificación de los Requisitos de los 8 proyectos que se analizan en la investigación, y el procesamiento de la información obtenida (Ver resultados del análisis de documentos: Anexo 9) se pudo llegar a las conclusiones siguientes:

- ✓ El 37.5% de los proyectos analizados no cuenta con el Documento de la Arquitectura de Software. (Ver tabla de Otras Observaciones: Anexo 8)
- ✓ El total de los proyectos analizados cuenta con el Documento de Especificación de Requisitos, aunque el 12.5% no posee la descripción de los RNF.

A continuación se muestran los resultados obtenidos según los objetivos definidos:

Objetivo 1: Especificación y Claridad de los RF y RNF prioritarios o críticos para la Arquitectura.

- ✓ En el 62.5% de los documentos analizados se encuentran identificados los CU y/o RF críticos para la arquitectura. La priorización se adecúa al tipo de software que se desarrolla.
- ✓ Los RF se encuentran claramente definidos y son posibles de verificar para el 100% de los proyectos analizados.
- ✓ Los RNF presentan algunos problemas de ambigüedad y son imposibles de verificar, fundamentalmente los de Eficiencia y Rendimiento, para un 37.5% de los proyectos analizados.

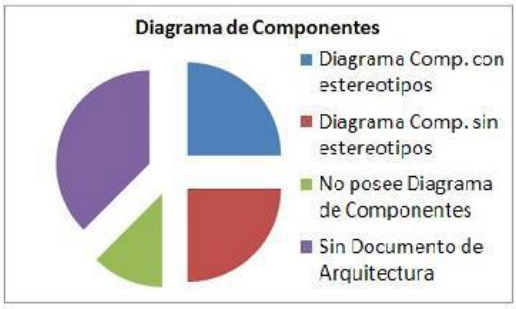
Objetivo 2: Nivel de detalle de la Vista Lógica o estructural.

- ✓ En un 62.5% de los proyectos analizados se especifican los elementos o clases significativos para la arquitectura, así como su organización en paquetes o subsistemas.
- ✓ Solamente en un 37.5% de los estos proyectos se describen cada uno de los elementos o clases significativos para la arquitectura.

- ✓ El estilo de agrupación utilizado es el Modelo 3 capas para un 50% de los proyectos analizados, encontrándose estas capas agrupadas a su vez en paquetes o subsistemas. El 12.5% solo agrupa utilizando paquetes o subsistemas.
- ✓ En ningún caso se fundamenta la elección del estilo de agrupación y solamente en un 12.5% se especifican las relaciones entre las agrupaciones.

Objetivo 3: Nivel de detalle de la Vista de Implementación.

- ✓ La gráfica muestra los resultados del análisis de la vista de implementación, donde se puede observar que para un 25% de los proyectos analizados se especifican los componentes con sus respectivos estereotipos, para otro 25% no se especifican los estereotipos y un 12.5% no cuenta con el diagrama de componentes.



Gráfica 5: Diagramas de Componentes

- ✓ Se puede decir además que en los proyectos donde se especifican los componentes de la vista de implementación, los cuales constituyen el 50% de la muestra analizada, no se especifican las relaciones entre los mismos.

Objetivo 4: Nivel de detalle de la Vista de Despliegue.

- ✓ En un 12.5% de los proyectos analizados no es necesario que se especifiquen los elementos de despliegue y en un 37.5% se especifican los mismos. Solamente en un 25% de estos proyectos se describen las relaciones entre estos elementos.
- ✓ Ningún proyecto especifica el mapeo o relación de los elementos de despliegue con los componentes de implementación.

Objetivo 5: Nivel de detalle de la Vista de Datos.

En la totalidad de los proyectos analizados no se cuenta con una gran cantidad de elementos persistentes, por lo que no es necesario contar con grandes bases de datos.

Solamente en un proyecto es necesario que se especifique la Vista de Datos. La descripción de la misma se realiza de forma escasa y no muestra el Modelo de Datos.

Objetivo 6: Restricciones Arquitectónicas.

- ✓ En un 62.5% de los proyectos analizados se describen las restricciones arquitectónicas de forma muy escasa.
- ✓ En el 25% de estos proyectos no se describen los RNF significativos para la arquitectura y en un 37.5% de los mismos se realiza de forma escasa, se hace de forma general y no se especifica cada restricción por requisito.

2.8 Conclusiones del Análisis de Documentos

Luego del análisis de los Documentos de Arquitectura y Especificación de Requisitos de los proyectos de RV de la Facultad 5, se concluye reafirmando las carencias actuales en el proceso de diseño arquitectónico expuestas luego de la realización de entrevistas a los Arquitectos de Software, Diseñadores y Programadores de estos proyectos. La principal deficiencia existente se centra en el rechazo actual que se le ha ido creando al diseño de más alto nivel en estos proyectos, donde por lo general se da un gran salto desde la especificación de los requisitos y la elaboración de los guiones técnicos hacia la codificación, ya sea por falta de tiempo o por no lograr la adaptación de una metodología determinada a cada proyecto según sus necesidades.

En la mayor parte de estos proyectos se redactan los Documentos de Arquitectura después de concluida la programación del software solo por formalidad, lo que trae como consecuencia que el documento no posea la calidad requerida, ni se cumplan los objetivos propuestos con la realización de este artefacto, debido a que no facilita el trabajo de los diseñadores y programadores en la construcción de un producto de mayor calidad final.

Actualmente en estos proyectos se hace necesario que los RF y RNF sean especificados de forma clara, concreta y sean además comprobables, de esta forma se podrán tener en cuenta para definir la arquitectura del sistema y se podrá evaluar esta arquitectura en función de los requisitos.

El documento de arquitectura de software debe ajustarse a las características específicas de un software de RV y poseer descripciones breves, concretas y útiles para actividades posteriores como el diseño detallado y la implementación.

2.9 Propuesta de Proceso de Diseño Arquitectónico para Software de Realidad Virtual.

En la actualidad los sistemas informáticos son cada vez más complejos, por lo que para construir un software de alta calidad final y que satisfaga los requisitos especificados por el cliente se hace necesario realizar una serie de actividades organizadas antes de llegar a la implementación del mismo. La arquitectura a definir juega un papel muy importante en el resultado final entregado a los usuarios, debido a que dependiendo de su definición y diseño, no sólo el desempeño del sistema, sino los objetivos planteados en el negocio, pueden ser llevados a un fin satisfactorio.

En la Facultad 5 de la Universidad de las Ciencias Informáticas no se cuenta con una guía que especifique los pasos a seguir para desarrollar el diseño de alto nivel para la producción de software de RV y que tenga en cuenta las características particulares de los mismos.

Luego de realizar un análisis del proceso de diseño arquitectónico planteado en diferentes metodologías de desarrollo de software, tanto ágiles como tradicionales, y después de detectar las principales deficiencias con que cuenta hoy el proceso en los proyectos existentes en la Facultad 5 pertenecientes al polo de RV, se procede a describir en este capítulo una propuesta que brinde solución a los problemas planteados y que se ajuste a las características específicas de los software de RV.

El proceso se encuentra constituido por 3 actividades, cada una de éstas se realiza a través de una serie de pasos definidos y a partir de información de entrada genera un conjunto de salidas con un valor agregado. El proceso se caracteriza además por ser iterativo e incremental, al aplicar este modelo se mejoran y agregan funcionalidades para cada iteración, lo que permite crear cada vez versiones más completas del software. El proceso tiene en cuenta algunos de los principios definidos para las metodologías ágiles, de planificación adaptable a los cambios, por lo que permite capturarlos y

adaptarse a los mismos. Es un proceso relativamente corto, por lo que permite obtener el diseño arquitectónico de un sistema realizando una pequeña cantidad de pasos. Aboga por la elaboración de diseños sencillos, con la menor complejidad posible, donde se dedique la mayor parte del esfuerzo al diseño y construcción de la parte gráfica del entorno virtual. Propone además que la escritura de documentación sea adaptable a las características del equipo de desarrollo.

Se toma como principal referencia el proceso de diseño arquitectónico propuesto por RUP puesto que esta metodología es la que hace mayor énfasis en la arquitectura y en el diseño de una visión inicial que permita la organización de las partes más relevantes de un sistema y a través de la cual todos los involucrados en el desarrollo del proyecto puedan tener una visión común y una perspectiva clara del sistema completo. Se sintetizaron las actividades, tareas y pasos propuestos por RUP para el diseño de más alto nivel, tomando solamente aquellos que fuesen indispensables para el diseño arquitectónico y de esta forma dar agilidad a este proceso.

Por lo general, en los proyectos pertenecientes al polo de RV de la facultad 5 se descuidan los riesgos asociados a la arquitectura y a la toma de decisiones arquitectónicas, por este motivo se incorporaron al proceso algunos de los pasos de la disciplina de Administración de Riesgos que propone la metodología MSF Ágil con la intención de asignar al arquitecto de software la principal responsabilidad de la identificación, mitigación, monitoreo y administración de los riesgos técnicos. Los riesgos relacionados con la arquitectura son lógicamente los más críticos en un proyecto y amenazan la calidad temporal del software debido a que si alguno se materializa, la implementación puede llegar a ser difícil o imposible.

El proceso definido propone además una guía elemental de los pasos a seguir para evaluar la arquitectura de software en una etapa temprana del proyecto y de esta forma minimizar los riesgos asociados a la misma.

Se incorporan diferentes prácticas planteadas en las metodologías ágiles estudiadas. Para realizar el diseño de alto nivel del software se podrán seleccionar aquellas prácticas que realmente agreguen valor al proceso y tenerlas en cuenta para la realización de las actividades.

La siguiente figura muestra los elementos que componen el proceso así como las relaciones entre estos:

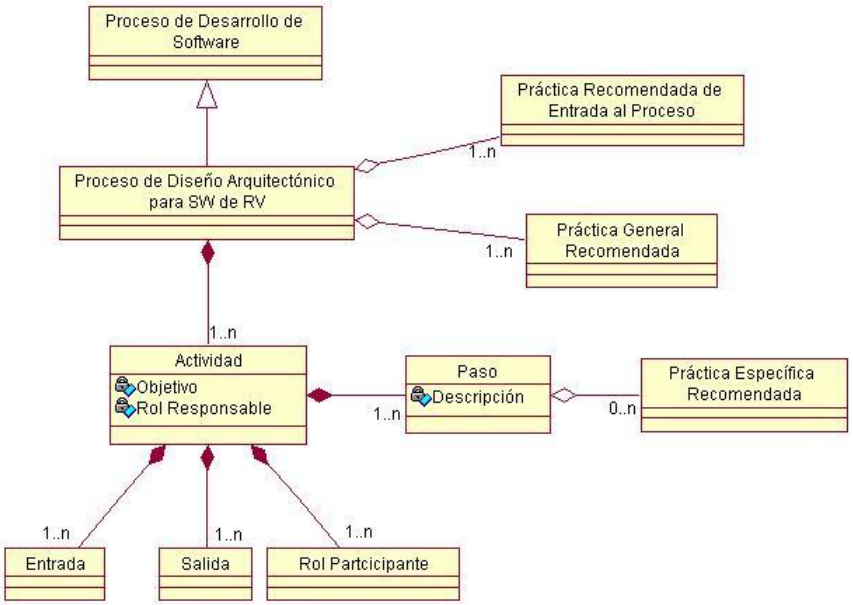


Figura 3: Estructura del Proceso de Diseño Arquitectónico para Software de RV.

El proceso está descrito en forma de diagrama de actividades en la siguiente figura:

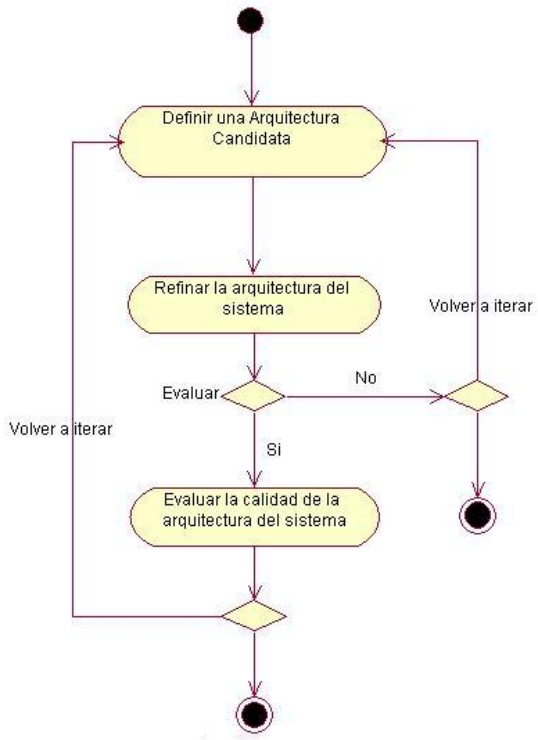


Figura 4: Actividades del Proceso de Diseño Arquitectónico para Software de RV.

2.9.1 Descripción del Proceso

Para cada actividad del proceso se establecen y describen los objetivos a cumplir con cada una, el rol responsable de la actividad, así como el resto de los roles participantes, las entradas, salidas y la secuencia de pasos a seguir para desarrollar la misma. Para cada iteración del proceso se irán adaptando las actividades para lograr refinar la arquitectura del sistema.

Artefactos de entrada y salida: La estructura de los artefactos que se utilizarán como entradas al proceso así como la de los artefactos resultantes luego de fundamentar las actividades del mismo es la que se encuentra establecida actualmente por la Dirección de Calidad de la Infraestructura Productiva de la UCI. (Documento de Arquitectura V 2.0, Plan de Mitigación de Riesgos V 2.0, etc.)

Actividad 1: Definir una Arquitectura Candidata.

Objetivos:

- ✓ Obtener una visión inicial del sistema a construir.
- ✓ Identificar las abstracciones claves del sistema.
- ✓ Seleccionar las herramientas a utilizar.
- ✓ Identificar y analizar los principales riesgos asociados a las herramientas seleccionadas, para trazar estrategias de mitigación.
- ✓ Lograr un seguimiento y control de los riesgos identificados.
- ✓ Definir la organización de alto nivel de los subsistemas.
- ✓ Obtener una visión del despliegue del software.

Rol Responsable: Arquitecto de Software.

Roles Participantes: Líder de Proyecto, Programadores.

	Obligatorias	**Opcionales**
Entradas	RF RNF Documento Visión	Modelo de CU Modelo de Escenarios Guión Técnico.

	Documento de Arquitectura (Priorización de CU, Escenarios, Requisitos o Guiones Técnicos) (primera iteración) Plan de mitigación de Riesgos. Modelo de despliegue. (posteriores iteraciones) Modelo de análisis (posteriores iteraciones).	Glosario de Términos Listado de Nuevas Funcionalidades y Cambios. (posteriores iteraciones) Conceptos de Uso (posteriores iteraciones) Documento de Evaluación (posteriores iteraciones) Documento de Arquitectura (actualizado) (posteriores iteraciones)
Salidas	Visión Inicial de la Arquitectura del Sistema. Abstracciones claves del diseño. Modelo de despliegue. Plan de mitigación de Riesgos (actualizado). Subsistemas de alto nivel	Fundamentación de los frameworks y herramientas. Documento de Arquitectura (actualizado)

Un Guión Técnico es una secuencia lógica y conceptual de reglas y condiciones para la elaboración de funcionalidades expresadas en forma de historia, en este guión textual se explican los pasos a seguir para construir las funcionalidades del software y se describen las características fundamentales que son especificadas por los clientes o por las personas que idean el software, incluyendo la interacción necesaria entre los usuarios del sistema y el sistema.

Los Conceptos de Uso son definidos por (Trujillo, 2007), estos se utilizan cuando en un entorno virtual existen agentes autónomos, básicamente lo que hace un concepto de uso es describir el propósito de una acción, cómo funciona y cuando debe realizarse, puesto que la misma es automática.

Las **condiciones mínimas** para comenzar esta actividad son las siguientes:

- Los RF y RNF han sido correctamente derivados y especificados de forma que sean medibles y verificables.

Pasos:

1. Elaborar una visión inicial de la Arquitectura del Sistema.

Este paso permite definir una idea primaria de lo que podría ser en un futuro la arquitectura, basada en las especificidades del cliente y las primeras decisiones en torno al diseño del entorno, se ilustran las primeras suposiciones de trabajo sobre la implementación de la visión, así como las decisiones sobre la arquitectura física y lógica, y los requisitos no funcionales del sistema. Tiene la forma de un gráfico

de iconos con muchas imágenes. Conceptualmente, ilustra la naturaleza de la solución propuesta, a la vez que transmite las ideas gobernantes e incluye los principales bloques de construcción.

También se puede basar la arquitectura en una arquitectura de referencia o en otros patrones de arquitectura. En este paso también se podrá seleccionar el estilo arquitectónico a utilizar.

Para una primera iteración no es necesario que todos los requisitos funcionales se encuentren definidos, esta visión inicial solo contemplará los requisitos que se encuentren especificados hasta el momento. A medida que se comience a avanzar en el proyecto se podrán adicionar y modificar funcionalidades, estas se irán incorporando a la visión en posteriores iteraciones.

El estilo arquitectónico seleccionado deberá ser fundamentado en el Documento de Arquitectura del proyecto (apartado 7: “Estilos Arquitectónicos”), así como los patrones utilizados (apartado 7.1: “Patrones Arquitectónicos”).

Práctica Específica Recomendada: Se recomienda incluir la visión inicial en el Documento de Arquitectura del proyecto (apartado 3: “Representación arquitectónica”).

2. Identificar abstracciones claves.

Se identifican los conceptos claves que el sistema debe poder manejar y se manifiestan como abstracciones del diseño clave.

Para el caso de los Software de RV se debe tener en cuenta la importancia del proceso del diseño 3D del entorno virtual antes que ningún otro en la fase de diseño. Se deben definir primeramente las características de cada uno de los avatares, agentes, escenarios, objetos, etc., puesto que la idea es dar soporte a cada objeto gráfico con un objeto de una de las clases que aparecen en el diagrama de clases. Por tanto, para este paso lo más recomendable es definir como abstracciones claves los principales elementos que componen el mundo virtual de la aplicación con el cual el usuario interactúa y sus propiedades gráficas, de comportamiento y de desempeño.

Presente las abstracciones claves en uno o varios diagramas de clases y cree una breve descripción para cada una. Este paso deberá ser documentado en el apartado 8: “Vista Lógica” del Documento de Arquitectura del proyecto.

3. Selección de los frameworks y herramientas a utilizar.

Teniendo en cuenta las características particulares de los software de RV, donde se hace necesario producir gráficos en 3 dimensiones, resulta mucho más práctico reutilizar librerías que brinden un conjunto de funcionalidades predefinidas que comenzar a implementar las mismas desde cero. Esto contribuye a minimizar el tiempo de desarrollo del proyecto, así como el esfuerzo empleado por parte del equipo de desarrollo.

Se sugiere desarrollar esta actividad como una especie de taller o trabajo en grupo donde participen los roles principales en el desarrollo (líder del proyecto, arquitecto, algunos programadores, entre otros) donde se tomen las decisiones sobre las herramientas o frameworks a utilizar.

Antes de realizar el taller el arquitecto de software deberá efectuar las siguientes tareas:

- ✓ Analizar las necesidades del software y las principales funcionalidades que deberá brindar.
- ✓ Realizar un listado de las herramientas más importantes disponibles en el mercado que cubran estas funcionalidades.
- ✓ Seleccionar un conjunto de indicadores que permita medir las herramientas candidatas en base a los mismos.

  Para realizar esta tarea se pueden utilizar los indicadores definidos en la Ayuda de Rational (Paso 2: Recopilar Información sobre las herramientas, Tarea: Seleccionar y adquirir herramientas, Flujo de Trabajo Ambiente)

  Otra posibilidad para realizar la selección de los indicadores pudiera ser basar la misma en los 6 atributos claves de calidad para el desarrollo de software identificados en el estándar ISO 9126-1, los cuales son: (ISO/IEC, 2002)

  - Funcionalidad
  - Fiabilidad
  - Usabilidad
  - Eficiencia
  - Mantenibilidad
  - Portabilidad

- ✓ Distribuir las herramientas candidatas entre los roles seleccionados (programadores, diseñadores, entre otros) para realizar el estudio de estas herramientas en base a los indicadores definidos.
- ✓ Determinar el Modelo de Decisión[5] que será empleado para la selección de las herramientas a utilizar en el proyecto y explicar el mismo a los involucrados en esta tarea.

  Para realizar esta tarea se pueden utilizar métodos tales como la recolección, descripción, visualización y resumen de los datos; provenientes de la estadística descriptiva. Los datos pueden ser resumidos numérica o gráficamente.

Posteriormente se deberá realizar un taller guiado por el Arquitecto de Software del proyecto, en el cual se procederá a tomar las decisiones colegiadas sobre qué herramientas usar y las estrategias de mitigación para los riesgos identificados, realizando las tareas que se muestran a continuación:

- ✓ Seleccionar las herramientas que más se adecúen a las necesidades del software que se desarrollará utilizando el modelo de decisión definido por el arquitecto de software.
- ✓ Definir herramientas alternativas para cada una de estas, las cuales se podrán utilizar en caso de problemas técnicos en alguna de las herramientas seleccionadas anteriormente.
- ✓ Fundamentar la selección realizada en el Documento de Arquitectura de Software (apartado 2: “Ambiente de Desarrollo”).
- ✓ Identificar los riesgos asociados a los frameworks y herramientas seleccionadas.
- ✓ Asignar prioridades a cada uno de ellos, de esta forma se comenzará administrando los riesgos más importantes.
- ✓ Trazar planes y estrategias para la mitigación de los riesgos.
- ✓ Actualizar el documento Plan de Mitigación de Riesgos, incorporando los riesgos asociados a las herramientas seleccionadas y las estrategias de mitigación.

Es necesario hacer un proceso riguroso en la selección y manejo de los riesgos asociados a las herramientas seleccionadas, pues puede ocurrir que si se detecta en una fase muy avanzada del

[5] Modelo de Decisión: Herramienta para la evaluación de alternativas utilizada en la toma de decisiones.

proyecto que la herramienta no satisface las necesidades del mismo, la migración a otra herramienta puede tener un costo tan alto (por ejemplo tiempo de aprendizaje y migración de modelos actuales) que haga fracasar el proyecto. Por este motivo, este paso se realiza fundamentalmente en la primera iteración. Si surgen nuevas necesidades en el proyecto que requieran la incorporación o uso de nuevas herramientas, se deberá realizar este paso nuevamente.

En posteriores iteraciones se pueden añadir nuevos riesgos que no fueron detectados en una primera iteración o riesgos asociados a las herramientas seleccionadas que fueron detectados por el equipo de evaluadores, así como las principales decisiones para minimizar su impacto.

La siguiente tabla muestra las herramientas más usadas para desarrollar software de RV en la Facultad 5:

Soporte Gráfico	Soporte gráfico STK Soporte gráfico Ogre3D 1.6 Soporte gráfico SDL Soporte gráfico OpenSceneGraph Soporte gráfico OSGART
API	API física ODE 0.9 API física Fhysx 2.8.1 API física Box2D API de sonido SDL 2.8 API de sonido SoundToolkit API de sonido OpenAL 1.1 API de sonido SDL_Mixer API red HawKNL 1.7 API de imágenes DevIL API de IO OIS 0.99 API de interfaz CEGUI 0.5
Otras	Estándar de código Ogre Modificado Estándar de código Java Estándar de código STK Estándar de código STK Modificado

Tabla 13: Principales herramientas utilizadas en los proyectos de RV de la Facultad 5

Práctica Específica Recomendada 1: Cuando el equipo de desarrollo es pequeño y se encuentra ubicado en un mismo local lo más recomendable es practicar la comunicación cara a cara y evitar la documentación innecesaria dentro del equipo de desarrollo. Por tanto, es más conveniente documentar

la fundamentación de los frameworks y de las herramientas seleccionadas solo si es exigida por el cliente, postergando la misma para etapas posteriores de desarrollo, puede ser cuando ya se esté implementando el software y el arquitecto se encuentre más liberado de trabajo.

Práctica Específica Recomendada 2: Utilizar listas de chequeo para identificar los riesgos asociados a las herramientas seleccionadas.

4. Seguir y controlar riesgos.

Se supervisa el estado de los riesgos y el progreso de sus planes de acción. Se controla que los nuevos cambios no alteren las características, recursos o programación del proyecto.

Este paso debe hacerse de manera constante (si es posible semanal), y tenerse en cuenta a partir de la segunda iteración del proceso.

5. Definir la organización de alto nivel de los subsistemas.

Un subsistema de diseño se utiliza para encapsular las colaboraciones de modo que los clientes del subsistema puedan ignorar por completo el diseño interno del subsistema, incluso cuando utilicen los servicios que ofrece el subsistema.

En este paso se crea una estructura inicial para el modelo de diseño agrupando los elementos del modelo de análisis en subsistemas o paquetes de alto nivel. También se pueden utilizar patrones arquitectónicos para la organización de los subsistemas, en este caso los subsistemas se definen alrededor de la plantilla de arquitectura del patrón a utilizar.

Para cada subsistema se debe definir un nombre, así como una descripción corta que refleje los principales elementos que componen el mismo. Defina también las relaciones existentes entre los subsistemas de alto nivel especificados.

Teniendo en cuenta que para una primera iteración no se ha realizado el diseño detallado del software esta visión inicial de los subsistemas se realiza con gran nivel de abstracción. En las iteraciones siguientes se debe ir refinando la estructuración de los subsistemas a partir del modelo de análisis o del modelo de diseño y a partir de los cambios que surjan en las funcionalidades del software.

Los patrones utilizados deberán ser fundamentados en el Documento de Arquitectura del proyecto (apartado 7.1: “Patrones Arquitectónicos”).

6. Desarrollar una visión del despliegue del software.

En este paso se procede a desarrollar una visión inicial de alto nivel de cómo quedará desplegado el software, componentes de hardware a utilizar (casco visor, consolas, tarjetas de video, vehículos, cámaras de video, sensores remotos, etc.) los protocolos de comunicación entre ellos, así como la descripción de sus propiedades físicas (memoria, velocidad, etc.).

Para posteriores iteraciones se debe refinar la visión del despliegue teniendo en cuenta cambios que surjan en las funcionalidades y en los requisitos de calidad especificados.

Se actualiza la Vista de Despliegue del Documento de Arquitectura del proyecto (apartado 10: "Vista de Despliegue") documentando la descripción de la estructuración del modelo, así como las propiedades físicas de los componentes hardware.

Actividad 2: Refinar la arquitectura del sistema.

Objetivos:

- ✓ Perfeccionar la arquitectura incorporando restricciones impuestas por los RNF.
- ✓ Analizar las interacciones de las clases de análisis para identificar interfaces de subsistemas.
- ✓ Perfeccionar la arquitectura, incorporando la reutilización donde sea posible.
- ✓ Volver a equilibrar la estructura del modelo de diseño a partir de los nuevos elementos del modelo.
- ✓ Identificar mecanismos de persistencia.
- ✓ Establecer la estructura de los elementos de implementación.

Rol Responsable: Arquitecto de Software

	Obligatorias	Opcionales
Entradas	RF RNF Visión inicial de la arquitectura Abstracciones claves del diseño	Modelo de CU. Modelo de Escenarios. Conceptos de Uso. Guión Técnico.

	Modelo de despliegue Subsistemas de alto nivel Plan de Mitigación de Riesgos Modelo de análisis (posteriores iteraciones) Modelo de diseño (posteriores iteraciones)	Documento Visión. Glosario de términos. Documento de la arquitectura. Documento de Evaluación (posteriores iteraciones)
Salidas	Visión de la Arquitectura del sistema.(actualizada) Subsistemas e Interfaces de alto nivel Mecanismo de Persistencia Plan de Mitigación de Riesgos (actualizado) Diagrama de vista de procesos (posteriores iteraciones) Modelo de diseño (actualizado) (posteriores iteraciones) Subsistemas de implementación (posteriores iteraciones)	Documento de la arquitectura (actualizado)

Las **condiciones mínimas** para comenzar la actividad son las siguientes:

- Los RF y RNF han sido correctamente derivados y especificados de forma que sean medibles y verificables.
- Se ha definido una visión inicial para la arquitectura del sistema.
- Se han identificado los subsistemas de diseño de alto nivel.

Pasos:

1. Incorporar restricciones impuestas por los RNF a la visión de la arquitectura.

Para una primera iteración se analizan los RNF que tienen un impacto significativo para la arquitectura y se comienzan a incorporar restricciones arquitectónicas a la visión inicial, se identifican y analizan los riesgos asociados a las decisiones arquitectónicas tomadas y los riesgos asociados a los requisitos de calidad especificados. Se trazan estrategias para la mitigación de los riesgos. Se actualiza el Plan de Mitigación de Riesgos.

Las restricciones arquitectónicas deberán ser documentadas en el apartado 4: “Objetivos y Restricciones Arquitectónicas” del Documento de Arquitectura del proyecto. Las restricciones

relacionadas a los RNF de tamaño y rendimiento deberán ser especificadas en el apartado 5: “Tamaño y Rendimiento” del Documento de Arquitectura del proyecto.

Se debe especificar además cómo y en qué medida la arquitectura soporta cada uno de estos requisitos en el apartado 13: “Calidad” del Documento de Arquitectura del proyecto.

Para posteriores iteraciones se incorporan a la arquitectura los requisitos de calidad identificados por el equipo de evaluadores que no fueron tenidos en cuenta anteriormente, se analizan además los que no son soportados adecuadamente por la arquitectura y se continúa refinando la misma en base a estos requisitos.

Considerando que un software de RV debe ser capaz de producir efectos visuales, auditivos y táctiles, coordinar estímulos que recibe el usuario con sus movimientos y acciones acordes al mundo simulado, analizar informaciones de entradas producidas de las interacciones con el mundo virtual y enviar una posible respuesta al usuario de forma rápida, entre otras características específicas, es importante que la arquitectura del sistema soporte los RNF especificados, dando prioridad a los requisitos de:

- Eficiencia: Estos software por lo general consumen muchos recursos, debido a la gran cantidad de gráficos en 3D que poseen, por lo que es importante asegurarse de que el hardware donde será instalado finalmente tenga un desempeño apropiado relacionado a la cantidad de recursos usados. Se debe tener en cuenta además que estos sistemas son por lo general aplicaciones de tiempo real, por lo que deben tener alto grado de velocidad de procesamiento o cálculo, tiempo de respuesta, de recuperación y disponibilidad.
- Seguridad: Se deben realizar todas las validaciones necesarias para garantizar la seguridad del sistema y su funcionamiento. También este debe ser protegido del mal manejo por parte de los usuarios.

2. Identificar interfaces de subsistemas.

Una interfaz es un elemento público de un subsistema que proporciona una capa de encapsulación, lo cual permite mantener oculto el diseño interno del subsistema de otros elementos del modelo. Las interfaces son importantes para los subsistemas, estas permiten la separación de la declaración del comportamiento (la interfaz) de la realización del comportamiento (las clases específicas dentro del subsistema que realizan la interfaz).

Este paso se realiza a partir de una segunda iteración, cuando se haya avanzado en el diseño detallado y se procede a identificar las interfaces para cada subsistema, realizando las siguientes tareas:

- ✓ Identificar un conjunto de interfaces candidatas para cada subsistema.
- ✓ Buscar similitudes entre las interfaces candidatas: si existen las mismas operaciones en varias interfaces, refactorizar las interfaces, extrayendo las operaciones comunes a una nueva interfaz, puesto que el objetivo es mantener la coherencia de las interfaces y eliminar las operaciones redundantes.
- ✓ Definir las relaciones de dependencia entre la interfaz y las interfaces de las que depende: las dependencias de interfaz definen las dependencias primarias entre los elementos del modelo de diseño.
- ✓ Crear asociaciones de realización entre el subsistema y las interfaces que realiza: una realización del subsistema a una interfaz indica que hay uno o varios elementos dentro del subsistema que realizan operaciones de la interfaz.
- ✓ Empaquetar interfaces: para gestionarlas, las interfaces se deben agrupar en uno o varios paquetes propiedad del arquitecto de software. Si cada interfaz la realiza un único subsistema, las interfaces se pueden colocar en el mismo paquete con el subsistema. Si las interfaces las realizan más de un subsistema, se colocarán en un paquete aparte propiedad del arquitecto de software. Esto permite gestionar y controlar las interfaces de forma independiente de los subsistemas.

3. Identificar oportunidades de reutilización.

Se identifican componentes comunes que puedan ser reutilizados para la construcción del nuevo sistema a partir de sistemas similares construidos anteriormente o componentes disponibles en el mercado. Los componentes existentes se deben examinar para determinar su idoneidad y compatibilidad con la arquitectura del software.

4. Revertir la ingeniería de los componentes reutilizables.

Los componentes existentes, ya sean componentes desarrollados en iteraciones anteriores que no se han incluido todavía en el modelo de diseño o componentes adquiridos, deben revertir la ingeniería y se deben incorporar en el modelo de diseño. En el modelo de diseño, esos componentes se representan normalmente como un subsistema con una o varias interfaces. Se debe tener en cuenta los nuevos elementos de modelo en la organización del modelo de diseño y volver a equilibrar la estructura del mismo cuando sea necesario.

Este paso se debe realizar a partir de una segunda iteración del proyecto, cuando se haya avanzado en el diseño detallado.

5. Actualizar la vista lógica.

Cuando el diseño de clases, paquetes y subsistemas (elementos de modelo) es importante desde el punto de vista de la arquitectura, se deben incluir en el apartado 8: "Vista Lógica" del Documento de Arquitectura del proyecto. Esto garantiza que los elementos del modelo de diseño significativos para la arquitectura se comuniquen a los miembros del equipo de desarrollo.

6. Describir la arquitectura de tiempo de ejecución

En este paso se define una arquitectura del proceso para el sistema en términos de clases activas e instancias[6], y la relación de éstas con los procesos y las hebras del sistema operativo, realizando las siguientes tareas:

- ✓ Analizar los requisitos de concurrencia: se consideran requisitos de concurrencia aquellos dirigidos principalmente por demandas de concurrencia naturales en el dominio del problema, así como los de concurrencias impuestas por los requisitos no funcionales del sistema.

Por ejemplo:

- Para proporcionar unos tiempos de respuesta correctos, es recomendable colocar las actividades con una gran carga computacional en una hebra o proceso propio, de forma que el sistema pueda responder a las entradas de usuario mientras se realizan los cálculos, aunque sea con menos recursos.

[6] Instancias de clases activas: Los objetos activos (es decir, las instancias de las clases activas) se utilizan para representar hebras de ejecución concurrentes en el sistema. La correlación de objetos activos con procesos o hebras reales del sistema operativo varía según los requisitos de capacidad de respuesta.

- Un sistema cuyo comportamiento se debe distribuir entre varios procesadores o nodos virtualmente requiere una arquitectura de varios procesos.

✓ Definir los procesos y las hebras que existirán en el sistema: para cada flujo de control aparte que necesite el sistema, cree un proceso o una hebra (proceso ligero). La hebra se debe utilizar en aquellos casos en los que se necesite un flujo de control anidado (por ejemplo, si dentro de un proceso hay necesidad de flujo de control independiente a nivel de subtareas).

✓ Identificar cuándo se crean y destruyen los procesos y las hebras: En arquitecturas de varios procesos, los nuevos procesos (o hebras) se despliegan o se bifurcan del proceso inicial creado por el sistema operativo cuando se inicia la aplicación. Estos procesos se deben destruir también explícitamente. La secuencia de sucesos que lleva a la creación y la destrucción de procesos se debe determinar y documentar, así como el mecanismo de creación y supresión.

Además se deben realizar las siguientes tareas a partir de una segunda iteración del proyecto, cuando se haya avanzado en el diseño detallado.

✓ Identificar mecanismos de comunicación entre procesos: identificar los medios que utilizarán los procesos y las hebras para comunicarse. Los Mecanismos de Comunicación entre Procesos (IPC) permiten enviar mensajes entre objetos que se ejecutan en procesos separados.

✓ Asignar recursos de coordinación entre procesos: asignar recursos escasos. Anticipar y gestionar los posibles cuellos de botella de rendimiento.

✓ Correlacionar los procesos con el entorno de implementación: correlacionar los "flujos de control" con los conceptos soportados por el entorno de implementación. Los procesos conceptuales se deben correlacionar con el entorno operativo.

✓ Correlacionar elementos de diseño con hebras de control: determinar en qué hebras de control se deben ejecutar las clases y los subsistemas. Las instancias de una determinada clase o subsistema se deben ejecutar como mínimo en una hebra de control que proporcione el entorno de ejecución de la clase o el subsistema; de hecho, se pueden ejecutar en varios procesos diferentes.

✓ Actualizar el Documento de Arquitectura del proyecto (apartado 9: Vista de Procesos), incorporando el diagrama de vista de procesos que muestra la composición de los procesos e

hilos, y la distribución de las clases en estos procesos e hilos. Describir las tareas (procesos, hilos, tareas programadas, eventos y notificaciones) involucrados en la ejecución del sistema y la ubicación de las clases y objetos necesarios para estas tareas de ser necesario.

7. Definir mecanismo de persistencia.

Teniendo en cuenta que por lo general para desarrollar software de RV no es necesario contar con grandes bases de datos, debido a que los mismos no cuentan con una gran cantidad de elementos persistentes, en este paso se procede a analizar los datos persistentes que poseerá el sistema para determinar cómo quedarán almacenados los mismos, se analizarán los estándares existentes para guardar la información de los entornos tridimensionales con que se trabajará o se procederá a crear estándares propios. Se debe incluir en el documento de arquitectura de software la descripción de los datos arquitectónicamente significativos para de esta forma fundamentar el mecanismo de persistencia seleccionado.

Esta información se especifica en el apartado 12: “Vista de Datos” del Documento de Arquitectura de Software.

8. Definir la estructuración del modelo de implementación.

En este paso se definen los subsistemas de implementación a partir del modelo de diseño. Se crea un diagrama que represente la estructura del modelo de implementación, adaptando la estructura del modelo para que refleje las restricciones del lenguaje de implementación. Se definen las dependencias entre los subsistemas, determinando para cada uno, que otros subsistemas importa. El arquitecto de software deberá decidir la estructura de elementos de configuración que se va a aplicar al modelo de implementación.

Se actualiza el apartado 11: “Vista de Implementación” del Documento de Arquitectura del proyecto. Este apartado contiene diagramas de componentes que muestran las capas y la asignación de subsistemas de implementación a las capas, así como las dependencias de importación entre los subsistemas.

Este paso se debe realizar a partir de una segunda iteración del proyecto, cuando se haya avanzado en el diseño detallado.

Práctica Específica Recomendada 1: Se recomienda realizar el mapeo o relación de los elementos de despliegue con los componentes de implementación.

Actividad 3: Evaluar la Calidad de la Arquitectura del Sistema. (Opcional)

Los pasos de esta actividad están basados en el Procedimiento para Evaluar Arquitecturas de Software planteado por (Guelmes & Carballo, 2009).

Objetivos:

- ✓ Asegurar que la arquitectura propuesta para el sistema es capaz de soportar en alguna medida las necesidades del usuario y los atributos de calidad especificados.
- ✓ Evaluar el impacto de los riesgos de las decisiones arquitectónicas tomadas hasta el momento para los atributos de calidad y requisitos especificados.

Roles Responsables: Evaluadores (Grupo de especialistas de la calidad), Arquitecto de Software.

Roles Participantes: Relacionados (personas involucradas de alguna manera con el software a evaluar: programadores, usuarios, gerentes, entre otros).

	Obligatorias	Opcionales
Entradas	Documento de la arquitectura (actualizado) RF RNF Plan de Mitigación de riegos Glosario de Términos Documento de evaluación (anteriores evaluaciones)	Modelo de CU Modelo de escenarios Realización de CU o Escenarios Conceptos de Uso Guión Técnico Modelo de análisis
Salidas	Documento de evaluación	

Para lograr buenos resultados luego de realizar esta actividad del proceso se necesita un nivel mínimo de madurez de la arquitectura antes de que se le pueda aplicar exitosamente el proceso de evaluación.

Por tanto, las **condiciones mínimas** para comenzar la actividad son las siguientes:

- Los requisitos o atributos de calidad han sido correctamente derivados y especificados de forma que sean medibles y verificables.
- La arquitectura ha sido documentada apropiadamente en términos de diagramas, modelos y documentos cumpliendo con las normas de calidad establecidas en el Proyecto, Polo Productivo o Institución.
- La arquitectura tiene un nivel de especificación que hace factible someterla al proceso de evaluación.
- Los miembros del equipo de evaluación poseen conocimientos sobre arquitectura y preferiblemente experiencia como arquitectos en el tipo de sistema que se evalúa.

Pasos:

1. Solicitar la evaluación de la arquitectura del proyecto.

El arquitecto de software debe solicitar el proceso de evaluación de la arquitectura y enviar la documentación necesaria (Documento de Especificación de Requisitos, Documento de Arquitectura de Software) para que se inicie el proceso de evaluación.

2. Determinar si el proyecto reúne las condiciones para comenzar la evaluación.

El equipo de evaluadores verifica que los requisitos o atributos de calidad han sido correctamente derivados y especificados de forma que sean medibles y verificables, la arquitectura ha sido documentada apropiadamente en términos de diagramas, modelos y documentos cumpliendo con las normas de calidad establecidas en el Proyecto, Polo Productivo o Institución, la arquitectura tiene un nivel de especificación que hace factible someterla al proceso de evaluación.

En caso de que la arquitectura no posea un nivel de madurez que haga factible someterla al proceso de evaluación, el jefe del equipo de evaluadores debe comunicar al arquitecto del proyecto las razones de la No Aprobación.

3. Presentación de la arquitectura del sistema.

En este paso el arquitecto de software debe exponer al equipo de evaluadores un conjunto de aspectos, tales como:

- Las funcionalidades más importantes del sistema.
- Los objetivos del negocio.
- Los requisitos de calidad a ser satisfechos por la arquitectura.
- La estructura lógica y las relaciones entre los componentes de la arquitectura.
- Las restricciones técnicas (sistema operativo, hardware, otros sistemas con los que debe interactuar).
- Las decisiones arquitectónicas de diseño (patrones, etc.)
- Los riesgos asociados a cada escenario.

Para evaluar los riesgos asociados a cada escenario el arquitecto de software se puede auxiliar en la siguiente tabla:

Escenario	**Atributo de calidad que afecta.**	**Decisiones arquitectónicas que involucra.**	**Riesgos para el sistema.**	**Impacto (alto, medio, bajo)**

Tabla 14: Guía para evaluar riesgos asociados a cada escenario

4. Selección del método para evaluar la arquitectura del software.

En este paso el equipo de evaluadores selecciona el método a emplear para evaluar la arquitectura del sistema de acuerdo a sus características específicas.

La siguiente tabla resume algunos de los métodos que se podrán utilizar para evaluar la arquitectura de un sistema. Para obtener mayor información acerca de estos métodos de evaluación consultar (SEI).

Método de Evaluación	**Breve descripción**
Architecture Tradeoffs Analysis	Está basado en tres áreas distintas: los estilos arquitectónicos, el análisis de atributos de calidad y el método de evaluación SAAM (Software Architecture Analysis Method).

Method (ATAM)	Apoya a los involucrados con el proyecto a entender las consecuencias de las decisiones arquitectónicas respecto a los atributos de calidad del sistema.
Método de Bosch	El método tiene dentro de sus objetos de estudio los siguientes elementos: estilos arquitectónicos, vistas arquitectónicas, patrones arquitectónicos y patrones de diseño. Plantea además que el proceso de evaluación debe ser visto como una actividad iterativa, que forma parte del proceso de diseño, también iterativo.
Método de Evaluación para Arquitecturas Basadas en Componentes (MECABIC).	El MECABIC está inspirado en ATAM, aunque con el objetivo de facilitar su aplicación sobre las Arquitecturas de Software Basadas en Componentes.

Tabla 15: Métodos para evaluar la calidad de la arquitectura de un sistema

5. Presentación del método para evaluar la arquitectura.

En este paso el Jefe de evaluadores presenta el método ante los participantes del proyecto, explicando la función que deberá cumplir cada persona implicada en el proceso de evaluación de la arquitectura del proyecto.

6. Evaluar la arquitectura del software.

El equipo evaluador valora las decisiones arquitectónicas y elabora el documento final donde debe incluir los riesgos no considerados para las decisiones arquitectónicas, así como su posible impacto, los atributos de calidad significativos que se deben considerar, propuesta de decisiones arquitectónicas más efectivas que las propuestas inicialmente, valorando el esfuerzo de llevar a cabo las ventajas de su ejecución, decisiones arquitectónicas que es necesario cambiar por sus consecuencias para el proyecto o la institución.
Finalmente se debe llegar a un consenso entre los criterios del equipo de evaluadores y el arquitecto sobre los aspectos que serán modificados a partir de los señalamientos y recomendaciones hechas. El equipo evaluador puede determinar si la arquitectura deberá ser sometida a otra evaluación de tipo conceptual antes de ser implementada.

Prácticas Recomendadas para las entradas del proceso:

1. En un software de RV por lo general existe una gran cantidad de caminos de interacción de un usuario. Por tanto, se hace más conveniente que el proceso de desarrollo de software sea dirigido por escenarios y no por CU, puesto que un escenario significa un único camino que

conduce a alcanzar determinados objetivos. Los escenarios resultan ser una entrada con mayor nivel de detalle para el proceso de diseño arquitectónico.

2. Se recomienda postergar la documentación detallada de los diferentes Modelos (CU, Escenarios, Análisis, Diseño, entre otros) para etapas posteriores de desarrollo y centrar la atención en el uso de modelos simplificados con mayor interacción entre arquitectos y analistas.

Prácticas Recomendadas Generales del proceso:

1. Se recomienda que se realicen al menos 2 iteraciones del proceso para garantizar que la arquitectura de software quede bien especificada y soporte las restricciones impuestas por los requisitos de calidad.

2. Se recomienda aprender de los riesgos identificados formalizando las lecciones aprendidas, los elementos y herramientas relevantes del proceso y plasmar toda esta información en un formato reutilizable para el equipo de desarrollo.

3. Se recomienda lograr que la información del proyecto fluya fácilmente entre los miembros del equipo, integrando todas las contribuciones y promoviendo la combinación de diferentes puntos de vista, para de esta forma involucrar a todo el equipo en la toma de las decisiones fundamentales.

4. Se recomienda que el final de cada iteración coincida con el comienzo de la otra, para de esta forma analizar los posibles cambios que puedan haber surgido en las necesidades del negocio y comenzar a incorporar las mismas a la arquitectura del sistema. Además se deberán analizar los resultados de una iteración y con estos resultados guiar y fortalecer la siguiente iteración.

5. Se recomienda que el rol de Arquitecto de Software sea desempeñado por una persona que posea los conocimientos necesarios, que posea además experiencia en proyectos de RV para de esta forma desarrollar las actividades del proceso con mayor calidad.

6. Se recomienda realizar la tercera actividad del proceso en al menos una iteración para de esta forma garantizar que el sistema cumple con los servicios y la funcionalidad que espera el usuario, además de los atributos de calidad asociados antes de comenzar a implementar, debido a que una evaluación temprana puede evitar un posible fracaso del proyecto.

7. Cuando el equipo de desarrollo es pequeño y se encuentra ubicado en un mismo local lo más recomendable es practicar la comunicación cara a cara y evitar la documentación innecesaria dentro del equipo de desarrollo. Teniendo en cuenta que la arquitectura irá creciendo a medida que surjan nuevas necesidades de negocio y además que para el desarrollo de la tercera actividad del proceso es necesario que la arquitectura se encuentre documentada apropiadamente en términos de diagramas, modelos y documentos, se recomienda actualizar las diferentes vistas del documento de arquitectura del proyecto justo antes de comenzar a evaluar y si no se va a evaluar entonces se podrá documentar una vez concluido el proceso y el arquitecto de software se encuentre más liberado de trabajo, realizando una descripción corta y centrada en lo elemental. La comunicación entre arquitectos, analistas, diseñadores, programadores y otros roles podrá realizarse a través de la comunicación oral, apoyándose en los diferentes modelos generados en la herramienta de modelado.
8. Se recomienda emplear mayor esfuerzo y priorizar el diseño 3D del entorno virtual y de cada uno de los elementos del mismo, puesto que su estructura será completamente distinta dependiendo de las acciones que tengan que realizar. El hecho de que estén claras las diferentes necesidades en la estructura de estos elementos implicará algunas diferencias en el diagrama de clases resultante, puesto que la idea es dar soporte a cada objeto gráfico con un objeto de una de las clases que aparezcan en el diagrama de clases, para poder manejar así sus propiedades y sus acciones.

2.10 Conclusiones Parciales

A partir de los objetivos propuestos para este capítulo definidos en la introducción se puede decir que:

- ✓ Se realizaron entrevistas a los arquitectos de software, diseñadores y programadores de una muestra de los proyectos que componen el polo de Realidad Virtual de la Facultad 5.
- ✓ Se analizaron los documentos de arquitectura y especificación de requisitos de estos proyectos.
- ✓ Se detectaron las deficiencias actuales que existen en el proceso de diseño arquitectónico en los proyectos de la Facultad 5.

Concluyendo que:

- ✓ En estos proyectos se intenta aplicar la metodología RUP con todos sus pasos y actividades, sin tener en cuenta que la misma no ha sido diseñada específicamente para este tipo de software, por lo que el proceso de diseñar la arquitectura se torna muy largo y genera documentación excesiva, lo cual afecta el tiempo de desarrollo del proyecto.
- ✓ Teniendo en cuenta lo anterior y que además los productos deben ser terminados en un tiempo relativamente corto, en la mayor parte de estos proyectos no se realiza el diseño de alto nivel pasando directamente a la codificación.
- ✓ Por lo antes expuesto se definió y describió de forma detallada una propuesta de Proceso de Diseño Arquitectónico para Software de RV, en el cual se tienen en cuenta todas la sugerencias planteadas por los entrevistados de estos proyectos, así como algunas consideraciones y principios planteados en las metodologías ágiles estudiadas, brindando solución al problema científico de la investigación.
- ✓ Esta propuesta debe contribuir a que el proceso de diseñar la arquitectura pueda utilizarse para construir software de RV, y de esta forma elevar la calidad del diseño de alto nivel en los mismos.

CAPÍTULO 3

Validación de la Solución

3.1 Introducción

En el presente capítulo se analizan los resultados obtenidos luego de diseñar la arquitectura del software Kinetic a partir del proceso propuesto en el capítulo anterior, asegurando de esta forma la utilidad y efectividad del mismo y contribuyendo a que se defina una arquitectura inicial en un período corto de tiempo para este software. Se recolectan además diferentes críticas y sugerencias brindadas por los involucrados en la validación del proceso.

3.2 Kinetic

Como resultado de la colaboración entre la UCI y el Centro Nacional de Rehabilitación Julio Díaz, surge la proyección de desarrollar productos dirigidos en especial a los profesionales relacionados con investigaciones biomecánicas en las personas, permitiendo realizar la evaluación y valoración de la postura y la función motriz.

Kinetic es un sistema analizador biomecánico que aprovecha el potencial que ofrece la Realidad Aumentada (RA)[7] de sobreponer información virtual sobre el mundo real, el cual es desarrollado actualmente por el proyecto Laboratorio de Realidad Aumentada (ARLab) perteneciente al Área Temática Visualización Científica (ATV) del polo de RV de la Facultad 5.

El objetivo fundamental propuesto con el producto Kinetic es automatizar la evaluación biomecánica de las personas. Esta evaluación se realiza actualmente a través de la observación clínica, caracterizada por un importante componente subjetivo que introduce determinados márgenes de medición, cualitativos y cuantitativos en las mediciones.

El producto final brindará la posibilidad de optimizar la evaluación postural y funcional de los pacientes afectados, incrementar los niveles de precisión en los sistemas evaluadores tanto posturales como funcionales, estandarizar los criterios de evaluación entre los especialistas y uniformar el

[7] Realidad Aumentada: Brinda la posibilidad de insertar objetos, cuerpos o información senso – perceptual de carácter virtual, generados por computadoras en un entorno de la realidad, con la condición de ser interactiva con el usuario en tiempo real.

procesamiento de datos a través de métodos de digitalización compatibles con otros sistemas informáticos.

3.3 Diseño de la arquitectura del software Kinetic a partir del Proceso de Diseño Arquitectónico para Software de RV

Luego de una pequeña familiarización con la visión y los principales requisitos del software Kinetic, el cual se desarrolla actualmente en el proyecto ARLab de la Facultad 5, se procedió a realizar un trabajo en conjunto con la arquitecta de software y el líder del proyecto para diseñar la arquitectura del mismo siguiendo los pasos del proceso definido.

Se realizó una primera iteración de las dos primeras actividades que componen la propuesta de solución con el propósito fundamental de obtener una versión inicial de la arquitectura del sistema y de esta manera verificar que realmente la solución propuesta satisface las necesidades actuales que existen cuando se diseña la arquitectura de un software de RV.

A continuación se muestran algunos ejemplos de cómo se fue diseñando la arquitectura de este software.

Actividad 1

Los artefactos utilizados como entrada a esta actividad fueron los siguientes:

- ✓ Especificación de RF y RNF
- ✓ Documento Visión
- ✓ Documento de Arquitectura (Priorización de CU)
- ✓ Plan de mitigación de Riesgos

Paso 1

En este paso se definió la visión inicial de la arquitectura del sistema basando la misma en el estilo arquitectónico de arquitecturas estratificadas[8].

[8] Arquitectura Estratificada: Se crean diferentes capas y cada una realiza operaciones que progresivamente se aproxima más al cuadro de instrucciones de la máquina. En la capa externa, los componentes sirven a las operaciones de interfaz de usuario. En la capa interna, los componentes realizan operaciones de interfaz de sistema. Las capas intermedias proporcionan servicios de utilidad y funciones del software de aplicaciones.

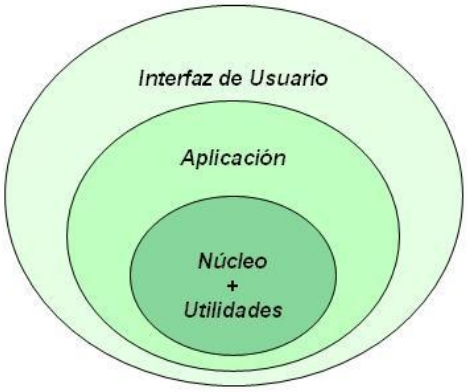


Figura 5: Visión inicial de la arquitectura de Kinetic.

Núcleo	Trabajo con el motor gráfico Trabajo con la visión por computadora
Aplicación	Lógica Acceso a datos
Interfaz de Usuario	Interacción del usuario con el sistema a través de ventanas y de la RV

Tabla 16: Breve descripción de los elementos que componen la visión inicial de la arquitectura del software.

Paso 2

Para este paso se procedió a identificar los conceptos claves que el sistema debe ser capaz de manejar en un futuro y se representaron los mismos a través de un diagrama de clases. Posteriormente se procedió a realizar una breve descripción de cada uno de estos conceptos. A continuación se muestra el diagrama de abstracciones claves del diseño elaborado.

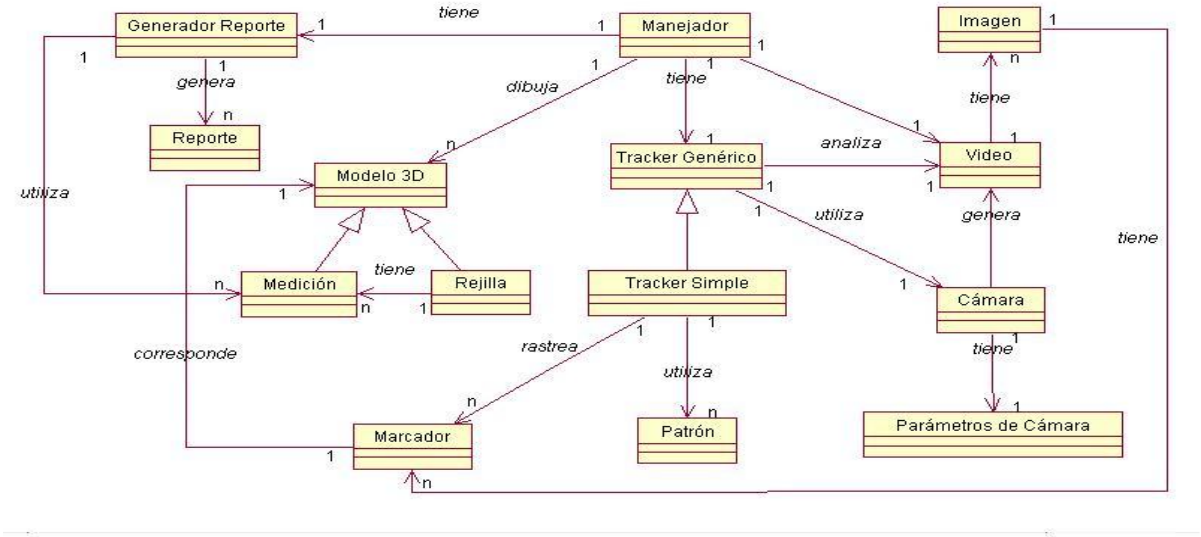


Figura 6: Abstracciones claves del diseño.

Paso 3

En este caso no existió la posibilidad de realizar el taller propuesto en el proceso de diseño definido, puesto que es un proyecto bastante pequeño y además cuando se comenzó a trabajar a partir de la solución propuesta, el líder del proyecto y el arquitecto de software ya habían hecho un estudio previo de las herramientas disponibles en el mercado y habían seleccionado las que más se adecuaban a las necesidades de Kinetic.

Para este paso se procedió a identificar herramientas alternativas las cuales podrán ser utilizadas en caso de problemas con alguna de las definidas, así como los principales riesgos asociados a las herramientas seleccionadas.

La siguiente tabla muestra las herramientas seleccionadas por el líder de proyecto y el arquitecto de software:

Soporte Gráfico	Soporte gráfico OpenSceneGraph Soporte gráfico OSGART
API	API de interfaz CEGUI 0.5

Otras	Estándar de código Java

Tabla 17: Herramientas seleccionadas.

Paso 5

Para este paso se definieron los subsistemas de alto nivel para Kinetic así como su organización, los cuales fueron agrupados utilizando el modelo tres capas y se muestran a continuación:

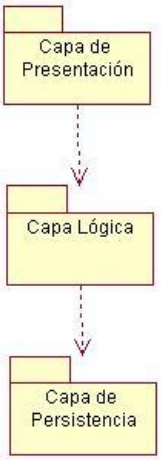


Figura 7: Diagrama inicial de los subsistemas de alto nivel

Descripción de los subsistemas de alto nivel:

Capa de Presentación: Este subsistema contiene dos paquetes, uno que agrupa los gráficos por computadora y otro para las clases de la interfaz de usuario.

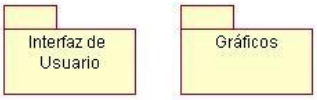


Figura 8: Paquetes que componen la Capa de Presentación

Capa Lógica: Este subsistema contiene dos paquetes, uno que agrupa los elementos para la detección de video y otro para las clases controladoras.

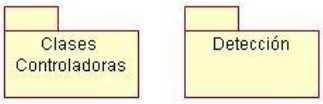


Figura 9: Paquetes que componen la Capa Lógica

Capa de Persistencia: Este subsistema contiene dos paquetes, uno que agrupa las clases persistentes que guardan los datos de los diagnósticos, los del personal médico autorizado para la utilización del software y los datos personales de cada paciente. El otro paquete estará compuesto por los elementos persistentes referente a los patrones de imagen usados en la RA.

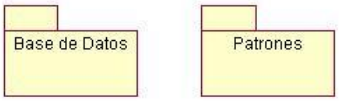


Figura 10: Paquetes que componen la Capa de Persistencia

Paso 6

Para este paso se definió una visión inicial de cómo quedará desplegado el producto Kinetic en un futuro, la cual se muestra a continuación:

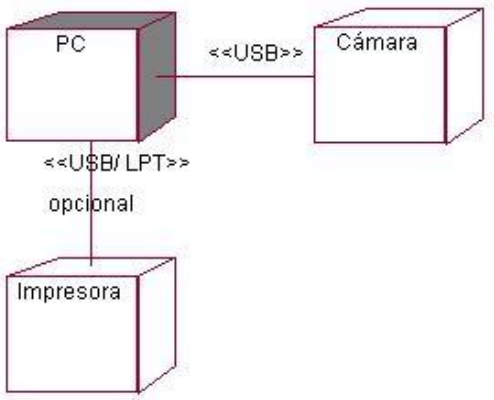


Figura 11: Despliegue inicial del software Kinetic

Actividad 2

Los artefactos utilizados como entrada a esta actividad fueron los siguientes:

- ✓ Especificación de RF y RNF

- ✓ Documento Visión
- ✓ Modelo de CU
- ✓ Visión inicial de la arquitectura
- ✓ Abstracciones claves del diseño
- ✓ Subsistemas de alto nivel
- ✓ Modelo de Despliegue
- ✓ Documento de Arquitectura
- ✓ Plan de Mitigación de Riesgos

Paso 1:

En la primera actividad del proceso se tuvieron en cuenta los RNF para definir la visión inicial de la arquitectura, teniendo en cuenta esto y que además es un software pequeño, que no contiene muchos módulos, para este paso se mantuvo la visión especificada inicialmente.

Paso 3:

En este paso se identificaron componentes reutilizables libres que se encuentran disponibles en el mercado, los cuales componen el núcleo, estos son:

- ✓ Soporte gráfico OSGART
- ✓ API de interfaz CEGUI 0.5

Paso 6:

No se cuenta con requisitos de concurrencia, por lo que no es necesario definir varios procesos.

Paso 7:

Para el almacenamiento de los elementos persistentes referentes a los patrones de imagen usados para la RA, calibración de la cámara y configuración de videos se utilizarán ficheros binarios y ficheros XML.

Para el almacenamiento de los elementos persistentes referentes a la información relativa a los datos de los pacientes, de los diagnósticos, y del personal médico autorizado para la utilización del software se utilizará el servidor de bases de datos SQLite.

3.4 Resultados Obtenidos

Solo existió la posibilidad de realizar una iteración de las dos primeras actividades del proceso propuesto debido a la falta de tiempo, lo que representa un 66,7 % del mismo, puesto que era necesario avanzar en cuanto a diseño detallado para lograr realizar los pasos que fueron definidos para realizarlos a partir de una segunda iteración, tales como: identificar interfaces de subsistemas, revertir la ingeniería de los componentes reutilizables, describir la arquitectura de tiempo de ejecución y definir la estructuración del modelo de implementación. La tercera actividad no fue llevada a cabo debido a que era necesario tener una arquitectura mucho más refinada y robusta para lograr una evaluación exitosa, además es una actividad que requiere de un determinado tiempo, del cual no se disponía.

Una vez realizada la primera iteración del proceso para las dos primeras actividades del mismo, quedó conformada una visión inicial de alto nivel para el software Kinetic, que contempla los principales elementos del sistema, las relaciones entre ellos y los principios fundamentales que orientan su diseño y evolución. Esta arquitectura debe ser refinada realizando al menos otra iteración del proceso.

Posteriormente se podrán realizar tantas iteraciones como se considere necesario hasta obtener una base robusta para la construcción del software. Se debe además realizar al menos una evaluación temprana de la arquitectura propuesta para de esta forma asegurar que el sistema cumple con los servicios y la funcionalidad que espera el usuario antes de comenzar a implementar.

Como resultado principal del intercambio realizado utilizando el proceso para diseñar la arquitectura del producto Kinetic, se detectaron algunas deficiencias en el proceso propuesto inicialmente. El proceso definido de manera teórica, basado en diferentes procesos de diseño arquitectónico planteados en algunas de las metodologías de desarrollo de software estudiadas y en un estudio del proceso en diferentes proyectos de la Facultad 5, fue llevado a la práctica y de esta forma se obtuvieron diferentes críticas y sugerencias por parte de los encargados de definir la arquitectura de Kinetic (líder de proyecto, arquitecto de software).

La siguiente tabla muestra un resumen de los principales cambios realizados luego de llevar a la práctica el proceso planteado inicialmente:

	Cantidad de pasos adicionados	Cantidad de pasos modificados	Cantidad de pasos eliminados
Actividad 1	**0**	**2**	**0**
Actividad 2	**0**	**1**	**0**

Tabla 18: Cambios realizados en el proceso inicial propuesto.

3.5 Encuestas realizadas a los involucrados en la validación del proceso propuesto (líder de proyecto, arquitecto de software)

Se aplicó una encuesta a los involucrados en el proceso de diseño arquitectónico del software Kinetic (líder de proyecto, arquitecto de software), persiguiendo los siguientes objetivos: (Ver preguntas de la encuesta: Anexo 10)

- ✓ Obtener criterios acerca del proceso propuesto.
- ✓ Determinar si la solución propuesta satisface las necesidades actuales existentes en los proyectos de RV de la Facultad 5.
- ✓ Determinar si la solución propuesta contribuye a disminuir el tiempo y esfuerzo empleado por parte de los arquitectos de software para realizar este trabajo.
- ✓ Determinar si la solución propuesta contribuye a elevar la calidad del diseño de más alto nivel en la construcción de software de RV.

3.6 Encuestas realizadas a los diseñadores del software Kinetic

Luego de obtener una especificación de la arquitectura inicial para el software Kinetic siguiendo el Proceso de Diseño Arquitectónico para Software de RV, se aplicó una encuesta a los diseñadores del mismo, persiguiendo los siguientes objetivos: (Ver preguntas de la encuesta: Anexo 10)

- ✓ Determinar si la arquitectura inicial propuesta para Kinetic contribuye a facilitar el trabajo de los diseñadores de este software.

- ✓ Determinar si la arquitectura inicial propuesta contribuye a obtener diseños finales de mayor calidad.

3.7 Resultados de las Encuestas

Luego de aplicar las encuestas diseñadas a los involucrados en la validación de la propuesta de solución (Líder de proyecto y Arquitecto de Software de Kinetic) se obtuvieron los siguientes resultados: (Ver las respuestas de las encuestas en los Anexo 11)

- ✓ Para las 3 primeras preguntas el 100% de los encuestados estuvo de acuerdo en que la utilización del proceso propuesto para diseñar la arquitectura del software Kinetic contribuyó a facilitar el trabajo del arquitecto de software, así como a disminuir el tiempo y esfuerzo requerido para realizar esta tarea.

- ✓ Para las preguntas 4 y 5 el 100% de los encuestados coincide en que el proceso definido se ajusta a las características y necesidades actuales de un software de Realidad Virtual y además contribuye a realizar un diseño arquitectónico con mayor eficacia en este tipo de software.

- ✓ Para la pregunta 7 el 100% de los encuestados considera que la propuesta de solución posee un alto nivel de creatividad y efectividad.

Los involucrados en el proceso de validación consideran además que la propuesta de solución es una guía indispensable para diseñar la arquitectura en un software de RV y que contribuyó en gran medida para definir la arquitectura del producto Kinetic.

La arquitectura inicial propuesta para Kinetic fue sometida a la valoración de los diseñadores del proyecto ARLab a través de la aplicación de la encuesta diseñada y se obtuvieron los siguientes resultados: (Ver las respuestas de las encuestas en los Anexo 12)

- ✓ Para las 3 preguntas de la encuesta el 100% de los diseñadores considera que la arquitectura inicial definida para este software contribuye en gran medida el trabajo en cuanto a diseño detallado y a obtener diseños de mayor calidad final en un período corto de tiempo.

A partir de los resultados obtenidos se pudo determinar que realmente la propuesta de solución cumple con el objetivo general propuesto en esta investigación.

3.8 Conclusiones Parciales

A partir de los objetivos propuestos para este capítulo definidos en la introducción se puede decir que:

- ✓ Se definió la arquitectura inicial de un software de RV (Kinetic) que se desarrolla en la Facultad 5 realizando una primera iteración de las dos primeras actividades del proceso propuesto en el capítulo anterior.
- ✓ Se realizaron diferentes modificaciones en el proceso inicial presentado a los involucrados en la validación, teniendo en cuenta sus críticas y sugerencias.
- ✓ Se aplicaron encuestas a los involucrados en la validación de la propuesta de solución.
- ✓ Se aplicaron encuestas a los diseñadores de Kinetic luego de facilitarles la arquitectura inicial definida para este software.

Concluyendo que:

- ✓ La utilización del proceso propuesto contribuyó a facilitar el trabajo del arquitecto de software, así como a disminuir el tiempo y esfuerzo requerido para diseñar la arquitectura del software Kinetic.
- ✓ El 100% de los encuestados considera que el proceso definido se ajusta a las características y necesidades actuales de un software de Realidad Virtual y que además contribuye a realizar un diseño arquitectónico con mayor eficacia en este tipo de software.
- ✓ El 100% de los diseñadores de Kinetic considera que la arquitectura inicial propuesta para este software contribuirá a facilitar su trabajo y a obtener un producto de mayor calidad final.
- ✓ Se confirmó que la propuesta de solución cumple con el objetivo general planteado en la investigación.

Conclusiones

Posterior al estudio del proceso de Diseño Arquitectónico planteado en las diferentes Metodologías de Desarrollo de Software estudiadas (RUP, XP, MSF Ágil y FDD) se agruparon una serie de pasos, principios y prácticas planteados en las mismas que se ajustaban a las características específicas de un software de RV. Luego de haber analizado el proceso en los distintos proyectos que pertenecen al polo de Realidad Virtual de la Facultad 5 a través de entrevistas realizadas a los Líderes de Proyecto y Arquitectos de Software y del análisis de los Documentos de Arquitectura y Especificación de Requisitos de estos proyectos, se detectaron una serie de deficiencias y se recopilaron gran cantidad de sugerencias en función de la elaboración de un proceso futuro.

Seguidamente se elaboró una propuesta de Proceso de Diseño Arquitectónico para Software de RV, la cual tiene en cuenta todo lo planteado anteriormente y se caracteriza además por combinar la disciplina de Administración de Riesgos que propone MSF Ágil y la elaboración de una visión inicial del sistema como propone RUP, incorporando diferentes principios planteados en las metodologías ágiles estudiadas. Es un proceso corto, que propone 3 actividades básicas y una serie de prácticas asociadas, las cuales son ajustables al proyecto que se desarrolle.

Finalmente se utilizó la propuesta de solución para diseñar la arquitectura inicial del Software Kinetic, el cual se desarrolla actualmente en el polo de RV de la Facultad 5. De esta manera el proceso inicial propuesto fue llevado a la práctica y se le realizaron una serie de modificaciones provenientes de las críticas y sugerencias planteadas por la arquitecta de software y el líder del proyecto ARLab. Se realizó además una encuesta a los involucrados en la validación del proceso y se pudo determinar que el 100% de los mismos considera que el proceso definido se ajusta a las características y necesidades actuales de un software de Realidad Virtual, verificando de esta manera que la propuesta de solución cumple con el objetivo general planteado en la investigación.

Recomendaciones

1. Extender la utilización del proceso propuesto a otros proyectos pertenecientes al polo de Realidad Virtual de la Facultad 5 de la Universidad de las Ciencias Informáticas, para lograr una mejora del mismo teniendo en cuenta las necesidades y características específicas de estos proyectos.

2. Someter la propuesta de Proceso de Diseño Arquitectónico para Software de Realidad Virtual a la valoración de todos los arquitectos de software de los proyectos pertenecientes al polo de RV de la Facultad 5 a través de talleres de discusión, para que puedan señalar sus deficiencias y proponer mejoras.

3. Realizar un estudio en el polo de Hardware y Automática de la Facultad 5, para analizar qué características del proceso propuesto son extensibles al mismo.

Referencias Bibliográficas

Arias, Y., & Martínez, Y. (2008). “Adaptación de Microsoft Solutions Framework Agile (MSF Ágil) al proceso de desarrollo de videojuegos.”

Aukstakalnis. (1992). “The Silicon Mirage.”

Ayuda Rational. (2003).

Bauta Gómez, R. M., & Castillo Barbosa, D. (2007). “Propuesta de estrategia ágil para el desarrollo de videojuegos.”

Calabria, L. (2003). “Metodología FDD.” Universidad ORT Uruguay.

Obtenido de: http://athenea.ort.edu.uy/publicaciones/ingsoft/investigacion/ayudantias/metodologia_FDD.pdf

Clements, P. (1996). “A Survey of Architecture Description Languages. Proceedings of the International Workshop on Software Specification and Design."

Evans, J. R., & Lindsay, W. (2000). “Administración y Control de la Calidad.” México: International Thomson.

Guelmes León, Y., & Carballo Muñoz, L. (2009). “Procedimiento para evaluar Arquitecturas de Software.” Universidad de las Ciencias Informáticas.

IEEE-1471. (2000). “Recommended Practice for Architectural Description of Software-Intensive Systems.”

Obtenido de: http://www.enterprise-architecture.info/Images/Documents/IEEE%201471-2000.pdf

ISSI. (2003). “Metodologías Ágiles en el Desarrollo de Software.” Alicante.

Obtenido de: http://www.willydev.net/descargas/prev/TodoAgil.pdf

ISO/IEC. (2002). Software Engineering – Software quality – General overview, reference models and guide to Software Product Quality Requirements and Evaluation (SQuaRE). Reporte JTC1/SC7/WG6.

Jacobson, I. (1998). “Applying UML in the Unified Process”.

Jacobson, I., Booch, G., & Rumbaugh, J. (1999). “El Proceso Unificado de Desarrollo de Software.”

Krajewki, L. J., & Ritzman, L. P. (2000). “Administración de Operaciones.”

Kruchten, P. (1995). “The 4+1 View Model of Architecture”.

Pita, Y., & Álvarez, Y. (2007).“Determinación de los roles, responsabilidades y conocimientos necesarios para el Proceso de Producción de Software de Realidad Virtual.”

Platt, M. (2002). “Microsoft Architecture Overview: Executive summary”.

Pressman, R. S. (2002). “Ingeniería de Software: Un enfoque Práctico”.

programacionextrema.org (octubre de 2002). Obtenido de http://www.programacionextrema.org/articulos/designdead.es.html

Reinoso, C. B. (2004). “Introducción a la Arquitectura de Software.” Universidad de Buenos Aires. Obtenido de:

http://download.microsoft.com/download/4/F/F/4FF88340-43CC-4C5B-8E50-9002969D0DD/20051122-ARC-BA.ppt

Ross, D. (1977). “Structured analysis (SA): A language for communicating ideas”. IEEE Transactions on Software Engineering.

Rumbaugh, J., Jacobson, I., & Booch, G. (1999). “El Lenguaje Unificado de Modelado.”

Schmuller, J. (2000). “UML en 24 horas. México.”

SEI. Software Engineering Institute. [En línea] http://www.sei.cmu.edu/.

Shaw, M., & Garlan, D. (1996). “Software Architecture. Perspectives of an Emerging Discipline.”

Spaces, W. L. (2005). "Qué hay para decir de MSF (Microsoft Solution Framework)."

Szyperski, C. (2000). “Components and Architecture.” Software Development Online.

Trujillo, I. L. (2007). “Propuesta de Proceso de Software para sistemas de Realidad Virtual.”

Wolf, A. L., & Perry, D. E. (1992). “Foundations for the study of software architecture.”

Zavalar, J. (2008). “Ingeniería de Software.”

Obtenido de: http://www.angelfire.com/scifi/jzavalar/apuntes/IngSoftware.html

Bibliografía

Arquitectura de Software: Diseño de Sistemas. Obtenido de:

httpwww.eici.ucm.clAcademicosRVillarroeldescargasingsw1ArquitecturaSoftware.pdf

Canós, J. H., Letelier, P., & Penadés, M. d. "Metodologías Ágiles en el desarrollo de Software." Universidad Politécnica de Valencia. Obtenido de:

http://www.willydev.net/descargas/prev/TodoAgil.pdf

Capote, Y. D. (2009). "Lineamientos para los Arquitectos. Área Temática de Videojuegos, Facultad 5, UCI.

Colombia, M. C. (2000). "Microsoft Solutions Framework (MSF): Disciplinas y buenas prácticas para el desarrollo e implantación de proyectos."

Consultores, G. 2006. Disciplina de administración del proyecto - M.S.F. [Online] 2006.

Obtenido de: http://www.gpicr.com/msf.aspx.

Meda, R., & Ierache, J. "Una Propuesta de Conjunción de Elementos Metodológicos en común dentro de los Enfoques ágiles para el Desarrollo de Software." Buenos Aires: Facultad de Informática, Ciencias de la Comunicación y Técnicas Especiales.

Mendoza Sánchez, M. A. (2004). "Metodologías de Desarrollo de Software." Perú. Obtenido de:

http://www.willydev.net/descargas/cualmetodologia.pdf

Microsoft.com. (s.f.). Obtenido de:

http://www.microsoft.com/colombia/portafolio/msf.htm

Microsoft. (2007). "Metodologías Ágiles en el Desarrollo de Software y la propuesta de Microsoft -Ser o no ser ágil, he ahí el dilema."

Obtenido de: http://www.microsoft.com/colombia/empresas/clientepreferencial/actualizacion-gerencial/febrero/innovaciones2.mspx

Molpeceres, A. (2003). "Procesos de desarrollo: RUP, XP y FDD."

Obtenido de: http://www.willydev.net/descargas/articulos/general/cualxpfddrup.PDF

Reinoso, C. B. “Métodos Ágiles en Desarrollo de Software.” Universidad de Buenos Aires.

Zeledón, J. “Conceptos y Evolución de la Ingeniería del Software.”

Obtenido de: http://html.rincondelvago.com/conceptos-y-evolucion-de-la-ingenieria-del-software.html

Anexos

Anexo 1 Proyectos de RV Facultad 5

Identificador	Área Temática
01	Video Juegos
02	Video Juegos
03	Video Juegos
04	Video Juegos
05	Simuladores
06	Simuladores
07	Simuladores
08	Simuladores
09	RV para la Salud
010	RV para la Salud
011	Diseño y Animación
012	Diseño y Animación
013	Diseño y Animación
014	Diseño y Animación

Tabla 6: Listado de proyectos de realidad virtual de la Facultad 5. (Se omite el nombre de los proyectos para no revelar los problemas específicos existentes en cada uno y mantener la privacidad de los mismos)

Anexo 2 Entrevistas

Entrevista 1: Realizada a los Arquitectos de Software de los Proyectos de Realidad Virtual de la Facultad 5.

Preguntas:

1) ¿En qué etapa se encuentra el proyecto?
2) ¿Requiere el proyecto que se especifique la Arquitectura de Software? En caso de que la respuesta sea negativa, ¿por qué?
3) ¿Siguen alguna metodología o proceso definido para diseñar la arquitectura?
4) ¿Considera que la misma se adecúa al tipo de proyecto? ¿Por qué?
5) ¿Tienen por escrito este proceso o metodología?
6) ¿Qué nivel de detalle o especificación posee la arquitectura en su proyecto?
7) ¿Qué información o artefactos de entrada utilizan?
8) A grandes rasgos, ¿qué pasos o actividades realizan para diseñar la arquitectura?
9) ¿Generan alguna salida diferente o adicional al documento de arquitectura?
10) ¿Qué conocimientos y habilidades considera debe poseer un arquitecto de software?

Entrevista 2: Realizada a los Programadores y Diseñadores de los Proyectos de Realidad Virtual de la Facultad 5.

Preguntas:

1) ¿Qué información de entrada reciben para comenzar su trabajo?
2) ¿Consideran útil el Documento de Arquitectura?
3) ¿Trabajan a partir de él y lo respetan?
4) ¿Es útil el nivel de especificación de la arquitectura?

Anexo 3 Definición de objetivos y posibles respuestas de las entrevistas

Objetivos (entrevistas a los Arquitectos de Software)

1. Conocer si el proyecto analizado requiere que se especifique la Arquitectura de Software.

   Pregunta no: 2

   Posibles Respuestas:

   - Si

- No

2. Conocer la Metodología seleccionada para el desarrollo del software y analizar si la misma satisface las necesidades del proyecto.

   Preguntas no: 3, 4, 5

   Posibles Respuestas:

   Preguntas 4 y 5:

   - Si
   - No

3. Conocer artefactos de entrada y salida del proceso de diseño arquitectónico actual.

   Preguntas no: 7, 9

4. Determinar el nivel de detalle con que se especifica la arquitectura de software en los proyectos analizados.

   Pregunta no: 6

   Considerando las siguientes especificaciones de la arquitectura de software:

   Especificaciones Básicas:

   - Identificación de RF y/o Casos de Uso críticos para la arquitectura o con algún tipo de priorización.
   - Existencia del Diagrama de Clases o equivalente.
   - Elección de un estilo de agrupación.
   - Especificación de los componentes con sus respectivos estereotipos.
   - Descripción de los elementos de despliegue.
   - Existencia del Modelo de Datos si el proyecto lo requiere.

- Descripción de los RNF significativos para la arquitectura.

Especificaciones Adicionales:

- Descripción de elementos significativos del diagrama de clases o equivalente.
- Especificación de las relaciones entre las agrupaciones.
- En caso de aplicar patrones fundamentar su utilización.
- Especificación de las relaciones entre componentes.
- Descripción de las relaciones entre los elementos de despliegue.
- Mapeo o relación de los elementos de despliegue con los componentes de implementación.
- Descripción de elementos significativos del modelo de datos.
- Descripción de las restricciones arquitectónicas.

Se consideran las siguientes clasificaciones:

- Alto: Se considera un nivel alto cuando la arquitectura cumple con la totalidad de las especificaciones básicas y al menos cinco de las adicionales.
- Medio: Se considera un nivel medio cuando la arquitectura cumple con al menos cuatro de las especificaciones básicas y al menos dos de las adicionales.
- Bajo: Se considera un nivel bajo cuando la arquitectura cumple con al menos cuatro de las especificaciones básicas.
- Muy Bajo: Se considera un nivel muy bajo cuando la arquitectura cumple con al menos una de las especificaciones básicas.
- Sin especificar.

5. Conocer los pasos fundamentales para diseñar la arquitectura en este tipo de software, así como si el desarrollo de este proceso va en correspondencia con la etapa en la que se encuentra el proyecto.

Preguntas no: 1, 8

Posibles Respuestas:

Pregunta 8:

- Descripción de los pasos
- Se realiza mientras se escribe el código.
- Se realizó una ingeniería inversa.

6. Determinar si los arquitectos de software de estos proyectos poseen la experiencia necesaria y los conocimientos acerca del alcance de la responsabilidad del rol que desempeñan.

   Pregunta no: 10

   Posibles Respuestas:

   Considerando que un Arquitecto de Software debe poseer los siguientes conocimientos y habilidades:

   Conocimientos:

   - Comprensión del dominio del problema
   - Lenguajes de Programación
   - Patrones de Arquitectura
   - Estilos Arquitectónicos
   - ADLs o UML
   - Frameworks Arquitectónicos
   - Metodologías de Desarrollo de Software
   - Herramientas de Desarrollo de Software
   - RNF

Habilidades:

- Visión Técnica
- Pensamiento Conceptual
- Experiencia en todo el ciclo de vida del desarrollo de software.
- Saber trabajar en equipo.
- Habilidad para convencer y guiar al equipo.
- Buen comunicador.

Para determinar si los Arquitectos de Software de los proyectos de Realidad Virtual de la Facultad 5 conocen el alcance de la responsabilidad del rol que desempeñan las repuestas se clasificarán de la siguiente manera:

- Elevados: Deben mencionar al menos diez de los conocimientos o habilidades.
- Medios: Deben mencionar al menos cuatro de los conocimientos o habilidades.
- Escasos: Deben mencionar al menos uno de los conocimientos o habilidades.

Considerando que un Arquitecto de Software posee experiencia cuando ha participado directamente en al menos 2 proyectos de software grandes, donde existan requisitos conflictivos entre si de parte de varios participantes (Stakeholders) o posea más de 5 años de experiencia trabajando en proyectos de Desarrollo de Software.

Se clasificará de la siguiente manera:

- Experiencia: Cumple con lo expuesto anteriormente.
- Poca experiencia: No cumple con lo expuesto anteriormente.

Objetivos (entrevista a los diseñadores)

1. Conocer los artefactos de entrada que reciben los diseñadores para comenzar a trabajar.

Pregunta no: 1

2. Conocer si los Diseñadores de los proyectos analizados consideran útil el nivel de especificación de la arquitectura, así como el Documento de la Arquitectura.

Pregunta no: 2, 4

Posibles Respuestas:

- Si
- No

3. Conocer si los Diseñadores de los proyectos analizados trabajan a partir del Documento de Arquitectura y lo respetan.

Pregunta no: 5

Posibles Respuestas:

- Si
- No

Objetivos (entrevista a los programadores)

1. Conocer los artefactos de entrada que reciben los programadores para comenzar a trabajar.

Pregunta no: 1

2. Conocer si los Programadores de los proyectos analizados consideran útil el nivel de especificación de la arquitectura, así como el Documento de la Arquitectura.

Pregunta no: 2, 4

Posibles Respuestas:

- Si
- No

3. Conocer si los Programadores de los proyectos analizados trabajan a partir del Documento de Arquitectura y lo respetan.

Pregunta no: 5

Posibles Respuestas:

- Si
- No

Anexo 4 Parámetros para evaluar Documentos

Objetivo	Parámetros
1. Especificación y Claridad de los Requisitos Funcionales (RF) y No Funcionales (RNF) prioritarios o críticos para la Arquitectura.	1.1 Existencia del documento de Arquitectura.
	1.2 Existencia del documento de especificación de RF y RNF.
	1.3 Identificación de RF y/o Casos de Uso (CU) críticos para la arquitectura o con algún tipo de priorización.
	1.4 Priorización de RF y/o CU adecuada para el tipo de Software.
	1.5 CU, RF (críticos) y RNF (claves para la arquitectura) claramente especificados (posibles de verificar).
2. Nivel de detalle de la Vista Lógica o estructural.	2.1 Determinar si es necesaria la vista.
	2.2 Existencia del Diagrama de Clases o equivalente.
	2.3 Descripción de elementos significativos del diagrama o equivalente.
	2.4 Elección de un estilo de agrupación y fundamentación de la elección.
	2.5 Especificación de las relaciones entre las agrupaciones.
	2.6 Patrones utilizados y su fundamentación.
3. Nivel de detalle de la Vista de Implementación.	3.1 Determinar si es necesaria la vista.
	3.2 Especificación de los componentes con sus respectivos estereotipos.

	3.3 Especificación de las relaciones entre componentes.
4. Nivel de detalle de la Vista de Despliegue	4.1 Determinar si es necesaria la vista.
	4.2 Descripción de los elementos de despliegue.
	4.3 Descripción de las relaciones entre los elementos de despliegue.
	4.4 Mapeo o relación de los elementos de despliegue con los componentes de implementación.
5. Nivel de detalle de la Vista de Datos	5.1 Determinar si es necesaria la vista.
	5.2 Existencia del Modelo de Datos.
	5.3 Descripción de elementos significativos del modelo.
	5.4 Elección de un estilo de agrupación y fundamentación de la elección.
	5.5 Especificación de las relaciones entre las agrupaciones.
6. Restricciones Arquitectónicas	6.1 Descripción de las restricciones arquitectónicas.
	6.2 Descripción de los RNF significativos para la arquitectura.

Tabla 7: Parámetros para evaluar los Documentos de Arquitectura de Software de los proyectos de RV.

Anexo 5 Resultados de las entrevistas realizadas a los Arquitectos de Software.

	Proyectos							
Preg.	**01**	**02**	**03**	**04**	**07**	**08**	**09**	**013**
1	Terminado	Terminado	Terminado	Elaboración	Segunda iteración (Elaboración y Construcción)	Inicio	Terminada fase de Habilidades Básicas	Desarrollo

2	Si	Si	Si	Si	Si	Si	Si	Si, aunque no de forma tan rigurosa.
3	RUP	RUP	RUP	RUP	RUP adaptada a las necesidades del proyecto.	RUP	RUP	Formalmente RUP
4	No se adecúa pero no dominan otra.	No se adecúa pero no dominan otra.	No se adecúa pero no dominan otra.	No se adecúa pero no dominan otra.	Si, estamos aplicando RUP pero adaptándolo a nuestras necesidades.	No se adecúa pero no dominan otra.	No se adecúa pero no dominan otra.	No se adecúa por eso en la práctica hacemos una combinación de XP, RUP, FDD y Crystal Clear.
5	Si	Si	Si	Si	Si	Si	Si	No
6	Medio	Medio	Bajo	Medio	Medio	Medio	Medio	Sin especificar
7	RF y RNF	RF y RNF	RF y RNF	Descripción de CU. RF y RNF	RF y RNF	RF y RNF	RF y RNF	RF y RNF
8	Ingeniería Inversa.	Ingeniería Inversa.	Ingeniería Inversa.	Estudiar Herramientas, Plantear arquitectura propia y Realizar el diseño de cada módulo.	Se realiza mientras se programa.	Ingeniería Inversa.	Documentarse Definir la Metodología Establecer la línea base de la arquitectura.	No se ha realizado a pesar de la etapa en la que se encuentra el proyecto.
9	No	No	No	No	No	No	No	No
10	Rol de mayor conocimiento Lenguajes de programación.	Dominio del problema.	Dominio del problema. Metodologías. Lenguajes de Programación Patrones de	Patrones de arquitectura y Experiencia.	Patrones de arquitectura, Herramientas, Metodologías. Lenguajes de	Tecnologías Herramientas RNF Lenguajes de programación.	Conocimientos de arquitectura de Software Herramientas	Experiencia. Herramientas. Metodologías.

			arquitectura.		Programación			

Tabla 8: Resultados de las entrevistas realizadas a los arquitectos de software.

Anexo 6 Resultados de las entrevistas realizadas a los Diseñadores.

	Proyectos							
Preg.	**01**	**02**	**03**	**04**	**07**	**08**	**09**	**013**
1	RF	RF (de forma verbal) Guión Técnico	RF Guión Técnico	RF	RF Guión Técnico	RF	RF	RF (de forma verbal)
2	Si	Si, aunque depende del tipo de proyecto.	Si	Si Se especifican patrones, paquetes.	Si	Si, aunque depende del tipo de proyecto.	Si	En este proyecto muy poco.
3	No Ingeniería Inversa.	No Ingeniería Inversa.	No Ingeniería Inversa.	Si	No, nunca he visto el de mi proyecto.	No Ingeniería Inversa.	No, nunca he visto el de mi proyecto.	No, aunque deberíamos
4	Si	Si, porque da una visión general del proyecto y ayuda a diseñar a bajo nivel.	Si	Si	Si	No	Si	Si, aunque depende del tipo de proyecto.

Tabla 9: Resultados de las entrevistas realizadas a los diseñadores.

Anexo 7 Resultados de las entrevistas realizadas a los Programadores.

	Proyectos							
Preg.	**01**	**02**	**03**	**04**	**07**	**08**	**09**	**013**
1	Requisitos (de forma verbal)	Guión Técnico	Guión Técnico	Requisitos	Requisitos (de forma verbal) Guión Técnico	Requisitos (de forma verbal)	Requisitos (de forma verbal)	Requisitos (de forma verbal)
2	Si	Si	Si, porque siempre que se hace la arquitectura se logran mejores resultados.	Si, porque recoge la lógica de lo que se va a hacer y a partir de la arquitectura sale la implementación.	Si	Si, porque cuando uno codifica, traduce a código fuente lo que te da la arquitectura.	Si	Si
3	No, nunca he visto el de mi proyecto.	No, nunca he visto el de mi proyecto.	No, nunca he visto el de mi proyecto.	Si	No, nunca he visto el de mi proyecto.	No, nunca he visto el de mi proyecto.	No, nunca he visto el de mi proyecto.	No, nunca he visto el de mi proyecto.
4	Si	Si	Si	Si	Si	Si	Si	Si

Tabla 10: Resultados de las entrevistas realizadas a los programadores.

Anexo 8 Otras Observaciones.

Proyecto	Otras Observaciones
01	El Arquitecto de Software es una estudiante de 3er año y no conoce prácticamente nada del tema, por lo que es el líder quien realizó el diseño arquitectónico en el proyecto.
03	El líder de proyecto fue quien diseñó toda la arquitectura del software. No posee Documento de Arquitectura de Software.

07	El líder de proyecto es quien realiza el diseño arquitectónico del software. Toda la arquitectura se encuentra especificada aunque a pesar de la fase en la que se encuentra el proyecto la misma no esta registrada en un documento formal.
013	El Arquitecto de Software es una estudiante de 2do año que no conoce prácticamente nada del tema, por lo que es el líder quien realiza el diseño arquitectónico en el proyecto. A pesar de la etapa en la que se encuentra el proyecto no se ha especificado aún la arquitectura de software.

Tabla 11: Otras observaciones de los proyectos de RV.

Anexo 9 Análisis de Documentos

	Objetivos					
Proy.	1	2	3	4	5	6
01	1.1 Si.	2.1 Si.	3.1 Si.	4.1 Si.	5.1 En esta versión del software no se tienen elementos persistentes arquitectónicamente significativos en el modelo de datos	6.1 Descripción de las restricciones arquitectónicas escasa de información.
	1.2 No posee descripción de los RNF.	2.2 Diagrama de elementos arquitectónicamente significativos, diagrama de paquete que ilustra la organización del modelo de diseño.		4.2 Los elementos de despliegue se encuentran especificados.		6.2 Descripción de los RNF arquitectónicamente significativos escasa de información.
	1.3 Identificados los CU arquitectónicamente significativos.	2.3 No presenta descripción.	3.2 Diagrama de componentes con sus estereotipos.	4.3 Se describen las relaciones entre los elementos de despliegue.		
		2.4 Estilo en capas y no presenta fundamentación.				

	1.4 Priorización adecuada de CU.	2.5 No se especifican las relaciones entre capas.	3.3 No se especifican.	4.4 No presenta especificada la relación entre los elementos de despliegue y los componentes de implementación.		
	1.5 RF redactados de forma correcta y son posibles de verificar.	2.6 Patrón Modelo de 3 Capas sin fundamentación.				
02	1.1 Si.	2.1 Si	3.1 No se encuentra en el documento la vista de implementación.	4.1 No se encuentra en el documento la vista de despliegue.	5.1 No requiere de la Vista de Datos.	6.1 Descripción de las restricciones arquitectónicas escasa de información.
	1.2 Si.	2.2 Diagrama de paquetes arquitectónicamente significativos y Diagrama general de Clases del sistema.				6.2 No
	1.3 Identificados los CU arquitectónicamente significativos.	2.3 Presenta descripción.				
	1.4 Priorización adecuada de CU y RF.	2.4 En paquetes y sin fundamentación.				
	1.5 RF y RNF redactados de forma correcta y son posibles de verificar.	2.5 No se especifican.				
		2.6 No se especifica la utilización de algún patrón.				
03	Este proyecto cuenta con el documento de Arquitectura de Software y se especifican los RF y RNF. No posee identificados los CU arquitectónicamente significativos. Los RF y RNF se encuentran redactados de forma correcta y son posibles de verificar.					
04	1.1 Si.	2.1 Si.	3.1 Si.	4.1 No es	5.1 Si.	6.1 Descripción de las

				necesario mostrar el despliegue debido a que consta de una PC.	5.2 En el documento no se encuentra el modelo de datos.	restricciones arquitectónicas escasa de información.
	1.2 Si.	2.2 Diagrama de elementos arquitectónicamente significativos.				
	1.3 Identificados los CU y requisitos funcionales arquitectónicamente significativos.	2.3 No presenta descripción.	3.2 Diagrama de componentes sin especificación de sus estereotipos correspondientes.		5.3 Descripción muy escasa.	6.2 Descripción de los RNF arquitectónicamente significativos escasa de información.
					5.4 No se especifica.	
					5.5 No se describe.	
	1.4 Priorización adecuada de CU y RF.	2.4 Alineamiento en subsistemas, paquetes y capas, no presenta fundamentación.				
	1.5 RF y RNF redactados de forma correcta y son posibles de verificar.	2.5 No se especifican las relaciones entre capas.	3.3 No se especifican.			
		2.6 Patrón Modelo de 3 Capas sin fundamentación.				
07	Este proyecto no cuenta con el documento de Arquitectura de Software, solamente se especifican los RF y RNF. No se identifican los RF y/o Casos de Uso críticos para la arquitectura. Los RF y RNF están redactados de forma correcta y son posibles de verificar.					
08	1.1 Si.	2.1 Si.	3.1 Si.	4.1 Si.	5.1 La herramienta no requiere de ningún elemento persistente para datos.	6.1 Descripción de las restricciones arquitectónicas escasa de información.
	1.2 Si.	2.2 Organización en paquetes y subsistemas,		4.2 Los elementos de despliegue se encuentran		

		diagrama con las clases arquitectónicamente significativas del modelo de diseño.		especificados.		6.2 Descripción de los RNF arquitectónicamente significativos escasa de información.
	1.3 Identificados los CU arquitectónicamente significativos.	2.3 Presenta descripción.	3.2 No están especificados los componentes.	4.3 Se describen las relaciones entre los elementos de despliegue.		
	1.4 Priorización adecuada de CU.					
	1.5 RF redactados de forma correcta y son posibles de verificar. Los RNF presentan algunas ambigüedades, los RNF de eficiencia son imposibles de probar.	2.4 Estilo en capas y no presenta fundamentación.				
		2.5 Especificadas las relaciones entre capas.	3.3 No se especifican.	4.4 No presenta especificada la relación entre los elementos de despliegue y los componentes de implementación.		
		2.6 Patrón Modelo de 3 Capas sin fundamentación.				
09	1.1 Si.	2.1 Si.	1.3 Si	4.1 Si.	5.1 En esta versión del software no se tienen elementos persistentes arquitectónicamente	6.1 Descripción de las restricciones arquitectónicas escasa de información.
	1.2 Si.	2.2 Diagrama de elementos arquitectónicamente significativos y		4.2 Los elementos de despliegue se		

		diagrama de paquetes.		encuentran especificados.	significativos en el modelo de datos.	6.2 No
	1.3 Identificados los CU arquitectónicamente significativos.	2.3 Presenta descripción.	3.2 Posee diagrama de Componentes con estereotipos.	4.3 No se describen las relaciones entre los elementos de despliegue.		
	1.4 Priorización adecuada de CU.	2.4 Estilo en capas y no presenta fundamentación.				
	1.5 RF redactados de forma correcta y son posibles de verificar. Los RNF de Rendimiento presentan algunas ambigüedades y son imposibles de probar.	2.5 No se especifican las relaciones entre capas.	3.3 No tiene las relaciones especificadas.	4.4 No presenta especificada la relación entre los elementos de despliegue y los componentes de implementación.		
		2.6 Patrón Modelo de 3 Capas sin fundamentación.				
013	Este proyecto no cuenta con el documento de Arquitectura de Software, solamente se especifican los RF y RNF. No se identifican los RF y/o Casos de Uso críticos para la arquitectura. Los RF están redactados de forma correcta y son posibles de verificar. Los RNF de Eficiencia presentan algunas ambigüedades y son imposibles de probar.					

Tabla 12: Análisis de documentos.

Anexo 10 Encuestas

Encuesta 1: Realizada a los involucrados en la validación del proceso propuesto (Líder de Proyecto y Arquitecto de Software de Kinetic).

Preguntas:

1. ¿Contribuyó el proceso propuesto a facilitar el trabajo en cuanto a diseño arquitectónico del software Kinetic?
   Si____ No_____

2. ¿Contribuyó el proceso propuesto a disminuir el tiempo empleado para diseñar la arquitectura del software Kinetic?
   Si____ No_____

3. ¿Contribuyó el proceso propuesto a disminuir el esfuerzo para diseñar la arquitectura de Kinetic?
   Si____ No_____

4. ¿Considera usted que la propuesta de Proceso de Diseño Arquitectónico para Software de Realidad Virtual se ajusta de manera general a las necesidades reales de los proyectos de Realidad Virtual?
   Si____ No_____

5. ¿Considera usted que se pueda diseñar la arquitectura de un software de RV con mayor eficacia si se utiliza el proceso propuesto?
   Si____ No_____

6. ¿Qué valoración en general diera usted a la propuesta desarrollada?

7. ¿En que escala del 1 al 3 colocaría usted la propuesta desarrollada?

a) 1: nivel bajo de creatividad y efectividad.
b) 2: nivel medio de creatividad y efectividad.
c) 3: nivel alto de creatividad y efectividad.

Encuesta 2: Realizada a los diseñadores del software Kinetic

Preguntas:

1. ¿Considera usted que la especificación de la arquitectura inicial propuesta para el software Kinetic contribuye a facilitar el trabajo en cuanto a diseño detallado?

   Si____ No_____

2. ¿Contribuye esta arquitectura inicial propuesta a disminuir el esfuerzo de los diseñadores para realizar su trabajo?

Si____ No_____

3. ¿Contribuye esta arquitectura inicial propuesta a obtener diseños de mayor calidad?

   Si____ No_____

Anexo 11 Resultados de las encuestas realizadas a los involucrados en la validación del proceso

Preguntas	Arquitecto de Software	Líder de Proyecto
1	Si	Si
2	Si	Si
3	Si	Si
4	Si	Si
5	Si	Si
6	Es una guía indispensable para los proyectos de realidad virtual, fundamentalmente por la poca experiencia que existe de estos temas en el polo.	En general la propuesta desarrollada tuvo una gran influencia para definir la arquitectura del producto Kinetic del Laboratorio de Realidad Aumentada. Contribuyó a reducir los esfuerzos realizados por el equipo de desarrollo en la definición de la arquitectura, aunque se recomienda que se haga una prueba más exhaustiva en varios proyectos que utilicen Realidad Virtual para asentar mejor los procesos y actividades que se proponen, porque aunque se tuvieron buenos resultados, el proyecto tiene componentes de Realidad Virtual pero no está centrado en esa tecnología y se tuvieron en cuenta además otros aspectos que normalmente en otro proyecto de RV no hacen falta, por ejemplo el modulo de visión por computadoras.
7	3: nivel alto de creatividad y efectividad.	3: nivel alto de creatividad y efectividad.

Tabla 19: Resultados de las encuestas realizadas a los involucrados en la validación del proceso

Anexo 12 Resultados de las encuestas realizadas a los diseñadores del software Kinetic

Preguntas	Diseñador 1	Diseñador 2
1	Si	Si
2	Si	Si
3	Si	Si

Tabla 20: Resultados de las encuestas realizadas a los diseñadores del software Kinetic