UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

TESIS EN OPCIÓN AL TÍTULO ACADÉMICO DE INGENIERO EN CIENCIAS INFORMÁTICAS

Propuesta de arquitectura y proceso de desarrollo para la construcción de aplicaciones web con escenarios tridimensionales interactivos

Autor: Gilberto Cao Tarrero.

Tutor: Ing. Fernando Jiménez López

Ciudad de la Habana, 25 de junio del 2009
"Año del 50 Aniversario del Triunfo de la Revolución"
Curso 2008-2009

En 1961, el Che señaló:

"Estamos entrando en la era de la automatización y de la electrónica; tenemos que pensar en la electrónica en función del socialismo y en el tránsito al comunismo. La electrónica se convierte en un problema político fundamental del país. Hoy y mañana hay que preparar los cuadros para que en el futuro estén listos para tomar en sus manos toda la gran tarea tecnológica posterior y de la automatización cada vez más grande de toda la producción: la liberación del hombre por medio de la máquina."

Era el Che en este tiempo Ministro de Industrias y decidió entonces crear la Dirección de Automatización y Electrónica del Ministerio de Industrias. Subordinada a dicha dirección se creó la Escuela de Automatización, pionera en la formación de graduados de nivel medio y superior en control automático del país.

AGRADECIMIENTOS

A la Revolución por darme la oportunidad de crecer como profesional.

A mi familia por estar desde el principio, esto también es de ellos.

A mi mamá, por ser mi mejor amiga.

A mi papá, por enseñarme de todo un poco.

A mi tía, por siempre darme una luz y enseñarme el camino.

A Ricardo, por tratarme como un hijo más de los tantos que tiene.

A mi hermano y a mi hermana.

A mi abuela, que sin lugar a dudas estaría aquí hoy llorando, “o en la casa preparando la fiesta con Caruca”.

A Jose que viene desde el IPVCE dándose golpes conmigo.

A mi gente de la UCI.

ABSTRACT

The emergence of new communication models using Information Technologies and Communications, has forced developers to raise standards of usability and functionality. Improvements in hardware and software architecture changes have led to the development of more advanced systems such as Rich Internet Applications (RIA), through which can include own desktop applications to a web browser . Due to the demand and the market opportunity it represents for the University of Information Sciences (UCI) to develop these applications, it is of interest for the Production of Virtual Reality Polo, design and implement a production line to include RIA scenarios three-dimensional interactive.

Hence, this paper explores the concepts of software architecture, software architecture domain specific application domain and the components of software engineering, to make a proposal for a software architecture domain and a specific software development process for the production of Rich Internet Applications that include three-dimensional interactive scenarios.

The proposal must contribute to the achievement of high levels of reuse and productivity, shorten development time, run processes organized, structured and disciplined, where the organization is capable of repeated and improved.

RESUMEN

El surgimiento de nuevos modelos de comunicación usando las Tecnologías de la Información y las Comunicaciones, ha obligado a los desarrolladores a elevar los niveles de usabilidad y funcionalidad. Las mejoras en el hardware y cambios de arquitecturas informáticas han conducido al desarrollo de sistemas más avanzados como las Aplicaciones Enriquecidas de Internet (RIA, por sus siglas en inglés), mediante las cuales se pueden incluir prestaciones propias de aplicaciones de escritorio a un navegador web. Debido a la demanda existente y la oportunidad de mercado que representa para la Universidad de Ciencias Informáticas (UCI) el desarrollo de estas aplicaciones, es de intereses para el Polo Productivo de Realidad Virtual, diseñar e implantar una línea de producción de RIA que incluyan escenarios tridimensionales interactivos.

De ahí que el presente trabajo se adentra en los conceptos de arquitectura de software, arquitectura de software de dominio específico, dominio de aplicación y los elementos que componen la ingeniería de software, con el objetivo de hacer una propuesta de una arquitectura de software de dominio específico y un proceso de desarrollo de software para la producción de Aplicaciones Enriquecidas de Internet que incluyan escenarios tridimensionales interactivos.

La propuesta debe contribuir a la obtención de altos niveles de reutilización y productividad, acortar el tiempo de desarrollo, ejecutar procesos organizados, estructurados y disciplinados, donde la organización tenga capacidad para repetirlos y mejorarlos.

ÍNDICE

INTRODUCCIÓN

El siglo XXI se enfrenta a la creciente implantación de la Sociedad de la Información y un nuevo modelo de vida basado en la comunicación hombre-máquina, debido a la presencia constante de los medios digitales y el uso masivo de las Tecnologías de la Información y las Comunicaciones (TIC) en todos los ámbitos. Esto implica la transformación global de los servicios y de los medios de comunicación, propiciando el surgimiento de nuevos modelos como el gobierno electrónico (*e-government*) y el comercio electrónico (*e-business*).

El desarrollo de estos modelos y la necesidad de elevar los niveles de usabilidad y funcionalidad de la aplicaciones para estos fines, mejoras en el hardware y cambios de arquitecturas informáticas han conducido al desarrollo de sistemas más avanzados y más complejos como las Aplicaciones Enriquecidas de Internet, conocidas por sus siglas en inglés RIA (*Rich Internet Applications*). Este término surgió en el año 2001, para agrupar aplicaciones soportadas por navegadores web y poseedoras de las ventajas de las aplicaciones tradicionales de escritorio.

El uso de aplicaciones web que permitan una interacción rápida y amigable con los clientes es un tema de interés constante de las empresas, en busca de elevar los mercados, las ventas o el número de clientes aprovechando sus ventajas. El aumento de la demanda de estas solicitudes ha crecido así como las expectativas de los clientes, hecho que obliga a los desarrolladores a buscar nuevas soluciones. Una de las alternativas es la inclusión de escenarios tridimensionales interactivos en la web, los cuales son cada vez más demandados en los sitios de Internet dirigidos al entretenimiento, entrenamiento, simulación, educación, defensa, medicina, arquitectura y la comercialización.

Cuba, como resultado del correcto aprovechamiento de las ventajas del alto nivel de preparación del capital humano disponible y en el camino por la invulnerabilidad económica, ha encontrado en la informática una posible base de desarrollo para el país y una fuente de ingresos. Por ello durante los últimos años se han ejecutado variadas estrategias, con el fin de fortalecer la industria del software, reorganizando, industrializando y diversificando sus mercados, con el objetivo de elevar la informatización masiva y ordenada de la sociedad y los niveles de exportación de software y servicios informáticos.

Una de las estrategias es la creación de la Universidad de Ciencias Informáticas (UCI). La

misma constituye un nuevo modelo de formación-investigación-producción en el campo de las TIC, basándose en la vinculación estudio-trabajo introduce un nuevo concepto universidad-productiva como modelo de formación. En esta institución los estudiantes y profesores se encuentran vinculados a proyectos productivos y sus resultados contribuyen al desarrollo económico, político y social del país. Uno de los polos productivos de la Universidad se encarga del desarrollo de aplicaciones de Realidad Virtual (RV), el cual ha puesto a disposición de los clientes productos tanto de simulación profesional como para el entretenimiento y la educación, todos desarrollados como aplicaciones de escritorio.

Debido a la demanda existente y la oportunidad de mercado que representa para la UCI y el país, es de intereses del polo de RV de la UCI diseñar e implantar una línea para el desarrollo de RIA que incluyan escenarios tridimensionales interactivos. Lo cual exige la asimilación de tecnologías para llevar a cabo el proceso de desarrollo con resultados rentables y sostenibles. Para alcanzar las metas esperadas durante la creación de aplicaciones informáticas, es necesario instrumentar buenas prácticas asociadas al aseguramiento y control de la calidad desde los mismos inicios de su desarrollo, se debe determinar toda la estructura que agrupa sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución.

Teniendo en cuenta la situación definida se plantea el **problema científico** siguiente: ¿Cómo desarrollar Aplicaciones Enriquecidas de Internet que incluyan escenarios tridimensionales interactivos con altos niveles de productividad y reutilización?

De ahí se define como **objeto de estudio** las Aplicaciones Enriquecidas de Internet que incluyan escenarios tridimensionales interactivos.

Se plantea como **objetivo general** del trabajo de diploma: Desarrollar una propuesta de arquitectura de software de dominio específico y un proceso de desarrollo de software para la producción de Aplicaciones Enriquecidas de Internet que incluyan escenarios tridimensionales interactivos.

En esta investigación, teniendo en cuenta la relación entre el problema, el objeto de estudio y el objetivo de la investigación, se define como **campo de acción** las arquitecturas y procesos de desarrollo de software.

Para dar cumplimiento al objetivo de la tesis se plantearon las siguientes tareas de investigación:

- Caracterización de las Aplicaciones Enriquecidas de Internet.
- Caracterización de la Arquitectura de Software de Dominio Específico.
- Análisis de la aplicación de la Ingeniería de Software para el dominio de estudio.
- Caracterización de las diferentes tecnologías empleadas a nivel mundial para el desarrollo Aplicaciones Enriquecidas de Internet que incluyan escenarios tridimensionales interactivos.
- Identificación de las tecnologías y métodos usados para dar respuesta al problema planteado.
- Definición de los pasos para implantar la Arquitectura de Software de Dominio Específico.
- Desarrollo de un caso de estudio para demostrar la factibilidad de la propuesta.

El trabajo fue dividido en tres capítulos, a continuación se presentará el nombre de cada capítulo y su objetivo en un contexto global.

Capítulo 1: Fundamento Teórico, se brinda una visión general de los aspectos relacionados con las Aplicaciones Enriquecidas de Internet y los conceptos necesarios para su estudio, Arquitectura de Software de Dominio Específico (DSSA, por sus siglas en ingles), Ingeniería de Software, Modelo de Proceso de Desarrollo, Procesos, Metodología y Herramientas. Se hace un análisis del estado actual de las tecnologías a utilizar, así como los métodos y herramientas.

Capítulo 2: Propuesta de DSSA y Proceso de Desarrollo de Software, se describe la Universidad de las Ciencias Informáticas, los métodos, procedimientos y técnicas utilizadas para llevar a cabo la investigación. Se presenta la DSSA y el Proceso de Desarrollo de Software se identifican los elementos fundamentales de la propuesta, hasta llegar a las partes que la componen. Se establece el Dominio de Aplicación, los Requisitos de Referencia, la Arquitectura de Referencia y se describen los flujos de trabajo de ingeniería.

Capítulo 3: Implantación y caso de estudio, se describe el proceso de implantación y un caso de estudio basado en la propuesta. La implantación describe las 10 prácticas propuestas por CMMI para la institucionalización de las áreas de procesos y el caso de estudio ejecuta los flujos de trabajo de Requerimiento, Arquitectura y Diseño, Implantación y Prueba.

CAPÍTULO 1: FUNDAMENTO TEÓRICO

1.1 Introducción

En el presente capítulo se brinda una visión general de los aspectos relacionados con los las Aplicaciones Enriquecidas de Internet y los conceptos necesarios para el estudio. Arquitectura de Software de Dominio Específico (DSSA), Ingeniería de Software, Modelo de Proceso de Desarrollo, Procesos, Metodología y Herramientas son los principales conceptos asociados al dominio del problema que son necesarios para la investigación y la propuesta de solución. Se hace un análisis del estado actual de las tecnologías a utilizar, así como los métodos y herramientas que pudieran ser adecuadas.

1.2 Aplicaciones Enriquecidas de Internet

El siglo XXI se enfrenta a la creciente implantación de la Sociedad de la Información debido a la presencia constante de los medios digitales y el uso masivo de las Tecnologías de la Información y las Comunicaciones (TIC) en todos los ámbitos. Esto implica el surgimiento de nuevos modelos de comunicación como: el gobierno electrónico (*e-government*), el comercio electrónico (*e-business*), entre otros. Su creciente demanda ha obligado a los desarrolladores a enfrentar el problema de cómo lograr en aplicaciones web incorporar los niveles de usabilidad y funcionalidad, elementos de multimedia y de interactividad fácilmente alcanzado en aplicaciones escritorios.

En un estudio realizado en el 2005 se llevó a cabo un profundo análisis del panorama completo de la tecnología informática (TI). En el informe de esta investigación constan dos elementos importantes: (ACCENTURE, 2005)

1. Las principales tendencias que forjarán la tecnología de la información durante la próxima década son:
    a. Integración: de una integración reducida a una integración libre.
    b. Infraestructura: de la escasez a la abundancia.
    c. Información: del control al caos.
    d. Virtualización: del hardware, software y procesos de negocios.

2. La virtualización será cada vez mayor en cuanto a Software proporcionado como servicio, una tendencia que será particularmente atractiva con el surgimiento de tecnologías enriquecidas de aplicaciones por Internet.

Sobre este mismo tema se puede leer en las distintas fuentes bibliográficas acerca de la tendencia de las aplicaciones web a volverse cada vez más complejas y multimedia. Con las tecnologías actuales, se hace crecientemente complicado lograr que las mismas funcionen con eficacia e inmediatez. Debido a esto, han surgido diversas arquitecturas de programación, conocidas como Aplicaciones Enriquecidas de Internet, que posibilitan la ejecución de variadas funciones en el navegador sin necesidad de sobrecargar ni el ancho de banda ni los servidores.(DELGADO, 2007)

La compañía desarrolladora de software *Macromedia* acuñó el término aplicación rica de Internet (RIA) para describir el futuro de las aplicaciones, según su definición, una RIA es participativa, interactiva, ligera, y flexible. Las RIA ofrecen la flexibilidad y facilidad de uso inteligente de una aplicación de escritorio y añade el amplio alcance de las aplicaciones web tradicionales. (FAIN, 2007).

Se afirma que las RIA son aplicaciones web que tienen características y funcionalidades de una aplicación de escritorio común, con la gran diferencia de que las RIA no necesitan instalar la aplicación en la máquina local del usuario pues son accesibles desde un navegador web (Firefox, IE, Opera, etc.), este requisito se conoce como portabilidad y le da la posibilidad al usuario de acceder a las aplicaciones desde cualquier plataforma y siempre funcionaran y se verán igual. (CHAVEZ, 2006).

Josep Estudis Figueras plantea: “Las Aplicaciones de Internet Enriquecidas o *Rich Internet Applications* (RIA), consisten en el aprovechamiento de la experiencia del usuario en herramientas y funciones de escritorio tan naturales como copiar, cortar y pegar, redimensionar columnas, y ordenar etc., con el alcance y la flexibilidad de presentación y despliegue que ofrecen las aplicaciones o páginas web junto con lo mejor de la multimedia (voz, vídeo, etc.).”(ESTUDIS, 2009)

Se puede decir que este tipo de aplicación posee más ventajas que las tradicionales aplicaciones web. Esta surge como una combinación de las ventajas que ofrecen las aplicaciones web y las aplicaciones tradicionales de escritorio. La diferencia radica en que en

las aplicaciones web tradicionales hay una recarga continua de páginas cada vez que el usuario pulsa sobre un enlace. De esta forma se produce un tráfico muy alto entre el cliente y el servidor, llegando muchas veces, a recargar la misma página. En las RIA solo es necesario cargar la interfaz una vez, haciendo pedidos específicos de información al servidor sin necesidad de recargar los elementos que intervienen en el proceso, solo mostrar la respuesta del servidor, reduciendo así el tiempo de espera del usuario.

1.3 Ingeniería de Software

La industria del software cada día cobra mayor auge, son miles las entidades incursionando en esta área. Revisando lo sucedido durante los últimos años se puede apreciar resultados asombrosos, de simples programas para cálculos matemáticos se ha transitado a complejos sistemas, con altas velocidades de procesamiento. Sin embargo, al analizar los datos estadísticos que se muestran en la Tabla 1-1 acerca de los fallos de proyectos, tomado de los *CHAOS REPORT* de *Standish Group* correspondientes a los años 1994 y 2002, (GROUP, 2006), se hace constar que a pesar de conocer el incremento en número de los proyectos de desarrollo, no pasa igual con la ejecución eficiente y eficaz de los mismos, cuyas mejoras se mantienen en niveles inferiores a los niveles de calidad que desarrolladores y clientes realmente necesitan.

Descripción	1994	2002
Proyectos completados en tiempo y según los costos previstos	16%	34%
Proyectos cancelados	31%	15%
Proyectos con grandes problemas de desarrollo	53%	51%
Promedio de costos sobrepasados	189%	43%
Planificación sobrepasada	222%	82%
Entregas vs. características solicitadas	61%	52%

Tabla 1-1 Resumen del CHAOS REPORT de Standish Group en los años 1994 y 2002

El origen de la ingeniería de software se debe precisamente a estos problemas, el desarrollo de software viene padeciendo de retrasos considerables en la planificación, poca productividad, elevadas cargas de mantenimiento, demandas cada vez más desfasadas con las ofertas, baja

calidad y fiabilidad del producto, y dependencia de los realizadores. Esto es lo que se ha denominado comúnmente crisis del software o aflicción crónica (PRESSMAN, 2002). Ello ha venido originado por una falta de formalismo, metodologías, herramientas de soporte y administración eficaz de los proyectos de desarrollo de software.

Según Pressman: “Sin tener en cuenta como lo llamemos, el conjunto de problemas encontrados en el desarrollo del software de computadoras no se limitan al software que no funciona correctamente. Es más, el mal abarca los problemas asociados a cómo desarrollar software, como mantener el volumen cada vez mayor de software existente y como poder esperar mantenernos al corriente de la demanda creciente del software”. (PRESSMAN, 2002)

Actualmente está surgiendo una gran expectativa ante la evolución de la ingeniería del software, al ir apareciendo nuevos métodos y herramientas formales que van a permitir en el futuro un planteamiento de ingeniería en el proceso de elaboración de software. Dicho planteamiento vendrá a paliar la demanda creciente por parte de los usuarios, permitiendo dar respuesta a los problemas.

Varias son las definiciones dadas sobre ingeniería de software entre ellas la de Bauer en 1972: “Ingeniería del Software trata del establecimiento de los principios y métodos de la ingeniería a fin de obtener software de modo rentable que sea fiable y trabaje en máquinas reales”. (BAUER, 1972).

Según Bauer, se busca aplicar producción de software en gran escala, estandarización de tareas, estandarización del control, división del trabajo, mecanización y automatización para desarrollar software de manera rentable y que sea funcional.

En 1976 enuncia Bohen: “Ingeniería de Software es la aplicación práctica del conocimiento científico en el diseño y construcción de programas de computadora y la documentación asociada requerida para desarrollar, operar (funcionar) y mantenerlos. Se conoce también como desarrollo de software o producción de software”. (BOHEN, 1976).

Bohen pone nuevos elementos cuando menciona aplicar el conocimiento científico, concibe la documentación como elemento importante en el desarrollo y por ultimo no solo se enmarca en el desarrollo del producto hasta su construcción sino las actividades post producción de implantación y mantenimiento.

Zelkovitz en 1979 puntualiza: ”Ingeniería de software es el estudio de los principios y metodologías para el desarrollo y mantenimiento de sistema de software”. (ZELKOVITZ, 1979).

En este caso Zelkovitz coincide con Bauer en que la ingeniería no se enmarca en la producción de los productos sino que llega hasta las etapas de mantenimiento, pero su mayor aporte de es mencionar el papel de las metodologías para guiar el desarrollo del software.

En 1993 la IEEE establece como estándar el concepto: “La aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación (funcionamiento) y mantenimiento del software; es decir, la aplicación de ingeniería al software.” (IEEE, 1993).

La IEEE precisa nuevamente que no solo es el desarrollo sino también las etapas de operación y mantenimiento, pero menciona características importantes el enfoque del área de producción donde el trabajo sea sistemático, disciplinado y cuantificable.

Pressman en el 2002, expresa: “La Ingeniería de Software es una tecnología multicapa”. En la que, según él: “El proceso de la ingeniería del software es la unión que mantiene juntas las capas de tecnología y que permite un desarrollo racional y oportuno de la ingeniería del software”. “Los métodos de la ingeniería de software indican cómo construir técnicamente el software”. Y “Las herramientas de la ingeniería del software proporcionan un enfoque automático o semiautomático para el proceso y para los métodos”. (PRESSMAN, 2002).

Pressman es el primero en mencionar la interacción de los elementos: proceso, métodos y herramientas, así como el papel de cada uno de ellos, que son los que transforman el desarrollo de software de una actividad artesanal a una actividad industrial. Donde el enfoque es a aplicar los principios de la era industrial para alcanzar un desarrollo más moderno, estandarizado, repetible y mejorable continuamente, elevando la calidad y la productividad del equipo de desarrollo.

Estos autores concuerdan que la ingeniería de software está llamada a dejar atrás el desarrollo artesanal de sistemas por diferentes vías: el uso de los principios y métodos de la ingeniería, la aplicación práctica del conocimiento científico, el estudio de los principios y metodologías, la aplicación de un enfoque sistemático, disciplinado y cuantificable. Pero solo Pressman menciona el cómo lograrlo y es la fortaleza de definir un proceso acorde a las características del producto, estandarizado, repetible y mejorable continuamente, que para ello se base en los métodos y las herramientas. Siendo esta la definición que se adapta al desarrollo del presente trabajo.

1.4 Arquitectura de Software

A medida que los productos fueron ganando en complejidad, dada la necesidad de resolver problemas más complicados y abarcadores se identificaron propiedades comunes acompañadas de soluciones muy similares en su estructura.

En el año 1968, Edsger Dijkstra de la Universidad Tecnológica en Holanda, propuso que "se hiciera una estructuración correcta de los sistemas de software antes de lanzarse a programar". (GARLAN, 1994). Esto conforma básicamente el concepto de lo que hoy se conoce como Arquitectura de Software. Más tarde en la conferencia de la OTAN en 1969, P.I.Sharp, "formula un conjunto de apreciaciones sorprendentes, comentando las ideas de Dijkstra, estableciendo la diferencia entre ingeniería y arquitectura de software". (GARLAN, 1994).

Hasta la década de los 90, el término fue visto de distintas maneras, sobretodo muy ligado al diseño, se hablaba de un nivel de abstracción, sin embargo aún no estaba en su lugar los elementos para el desarrollo y uso de este conocimiento. En la bibliografía consultada existen muchas definiciones sobre arquitectura de software.

Se reconoce a "*Studying Software Architecture Through Design Spaces and Rules*", de Thomas G. Lane, como el primer libro publicado por el Instituto de Ingeniería de Software (*Software Engineering Institute*, SEI) de la Universidad Carnegie Mellon, sobre el tema de la Arquitectura de Software, en el año 1990. En este Lane establece una definición "Arquitectura de software es el estudio de la estructura a gran escala y el rendimiento de los sistemas de software. Aspectos importantes de la arquitectura de un sistema incluye la división de funciones entre los módulos del sistema, los medios de comunicación entre módulos, y la representación de la información compartida."(LANE, 1990).

Lane menciona como la concepción de componentes ("división de funciones entre los módulos del sistema"), y de conectores ("medios de comunicación entre módulos"), van tomando su espacio dentro de la definición. En este libro, Lane brinda un concepto interesante, "espacios de diseño", los cuales proporcionan un marco para la confección de normas que pueden ayudar a un diseñador en la selección de un arquitectura apropiada para las necesidades funcionales de un nuevo sistema.

Garlan y Shaw en 1994 la conceptualizaron como "una colección de componentes y conectores unidos con una descripción de la interacción entre esos componentes y conectores". (GARLAN, 1994).

Este planteamiento se debe al aumento del tamaño y complejidad de los productos de software, el problema principal no radica ya, en los algoritmos y las estructuras de datos, sino que se dirige a la organización de los componentes que conforman el sistema. En esa ponencia se introduce por primera vez el término estilos arquitectónicos y se definen una gran gama de ellos.

Clements en 1996 la definen como “la estructura de las estructuras de un sistema, la cual comprende componentes de software, las propiedades visibles externas de estos componentes, y las relaciones entre ellos.”(CLEMENTS, 1996)

En esta nueva propuesta se enumeran cinco temas fundamentales: diseño o selección de la arquitectura, representación de la arquitectura, evaluación y análisis. En el 2000 se publica la versión definitiva de la recomendación IEEE Std 1471, que procura homogeneizar y ordenar la nomenclatura de descripción arquitectónica y homologa los estilos como un modelo fundamental de representación conceptual. Está se considera la definición más utilizada en las bibliografías, “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, 2000).

Independientemente a la variedad de definiciones literarias se convergen en que la arquitectura en el desarrollo de cualquier tipo de software representa las bases sobre las que se desarrollará el sistema. Brinda una estructura que soporta las soluciones a cada tipo de problema que pueda aparecer en el desarrollo. Define la organización e interacción entre los distintos componentes del mismo. Asegura que los requerimientos más importantes puedan ser evaluados e implementados. Una arquitectura de software también permite flexibilidad en el sistema pues facilita la ejecución de futuros cambios. Promueve la reutilización de componentes existentes como librerías de clases, sistemas legados y de aplicaciones de terceros.

La arquitectura de software provee una descripción de alto nivel de los subsistemas que comprenden la arquitectura del sistema. Está atada a comprender cómo las mayores operaciones del negocio son soportadas. Ayuda a la planificación de recursos y la asignación de tareas, debido a que el trabajo de desarrollo puede ser particionado a través de los subsistemas y los esfuerzos de desarrollo individual pueden proceder en paralelo. Cuando los equipos de desarrollo están geográficamente dispersados puede minimizar el número de

interacciones requeridas.

1.4.1 Arquitectura de Software de Dominio Específico

En la década de los años 80, DARPA afrontó un fuerte problema desarrollando un gran sistema de software para la defensa, con un acercamiento llamado Arquitectura de Software de Dominio Específico (DSSA). La motivación y fundamento para este programa fue basado en las siguientes suposiciones que Kaiser comenta:

“El desarrollo sobre dominios específicos puede ser optimizado. La reutilización de módulos es más probable si estos se obtienen de otras aplicaciones dentro del mismo dominio. “(KAISER 2005)

El objetivo principal fue reutilizar tantos objetos como fuera posible; además de minimizar la intención de resolver diferentes tipos de problemas antes de reinventar una solución para cada problema nuevo.

En 1992 Mettala y Graham plantean: “se observa una tendencia marcada hacia la búsqueda de un modelo de estructuración de software, y se diseñan un conjunto de modelos de dominios, basados en diseños genéricos”. (METTALA, 1992).

Según Kaiser “un dominio es un área de interés, usualmente representando un espacio del problema que es un subconjunto de una o más disciplinas.”(KAISER 2005). En este sentido el dominio se delimita en el alcance del negocio de aplicación o en el ámbito técnico. Por ejemplo, aplicaciones web desarrolladas sobre JEE, específicamente las que utilicen el contenedor de inversión de control de *Spring Framework* e *Hibernate* para la persistencia o las RIA que permiten interactuar con 3D, desarrolladas con el lenguaje de programación *ActionScript* 3.0 y cuyo ambiente de desarrollo sea soportado por software libre. Cuando se construye una aplicación, se está desarrollando software para un espacio del problema, es decir, un conjunto de problemas a ser resueltos dentro del dominio. En fin, un dominio es definido por un conjunto de problemas o funciones comunes que las aplicaciones en ese dominio pueden resolver o hacer.

En la propuesta de DARPA definió 4 pasos claves para las DSSA: (KAISER 2005)

- Un modelo del dominio es desarrollado para establecer una terminología estándar y una semántica común para el dominio del problema.
- Un conjunto de requerimientos de referencia para todas las aplicaciones dentro del

dominio del problema que es desarrollado.

- Una arquitectura de referencia es definida para una solución genérica, estandarizada y basada en componentes del sistema para anticipar los requerimientos del cliente.
- Nuevas aplicaciones son desarrolladas para determinar requerimientos específicos de la aplicación, seleccionar o generar componentes particulares y ensamblarlos acorde a la arquitectura de referencia.

Esta propuesta se puede ver en la figura 1-1:

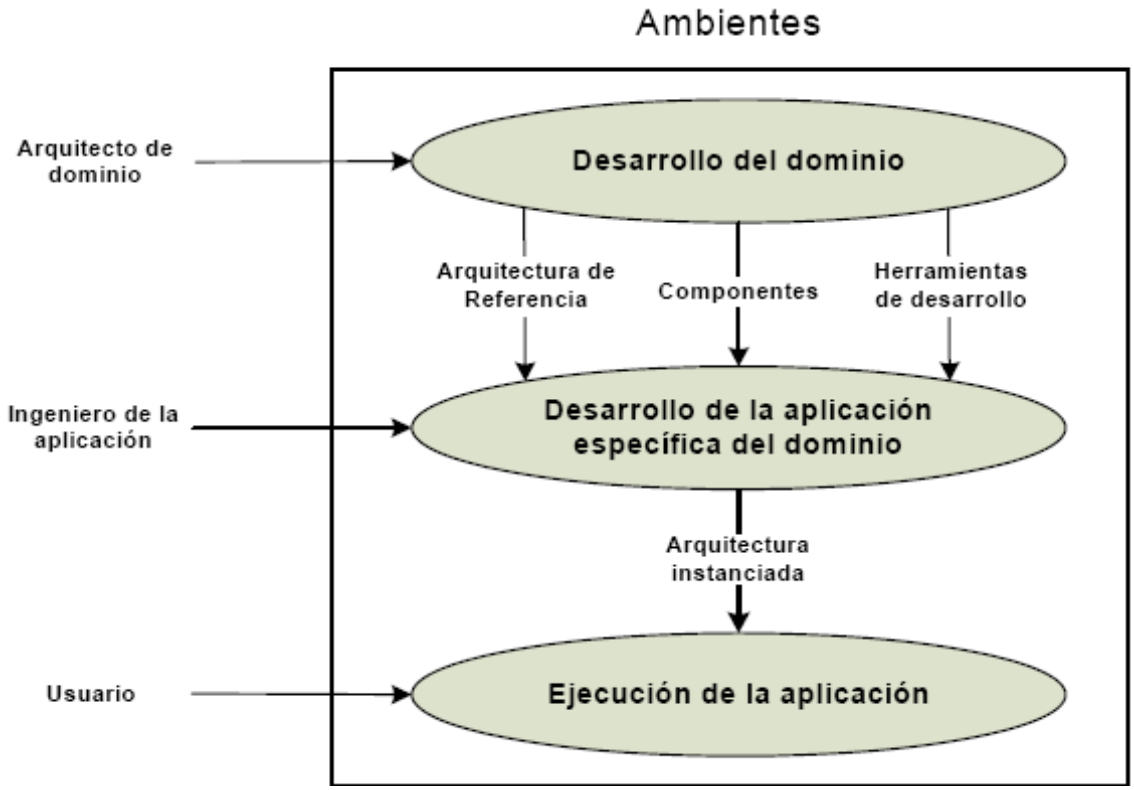


Figura 1-1: Modelo de Arquitectura de Software de Dominio Específico de DARPA.

La arquitectura de referencia puede incluir componentes concretos que deberían ser embebidos en todas las aplicaciones que derivan de ella. Además es adaptada para resolver los requerimientos particulares de los clientes. La actividad principal es ajustar a la medida la arquitectura de referencia usando los requerimientos de la aplicación para producir la arquitectura específica de la aplicación.

1.5 Modelo de Procesos de Software

En la bibliografía consultada no existe una receta de cómo aplicar los elementos de la Ingeniería de Software de manera tal que se tribute al éxito de los proyectos, pero la mayoría de los autores referencian a Pressman en el siguiente planteamiento: “Un modelo de proceso del software es una estrategia de desarrollo que acompaña al proceso, métodos, capas de herramientas y las fases genéricas. Se selecciona un modelo de proceso para la ingeniería del software según la naturaleza del proyecto y de la aplicación, los métodos y las herramientas a utilizarse y los controles y entregas que se requieren”. (PRESSMAN, 2002).

En el análisis de este planteamiento se evidencia el papel del modelo de proceso de software como rector para definir los elementos de la Ingeniería de Software. Está definición no busca imponer un estándar a partir de algún método particular, ni definir un equipo completo de métodos, procesos y herramientas aceptables. Todos los elementos deben ser adaptados al contexto del proyecto (recursos técnicos y humanos, tiempo de desarrollo, tipo de sistema, etc.) Se considera que a partir de las propuestas cada proyecto o línea de desarrollo debe examinar sus necesidades, elegir, definir y justificar la selección dentro de la definición de los requerimientos del software. En este epígrafe se hace referencia a los modelos de proceso del software más importante encontrado durante la investigación.

1.5.1 Modelo Lineal Secuencial

Al decir de Royce “es el enfoque metodológico que ordena rigurosamente las etapas del ciclo de vida del software, de forma tal que el inicio de cada etapa debe esperar a la finalización de la inmediatamente anterior. Comienza en un nivel de sistemas y progresa con el análisis, diseño, codificación, pruebas y mantenimiento”. (ROYCE, 1970). El modelo lineal secuencial comprende las fases que se muestran en el Anexo 1.

Sobre este modelo Hanna menciona “El modelo lineal secuencial es el paradigma más antiguo y más extensamente utilizado en la ingeniería del software. Sin embargo, la crítica del paradigma ha puesto en duda su eficacia (HANNA, 1995).

Los proyectos reales raras veces siguen el modelo secuencial puro que se propone; debido a que el modelo lineal puede acoplar interacción con otros indirectamente. Las debilidades más abordadas de este modelo es debido a que su aplicación en proyectos con requisitos

cambiantes puede causar confusión en el equipo de desarrollo sobre la fase en la que se encuentran, y por ende una versión del producto no estará disponible hasta que no haya avanzado el proyecto y puede que el software no cumpla con los requisitos del usuario por el largo tiempo de entrega del producto. Este modelo es de suma importancia, debe usarse solamente si se entienden a plenitud los requisitos y se adecua al desarrollo de productos sencillos y de poca complejidad, se desarrolla en un ciclo convencional de ingeniería donde una fase no empieza hasta que no termina la fase anterior.

1.5.2 Modelo de Construcción de Prototipos

Sobre este modelo Brooks plantea “este modelo comienza con la recolección de requisitos. El desarrollador y el cliente encuentran y definen los objetivos globales para el software, identifican los requisitos conocidos y las áreas del esquema en donde es obligatoria más definición. Entonces aparece un diseño rápido que lleva a la construcción de un prototipo. El prototipo es evaluado por el cliente/usuario y se utiliza para refinar los requisitos del software a desarrollar. La iteración ocurre cuando el prototipo se pone a punto para satisfacer las necesidades del cliente, permitiendo al mismo tiempo que el desarrollador comprenda mejor lo que se necesita hacer”. (BROOKS, 1975).

Los clientes por lo general definen un conjunto de procesos para informatización, pero les es muy difícil identificar los requisitos explícitos. En otros casos, los desarrolladores no tienen una clara comprensión de los requisitos. En estas situaciones, un paradigma de construcción de prototipos que permita la validación y verificación de los requisitos desde etapas tempranas puede ofrecer un mejor enfoque y mitigar los riesgos asociados a la gestión y desarrollo de los mismos. Mitiga los efectos de los requisitos cambiantes. Este enfoque entorpece el desarrollo si la herramienta de modelado del prototipos es incompatible con el de desarrollo de la aplicación porque si no se duplicarían los esfuerzos para el desarrollo de la capa interfaz. Este modelo se puede asociar a cualquiera de los modelos restantes para la etapa de Requerimiento. El modelo anterior se puede apreciar en el Anexo 2.

1.5.3 Modelo de Desarrollo Rápido de Aplicaciones

A partir de la propuesta del modelo lineal secuencial y la debilidad de no tener un producto hasta avanzado el proyecto, Martin propone: “El modelo DRA es una adaptación a alta

velocidad del modelo lineal secuencial, permite construir sistemas utilizables en poco tiempo, normalmente de 60 a 90 días, frecuentemente con algunas concesiones. Se utiliza una construcción basada en componentes". (MARTIN, 1991). El modelo se puede apreciar en el Anexo 3.

Como se aprecia en la figura del Anexo 3 este modelo permite el desarrollo en paralelo de varios equipos y la integración posterior del producto o el desarrollo en varios ciclos de vida evolutivos o en espiral. El modelo permite construir sistemas utilizables en poco tiempo, utiliza una construcción basada en componentes y por ende sus ventajas, los entregables pueden ser fácilmente trasladados a otra plataforma y permite visibilidad temprana, una mayor flexibilidad y la codificación manual es menor. Tiene como limitantes para proyectos grandes que requieren recursos humanos suficientes y no es adecuado cuando los riesgos técnicos son altos.

1.5.4 Modelos Evolutivos del proceso del software

En la mayoría de los proyectos grandes mayores de 40 casos de uso, los requisitos del producto cambian conforme el desarrollo procede, haciendo que el camino que lleva al producto final constantemente tenga que ser redefinido; los compromisos y las exigencias del cliente impulsan a que sea imposible finalizar un producto completo, por lo que se debe introducir un enfoque que propicie cumplir la presión competitiva. Muchas veces en estos casos se consigue comprender perfectamente el conjunto de requisitos del producto en etapas avanzadas del desarrollo por lo que se necesita un modelo de proceso que sea diseñado para obtener parte del software en la medida que se vaya comprendiendo.

Una solución a esto es haciendo uso de los modelos evolutivos, los cuales se caracterizan por ser iterativos y donde cada secuencia lineal produce un incremento del software. Esto permite a los ingenieros del software desarrollar versiones cada vez más completas del software. Los modelos evolutivos tiene tres variantes: incrementales, en espiral y desarrollo concurrente. (MCDERMID, 1993).

Según estos autores el incremental combina elementos del modelo de cascada (aplicados repetitivamente) con la filosofía interactiva de construcción de prototipos. El primer incremento es un producto esencial (núcleo), se afrontan requisitos básicos y muchas funciones extras (conocidas o no) quedan para los siguientes incrementos. El cliente usa el producto central y en

base a la utilización y/o evaluación se desarrolla un plan para el incremento siguiente. Este proceso se repite hasta que se elabora el producto completo. El modelo incremental entrega un producto operacional en cada incremento. El desarrollo incremental es un modelo útil cuando no está disponible el personal que se necesita para una implementación completa en la fecha límite que se haya establecido para el proyecto. Este modelo se puede apreciar en el Anexo 4.

El modelo evolutivo en espiral fue propuesto inicialmente por Boehm y a su juicio es un modelo que conjuga la naturaleza iterativa de construcción de prototipos con los aspectos controlados y sistemáticos del modelo lineal secuencial. En el modelo espiral, el software se desarrolla en unas series de versiones incrementales, durante las primeras iteraciones, la versión incremental puede ser un modelo en papel o un prototipo y durante las últimas iteraciones, se producen versiones cada vez más completas del sistema. El modelo se divide en un número de actividades estructurales llamadas regiones de tareas que pueden ser de tres a seis. Este modelo se puede apreciar en el Anexo 5. (BOEHM, 1998)

Para explicar el modelo concurrente Davis y Sitaram han descrito un ejemplo “un cambio de requisitos durante el último desarrollo implica que se está escribiendo requisitos, diseñando, codificando, haciendo pruebas y probando la integración todo al mismo tiempo.”(DAVIS, 1994).

Generalmente este modelo se utiliza en grandes proyectos de desarrollo, con equipos grandes y que los requisitos sean cambiantes, en el caso específico de la vertiente concurrente se utiliza a menudo como el paradigma de desarrollo de aplicaciones cliente/servidor. Al decir de Sheleg, cuando se aplica a cliente/servidor, el modelo de proceso concurrente define actividades en dos dimensiones, una dimensión de sistemas y una dimensión de componentes. Los aspectos del nivel de sistemas se afrontan mediante tres actividades: diseño, ensamblaje y uso. La dimensión de componentes se afronta con dos actividades: diseño y realización. La concurrencia se logra de dos formas: las actividades de sistemas y de componentes ocurren simultáneamente y pueden modelarse con el enfoque orientado a objeto. Una aplicación cliente/servidor típica se implementa con muchos componentes, cada uno de los cuales se pueden diseñar y realizar concurrentemente. (SHELEG, 1994).

De manera general los clientes no tienen que esperar hasta que el sistema se entregue completamente para comenzar a hacer uso de él. Se pueden aclarar los requisitos que no tengan claros conforme ven las entregas del sistema. Esto disminuye el riesgo de fracaso de

todo el proyecto, puede aplicarse a cualquier proyecto sin importar la complejidad, es apropiado para sistemas orientados objeto y se puede aplicar el enfoque de construcción de prototipos en cualquier etapa de evolución del producto.

Tiene como limitantes la definición de los requisitos para cada incremento, el cual debe ser pequeño para limitar el riesgo. Aumentar la funcionalidad es difícil para establecer las correspondencias de los requisitos contra los incrementos y detectar las unidades o servicios genéricos para todo el sistema los cuales se desarrollan en las primeras etapas, esto incide en que le proyecto tienda a ser incontrolable.

Modelo de Desarrollo basado en componentes

El uso del paradigma Orientado a Objeto ha creado el marco de trabajo idóneo para introducir el modelo de desarrollo basado en componentes. Según Nierstrasz: “El modelo de desarrollo basado en componentes conduce a la reutilización del software, lo que beneficia a los ingenieros de software. Es evolutivo por naturaleza, y exige un enfoque iterativo para la creación del software. Configura aplicaciones desde componentes preparados de software llamados clases.”. (NIERSTRASZ, 1992). La estructura de este modelo se puede apreciar en el Anexo 6.

Este modelo permite alcanzar un alto índice de productividad y la reutilización del software, reduce el tiempo de entrega, el costo y esfuerzo del proyecto, disminuye los riesgos durante el desarrollo. Pero requiere de actividades para identificar los componentes, así como su clasificación y almacenamiento en repositorios, el flujo de procesos demanda del ensamblaje de componentes a través de la actividad de ingeniería. Reúne en él las características más importantes de los modelos anteriores.

1.5.5 Consideraciones sobre los Modelos de Proceso de Software

Después de analizar los distintos modelos de producción y ver las características de cada uno se desarrolló una tabla de comparación entre los modelos basado en las ventajas y desventajas de cada uno, para ello ver el Anexo 7.

Basado en los elementos más significativos de cada uno y teniendo en cuenta las particulares de la línea de producción de ser desarrollos cortos entre uno y dos meses con equipos

pequeños de cinco a diez personas, donde los requisitos pueden ser modelados íntegramente en prototipos, los despliegues deben ser inmediatos, el estilo arquitectónico es Cliente/Servidor. Se seleccionaron dos para combinarlos y utilizarlos en el proyecto, los cuales son el Modelo de Construcción de Prototipos y el Modelo de Desarrollo de Rápido de Aplicaciones.

El segundo combina la fortaleza de un desarrollo basado en componentes, al modelo lineal secuencial, con la idea de línea de producción y montaje del proceso industrial; es posible notar también una fuerte presencia de intercambio e integración de piezas, en el caso de la línea de producción de software, esas piezas son denominadas componentes de software. Permite además el desarrollo evolutivo en cualquiera de sus vertientes incremental, espiral o concurrente incorporando por tanto sus ventajas. Para mitigar sus desventajas se propone combinarlo con el primero que se menciona así tener una mejor comprensión y compromisos con los requisitos del sistema.

La propuesta incide fuertemente en la reducción del tiempo de desarrollo de los proyectos y por ende en los costos, elevar los niveles de productividad y mitigar los riesgos técnicos, al basarse en componente y validar y verificar los requisitos a través de prototipos. Permite que el sistema en caso necesario pueda ser fácilmente desarrollado en otra plataforma, la codificación manual se disminuye y es más flexible. Si el sistema es grande se pueden desarrollar a través de varios equipos trabajando de manera paralela, o en espiral o con un equipo que trabaje en iteraciones. Es de fácil el entendimiento del modelo por los desarrolladores trabajan pues es ciclo convencional de ingeniería.

Es importante señalar que requiere de desarrolladores y clientes comprometidos con actividades rápidas, cuando el proyecto es grande se necesita de una organización de muchos equipos y demanda de mayor efectividad en la gestión de proyecto, en sistemas que no se pueda definir una arquitectura claramente o los componentes que la componen este enfoque puede que no obtenga los resultados que se esperan.

1.6 Proceso de desarrollo de software

El proceso de desarrollo de software según Jacobson "es aquel en que las necesidades del usuario son traducidas en requerimientos de software, estos requerimientos transformados en

diseño y el diseño implementado en código, el código es probado, documentado y certificado para su uso operativo". Concretamente "define quién está haciendo qué, cuándo hacerlo y cómo alcanzar un cierto objetivo". (JACOBSON, 1998)

Jacobson concibe el proceso de desarrollo de software en diferentes etapas en el que las necesidades del cliente se transforman en el desarrollo de diferentes actividades para obtener el producto final y que establece lo que debe hacer los desarrolladores, cuándo y cómo para obtener el producto final. Estos elementos generalmente se responden al configurar una metodología para el proceso. El proceso debe tener un diagrama de flujo que describa el orden lógico de actividades. Estas son asignadas a un rol, además debe tener tareas para ejecutarlas y artefactos de entrada y salidas. Los artefactos terminados deben cumplir las especificaciones de calidad que se le propone en la lista de chequeo. Mediante la cual se especifican una serie de puntos a evaluar. En todas las fases del desarrollo se deben tener en cuenta estos puntos en función de que al finalizar cada actividad haya que hacer el menor número de correcciones posibles.

Por su parte Pressman expresa: “El proceso es un dialogo en el que se reúne el conocimiento y se incluye en el software para convertirse en software. El proceso proporciona una interacción entre los usuarios y los diseñadores, entre los usuarios y las herramientas de desarrollo, y entre los diseñadores y las herramientas de desarrollo.” (PRESSMAN, 2005).

En este caso Pressman incluye que la interacción no solo es entre desarrolladores y clientes sino que a su vez estos interactúan con las herramientas que semiautomatizan o automatizan el proceso. Incluye por tanto un nuevo elemento a tener en cuenta al describir un proceso y es poner las herramientas a utilizar para desarrollar las actividades. En la figura 1-2 se representan los elementos que componen un proceso y su interrelación.

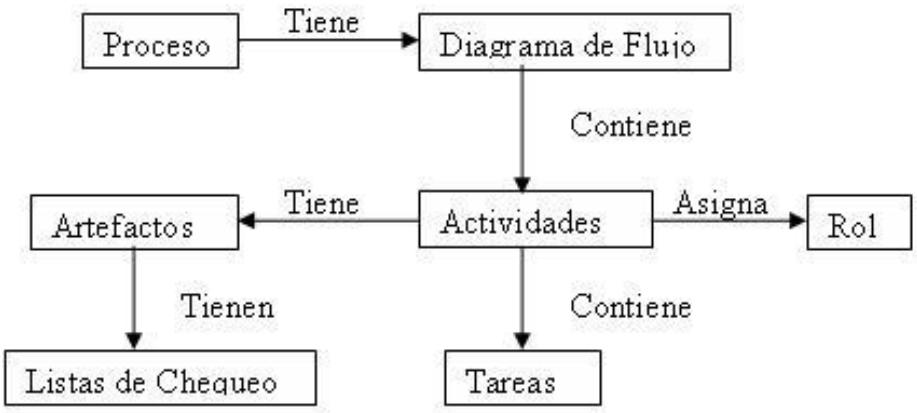


Figura 1-2: Representación de los elementos que forman el proceso.

El proceso juega un papel determinante para el éxito de un proyecto y la satisfacción del cliente, si el proceso es débil, el producto final va a sufrir indudablemente". (DAVIS, 1995). 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. Capturar y presentar las mejores prácticas que el estado actual de la tecnología permite. En consecuencia, reduce el riesgo y hace el proyecto más predecible. El efecto global es el fomento de una visión y una cultura global. (Jacobson, 2000)

El proceso de desarrollo de software no es único; no existe un proceso de software universal que sea efectivo para todos los contextos de proyectos de desarrollo. Debido a esta diversidad, es difícil encontrar en la literatura un proceso que sirva a la medida para los proyectos, por esta razón cada proyecto o línea de producción debe elegir, definir y justificar la metodología que guía su proceso a partir del modelo de producción y los requerimientos del software.

1. 7 Metodologías de desarrollo de software

Las metodologías de desarrollo de software surgieron a raíz de la necesidad de controlar y documentar proyectos cada vez más complejos, impulsadas principalmente por instituciones económicamente importantes y con requisitos de seguridad y fiabilidad en sus sistemas sumamente estrictos.

La palabra metodología proviene de método, que quiere decir poner orden, organizar, ordenar, sistematizar. Se puede decir que una metodología es un conjunto de métodos utilizados en una investigación o proceso. Según la norma 1074 de IEEE toda "metodología de desarrollo de

software debe incluir la forma en que se va a realizar la captura de requisitos, el diseño, la implementación y prueba". (IEEE, 1997)

En la situación actual el uso correcto de una metodología adquiere una importancia mayor debido a las tendencias de desarrollo en equipo y con la finalidad de eliminar de una vez el caos existente en el mundo del desarrollo de software.

Las metodologías se basan en una combinación de los modelos de proceso genéricos (cascada, evolutivo, incremental, etc.). Estas son básicamente un conjunto de procedimientos que imponen una serie de pasos sobre el desarrollo del software, permiten producir y mantener un producto garantizando su fiabilidad y calidad. No existe una metodología aplicable a todos los proyectos, debe ser seleccionada de acuerdo al contexto del mismo: recursos técnicos y humanos, tiempo de desarrollo, tipo de sistema, etc. En la actualidad las metodologías se clasifican en robustas o tradicionales y ligeras o ágiles.

Las tradicionales describen un proceso de desarrollo de software configurable que se adapta a través de los proyectos a variados tamaños y diferentes niveles de complejidad. Se basa en muchos años de experiencia en el uso de la tecnología orientada a objetos y en el desarrollo de software. Se centran en el control del proceso, establecen de manera rigurosa las actividades involucradas, los artefactos que se crean y las herramientas y notaciones que serán usadas. Dentro de estas metodologías se encuentran: *Microsoft Solution Framework* (MSF) y Proceso Unificado de Desarrollo (RUP, *Rational Unified Process*), esta última es un proceso de desarrollo de software que utiliza como notación Lenguaje Unificado de Modelado (UML).

Las metodologías ágiles se basan en iteraciones muy cortas y le dan mayor valor al individuo, dentro de estas metodologías se destacan SCRUM, *Adaptive Software Developmen* (ASD), *Crystal Methodologies*, XP (*eXtreme Programming*), etc.

Aunque los creadores e impulsores de las metodologías más populares coinciden con los principios enunciados anteriormente, cada metodología tiene características propias y hace hincapié en algunos aspectos más específicos. En el Anexo 8 se muestra una comparación entre ellas.

A continuación se resumen características de tres de ellas, estas son utilizadas con éxito en proyectos reales pero les faltaba una mayor difusión y reconocimiento.

1.7.1 Proceso Unificado de Desarrollo

El Proceso Unificado de Desarrollo, RUP por sus siglas en inglés es creado en la compañía Rational, donde confluyen 'los tres amigos' como se llaman a sí mismos o los tres grandes Grady Booch, James Rumbaugh e Ivar Jacobson en tres décadas de trabajo.

Al decir de Letelier “El objetivo de RUP es asegurar la producción de software de alta calidad, que satisfaga los requerimientos de los usuarios finales (respetando el plan y el presupuesto). Se caracteriza por estar guiado por los casos de uso, estar centrado en la arquitectura y ser iterativo e incremental. RUP brinda la posibilidad de elegir el conjunto de componentes de procesos que concuerdan con las necesidades del proyecto. Al ser partidario de la unificación del equipo de trabajo asigna responsabilidades, artefactos y tareas de manera que cada miembro individualmente perciba lo que tiene que hacer y como lo tiene que hacer”. (LETELIER, 2002).

Por otra parte el Grupo Soluciones Innova publica en su sitio: “Si se persigue la línea que propone RUP es viable, entrega un producto a tiempo y con confianza. RUP es una plataforma manejable de procesos de desarrollo que brinda muchísimos beneficios. Solo RUP provee un Framework de proceso configurable mediante el cual se puede seleccionar e implementar los componente adecuados para lograr que dicho proceso sea sólido y estable. Es capaz de que el proceso sea práctico, con guías para facilitar el despegue de la planificación del proyecto, además de una detallada documentación para cuando se haga la entrega del producto el cliente tenga la posibilidad de consultarlas y entender el software”. (GSI, 2007).

En esencia esta metodología guía a los equipos de proyecto en cómo administrar el desarrollo iterativo de un modo controlado mientras se balancean los requerimientos del negocio, el tiempo al mercado y los riesgos del proyecto. El proceso describe los diversos pasos involucrados en la captura de los requerimientos y en el establecimiento de una guía arquitectónica lo más pronto, para diseñar y probar el sistema hecho de acuerdo a los requerimientos y a la arquitectura. El proceso describe qué entregables producir, cómo desarrollarlos y también provee patrones. El proceso unificado es soportado por herramientas que automatizan entre otras cosas, el modelado visual, la administración de cambios y las pruebas. Se caracteriza básicamente por ser vital la captura de requisitos, iteración actual condicionada por la anterior, se necesita de un buen líder de proyecto para garantizar el trabajo del equipo de desarrollo, se realiza un gran número de artefactos lo que puede provocar

retrasos por mala preparación de los analistas, las responsabilidades están divididas y es aplicable a todo tipo de proyecto asumiendo sus extensiones.

1.7.2 Programación Extrema, XP

Programación Extrema, XP por sus siglas en inglés es la metodología ágil más difundida. Sobre ella Manuel Calero Solin expresa: “Germina como nueva disciplina para desarrollar software cerca de unos seis años atrás, pero ha causado una gran conmoción para los desarrolladores a nivel mundial. Su objetivo es muy simple, solo satisfacer al cliente e incrementar colosalmente el trabajo en equipo, donde los líderes del proyecto, clientes y programadores en si forman parte del grupo y están involucrados en todo el desarrollo del software”. (CALERO SOLIS, 2003).

Así mismo Beck explica: “Está centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. Se basa en realimentación 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 especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico”. (BECK, 2000).

Las principales características de esta metodología es que se centra en resolver el problema lo más rápido posible. Cada miembro del equipo debe estar listo para enfrentar cualquier problema y la instrumentación de los comentarios “El cliente se introduce en el equipo de desarrollo”, “Hago algo y lo pruebo”, “Termino todo y después integro”.

Los obstáculos más comunes surgidos en proyectos XP al decir de Larman: “son la “fantasía” de pretender que el cliente se quede en el sitio y la resistencia de muchos programadores a trabajar en pares”. (LARMAN, 2004). Craig Larman señala como factores negativos la ausencia de énfasis en la arquitectura durante las primeras iteraciones (no hay arquitectos en XP) y la consiguiente falta de métodos de diseño arquitectónico. Un estudio de casos de Bernhard Rumpe y Astrid Schröder sobre 45 proyectos reveló que las prácticas menos satisfactorias de XP han sido “la presencia del usuario en el lugar de ejecución y las metáforas, que el 40% de los encuestados no comprende para qué sirven o cómo usarlas”. (BERNHARD, 2001).

1.7.3 SCRUM

SCRUM ha sido desarrollada por Ken Schwaber, Jeff Sutherland y Mike Beedle. Define un marco para la gestión de proyectos, que se ha utilizado con éxito durante los últimos 10 años.

Martin en su libro sobre la esencia de la metodología expresa: "Está especialmente indicada para proyectos con un rápido cambio de requisitos. Sus principales características se pueden resumir en dos: El desarrollo de software se realiza mediante iteraciones, denominadas *sprints*, con una duración de 30 días. El resultado de cada *sprint* es un incremento ejecutable que se muestra al cliente. La segunda característica importante son las reuniones a lo largo proyecto, entre ellas destaca la reunión diaria de 15 minutos del equipo de desarrollo para coordinación e integración". (SCHWABER, 1999).

Por su parte Juan Palacio puntualiza: "Es una metodología de desarrollo muy simple, que requiere trabajo duro porque no se basa en el seguimiento de un plan, sino en la adaptación continua a las circunstancias de la evolución del proyecto.

1. Es un modo de desarrollo de carácter adaptable más que predictivo.
2. Orientado a las personas más que a los procesos.
3. Emplea la estructura de desarrollo ágil: incremental basada en iteraciones y revisiones". (PALACIO, 2006).

La metodología propicia el desarrollo iterativo e incremental, y por ende asume las ventajas de este modelo de proceso de desarrollo. Conduce a que el cliente no tiene que esperar hasta que el sistema se entregue completo para usarlo, se pueden aclarar los requisitos, disminuye el riesgo de fracaso de todo el proyecto, puede aplicarse a cualquier proyecto sin importar la complejidad. Tiene como limitante la definición de los requisitos para cada Sprint. Al igual que XP no centra su desarrollo en la arquitectura.

1.7.4 Consideraciones sobre las metodologías

Las metodologías seleccionadas no son fáciles de comparar entre sí conforme a un pequeño conjunto de criterios. XP y RUP han definido claramente sus procesos, mientras SCRUM es bastante más difuso en ese sentido, limitándose a un conjunto de principios y valores. Lo que tienen en común es su modelo de desarrollo incremental (pequeñas entregas con ciclos rápidos), cooperativo (desarrolladores y usuarios trabajan juntos en estrecha comunicación), directo (el método es simple y fácil de aprender) y adaptativo (capaz de incorporar los

cambios). Pero solo RUP establece un desarrollo centrado en la arquitectura y guiado por casos de uso lo que se alinea con la propuesta de la línea de producción que se propone. Estructuralmente se asemejan al modelo RAD (Desarrollo Rápido de Aplicaciones) por el desarrollo en iteraciones y basada en componentes, con un ciclo de ingeniería claro en sus 6 disciplinas de ingeniería y 3 de soporte, propone una etapa de diseño independiente de la plataforma en el análisis lo que coincide con que puede ser desarrollado en varias plataformas. Su guiado en caso de uso propone el desarrollo de prototipos evolutivos lo que alinea con el modelo de construcción de prototipos.

Debido a que el software que se pretende desarrollar es de ciclos cortos con equipos pequeños la metodología se debe aligerar su configuración como ella misma proponen en la disciplina de Ambiente.

A esto se le suma la comparación de los elementos necesarios para guiar el proceso (roles, artefactos, actividades y flujo de trabajo) que se presentan en el Anexo 9. Donde se puede evidenciar que de los tres es el más descrito y completo para alinearse con los Lineamientos de Calidad y el Expediente de Proyecto establecido de manera obligatoria por la Dirección de Calidad en la UCI.

1.8 Bases tecnológicas

Contar con aplicaciones desarrolladas en computadoras posibilita un acceso rápido y fácil a la información, una buena gestión de la información conduce a tomar las decisiones más acertadas en cada momento. Frases como esta son comúnmente usadas para motivar a las entidades a la informatización masiva y ordenada de sus procesos, en la búsqueda de elevar sus niveles de planificación, gestión y producción.

En la actualidad todo desarrollo a gran escala es asistido por herramientas que se encargan de analizar los procesos, organizar tareas de trabajo, controlar y supervisar el progreso y gestionar la calidad técnica. Los procesos de desarrollo de software modernos también son soportados por herramientas. “Hoy, es impensable desarrollar software sin utilizar un proceso soportado por herramientas. El proceso y las herramientas vienen en el mismo paquete: las herramientas son esenciales en el proceso”. (JACOBSON, 2000).

Otro aspecto a tener en cuenta es que las herramientas de la Ingeniería del software proporcionan un enfoque automático o semiautomático para el proceso y para los métodos. (PRESSMAN, 2002).

Los autores concuerdan en el papel de las herramientas como parte fundamental de la ingeniería de software y tributa a la efectividad de los procesos, elevan los niveles de productividad y disminuyen los riesgos técnicos.

En el entorno de la UCI coexisten un grupo de Direcciones de Servicios subordinadas a la Dirección General de la Infraestructura Productiva encargadas de establecer los lineamientos, disposiciones y procesos en diferentes áreas del conocimiento que se manifiestan en la producción. Ellas han definido un conjunto de herramientas, por eso en el epígrafe mencionaremos las que han sido establecidas y realizaremos el análisis y selección de aquellas que son necesarias para el desarrollo de la línea de producción que se propone pero que no han sido identificadas.

La Dirección de Calidad de la Universidad coordina un proyecto de mejora con el fin de fortalecer la producción, reorganizando e industrializando sus procesos bajo los paradigmas de calidad, con el objetivo de alcanzar un proceso de desarrollo de software administrado, está trabajando por institucionalizar las buenas prácticas correspondiente al nivel 2 del Modelo de Capacidad de Madurez Integrada de CMMI. Como parte de ello han establecido las siguientes herramientas:

- Herramienta para la trazabilidad: *Open Source Requirements Management Tool* (OSRMT)
- Herramientas para el modelado: *Visual Paradigm o StarUML*.
- Herramienta para la gestión de proyectos: *RedMine, Project.Net*.

El Laboratorio Industrial de Pruebas de Software tiene la responsabilidad de desarrollar y coordinar las pruebas de liberación de los productos que se desarrollan en las diferentes áreas de la Universidad. En el mismo se han estado aplicando diferentes herramientas entre ellas el *J-Meter* y *Selenium*.

La Dirección Técnica de la Infraestructura Productiva está desarrollando el entorno para realizar las salvas automáticas de los proyectos usando Bacula.

1.8.1 Ambiente Integrado de Desarrollo

Un Ambiente Integrado de Desarrollo conocido por IDE por sus siglas en inglés, *Integrated Development Enviroment* es un programa compuesto por un conjunto de herramientas para el programador pueden usarse uno o varios lenguajes de programación. En él se agrupan varios

programas de aplicación, un editor de código, un compilador, un depurador y un constructor de interfaz gráfica. A continuación se caracterizan un conjunto de herramientas para formar en EID.

1.8.2 Herramienta de programación: Eclipse

Metafóricamente, Eclipse es como una tienda para herreros, donde no solamente se hacen productos, sino que además se hacen las herramientas para hacer los productos. Cuando se descarga el Eclipse SDK, se obtiene un equipo de instrumentos para desarrollo en Java o *Java Development Toolkit* (JDT) para escribir y depurar programas en Java; además se obtiene un ambiente de desarrollo de *plugin* (*Plug-in Development Enviroment)* para heredar de Eclipse. Si todo lo que se quiere es un IDE para Java, no se necesita nada además que el JDT. Esto es para lo que la mayoría las personas usan Eclipse. (DAVID GALLARDO 2003)

Aunque Eclipse es escrito en Java y su principal uso es como IDE para Java, este es un lenguaje neutral. El soporte para desarrollo en Java es proveído por un componente embebido o *plugin*, pero además están disponibles *plugins* para otros lenguajes, como C/C++, Cobol, C# y *ActionScript* (DAVID GALLARDO 2003). En principio permite ejecutar un programa sobre cualquier plataforma. Es una extensible plataforma de código abierto (*open source*) para desarrollar herramientas. Es administrado y dirigido por un consorcio de compañías de desarrollo de software con un interés comercial en promover Eclipse como plataforma compartida para herramientas de desarrollo de software.

1.8.3 *Plugins*: *Adobe Flex Buider* y Subclipse

Un *plugin* o *plug-in*, es una aplicación que interactúa con otra para agregarle una funcionalidad específica y es ejecutada por la aplicación principal. En el caso particular de Eclipse no son más que un conjunto de clases que permiten hacerlo más extensible.

Adobe Flex Builder

Es un *plugin* altamente productivo basado en la herramienta de desarrollo Eclipse que permite la codificación inteligente, depuración interactiva y diseño de la interfaz de usuario de las Aplicaciones Enriquecidas de Internet. *Flex Builder* está disponible en ediciones *Standard* y *Professional*. Este potente *plugin* incluye editores de MXML, *ActionScript* y CSS, así como coloración de sintaxis, completamiento de código, colapso de código, y mucho más.

Subclipse

Subclipse es un *plugin* para Eclipse que adiciona integración para el control de versiones (*Subversion*, específicamente), permitiendo operaciones de sincronización, actualización, entre otras. Permite bloqueos a recursos para que otros usuarios no puedan modificarlo. Dispone de una vista de comparación entra el recurso local y remoto en caso que exista conflicto entre la versión del recurso local con el remoto. Muestra una vista del historial de versiones de los recursos con un conjunto de atributos de las acciones realizadas sobre el recurso.

Soporta conectarse a varios repositorios de control de versiones al mismo tiempo, permitiendo hacer operaciones sobre el repositorio directamente.

Es un *plugin* muy útil para el desarrollo colaborativo, en el que intervienen un conjunto de desarrolladores trabajando sobre el mismo proyecto, poniendo a disposición del equipo de desarrollo facilidades para el trabajo en equipo.

1.8.4 Framework: Flex

Antes de llegar a la definición de qué es un *framework*, es imprescindible mencionar los conceptos de componente y componente de software. El concepto comúnmente aceptado de un componente es llamado “componente binario”, el cual es un artefacto de software compilado que es integrado dentro de la aplicación en tiempo de ejecución (*runtime*). El código fuente no es usualmente disponible, por tanto los componentes no pueden ser modificados. Si los componentes están implementados en un lenguaje de programación orientado a objeto, ellos pueden ser extendidos a través de la herencia o delegación. (KAISER 2005)

Recientemente, el interés en reutilizar software ha sido cambiado de la reutilización de componentes simples a diseño de sistemas enteros o estructuras de aplicaciones. Un sistema software que pudiera ser reutilizado en este nivel para la creación de aplicaciones completas es llamado *framework*. Los *frameworks* son basados en la idea que deberían permitir la producción fácil de un conjunto de sistemas específicos pero similares, dentro de un cierto dominio comenzando desde una estructura genérica. Brevemente, los *frameworks* son arquitecturas genéricas integradas por un extensible conjunto de componentes. Además, los *frameworks* pueden contener *subframeworks* los cuales representen subconjuntos de componentes de un sistema más grande. (KAISER 2005)

No hay una definición común para los *frameworks*, pero existe un tema común: la reutilización. Una definición ampliamente aceptada es dada por R. E. Johnson, and B. Foote en 1988 en su publicación *Designing Reusable Classes* (R. E. JOHNSON, B. FOOTE 1988):

"Un *framework* es un conjunto de clases que personifican un diseño abstracto para soluciones de una familia de problemas relacionados..."

Otro punto de vista, según R. E. Johnson y V. F. Russo en *"Reusing Object-Oriented Designs"* incluye (R. E. JOHNSON, V. F. RUSSO):

"Una clase abstracta es un diseño para un objeto simple. Un *framework* es el diseño de un conjunto de objetos que colaboran para llevar a cabo un conjunto de responsabilidades. De esta manera los *frameworks* son diseños a escalas más grandes que las clases abstractas. Los *frameworks* son una forma de reutilizar el diseño a un nivel superior."

Flex

Es un *framework* orientado al desarrollo de RIA. Da la facilidad de desarrollarlas basado en componentes que implementan funcionalidades básicas y avanzada.

Las principales características son:

La disposición y el prototipo de las interfaces de la aplicación Flex vienen con una interfaz de diseño visual. Permite moverse de la vista *Design* (diseño) y *Code* (código) o si prefiere, puede ver ambas a la vez. Las dos vistas de la aplicación se mantienen sincronizadas, de manera que los efectos de los cambios hechos en una, se pueden ver en la otra. Presenta una vista para escoger todas las opciones para el contenedor o componente de MXML que esté escribiendo, una vista para estilizar y dar soporte de CSS, una vista para la conectividad y enlace con la BD y los componentes personalizados y MXML son compatibles con la vista de diseño.

El editor de código permite una sincronización permanente entre la vista de diseño y la de código, un cambio en una se refleja en la otra. Durante la codificación permite realizar sugerencias a través del coloreado del código, la conclusión de una etiqueta, y la validación, esto contribuyen a una mayor productividad. Presenta una ayuda de referencias y admite lenguaje MXML y *ActionScript* y es compatible con HTML, XHTML, XML, JSP, ASP, Microsoft ASP.NET, PHP, y Macromedia *ColdFusion*.

Brinda soporte para la depuración pues aísla los problemas en el código. Se alinea con el depurador de *ActionScript*, aprovecha la integración estrecha con el entorno de Flex para identificar los problemas de *ActionScript* con facilidad y sin perder tiempo. Incluye la supervisión del tráfico de la red fundamentalmente entre el servidor y la aplicación-cliente de Flex, y viceversa. Posee un navegador interno que le permite la vista preliminar de la aplicación en MXML en el navegador web sin siquiera salir del entorno de trabajo.

Brinda soporte para el despliegue dado que se encarga de gestionar los archivos de la aplicación para el desarrollo, puesta a prueba, montaje y producción y permite la copia a cualquier servidor.

Las principales ventajas son:

- Generar un HTML más completo, se puede configurar el programa para que después de compilar, abra una *URL* en concreto, cosa muy útil cuando se trabaja con aplicaciones de datos que se comunican con un servidor o cargan archivos externos.
- En un mismo proyecto se puede tener varias aplicaciones y cada una genera un SWF diferente.
- Sigue una programación orientada a objetos y por ende incluye las ventajas de este paradigma.

La depuración es bastante avanzada y cuenta con gestión de memoria. Permite saber en tiempo de ejecución, cuántas instancias de cada objeto existen, y cuál es su peso. Esto es muy útil para optimizar nuestra aplicación, y ver por dónde puede estar fallando el rendimiento.

1.8.5 Lenguajes de programación: *ActionScript* 3.0

En la bibliografía consultada hay numerosas características en la utilización de *ActionScript* 3.0 (AS3) para el desarrollo de RIA, una de ellas es la combinación de la programación Orientada a Objetos con el uso de MXML lo que hace de la programación de interfaces gráficas de usuario mucho más fácil. Emplea un modelo realizando todas las comunicaciones cliente servidor de manera asíncrona, de este modo el usuario puede continuar trabajando en la pantalla y nunca se le congela. Ofrece un balance entre lenguajes de tipo estático y dinámico permitiendo

combinar los estilos de Java y *JavaScript*. Se puede consultar en el Anexo 10 la tabla de comparación entre los lenguajes Java y AS3.

Las clases codificadas usando AS3 deben ser almacenadas en ficheros .as externos, se puede pensar que esto es un obstáculo, pero brinda la posibilidad de trabajar y organizar las clases en un estructura de carpetas y organizar por la tarea específica que desempeñan, esto evita los conflictos de nombres, permite la estructurar paquetes dentro de las capas lógicas y hacer más fácil la importación y reutilización. Existe una convención de nomenclatura muy común con los paquetes que vienen de lenguajes java.

1.8.6 Sistemas de gestión de base de datos: MySQL y PostgreSQL

MySQL

MySQL es un sistema de gestión de base de datos relacional, multihilos y multiusuario. Es muy utilizado en aplicaciones web y en plataformas (Linux/Windows-Apache-MySQL-PHP/Perl/Python), y por herramientas de seguimiento de errores como Bugzilla. Su popularidad como aplicación web está muy ligada a PHP, que a menudo aparece en combinación con MySQL. Presenta características como son: conectividad segura, transacciones y claves foráneas, replicación, búsqueda e indexación de campos de texto y disponibilidad en gran cantidad de plataformas y sistemas. (PECOS, 2009). Es un desarrollo que es privativo.

PostgreSQL

PostgreSQL cuenta con sofisticadas particularidades tales como la versión multicontrol de concurrencia (MVCC), replicación asincrónica, transacciones jerarquizadas, en línea, un sofisticado plan de consulta, y un excelente optimizador. Es altamente escalable, tanto en la formidable cantidad de datos que puede administrar como en la cifra de usuarios concurrentes que puede acomodar. Trabaja en varios sistemas operativos como Linux, UNIX y Windows. También resiste el acaparamiento de grandes objetos binarios, ya sean imágenes, sonidos o vídeo. Tiene interfaces de programación originario de C / C + +, Java, Net, Perl, Python, Ruby, Tcl, ODBC, entre otros". (PECOS, 2009) Además no es controlado por ninguna compañía responde al esfuerzo de una comunidad global de desarrolladores.

Analizando las características de los dos gestores de bases de datos a través de la tabla que se muestra en el Anexo 11 se puede concluir que MySQL carece de soporte para transacciones, *rollback's* y subconsultas. No es viable para su uso con grandes bases de datos, a las que se acceda continuamente, ya que no implementa una buena escalabilidad. El hecho de que no maneje la integridad referencial, hace de este gestor una solución pobre para muchos campos de aplicación. Sin embargo, PostgreSQL implementa todas estos requisitos e incluye la posibilidad de almacenar imágenes, sonidos y videos, elementos indispensables en la creación de RIA.

1.8.7 Herramientas de modelado 3D

Blender

Es una suite integrada de modelado 3D, animación, dibujado, post-producción, creación interactiva y reproducción de juegos. Blender tiene su propia interfaz de usuario, realizada enteramente en OpenGL y diseñada con la idea de aumentar las facilidades de uso. Están disponibles enlaces con las librerías de Python para su edición; brinda opciones de importación y exportación de formatos de archivos populares como *3D Studio* y *Wavefront Obj*, las cuales son implementadas por la comunidad de software libre. Animaciones, modelos para juegos u otros motores de terceras partes y contenido interactivo son productos de uso común en Blender.

1.8.8 Herramientas de modelado

Las herramientas de modelado utilizan comúnmente el Lenguaje Unificado de Modelado (UML), se utilizan para capturar, guardar, rechazar, integrar automáticamente información y diseño de documentación.

Visual Paradigm

Visual Paradigm es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El software de modelado UML ayuda a una más rápida construcción de aplicaciones de calidad, mejores y a un menor coste. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas. Además, la herramienta

es colaborativa, es decir, soporta múltiples usuarios trabajando sobre el mismo proyecto; genera la documentación del proyecto automáticamente en varios formatos como web o PDF y permite control de versiones. Cabe destacar igualmente su robustez, usabilidad y portabilidad. (*FREE DOWNLOAD MANAGER*, 2007). Por otro lado “Es software propietario y tiene licencia comercial”. (GIRALDO, ZAPATA, 2005). Está herramienta es una de las propuesta a institucionalizar en la universidad como parte del proceso de mejora, además se realizo el proceso de compra de la licencia para su uso.

Contiene entre sus opciones *Visual Paradigm* for *UML Enterprise Edition* es la de modelado UML y se integra con entornos de desarrollo como Eclipse y NetBean. Desarrolla el modelado desde el negocio hasta la generación de código e ingeniería inversa realizando una cuidadosa integración y de esta forma reduce el esfuezo dedicado a estas tareas. Permite generar los EBJ y así mismo descriptores de despliegue para varios servidores de aplicación. Puede emigrar sus diagramas al Visio y al *Rational Rose*, soportando UML 2.0.

Rational Rose

Rational Rose es una herramienta con plataforma independiente que ayuda a la comunicación entre los miembros del equipo, a monitorear el tiempo de desarrollo y a entender el entorno de los sistemas. Utiliza la notación estándar en la arquitectura de Software (UML), la cual permite a los arquitectos de software y desarrolladores visualizar el sistema completo utilizando un lenguaje común. Abarca todo el ciclo de vida de un proyecto: concepción y formalización del modelo, construcción de los componentes, transición a los usuarios y certificación de las distintas fases. Los diseñadores pueden tomar ventajas de esta herramienta, porque permite que la salida de una iteración sea la entrada de la próxima que está por venir. Puede generar código en Java, C++, *Visual Basic*. Ada, Corba y Oracle. (MENENDEZ, 2007). Es software propietario pero al pertenecer a una corporación de los Estados Unidos no puede adquirirse la licencia, pero además la licencia que otorgan es por puesto de trabajo lo que incrementa sustancialmente el costo de un producto.

1.8.9 Servidores de aplicaciones

Los servidores de aplicaciones se crearon con el fin de gestionar las peticiones del usuario y devolver a los mismos recursos a través de un protocolo de comunicación.

Servidor web: Servidor Apache

El Servidor Apache es un Servidor HTTP de código abierto, está patentado por licencia BSD. Muestra entre otras características mensajes de error altamente configurables, bases de datos de autenticación y negociado de contenido, es altamente adaptable, utiliza una gran suma de Perl, PHP y otros lenguajes de script pero también trabaja con Java. Ventajas que presenta: es modular, *open source*, multi-plataforma, extensible y popular ya que es muy fácil conseguir ayuda y soporte. (*THE APACHE SOFTWARE FOUNDATION*, 2009)

1.8.10 Consideraciones sobre las herramientas

El uso de las herramientas es un elemento indispensable para la optimización de cualquier proceso. Por lo tanto el mejoramiento continuo en el uso de las mismas por cada desarrollador y el uso de herramientas estandarizadas logra elevar los niveles de productividad. En las organizaciones se debe conocer qué utilidad tiene cada herramienta, los mecanismos para implantarla y la evaluación de las mismas. La amplia variedad de tecnologías existentes para el desarrollo de productos constituye un factor importante para decidir cuales utilizar. Como resultado del análisis anterior se puede identificar un catalogo de posibilidades y las ventajas de su uso pero sería necesario caracterizar el dominio de aplicación para decidir.

1.9 Consideraciones finales

En el capítulo se han identificado los elementos fundamentales para desarrollar la propuesta de una Arquitectura de Software de Dominio Específico compuesta por un Dominio de Aplicación, los Requisitos de Referencia, una Arquitectura de Referencia y las Herramientas del Ambiente de Desarrollo para desarrollar las aplicaciones de la línea de producción. Logrando de esta forma el más alto nivel de la reutilización de componentes y acortando el tiempo de desarrollo de la misma. Los elementos a tener en cuenta a la hora de realizar una DSSA son los que se mencionan a continuación:

- El Dominio de Aplicación debe describir el alcance del entorno de negocio y el ambiente técnico.

- Los Requisitos de Referencia deben enunciar los requerimientos funcionales y no funcionales que describen el Dominio de Aplicación.

- La Arquitectura de Referencia debe incluir la propuesta base de diseño de las capas lógicas de la aplicación, las convenciones o estándares de código y de recursos, la estructura de código y recursos y los mecanismos de colaboración entre los componentes.

- El Ambiente Integrado de Desarrollo está compuesto por una gama de herramientas para que los desarrolladores puedan usar. En él se identifica herramientas de programación, *plugins*, *framework*, gestores de base de datos, herramienta de modelado y modelado 3D, servidores de aplicaciones, entre otras herramientas de soporte y gestión.

Se ha identificado elementos importantes a tener en cuenta en las temáticas abordadas sobre Ingeniería de Software, Modelo de proceso de software, proceso, metodologías y herramientas llegando a la conclusión de que no existe una opción que lo reúne todo, se considera que es posible combinarlas. Por ello el proceso de desarrollo a proponer debe basarse en el Modelo de Construcción de Prototipos y el Modelo de Desarrollo Rápido de Aplicaciones, en la Metodología RUP y debe estar compuesto por el orden lógico de actividades, los roles, las tareas, los artefactos de entrada y salidas y las listas de chequeo. Esto garantiza el uso de buenas prácticas y el desarrollo de flujos de trabajo organizados, estructurado y disciplinado, donde la organización tenga capacidad para repetirlos y mejorarlos.

CAPÍTULO 2: PROPUESTA DE DSSA Y PROCESO DE DESARROLLO DE SOFTWARE

2.1 Introducción

En el presente capítulo se hace una reseña de la Universidad de las Ciencias Informáticas (UCI), por ser el ambiente en que se implantará este proceso, haciendo referencia a varias de sus políticas y normativas que influyen en la propuesta. Además se describen los métodos, procedimientos y técnicas utilizadas para llevar a cabo la investigación.

Se presenta la propuesta de Arquitectura de Software de Dominio Específico (DSSA, por sus siglas en inglés) y el Proceso de Desarrollo de Software, identificando sus elementos principales, hasta llegar a las partes que los componen. Se establece el Dominio de Aplicación, los Requisitos de Referencia, la Arquitectura de Referencia y se describen los flujos de trabajo de ingeniería.

2.2 Entorno de implantación: Universidad de las Ciencias Informáticas

Tradicionalmente la universidad genera conocimiento pero es muy difícil que lo aplique, por lo que se aleja de los problemas de la producción conllevando a que las investigaciones no respondan a las necesidades de la industria. Por otra parte a la industria le es muy difícil investigar pues tienen que depender de un mercado cada vez más exigente en tiempo, costo y calidad.

La búsqueda de soluciones ha conllevado a la vinculación Universidad-Industria; es esta una alianza estratégica de intercambio donde la primera obtiene la facilidad de aplicar sus investigaciones y de vincular sus estudiantes y profesores al mundo empresarial y de funcionar como una entidad empresarial, la segunda recibe el conocimiento, la innovación constante que generan las universidades y el empleo de capital humano capacitado, joven y barato en la producción. Este nuevo concepto implica que la producción pasa a jugar un papel tan importante como la docencia.

La Universidad de las Ciencias Informáticas (UCI) constituye un nuevo modelo de formación – investigación – producción, presenta una fuerte formación Docente – Productiva, las facultades se vinculan a proyectos productivos y se especializan en segundos perfiles asociados a polos

productivos tales como: Bioinformática, Informática Educativa y Multimedia, Realidad Virtual, Inteligencia Organizacional, Seguridad Informática, Administración de Redes y otras.

Uno de los polos productivos es Realidad Virtual (RV), el cual ha puesto a disposición de los clientes productos dedicados a la educación, el entretenimiento y el entrenamiento, desarrollados como aplicaciones de escritorio.

Para organizar la producción se creó en la Universidad la Infraestructura Productiva (IP) que dirige metodológicamente los proyectos, la cual se subordina a la Vicerrectoría de Producción. La IP presenta un conjunto de direcciones vinculadas a diferentes líneas de producción y áreas para la gestión de la producción, entre ellas existe la Dirección de Calidad, responsable de normar los elementos para la producción, asegurar la calidad de los proceso y productos, y desarrollar las pruebas de liberación y aceptación de las soluciones informáticas para su entrega al cliente.

Entre los elementos normados existe el Expediente de Proyecto versión 2 y los Lineamientos de Calidad. El primero busca establecer y estandarizar la documentación de los productos en la Universidad para su desarrollo, así como el mantenimiento y reutilización en desarrollos posteriores, en el mismo se recogen los elementos técnicos, de gestión y soporte. El segundo se dirige a establecer e institucionalizar las buenas prácticas de la ingeniería de software en los proyectos productivos.

2.3 Métodos, procedimientos y técnicas utilizados

El método científico de investigación es la forma de abordar la realidad, de estudiar la naturaleza, la sociedad y el pensamiento, con el propósito de descubrir su esencia y sus relaciones. (ROLANDO ALFREDO HERNÁNDEZ LEÓN 2002).

En la investigación como métodos de investigación se han utilizado los Métodos teóricos: histórico lógico, hipotético deductivo, sistémico.

Los métodos teóricos permiten comprender el fenómeno que se estudia, su evolución, elaborar la conjetura científica y propone las mejoras a los problemas que se identificaron. Se analizó cómo han madurado los conceptos de arquitectura e ingeniería de software desde su surgimiento hasta ahora y su impacto con análisis histórico lógicos. La influencia de los elementos modelo de proceso de desarrollo, procesos, métodos y herramientas en el desarrollo de proyectos informáticos desde el punto de vista sistémico. Se enfocaron las características de la producción de software al dominio de aplicación con un enfoque hipotético deductivo.

En la primera parte de esta investigación se desarrolló un estudio del estado del arte; se revisaron las bondades y deficiencias de cada una de las propuestas para modelos de desarrollo de software, metodologías y herramientas, y las tendencias en la resolución de esta problemática.

Es importante destacar que la revisión bibliográfica constituyó un método importante para la concepción del proyecto de investigación y para apropiarse de los conocimientos relacionados con el tema.

La investigación sigue además un método hipotético deductivo porque a partir de la conjetura inicial se plantearon objetivos específicos que en el transcurso de la investigación fueron cumpliéndose siguiendo métodos fundamentados.

2.4 Definición del dominio

Un dominio es definido por un conjunto de problemas o funciones comunes que las aplicaciones en ese dominio pueden resolver o hacer. Con las tecnologías actuales, se hace complicado lograr que las aplicaciones web funcionen con eficacia e inmediatez, debido al problema que representa para los desarrolladores la tendencia de las mismas a volverse más interactivas con la incorporación de contenidos multimedia. Como solución a este problema el autor propone desarrollar aplicaciones que contengan escenarios tridimensionales interactivos usando las ventajas de una DSSA.

El dominio presupone establecer una terminología estándar para el problema y las funciones a ser ejecutadas, en tal sentido se ha enunciado que los escenarios tridimensionales están compuestos por escenas en tres dimensiones y estas poseen elementos que pueden ser videos, sonido e imagen. Las funciones fundamentales se dirigen a que los usuarios puedan interactuar con los escenarios tridimensionales. Para soportar estas operaciones es necesario utilizar como estilo arquitectónico llamada-retorno específicamente n-capas y en la capa cliente el patrón modelo-vista controlador desarrolladas con Adobe Flex y soportadas por Adobe Flash Player, que incluyan escenarios tridimensionales interactivos y permitan al cliente una visualización óptima desde cualquier plataforma. Los datos que se manejan son los modelos 3D, los mapas de texturas y las imágenes. En la Figura 2-1 se representan los elementos que componen el dominio, diferenciando los ubicados en los clientes de aquellos ubicados en el servidor.

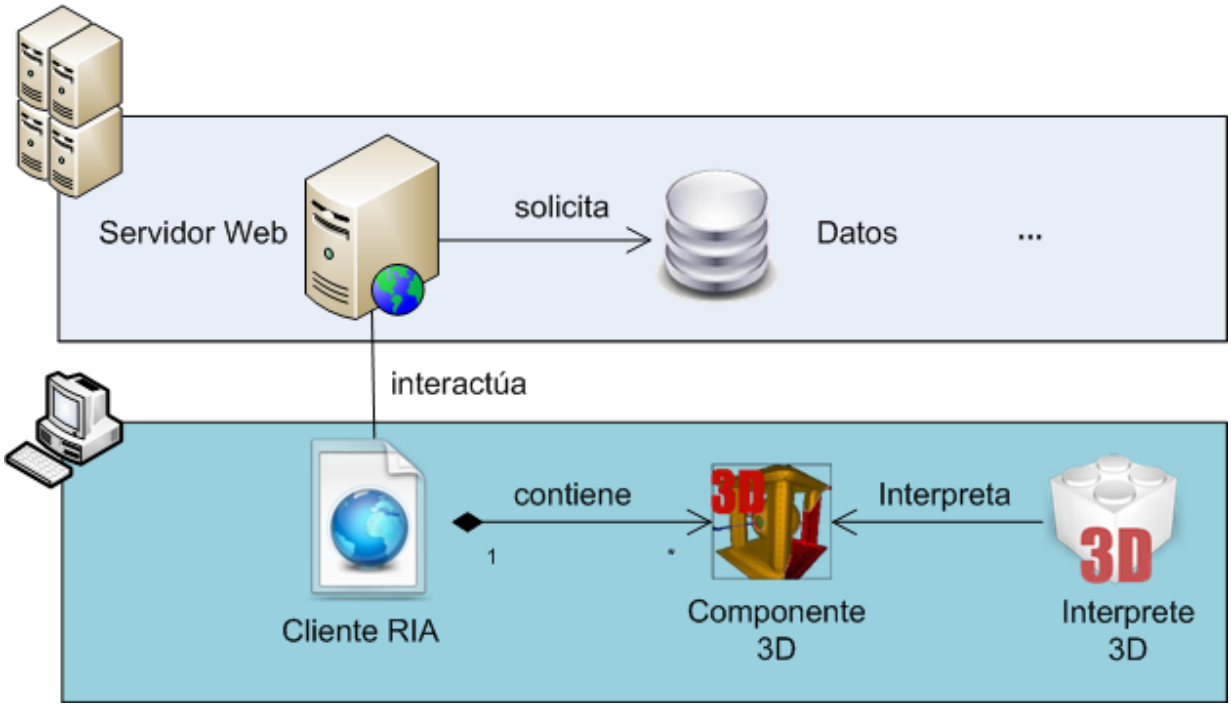


Figura 2-1: Representación gráfica del dominio de referencia.

2.5 Requisitos de Referencia

Varias aplicaciones se han desarrollado en el ámbito correspondiente al dominio definido anteriormente. En diferentes áreas de estas aplicaciones destaca el uso de elementos, semejantes en necesidades, para problemas similares. Estos problemas repetidos convergen en requerimientos comunes para toda aplicación en este dominio, y son expresados a través de los requisitos de calidad de la ISO-9126 establecidos en la UCI, los mismos son expuestos a continuación:

2.5.1 Funcionalidad

La arquitectura debe ser capaz de soportar la representación de escenarios tridimensionales, el comportamiento de los elementos que lo integran y la interactividad con los usuarios. Deben poder interactuar con otros sistemas como portales web, Bases de Datos y Aplicaciones locales. Desde el punto de vista de seguridad debe manejar los requisitos de integridad y confidencialidad de la información.

2.5.2 Usabilidad

Es necesario combinar los objetos tridimensionales con imágenes, videos y sonidos de manera que la información esté estructurada para simular el mundo real y elementos de la cotidianidad. Para interactuar el usuario con el escenario puede hacerlo usando el ratón y el teclado indistintamente con facilidad de uso y compresión.

2.5.3 Fiabilidad

El sistema debe ser altamente tolerante a los fallos, permitir que el usuario continúe interactuando con el escenario sin poder actualizarlo.

2.5.4 Portabilidad

Las aplicaciones desarrolladas deben ser multiplataforma, y capaz de ejecutarse en diversos dispositivos sin necesidad de adaptarlos a los diferentes ambientes. Los usuarios deben acceder desde cualquier plataforma y siempre obtener el mismo resultado.

2.5.5 Mantenibilidad

El sistema debe ser altamente mantenible para ello se usarán estándares de codificación, contenedor SWF, el desarrollo es basado en componentes reutilizables y en un modelo de construcción de prototipos.

2.5.6 Eficiencia

Es necesario responder a las peticiones del cliente en el menor tiempo posible, lograr la interacción de todas las capas de la aplicación de la manera más eficiente posible y aprovechar los recursos disponibles en el modelo Cliente/Servidor. Aumentar la velocidad de las consultas a la Base de Datos. El tráfico de información en la red no debe saturar el ancho de banda, y al mismo tiempo es importante mantener una calidad elevada en la visualización y en la optimización de los modelos 3D presentados al cliente.

2.6 Arquitectura de Referencia

La DSSA presenta una propuesta de arquitectura base para: orientar el diseño de las capas lógicas a través de una propuesta de diseño base; organizar la forma de codificar según las propuestas de convenciones o estándares de códigos y recursos; brindar una estructura física para soportar el código, creando así un esqueleto base; y proponer mecanismos de colaboración entre los componentes integrados en ella. Está basada fundamentalmente en el

estilo arquitectónico Cliente-servidor y Arquitectura en capas, usando el patrón modelo-vista-controlador.

2.6.1 Diseño de las capas lógicas

La estructura del diseño de las capas lógicas se representa en la figura 2-2.

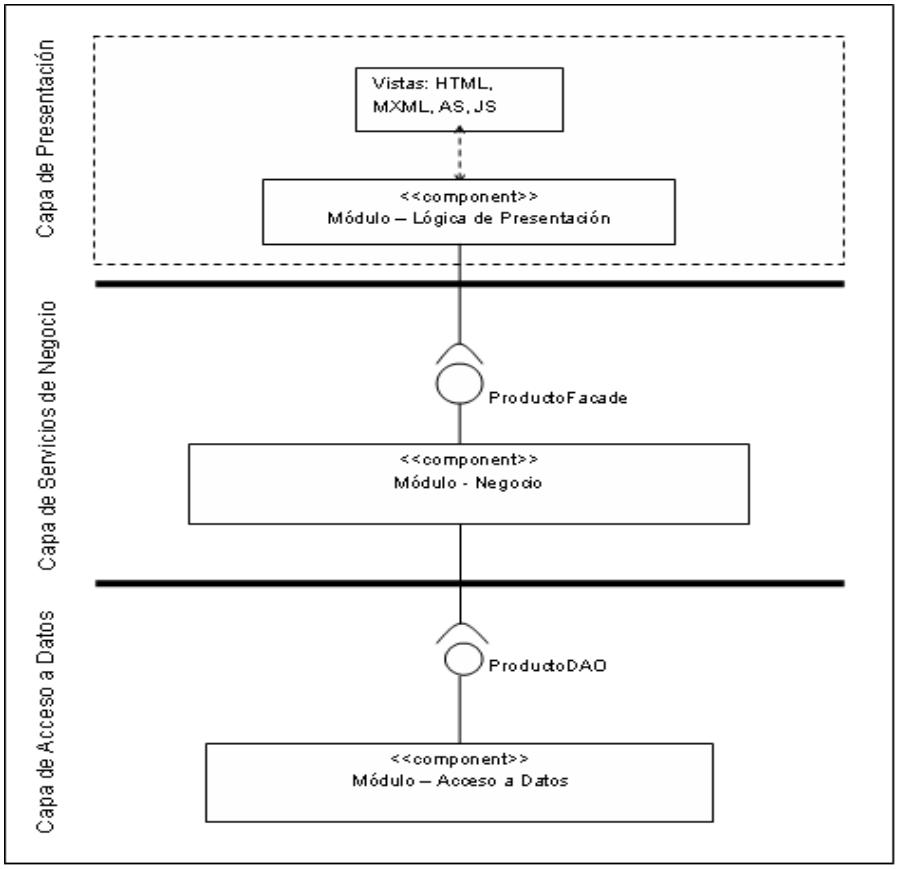


Figura 2-2: Representación de la arquitectura de las capas lógicas.

En la figura 2-3 los objetos de dominios son pasados hasta la capa de presentación, en la cual mostrará su representación a partir de los datos que contienen, pero no podrán ser modificados, ya que esto solo ocurre dentro de los contextos transaccionales definidos en la capa de servicios de negocio.

En una aplicación usando la propuesta de DSSA, sus capas lógicas corren en un servidor web o un servidor de aplicaciones. No se considera una arquitectura distribuida, aunque pudiera adaptarse a una aplicación que lo requiera.

Este estilo ayuda a desacoplar las capas arquitectónicas. De esta forma cada capa puede ser modificada tanto como sea posible sin provocar un impacto en las demás capas. Una capa no es consciente de lo que le ocurre a la capa superior, su dependencia es puramente con la capa inmediata inferior. Esta dependencia entre capas es normalmente entre interfaces, asegurando que el acople sea el más bajo posible. Seguidamente se analizará en más detalles cada una de las capas propuestas anteriormente comenzando de abajo hacia arriba según la figura 2-2.

Capa de acceso a datos

Manteniendo la filosofía de que es mejor programar orientado a interfaces que a clases, se soporta el uso de interfaces de acceso a datos entre la capa de servicios de negocio y el API de persistencia utilizado. El término de Objeto de Acceso a Datos o en inglés, *Data Access Object* (DAO) (DEEPAK ALUR 2003), es ampliamente usado en el desarrollo de software. Los DAO encapsulan la persistencia de los objetos de dominios, proveen la persistencia de los objetos transitorios y las actualizaciones de los objetos existentes en la base de datos. Las implementaciones de los DAO estarán disponibles para los objetos (típicamente para los objetos de negocio) haciendo uso de la inyección de dependencias con los objetos de negocio y las instancias de los DAO. Las interfaces de los DAO contienen básicamente los siguientes tipos de métodos:

- Métodos para descubrir: Estos localizan los objetos almacenados para ser usados por la capa de servicios de negocio.
- Métodos para persistir o salvar: Estos hacen persistentes a los objetos transitorios.
- Métodos para eliminar: Estos eliminan a los objetos guardados en el medio de almacenamiento (generalmente una base de datos).
- Métodos para conteos y otras funciones agregadas: Estos devuelven los resultados de operaciones que son más eficientes implementarlas usando funcionalidades de la base de datos (procedimientos almacenas, etcétera.) que iterar sobre los mismos objetos.

En la figura 2-3, se muestra un ejemplo simple de los elementos que debe presentar un diagrama de clase de esta capa. Tomemos por ejemplo una tienda virtual.

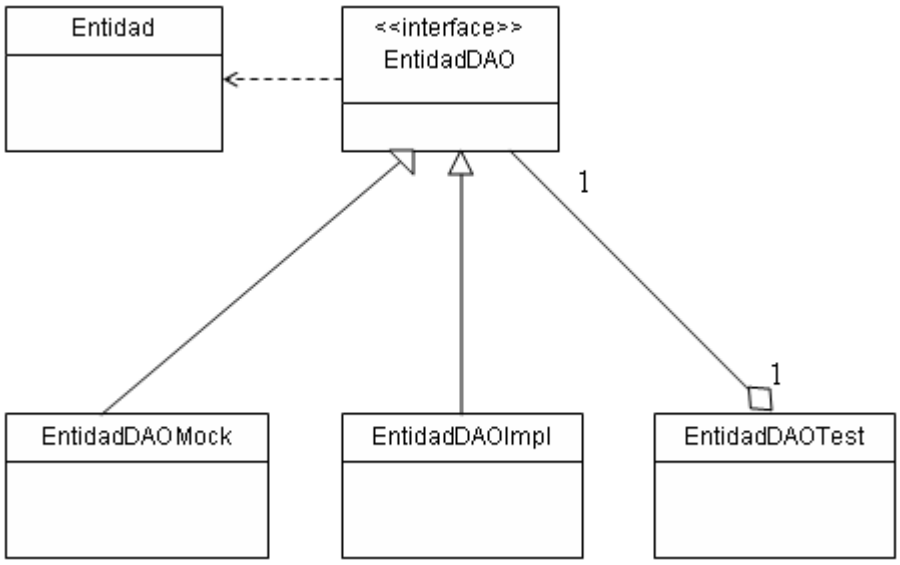


Figura 2-4: Diagrama de clases de la capa de acceso a datos.

- Entidad: es la entidad de dominio a ser persistida por los métodos expuesto en la interfaz EntidadDAO.
- EntidadDAO: Es la interfaz que expone a la capa de servicios de negocio los métodos del DAO para persistir la entidad de dominio Entidad.
- EntidadDAOImpl: Esta clase es la implementación de la interfaz que realmente se conecta a la base de datos usando LCDS. Dando de esta forma la funcionalidad real a los métodos de EntidadDAO.
- EntidadDAOMock: Esta clase implementa los métodos expuestos en la interfaz EntidadDAO pero generando datos falsos estáticamente o usando objetos moqueados o falsos, sin conectarse a ninguna base de datos u otra fuente de almacenamiento. Esto permite el desarrollo en paralelo de las tres capas.
- EntidadDAOTest: Es la clase que prueba todos los métodos de la clase EntidadDAOImpl, utilizando valores que testeen casos extremos. Además, es la responsable de asegurar que los métodos del objeto de acceso a datos están completamente funcionales, para utilizar el DAO desde la capa de servicios de negocio. Al igual que la clase EntidadDAOMock, permite el desarrollo en paralelo de las tres capas.

Capa de servicios de negocio

A pesar de usar la arquitectura que define la propuesta, los objetos de dominio pueden tener lógica de negocio. Sin embargo, la capa de servicios de negocio tiene un rol importante. En esta capa radican los objetos de negocio o *Business Objects* (DEEPAK ALUR 2003). Los objetos de negocio separan los datos y la lógica de negocio usando un modelo de objetos.

Estas son las funcionalidades que presenta esta capa:

- Lógica de negocio específica de procesos de negocios: A veces es más oportuno para los objetos de dominio contener lógica de negocio aplicable a muchos casos de uso específicos. Sin embargo, existen casos de usos específicos que son realizados en la capa de servicios de negocios. Pero se propone que en esta definición de arquitectura los objetos de dominio no presenten ningún tipo de lógica de negocio, sino que esta responsabilidad recaiga sobre los objetos de negocio, permitiendo usar a los objetos de dominio como *Transfer Objects* que se mueven entre las capas arquitectónicas de la aplicación.
- Puntos de entrada muy bien definidos para las operaciones de negocio implementadas: Los objetos de negocio brindan las interfaces usadas por la capa de presentación.
- Control de transacciones: Aunque a veces la lógica de negocio pueda ser movida a objetos de dominio, el control de transacciones no debería. Sino que las políticas transaccionales de la aplicación deberían ser planteadas al nivel de los objetos de negocio.
- Ejecución de restricciones de seguridad: Las restricciones de seguridad en esta capa es en los puntos de entradas a la capa media, es decir en los objetos de negocio.

En la figura 2-4 se muestra en más detalles el diseño básico de clases en esta capa, semejante a como se hizo con la capa explicada anteriormente. Es importante resaltar que estos diseños básicos de clases de las capas, pueden ser enriquecidos a través del desarrollo de software de una aplicación específica del dominio, según sus requerimientos. Se pretende lograr la estructuración básica de las aplicaciones a nivel de diseño que utilicen la arquitectura base propuesta, orientando a los diseñadores en cómo deben realizar el diseño de cada una de las capas lógicas de la aplicación.

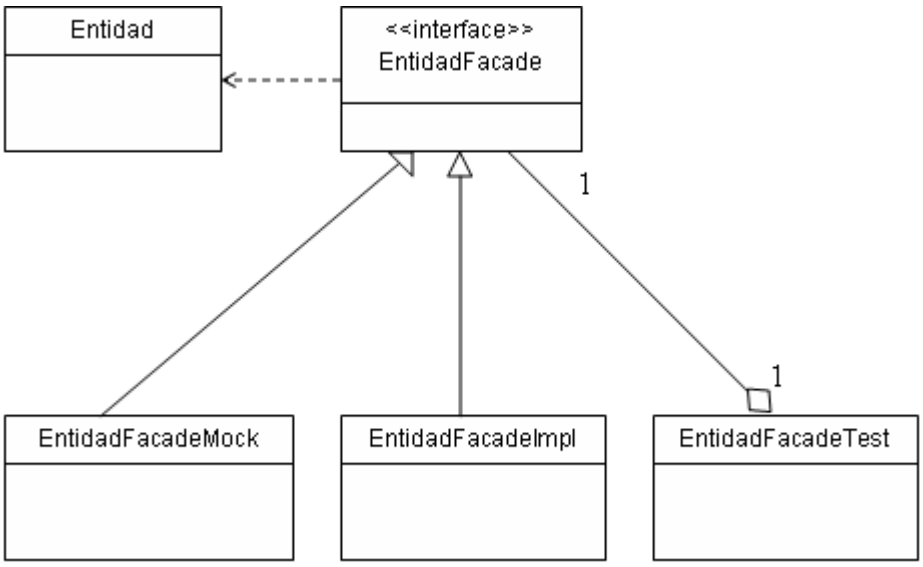


Figura 2-4: Diagrama de clases de la capa de servicio de negocio.

Los elementos que intervienen en la figura 2-4, tienen un comportamiento parecido, pero no exactamente igual al de la figura 2-3:

- Entidad: Es la entidad de dominio que utilizarán los métodos expuestos en la interfaz de negocio EntidadFacade.
- EntidadFacade: Es la interfaz que expone a la capa de presentación los métodos para las operaciones de negocio sobre la entidad de dominio Entidad. A modo de aclaración: puede ocurrir que un objeto de negocio exponga métodos de negocio de varias entidades del dominio o de procesos específicos de negocio.
- EntidadFacadeImpl: Esta clase es la implementación de la interfaz de negocio desarrollando los métodos que contendrán la lógica de negocio.
- EntidadFacadeMock: Esta clase implementa los métodos expuestos en la interfaz EntidadFacade, pero generando datos falsos estáticamente o usando objetos moqueados o falsos, sin realizar ninguna operación de negocio real y sin utilizar ningún DAO real, en última instancia usando los DAO falsos.
- EntidadFacadeTest: Es la clase que prueba todos los métodos de la clase EntidadFacadeImpl, utilizando valores que prueben los casos extremos. Esta clase es la responsable de asegurar que la lógica de negocio implementada en el objeto EntidadFacadeImpl responde a los requerimientos que debe cumplir. En estas pruebas el objeto de negocio (EntidadFacadeImpl), no tiene necesariamente que usar los DAO

reales, sino los DAO falsos, ya que el objetivo de estas pruebas no es probar la parte de persistencia de las operaciones de negocio implementadas.

Capa de presentación

En la propuesta la capa de presentación descansa sobre una capa de servicios de negocio. Esto significa que esta capa será fina y no contendrá lógica de negocio, sino simplemente lo concerniente a aspectos de presentación, por ejemplo, el código para manipular las interacciones web. Por tanto, se pueden elegir más de una capa de presentación en la misma aplicación, sin impactar en las capas arquitectónicas inferiores, las cuales no deberían tener conocimiento sobre la capa de presentación.

Esta capa se basa en el patrón modelo-vista-controlador y es la responsable de tratar con las interacciones del usuario y obtener los datos que pueden ser mostrados en un formato determinado. Normalmente está compuesta por tres tipos de objetos:

- Controladores: Estos objetos son responsables de procesar las entradas del usuario en forma de peticiones HTTP, invocando las funcionalidades necesarias, expuestas por la capa de servicios de negocio y devolviendo un modelo requerido para ser mostrado.
- Modelo: Estos objetos contienen los datos resultantes de la ejecución de la lógica de negocio, los cuales deberían ser mostrados en la respuesta.
- Vistas: Estos objetos son responsables de mostrar el modelo en la respuesta de la petición. La forma de mostrar el modelo podrá ser de diferentes tipos de vistas, por ejemplo, archivos JSP, HTML, SWF, documentos de Excel, etcétera. Las vistas no son responsables de modificar los datos o incluso de obtener los datos; estas simplemente sirven para mostrar los dados del modelo que han sido suministrados por un controlador.

2.6.2 Convenciones o estándares de código y de recursos

Con el fin de lograr un lenguaje común y uniforme para las distintas aplicaciones que utilicen la arquitectura base definida, se especifican convenciones de nombres y estándares de código para los distintos recursos las aplicaciones.

Se definen convenciones de nombres para las distintas clases dependiendo de las funciones

que tengan cada una de estas en la aplicación:

Capa de Acceso a Datos

- Las interfaces que representan las operaciones sobre los objetos de acceso a datos, correspondientes al patrón de diseño *Data Access Object* (DEEPAK ALUR 2003) terminan con la palabra “DAO”. Ejemplo: EntidadDAO.
- Las implementaciones reales de las interfaces DAO comienza con el nombre de la interfaz correspondiente y terminan con la palabra “Impl”. Ejemplo: EntidadDAOImpl.
- Las implementaciones falsas de las interfaces DAO comienzan con el nombre de la interfaz correspondiente y terminan con la palabra “Mock”. Ejemplo: EntidadDAOMock.
- Las clases utilizadas para realizar pruebas a los interfaces DAO comienzan con el nombre de la interfaz correspondiente y terminan con la palabra “Test”. EntidadDAOTest.

Capa de servicios de negocio

- Las interfaces que representan las operaciones del negocio terminarán con la palabra “Facade”. Ejemplo: EntidadFacade.
- Las implementaciones de las interfaces de negocio comenzarán con el nombre de la interfaz correspondiente y terminaran con la palabra “Impl”. Ejemplo: EntidadFacadeImpl.
- Las implementaciones falsas de las interfaces de negocio comienzan con el nombre de la interfaz correspondiente y terminan en la palabra “Mock”. Ejemplo: EntidadFacadeMock.
- Las clases utilizadas para probar las funcionalidades del negocio terminan con la palabra “Test”. Ejemplo: EntidadFacadeTest.

Capa de Presentación

- Las clases que manejan el flujo de la capa de presentación, es decir, los controladores, terminarán con la palabra “Controller”.
- Las clases utilizadas para hacer pruebas de unidad a los Controladores comenzarán con el nombre del controlador correspondiente y terminará con la palabra “Test”.
- Las clases utilizadas para validar datos (Validadores) terminarán con la palabra “Validator”.

- Las clases utilizadas para guardar datos procedentes de las vistas, utilizando el patrón Command (GRAND) definido en los patrones GoF, tendrán un nombre lógico de acuerdo a los datos que contiene seguido de la palabra “Command”.

Recursos

- Las páginas HTML que constituyen recursos estáticos tendrán la extensión “.html”.
- Las imágenes en la aplicación tendrán la extensión “.jpg” o “png”.
- Las clases en *ActionScript* tendrán la extensión “.as”.
- Los modelo 3D tendrán la extensión “.obj” o “3ds”.
- Los mapas de textura tendrán la extensión “.bmp”.
- Las clases JavaScript tendrán la extensión “.js”.

También se define convenciones de nombre para archivos que tienen que ver con la configuración de la aplicación.

Los archivos tendrán la siguiente estructura:

[nombre del proyecto]-[subsistema]-[módulo]-[capa lógica]-context-[tipo].xml

- [nombre del proyecto]: representa el nombre del proyecto en abreviatura.
- [subsistema], [módulo]: indican las unidades organizacionales utilizadas en el proyecto de forma de árbol, pueden ser tantas como la complejidad del mismo lo requiera según las estructuras definidas para cada nivel de complejidad.
- [capa lógica]: representa la capa lógica que maneja. Se establecen las siguientes clasificaciones explicada a continuación: [*dataaccess*]: Contiene los *beans* con las funcionales referentes a la capa de acceso a datos. [*servlet*]: Encapsula todo los *beans* de la capa de presentación. Cuando no se especifica el tipo se toma por defecto que se refiere a las operaciones de negocio de la aplicación, en este XML se definen las fachadas y otros objetos que representen la capa de servicios.
- [tipo]: Se refiere al tipo de función que cumplirá en la aplicación, si es utilizado para definir pruebas a la capa correspondiente el valor sería “test”, cuando se utiliza para configurar los *beans* falsos sería “mock”.

2.6.3 Estructuras de código y recursos

Teniendo en cuenta la arquitectura propuesta se define una estructura que permite organizar cada tipo de clase o componente necesario en el desarrollo de software.

Organización de la aplicación en unidades organizacionales.

Con el objetivo de dividir y organizar las aplicaciones se crean paquetes por cada unidad organizacional que se defina. En la arquitectura propuesta se definen estas unidades: subsistemas y módulos. Un subsistema se refiere a un conjunto de funcionalidades del sistema que tienen características muy propias y que a su vez están subdivididas en módulos. Un módulo encapsula un conjunto de funciones que debe realizar el sistema, las cuales son agrupadas por tener características muy similares y se definen en la etapa de diseño. El uso de subsistemas y módulos en la aplicación final está regido atendiendo a la complejidad que tenga la misma de acuerdo a las clasificaciones definidas.

Clasificaciones de complejidad.

Se crearon tres clasificaciones atendiendo a la complejidad que podría presentar el proyecto a realizar: baja, mediana y alta. Para cada una define una estructura de paquetes correspondiente atendiendo a las posibles necesidades que pueda presentar.

Baja: se define para soportar las necesidades de las aplicaciones que sean de tamaño pequeño debido a que presentan un solo módulo. Esta contiene el paquete raíz, el nombre del módulo y a continuación toda la estructura referente al mismo (Figura 2-5).

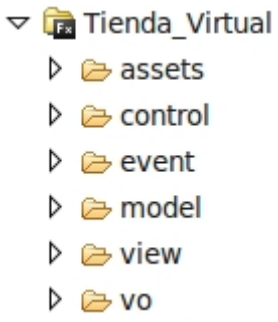


Figura 2-5: Representación de la clasificación de complejidad baja.

Mediana: responde a sistemas no muy grandes, que contengan solo un conjunto de módulos y no sea necesario definir subsistemas. Esta estructura está compuesta por el paquete raíz, seguido en el árbol por los paquetes de los módulos definidos (Figura 2-6).

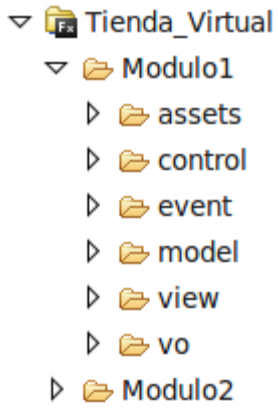


Figura 2-6: Representación de la clasificación de complejidad media.

Alta: responde a sistemas complejos, que contengan subsistemas y módulos, permitiendo además que se adicionen otros niveles organizacionales definidos por el equipo de arquitectura. Esta estructura está compuesta por el paquete raíz, seguido por los paquetes referentes a los subsistemas. Cada subsistema tendrá como hijos en el árbol de paquetes a los módulos que responden al mismo (Figura 2-7)

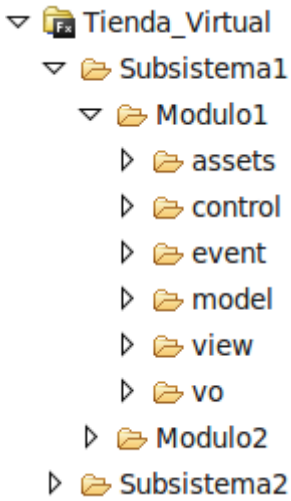


Figura 2-7: Representación de la clasificación de complejidad alta.

Estructura más atómica en organización de paquetes

La unidad más atómica en la estructuración de paquetes que se define es el módulo. Un módulo se puede comparar con una pequeña máquina que puede actuar por sí sola, siempre y cuando estén activas las demás máquinas de las que esta depende para su funcionamiento. En un módulo generalmente existen todas las capas lógicas que se definen en la arquitectura base propuesta. Cada componente que se desarrolle en un módulo tiene un lugar definido en esta estructura de paquete acorde a la complejidad de la aplicación. A continuación se expone esta estructura (Figura 2-8).

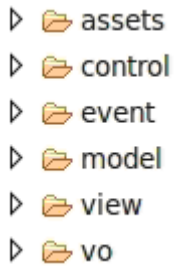


Figura 2-8: Representación de la estructura de paquete.

- *control*: carpeta que contiene las clases controladoras, encargada de escuchar los eventos lanzados por las vistas y llevará a cabo las acciones necesarias.
- *event*: en esta carpeta se guardan todos los eventos personalizados que se deban crear.
- *model*: carpeta donde se guarda la clase de nuestro modelo de datos y todas las clases que puedan estar relacionadas.
- *view*: la carpeta *view* es donde deben guardarse todos los archivos .mxml (y .as de soporte) que representen las diferentes vistas y subvistas en la aplicación.
- *vo*: pequeñas clases sin lógica (solo propiedades) que representan entidades en la aplicación. En la carpeta vo deben guardarse todos los *Value Objects* que sean necesarios.

- *assets*: carpeta que contiene los recursos que se utilizan en el proyecto, dígase imágenes, modelos tridimensionales y sonidos.

2.6.4 Mecanismos de colaboración entre los componentes.

Los mecanismos de colaboración son estrategias propuestas para establecer la forma de comunicación entre los componentes del software. Según las estructuras de código planteadas en la sección anterior, que dividen al sistema en estructuras físicas denominadas subsistemas y módulos de acuerdo con la complejidad del software (baja, media o alta), se proponen las estrategias de comunicación o colaboración entre ellos.

Como ya se ha explicado, un subsistema agrupa una colección de módulos. Por consiguiente, si se detecta que entre los módulos de un mismo subsistema existen dependencias, es decir, que algunos módulos necesitan funcionalidades implementadas en otros, entonces hay dos posibles propuestas de estrategia de colaboración a usar. Un módulo se puede ver como el encapsulamiento a nivel de componente de software de un conjunto de Historias de Usuario.

Dependencia directa

La dependencia directa es cuando un módulo específico de la aplicación tiene funcionalidades expuestas en una o más interfaces que otros módulos necesitan y éstos las utilizan directamente. En la figura 2-9 se muestra un ejemplo de las dependencias directas que existen entre el Módulo I y Módulo III con el Módulo II mediante la interfaz ExpedienteFacade. Esta figura es un ejemplo de una de las estrategias colaboración entre módulos de un sistema software.

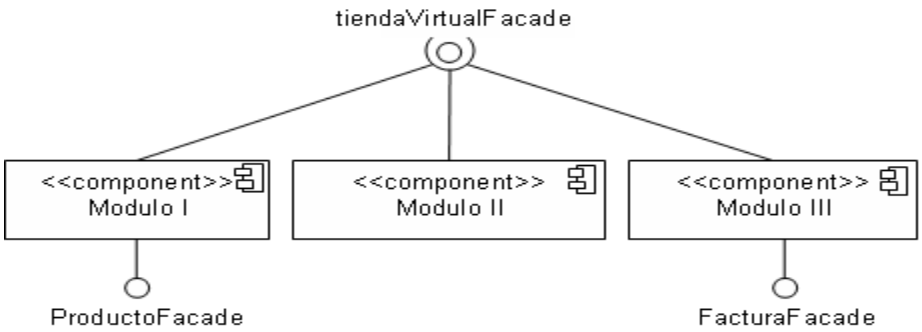


Figura 2-9 Representación de dependencias directas

Dependencia indirecta

La dependencia indirecta es cuando existen dependencias directas entre varios módulos, y no se desea que existan debido a que su impacto arquitectónico no ponga en riesgo algún requerimiento funcional o no funcional de la aplicación. Tomando como ejemplo la figura 2-9, se desea eliminar las dependencias directas que existen con el Módulo II a través de las funcionalidades expuestas en la interfaz *virtualShopFacade*, porque como se muestra en la figura, el Módulo II es imprescindible para los otros dos módulos ya que estos necesitan de la interfaz *virtualShopFacade*; y la aplicación tiene como uno de sus requerimientos no funcionales que en algunos casos la aplicación no contenga el Módulo II.

Para darle solución a este problema, como muestra la figura 2-10, se crea un nuevo módulo llamado "*common*" ("común", en español), para el que se moverá la interfaz *virtualShopFacade* y todas las clases y recursos implicados en soportar las funcionalidades expuestas por la interfaz. De esta forma, los módulos que antes dependían directamente del Modulo II a causa de usar las funcionalidades de la interfaz *virtualShopFacade* (ver figura 2-9), ya no lo harán, sino que tendrán dependencia directa del módulo común (ver figura 2-10), por lo que se logro cumplir con el requerimiento no funcional que presentaba la aplicación de poder eliminar en algún momento el Modulo II y los demás módulos continuaran su funcionamiento sin problemas.

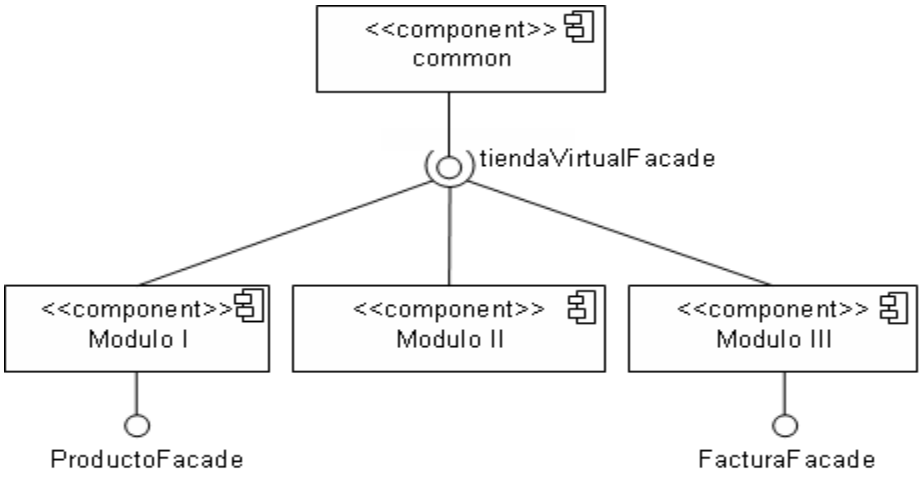


Figura 2-10 Ejemplo de dependencia directa del módulo common

Este módulo común, tiene la misma estructura básica de los módulos planteada en la sección anterior. Es un módulo que no se identificó inicialmente cuando se diseño el sistema, pero fue necesaria su creación para lograr eliminar dependencias directas entre los módulos específicos de la aplicación y moverlas hacia un módulo común para cumplir con requerimientos específicos de la aplicación.

Todo este proceso es entre los módulos específicos de un subsistema por lo que el módulo común creado pertenece también al subsistema. Además, este proceso también es aplicable para establecer este mecanismo de colaboración a nivel de subsistemas. La comunicación entre módulos y entre subsistemas es a través de las interfaces de la capa de servicios de negocio, que son puntos de entradas a la lógica de negocio implementada en ellos. De esta forma se logra un mayor desacople entre ellos, al ocultar detrás de las interfaces las implementaciones reales de cada funcionalidad.

2.7 Ambiente de Desarrollo

Para definir la propuesta de ambiente de desarrollo para el desarrollo de aplicaciones que se enmarquen en los elementos descritos hasta el momento se realizó un estudio de las características de cada herramienta mencionada en el capítulo 1. En el desarrollo de esta sección se precisaran las mismas.

Se propone Eclipse como herramienta para la programación. Incluyendo los *plugins Adobe Flex Builder* que amplían sus funcionalidades como plataforma de desarrollo de RIA y Subclipse para la integración con la herramienta de control de versiones. Para dar cumplimiento a los requerimientos de referencia se identificó el *framework* Flex. Utilizando como lenguaje de programación *ActionScript* 3.0. Para el servidor de aplicaciones Apache. Según el dominio de aplicación y los requerimientos de referencia se recomienda como gestor de base de datos usar PostgreSQL. Como herramienta de modelado se proponen dos una para modelar los escenarios tridimensionales Blender y para modelar la aplicación en sus diferentes modelos y vista *Visual Paradigm.*

Se propone además para la gestión de Requisitos OSRMT, para el Modelado Visual Paradigm para la gestión de proyecto *RedMine* por ser propuestas del programa de mejora que lleva la

UCI. Para el control de Versiones se utilizará el *Subversion* y para Seguimiento y control de errores el Trac y para salvas automáticas el Bacula por las facilidades de uso y la integración que existe entre estas.

2.8 Proceso de desarrollo de software

El desarrollo de software actual está siendo orientado por metodologías y modelos de proceso de desarrollo que se combinan y adaptan a las condiciones de las organizaciones. Mantener bien organizado y localizado cada elemento de la metodología es una tarea muy importante. En áreas de producción pensadas para elaborar sistemas diferentes esto se vuelve una necesidad vital. Definir el proceso productivo para el desarrollo del producto, es el objetivo fundamental de este epígrafe. El flujo de trabajo comprende el conjunto de actividades, los roles, las tareas, los artefactos de entrada y salidas y las listas de chequeo para guiar el desarrollo de los proyectos productivos.

Para garantizar una buena organización del proceso de desarrollo se van a tener en cuenta el Modelo de Construcción de Prototipos y el Modelo de Desarrollo Rápido de Aplicaciones y la Metodología RUP. Esta combinación permite que los usuarios puedan verificar el progreso del proyecto y el cumplimiento de lo establecido, además estará apoyado en el Modelo Basado en Componentes para obtener una arquitectura robusta y promover la reutilización efectiva de software.

El ciclo de vida de los productos, subsistemas o componentes contiene los siguientes flujos organizados consecutivamente: estudio preliminar, requerimientos, arquitectura y diseño, implementación, pruebas, despliegue y soporte. Como se muestra en la figura 2-11.

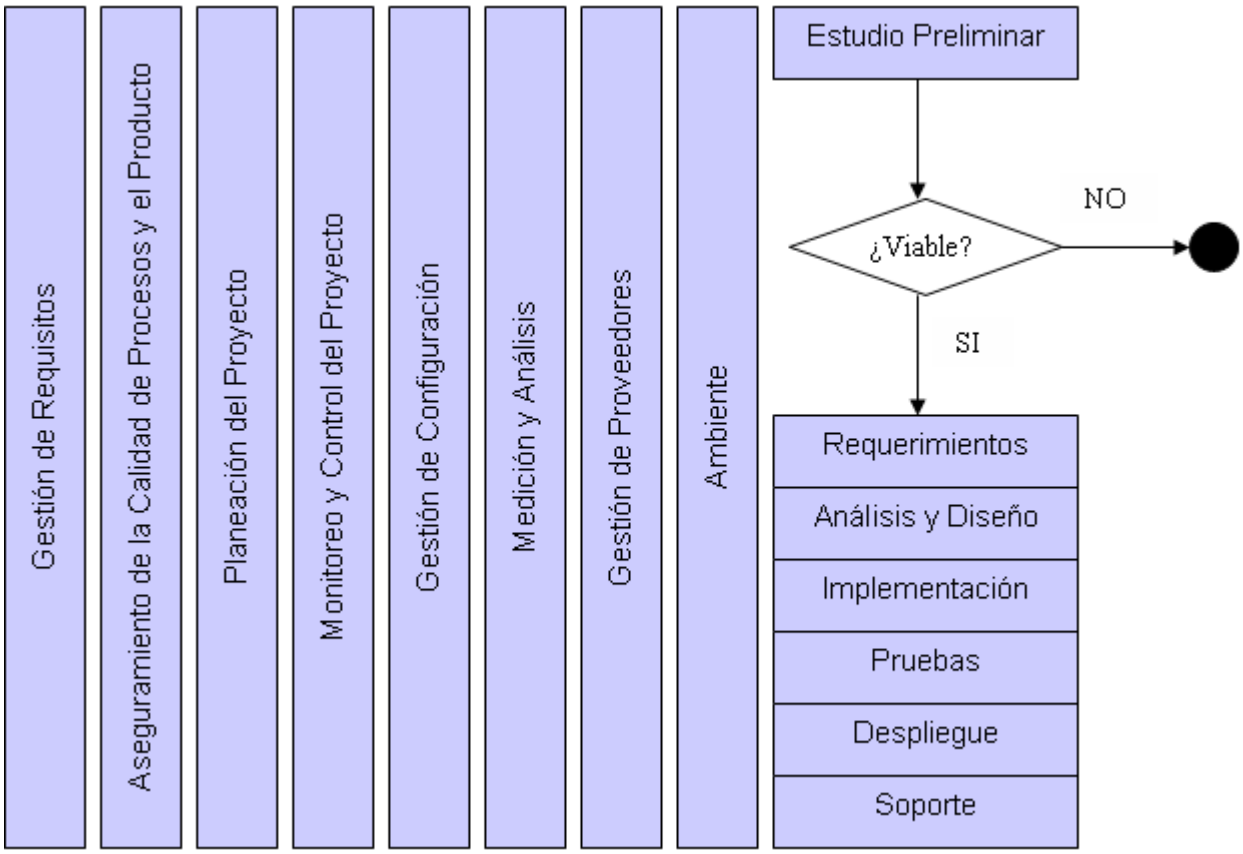


Figura 2-11: Representación de los flujos de procesos.

Se considera que se deben incorporar paralelo al ciclo de vida los flujos de trabajo de las categorías de gestión de proyecto, soporte y ingeniería que conforman el nivel 2 de CMMI que se están definiendo como parte del proceso de mejoras a realizar en la UCI, ellos son: Gestión de Requisitos, Aseguramiento de la Calidad de Procesos y el Producto, Planeación del Proyecto, Monitoreo y Control del Proyecto, Gestión de Configuración, Medición y Análisis, Gestión de Proveedores, los cuales no se detallan en este epígrafe pues estos se incluyen dentro de los libros de proceso que emite la Dirección de Calidad y el de Ambiente que propone RUP.

A continuación se describen las principales actividades, los roles y los artefactos a desarrollar en cada uno de los flujos, el resto de los elementos se describen en los anexos que se referencian. Los artefactos y las listas de chequeo corresponden con el expediente de proyecto y las del laboratorio de prueba que establece la Dirección de Calidad de la UCI respectivamente.

2.8.1 Estudio Preliminar

En el flujo el cliente solicita el desarrollo de una aplicación, en esta etapa se forma un pequeño equipo de un Líder de proyecto, un arquitecto y un analista para que realicen un levantamiento inicial de las necesidades del cliente y evalúen la viabilidad económica.

Las actividades más importantes a realizar en este flujo son:

- **Analizar las necesidades del cliente,** incluye encontrar los términos comunes, determinar las fuentes de información, recoger información sobre el escenario a desarrollar, describir los elementos del escenario y la relación con el actor y evaluar los resultados para determinar si es viable o no ejecutar el proyecto, además si no se reúne la información necesaria se puede repetir estas tareas. Los artefactos que se obtiene en esta actividad es las primeras versiones del informe y proyecto técnico.
- **Entender las necesidades del cliente,** incluye identificar las demandas del cliente, desarrollar las reglas de comportamiento de los elementos del escenario, determinar áreas a priorizar dentro del escenario o escena, manejar las dependencias entre los elementos, por último se deben evaluar los resultados para determinar si es viable o no ejecutar el proyecto, además si no se reúne la información necesaria se puede repetir estas tareas. Como resultado se refina el informe y proyecto técnico.
- **Supervisar el alcance del proyecto**, incluye definir el contexto inicial del proyecto, analizar la arquitectura, analizar las tecnologías, decidir la estrategia para el desarrollo del proyecto, desarrollar documento visión y evaluar los resultados para determinar si es viable o no ejecutar el proyecto, además si no se reúne la información necesaria se puede repetir estas tareas. Como resultado se refina el informe y proyecto técnico y se obtiene las primeras versiones del documento visión.

En los Anexos 12 y 13 se puede observar la descripción gráfica y textual de este flujo.

2.8.2 Requisitos

En este flujo el equipo crece en analista y diseñadores gráficos en dependencia del alcance del proyecto y su complejidad. El mayor peso de las actividades se centra en los encuentros entre los analistas y diseñadores gráficos con los clientes para hacer un levantamiento de los requisitos tanto funcionales como no funcionales y validación de las historias de usuario,

prototipos y requerimientos. El cierre de este flujo es la revisión técnica de los requisitos para su verificación.

Las actividades más importantes a realizar en este flujo son:

- **Desarrollar el diseño del escenario**, está actividad se ejecuta para cada escenarios que demande la aplicación e incluye recoger información, describir las características de los usuarios, identificar los elementos de las escenas, definir la estructura de las escenas para conformar el escenario, diseñar los detalles de los elementos de las escenas, diseñar e implementar los prototipos de los escenarios, luego se realizan talleres para validar los prototipos si es necesario se refinan hasta estar aprobados por el cliente. Como resultado de esta actividad se obtiene el prototipo que se contempla dentro de la historia de usuario.
- **Desarrollar las historias de usuario**, está actividad se ejecuta para cada escenarios que demande la aplicación, se debe ejecutar en paralelo con la anterior e incluye recoger información, describir relación entre actores y el escenario, revisar y refinar los escenarios, detallar el flujo de eventos, describir precondiciones y poscondiciones, describir los puntos de extensión, luego se realizan talleres para validar requisitos si es necesario se refinan hasta estar aprobados por el cliente, en la medida que se aprueban se priorizan los escenarios. Como resultado de esta actividad se obtiene la historia de usuario.
- **Describir los requisitos,** incluye detallar los requisitos, identificar requerimientos comunes, organizar las historias de usuarios en paquetes, luego se realizan talleres para validar los prototipos si es necesario se refinan hasta estar aprobados por el cliente. Como resultado de esta actividad se obtiene la especificación de requisitos.
- **Revisar técnicamente los requisitos**, incluye desarrollar reunión de revisión de los requisitos, preparar registro de revisión y defectos del documento y asignar responsabilidades para la solución de los defectos. Como resultado de esta actividad se obtiene la lista de no conformidades.
- **Desarrollar documento visión**, incluye definir las interfaces del sistema, identificar las restricciones que se imponen en el sistema, formular declaraciones del problema, definir características básicas del sistema, además se debe evaluar los resultados para determinar la magnitud del proyecto y la complejidad si es necesario se refina el documento visión. Como resultado de esta actividad se obtiene una nueva versión del documento visión.

En los Anexos 14 y 15 se puede observar la descripción gráfica y textual de este flujo.

2.8.3 Arquitectura y Diseño

Durante este flujo se adapta la arquitectura y se realiza el diseño para ser utilizado en la implementación. El arquitecto es el responsable de definir la arquitectura y decidir que componentes reutilizables pueden ser utilizados. Los diseñadores son los que más nutren el equipo y los que más peso tienen en esta etapa se encargan de realizar el diseño de clases, escenarios y escenas, y el de base datos de realizar el diseño de la BD.

Las actividades más representativas de este flujo son:

- **Analizar la arquitectura**, a partir del análisis de la DSSA, desarrollar vista general de la arquitectura, estudiar recursos disponibles, definir la organización de los escenarios y las escenas, identificar interacciones entre escenas y escenarios, desarrollar vista general de despliegue, identificar mecanismos de diseño y revisar los resultados. Está actividad tiene como resultado la descripción de la arquitectura y el modelo de despliegue.
- **Identificar elementos del diseño**, incluye identificar escenas, escenarios y clases, identificar interfaces entre escenas y escenarios y revisar los resultados. Esta actividad obtiene el modelo de diseño.
- **Diseñar escenario**, está actividad se ejecuta para cada escenario de la aplicación e incluye distribuir la conducta de las escenas, documentar las escenas, describir las dependencias entre escenas y revisar los resultados. Se obtiene como resultado una nueva versión del modelo de diseño.
- **Desarrollar el modelo de diseño**, está actividad se ejecuta para cada paquete de escenarios e incluye diseñar componentes, crear diseño de clases inicial, identificar las clases persistentes, definir la visibilidad de las clases, definir operaciones, definir los métodos, definir estados, definir atributos, definir dependencias, definir asociaciones, definir generalizaciones, definir la estructura interna, resolver las colisiones, tratar los requisitos no funcionales en general, revisar los resultados. Se obtiene como resultado el modelo de diseño.
- **Incorporar elementos del diseño existentes**, incluye identificar oportunidades de re-uso, actualizar la organización del modelo de diseño y revisar los resultados, se obtiene una nueva versión de modelo de diseño.

- **Diseñar la Base de Datos**, está actividad se ejecuta en paralelo a desarrollar el modelo de diseño e incluye Desarrollar el modelo de datos lógicos, desarrollar el diseño físico de la base de datos, definir el dominio, crear elementos iniciales del diseño físico de la base de datos, definir las tablas de referencia, crear llave primaria y las restricciones, definir datos y entrada en vigor de las reglas, normalizar el diseño de la base de datos, optimizar el rendimiento, optimizar el acceso a datos, definir las características de almacenamiento, diseñar los procedimientos para almacenar las clases de conducta en la base de datos y revisar resultados. Se obtiene como resultado el modelo de diseño completo.
- **Revisar la Arquitectura**, incluye desarrollar las reuniones de revisión de la arquitectura, preparar registro de revisión y defectos del documento, asignar responsabilidades para la solución de los defectos. Se ejecuta terminada la actividad de analizar la arquitectura y obtiene como resultado una lista de no conformidades.
- **Revisar el diseño**, incluye revisar cada elemento del diseño y revisar el cumplimiento de las pautas de diseño, preparar registro de revisión y de defectos del documento y asignar responsabilidades para la solución de los defectos. Se obtiene como resultado una lista de no conformidades.

En los Anexos 16 y 17 se puede observar la descripción gráfica y textual de este flujo.

2.8.4 Implementación

Este es el flujo que más trabajo tiene, pues es donde precisamente se construye el producto. El equipo crece en programadores, el arquitecto centra su trabajo en dividir el desarrollo en partes más manejables y distribuir su construcción entre los programadores. Estas partes construidas son integradas por el arquitecto posteriormente. Se desarrolla por parte de los analistas los manuales de la aplicación.

Las actividades más importantes se describen a continuación:

- **Desarrollar el modelo de implementación**, incluye establecer la estructura del modelo de implementación, ajustar las escenas y escenarios, definir las importaciones para cada escena y escenarios de implementación, decidir el tratamiento de los ejecutables, actualizar la vista de implementación y evaluar los resultados. Como resultado se obtiene el modelo de implementación.
- **Realizar plan de integración**, incluye identificar escenarios y escenas, definir cambios de construcción, definir series de construcción, evaluar la integración del plan de

construcción y evaluar los resultados. Se obtiene como resultado una versión mejorada del modelo de implementación.

- **Implementar los elementos del diseño**, está actividad se ejecuta para cada paquete de implementación e incluye preparar la implementación, transformar diseño en implementación, completar la implementación, evaluar la implementación y proporcionar información de retorno al diseño. Se obtiene como resultado el código fuente de cada elemento.
- **Integrar escenas**, se desarrolla para cada escenario e incluye integrar las escenas y entregar el escenario implementado. Se obtiene como resultado el código fuente de cada escenario.
- **Integrar escenarios**, incluye aceptar los escenarios, integrar los escenarios y promover la línea base de la arquitectura. e obtiene como resultado el código fuente de la aplicación.
- **Desarrollar los artefactos de la instalación**, incluye construir el material de apoyo de usuario. Se obtiene como resultado la BD.
- **Revisar el código fuente y manual de usuario**, incluye establecer puntos de control para la implementación, revisar las escenas y escenarios, preparar registro de revisión y de defectos del documento y asignar responsabilidades para la solución de los defectos. Se obtiene como salida de esta actividad la lista de no conformidades.

En los Anexos 18 y 19 se puede observar la descripción gráfica y textual de este flujo.

2.8.5 Prueba

En el flujo se desarrollan las pruebas del sistema permitiendo que este quede los más libremente posible libre de defectos. Los roles que más peso tienen en esta etapa son el diseñador de prueba y el probador.

Las actividades más representativas de este flujo son:

- **Desarrollar el plan de prueba**, incluye identificar los elementos a probar, formular la declaración de la misión de prueba e identificar los niveles, tipos y métodos de prueba. Se obtiene como resultado una primera versión del plan de pruebas.
- **Identificar los objetivos de la prueba**, incluye identificar los elementos dinámicos y eventos del sistema, los elementos de infraestructura de prueba, las necesidades del plan especificaciones y definir los requisitos y la infraestructura de la prueba, por último

se evalúan y verificar los resultados. Se obtiene como resultado una versión mejorada del plan de pruebas.

- **Identificar los mecanismo de prueba**, incluye examinar la arquitectura del software y designar sus ambientes, identificar y definir los mecanismos de prueba a usar y evaluar y verificar los resultados. Se obtiene como resultado una versión mejorada del plan de pruebas.
- **Diseñar las pruebas**, incluye identificar las condiciones de la entrada, rendimiento y ejecución, los puntos de observación, puntos de control y los métodos apropiados, definir fuentes de datos requeridas, valores y rangos, asignar los recursos suficientes para realizar las pruebas, mantener las relaciones de trazabilidad y evaluar y verificar los resultados. Se obtiene como resultado el diseño de prueba.
- **Ejecutar las pruebas**, incluye seleccionar la técnica de implementación apropiada, preparar las condiciones previas de ambiente de prueba, implementar la prueba, establecer los juegos de datos externos, verificar la implementación de la prueba, restaurar el ambiente de prueba en un estado conocido, mantener las relaciones de trazabilidad y evaluar y verificar los resultados. Se obtiene como resultado la lista de no conformidades.
- **Evaluar las prueba**, incluye examinar todas las incidencias de las pruebas, crear las solicitudes de cambio, analizar y evaluar el estado de la aplicación, hacer una valoración de la experiencia del desarrollo, hacer una valoración de riesgos técnicos y de los resultados de las pruebas y evaluar y verificar sus resultados. Se obtiene como resultado una minuta de la reunión y las solicitudes de cambio.

En los Anexos 20 y 21 se puede observar la descripción gráfica y textual de este flujo.

2.8.6 Despliegue

En este flujo se despliega y prueba el producto en su entorno de ejecución final, en caso de detectar inconformidades con el mismo se retorna para su corrección. En caso contrario, el cliente firma el acta de aceptación, expresa su conformidad con el producto y se cierra el contrato.

Las actividades más representativas de este flujo son:

- **Desarrollar el plan de instalación**, incluye planificar cómo empaquetar el software, cómo distribuir el software, cómo instalar el software y proporcionar ayuda y asistencia a los usuarios. Estos resultados se documentan en el plan de despliegue.

- **Desarrollar los materiales de instalación**, incluye desarrollar los *scripts* de instalación, los archivos de configuración, las instrucciones de instalación, un esquema de los materiales de formación, escribir los materiales de formación, producir un plan de alto nivel para la información que se va a presentar y revisar los resultados. Como resultado se obtienen los materiales de apoyo.
- **Ejecutar la instalación**, incluye ejecutar la instalación, preparar el ambiente para las pruebas de aceptación, realizar las pruebas de aceptación, preparar registro de revisión y de defectos del documento, asignar responsabilidades para la solución de los defectos. Como resultado se obtiene una lista de no conformidades y/o el acta de aceptación.

En los Anexos 22 y 23 se puede observar la descripción gráfica y textual de este flujo.

2.8.7 Soporte

En el flujo se brindan los servicios de mantenimiento de la aplicación posventa. Estos servicios se pueden brindar a través de una base de conocimiento pública, un centro de llamadas o dándole asistencia en las instalaciones del cliente directamente.

Las actividades más representativas de este flujo son:

- **Preparar Ambiente de Soporte**, incluye preparar la base de conocimiento para el soporte, especialistas para brindar el soporte por el centro de llamadas y el ambiente para recibir solicitudes de soporte en el cliente.
- **Responder solicitud de soporte**, incluye evaluar solicitud de soporte, responder solicitud, recoger errores o inconformidades de los usuarios con el producto y actualizar la base de conocimiento.

En los Anexos 24 y 25 se puede observar la descripción gráfica y textual de este flujo.

2.8.8 Ambiente

Es un flujo que no forma parte del ciclo de vida, se considera de apoyo y enfoca las actividades necesarias para configurar el proceso para un proyecto. Describe las actividades que se requieren para desarrollar las pautas que soportan el proyecto. El propósito de las actividades de ambiente es proporcionar a la organización el ambiente de desarrollo de software definiendo los procesos y las herramientas que soportaran el resto de los flujos para la ejecución satisfactoria de todas las actividades.

Las actividades más representativas de este flujo son:

- **Configurar el proceso**, incluye analizar el proyecto, configurar los procesos, preparar los procesos para el proyecto y preparar materiales de capacitación. Estos resultados se describen el documento ambiente de desarrollo.
- **Definir las plantillas del proyecto**, incluye preparar las plantillas y los materiales de capacitación, esto refina el documento ambiente de desarrollo.
- **Preparar el entorno para el proyecto**, incluye instalar las herramientas en el servidor, personalizar las herramientas en el servidor, integrar las herramientas e instalar y personalizar las herramientas para los clientes
- **Preparar el personal**, incluye capacitar a los miembros del proyecto e informar los cambios del proceso y las herramientas.

En los Anexos 26 y 27 se puede observar la descripción gráfica y textual de este flujo.

Las personas involucradas en los procesos forman un factor importante para el éxito de un proyecto por lo que su organización es fundamental. Teniendo en cuenta esto cada persona implicada ocupa uno o varios roles determinado en dependencia de sus conocimientos, habilidades y valores. La definición de los roles y las responsabilidades y la estructura organizacional propuesta para la ejecución de las actividades se muestran en las tablas de los Anexos 28 y 29.

Los artefactos de entrada y salida se corresponden con los del expediente de proyecto establecido por la Dirección de Calidad de la UCI, la definición de los mismos se encuentran en el Anexo 30 y su relación con el expediente de proyecto en el Anexo 31.

2.9 Consideraciones finales

En el presente Capítulo se describió la necesidad de crear una DSSA para diseñar e implantar una línea para el desarrollo de Aplicaciones Enriquecidas de Internet que incluyan escenarios tridimensionales interactivos en el polo de Realidad Virtual de la UCI. Se describieron los métodos, procedimientos y técnicas utilizadas para llevar a cabo la investigación.

Se llegó a la definición del dominio en que se enmarca la DSSA, los requerimientos de referencias identificados en este dominio. Además, se expuso la propuesta de arquitectura de referencia, permitiendo de esta forma realizar el diseño de las capas lógicas de una aplicación guiados por la propuesta. Se ha explicado la propuesta de convenciones de nombres o estándares de codificación tanto para códigos fuentes y recursos, las estructuras de código

según la clasificación de la complejidad que presentan las aplicaciones y los mecanismos de colaboración entre las distintas divisiones organizacionales propuestas.

Se presentó un proceso de desarrollo de software que define y organiza los principales flujos de trabajo. Se describe la organización estructural de las personas, la definición de roles y responsabilidades.

La DSSA y el proceso de desarrollo de software que se describe en el capítulo desde el punto de vista teórico instrumenta los principios de la industrialización de la producción a través de la definición y establecimiento de todos sus procesos, conllevando a concebir altos niveles de reutilización y productividad, acortando el tiempo de desarrollo de las producciones, garantizando el uso de buenas prácticas para la IS, realizando procesos organizados, estructurado y disciplinado, donde la organización tenga capacidad para repetirlos y mejorarlos.

CAPÍTULO 3: IMPLANTACIÓN Y CASO DE ESTUDIO

3.1 Introducción

En el presente se describe el proceso de implantación y un caso de estudio. La implantación describe las 10 prácticas propuestas por CMMI para la institucionalización de las áreas de procesos y el caso de estudio ejecuta los flujos de trabajo de Requerimiento, Arquitectura y Diseño, Implantación y Prueba.

3.2 Elementos de implantación de la DSSA

Definir el proceso de implantación implica poner al proyecto en posición para realizar su misión con todos los elementos que conforman la DSSA y en condiciones de realizar su trabajo de manera eficaz y eficiente. Para implantar la propuesta se necesita realizar una serie de actividades y sobre todo debe existir motivación por parte de los integrantes. Estas actividades necesitan el asesoramiento de una persona preparada y que tenga dominio de la propuesta.

Para la realización de la misma se hizo referencia al actual proceso de mejora CMMI el cual presenta una serie de prácticas genéricas las cuales servirán de base para realizar diferentes actividades, las mismas se detallarán a continuación.

3.2.1 Establecer políticas de organización

Se debe establecer y mantener una política de la organización para garantizar la planificación y realización del proceso. Los directivos de la línea son los responsables de constituir y comunicar a los integrantes los principios, la dirección, y las expectativas de la línea de producción

3.2.2 Planeación del proceso

Se debe realizar un plan para llevar a cabo de implantación, determinando todo lo que se necesita para ejecutarlo y alcanzar los objetivos establecidos, está actividad es responsabilidad de los directivos de la línea.

3.2.3 Proveer recursos

Se debe asegurar que todos los recursos necesarios para llevar a cabo el proceso estén disponibles en todo momento incluyendo el personal calificado, las herramientas propuestas y las instalaciones físicas adecuadas.

3.2.4 Asignar responsabilidades

Se debe asignar a cada persona del equipo de desarrollo del proyecto uno o más roles y sus responsabilidad.

3.2.5 Entrenamiento y capacitación del personal

Con el objetivo de capacitar a los integrantes del proyecto para enfrentar la manera de producir se deben realizar talleres para introducir los elementos de la DSSA y crear una conciencia referente a la importancia de su aplicación. Se deben efectuar cursos orientados al trabajo con las herramientas y de esta forma garantizar que los mismos puedan desarrollar su trabajo eficazmente.

3.2.6 Configuración

Se debe incorporar las buenas prácticas del área de proceso de CMMI que está definiéndose como parte del proceso de mejora para mantener la integridad de los datos y del producto. Se van a establecer diferentes niveles de control, se contará con políticas de seguridad de la información y con un repositorio donde se localizará la última versión del producto de trabajo en uso, al igual que otros documentos de interés para el proyecto.

3.2.7 Identificar involucrados relevantes

Se deben definir los interesados durante la ejecución del proceso de implantación, estos se mantendrán motivados y comprometidos.

3.2.8 Monitoreo y control del proceso

Se debe realizar un seguimiento continuo del desempeño laboral de cada uno de los miembros del equipo y también se debe efectuar un seguimiento preciso de cada elemento de la DSSA. Se controlará el proceso de acuerdo al plan trazado, para lograr esto se establecerá una

vigilancia diaria y una supervisión estricta del mismo, permitiendo tomar medidas cuando sea necesario. Se debe contar con personas dentro de la línea pero ajenas al equipo de desarrollo, que certifiquen que el proceso se realizó según lo previsto y de acuerdo a las normas y procesos descritos.

3.2.9 Retroalimentar a los directivos

Se deben mantener informados a los directivos de lo que se está realizando en cada momento y se consultarán en caso de ser necesario.

3.2.10 Evaluar resultados

Se debe realizar una evaluación de la implantación de la DSSA, lo cual valdrá para conocer los procesos que hay que perfeccionar y las mejoras que se obtuvieron. Esto servirá de base para determinar la viabilidad práctica y mejorar la propuesta.

3.3 Caso de estudio

En vista de lograr un mejor entendimiento de la propuesta de DSSA y del proceso de desarrollo, se desarrolla el caso de estudio de baja complejidad para describir los principales elementos que lo componen. El mismo está compuesto por varios flujos, en este ejemplo debido al bajo nivel de complejidad se ejecutarán solo Requerimiento, Arquitectura y Diseño, Implementación y Pruebas; estos representan los flujos de trabajo de la ingeniería más importantes y generan los artefactos más significativos, que son almacenados en los documentos del expediente de proyecto establecido por la Dirección de Calidad de Software de la UCI. Estos documentos pueden ser consultados los anexos correspondientes.

3.3.1 Enunciado del caso de estudio

Una tienda virtual tiene la posibilidad de mostrar un grupo de entidades, de las cuales se brinda la vista 3D y una descripción correspondiente. Si el usuario lo desea, la entidad seleccionada puede incorporarse a un pedido. Terminado el pedido, el sistema ofrece la opción de enviar la factura, pidiendo los datos del usuario para realizar esta operación.

3.3.2 Resultados del Caso de Estudio

Para el desarrollo del piloto se ejecutaron todas las actividades de los flujos de trabajo Requerimiento, Arquitectura y Diseño, Implementación y Prueba. El equipo de desarrollo estuvo compuesto por una persona que asumió los diferentes roles durante el proceso.

Los principales resultados obtenidos en las diferentes etapas y reflejados en los artefactos se mencionan a continuación:

Requerimiento

Se generaron dos documentos Especificación de requisitos e Historias de usuario, los mismos se pueden ver en el Anexo 32 y 33 respectivamente.

Requerimientos funcionales

1. Mostrar entidades

   Se muestran las entidades en una vista 3D.

2. Mostrar descripción

   Se muestra la descripción de la entidad seleccionada por el cliente, de la cual se brindan los siguientes datos: Nombre, Descripción, Tamaño, Cantidad a solicitar.

3. Interactuar con la vista en 3D

   Se muestra la vista 3D, permite al cliente la interacción a través del ratón.

4. Gestionar pedido

   Gestionar pedido incluye:

   - Insertar nuevo pedido
   - Eliminar pedido

   De los pedidos se registran los siguientes datos: Nombre, Cantidad a solicitar, Tamaño, Tamaño total del pedido.

5. Mostrar factura

Se muestra la factura del pedido realizado por el cliente, se muestran los datos del pedido y se solicitan los siguientes datos: Nombre del cliente, Apellidos del cliente y Código de la Tarjeta de Crédito.

6. Enviar Factura

   Termina la aplicación.

Se considero que solo debía tener una Historia de Usuario: Realizar pedido donde se capturaran todas las funcionalidades. En la tabla 3-1 se muestra la especificación de la Historia de Usuario.

Historia de Usuario	
Número: 1	**Usuario: Cliente**
Nombre historia : Realizar pedido	
Prioridad en negocio: crítico	**Riesgo en desarrollo: alto**
Puntos estimados: 5	**Iteración asignada: 1**
Programador responsable: Gilberto Cao Tarrero	
Descripción: El sistema muestra al cliente tres vistas en la interfaz. La vista 1, Vista 3D: muestra una vista 3D de la entidad seleccionada. Al cliente navegar por las entidades, se activa la vista 2. La vista 2, Detalles: muestra los detalles de la entidad seleccionada por el cliente, con los datos: Nombre, Descripción, Tamaño, y el dato Cantidad a solicitar, con el valor 1 por defecto. El cliente puede modificar la Cantidad a solicitar y añadir una línea de	

pedido, que se registraría en la vista 3. La vista 3, Pedido: muestra el pedido, con los siguientes datos: Nombre, Cantidad a solicitar, Tamaño, Tamaño total del pedido. En la medida que se solicite añadir entidades en la vista tres, se incorporaran elementos a este pedido. El cliente tiene la opción de borrar una línea de pedidos de la lista de pedidos y se actualiza esta y el tamaño total se recalcula, o el cliente puede solicitar la factura. Si el cliente selecciona la factura, se muestra una nueva interfaz Datos del Envío solicitando los datos para llenar la factura: Nombre del cliente, Apellidos del cliente, Código de la Tarjeta de Crédito. Se muestran los datos del pedido y se da la opción de volver para modificar la lista de pedidos o enviar la factura.
Observaciones: La funcionalidad que se describe es el núcleo de la tienda virtual.

Tabla 3-1 Historia de Usuario Realizar Pedido.

Se desarrollaron además los prototipos de interfaz grafica que se muestran en las figuras 3-1y 3-2.

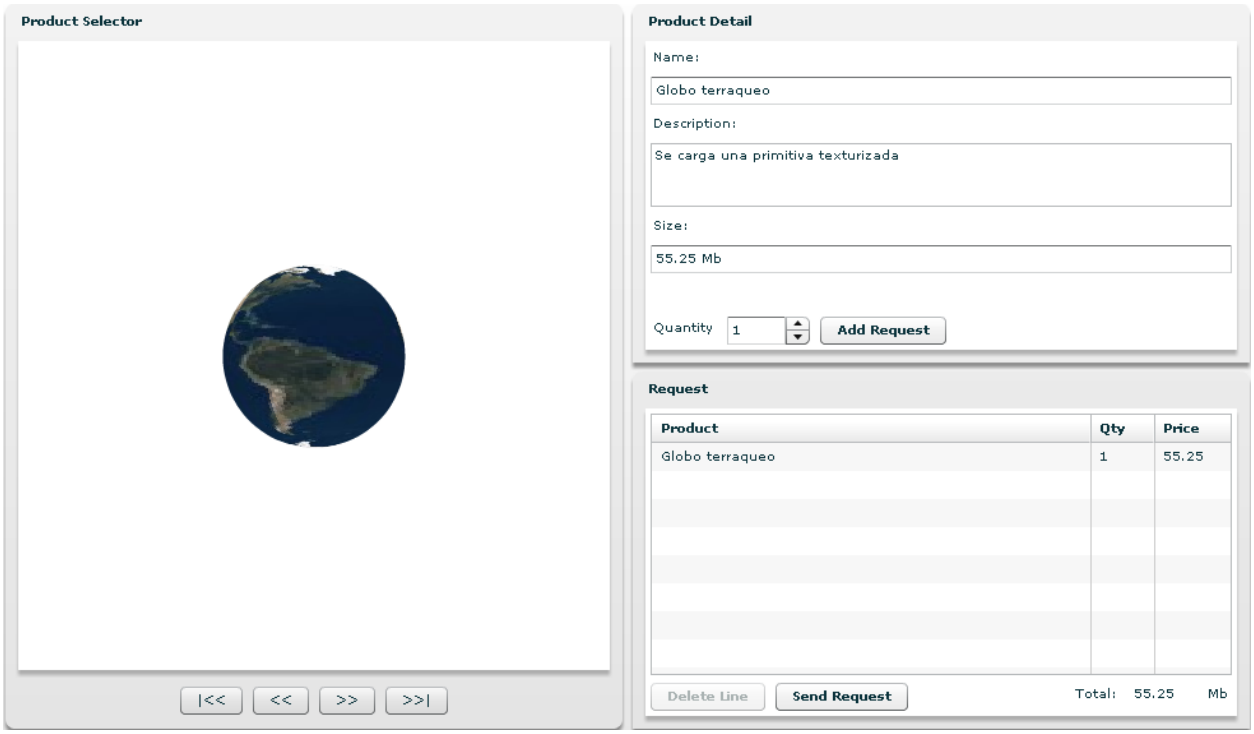


Figura 3-1 . Interfaz gráfica inicial.

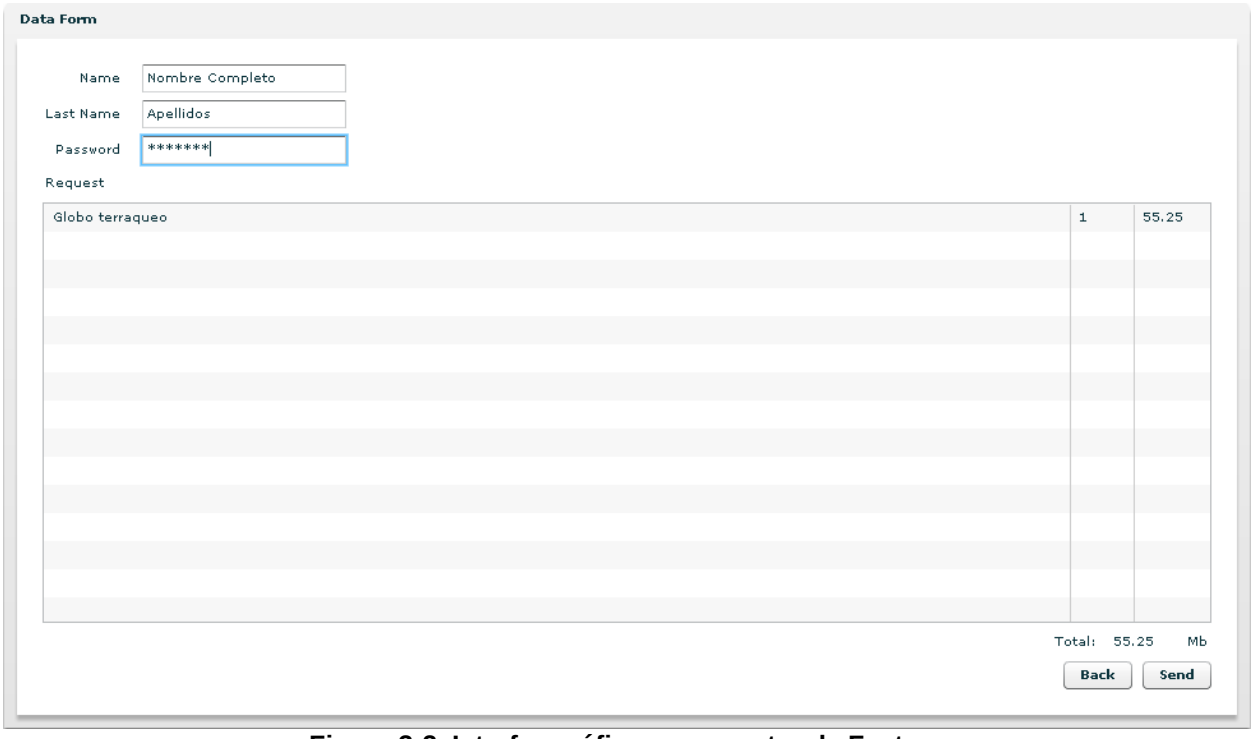


Figura 3-2. Interfaz gráfica para mostrar la Factura

Arquitectura y Diseño

Se generaron dos documentos Arquitectura de Software y Modelo de Diseño los mismos se pueden ver en el Anexo 34 y 35 respectivamente. .

En el documento de Arquitectura de Software se mencionaron las herramientas que conforman el Ambiente de Desarrollo de la propuesta y los demás elementos se referencia la definición de la arquitectura de software de dominio específico: Arquitectura para el desarrollo de Aplicaciones Enriquecidas de Internet que incluyan escenarios tridimensionales interactivos.

Se desarrollo la vista de despliegue con los procesadores

Cliente: tiene la responsabilidad de ejecutar la capa lógica de presentación.

Servidor: tiene la responsabilidad de ejecutar las capas lógicas de servicios de negocio y acceso a datos.

Base de Datos: tiene la responsabilidad de ejecutar el gestor de base de datos.

Quedando como se refleja en la figura 3-3.

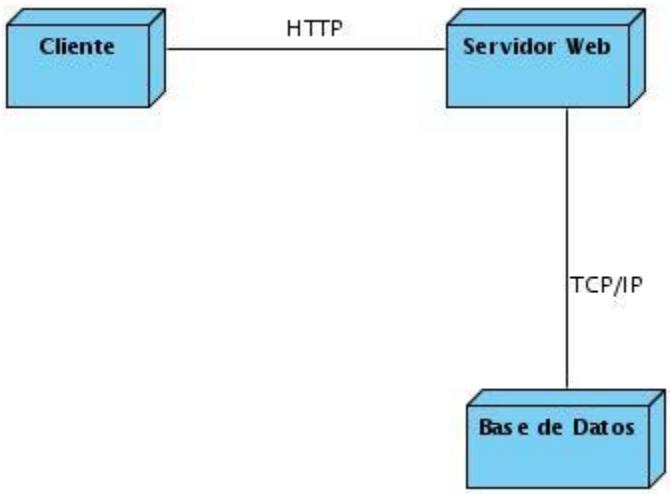


Figura 3-3 Diagrama de Despliegue.

Posteriormente se desarrollo el modelo de diseño para el cual se desarrollo los diagramas de clases de las capas lógicas de la DSSA propuesta las cuales se muestran en las siguientes figuras.

Capa de Acceso a Datos

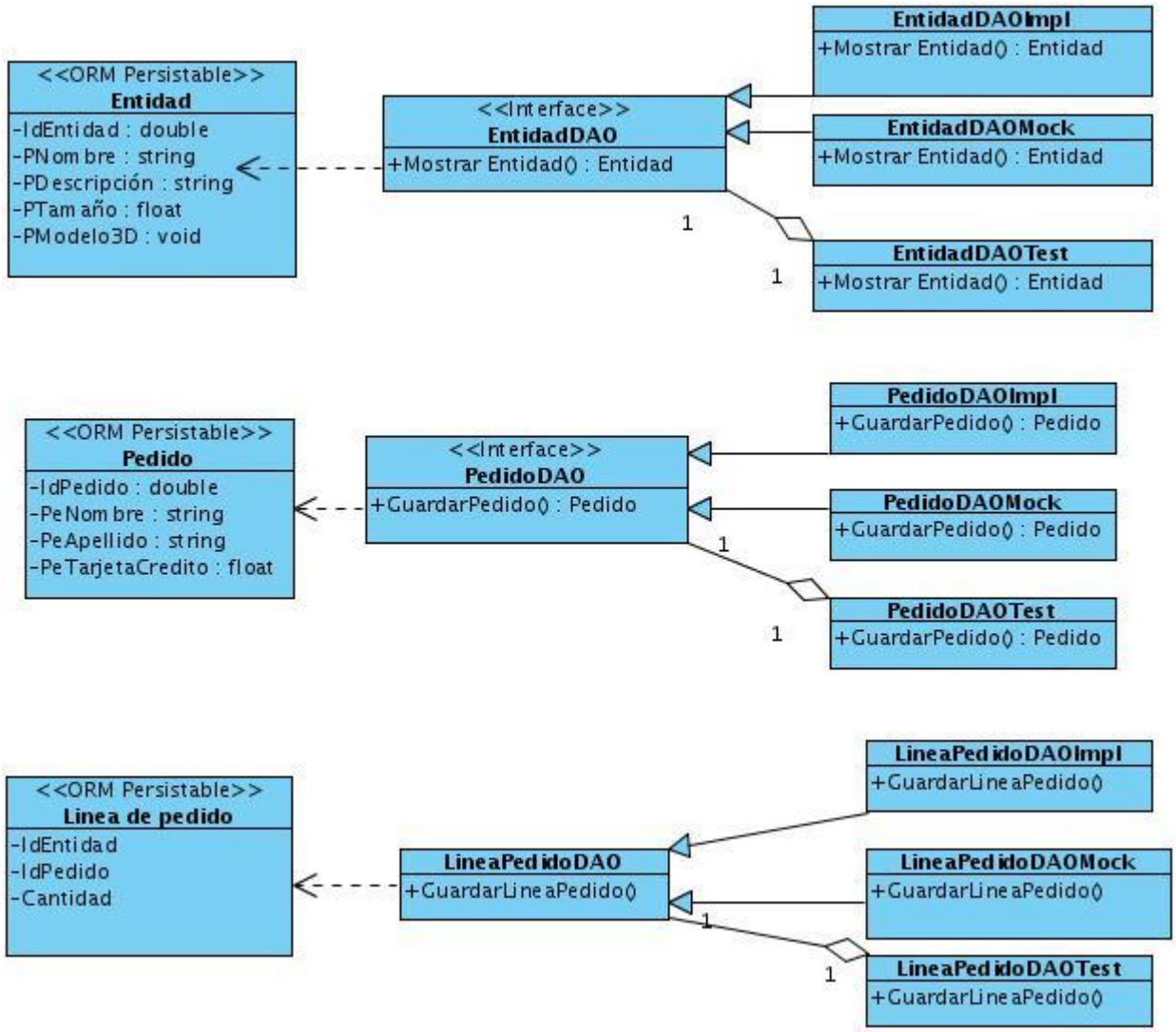


Figura 3-4 Diagrama de Clases de la Capa de Acceso a Datos.

Capa de Servicio de Negocio

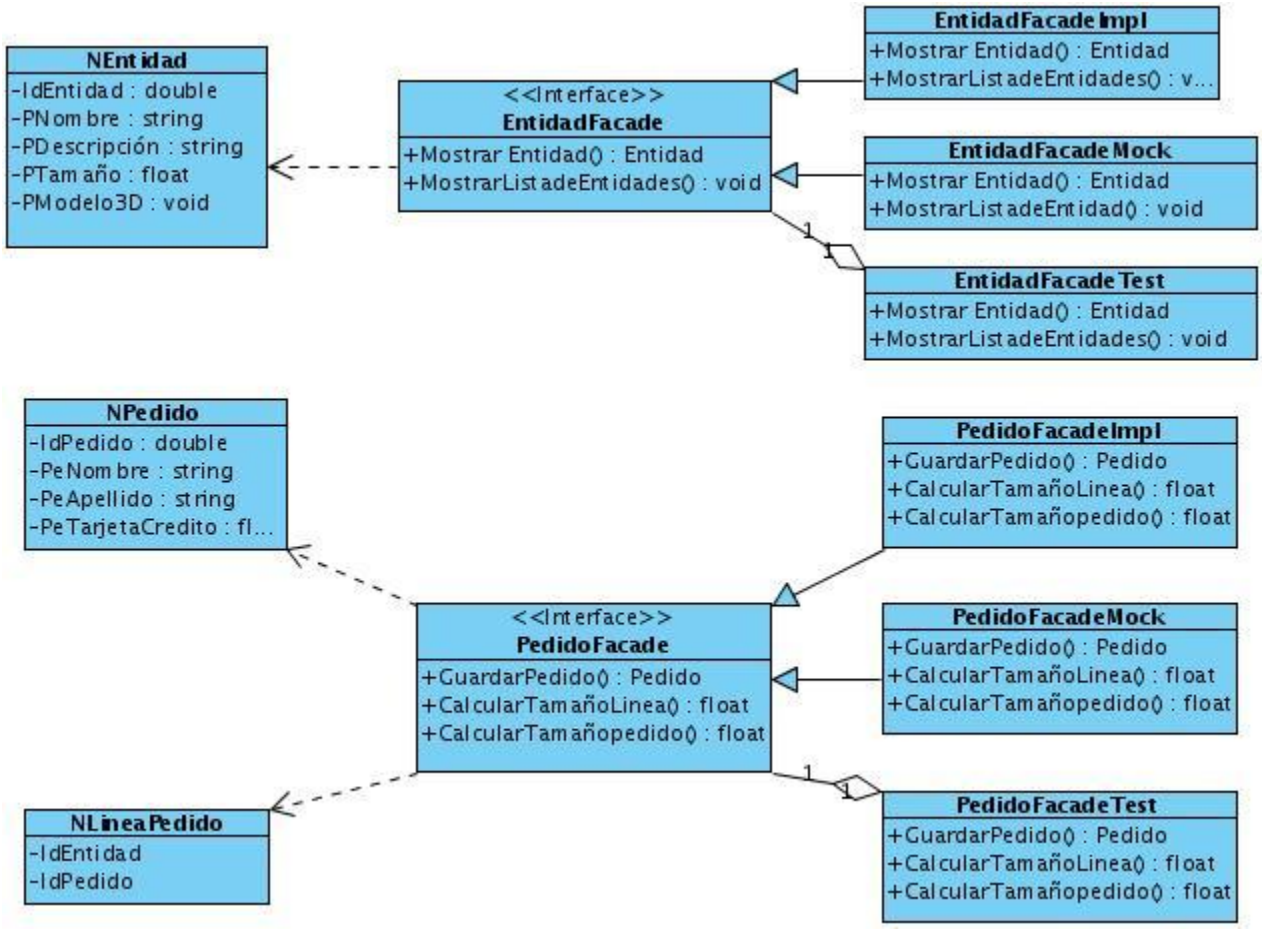


Figura 3-5 Diagrama de Clase de la Capa de Servicio de Negocio

Capa de Presentación

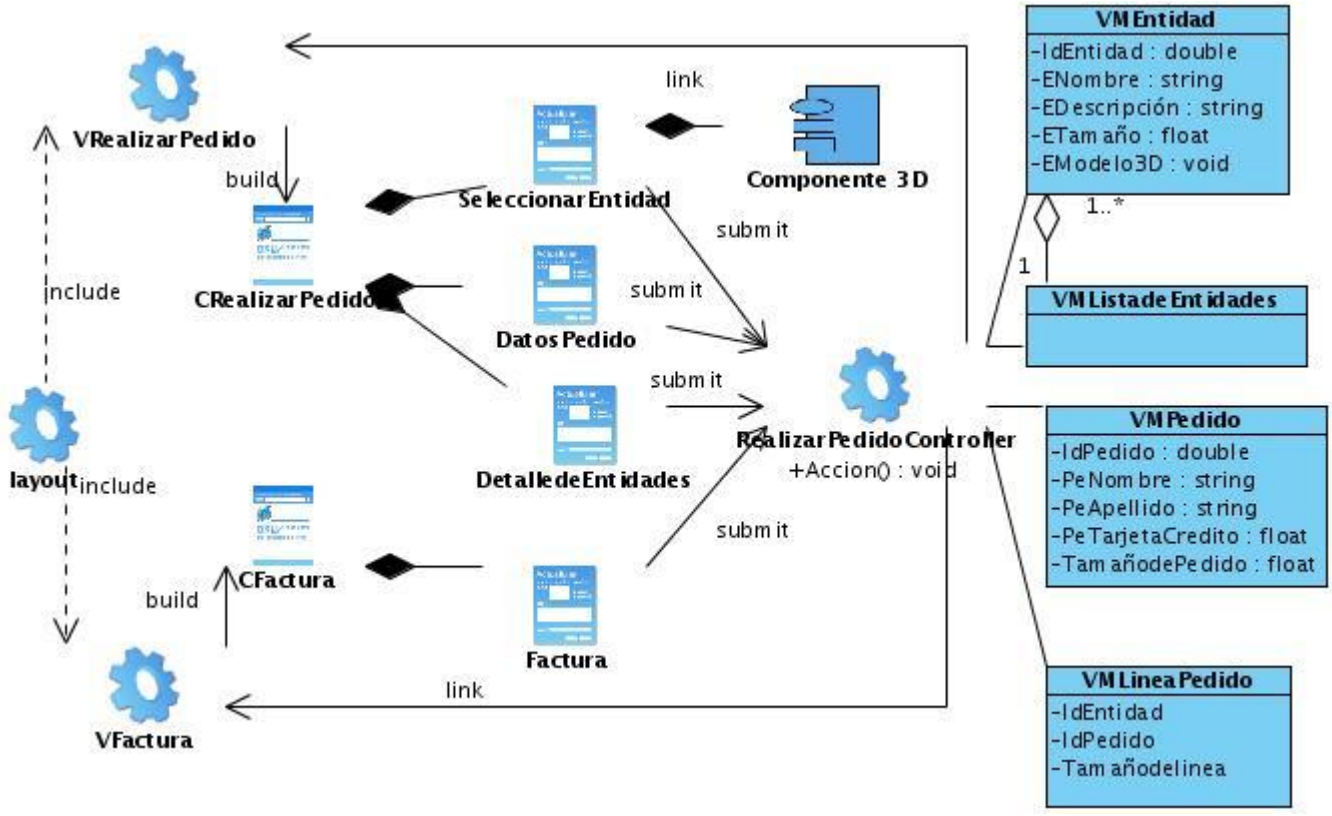


Figura 3-6 Diagrama de Clases de la Capa de Presentación.

Posteriormente se desarrollo el Diseño de la Base de Datos que se refleja en el documento de Arquitectura de Software.

La base de datos va a estar formada por tres tablas: Entidad, Pedido y EntidadPedidos las cuales tienen las características que se muestran en la tabla 3-2.

Entidad	Llave primaria	Número de Atributos
Entidad	Identidad	5
Pedido	Impedido	4
Entidadpedidos	IdPedido, IdEntidad	3

Tabla 3-2 Descripción de las tablas de la Base de Datos.

El Diagrama de Entidad Relación que se obtuvo se muestra en la figura 3-7.

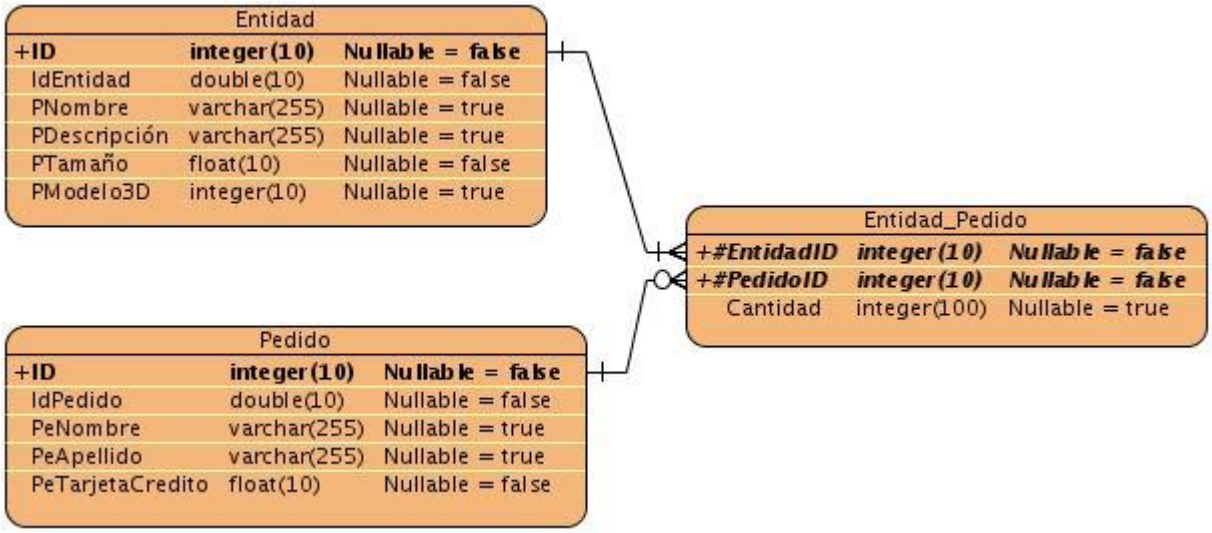


Figura 3-7 Diagrama Entidad Relación.

Cada una de las tablas posee los datos que se muestran en las siguientes tablas.

Nombre: Entidad Descripción: Recopila los datos relacionados con la descripción de las entidades disponibles en la tienda virtual		
Campo	Tipo	Descripción
IdEntidad	INTERGER [5]	Este campo almacena el código de identificación de la entidad
PNombre	VARCHAR [20]	Este campo almacena el nombre de la entidad
PDescripción	VARCHAR[60]	Este campo almacena la descripción de la entidad
PTamaño	FLOAT[10]	Este campo almacena el Tamaño de la entidad
PModelo3D	BIT	Este campo almacena el modelo en 3D de la entidad

Tabla 3-3 Descripción de la tabla Entidad de la Base de Datos

Nombre: Pedido		
Descripción: Recopila los datos relacionados con la factura		
Campo	Tipo	Descripción
IdPedido	INTERGER [5]	Este campo almacena el código de identificación de la entidad
PeNombre	VARCHAR [20]	Este campo almacena el nombre del usuario
PeApellidos	VARCHAR[60]	Este campo almacena los apellidos del usuario
PeContraseña	INTERGER[16]	Este campo almacena la contraseña del usuario

Tabla 3-4 Descripción de la tabla Pedido de la Base de Datos

Nombre: EntidadPedidos		
Descripción: Recopila los datos relacionados con las entidades solicitadas en los pedidos		
Campo	Tipo	Descripción
Impedido	CHAR[5]	Este campo almacena el código del pedido
IdEntidad	CHAR[5]	Este campo almacena el código de identificación de la entidad
EPCantidad	NUMERIC(8, 0)	Este campo almacena la cantidad de entidades solicitadas en el pedido

Tabla 3-5 Descripción de la tabla EntidadPedidos de la Base de Datos

Implementación

En el flujo de implementación se transformó el diseño en implementación y se construyo el material de apoyo de usuario obteniéndose como artefactos más importantes el código de la aplicación y el Manual de Usuario el cual se muestra en el Anexo 36.

Prueba

En el flujo de prueba se realizaron las actividades de revisión del código y los artefactos generados en los flujos de trabajo. En las actividades de este flujo de trabajo se desarrollaron Plan de pruebas y Diseño casos de prueba los cuales se muestran en el Anexo 37 y 38.

Se desarrollaron dos Iteraciones y una revisión de seguimiento quedando el sistema libre de no conformidades en la segunda Iteración. Se definió el entorno de prueba en función de los requisitos no funcionales.

El diseño de prueba se desarrollo a partir de la Historia de Usuario y abarco todas las funcionalidades. En la misma se comprobó que:

- El sistema inicie mostrando las tres vistas de la interfaz, en la vista 1 la Vista 3D de la entidad seleccionada, la vista 2 inicializada en la primera entidad de la lista (Nombre, Descripción, Tamaño) y la vista 3 en blanco (Nombre, Cantidad solicitadas, tamaño y tamaño total del pedido).
- El sistema muestra los datos de la entidad en la vista a 2 al navegar por las entidades de la vista 1.
- El sistema inicializa el dato de la Cantidad de unidades en uno 1.
- El sistema permita modificar la cantidad de unidades validando que siempre sea mayor

que 0, y un número entero.

- El sistema permite que el cliente puede interactuar con la vista en 3D.
- El sistema permite agregar una entidad al pedido y se refleja en la vista 3.
- El sistema muestra en la vista 3 las entidades que el cliente ha solicitado.
- El sistema permite eliminar línea de pedido, se actualiza el pedido y el tamaño total se recalcula.
- El sistema permite realizar el pedido siempre que tenga al menos una entidad solicitada.
- El sistema debe inhabilitar las opciones Borrar línea de pedido y realizar pedido cuando no hayan entidades solicitadas.
- El sistema muestra una nueva interfaz solicitando los datos de la factura Nombre del cliente, Apellidos del cliente, Contraseña.
- El sistema inhabilita la opción de enviar factura hasta que estén lleno los datos del cliente.

3.3.3 Consideraciones del Caso de Estudio

En el caso de estudio se desarrollaron la actividades del los flujos de proceso Requerimiento, Arquitectura y Diseño, Implementación y Prueba. Al basarse el desarrollo en el proceso definido con un orden lógico de las actividades, las tareas y los artefactos permitió que el desarrollador supiera que, cuando y como hacer para obtener la aplicación. La documentación obtenida es suficiente para describir el proceso de desarrollo y documentar el producto lo que facilita su liberación y aceptación por el cliente.

El uso de la DSSA disminuyó el tiempo de desarrollo pues solo fue necesario instanciar los elementos comprendidos en los requisitos de referencia. Se utilizaron los requisitos no funcionales, la arquitectura de las capas lógicas y la organización de las clases en ellas, la convención y los estándares de código y recursos, la estructura de organización y los mecanismo de colaboración estos elementos organizaron el ambiente de desarrollo y permitieron realizar un trabajo organizado y estructurado, lo que permite que la aplicación obtenida sea fiable y mantenible por otros desarrolladores.

La combinación del proceso y la DSSA propuesta permiten concebir altos niveles de reutilización y productividad, acortar el tiempo de desarrollo de las producciones y realizar un proceso guiado donde se sepa que hacer en cada una de las etapas.

3.4 Consideraciones finales

En el presente se definió la estrategia de implantación de la propuesta de DSSA con un conjunto de buenas prácticas a instrumentar y se desarrolló un caso de estudio de referencia para la aplicación de la DSSA y el proceso de desarrollo de software.

CONCLUSIONES

Se definió y desarrolló una arquitectura de software de dominio específico para la producción de Aplicaciones Enriquecidas de Internet que incluyan escenarios tridimensionales interactivos. Regido por modelos de proceso de desarrollo, metodologías, herramientas y buenas prácticas. Se definió el Dominio de Aplicación, los Requisitos de Referencia, una Arquitectura Base, el Flujo de Trabajo y las Herramientas del Ambiente de Desarrollo con el objetivo de:

- Concebir altos niveles de reutilización y productividad.
- Acortar el tiempo de desarrollo de las producciones.
- Realizar procesos organizados, estructurados y disciplinados, donde la organización tenga capacidad para repetirlos y mejorarlos.

Se definieron los elementos para la implantación y un caso de estudio para usar como referencia.

RECOMENDACIONES

Los objetivos del trabajo no abarcan todos los elementos a definir; los cuales son amplios y diversos. Por lo que se propone:

1. Definir posibles métricas y/o indicadores que se deberían tener en cuenta para garantizar el éxito o fracaso.
2. Realizar un estudio de factibilidad económica que cuantifique la factibilidad de la propuesta.
3. Aplicar, evaluar y revisar la implantación de la DSSA en diversos proyectos.
4. Desarrollar casos de estudio más complejos.

REFERENCIAS BIBLIOGRÁFICAS

ADOBE Flex Builder 3. Create engaging, cross-platform Rich Internet Applications, Adobe System Inc, 2007

DEEPAK ALUR, J. C., DAN MALKS, Core J2EE™ Patterns: Best Practices and Design Strategies, Second Edition, p. 0-13-142246-4, 2003

BANDINELLI, S. et al, Modeling and Improving an Industrial Software Process, IEEE Trans. Software Engineering, vol. 21, n.", pp. 440-454, Mayo 1995

BAUER, E. L., Software Engineering, Information Processing, 71, North Holland Publishing Co, Amsterdam, 1972

BECK, K. 2000. “Extreme Programming Explained. Embrace Change”, Pearson Education, 1999, Traducido al español como: “Una explicación de la programación extrema. Aceptar el cambio”, Addison Wesley, 2000

BERNHARD Rumpe y Astrid Schröder, “Quantitative survey on Extreme Programming Projects”, Informe, Universidad de Tecnología de Munich, 2001.

BOEHM, B. W, Software Engineering, IEEE Transactions on Computers, C-25, num. 12, diciembre, pp. 1226-.1241, 1976

BOEHM, B., Using WINWIN Spiral Model: A the Case Study,Computer, vol. 31, n."7, pp. 33-44, Julio 1998

BROOKS, F., The Mytical Man-Month, Addison-Wesley, 1975

CALERO SOLIS, Manuel, Una explicación de la programación extrema (XP), V Encuentro usuarios xBase, [En línea] 2003. [Citado el: 16 de 2 de 2009.] http://www.willydev.net/InsiteCreation/v1.0/descargas/prev/explicaxp.pdf.

CLEMENTS, Paul. Coming attractioons in Software Arquitecture. Pensilvania, EE.UU. : Software Engineering Institute,University Carnegie Mellon, Enero, 1996

LARMAN, Craig, Agile & Iterative Development. A Manager’s Guide. Reading, Addison Wesley, 2004

CHAVEZ, Enrique, Rich Internet application, http://www.riactive.com/2006/12/06/rich-internet-application-definicion/, 2006

DAVIS, A., y P. Sitaram, ccA Concurrent Process Model for Software Developement,,, Software Engineering Notes, ACM Press, vol. 19, n." 2, pp. 38-51, Abril 1994

DAVIS, M.J., ccprocess and Product: Dichotomy or Duality, Software Engineering Notes, ACM Press, vol. 20, n." 2, pp. 17- 18, Abril 1995

DELGADO, Antonio, Aplicaciones Ricas de Internet: la Red desde el escritorio, http://www.comsumer.es, Noviembre, 2007

ESTUDIS, Josep Figueras, RIA y Adobe Flex un nuevo concepto de aplicaciones web, 2009

FAIN,Yakov, Victor Rasputnis y Anatole Tartakovsky, Aplicaciones Ricas de Internet con ADOBE FLEX Y JAVA. 2007

GARLAN D., M. Shaw. An Introduction to Software Architecture, http://www.sei.cmu.edu/publications/documents/94.reports/94.tr.021.html, 1994

GARLAN, An Introduction to Software Architecture, http://www.sei.cmu.edu/publications/documents/94.reports/94.tr.021.html, CMU/SEI-94-TR-021, Enero, 1994

GROUP, S. The Chaos Report, 2006

GSI. 2007. GRUPO SOLUCIONES INNOVA. [En línea] 2007. [Citado el: 27 de 3 de 2009.] http://www.rational.com.ar/herramientas/rup.html.

Hanna, M, Farewell Waterfalls, Software Magazine, pp.38-46, Mayo 1995

IEEE Std 1074-1997, IEEE Standard for Developing Software Life Cycle Processes

IEEE: Standards Collection: Somare Engineering, IEEE Standard 610.12-1990,IEEE, 1993.

JACOBSON, I. "Applying UML in The Unified Process" Presentación. Rational Software. Presentación disponible en http://www.rational.com/uml como UMLconf.zip, 1998

JACOBSON, I. y BOOCH, G. y RUMBAUGH, J., “El Proceso Unificado de Desarrollo de software”, Prólogo, Capítulos 1-5, Apéndice A. Visión General de UML, Apéndice B. . s.l. : Addison-Wesley, Páginas 3-104, 407-424, 2000

JACOBSON, I. y BOOCH, G. y RUMBAUGH, J. 2000. “El Proceso Unificado de Desarrollo de software”.Epígrafe 2.5.5 Las herramientas dan soporte al ciclo de vida completo,: Addison-Wesley, 2000. Páginas 28-29.

KAISER, S. H, Software Paradigms, p. 0471483478, 2005

LANE, Thomas G. Studying Software Architecture Through Design Spaces and Rules, University Carnegie Mellon : Pittsburgh, Pennsylvania 15213, http://www.sei.cmu.edu/pub/documents/90.reports/pdf/tr18.90.pdf, Noviembre, 1990

LETELIER, P. (2002). Rational Unified Process (RUP). Recuperado el 2 de 3 de 2009, de Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia : https://pid.dsic.upv.es/C1/Material/Documentos%20Disponibles/Introducci%C3%B3n%20a%20RUP.doc

MARTIN, J.. Rapid Aplication Developement, Prentice-Hall. 1991

MCDERMID, J. y P. Rook, ccsoftware Development Process Models, en Software Engineer's Reference Book, CRC Press, pp. 15126-15128, 1993

METTALA Erik, Marc H. Graham. The Domain-Specific Software Arquitecture Programs. s.l. : Special Report, Junio, 1992

NIERSTRASZ, <Component-Oriented Software Developement, CACM, vol. 35, n." 9, pp.160-165, Septiembre 1992

PALACIO, Juan. 2006. Navegapolis. El modelo Scrum. [En línea] 2006. [Citado el: 4 de 3 de 2009.] http://www.navegapolis.net/content/view/694 .

PECOS, Daniel. 2009. PostgreSQL vs. MySQL. [En línea] 2009. [Citado el: 20 de 3 de 2009.] http://www.netpecos.org/docs/mysql_postgres/index.html.

PIMENTEL González, Luis Alberto, Iósev Pérez Rivero; ArBaWeb: ARQUITECTURA BASE SOBRE LA WEB, 2007

PRESSMAN, R. S. Ingeniería del software, un Enfoque Práctico. Quinta edición. Madrid, McGraw-Hil, 2002

RAMOS Blanco, Kariné, IPP-3510_2009 Libro de Proceso para la Administración de Requisitos, 2009

ROYCE, W.W. Managintgh e Developement of Large Software Systems: Concepts and Techniques,Proc. WESCON. 1970

SCHWABER K, Beedle M., Martin R.C. “Agile Software Development with SCRUM”. Prentice Hall. 2001.Prentice Hall. 1999.

SHELEG, W., ccConcurrent Engineering: A New Paradign for CIS Developement, Application Development Trends, vol. 1, n." 6, pp. 28-33, Junio 1994

SUÁREZ Batista, Anisbert,IPP_3530 Libro de Proceso para PMC , 2009

ZELKOVITZ M. V., Shaw A. C. y Gannon J. D., Principles o Software Engineering and Design. Prentice Hall, Englewoods Clif, 1979

BIBLIOGRAFÍA

AAEN, IVAN, MATHIASSEN, LARS y BOTCHER, PETER. 1997. Software Factories. [En línea] 1997. [Citado el: 2 de 3 de 2009.] http://www.cin.ufpe.br/~in953/lectures/papers/Software_Factories_17.pdf.

ABREU, BARTOLOMEO, YANEDI y COLOME, DUNIA MARIA. 2007. Portal de los Institutos Politécnicos de Informática. La Habana : s.n., 2007.

ACCENTURE. 2009. [En línea] 2009. [Citado el: 27 de 4 de 2009.] http://www.accenture.com/Countries/Spain/About_Accenture/default.htm.

ADOBE Flex Builder 3. Create engaging, cross-platform Rich Internet Applications, Adobe System Inc, 2007

ALHAMBRA-EIDOS. 2007. [En línea] 2007. [Citado el: 2 de 5 de 2009.] http://www.alhambra-eidos.es/index.html.

ALTAMIRANDA, JUNIOR. 2009. PLATAFORMA GFORGE, Universidad de los Andes. [En línea] 2009. [Citado el: 3 de 2 de 2009.] http://sistemas.fsl.fundacite-merida.gob.ve/docman/view.php/16/116/gforge.pdf.

ÁLVAREZ, MIGUEL ANGEL. 2003. Desarrollo Web. Zend Studio. [En línea] 2003. [Citado el: 10 de 3 de 2009.] http://www.desarrolloweb.com/articulos/1178.php.

ANONIMO. 2008. Microsoft y Tata Consultancy Services proporcionan servicios a los proveedores de telecomunicaciones. [En línea] 2008. [Citado el: 2 de 5 de 2009.] http://www.prnewswire.co.uk/cgi/news/release?id=145706.

BANDINELLI, S. et al, {{Modelingand Improving an Industrial Software Process,,, IEEE Trans. Software Engi-neering, vol. 21, n.", Mayo 1995, pp. 440-454.

BASILI, VICTOR R, CALDIERA, GIANLUIGI y CANTONE, GIOVANNI. 1992 . A Reference Archiecture for the Component Factory. ACM Transaction on Software Engineering and Methodology. 1992 .

BAUER, E. L. 1972. Software Engineering, Information Processing. 71, North Holland Publishing Co,Amsterdam : s.n., 1972.

BECK, K. 2000. “Extreme Programming Explained. Embrace Change”,Pearson Education,1999,Traducido al español como: “Una explicación de la programación extrema. Aceptar el cambio”, Addison Wesley. 2000.

BOEHM, B. 1976. Software Engineering, IEEE Transactions on Computers. 1976. C-25, num. 12, pp. 1226-.1241.

BOEHM, B. 1998. Usingth e WINWIN Spiral Model: A Case Study, Computer, vol. 31, n."7, pp. 33-44. 1998.

BERNHARD RUMPE y ASTRID SCHÖDER. “Quantitative survey on Extreme Programming Projects”. Informe, Universidad de Tecnología de Munich, 2001.

BROOKS, F. 1975. The Mytical Man-Month, Addison-Wesley. 1975.

CHAVEZ, Enrique, Rich Internet application, http://www.riactive.com/2006/12/06/rich-internet-application-definicion/, 2006

CALERO SOLIS, MANUEL. 2003. Una explicación de la programación extrema (XP).V Encuentro usuarios xBase. [En línea] 2003. [Citado el: 16 de 2 de 2009.] http://www.willydev.net/InsiteCreation/v1.0/descargas/prev/explicaxp.pdf.

CANÓS, JOSE H, LETELIER, PATRICIO y PENADÉS, Mª CARMEN. Metodologías Ágiles en el Desarrollo de Software. [En línea] [Citado el: 23 de 2 de 2009.] http://www.willydev.net/descargas/prev/TodoAgil.Pdf.

CANTONE, GIOVANNI. 1992. Software Factory: Modeling the Improvement. 'Competitive Performance Through Advanced Technology'. Third International Conference on (Conf. Publ. No. 359). 1992.

CAPGEMINI. 2009. [En línea] 2009. [Citado el: 03 de 05 de 2009.] http://www.es.capgemini.com/quienes/.

CLEMENTS, Paul. Coming attractioons in Software Arquitecture. Pensilvania, EE.UU. : Software Engineering Institute,University Carnegie Mellon, Enero, 1996

COMUNIDAD JOOMLA. 2008. [En línea] 2008. [Citado el: 2 de 5 de 2009.] http://comunidadjoomla.org/.

CUSUMANO y A, MICHAEL. 1989. Software Factory: A Historical Interpretation. IEEE Software. 1989.

DAVID, A., y P. SITARAM, ccA Concurrent Process Model for Software Developement,,, Software Enginee-ring Notes, ACM Press, vol. 19, n." 2, Abril 1994, pp. 38-5 1.

DAVID, M.J., ccprocess and Product: Dichotomy or Duality,,, Software Engineering Notes, ACM Press, vol. 20, n." 2, Abril 1995, pp. 17- 18.

DBPEDIA. 2009. [En línea] 2009. [Citado el: 4 de 2 de 2009.] http://dbpedia.org/page/Repository.

DEEPAK ALUR, J. C., DAN MALKS, Core J2EE™ Patterns: Best Practices and Design Strategies, Second Edition, p. 0-13-142246-4, 2003

DE GRACIA, CARLOS. 2008. Estudio Comparativo: Dreamweaver CS3 vs. Visual Studio 2008. Enfocado en el manejo de XHTML, CSS y JavaScript. [En línea] 2008. [Citado el: 13

de 1 de 2009.] http://www.scribd.com/doc/3634510/Dreamweaver-CS3-vs-Visual-Studio-2008.

DELGADO, Antonio, Aplicaciones Ricas de Internet: la Red desde el escritorio, http://www.comsumer.es, Noviembre, 2007

DERMID, J. y P. ROOK. 1993. ccsoftware Development Process Models,en Software Ingineer's Reference Book, CRC Press, pp. 15126-15128. 1993.

DÍAZ FLORES, MIRIAN MILAGROS. Diferencia de RUP y XP, Escuela de Ingeniería de Sistemas. [En línea] [Citado el: 12 de 3 de 2009.] http://www.usmp.edu.pe/publicaciones/boletin/fia/info49/articulos/RUP%20vs.%20XP.pdf.

ESTUDIS, Josep Figueras, RIA y Adobe Flex un nuevo concepto de aplicaciones web, 2009

FABRI, JOSÉ AUGUSTO, SCHNECK D, MARCELO y S, M D M. 2004. ElabTI: Um ambiente real e replicável de produção de software. Brasil Escola Politecnica USP. 2004.

FACTORY, V. S. 2006. Web Corporativa Vector Software Factory. [En línea] 2006. [Citado el: 25 de 3 de 2009.] http://www.vectorsf.com/h/index2.html?preSet=noticias_2006.html.

FAIN,Yakov, Victor Rasputnis y Anatole Tartakovsky, Aplicaciones Ricas de Internet con ADOBE FLEX Y JAVA. 2007

FERNANDES, AGUINALDO ARAGON y DESCARTES DE SOUZA, TEIXEIRA. 2004. Fábrica de Software:Implementação e Gestão de Operações. 2004.

FERNSTROM, C, NARFELT, K H y OHLSSON, L. 1992. Software Factory Principles, Architecture and Experiments. IEEE Software. 1992.

FREE DOWNLOAD MANAGER. 2007. Visual Paradigm for UML . [En línea] 2007. [Citado el: 13 de 2 de 2009.] http://www.freedownloadmanager.org/es/downloads/Paradigma_Visual_para_UML_(M%C3%8D)_14720_p.

GARLAN D., M. Shaw. An Introduction to Software Architecture, http://www.sei.cmu.edu/publications/documents/94.reports/94.tr.021.html, 1994

GARLAN, An Introduction to Software Architecture, http://www.sei.cmu.edu/publications/documents/94.reports/94.tr.021.html, CMU/SEI-94-TR-021, Enero, 1994

GARZÁS PARRA, JAVIER y PIATTINI VELTHUIS, MARIO G. 2007. FABRICAS DEL SOFTWARE: EXPERIENCIAS, TECNOLOGÍAS Y ORGANIZACIÓN. s.l. : RA-MA, 2007.

GEPSEA. 2009. Grupo de Estudios Prospectivos Sociedad Economía y Ambiente. La sociedad del Conocimiento. [En línea] 2009. [Citado el: 1 de 2 de 2009.] http://personales.com/venezuela/merida/gepsea/sc.htm.

GIRALDO, LUIS y ZAPATA, YULIANA. 2005. Herramientas de desarrollo de ingeniería de SW para Linux . [En línea] 2005. [Citado el: 29 de 1 de 2009.] http://hugolopez.phi.com.co/docs/download/file=Giraldo-Zapata-Herramientas%20de%20ISW.pdf,_id=17.

GROUP. 2006. GROUP,S. The Chaos Report. 2006.

GRUPO SMS. 2008. [En línea] 2008. [Citado el: 14 de 4 de 2009.] http://www.grupo-sms.com/index.asp.

GSI. 2007. GRUPO SOLUCIONES INNOVA. [En línea] 2007. [Citado el: 27 de 3 de 2009.] http://www.rational.com.ar/herramientas/rup.html.

GUGLIELMETTI, MARCOS. 2004. Mastermagazine. Definición de Portal. . [En línea] 2004. [Citado el: 5 de 5 de 2009.] http://www.mastermagazine.info/termino/6349.php.

Hanna, M, Farewell Waterfalls, Software Magazine, pp.38-46, Mayo 1995

HERNÁNDEZ LEÓN, ROLANDO ALFREDO y G., S. C. 2002. El Paradigma Cuantitativo de la Investigación Científica. 2002.

IBERMATICA. 2007. [En línea] 2007. [Citado el: 24 de 3 de 2009.] http://www.ibermatica.com/ibermatica.

IEEE. 1993. IEEE: Standards Collection: Somare Engineering. 1993. Standard 610.12-1990.

IEEE Std 1074-1997, IEEE Standard for Developing Software Life Cycle Processes

INDIGO. 2009. ¿Qué es un Portal Web? [En línea] 2009. [Citado el: 13 de 2 de 2009.] http://www.indigo.com.mx/temas-de-ayudausuarios/40-ique-es-un-portal-web.html?9ceb1fcf0e3e09e23cfa5c3094ad1bd9=611e5154602837870d7317005b8b5b23.

INDRA. 2008. [En línea] 2008. [Citado el: 1 de 2 de 2009.] http://www.indra.es/servlet/ContentServer?cid=1082008092186&pagename=IndraES%2FPage%2FEstructMenuRastroModulos&Language=es_ES&pid=1082008092186&c=Page.

JACOBSON, I. "Applying UML in The Unified Process" Presentación. Rational Software. Presentación disponible en http://www.rational.com/uml como UMLconf.zip, 1998

JACOBSON, I. y BOOCH, G. y RUMBAUGH, J., “El Proceso Unificado de Desarrollo de software”, Prólogo, Capítulos 1-5, Apéndice A. Visión General de UML, Apéndice B. . s.l. : Addison-Wesley, Páginas 3-104, 407-424, 2000

JACOBSON, I. y BOOCH, G. y RUMBAUGH, J. 2000. “El Proceso Unificado de Desarrollo de software”.Epígrafe 2.5.5 Las herramientas dan soporte al ciclo de vida completo,: Addison-Wesley, 2000. Páginas 28-29.

KAISER, S. H, Software Paradigms, p. 0471483478, 2005

KEN y J., y R. HUNTER. 1994. Inside RAD, McGraw-Hill. 1994

LANE, Thomas G. Studying Software Architecture Through Design Spaces and Rules, University Carnegie Mellon : Pittsburgh, Pennsylvania 15213, http://www.sei.cmu.edu/pub/documents/90.reports/pdf/tr18.90.pdf, Noviembre, 1990

LARMAN, Craig, Agile & Iterative Development. A Manager's Guide. Reading, Addison Wesley, 2004

LETELIER, P. (2002). Rational Unified Process (RUP). Recuperado el 2 de 3 de 2009, de Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia : https://pid.dsic.upv.es/C1/Material/Documentos%20Disponibles/Introducci%C3%B3n%20a%20RUP.doc

MARTIN, J.. Rapid Aplication Developement, Prentice-Hall. 1991

MCDERMID, J. y P. Rook, ccsoftware Development Process Models, en Software Engineer's Reference Book, CRC Press, pp. 15126-15128, 1993

MENENDEZ, ROSA. 2007. Rational Software Corporation. [En línea] 2007. [Citado el: 23 de 1 de 2009.] http://www.usmp.edu.pe/publicaciones/boletin/fia/info36/proyectos.html.

Metodologías Ágiles en el Desarrollo de Software. [En línea] [Citado el: 24 de 2 de 2009.] http://www.willydev.net/descargas/prev/TodoAgil.Pdf.

METTALA Erik, Marc H. Graham. The Domain-Specific Software Arquitecture Programs. s.l. : Special Report, Junio, 1992

NIERSTRASZ, <Component-Oriented Software Developement, CACM, vol. 35, n." 9, pp.160-165, Septiembre 1992

PALACIO, Juan. 2006. Navegapolis. El modelo Scrum. [En línea] 2006. [Citado el: 4 de 3 de 2009.] http://www.navegapolis.net/content/view/694 .

PASTOR, PALOMA. 2007. GRUPO GESFOR. [En línea] 2007. [Citado el: 19 de 3 de 2009.] http://www.gesfor.es/htm/08/salaprensa/doc/dosierprensaMarmickVS2.doc.

PECOS, Daniel. 2009. PostgreSQL vs. MySQL. [En línea] 2009. [Citado el: 20 de 3 de 2009.] http://www.netpecos.org/docs/mysql_postgres/index.html.

PIMENTEL González, Luis Alberto, Iósev Pérez Rivero; ArBaWeb: ARQUITECTURA BASE SOBRE LA WEB, 2007

PRESSMAN, R. S. Ingeniería del software, un Enfoque Práctico. Quinta edición. Madrid, McGraw-Hil, 2002

PROCESS MODELS,,, en Software Ingineer's Reference Book, CRC Press, 1993, pp. 15126-15128.

QUANTA PLUS. 2005. [En línea] 2005. [Citado el: 25 de 2 de 2009.] http://quanta.kdewebdev.org/.

RAMOS Blanco, Kariné, IPP-3510_2009 Libro de Proceso para la Administración de Requisitos, 2009

RAVEST, CECILIA. 2008. [En línea] 2008. [Citado el: 28 de 4 de 2009.] http://www.bnamericas.com/company-profile/tecnologia/Tata_Consultancy_Services_Iberoamerica_S,A,-TCS_Iberoamerica.

REYERO, JOSE A. 2006. DRUPAL. [En línea] 2006. [Citado el: 13 de 1 de 2009.] http://drupal.org.es/drupal.

RÍOS, YANOSKY. 2005. Modelo funcional de la Factoría de Software de la UCI para la línea Carrefour. Ciudad Habana : s.n., 2005.

ROCKWELL, R. G., M. H. 1993. The Eureka Software Factory CoRe: A Conceptual Reference Model for Software Factories. Software Engineering Environments Conference. 1993.

ROYCE, W.W. 1970. Managintgh e Developement of Large Software Systems: Concepts and Techniques,Proc. WESCON. 1970.

SCHWABER K, Beedle M., Martin R.C. “Agile Software Development with SCRUM”. Prentice Hall. 2001.Prentice Hall. 1999.

SHELEG, W., ccConcurrent Engineering: A New Para-dign for CIS Developement,,, Application Development Trends, vol. 1, n." 6, Junio 1994, pp. 28-33.

SKYNETSTUDIO. 2008. Tecnología. [En línea] 2008. [Citado el: 5 de 5 de 2009.] http://www.skynet-studio.com.ar/tecnologia.aspx.

SOFTWARE ENGINEERING INSTITUTE. 2006. CMMI® for Development,Version 1.2. Pittsburgh : s.n., 2006. PA 15213-3890.

SUÁREZ Batista, Anisbert,IPP_3530 Libro de Proceso para PMC , 2009

SUN. 2007. Desarrollo de aplicaciones multiplataforma con NetBeans IDE . [En línea] 2007. [Citado el: 18 de 1 de 2009.] http://www.sun.com/emrkt/innercircle/newsletter/spain/0207spain_feature.html.

TÁPANES ROBAU, DAYSARÍH y RODRIGUEZ BATISTA, ARMANDO. 2006. La transferencia de tecnología asociada al proceso inversionista en Cuba en el cuatrienio 2002-2005. [En línea] 2006. [Citado el: 12 de 3 de 2009.] http://www.bibliociencias.cu/gsdl/collect/eventos/index/assoc/HASH0111/380f07a0.dir/doc.pdf.

THE APACHE SOFTWARE FOUNDATION. 2009 . . [En línea] 2009 . [Citado el: 5 de 5 de 2009 .] http://www.apache.org/.

TID. 2007. PROYECTO VULCANO. Forja de proyectos software de calidad. Estudio de herramientas de certificación. [En línea] 2007. [Citado el: 24 de 3 de 2009.] http://www.ines.org.es/vulcano/wp-content/uploads/2007/09/d6-estudio-de-herramientas-de-certificacion-tid.pdf.

TRUJILLO CASAÑOLA, YAIMI. 2007. Modelo de factoría de software aplicando inteligencia. Ciudad de la Habana : s.n., 2007.

T-SYSTEMS. 2009. [En línea] 2009. [Citado el: 4 de 2 de 2009.] http://www.t-systems.es/tsi/es/100100/Homepage.

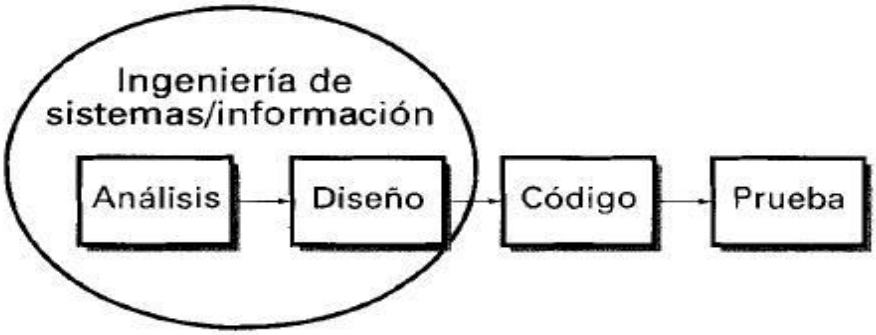
VICTOR R, BASILI, GIANLUIGI, CALDIERA y CANTONE, GIOVANNI. 1992. A Reference Archiecture for the Component Factory. ACM Transaction on Software Engineering and Methodology. 1992

WEB21. 2009. Portales Web a Medida. [En línea] 2009. [Citado el: 4 de 5 de 2009.] http://www.web21.es/index.php?opcion=1&id_nodo=136.

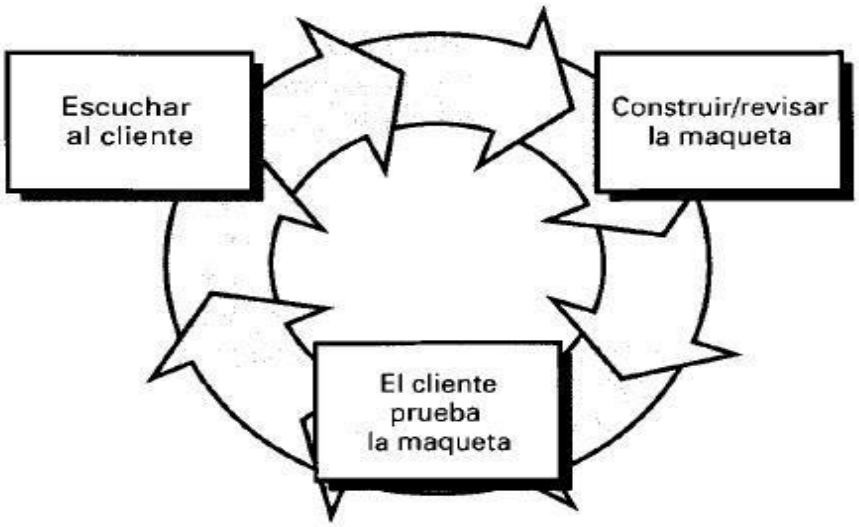
ZELKOVITZ M. V., Shaw A. C. y Gannon J. D., Principles o Software Engineering and Design. Prentice Hall, Englewoods Clif, 1979

ANEXOS

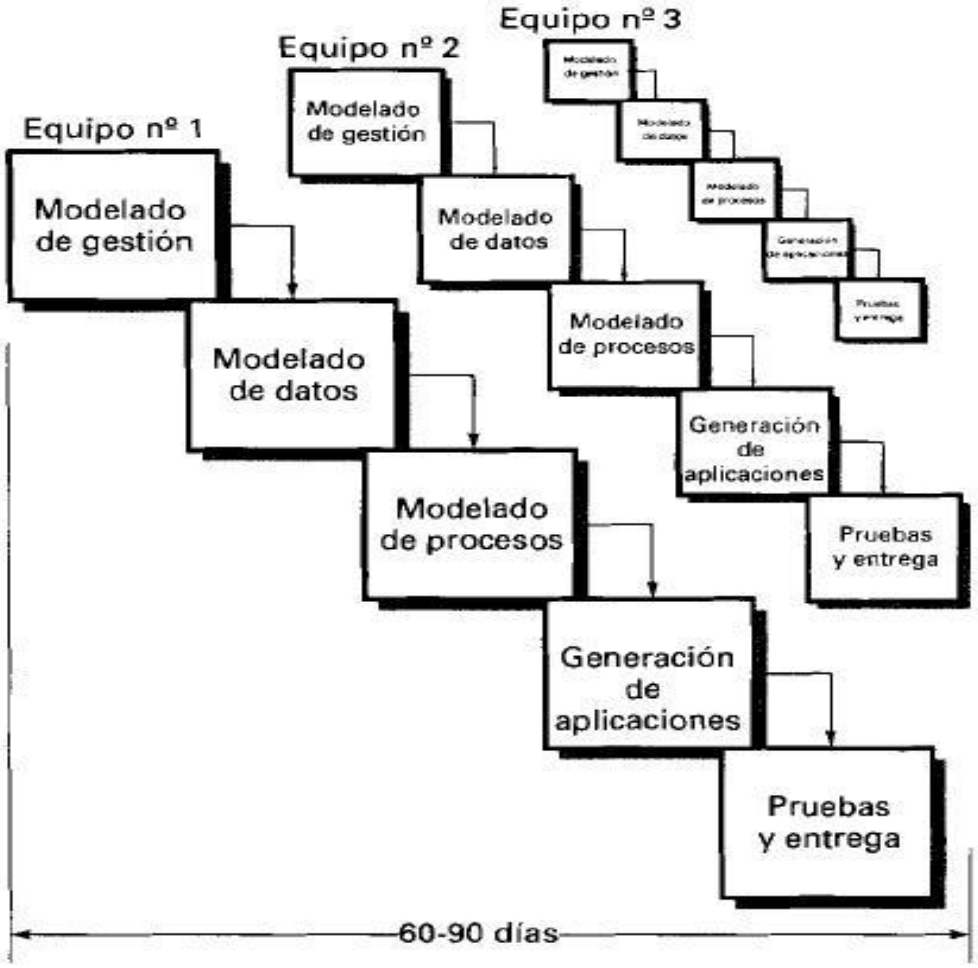
Anexo 1: Modelo lineal secuencial (Modelo en Cascada)



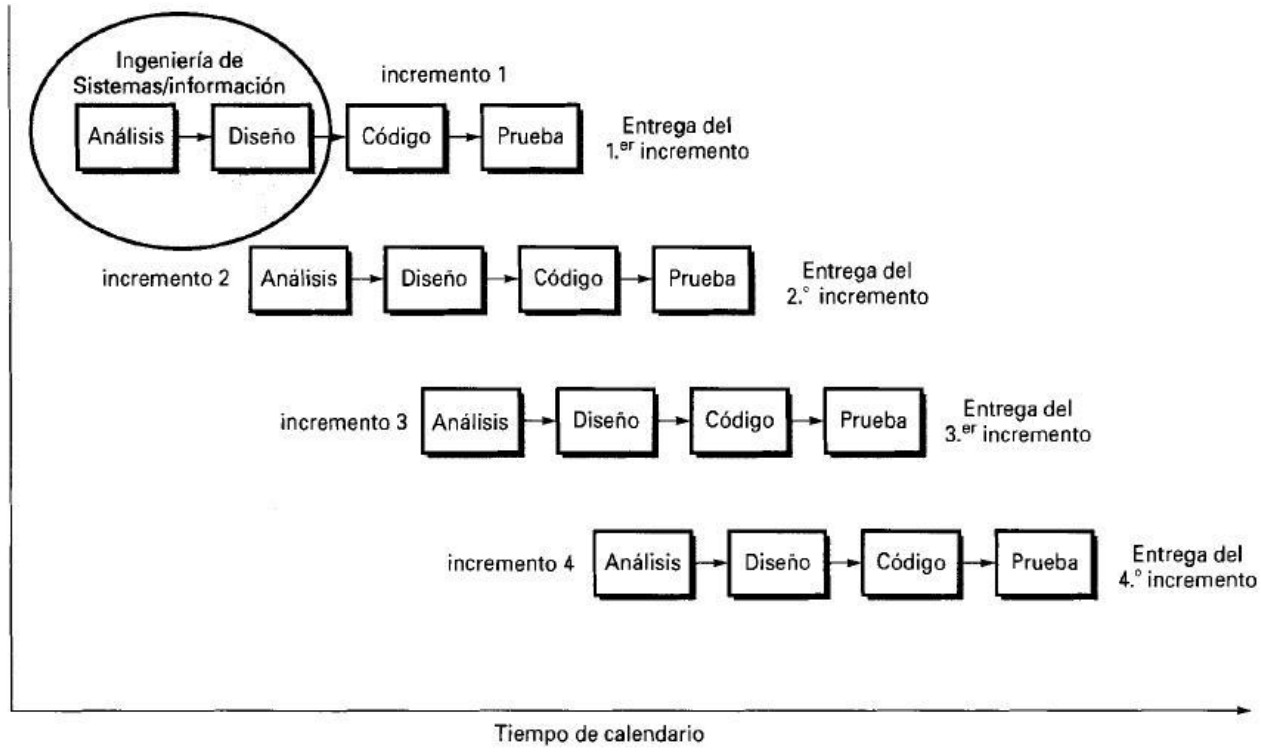
Anexo 2: Modelo construcción de prototipos



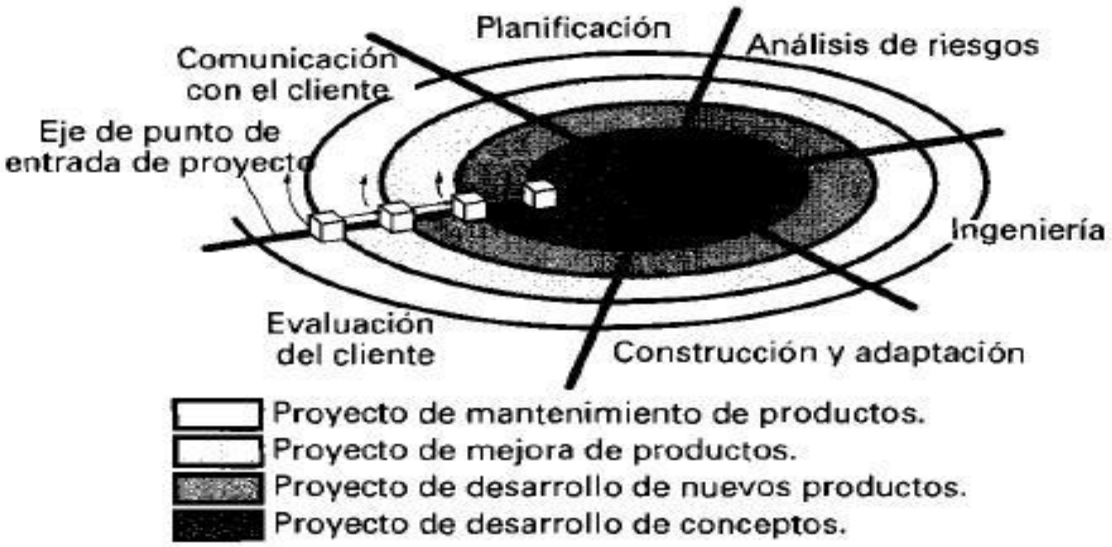
Anexo 3: Modelo DRA (Desarrollo Rápido de Aplicaciones)



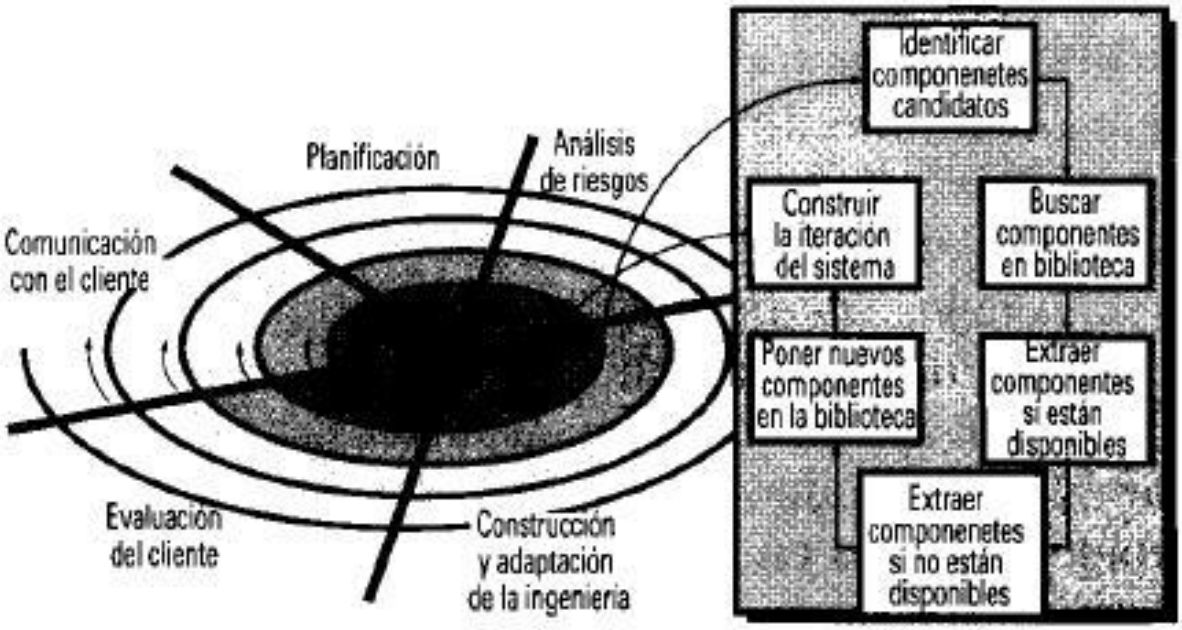
Anexo 4: Modelo Evolutivo del proceso del software: Incremental



Anexo 5: Modelo Evolutivo del proceso del software: Espiral



Anexo 6: Modelo de Desarrollo basado en componentes



Anexo 7: Matriz de comparación entre modelos de desarrollo del software

Nombre del Modelo	Acrónimo	Tipo de Modelo	Ventajas	Desventajas
Lineal Secuencial	No tiene	Cascada	- Paradigma más antiguo y más extensamente utilizado en la ingeniería del software. -Modelado según el ciclo de ingeniería convencional. - Proporciona una plantilla en la que se encuentran métodos para análisis, diseño, codificación, pruebas y mantenimiento. - Una fase no comienza hasta que termine la fase anterior y generalmente se incluye la corrección de los problemas encontrados en fases previas.	-Requiere que el cliente exponga explícitamente al principio todos los requisitos, lo que comúnmente es difícil que ocurra. - No estará disponible ninguna versión del programa hasta que el proyecto no esté muy avanzado. - Existe una alta probabilidad de que el software no cumpla con los requisitos del usuario por el largo tiempo de entrega del producto.
Construcción de Prototipos	No tiene	Iterativo	- Puede servir como primera versión del sistema. Se construye para servir como un mecanismo de definición de requisitos. -Los usuarios y desarrolladores logran un mejor entendimiento del sistema. Esto se refleja en una mejora de la calidad del software.	-Para el rápido desarrollo se necesitan herramientas que pueden ser incompatibles con otras o que poca gente sabe utilizar. - De forma demasiado frecuente la gestión de desarrollo del software es muy lenta.
Desarrollo Rápido de Aplicaciones	DRA	Cascada	- Permite construir sistemas utilizables en poco tiempo -Se utiliza una construcción basada en componentes. - Los entregables pueden ser fácilmente trasladados a otra plataforma. - Visibilidad temprana, una mayor flexibilidad y la codificación manual es menor.	- Requiere clientes y desarrolladores comprometidos, sino los proyectos fracasarán. - Para proyectos grandes requiere recursos humanos suficientes. - No es adecuado cuando los riesgos técnicos son altos.
Incremental	No tiene	Cascada, Iterativo	-Los clientes no tienen que esperar hasta que el sistema se entregue completamente para comenzar a hacer uso de él. - Los clientes pueden aclarar los requisitos que no tengan claros conforme ven las entregas del sistema. - Se disminuye el riesgo de fracaso de todo	- Cada incremento debe ser pequeño para limitar el riesgo (menos de 20.000 líneas). - Cada incremento debe aumentar la funcionalidad. - Es difícil establecer las correspondencias de los requisitos

			el proyecto.	contra los incrementos. - Es difícil detectar las unidades o servicios genéricos para todo el sistema.
Espiral	No tiene	Cascada, Iterativo	-Puede adaptarse y aplicarse a lo largo de la vida del software de computadora. - Las tareas se aplican a cualquier proyecto de software que se realice, sin tener en cuenta el tamaño ni la complejidad. - Son apropiados particularmente para el desarrollo de sistemas orientados a objetos. - Toma en consideración explícitamente el riesgo. - Permite a quien lo desarrolla aplicar el enfoque de construcción de prototipos en cualquier etapa de evolución del producto.	- Su principal desventaja es que es nuevo (1988) y no se ha utilizado tanto como otros modelos de ciclo de vida. - Requiere una considerable habilidad para la evaluación del riesgo. -Puede resultar difícil convencer a algunos clientes que el proceso es controlable.
Desarrollo basado en componentes	No tiene	Iterativo	-Permite alcanzar un alto índice de productividad y la reutilización del software. -Reduce el tiempo de entrega, el costo y esfuerzo del proyecto. - Disminuye los riesgos durante el desarrollo.	- Los compromisos en los requisitos son inevitables, por lo cual puede que el software no cumpla las expectativas del cliente. - Las actualizaciones de los componentes adquiridos no están en manos de los desarrolladores del sistema.

Anexo 8: Matriz de comparación entre metodologías tradicionales y ágiles

Metodologías Ágiles	Metodologías Tradicionales
Basadas en heurísticas provenientes de prácticas de producción de Código Fuente	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo.
Especialmente preparados para cambios durante el proyecto.	Cierta resistencia a los cambios.
Impuestas internamente (por el equipo)	Impuestas externamente
Proceso menos controlado, con pocos principios.	Proceso mucho más controlado, con numerosas políticas/normas
No existe contrato tradicional o al menos es bastante flexible.	Existe un contrato prefijado.
El cliente es parte del equipo de desarrollo.	El cliente interactúa con el equipo de desarrollo mediante reuniones.
Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio.	Grupos grandes y posiblemente distribuidos.
Pocos artefactos.	Más artefactos.
Pocos roles.	Más roles.
Menos énfasis en la arquitectura del software.	La arquitectura del software es esencial y se expresa mediante modelos.

Anexo 9: Matriz de comparación entre las metodologías de desarrollo de software

Nombre de la metodología	RUP	XP	SCRUM
Actividades	Modelar diagrama de casos de uso del negocio y del sistema, Análisis de la arquitectura, Captura de requisitos, Diseñar la arquitectura, Identificar clases de diseño, Creación del modelo de diseño, del análisis, de despliegue y de implementación. Implementación de la arquitectura, Implementar clases, Integrar producto, Planificar prueba, Diseñar prueba, Ejecutar pruebas, Evaluar pruebas, Reportar defectos, Control de cambio en la configuración, Generación de informes de estado, Auditorías de la configuración.	Planificación, Diseño, Codificación, Pruebas.	Planificación de sprint, Reunión diaria, Revisión de sprint.
Flujos de trabajo	Modelado del negocio, Requisitos, Análisis y Diseño, Implementación, Pruebas, Despliegue, Gestión del proyecto, Configuración y control de cambios, Ambiente.	Exploración, Planificación de la Entrega (Release), Iteraciones, Producción, Mantenimiento y Muerte del Proyecto	Todo el trabajo se hace en Sprint. Cada Sprint es una iteración de 30 días de calendario consecutivos.
Artefactos	Modelo de Casos de Uso, Modelo de Análisis y Diseño, Modelo de Despliegue, Modelo de Implementación, Diagrama de clases ,de interacción y subsistema de diseño, Realizaciones de casos de uso, Especificación de Requisitos, Descripción de la arquitectura software, Plan de integración de construcciones, Modelo de prueba, Plan de prueba, Casos de prueba, Evaluación de prueba, Reporte de defectos, Documento de Visión, Glosario de términos, Lista de riesgos y plan de contingencia, Plan del proyecto, Plan de Aseguramiento de la Calidad, Manual de Usuario	Historias del Usuario, Tareas de Ingeniería, Pruebas de Aceptación, Pruebas Unitarias y de Integración, Plan de la Entrega, Código Fuente	Pila del producto, Pila del sprint, Incremento.
Roles	Gestores (Líder del proyecto), Analistas, Desarrollador/Diseñador, Especialistas en Prueba, Apoyo, Otros roles.	Jefe de Proyecto, Programador, Cliente, Encargado de Pruebas, Rastreador, Entrenador.	Propietario del producto, Equipo de desarrollo, Scrum Manager.

Anexo 10: Matriz de comparación entre Java y AS3

	Java 5	ActionScript 3
Empaquetamiento de librerías de clases	.jar	.swc
Herencia	Class Employee extends Person{...}	Class Employee extends Person{...}

Declaración de Variables e Inicialización	String firtName=”John”;
	Date shipDate=new Date();
	init i;
	int a, b=10;
	double salary;
		var firstName:String=”John”;
		
		var shipDate:Date=new Date();
		var i:int;
		
		var a:int, b:int=10;
		var salary:Number;

Variables no declaradas	No existen	Si declaramos una variable pero no especificamos el tipo, el tipo * será aplicado. Un valor por defecto es: undefined var MyVar:*;
Rango de acción de variables	Bloques. Locales: declaradas en métodos o bloques. Miembros: declaradas a nivel de clases. No hay variables globales	No hay rango de acción a nivel de bloques: el espectro mínimo es la función. Locales: declaradas dentro de la función. Miembros: declaradas a nivel de clases. Si una variable es declarada fuera de cualquier función o definición de clases, esta se considera de tipo global.
Strings	Almacena secuencias de caracteres Unicode de dos byte	Almacena secuencias de caracteres Unicode de dos byte
Terminación de sentencias con punto y coma (;)	Obligatorio	Si escribes una sentencia por línea puedes omitirlo
Operador de igualdad estricta (comparación de objetos sin conversión de tipos)	No existe	=== para uso estricto de no_igualdad !==

Calificados de constantes	La palabra clave “final” final int STATE=”NY”;	La palabra clave “const” const STATE:int=”NY”;
Chequeo de tipado	Estático, chequeado en tiempo de compilación	Dinámico, chequeado en runtime. Estático, llamado strict-mode, el cual existe por defecto en Flex Builder
Operador para chequeo de tipado	“**instanceof**”	“**is**” chequea tipo de datos, ejemplo: if (myVar **is** String){...} El operador “**is**” es un remplazo para el antiguo operador “**instanceof**”

Operador “**as**”	No existe	Semejante al operador “**is**”, pero no retorna un Booleano sino el valor de la expresión:
		
		var orderId:String=”123”;
		
		var orderIdN:Number=orderId as Number;
		
		trace(orderIdN);//prints 123

Primitivas	byte, int, long, float,
	
	double,short, boolean, char
		Todas las primitivas en ActionScript son objetos. Boolean, int, uint, Number, String.
		
		Las siguientes líneas son equivalentes:
		var age:int = 25;
		
		var age:int = new int(25);

Tipos complejos	No existen	Array, Date, Error, Function, RegExp, XML, and XMLList
Declaración de arreglos y creación de instancias	int quarterResults[]; quarterResults = new int[4]; int quarterRe-sults[]={25,33,56,84};	Arreglos en ActionScript se asemejan a los ArrayList de Java var quarterResults:Array =new Array(); ó

		var quarterResults:Array=[]; var quarterResults:Array= [25, 33, 56, 84]; AS3 también tiene asociados arreglos que usan elementos con nombre, en lugar de indexados numéricos(semejante a las tablas Hash)
Clase superior en el arbol de herencia	Object	Object

Sintaxis de casting. Chequear Person con la clase Objeto	Person p=(Person)
	
	myObject;
		var p:Person= Person(myObject);
		
		ó
		var p:Person= myObject as Person;

Upcasting	class Xyz extends Abc{} Abc myObj = new Xyz();	class Xyz extends Abc{} var myObj:Abc=new Xyz();
Variables sin tipado	No existen	var myObject:* var myObject:

Empaquetar	package com.xyz;
	
	class myClass {...}
		package com.xyz{
		
		class myClass{...}
		
		}
		
		Al empaquetar en ActionScript puede incluir no solo clases, sino funciones separadas también.

Nivel de acceso de las clases	public, private, protected si ninguno es especificado, las clases tienen nivel de acceso de paquetes	public, private, protected si ninguno es especificado, las clases tienen un nivel de acceso interno(Semejante al nivel de acceso de paquetes en Java)
Nivel de acceso	No existe	Similar a los namespaces de XML.

personalizado: namespaces		namespace abc; abc function myCalc(){} ó abc::myCalc(){} use namespace abc;

Salida por consola	System.out.println();	// solo en modo de depuración
		
		trace();

Import	import com.abc.*; import com.abc.MyClass;	import com.abc.*; import com.abc.MyClass; Los paquetes deben ser importados incluso si los nombres de las clases son calificados completamente en el código
Pares desordenados de valores claves	Hashtable, Map Hashtable friends = new Hashtable(); friends.put(“good”, “Mary”); “Bill”); friends.put(“best”, “Bill”); friends.put(“bad”,	Arreglos asociativos Permite hacer referencia a los elementos por su nombre sin tener que usar los indexs. var friends:Array=new Array(); friends[“good”]=”Mary”; friends[“best”]=”Bill”; friends[“bad”]=”Masha”; var bestFriend:String= friends[“best”]; friends.best=”Alex”;

	“Masha”); String bestFriend= friends. get(“best”); // bestFriend = Bill	Otra sintaxis: var car:Object = {make:”Toyota”, model:”Camry”}; trace (car[“make”], car.model); // Salida: Toyota Camry
Hoisting	No existe	El compilador mueve todas las declaraciones de las variables a la parte superior de la función, de manera que se pueda usar el nombre de la función incluso antes de ser declarada en el código

Instanciar objetos desde clases	Customer cmr = new Cus-
	
	tomer();
	
	Class cls = Class.forName(“Custumer”); Object myObj= cls.newInstance();
		var cmr:Customer = new Customer();
		
		var cls:Class=flash.utils.getClassByName(“Custom er”);
		var myObj:Object=new cls();

Clases privadas	private class myClass{...}	No existen clases privadas en ActionScript 3
Cnstructores privados	Soportados. Uso típico: clases singleton	No disponible. La implementación de constructores privados está pospuesta ya que aún no forman parte del estándar ECMAScript.
Clases y nombres de archivos	Un fichero puede tener múltiples declaraciones de clases, pero sola una de ellas puede ser pública, y el fichero debe tener el mismo nombre que esta clase.	Un fichero puede tener múltiples declaraciones de clases, pero solo uno de ellos puede ser colocado dentro de la declaración del paquete, y el paquete debe tener el mismo nombre de esta clase.
¿Que podemos poner dentro de los paquetes?	Clases e interfaces	Clases, interfaces, variables, funciones, espacios de nombres reservados, y declaraciones ejecutables
Clases dinámicas(define un objeto que	No existen	dynamic class Person { var name:String;

puede ser modificado en tiempo de ejecución al agregarle o modificarle propiedades y métodos)		}
		
		//agregar una variable dinámicamente
		// y una función
		Person p= new Person();
		
		p.name=”Joe”;
		
		p.age=25;
		
		p.printMe = function () {
		
		trace (p.name, p.age);
		
		}
		
		p.printMe(); // Joe 25

Funciones de cierre	No existe. Es una propuesta para Java7	myButton.addEventListener(“click,” my- Method); Un cierre es un objeto que representa una instantánea de una función con su contexto léxico (valores de variables, objetos en el ámbito de aplicación). Una función de cierre puede ser pasada como argumento y ejecutada sin ser parte de cualquier objeto
Clases abstractas	Soportada	No hay ningún mecanismo para marcar los métodos de las clases con la palabra clave “abstract” que refuerce la aplicación de estos métodos en los descendientes
Interfaces	class A implements B{...} las interfaces pueden incluir declaraciones de métodos y variables finales	class A implements B{...} las interfaces solo pueden incluir declaraciones de funciones

Tratamiento de errores	Palabras claves: try, catch, throw, finally, throws	Palabras claves: try, catch, throw, finally Un método no tiene que declarar excepciones. Puede lanzar no solo objetos de tipo Error, sino números también: throw 25.3; Flash Player termina el script en caso de existir un error no capturado.
Expresiones regulares	Soportada	Soportada

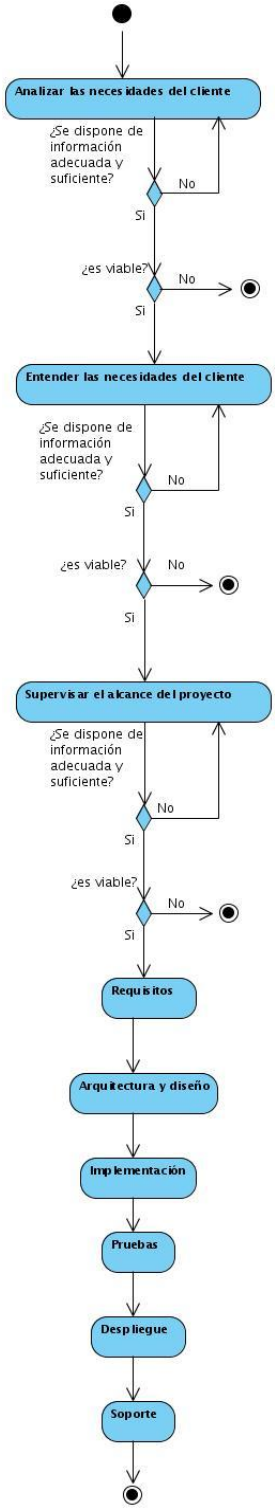
Anexo 11: Matriz de comparación entre los gestores de base de datos

Herramienta	Funcionalidad	Plataforma	Lenguaje	Licencia	Ventajas	Desventajas
PostgreSQL	-Permite la - Replicación asincrónica. -Soporta el uso de índices, reglas y vistas. -Permite la gestión de diferentes usuarios, como también los permisos asignados a cada uno de ellos. -Permite la declaración de funciones propias, así como la definición de disparadores. -Incluye herencia entre tablas (aunque no entre objetos, ya que no existen).	-Linux -Windows -Unix	- Tiene interfaces de programación originario de C / C + +, Java, Net, Perl, Python, Ruby, Tcl, ODBC, entre otros. - Se encuentra mayormente escrito en C.	-BSD	- Un excelente optimizador. -Incorpora una estructura de datos array. -Es altamente escalable. - Resiste el acaparamiento de grandes objetos binarios. -Implementa el uso de rollback's, subconsultas y transacciones, haciendo su funcionamiento mucho más eficaz. -Tiene la capacidad de comprobar la integridad referencial, así como también la de almacenar procedimientos en la propia base de datos	-La velocidad de respuesta que ofrece con base de datos relativamente pequeñas puede parecer un poco deficiente. -Consume gran cantidad de recursos. -Es de 2 a 3 veces más lento que MySQL.
MySQL	-Permite administrar base de datos. -Gestión de usuarios y passwords, manteniendo un muy buen nivel de seguridad en los datos.	-Linux -Windows -Mac OS X	-Está escrito en una mezcla de C y C++. -Permite acceder a la base de datos a través de varios lenguajes, C, C++, C#, Pascal, Delphi, etc.	-GPL	-Conexión segura. - Multihilo y multiusuario. -Rapidez y facilidad de uso. -Excelente rendimiento -Gran portabilidad entre sistemas. -Facilidad de configuración e instalación. -Tiene una probabilidad muy reducida de corromper los datos.	-Carece de soporte para transacciones, rollback's y subconsultas. -No es viable para su uso con grandes bases de datos, a las que se acceda continuamente, ya que no implementa una buena escalabilidad. -El hecho de que no maneje la integridad referencial, hace de este gestor una solución pobre para muchos campos de aplicación.

Anexo 12: Descripción Textual: Estudio preliminar

Nombre de la actividad	Artefactos de entrada	Tareas	Artefactos de salida	Rol	Lista de Chequeo	Bases tecnológicas
Analizar las necesidades del cliente	Documentos del cliente	Encontrar los términos comunes Determinar las fuentes de información Recoger información sobre el escenario a desarrollar Describir los elementos del escenario y la relación con el actor Evaluar los resultados Desarrollar glosario de términos, informe técnico y técnico	Plantilla DCS Informe técnico Plantilla DCS Proyecto técnico Plantilla DCS Glosario de términos	Analista Arquitecto Líder del proyecto	Lista de chequeo-Informe técnico Lista de chequeo-proyecto técnico Lista de Chequeo-Glosario de términos	Procesador de Texto
Entender las necesidades del cliente	Documentos del cliente Plantilla DCS Informe técnico Plantilla DCS Proyecto técnico Plantilla DCS Glosario de términos	Identificar las demandas del cliente Desarrollar las reglas de comportamiento de los elementos del escenario Determinar áreas a priorizar dentro del escenario o escena Manejar las dependencias entre los elementos Evaluar los resultados Refinar Informe y proyecto técnico	Plantilla DCS Informe técnico Plantilla DCS Proyecto técnico	Analista Arquitecto Líder del proyecto	Lista de chequeo-Informe técnico Lista de chequeo-Proyecto técnico	Procesador de Texto
Supervisar el alcance del proyecto	Documentos del cliente Plantilla DCS Informe técnico Plantilla DCS Proyecto técnico	Definir el contexto inicial del proyecto Analizar la arquitectura Analizar las tecnologías Decidir la estrategia para el desarrollo del proyecto Desarrollar documento visión Evaluar los resultados Actualizar informe y proyecto técnico.	Plantilla DCS Informe técnico Plantilla DCS Proyecto técnico Plantilla DCS Documento visión	Analista Arquitecto Líder del proyecto	Lista de chequeo-Informe técnico Lista de chequeo-proyecto técnico Lista de chequeo-Documento visión	Procesador de Texto

Anexo 13: Descripción Gráfica: Estudio preliminar

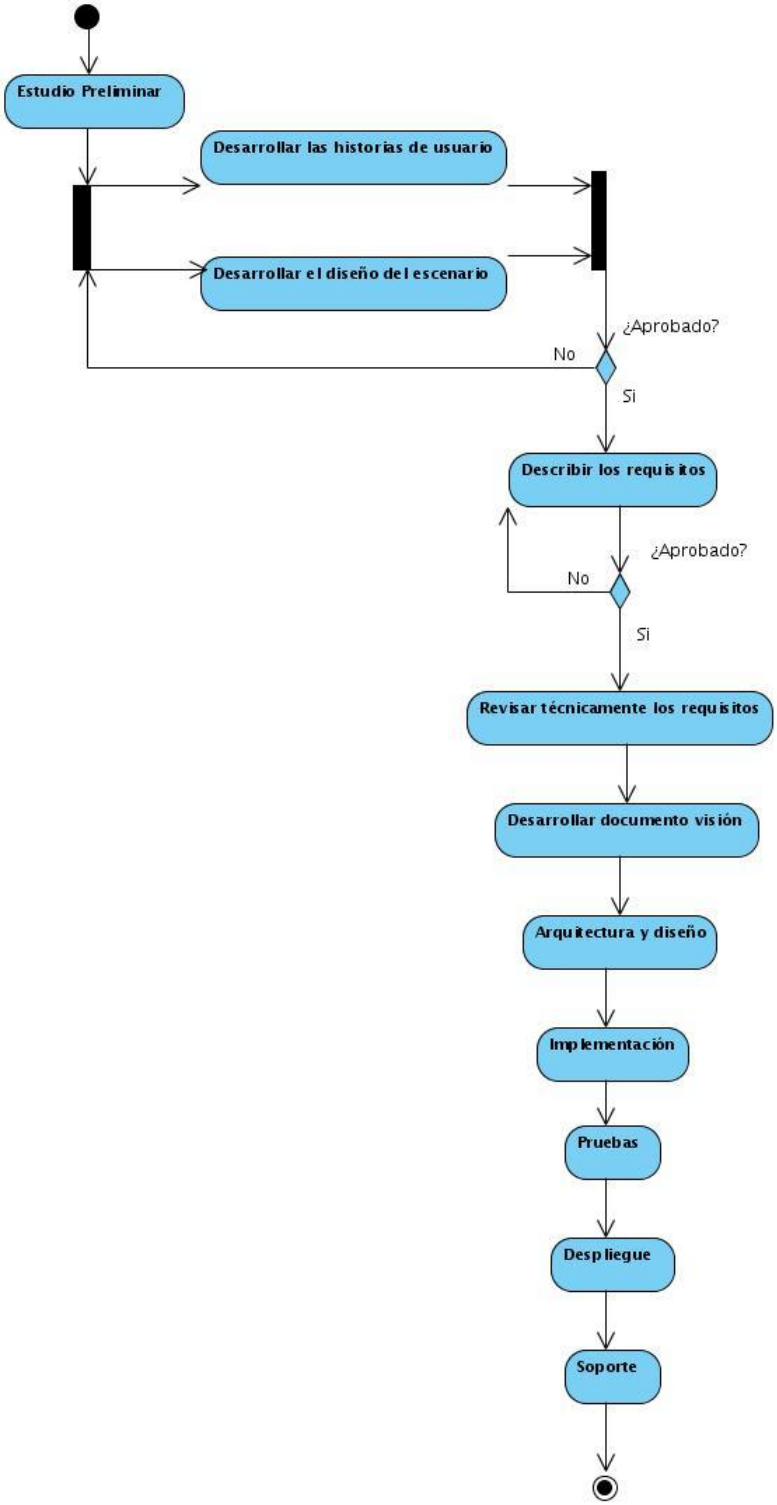


Anexo 14: Descripción Textual: Requisitos

Nombre de la actividad	Artefactos de entrada	Tareas	Artefactos de salida	Rol	Lista de Chequeo	Bases tecnológicas
Desarrollar el diseño del escenario	Plantilla DCS Informe técnico Plantilla DCS Proyecto técnico Plantilla DCS Documento visión Plantilla DCS Glosario de Términos	Recoger información Describir las características de los usuarios Identificar los elementos de las escenas. Definir la estructura de las escenas para conformar el escenario Diseñar los detalles de los elementos de las escenas Diseñar los prototipos de los escenarios Implementar los prototipos de los escenarios. Realizar Talleres para validar los prototipos Refinar prototipos	Plantilla DSC Historias de usuario (Prototipos)	Diseñador Gráfico	Lista de Chequeo-Historias de usuario	Procesador de Texto Blender
Desarrollar la historia de usuario	Plantilla DCS Informe técnico Plantilla DCS Proyecto técnico	Recoger información Describir relación entre actores y el escenario Revisar y refinar los escenarios Detallar el flujo de eventos Describir precondiciones Describir poscondiciones Describir puntos de extensión Realizar talleres para validar requisitos Priorizar los escenarios Refinar las historias de usuario	Plantilla DSC Historias de usuario Plantilla DCS Especificación de requisitos	Analistas	Lista de Chequeo-Historias de usuario Lista de Chequeo-Especificación de requisitos	Procesador de Texto OSRMT
Describir los requisitos	Plantilla DCS Informe técnico Plantilla DCS Proyecto técnico	Detallar los requisitos Identificar requerimientos comunes Organizar las historias de usuarios en paquetes Realizar talleres para validar los requisitos Refinar requisitos	Plantilla DCS Especificación de requisitos	Arquitecto	Lista de Chequeo-Especificación de requisitos	Procesador de Texto OSRMT
Revisar técnicamente los Requisitos	Plantilla DSC Historias de usuario Plantilla DCS Especificación de requisitos Lista de Chequeo-Historias de usuario Lista de Chequeo-Especificación de requisitos	Desarrollar reunión de revisión de los requisitos Preparar registro de revisión y defectos del documento Asignar responsabilidades para la solución de los defectos	Plantilla DCS No Conformidades	Revisor Técnico	Lista de Chequeo-No Conformidades	Procesador de Texto OSRMT

Desarrollar documento visión	Plantilla DCS Informe técnico Plantilla DCS Proyecto técnico Plantilla DCS Documento visión Plantilla DCS Glosario de Términos Plantilla DSC Historias de usuario Plantilla DCS Especificación de requisitos	Definir las interfaces del sistema. Identificar las restricciones que se imponen en el sistema. Formular declaraciones del problema. Definir características básicas del sistema Evaluar los resultados Refinar Documento Visión	Plantilla DCS Documento visión	Analista Arquitecto Líder del Proyecto	Lista de chequeo-Documento visión	Procesador de Texto

Anexo 15: Descripción Gráfica: Requisitos

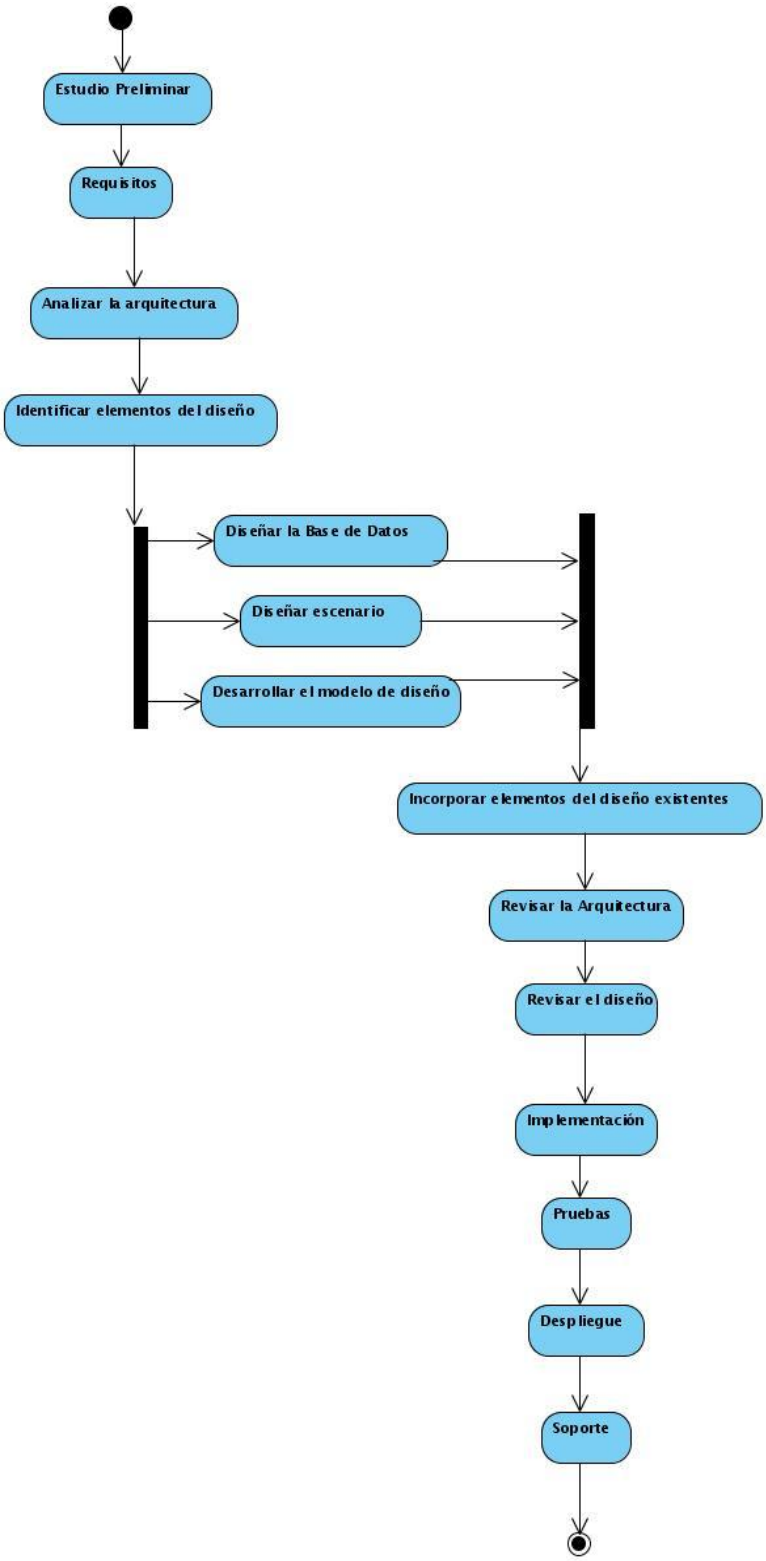


Anexo 16: Descripción Textual: Arquitectura y diseño

Nombre de la actividad	Artefactos de entrada	Tareas	Artefactos de salida	Rol	Lista de Chequeo	Bases tecnológicas
Analizar la arquitectura	Plantilla DSC Historias de usuario Plantilla DCS Especificación de requisitos	Desarrollar vista general de la arquitectura Estudiar recursos disponibles Definir la organización de los escenarios y las escenas Identificar interacciones entre escenas y escenarios Desarrollar vista general de despliegue Identificar mecanismos de diseño Revisar los resultados	Plantilla DCS Arquitectura de Software	Arquitecto	Lista de Chequeo-Arquitectura de Software	Procesador de Texto Visual Paradigm
Identificar elementos del diseño	Plantilla DCS Arquitectura de Software	Identificar escenas y escenarios. Identificar clases Identificar interfaces entre escenas y escenarios Revisar los resultados	Plantilla DCS Modelo de Diseño	Arquitecto	Lista de Chequeo-Modelo de Diseño	Procesador de Texto Visual Paradigm
Diseñar escenario	Plantilla DCS Arquitectura de Software Plantilla DCS Modelo de Diseño	Distribuir la conducta de las escenas Documentar las escenas Describir las dependencias entre escenas Revisar los resultados	Plantilla DCS Modelo de Diseño	Diseñador	Lista de Chequeo-Modelo de Diseño	Procesador de Texto Visual Paradigm
Desarrollar del modelo de Diseño	Plantilla DCS Arquitectura de Software Plantilla DCS Modelo de Diseño	Diseñar componentes Crear diseño de clases inicial Identificar las clases persistentes Definir la visibilidad de las clases Definir operaciones Definir los métodos Definir estados Definir atributos Definir dependencias Definir asociaciones Definir generalizaciones Definir la estructura interna. Resolver las colisiones Tratar los requisitos no funcionales en general Revisar los resultados	Plantilla DCS Modelo de Diseño	Diseñador	Lista de Chequeo-Modelo de Diseño	Procesador de Texto Visual Paradigm
Incorporar elementos del diseño existentes	Plantilla DCS Arquitectura de Software Plantilla DCS Modelo de Diseño	Identificar oportunidades de re-uso Actualizar la organización del modelo de diseño Revisar los resultados	Plantilla DCS Modelo de Diseño	Arquitecto	Lista de Chequeo-Modelo de Diseño	Procesador de Texto Visual Paradigm
Diseñar la Base de Datos	Plantilla DCS Modelo de Diseño	Desarrollar el modelo de datos lógicos Desarrollar el diseño físico de la base de datos Definir el dominio Crear elementos iniciales del diseño físico de la base de	Plantilla DCS Modelo de Diseño	Diseñador de BD	Lista de Chequeo-Modelo de Diseño	Procesador de Texto Visual Paradigm

		datos Definir las tablas de referencia Crear llave primaria y las restricciones Definir datos y entrada en vigor de las reglas Normalizar el diseño de la base de datos Optimizar el rendimiento. Optimizar el acceso a datos Definir las características de almacenamiento Diseñar los procedimientos para almacenar las clases de conducta en la base de datos Revisar resultados				
Revisar la Arquitectura	Plantilla DCS Arquitectura de Software Lista de Chequeo-Arquitectura de Software	Desarrollar las reuniones de revisión de la arquitectura Preparar registro de revisión y defectos del documento Asignar responsabilidades para la solución de los defectos	Plantilla DCS No Conformidades	Revisor Técnico	Lista de Chequeo-No Conformidades	Procesador de Texto Visual Paradigm
Revisar el diseño	Plantilla DCS Modelo de Diseño Lista de Chequeo-Modelo de Diseño	Revisar cada elemento del diseño Revisar el cumplimiento de las pautas de diseño Preparar registro de revisión y de defectos del documento Asignar responsabilidades para la solución de los defectos	Plantilla DCS No Conformidades	Revisor Técnico	Lista de Chequeo-No Conformidades	Procesador de Texto Visual Paradigm

Anexo 17: Descripción Gráfica: Arquitectura y diseño

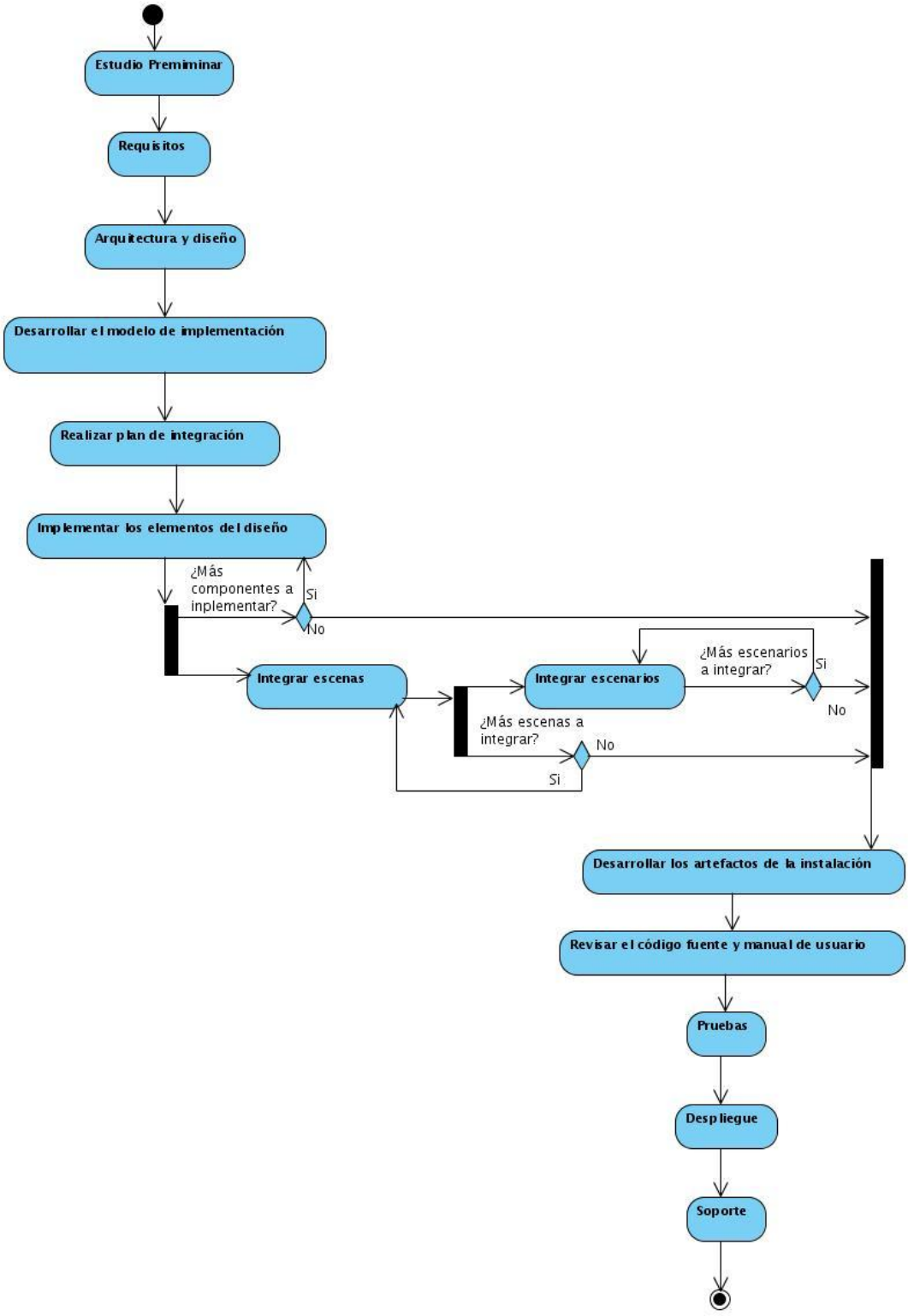


Anexo 18: Descripción Textual: Implementación

Nombre de la actividad	Artefactos de entrada	Tareas	Artefactos de salida	Rol	Lista de Chequeo	Bases tecnológicas
Desarrollar el modelo de implementación	Plantilla DCS Arquitectura de Software Plantilla DCS Modelo de Diseño	Establecer la estructura del modelo de implementación Ajustar las escenas y escenarios Definir las importaciones para cada escena y escenarios de implementación Decidir el tratamiento de los ejecutables Actualizar la vista de implementación Evaluar los resultados	Plantilla DCS Arquitectura de Software	Arquitecto	Lista de Chequeo-Arquitectura de Software	Procesador de texto Visual Paradigm
Realizar plan de integración	Plantilla DCS Arquitectura de Software	Identificar escenarios y escenas Definir cambios de construcción Definir series de construcción Evaluar la integración del plan de construcción Evaluar los resultados	Plantilla DCS Arquitectura de Software	Arquitecto	Lista de Chequeo-Arquitectura de Software	Procesador de texto Visual Paradigm
Implementar los elementos de diseño	Plantilla DSC Historias de usuario Plantilla DCS Especificación de requisitos Plantilla DCS Modelo de Diseño	Preparar la implementación Transformar diseño en implementación Completar la implementación Evaluar la implementación Proporcionar información de retorno al diseño	Código Fuente	Programador	Lista de Chequeo-Producto	Procesador de Texto Eclipse
Integrar escenas	Plantilla DSC Historias de usuario Plantilla DCS Especificación de requisitos Plantilla DCS Modelo de Diseño Código Fuente	Integrar las escenas Entregar el escenario implementado	Código Fuente	Arquitecto	Lista de Chequeo-Producto	Procesador de Texto Eclipse
Integrar escenarios	Plantilla DSC Historias de usuario Plantilla DCS Especificación de requisitos Plantilla DCS Modelo de Diseño Código Fuente	Aceptar los escenarios Integrar los escenarios Promover la línea base de la arquitectura	Código Fuente	Arquitecto	Lista de Chequeo-Producto	Procesador de Texto Eclipse
Desarrollar los artefactos de la instalación	Plantilla DSC Historias de usuario Plantilla DCS Especificación de requisitos Plantilla DCS Modelo de Diseño Código Fuente	Construir el material de apoyo de usuario	Manual de usuario	Analista	Lista de Chequeo-Manual de Usuario	Procesador de Texto Eclipse

Revisar el código fuente y manual de usuario	Plantilla DSC Historias de usuario Plantilla DCS Especificación de requisitos Plantilla DCS Modelo de Diseño Código Fuente	Establecer puntos de control para la implementación Revisar las escenas y escenarios Preparar registro de revisión y de defectos del documento Asignar responsabilidades para la solución de los defectos	Plantilla DCS No Conformidades	Revisor Técnico	Lista de Chequeo-No Conformidades	Procesador de Texto Eclipse

Anexo 19: Descripción Gráfica: Implementación

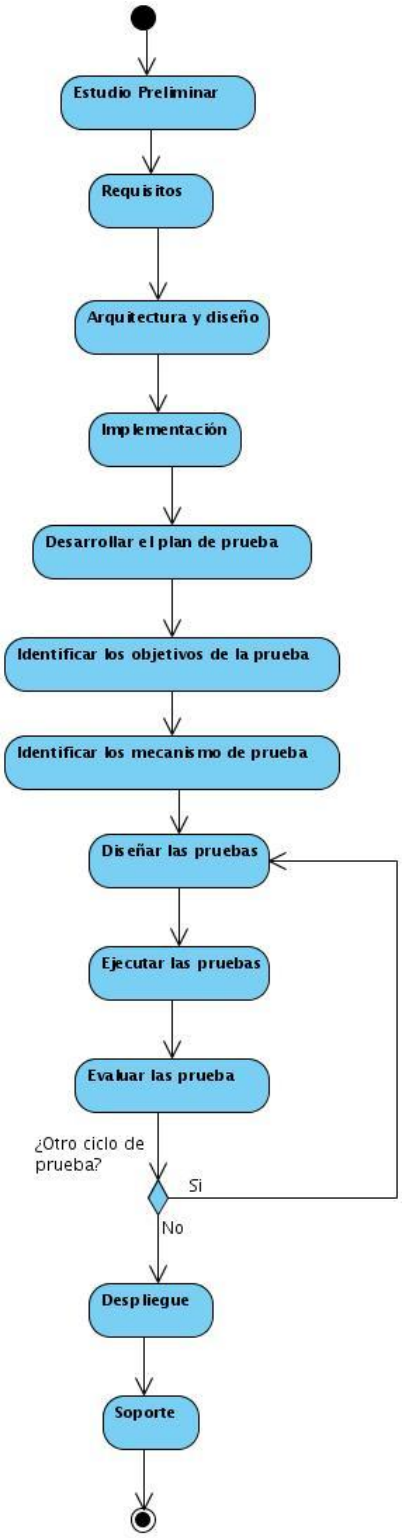


Anexo 20: Descripción Textual: Pruebas

Nombre de la actividad	Artefactos de entrada	Tareas	Artefactos de salida	Rol	Lista de Chequeo	Bases tecnológicas
Desarrollar el plan de pruebas	Plantilla DSC Historias de usuario Plantilla DCS Especificación de requisitos Plantilla DCS Modelo de Diseño Código Fuente	Identificar los elementos a probar Formular la declaración de la misión de prueba Identificar los niveles, tipos y métodos de prueba	Plantilla DCS Plan de pruebas	Diseñador de prueba	Lista de Chequeo-Plan de pruebas	Procesador de Texto Eclipse
Identificar los objetivos de la prueba	Plantilla DCS Plan de pruebas	Identificar los elementos dinámicos y eventos del sistema Identificar los elementos de infraestructura de prueba Identificar las necesidades del plan especificaciones-pruebas Definir los requisitos de prueba del software Definir la infraestructura de la prueba Evaluar y verificar los resultados	Plantilla DCS Plan de pruebas	Diseñador de prueba	Lista de Chequeo-Plan de pruebas	Procesador de Texto Eclipse
Identificar los mecanismo de prueba	Plantilla DCS Plan de pruebas Plantilla DCS Arquitectura de Software	Examinar la arquitectura del software y designar sus ambientes Identificar los mecanismos de prueba Definir los mecanismos de prueba a usar Evaluar y verificar los resultados	Plantilla DCS Plan de pruebas	Diseñador de prueba	Lista de Chequeo-Plan de pruebas	Procesador de Texto Eclipse
Diseñar las pruebas	Plantilla DCS Plan de pruebas Plantilla-Historia de Usuarios Código Fuente	Identificar las condiciones de la entrada, rendimiento y ejecución Identificar puntos de observación Identificar puntos de control Identificar los métodos apropiados Definir fuentes de datos requeridas, valores y	Plantilla DCS Diseño casos de prueba	Diseñador de prueba	Lista de Chequeo-Diseño casos de pruebas	Procesador de Texto

		rangos Asignar los recursos suficientes para realizar las pruebas Mantener las relaciones de traceabilidad Evaluar y verificar los resultados				
Ejecutar la pruebas	Plantilla DCS Plan de pruebas Plantilla-Historia de Usuarios Plantilla DCS Diseño casos de prueba Código Fuente	Seleccionar la técnica de implementación apropiada Preparar las condiciones previas de ambiente de prueba Implementar la prueba Establecer los juegos de datos externos Verificar la implementación de la prueba Restaurar el ambiente de prueba en un estado conocido Mantener las relaciones de traceabilidad Evaluar y verificar los resultados	Plantilla DCS No Conformidades	Probador	Lista de Chequeo-No Conformidades	Procesador de Texto Visual Paradigm
Evaluar las prueba	Plantilla DCS No Conformidades	Examinar todas las incidencias de las pruebas Crear las solicitudes de cambio Analizar y evaluar el estado de la aplicación Hacer una valoración de la experiencia del desarrollo Hacer una valoración de riesgos técnicos Hacer una valoración de los resultados de las pruebas Evaluar y verificar sus resultados	Plantilla DCS Minuta de reunión Plantilla DCS Solicitud de cambio	Analista, Arquitecto Líder de proyecto	Lista de Chequeo-Minuta de reunión Lista de Chequeo-Solicitud de cambio	Procesador de Texto

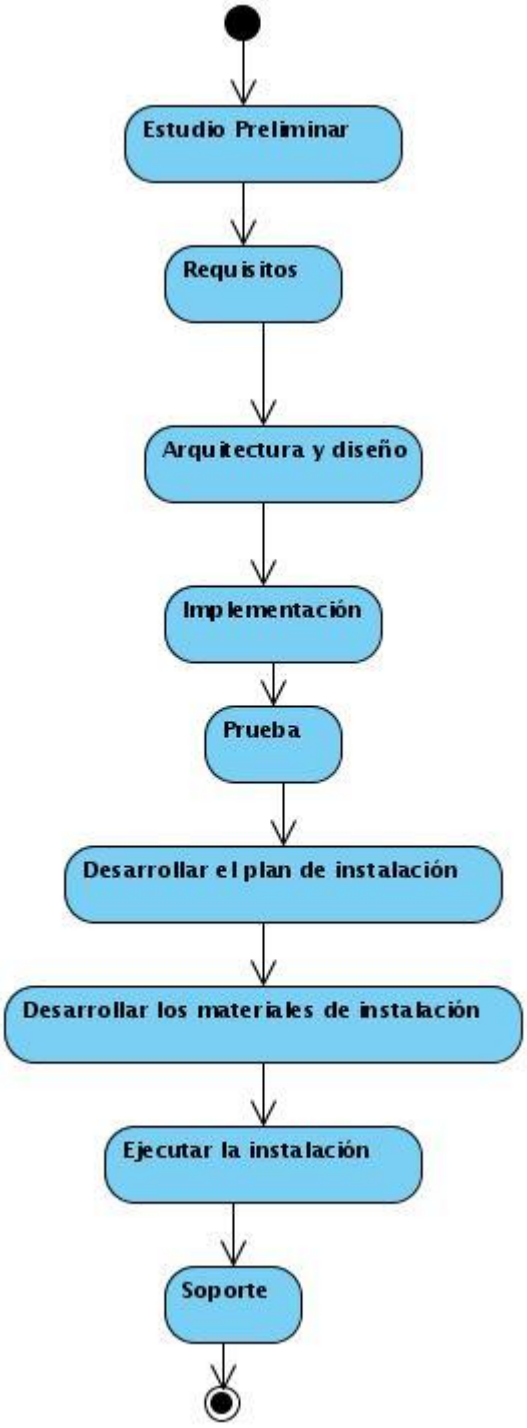
Anexo 21: Descripción Gráfica: Pruebas



Anexo 22: Descripción Textual: Despliegue

Nombre de la actividad	Artefactos de entrada	Tareas	Artefactos de salida	Rol	Lista de Chequeo	Bases tecnológicas
Desarrollar el plan de instalación	Plantilla DCS Arquitectura de Software	Planificar cómo empaquetar el software Planificar cómo distribuir el software Planificar cómo instalar el software Proporcionar ayuda y asistencia a los usuarios	Plantilla DCS Arquitectura de Software	Arquitecto	Lista de Chequeo-Arquitectura de Software	Procesador de Texto
Desarrollar los materiales de instalación	Plantilla DCS Arquitectura de Software	Desarrollar los scripts de instalación. Desarrollar los archivos de configuración Desarrollar instrucciones de instalación Desarrollar un esquema de los materiales de formación Escribir los materiales de formación Producir un plan de alto nivel para la información que se va a presentar Revisar los resultados	Materiales de apoyo	Analistas Programador		Procesador de Texto Eclipse
Ejecutar la instalación	Materiales de apoyo	Ejecutar la instalación Preparar el ambiente para las pruebas de aceptación Realizar las pruebas de aceptación Preparar registro de revisión y de defectos del documento Asignar responsabilidades para la solución de los defectos	Plantilla ALBET Acta de Aceptación Plantilla DCS No Conformidades	Arquitecto Analista Líder del proyecto Programadores Probadores		Procesador de Texto Eclipse

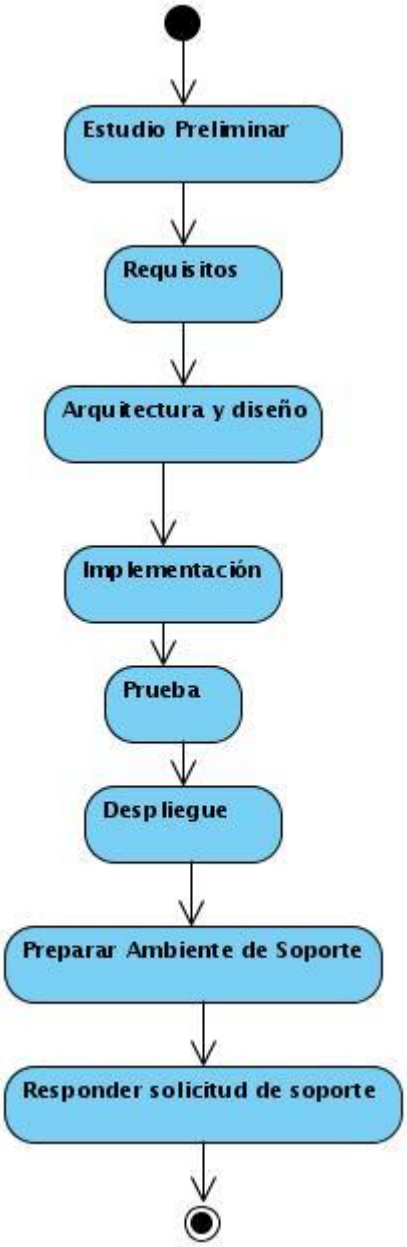
Anexo 23: Descripción Gráfica: Despliegue



Anexo 24: Descripción Textual: Soporte

Nombre de la actividad	Artefactos de entrada	Tareas	Artefactos de salida	Rol	Lista de Chequeo	Bases tecnológicas
Preparar Ambiente de Soporte	Código Fuente Plantilla DCS-Manual de Usuario	Preparar la base de conocimiento para el soporte Preparar especialistas para brindar el soporte por el centro de llamadas Preparar el ambiente para recibir solicitudes de soporte en el cliente	Plan de Soporte Base de conocimiento	Ingeniero de Soporte		Procesador de Texto
Responder solicitud de soporte	Plantilla DCS Solicitud de soporte	Evaluar solicitud de soporte Responder solicitud Recoger errores o inconformidades de los usuarios con el producto Actualizar la base de conocimiento	Lista de Solicitudes de mejora Lista de Lecciones aprendidas	Ingeniero de soporte		Procesador de texto

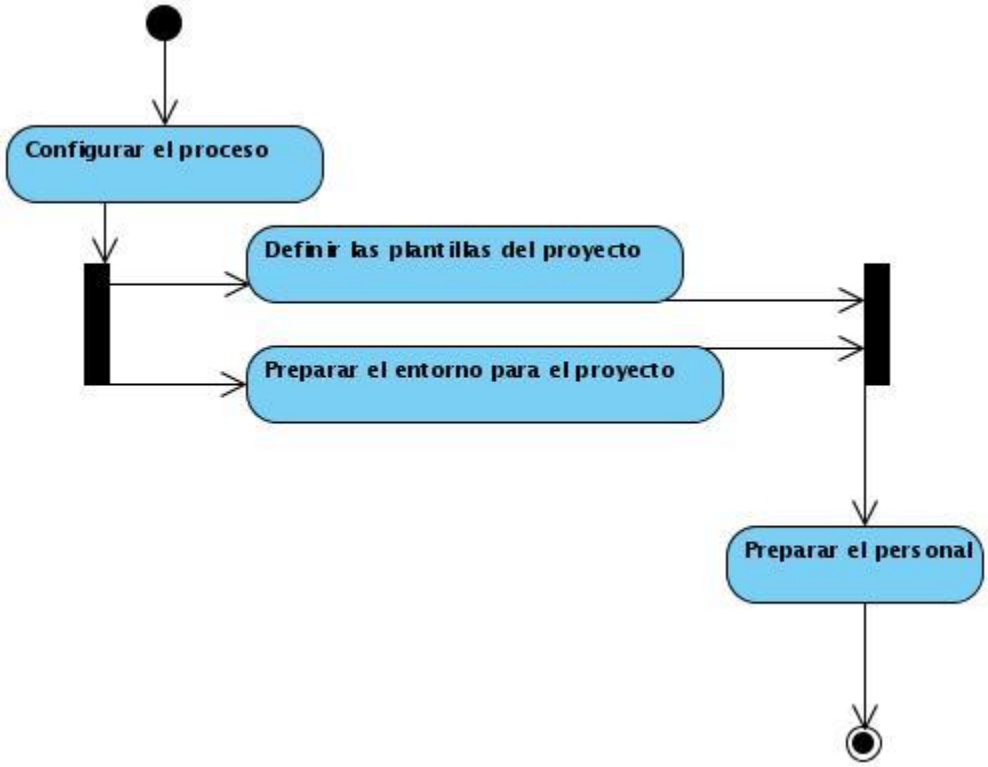
Anexo 25: Descripción Gráfica: Soporte



Anexo 26: Descripción Textual: Ambiente

Nombre de la actividad	Artefactos de entrada	Tareas	Artefactos de salida	Rol	Lista de Chequeo	Bases tecnológicas
Configurar el proceso	Plantilla DCS Documento Visión	Analizar el proyecto Configurar los procesos Preparar los procesos para el proyecto Preparar materiales de capacitación	Plantilla DCS Ambiente de desarrollo Plantilla DCS Plan de capacitación Materiales de Capacitación	Ingeniero de Procesos	Lista de Chequeo-Ambiente de desarrollo	Procesador de Texto
Definir las plantillas del proyecto	Plantilla DCS Ambiente de desarrollo Plantilla DCS Plan de capacitación	Preparar las plantillas Preparar materiales de capacitación	Plantilla DCS Ambiente de desarrollo Plantilla DCS Plan de capacitación Materiales de capacitación	Ingeniero de Procesos	Lista de Chequeo-Ambiente de desarrollo	Procesador de Texto
Preparar el entorno para el proyecto	Plantilla DCS Ambiente de desarrollo	Instalar las herramientas en el servidor Personalizar las herramientas en el servidor Integrar con las herramientas Instalar y personalizar las herramientas para los clientes		Ingeniero de Procesos		Procesador de Texto
Preparar al personal	Plantilla DCS Plan de capacitación Materiales de Capacitación	Capacitar a los miembros del proyecto Informar los cambios del proceso y las herramientas		Ingeniero de Procesos		Procesador de Texto

Anexo 27: Descripción Gráfica: Ambiente

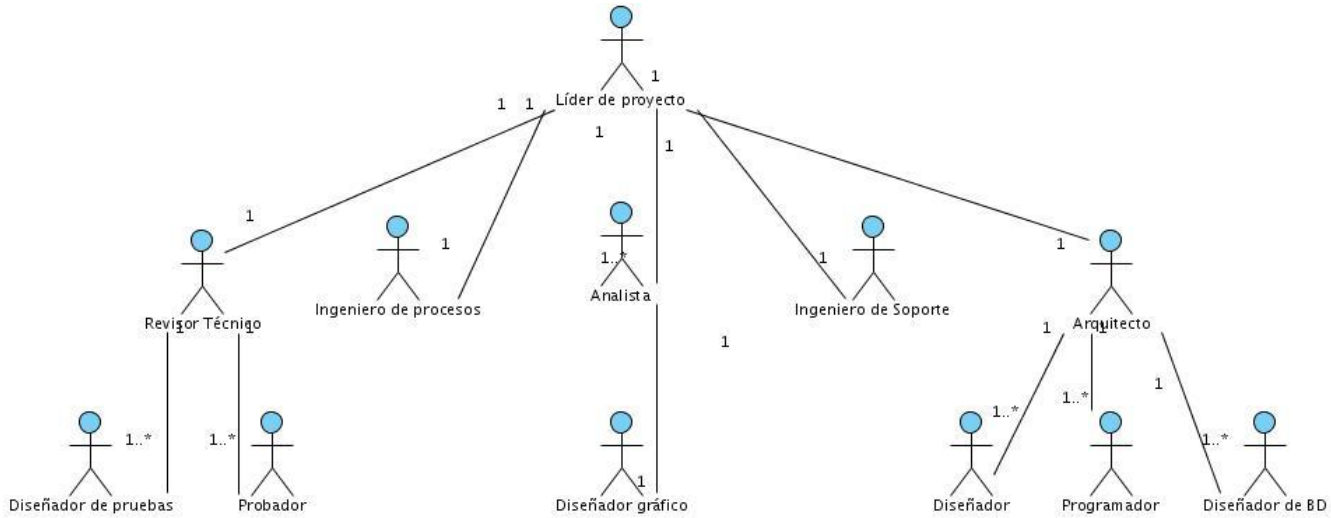


Anexo 28: Roles y Responsabilidades

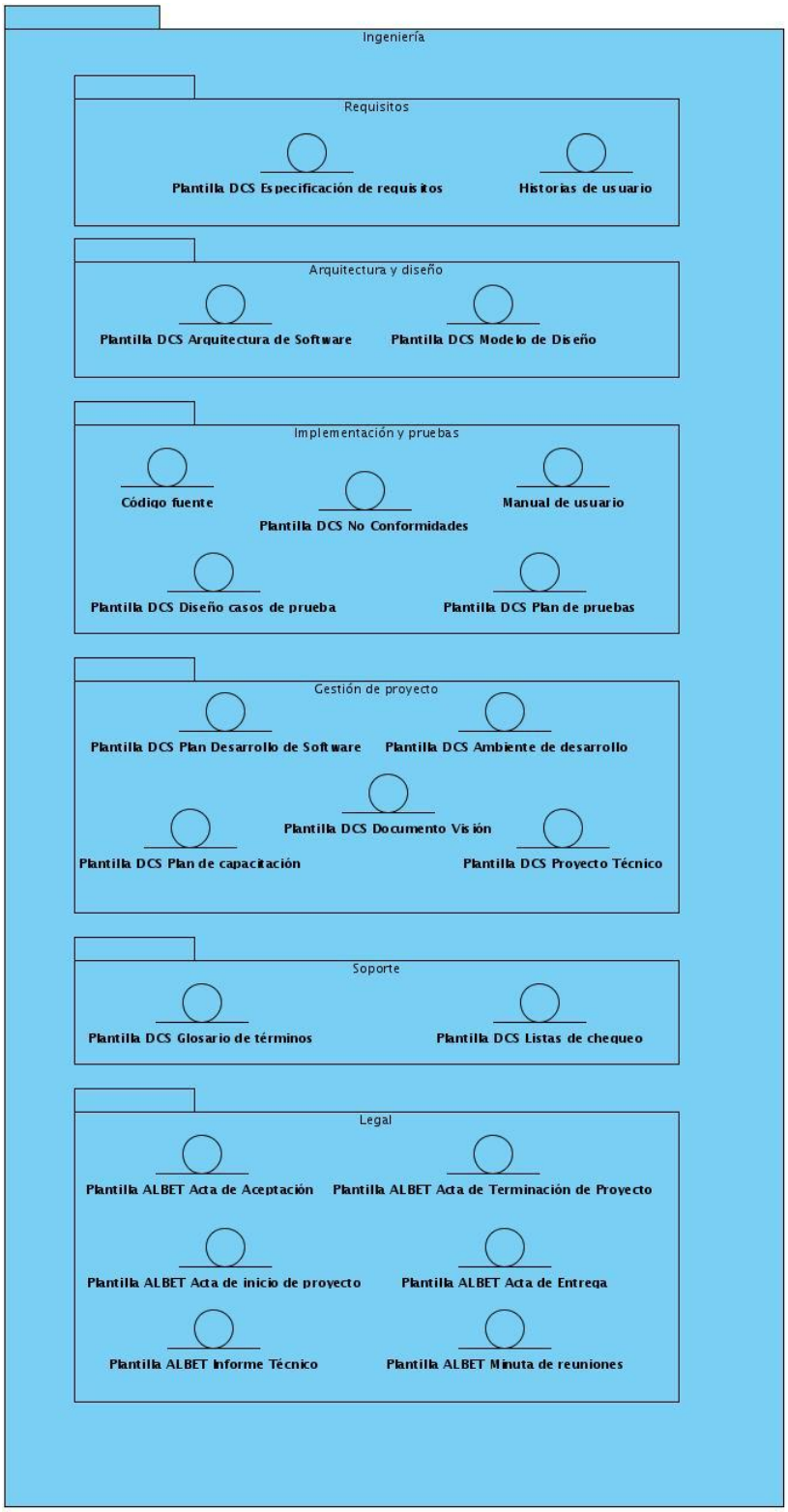
Rol	Responsabilidad
Analista	Participa con el cliente y el usuario final recogiendo necesidades y requerimientos. Captura los requisitos y define las prioridades. Realiza las Historia de usuario. Participa en el diseño de la solución y de las pruebas.
Arquitecto de software	Define todos los elementos bases de la arquitectura del proyecto Identifica todos los posibles escenarios de despliegue de la aplicación Identifica componentes horizontales de la aplicación Determina de conjunto con los diseñadores las interfaces de integración tanto internas como externas Elabora el documento de arquitectura para el expediente de proyecto Define las herramientas, bibliotecas, componentes, Frameworks y otros componentes que permitan acelerar y mejorar el trabajo del proyecto Define de conjunto con el jefe de proyecto el flujo de desarrollo basado en las herramientas identificadas Vela por el cumplimiento de los requerimientos de hardware Planifica la integración de los componentes del sistema
Diseñador de BD	Define la lógica de la base de datos Encargado de modelar e implementar la base de datos del sistema Define el gestor de base de datos a usar Define la herramienta de modelado para bases de datos relacionales Define las políticas de cambio sobre los elementos de datos Define los algoritmos de réplica, sincronización, respaldo, recuperación de la base de datos Define las políticas de almacenamiento de los datos Define las políticas de uso de los diferentes objetos de bases de datos ante situaciones particulares Usa el diagrama entidad-relación para generar el diseño físico de la base de datos Crea y mantiene el ambiente de la base de datos para el funcionamiento de la aplicación. Interviene en el ajuste del desempeño de la aplicación Administra gestor de BD Configura roles y usuarios Administra BD
Diseñador gráfico	Encargado de realizar todo el diseño gráfico que requiera el sistema Interviene en la creación del prototipo Define las pautas para el diseño gráfico.
Diseñador de pruebas	Participa en la confección de la estrategia y el plan de pruebas Identifica los métodos, las técnicas, herramientas y directrices apropiadas para implementar las pruebas necesarias Establece en ambiente de comprobación Encargado de diseñar casos de pruebas para el sistema Dirige la definición del enfoque de prueba y garantiza la implementación satisfactoria Diseña las pruebas Genera los casos y datos de prueba Puede fungir como probador Analiza y evalúa el estado del producto de trabajo comprobado
Diseñador	Traduce los requerimientos identificados por los analistas hacia la tecnología utilizada para desarrollar en el proyecto Vela que se cumplan todas las orientaciones del arquitecto con respecto al flujo de trabajo y las herramientas a utilizar Participa en la concepción del sistema a desarrollar de conjunto con los analistas y arquitectos En dependencia del tamaño del proyecto también podría encargarse del diseño de la BD sino esta tarea la llevaría otra persona

Ingeniero de procesos	Realiza el diagnóstico y evaluación de la organización o área que se va a trabajar Identifica posibles áreas de mejora Determina y diseñar los procesos de la organización o área Evalúa y propone mejoras al proceso de desarrollo de software que está llevando el proyecto
Ingeniero de Soporte	Diseñar el ambiente de soporte Generar los materiales necesarios para el soporte Analizar las solicitudes de soporte Responder las solicitudes de soporte
Líder de proyecto	Administra recursos Controla Recursos Monitorea la adherencia a procesos Participa en las RTF Participa en las revisiones con el cliente de los entregables Administra la capacitación interna al proyecto Participa en la elaboración de la visión general del proyecto Guía el proceso de identificación y mitigación de los riesgos Evalúa a los miembros del proyecto según su desempeño Desarrolla el plan de proyecto o de desarrollo de software Gestiona las interacciones con clientes y usuarios
Probador	Encargado de seguir los planes de pruebas Ejecuta los casos de prueba y genera no conformidades asociadas al mismo Registra los resultados de las pruebas Analiza los resultados de las pruebas realizadas
Programador	Convierte la especificación del sistema en código fuente ejecutable Desarrolla el diseño teniendo en cuenta la arquitectura Elabora las pruebas de unidad Desarrolla el prototipo de la interfaz de usuario Integra los componentes que forman parte de la escena Ejecuta los casos de prueba y generar no conformidades asociadas al mismo Registra y analiza los resultados de las pruebas
Revisor Técnico	Ejecutar las actividades de revisión Registra los resultados de la revisión Analiza los resultados de la revisión

Anexo 29: Estructura organizacional



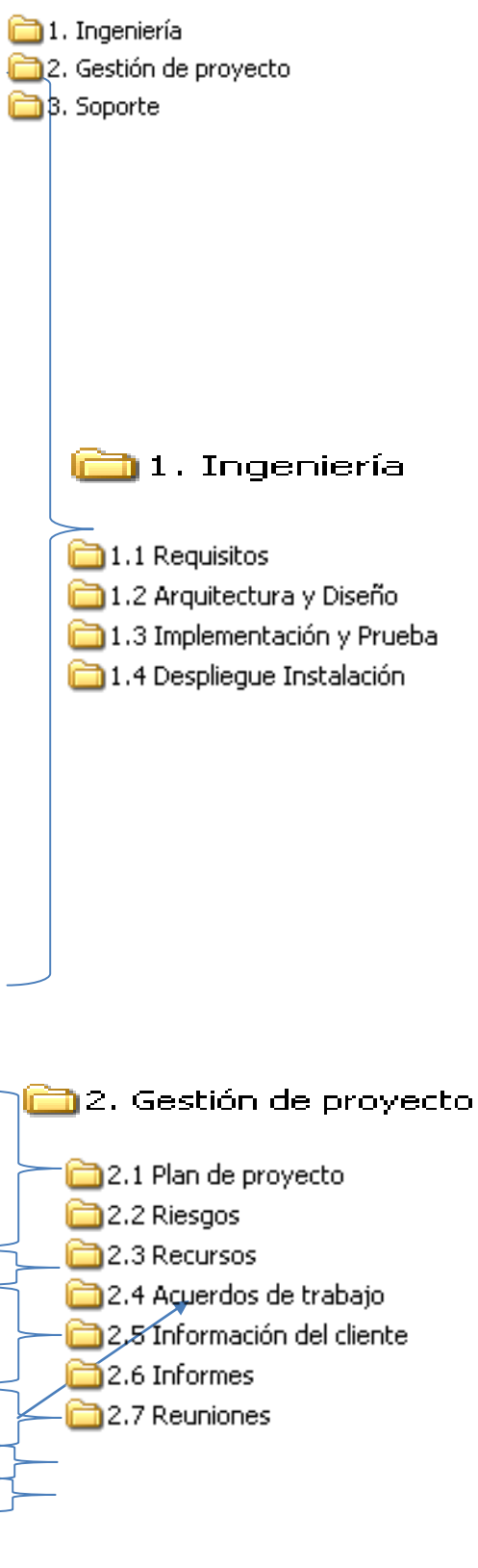
Anexo 30: Artefactos



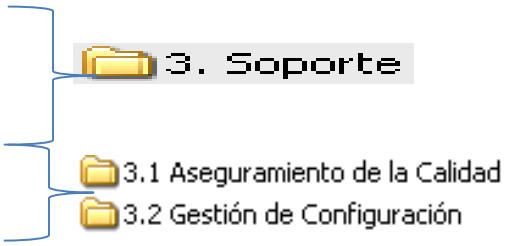
Anexo 31: Artefactos vs. Expediente de proyecto

Plantilla/ Face
Ingeniería
Requisitos
Plantilla DCS Diagrama de Proceso-Nombre del Proceso
Plantilla DCS Especificación de requisitos
Plantilla DCS Evaluación de Áreas de la Organización
Plantilla DCS Modelo de Casos de uso del sistema
Plantilla DCS Modelo del Dominio
Plantilla DCS Modelo del Negocio
Plantilla DCS Plan de gestión de requisitos
Historias de usuario
IDEF0 Diagram Shapes
Modelo de negocio con BPM
Arquitectura y diseño
Plantilla DCS Arquitectura de Información
Plantilla DCS Arquitectura de Software
Plantilla DCS Informe del Levantamiento de Información para la Arquitectura de Información
Plantilla DCS Modelo de Diseño
Implementación y pruebas
Código fuente
Manual de usuario
Plantilla DCS Casos de prueba
Plantilla DCS Diseño casos de prueba
Plantilla DCS Plan de pruebas
Plantilla DCS No Conformidades
Plantilla DCS Registro de prueba Unitaria o Integración
Despliegue e instalación
Plantilla DCS Modelo de Despliegue
Gestión de proyecto

Plantilla DCS Plan Desarrollo de Software
Plantilla DCS Presupuesto
Plantilla DCS Diario de Actividades
Plantilla DCS Pila de Sprint (Sprint Backlog)
Plantilla DCS Registro Tarea de ingeniería
Plantilla DCS Plan Mitigación de Riesgos
Plantilla DCS Ambiente de desarrollo
Plantilla DCS Plan de capacitación
Plantilla DCS Roles y responsabilidad
Plantilla DCS Documento Visión
Plantilla DCS Proyecto Técnico
Plantilla DCS Diagnóstico
Plantilla DCS Minuta de reunión



Soporte
Plantilla DCS Glosario de términos
Plantilla DCS Listas de chequeo
Plantilla DCS Plan aseguramiento de la calidad
Plantilla DCS Plan de mediciones
Plantilla DCS Solicitud de cambio
Plantilla DCS Pedido de cambio
Plantilla DCS Plan Gestión de Configuración
Legal
Plantilla ALBET Acta de Aceptación
Plantilla ALBET Acta de Entrega
Plantilla ALBET Acta de inicio de proyecto
Plantilla ALBET Acta de Terminación de Proyecto
Plantilla ALBET – Carta
Plantilla ALBET – Indefiniciones
Plantilla ALBET Informe Técnico
Plantilla ALBET Minuta de reuniones
Plantilla ALBET Proyectos Técnicos



Anexo 32: Especificación de requisitos

Introducción

Este documento describe la aplicación Tienda Virtual en requisitos funcionales.

Propósito

Tiene como propósito describir la aplicación en términos de requisitos. Un requisito no es más que una condición o capacidad que tiene que tener un sistema para satisfacer al cliente. Los requisitos funcionales son el conjunto de capacidades o funciones que el sistema debe cumplir. Los requisitos no funcionales son las propiedades o cualidades que el producto debe tener y representan las características del producto.

Alcance

Describir el alcance y propósito de la herramienta.

Enumerar los requisitos funcionales.

Los requisitos no funcionales no se describen pues coinciden con los establecidos por la arquitectura de software de dominio específico.

Definiciones, Acrónimos y Abreviaturas

Entidad: modelo 3D que se pone a disposición del cliente.

Pedido: uno o varias entidades solicitados por el cliente.

Factura: refleja las líneas de pedidos solicitadas por el cliente y el tamaño de cada uno y el total

Cliente: Persona que utiliza el servicio de compra de la tienda virtual.

Referencias

Código	Título
[2]	HU-Tienda Virtual

Funcionalidad

La tienda virtual tiene la posibilidad de mostrar un grupo de entidades, de las cuales se brinda la vista 3D y una descripción correspondiente. Si el usuario lo desea, la entidad seleccionada puede incorporarse a un pedido. Terminado el pedido, el sistema ofrece la opción de enviar la factura, pidiendo los datos del usuario para realizar esta operación.

1. Mostrar entidades

Se muestran las entidades en una vista 3D.

2. Mostrar descripción

Se muestra la descripción de la entidad seleccionada por el cliente, de la cual se brindan los siguientes datos: Nombre, Descripción, Tamaño, Cantidad a solicitar.

3. Interactuar con la vista en 3D

Se muestra la vista 3D, permite al cliente la interacción a través del ratón.

4. Gestionar pedido

Gestionar pedido incluye:

- Insertar nuevo pedido
- Eliminar pedido

De los pedidos se registran los siguientes datos: Nombre, Cantidad a solicitar, Tamaño, Tamaño total del pedido.

5. Mostrar factura

Se muestra la factura del pedido realizado por el cliente, se muestran los datos del pedido y se solicitan los siguientes datos: Nombre del cliente, Apellidos del cliente y Código de la Tarjeta de Crédito.

6. Enviar Factura

Termina la aplicación.

Anexo 33: Historias de usuario

Introducción

Este documento describe la aplicación Tienda Virtual en historias de usuario.

Propósito

Tiene como propósito describir la aplicación describiendo las historias de usuario, indicando quiénes son los actores del sistema y por qué lo son.

Los actores no son ninguna parte del sistema, ellos representan a cualquiera o algo que debe interactuar con el sistema. Un actor puede que:

Sólo brinde información de entrada al sistema.

Sólo reciba la información del sistema.

Brinde y reciba información.

Las historias de usuario son procesos que responden a las funcionalidades definidas en los requerimientos funcionales.

Alcance

Describir el alcance y propósito de la aplicación Tienda Virtual.

Describir cada una de las funcionalidades.

Definiciones, Acrónimos y Abreviaturas

Entidad: modelo 3D que se pone a disposición del cliente.

Pedido: uno o varias entidades solicitados por el cliente.

Factura: refleja las líneas de pedidos solicitadas por el cliente y el tamaño de cada uno y el total

Cliente: Persona que utiliza el servicio de compra de la tienda virtual.

Referencias

Código	Título
[1]	ER-Tienda Virtual

Especificación de Historias de Usuario

Realizar pedido

Historia de Usuario	
Número: 1	**Usuario: Cliente**
Nombre historia : Realizar pedido	
Prioridad en negocio: crítico	**Riesgo en desarrollo: alto**
Puntos estimados: 5	**Iteración asignada: 1**
Programador responsable: Gilberto Cao Tarrero	

Descripción:

El sistema muestra al cliente tres vistas en la interfaz.

La vista 1, Vista 3D: muestra una vista 3D de la entidad seleccionada. Al cliente navegar por las entidades, se activa la vista 2.

La vista 2, Detalles: muestra los detalles de la entidad seleccionada por el cliente, con los datos: Nombre, Descripción, Tamaño, y el dato Cantidad a solicitar, con el valor 1 por defecto. El cliente puede modificar la Cantidad a solicitar y añadir una línea de pedido, que se registraría en la vista 3.

La vista 3, Pedido: muestra el pedido, con los siguientes datos: Nombre, Cantidad a solicitar, Tamaño, Tamaño total del pedido. En la medida que se solicite añadir entidades en la vista tres, se incorporaran elementos a este pedido. El cliente tiene la

opción de borrar una línea de pedidos de la lista de pedidos y se actualiza esta y el tamaño total se recalcula, o el cliente puede solicitar la factura.

Si el cliente selecciona la factura, se muestra una nueva interfaz Datos del Envío solicitando los datos para llenar la factura: Nombre del cliente, Apellidos del cliente, Código de la Tarjeta de Crédito.

Se muestran los datos del pedido y se da la opción de volver para modificar la lista de pedidos o enviar la factura.

Observaciones:

La funcionalidad que se describe es el núcleo de la tienda virtual.

Prototipo de Interfaz Gráfica

Interfaz 1

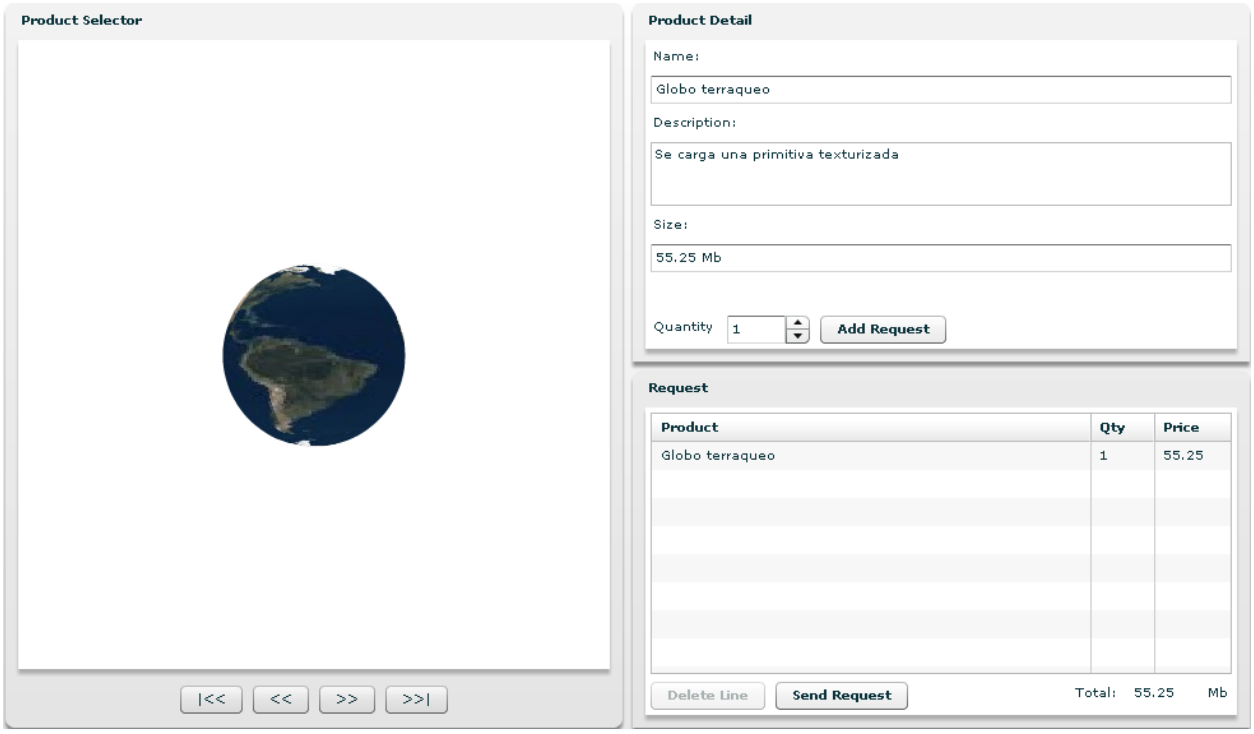


Figura 1. Interfaz gráfica inicial.

Interfaz 2

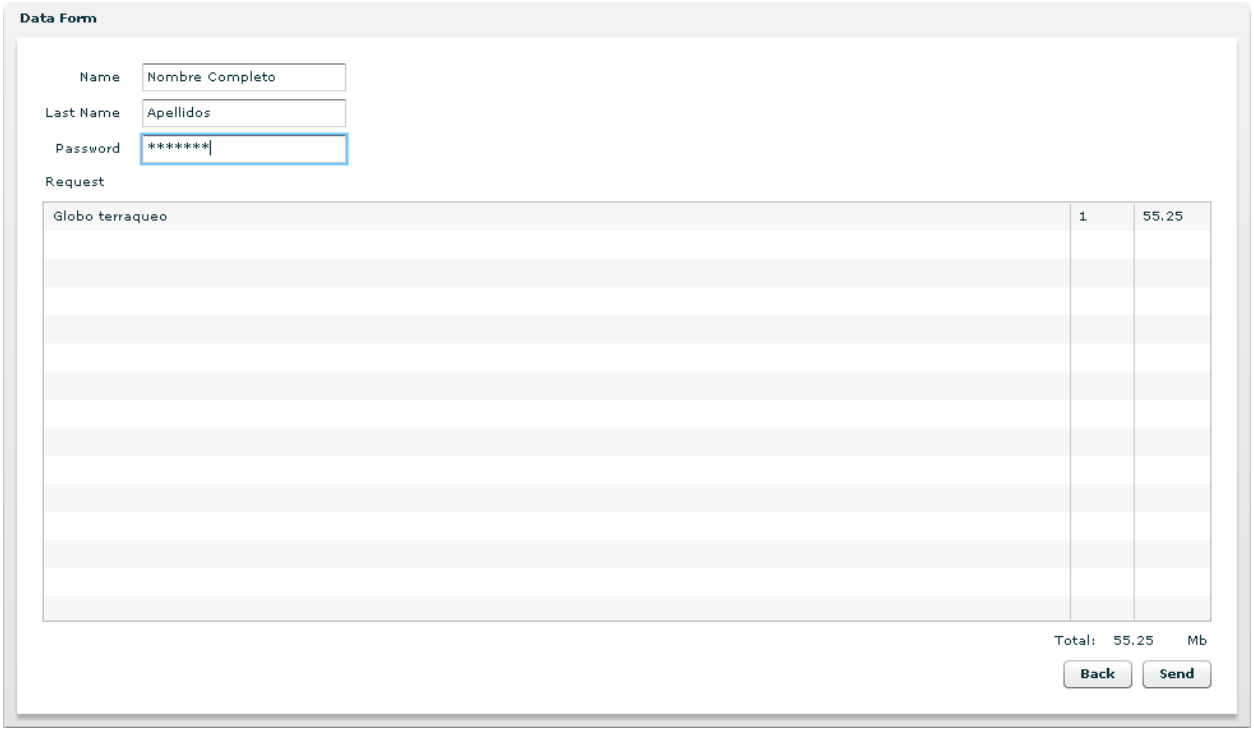


Figura 2. Interfaz gráfica para mostrar la Factura

Anexo 34: Arquitectura de Software

Introducción

Este documento describe las 4 + 1 vista de la arquitectura.

Propósito

Este documento describe la visión arquitectónica del producto, recoge y transmite las decisiones arquitectónicas que se hicieron, proporcionando una visión arquitectónica comprensiva del sistema, y utilizando para ello vistas arquitectónicas diferentes para representar aspectos diferentes del sistema.

Alcance

Este documento muestra una visión global del sistema Tienda Virtual.

Definiciones, Acrónimos y Abreviaturas

Arquitectura: 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.

Referencias

Código	Título
[1]	ER-Tienda Virtual
[2]	HU-Tienda Virtual
[3]	Propuesta de arquitectura y proceso de desarrollo para la construcción de aplicaciones web con escenarios tridimensionales interactivos

Ambiente de desarrollo

El proceso de desarrollo está basado en herramientas y métodos, en el proyecto debe conocerse qué utilidad presenta cada herramienta.

Herramientas Horizontales

Sistema operativo

El sistema operativo que se utilizará en servidores y cliente es Ubuntu 9.04

Seguridad

En el proyecto se establecerán niveles de acceso por roles y la autenticación de los desarrollares en los servidores y clientes.

Gestión de recursos

Para la gestión de recursos se utilizará Redmine.

Salvas automáticas.

Para las salvas automáticas se utilizará Bacula.

Control de versiones

Para el control de versiones se utilizará Subversion.

Gestión de proyecto

Para la gestión de proyectos se utilizará Redmine.

Gestión documental

Para la gestión documental se utilizará Subversion.

Seguimiento de errores

Para el seguimiento y control de errores se utilizará Trac.

Herramientas verticales

Herramientas de modelado

Para el modelado se utilizará Visual Paradigm 6.1.

Prueba (Pruebas Unitarias)

Para prueba se utilizará J-Meter y Selenium.

Entornos de desarrollo integrados (IDE)

Como IDE se utilizará Eclipse.

Lenguajes de programación

El lenguaje de programación a utilizar es ActionScript 3.0.

Framework o Componentes

El framework que se utilizará es Flex.

Servidor de Aplicaciones.

El servidor de aplicaciones a utilizar es Java Open Transaction Manager (JOTM).

Sistema Gestor de Bases de Datos.

El gestor de base de datos a utilizar es PostgreSQL.

Representación Arquitectónica

La representación arquitectónica a utilizar es la arquitectura de software de dominio específico: Arquitectura para el desarrollo de Aplicaciones Enriquecidas de Internet que incluyan escenarios tridimensionales interactivos.

Objetivos y Restricciones Arquitectónicas

Estos elementos se especifican en la arquitectura de software de dominio específico: Arquitectura para el desarrollo de Aplicaciones Enriquecidas de Internet que incluyan escenarios tridimensionales interactivos.

Tamaño y Rendimiento

La complejidad de la aplicación no se caracteriza en tener un crecimiento en el tamaño de la información que comprometa el rendimiento.

Vista de casos de uso

El sistema tiene un solo caso de uso Realizar pedido por lo que no se realiza la vista caso de uso.

Casos de uso arquitectónicamente significantes

No procede.

Nombre de caso de uso: Breve descripción.

La tienda virtual tiene la posibilidad de mostrar un grupo de entidades, de las cuales se brinda la vista 3D y una descripción correspondiente. Si el usuario lo desea, la entidad seleccionada puede incorporarse a un pedido. Terminado el pedido, el sistema ofrece la opción de enviar la factura, pidiendo los datos del usuario para realizar esta operación.

Estilos arquitectónicos.

Los estilos arquitectónicos a utilizar están descrito en la arquitectura de software de dominio específico: Arquitectura para el desarrollo de Aplicaciones Enriquecidas de Internet que incluyan escenarios tridimensionales interactivos.

Patrones arquitectónicos

La DSSA que se utiliza tiene como estilo arquitectónico llamada retorno y n-capas como patrón modelo vista controlador.

Vista Lógica

La vista lógica está descrita en la arquitectura de software de dominio específico: Arquitectura para el desarrollo de Aplicaciones Enriquecidas de Internet que incluyan escenarios tridimensionales interactivos.

Elementos del modelo arquitectónicamente significantes

Referenciado

Visión general de la arquitectura – Alineamiento de paquetes, subsistemas y capaz

Referenciado

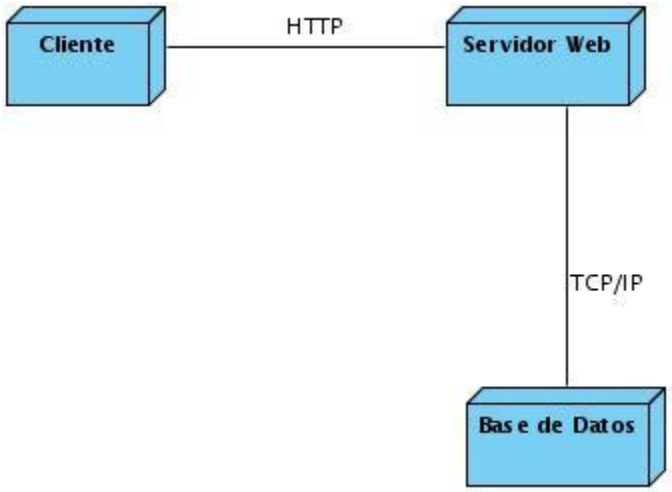
Vista de procesos

No procede

Vista de despliegue

En esta sección se la describen los nodos físicos de despliegue de la aplicación. Las capas lógicas se describen en la arquitectura de software de dominio específico: Arquitectura para el desarrollo de Aplicaciones Enriquecidas de Internet que incluyan escenarios tridimensionales interactivos.

Diagrama de despliegue.



Nombre del procesador:

Cliente: tiene la responsabilidad de ejecutar la capa lógica de presentación.

Servidor: tiene la responsabilidad de ejecutar las capas lógicas de servicios de negocio y acceso a datos.

Base de Datos: tiene la responsabilidad de ejecutar el gestor de base de datos.

Vista de Implementación

Posee un solo componente realizar pedido

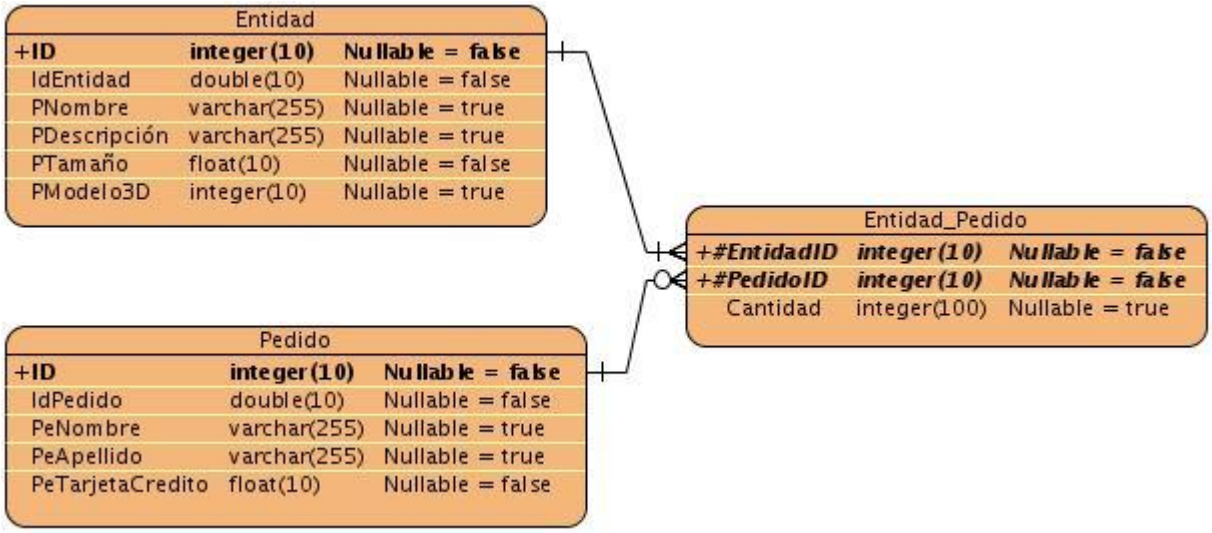
Vista de Datos

La base de datos va a estar formada por tres tablas entidad, pedido y EntidadPedidos

Entidad	Llave primaria	Número de Atributos
Entidad	Identidad	5
Pedido	Impedido	4

EntidadPedidos	IdPedido, IdEntidad	3

Diagrama de Entidad Relación



Cada una de las tablas posee los siguientes datos

Nombre: Entidad		
Descripción: Recopila los datos relacionados con la descripción de las entidades disponibles en la tienda virtual		
Campo	Tipo	Descripción
IdEntidad	INTERGER [5]	Este campo almacena el código de identificación de la entidad
PNombre	VARCHAR [20]	Este campo almacena el nombre de la entidad
PDescripción	VARCHAR[60]	Este campo almacena la descripción de la entidad
PTamaño	FLOAT[10]	Este campo almacena el Tamaño de la entidad
PModelo3D	BIT	Este campo almacena el modelo en 3D de la entidad

Nombre: Pedido

Descripción: Recopila los datos relacionados con la factura		
Campo	Tipo	Descripción
IdPedido	INTERGER [5]	Este campo almacena el código de identificación de la entidad
PeNombre	VARCHAR [20]	Este campo almacena el nombre del usuario
PeApellidos	VARCHAR[60]	Este campo almacena los apellidos del usuario
PeContraseña	INTERGER[16]	Este campo almacena la contraseña del usuario

Nombre: EntidadPedidos		
Descripción: Recopila los datos relacionados con las entidades solicitadas en los pedidos		
Campo	Tipo	Descripción
Impedido	CHAR[5]	Este campo almacena el código del pedido
IdEntidad	CHAR[5]	Este campo almacena el código de identificación de la entidad
EPCantidad	NUMERIC(8, 0)	Este campo almacena la cantidad de entidades solicitadas en el pedido

Calidad

Los elementos de calidad de la arquitectura se describen en la arquitectura de software de dominio específico: Arquitectura para el desarrollo de Aplicaciones Enriquecidas de Internet que incluyan escenarios tridimensionales interactivos.

Anexo 35: Modelo de Diseño

Introducción

En el documento se describe el sistema en términos de clases del diseño y sus relaciones.

Propósito

Con la realización de este documento se transforman los requisitos en el diseño del futuro sistema. Adaptándolo al diseño para lograr una correspondencia entre el entorno de implementación y los requisitos no funcionales. Creando una entrada apropiada y un punto de partida para actividades de implementación.

Alcance

El documento contiene el modelo de diseño.

Definiciones, Acrónimos y Abreviaturas

Entidad: modelo 3D que se pone a disposición del cliente.

Pedido: uno o varias entidades solicitados por el cliente.

Factura: refleja las líneas de pedidos solicitadas por el cliente y el tamaño de cada uno y el total

Cliente: Persona que utiliza el servicio de compra de la tienda virtual.

Referencias

Código	Título
[1]	ER-Tienda Virtual
[2]	HU-Tienda Virtual
[3]	Propuesta de arquitectura y proceso de desarrollo para la construcción de aplicaciones web con escenarios tridimensionales interactivos

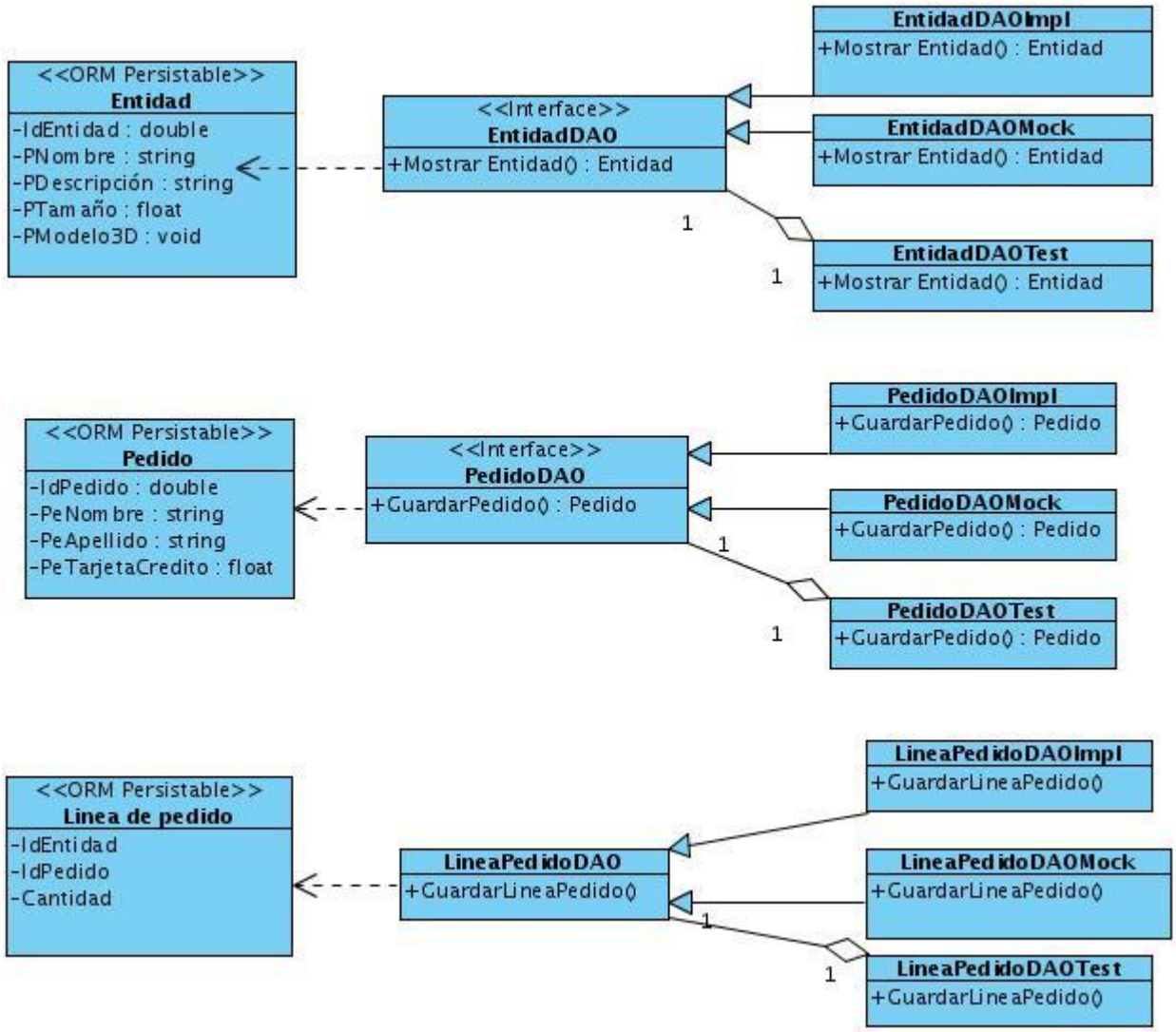
[4] AS-TiendaVirtual

Diagramas de paquetes

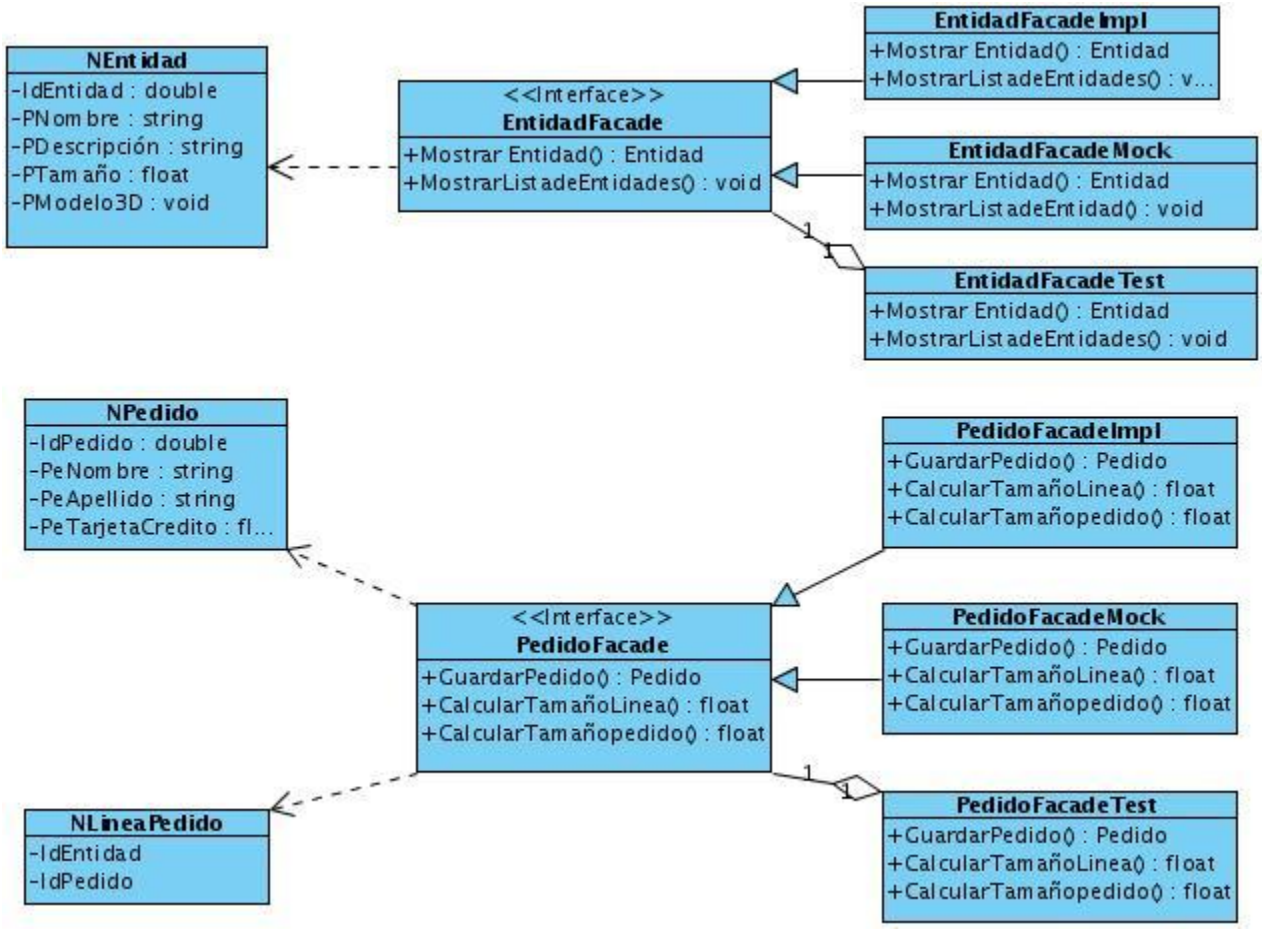
El sistema posee un solo módulo por tal motivo no tiene sentido realizar varios paquetes.

Diagrama de clases.

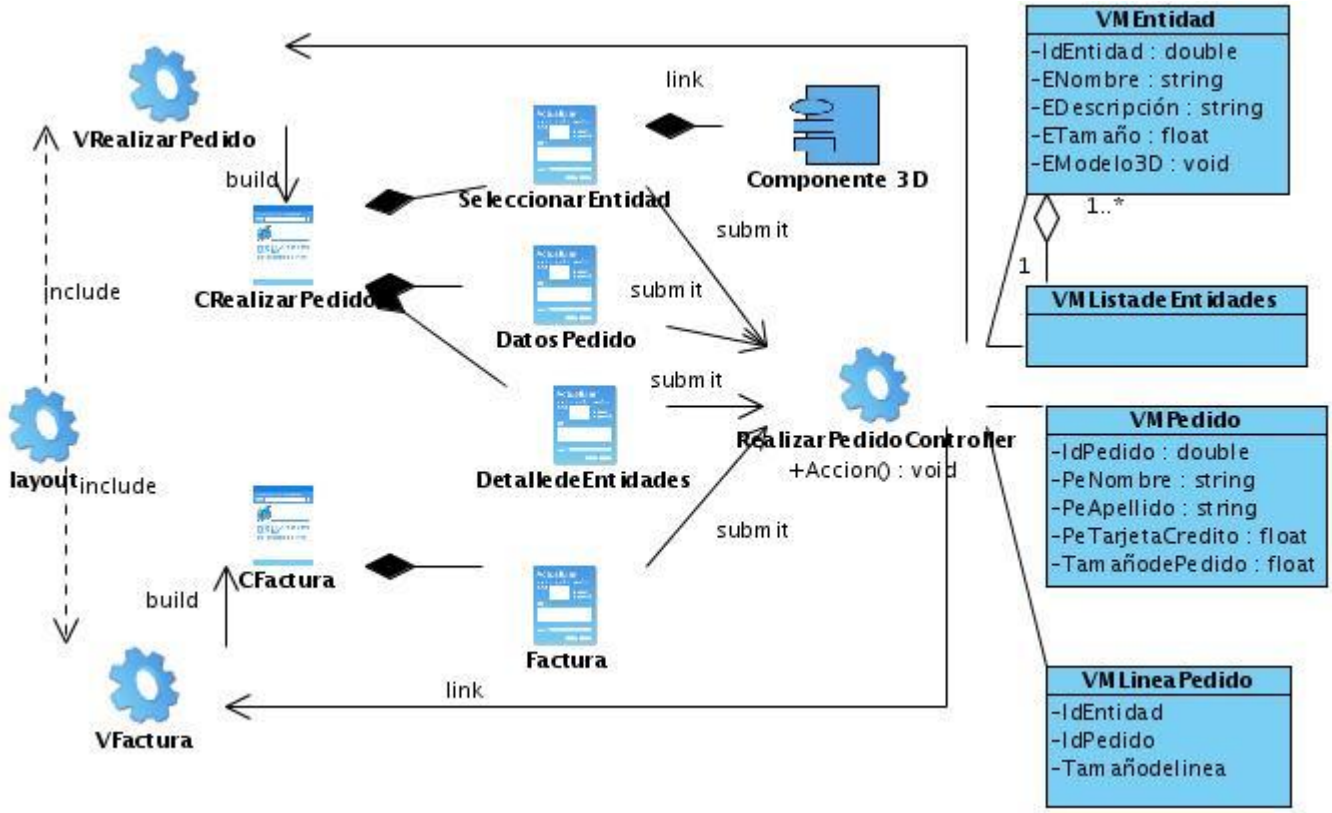
Capa de Acceso a Datos



Capa de Servicio de Negocio



Capa de Presentación



Las clases no se describen pues tienen las responsabilidades y los nombres en función de lo establecido en la arquitectura de software de dominio específico: Arquitectura para el desarrollo de Aplicaciones Enriquecidas de Internet que incluyan escenarios tridimensionales interactivos.

Anexo 36: Manual de Usuario

El sistema Tienda Virtual fue elaborado para que sirva de caso de estudio a los desarrolladores que deseen utilizar la arquitectura de software de dominio específico: Arquitectura para el desarrollo de Aplicaciones Enriquecidas de Internet que incluyan escenarios tridimensionales interactivos.

La tienda virtual muestra un grupo de entidades, de las cuales se brinda la vista 3D y una descripción correspondiente. Si el usuario lo desea, la entidad seleccionada puede incorporarse a un pedido. Terminado el pedido, el sistema ofrece la opción de enviar la factura, pidiendo los datos del usuario para realizar esta operación.

Con este manual se tiene la posibilidad de entender todas las funcionalidades del sistema dando una explicación de cada una de las partes que lo componen y su forma de uso.

FUNCIONALIDADES

Tienda virtual

Permite mostrar la lista de entidad en una vista 3D.

Muestra los detalles de la entidad seleccionada por el cliente. Los datos son: Nombre, Descripción y Tamaño.

Brinda la posibilidad de que los clientes interactúen con la vista en 3D de la entidad.

Permite que el cliente puede realizar un pedido para ello puede insertar y eliminar líneas de pedido al pedido, siempre debe insertar la cantidad a solicitar.

Calcula el tamaño de cada línea y el tamaño total del pedido y lo muestra al cliente.

Permite a los clientes llenar sus datos personales de la factura y consultar los datos del pedido: Nombre del cliente, Apellidos del cliente, Código de la Tarjeta de Crédito y se muestran los datos del pedido.

Permite almacenar el pedido.

TIENDA VIRTUAL

A través del módulo los clientes realizan las solicitudes de pedido a la tienda, para usar la aplicación no se requiere tener conocimientos previos de la aplicación o de algún sistema ofimático.

Al iniciar la aplicación se muestra en la pantalla la siguiente interfaz:

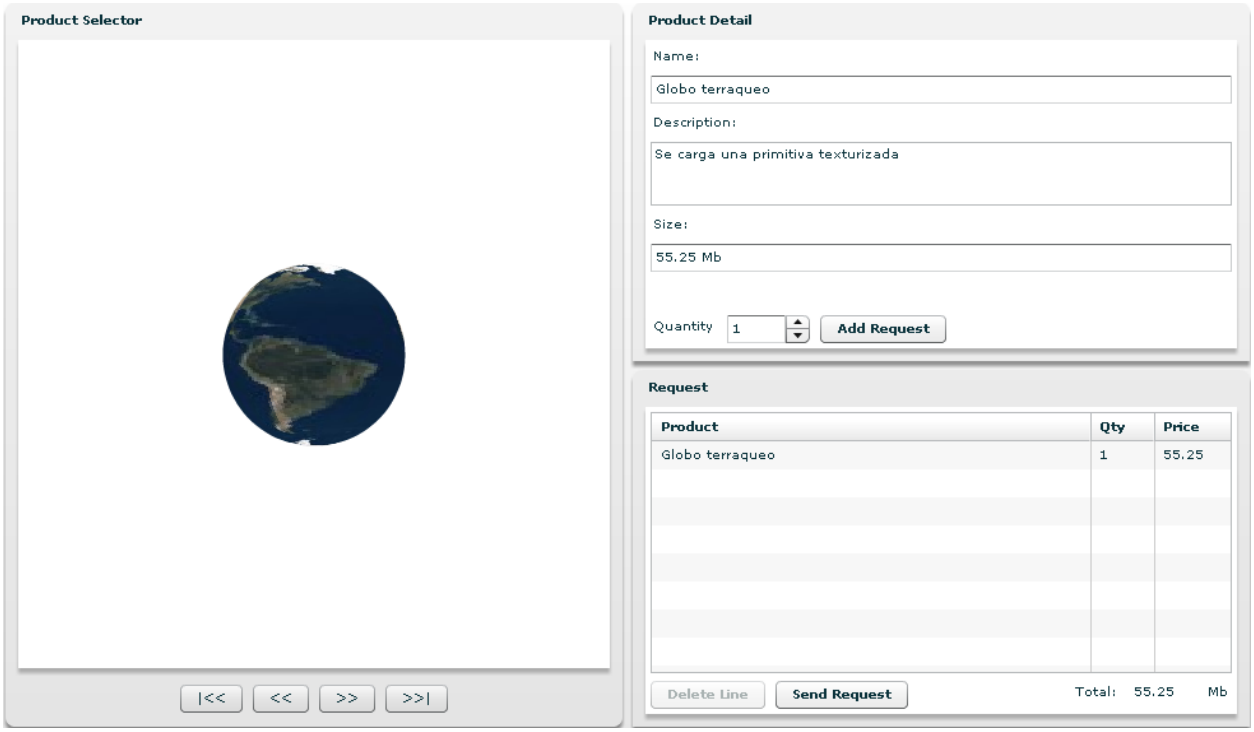


Figura 1. Interfaz gráfica inicial.

Mostrar lista entidades

En la vista Selector que se muestra en la parte izquierda de la pantalla. Se puede navegar por las vistas 3D de las entidades utilizando los botones primero, anterior, próximo y último.

Mostrar descripción

Se utiliza para consultar los datos de un entidad ante de la selección de la misma.

Seleccione de las entidades aquella que desea conocer sus detalles.

Inmediatamente consulte los detalles en la vista “Detalle” que se muestra en la parte superior derecha de la pantalla. Se puede ver el Nombre, la Descripción y el Tamaño.

Interactuar con la Vista 3D

Permite interactuar con la imagen en 3D para tener un mayor detalle de sus elementos.

Seleccione de la lista aquella entidad que desea consultar su vista en 3D.

Inmediatamente en la vista con el uso del ratón sostenido se puede desplazar por la imagen 3D.

Realizar pedido

Permite insertar y eliminar líneas de pedido al pedido, siempre debe insertar la cantidad de entidades que desea.

Insertar

Seleccione de la lista de entidades aquella que desea insertar en el pedido.

Llene la cantidad a solicitar.

Presione el botón “Añadir línea de pedido”.

Inmediatamente en la vista “Pedido” se muestra los datos de la línea de pedido solicitada.

Eliminar

Seleccione de la vista “Pedido” la línea a eliminar.

Presione le botón “Borrar Línea”.

Está línea será eliminada.

Calcular el tamaño de la línea

Permite calcular el tamaño de una línea y el del pedido completo.

Seleccione de la lista de entidades aquella que desea consultar el tamaño.

Llene la cantidad a solicitar que desea.

Inmediatamente en la Vista “Pedido” se insertara una línea con el tamaño de la misma.

Si desea conocer el tamaño del pedido completo en la vista “Pedido” se muestra el total.

Crear Factura

Permite que el cliente pueda llenar los datos necesarios para realizar el pedido.

Presione el botón “Realizar Pedido” de la vista “Pedido”.

Inmediatamente se muestra un formulario solicitando los datos: Nombre del cliente, Apellidos del cliente, Código de la Tarjeta de Crédito y se muestra los datos del pedido. Ver figura 2

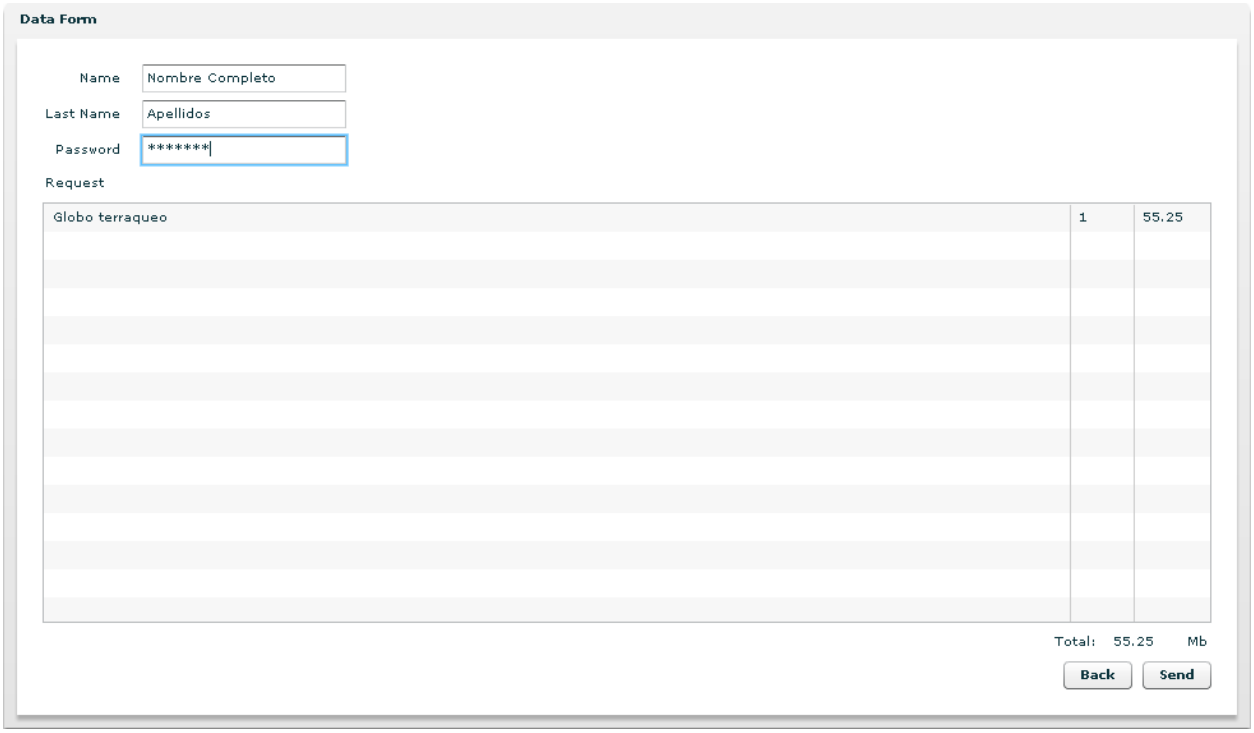


Figura 2. Interfaz gráfica para mostrar la Factura

Si desea realizar nuevamente el pedido presione el botón “Volver”.

Almacenar pedido

Permite que el cliente envíe el pedido a la tienda virtual.

Presione el botón “Enviar” de la vista “Pedido”.

Si fue satisfactorio el sistema le notifica con un mensaje.

Anexo 37: Plan de pruebas

Introducción

Este documento se reflejan las acciones a desarrollar en el proyecto para dejarlo lo mas libremente posible libre de errores.

Alcance

Contempla la estrategia de prueba del sistema, los métodos, tipos y niveles.

Definiciones, Acrónimos y Abreviaturas

Entidad: modelo 3D que se pone a disposición del cliente.

Pedido: uno o varias entidades solicitados por el cliente.

Factura: refleja las líneas de pedidos solicitadas por el cliente y el tamaño de cada uno y el total

Cliente: Persona que utiliza el servicio de compra de la tienda virtual.

Referencias

Código	Título
[1]	ER-Tienda Virtual
[2]	HU-Tienda Virtual
[3]	Propuesta de arquitectura y proceso de desarrollo para la construcción de aplicaciones web con escenarios tridimensionales interactivos
[4]	AS-TiendaVirtual
[5]	MD-TiendaVirtual

Roles y responsabilidades

Rol	Cantidad	Responsabilidad
Diseñador de pruebas	1	Participa en la confección de la estrategia y el plan de pruebas Identifica los métodos, las técnicas, herramientas y directrices apropiadas para implementar las pruebas necesarias Establece en ambiente de comprobación Encargado de diseñar casos de pruebas para el sistema Dirige la definición del enfoque de prueba y garantiza la implementación satisfactoria Diseña las pruebas Genera los casos y datos de prueba Puede fungir como probador Analiza y evalúa el estado del producto de trabajo comprobado
Probador	1	Encargado de seguir los planes de pruebas Ejecuta los casos de prueba y genera no conformidades asociadas al mismo Registra los resultados de las pruebas Analiza los resultados de las pruebas realizadas

Escenario de pruebas.

Se desarrollaran las pruebas en un entorno como el que se describe en los requisitos no funcionales.

Despliegue del sistema

Acápite 10. 1 del documento AS-Tienda Virtual

Recursos del sistema

Servidores

Recurso	Tipo
Servidor de Base de Datos	Sistema operativo Ubuntu 9.4, RAM 512 MB y Disco Duro de 5 Giga
Servidor web	Sistema operativo Ubuntu 9.4, RAM 512 MB y Disco Duro de 10 Giga

PC Clientes

PC Clientes	
Cantidad	10
Descripción	Sistema operativo Ubuntu 9.4, RAM 512 MB
Software base	Firefox
Servicios	
Realizar pedido	

Requerimientos a probar

Mostrar entidades

Mostrar descripción

Interactuar con la vista en 3D

Gestionar pedido

Mostrar factura

Enviar Factura

Estrategia de pruebas de aceptación

No procede

Evaluación de las pruebas

Nivel de importancia (IMP) de la No Conformidad

A – Alto

M – Media

B - Baja

Etapa de Detección (ED)

P – Prueba

R – Revisión Técnica

El estado (EST) de la No Conformidad para su seguimiento tendrá las siguientes categorías:

RA - Resuelta y Aprobada por el proyecto.

PD - Pendiente por solución por el proyecto.

PR – Pendiente por solución a nivel de polo.

PC – Pendiente por la parte cliente del proyecto.

AV – Aplazada para resolver en próximas versiones.

Cronograma

No.	Tarea	Fecha	Responsable	Participantes	Observaciones
1	Desarrollar las pruebas a CPR1	20/05/09	Gilberto Cao Tarrero	-	
2	Desarrollar las pruebas a CPR2	20/05/09	Gilberto Cao Tarrero	-	
3	Desarrollar las pruebas a CPR3	20/05/09	Gilberto Cao Tarrero	-	
4	Desarrollar las pruebas a CPR4	20/05/09	Gilberto Cao Tarrero	-	
5	Desarrollar las pruebas a CPR5	20/05/09	Gilberto Cao Tarrero	-	
6	Desarrollar las pruebas a CPR6	20/05/09	Gilberto Cao Tarrero	-	
7	Seguimiento a las no conformidades	25/05/09	Gilberto Cao Tarrero	-	
8	Segunda Iteración	30/05/09	Gilberto Cao Tarrero	-	

Anexo 38: Diseño casos de prueba

Descripción General

Este documento cubre el conjunto de pruebas funcionales relacionadas con la historia de usuario: Realizar pedido.

En esta historia hay que comprobar que:

El sistema inicie mostrando las tres vistas de la interfaz, en la vista 1 la lista de entidades (Vista 3D), la vista 2 inicializada en la primera entidad de la lista (Nombre, Descripción y Tamaño) y la vista 3 en blanco (Nombre, Cantidad a solicitar, Tamaño total de la línea y Tamaño total del pedido).

El sistema envía un mensaje si no hay entidades.

El sistema no puede permitir que los datos de la entidad sean modificados por el cliente en ninguna de las 3 vistas.

El sistema muestre los datos de la entidad en la vista a 2 al seleccionar una en la vista 3D o vista 1.

El sistema inicializa el dato de la Cantidad a solicitar en uno 1.

El sistema permita modificar la cantidad a solicitar validando que siempre sea mayor que 0, y un número entero.

El sistema permite que el cliente pueda interactuar con la vista en 3D.

El sistema permite agregar una línea de pedido al pedido y se refleja en la vista 3.

El sistema muestra en la vista 3 las líneas de pedido que el cliente a pedido.

El sistema permite eliminar la o las línea de pedido del pedido y se actualiza el pedido y el tamaño total se re-calcula.

El sistema permite realizar el pedido siempre que tenga al menos una entidad solicitada.

El sistema debe inhabilitar las opciones Borrar línea de pedido y realizar pedido cuando no hayan líneas de pedido solicitadas.

El sistema muestra una nueva interfaz solicitando los datos de la factura Nombre del cliente, Apellidos del cliente, Código de la Tarjeta de Crédito.

El sistema debe validar las entradas de los datos Nombre del cliente, Apellidos del cliente, Código de la Tarjeta de Crédito.

El sistema permite volver a seleccionar el pedido y se inicializa las sesiones anteriores con el pedido seleccionado.

El sistema permite almacenar los datos del pedido.

El sistema inhabilita la opción de enviar factura hasta que estén llenos los datos del cliente.

El sistema almacena correctamente el pedido en la Base de Datos.

A esta Historia de Usuario se le realizaron las siguientes pruebas:

2 Iteraciones de pruebas y una revisión de seguimiento a las no conformidades.

CPR 1: Mostrar entidades

Descripción de la Funcionalidad:

Mostrar todas las entidades en una vista 3D.

Flujo Central:

Abrir la aplicación

Condiciones de Ejecución:

Deben existir entidades.

CPR 2: Mostrar descripción

Descripción de la Funcionalidad:

Mostrar la descripción de la entidad seleccionada por el cliente, de ellos se dispone de los siguientes datos: Nombre, Descripción, Tamaño y Cantidad a solicitar.

Flujo Central: Mostrar entidades

El cliente debe seleccionar una entidad.

Condiciones de Ejecución:

Deben existir entidades y el cliente debe seleccionar una.

CPR 3: Interactuar con la vista en 3D

Descripción de la Funcionalidad:

Mostrar la vista en 3D de la entidad y permitir que el cliente interactúe con ella.

Flujo Central: Mostrar entidades

El cliente debe accionar con el ratón sobre la vista 3D.

Condiciones de Ejecución:

Deben existir entidades.

CPR 4: Gestionar pedido

Descripción de la Funcionalidad:

Insertar y eliminar líneas de pedido al pedido. Registrando registran los siguientes datos: Nombre, Cantidad a solicitar, Tamaño de la línea de pedido, Tamaño total del pedido.

Flujo Central: Mostrar descripción

El cliente selecciona borrar línea de pedido del pedido o realizar pedido.

Condiciones de Ejecución:

Deben existir entidades, el cliente debe tener entidades seleccionadas para el pedido.

CPR 5: Mostrar Factura

Descripción de la Funcionalidad:

Mostar la factura del pedido realizado por el cliente, se solicita los siguientes datos: Nombre del cliente, Apellidos del cliente, Código de la Tarjeta de Crédito y se muestran los datos del pedido.

Flujo Central: Gestionar pedido

El cliente solicita realizar factura.

Condiciones de Ejecución:

Deben existir líneas de pedido solicitados por el cliente para el pedido.

CPR 6: Enviar Factura

Descripción de la Funcionalidad:

El sistema almacena los datos de la factura

Flujo Central: Mostrar Factura

El cliente enviar factura.

Condiciones de Ejecución:

Deben existir líneas pedido solicitadas por el cliente para el pedido y deben estar llenos los datos del cliente.

GLOSARIO DE TÉRMINOS Y SIGLAS

Términos

ActionScript: es un lenguaje de programación orientado a objetos, que permite en especial el desarrollo de Aplicaciones Enriquecidas de Internet.

Actividad: conjunto de operaciones o tareas propias de una persona o entidad que permite que el trabajo a realizar sea descrito y entendido de manera precisa por aquellos que tienen que ejecutarlo.

Ambiente: entorno o suma total de aquello que rodea y que afecta y condiciona especialmente las circunstancias de vida de las personas o la sociedad en su conjunto. Comprende el conjunto de valores naturales, sociales y culturales existentes en un lugar y un momento determinado, que influyen en la vida del hombre y en las generaciones venideras.

Ambiente integrado de desarrollo: programa compuesto por un conjunto de herramientas para el programador pueden usarse uno o varios lenguajes de programación. En él se agrupan varios programas de aplicación, un editor de código, un compilador, un depurador y un constructor de interfaz gráfica

Aplicaciones Enriquecidas de Internet (RIA, del inglés Rich Internet Applications): es una arquitectura de programación, que posibilitan la ejecución de variadas funciones en el navegador sin necesidad de sobrecargar ni el ancho de banda ni los servidores. Las aplicaciones poseen las características de las aplicaciones escritorios y las aplicaciones web tradicionales ampliamente nutridas de multimedia.

Aplicaciones web: aplicaciones que los usuarios pueden utilizar accediendo a un servidor web a través de Internet o de una Intranet mediante un navegador.

Arquitectura: es el arte de construir, de acuerdo con un programa y empleando los medios diversos de que se dispone en cada época, tiene un fundamento científico y obedece a una técnica compleja.

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

Arquitectura de software de dominio específico: definición de una arquitectura de software que se realiza para un dominio específico.

Arquitecturas informáticas: es el diseño conceptual y la estructura operacional fundamental de un sistema de computadoras.

Artefacto: es un término general, para cualquier tipo de información creada, producida, cambiada o utilizada por los trabajadores en el desarrollo del sistema.

Bases Tecnológicas: abarcan el conocimiento sobre las tecnologías para la construcción del software, la gestión y el soporte del mismo.

Calidad: conjunto de propiedades y características de un producto o servicio que le confieren su aptitud para satisfacer unas necesidades explícitas o implícitas.

Calidad de software: grado con que el que un sistema, componente o proceso cumple con los requerimientos especificados y las necesidades o expectativas del cliente o usuario.

Capa de presentación: generalmente se identifica como la capa web. Esta capa debe ser tan fina como sea posible. Además, debe permitir diferentes capas de presentación, tales como una capa web y/o fachadas de servicios web remotos, sobre una simple y bien diseñada capa de negocio.

Capa de servicios de negocio: es la capa responsable de delimitar las transacciones y proveer un punto de entrada para las operaciones sobre el sistema. Esta capa no debería tener conocimiento sobre lo concerniente a la presentación y debería ser reutilizable.

Capa de acceso a datos: esta capa presenta un conjunto de interfaces independientes de la tecnología de acceso a datos, que son usadas para buscar y persistir los objetos persistentes. La capa de acceso a datos no debería de contener ningún tipo de lógica de negocio.

Capas lógica: distribución de forma física de los niveles de una arquitectura de software.

Componentes de código: es una parte física y reemplazable del sistema, que cumple y proporciona la realización de un conjunto de interfaces.

Comunicación: proceso de intercambio de mensaje entre dos elementos, un emisor y un receptor. Este intercambio de mensaje se realiza a través de un canal o medio, que es un dispositivo físico de transmisión. Para el intercambio de mensajes es imprescindible que el emisor y el receptor tengan un conocimiento o repertorio en común y así puedan realizar la codificación y decodificación de mensajes en forma congruente, lo que posibilita generar en el receptor un incremento del conocimiento.

Costo: es el sacrificio económico incurrido en la obtención de activos, con la finalidad de obtener beneficios futuros.

Despliegue: disciplina del proceso de desarrollo de software que tiene como objetivo implantar la aplicación desarrollada en las instalaciones del cliente.

Diseño: disciplina del proceso de desarrollo de software que tiene como objetivo describir la aplicación en términos de los desarrolladores, identificando los elementos que la componen en clases y sus relaciones para satisfacer las necesidades funcionales y la calidad de un sistema.

Dominio de aplicación: es un área de interés, usualmente representando un espacio del problema que es un subconjunto de una o más disciplinas.

e-business: el comercio electrónico, también conocido como e-commerce, consiste en la compra y venta de producto o de servicios a través de medios electrónicos, tales como Internet y otras redes de ordenadores.

Eficiencia: capacidad de alcanzar los objetivos y metas programadas con el mínimo de recursos disponibles y tiempo, logrando su optimización.

E-government: gobierno electrónico consiste en el uso de las Tecnologías de la Información y las Telecomunicaciones en los procesos internos de gobierno y en la entrega de los productos y servicios del Estado tanto a los ciudadanos como a la industria.

Escenario: es el espacio destinado para la representación de un dominio de aplicación.

Escenario tridimensional: es el espacio destinado para la representación de un dominio de aplicación en las tres dimensiones ancho largo y profundidad.

Escenarios tridimensionales interactivos

Equipo de desarrollo: es un grupo de trabajo constituido por una serie de profesores, investigadores, colaboradores y alumnos unidos en la ilusión de acometer un determinado proyecto o avanzar en el conocimiento y en la investigación teórica y aplicada.

Especificación lógica: traduce los escenarios de uso creados en el diseño conceptual en un conjunto de objetos de negocio y sus servicios. Se convierte en parte en la especificación funcional que se usa en el diseño físico. 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.

Estándar: lo que es establecido por la autoridad, la costumbre o el consentimiento general. En este sentido se utiliza como sinónimo de norma.

Estructura Organizacional: estructura que descompone la labor de la compañía en tareas especializadas, asigna éstas a personas y departamentos y coordina las tareas mediante la definición de vínculos formales entre personas y departamentos, estableciendo línea de autoridad y comunicación.

Estudio preliminar: etapa del proceso de desarrollo de software donde se determinan las necesidade**s d**el cliente y se determina la viabilidad económica de ejecutar el proyecto.

Fiabilidad: conjunto de atributos relacionados con la capacidad del software de mantener su nivel de prestación bajo condiciones establecidas durante un período de tiempo establecido.

Feedback o retroalimentación: es un elemento de evaluación que permite al emisor saber si el mensaje enviado es recibido y si fue interpretado correctamente por el receptor.

Framework: es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, un framework puede incluir soporte de programas, librerías y un lenguaje de *scripting* entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

Funcionalidad: conjunto de atributos que se relacionan con la existencia de un conjunto de funciones y sus propiedades específicas. Las funciones son aquellas que satisfacen lo indicado o implica necesidades.

Gestión: es la acción y efecto de gestionar o la acción o efecto de administrar. Comprende todas las actividades de una organización que implican el establecimiento de metas u objetivos, así como la evaluación de su desempeño y cumplimiento; además del desarrollo de una estrategia operativa que garantice la supervivencia de la misma, según al sistema social correspondiente.

Gestión de Calidad: es actualmente una alternativa empresarial indispensable para la supervivencia y la competitividad de las empresas en los mercados en los que actúa. A través de ella, se busca la optimización de recursos, la reducción de fallos y costes y la satisfacción propia y del cliente. Está medida por una serie de normas aplicables genéricamente a todas las organizaciones, sin importar su tipo, tamaño o su personalidad jurídica.

Gestión de configuración: es mantener la integridad de los productos que se obtienen a lo largo del desarrollo de los sistemas de información, garantizando que no se realizan cambios incontrolados y que todos los participantes en el desarrollo del sistema dispongan de la versión adecuada de los productos que manejan. La gestión de configuración se realiza durante todas las actividades asociadas al desarrollo del sistema, y continúa registrando los cambios hasta que éste deja de utilizarse. La gestión de configuración facilita el mantenimiento del sistema, aportando información precisa para valorar el impacto de los cambios solicitados.

Gestión del conocimiento: es la gestión de los activos intangibles que generan valor para la organización. La mayoría de estos intangibles tienen que ver con procesos relacionados de una u otra forma con la captación, estructuración y transmisión de conocimiento. Por lo tanto, la Gestión del Conocimiento tiene en el aprendizaje organizacional su principal herramienta.

Gestión de los Recursos Humanos: función administrativa en la que se maneja el reclutamiento, asignación, capacitación y el desarrollo de los miembros de una organización o empresa.

Gestión de proyecto: implica la planificación, supervisión y control del personal, del proceso y de los eventos que ocurren mientras evoluciona el software desde la fase preliminar hasta el soporte.

Herramientas: utensilios o provisiones necesarias para poder emprender un proyecto de software. Soportan los procesos de desarrollo de software modernos.

Herramientas CASE: conjunto de programas y ayudas que dan asistencia a los analistas, ingenieros de software y desarrolladores, durante todos los pasos del ciclo de vida de desarrollo de un software (Investigación Preliminar, Análisis, Diseño, Implementación e Instalación.).

Hardware: corresponde a todas las partes físicas y tangibles de una computadora: sus componentes eléctricos, electrónicos, electromecánicos y mecánicos; sus cables, gabinetes o cajas, periféricos de todo tipo y cualquier otro elemento físico involucrado; contrariamente al soporte lógico e intangible que es llamado software.

Implementación: disciplina del proceso de desarrollo de cuya finalidad es implementar los componentes y productos para satisfacer las necesidades funcionales y la calidad de un sistema.

Ingeniería de Software: Se puede definir como el tratamiento sistemático de todas las fases del ciclo de vida del software.

Mantenibilidad: conjunto de atributos relacionados con la facilidad de extender, modificar o corregir errores en un sistema software.

Metodologías de desarrollo de software: conjunto de procedimientos que imponen una serie de pasos sobre el desarrollo del software que permiten producir y mantener un producto garantizando su fiabilidad y calidad.

Modelos de comunicación: representa de manera teórica la o las técnicas de comunicación entre aplicaciones.

Modelos de procesos del software: es una estrategia de desarrollo que acompaña al proceso.

Persona: seres humanos que intervienen en el proceso de desarrollo, a diferencia del término abstracto trabajadores.

Navegador web: es un programa que permite visualizar la información que contiene una página web (ya esté esta alojada en un servidor dentro de la WWW o en uno local).

Norma: es una regla a seguir para alcanzar un fin determinado. Las normas se crean en Comisiones Técnicas de Normalización. Una vez elaborada la norma, esta es sometida durante seis meses a la opinión pública. Transcurrido este tiempo y analizadas las observaciones se procede a su redacción definitiva, con las posibles correcciones que se estimen, publicándose luego. Todas las normas son sometidas a revisiones periódicas con el fin de ser actualizadas.

Organización del proceso: es la forma en que distribuyen las tareas o actividades dentro del equipo de desarrollo, es asignar a cada persona del equipo el rol de acuerdo a las capacidades mostradas y velar por el cumplimiento de las tareas.

Persona: son seres humanos que intervienen en el proceso de desarrollo, a diferencia del término abstracto trabajadores. Los principales autores de un proyecto software son los arquitectos, desarrolladores, ingenieros de prueba, y el personal de gestión que les da soporte, además de los usuarios, clientes y otros interesados.

Planificación: es el establecimiento de objetivos, y la decisión sobre las estrategias y las tareas necesarias para alcanzarlas.

Plugin: es una aplicación que interactúa con otra para agregarle una funcionalidad específica y es ejecutada por la aplicación principal.

Portabilidad: conjunto de atributos relacionados con la capacidad de un sistema software para ser transferido desde una plataforma a otra.

Proceso: conjunto de actividades y resultados asociados que producen un resultado.

Proceso de desarrollo de software: es la definición del conjunto de actividades que guían los esfuerzos de las personas implicadas en el proyecto para transformar los requisitos de usuario en un producto.

Procesos: conjunto de pasos parcialmente ordenados con el propósito de alcanzar una meta.

Productividad del trabajo: indicador que refleja que tan bien se están usando los recursos de una economía en la producción de bienes y servicios

Producto: conjunto de artefactos que se crean durante la vida del proyecto, como los modelos, código fuente, ejecutables y documentación.

Proyecto: combinación de recursos humanos y no humanos reunidos en una organización temporal para conseguir un propósito, tiene un punto de de comienzo definido y con objetivos definidos mediante los que se identifican.

Proyecto de Software: el elemento organizativo a través del cual se gestiona el desarrollo de software. El resultado de un proyecto es una versión de un producto.

Prueba: disciplina del proceso de desarrollo de software cuyo propósito es integrar y probar el sistema.

Recursos: conjunto de elementos disponibles para resolver una necesidad o llevar a cabo una tarea.

Requerimiento: son capacidades o características que debe tener el sistema o modelo desarrollo para satisfacer la demanda y/o necesidad del cliente.

Requisitos: una condición o capacidad que tiene que tener un sistema para satisfacer al cliente.

Requisitos funcionales: son el conjunto de Capacidades o funciones que el sistema debe cumplir.

Requisitos no funcionales: son las propiedades o cualidades que el producto debe tener y representan las características del producto.

Repositorio: Es un sitio centralizado donde se almacena y mantiene información digital, como bases de datos o archivos informáticos.

Repositorio de componentes: Biblioteca de componentes software reutilizables. Los componentes almacenados en el repositorio deben tener una representación estándar y estar bien documentados, siendo el sistema gestor de la biblioteca el encargado de organizar, proteger y gestionar dichos componentes.

Reutilización: es la acción de volver a utilizar los bienes o productos ya elaborados y probados. Puede venir propiciada por una mejora o restauración o sin modificarse, usarlo en la creación de un nuevo producto.

Realidad Virtual: polo productivo de la Universidad de las Ciencias Informáticas que desarrolla aplicaciones que genera entornos sintéticos en tiempo real, representación de las cosas a través de medios electrónicos o representaciones de la realidad, una realidad ilusoria, pues se trata de una realidad perceptiva sin soporte objetivo, sin red extensa, ya que existe sólo dentro del ordenado.

Servidores de aplicaciones: un servidor en una red de computadoras que ejecuta ciertas aplicaciones con el fin de gestionar las peticiones del usuario y devolver a los mismos recursos a través de un protocolo de comunicación.

Soporte: disciplina del proceso de desarrollo de software que su objetivo es brindar los servicios de mantenimiento y corrección post venta de un producto.

Técnicas: sucesión ordenada de acciones que se dirigen a un fin concreto, conocido y que conduce a unos resultados precisos.

Tecnología: característica propia del ser humano consistente en la capacidad de éste para construir, a partir de materias primas, una gran variedad de objetos, máquinas y herramientas, así como el desarrollo y perfección en el modo de fabricarlos y emplearlos con vistas a modificar favorablemente el entorno o conseguir una vida más segura. El ámbito de la Tecnología está comprendido entre la Ciencia y la Técnica propiamente dichas.

Tecnologías de la Información y las Comunicaciones: son un conjunto de servicios, redes, software y dispositivos que tienen como fin la mejora de la calidad de vida de las personas dentro de un entorno, y que se integran a un sistema de información interconectado y complementario.

Tecnología informática: el estudio, diseño, desarrollo, puesta en práctica, ayuda o gerencia de los sistemas informáticos computarizados, particularmente usos del software y hardware.

Tiempo de desarrollo: variable no modificable. Sin embargo se puede estimar, organizar y medir. Mientras mejor se controla el uso del tiempo más eficiente será el trabajo.

Usabilidad: conjuntos de atributos relacionados con el esfuerzo necesitado para el uso, y en la valoración individual de tal uso, por un establecido o implicado conjunto de usuarios.

Web: es un sistema de documentos de hipertexto y/o hipermedios enlazados y accesibles a través de Internet. Con un navegador web, un usuario visualiza páginas web que pueden contener texto, imágenes, videos u otros contenidos multimedia, y navega a través de ellas usando hiperenlaces.

Siglas

3D: Tridimensional.

BD: Base de datos.

CASE: Computer Aided Software Engineering, Ingeniería de Software Asistida por Computadoras.

CMMI: Modelo de Madurez de Capacidades Integrado.

CVS: Sistema Concurrente de Versiones.

DAO: Data Access Object.

DRA: Desarrollo Rápido de Aplicaciones.

DSSA: Arquitectura de Software de Dominio Específico.

e-business: negocio o comercio electrónico.

e-government: gobierno electrónico.

HTML: Lenguaje de Marcas de Hipertexto o HyperText Markup Language.

IDE: Ambiente Integrado de Desarrollo.

IEEE: The Institute of Electrical and Electronics Engineers, el Instituto de Ingenieros Eléctricos y Electrónicos, una asociación técnico-profesional mundial dedicada a la estandarización, entre otras cosas.

IP: Infraestructura Productiva.

ISO: Organización Internacional de Estándares o International Organization for Standarization.

ISO 9001: La norma ISO 9001, es un método de trabajo, con el fin de mejorar la calidad y satisfacción de cara al consumidor. Está dirigido a mejorar los aspectos organizativos de una empresa.

MVC: Modelo Vista Controlador.

PHP: Hypertext Preprocessor.

PSP: Proceso de Software Personal o Personal Software Process.

RIA: Rich Internet Applications.

RUP: Proceso Unificado de Racional o Rational Unified Process.

RV: Realidad Virtual.

TIC: Tecnologías de la Información y las Comunicaciones.

TI: Tecnología Informática.

TSP: Proceso de Software en Equipo o Team Software Process.

XP: Programación Extrema o eXtreme Programming.

UCI: Universidad de Ciencias Informáticas.

UML: Lenguaje Unificado de Modelado

WWW: World Wide Web.