Universidad de las Ciencias Informáticas

Facultad 3

“Sistema para el Control de Activos de Software del Centro de Gobierno Electrónico”

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

Autores: Yuriel Guerra Rodríguez

Víctor Hernández Martiartu

Tutor: Ing. Yadira Lizama Mué

Co-tutores: Ing. Lissuan Fadraga Artiles

Ing. Alejandro Casanova Mutis

Ciudad de La Habana, 2013

“Año 55 de la Revolución”

DECLARACIÓN DE AUTORÍA

Declaramos que somos los autores de este trabajo y autorizamos a la Facultad 3 de la Universidad de las Ciencias Informáticas; así como al Centro para el desarrollo de Gobierno Electrónico para que hagan el uso que estimen pertinente del mismo.

Para que así conste firmamos la presente a los ____ días del mes de _____ del año 2013.

_____ _____

Yuriel Guerra Rodríguez — Autor

Víctor Hernández Martiartu — Autor

_____ _____

Ing. Yadira Lizama Mué — Tutora

Ing. Lissuan Fadraga Artiles — Co-tutor

_____

Ing. Alejandro Casanova Mutis

Co-tutor

AGRADECIMIENTOS

Agradecemos a todas las personas que de alguna forma han colaborado con la realización del presente trabajo de diploma, a nuestra tutora por ayudarnos y apoyarnos siempre.

Yuriel

A mi mamá, por ser la persona más importante de mi vida, quien me ha brindado un mundo lleno de amor y felicidad, por preocuparse por mí, por tantos consejos, por darme todo su amor y su cariño, por escucharme y por inspirarme a intentar ser cada día una mejor persona.

A mi papa, por cuidarme, por darme su fuerza y su valor, por apoyarme incondicionalmente en todas las decisiones que he tomado en la vida y por confiar siempre en mí.

A mi hermanita linda, por quererme tanto, por ser mi ejemplo a seguir, por confiar en mí y brindarme tanto amor, apoyo y consejos.

A mi novia por confiar en mí, por brindarme su amor y soportarme todos estos años.

A mi tutora Yadira por su preocupación y colaboración con el trabajo, por ayudarme, enseñarme, sin su ayuda este trabajo no hubiera tenido la misma calidad.

A la oponente Dailien que aunque en esta ocasión le tocó desempeñar el papel de villana, nos ayudó mucho, con sus críticas y sugerencias este trabajo obtuvo la calidad requerida.

A la profesora Mónica por sus charlas y consejos, fue como mi mamá en la universidad.

A todas las amistades que en estos años aquí en la universidad me han ayudado, comprendido y han logrado que los tenga como mi familia. A los de mi grupo en primer año, a los del 3501 que paso muchos momentos lindos, al piquete SOMOS TESIS por tantos momentos juntos vividos.

A mis amor hermanos Rodolfo, Reinier y Rodney que aunque este último año no estuvieron aquí siempre me apoyaron y nunca se despreocuparon de mí.

A mi amigos Luis Ángel, Raúl, Ricardo, Alfredo, Cesar, Adrián, Amílcar, Ángel, Abel, Addiel por todos los juegos de domino y discusiones que tuvimos que gracias a eso convirtieron los momentos más aburridos en fiestas muy alegres.

A mi compañero de tesis porque sin el este trabajo no hubiera sido posible.

A todos los que de alguna manera hemos compartido estos años de alegrías y preocupaciones, por haberme brindado la posibilidad de poder compartir con ustedes, y ser parte de esta familia UCI.

Victor

Empiezo agradeciendo a mis padres por todo el trabajo empeñado en mí, a mi padre por las madrugadas de viaje al hospital por mi condición de asmático e inculcarme el amor por los libros y las matemáticas. A mi mamá por su comprensión y su confianza en mí, gracias a los dos. A mi tía Lleya por mejorar mi español y mis habilidades matemáticas. Nunca imaginé trabajar tanto en tan poco tiempo, en 5 años de universidad por primera vez pensé que me quedaría calvo con tanta programación. Debo agradecer a todos los que ayudaron a que mantuviera mis cabellos en su lugar de una manera u otra. Un saludo para el sucio, que nunca escatimó recursos y conocimiento en pos de ayudarnos en lo que pudo y lo que no, nos buscó a los expertos. Al insurrecto que en más de una ocasión tuvo que sentarse a desentrañar nuestro marco de trabajo y tuvimos que buscar más de una manera de hacer las cosas, puro ingenio de ingenieros. A nuestra gran tutora Yadira, sí grande, desde el primer día nos dijo que sus tesis eran sobresalientes con creces. Sin perder jamás la calma ante un dúo de pogolotis, apodo que ganamos por ser un aparente desastre, siempre nos guio por el camino correcto, nos ayudó en todo, inclusive con su recurso más querido, Toshiba. A Cordone que jamás me dijo que no ante una duda en ingeniería de software. A mi novia, la cual gasto más de 1 Mb en ayuda a nuestro diseño metodológico y su preocupación, la cual fue tanta, que me llego a mí, a quien nunca nada le había preocupado nada. A mi compañero de Symfony Joel, que entre ambos más de una vez innovamos más de un método. También agradezco a todos, los cuales fueron muchos, que al preguntarme por mi tesis y con quien era, exclamaron en muchas ocasiones que sería una bomba, dando pie a un futuro desastre, gracias por sus ánimos, les aseguro que esto también es de ustedes, que el camino es hecho por los caminantes, pobre de aquellos que poseen piernas y no saben andar. A los profes Ariel, por su ayuda en física ante mis meticulosas dudas, Olga Lidia por hacer de este el mejor lugar de América Latina en primer y segundo año, Armando por alimentar mi curiosidad por teleinformática, Diana por intentar hacer de mi alguien menos aparatoso en ingeniería de software, Susana por su ímpetu de que todos aprendiéramos bien Discreta, a Filosofía, PHCUC y Pedagogía los cuales además de su asignatura nos educaron en más de una

ocasión. A Rubén, mi hermano en la universidad, al cual agradezco por enseñarme a programar en una hora, ese día y estos últimos meses han sido mi estudio en la universidad, un saludo. Agradecer a los que vieron en mí a un estudiante que no pasaría primer año, luego segundo, después tercero y pensaron que 4 sería mi final, un saludo por no dejar de recordármelo, ya soy ingeniero.

DEDICATORIA

Yuriel

A mis padres por guiarme y estar siempre ahí en todo momento, porque son lo mejor que tengo en la vida.

A mi hermana por ser mi ejemplo a seguir y siempre confiar en mí.

A mi familia y amistades por apoyarme y creer en mí.

Victor

Este título se lo dedico a mi abuelo Francisco y a mi tío Machado, a los cuales Dios antes de llevárselos a la gloria, siempre se preocuparon por mí con respecto en la universidad. A mis padres, mi hermano Daniel y a mi tía Regla que siempre me dijo que sería el ingeniero de la nueva generación de los Martiartu.

Gracias por su confianza.

RESUMEN

El Centro para el desarrollo del Gobierno Electrónico (CEGEL), adscrito a la Universidad de las Ciencias Informáticas (UCI) tiene como misión satisfacer necesidades de clientes gubernamentales mediante el desarrollo de productos, servicios y soluciones integrales de alta confiabilidad, calidad, competitividad, fidelidad y eficiencia, a partir de un personal altamente calificado. El desarrollo de soluciones enmarcadas en un dominio específico, el Derecho y la Informática Jurídica, hace que la necesidad de reutilización de los componentes de software que se generan aumente considerablemente. La reutilización de la información generada y de los componentes, permite agilizar el desarrollo de nuevos productos ante las oportunidades de negocio que tiene este Centro. Los activos desempeñan un papel importante en la reutilización de software, ya que son usados para desarrollar nuevas versiones de un programa con variaciones al original. De conjunto los repositorios de activos de software, permiten el almacenamiento, administración y recuperación de estos activos. El objetivo principal de este trabajo es desarrollar un Sistema para el Control de Activos de Software (código fuente y documentación) generados durante el proceso de desarrollo de los proyectos del Centro. Con este resultado se pretende contribuir al aumento de los niveles de reutilización en el proceso de desarrollo que incide positivamente en los niveles de calidad del mismo.

Palabras claves:
Activos de software, Control, Reutilización.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

La evolución constante de los procesos tecnológicos propiciados por el aumento del acceso, distribución y gestión del conocimiento asociado a la Informática y las Telecomunicaciones, es evidentemente palpable en la denominada “Sociedad de la Información”. La industria de software experimenta aceleradamente el impacto de este fenómeno por lo que la Universidad de las Ciencias Informáticas (UCI), máximo exponente de esta industria en Cuba, no está exenta de ello. La UCI tiene como misión “Formar profesionales comprometidos con su Patria y altamente calificados en la rama de la Informática. Producir aplicaciones y servicios informáticos, a partir de la vinculación estudio-trabajo como modelo de formación. Servir de soporte a la industria cubana de la informática”[1].

El Centro de desarrollo para el Gobierno Electrónico (CEGEL) adscrito a la UCI tiene como objetivo fundamental satisfacer las necesidades de clientes gubernamentales mediante el desarrollo de productos, servicios y soluciones integrales de alta confiabilidad, calidad, competitividad, fidelidad y eficiencia, a partir de un personal altamente calificado. La producción de soluciones enmarcadas en un dominio específico, la Informática Jurídica, hace que las características fundamentales de sus productos sean la composición por módulos, la configuración de sus dependencias, la adaptación de sus funcionalidades y la parametrización de la información que se muestra de acuerdo a las exigencias de este dominio de aplicación. Estos elementos evidencian que la necesidad de reutilización de los componentes de software que se desarrollan sea considerable.

“La Reutilización es el uso de un activo para la solución de diferentes problemas. Un activo reutilizable es un elemento tal como diseños, especificaciones, código fuente, documentación, unidades de prueba y manuales de procedimientos”, los cuales fueron diseñados para su uso en múltiples contextos [6].

El proceso de reutilización de la información generada y de los componentes, permite agilizar el desarrollo de nuevos productos ante las oportunidades de negocio que tiene el Centro y los activos de software desempeñan un papel fundamental en este proceso, ya que son usados para desarrollar nuevas versiones de un programa con variaciones al original. De conjunto los repositorios de activos de software, permiten el almacenamiento, administración y recuperación de este tipo de activos.

[1] Consultada en www.uci.cu/mision

Cuando se realiza un análisis sobre los niveles de reutilización existentes en el proceso de desarrollo de software en el Centro se pueden detectar las siguientes deficiencias:

- Aunque existe un esfuerzo encomiable por alcanzar la integración entre proyectos con tecnologías similares, todavía los resultados son mínimos en este sentido. Se puede apreciar un escaso nivel de reutilización de componentes, propiciado fundamentalmente por la escasa comunicación entre equipos de desarrollo y la poca centralización del control de los activos de software en el Centro.
- La inexistencia de métodos o herramientas que faciliten la reutilización de componentes propicia la duplicidad del esfuerzo en actividades propias del desarrollo de software, retrasos en el cronograma de los proyectos e incide, en la mayoría de los casos, negativamente en la calidad de los resultados deseados.

En la UCI existe un repositorio de activos de software con una estructura genérica y un sistema de administración básico, sin embargo, entre los equipos de desarrollo del Centro no se tiene un alto conocimiento de su existencia por lo que no se utiliza. Además, el software no establece un sistema mediador para la búsqueda automática de los activos, ni permite ofrecer información sobre el historial de los activos de software [1].

En el Centro de Informatización de Entidades (CEIGE), adscrito a la Facultad 3, se desarrolló un intento por solucionar estos problemas [1]. Este tuvo como resultados las necesidades de la implementación de un Sistema para la Administración de un Repositorio de Activos de Software de acuerdo a las características del proceso de desarrollo del Centro, la elaboración de los artefactos definidos para las etapas de Análisis y Diseño para este sistema, quedando sentadas las bases para la futura implementación del mismo. El presente trabajo pretende incorporar los principales resultados de la solución referida anteriormente.

Cuando se analiza la situación existente, se identifican las siguientes deficiencias que deben resolverse para garantizar una gestión y control eficiente de los activos de software, en función de mejorar los niveles de reutilización de software en CEGEL:

- Escaso nivel de búsqueda de activos de software, lo que atenta contra la identificación y localización de los activos más idóneos para satisfacer determinados requisitos funcionales y no funcionales.

- Inexistencia del control de historial de uso de los activos de software que permita establecer un orden cronológico de la evolución del mismo, así como reconocer el índice de reutilización del mismo a partir de la experiencia de otros proyectos.
- Escaso nivel de asociación entre activos dentro de un repositorio de software que permita la identificación de relaciones de dependencias hacia otros activos de software.
- No se relaciona información sobre las licencias de uso, referencias sobre resultados de investigación que lo hayan generado, recomendaciones identificadas por los desarrolladores para el desarrollo evolutivo del mismo que son de gran importancia para la gestión de los mismos.

Este análisis de la problemática se consolida al evaluar los resultados de la encuesta sobre Reutilización de Software en el Centro aplicada a jefes de proyecto, arquitectos, analistas y desarrolladores en el mes de diciembre del año 2012 presentada en el Anexo 1 del presente documento. Este ejercicio tuvo el objetivo de identificar el nivel de institucionalización de la reutilización mediante la determinación de las actividades destinadas a este fin, su inclusión en el modelo de desarrollo y las herramientas que lo facilitan. Se obtuvieron los siguientes resultados, graficados en el Anexo 2.

- El concepto de reutilización lo domina el 91% de los encuestados, donde el mayor dominio del mismo lo presentan los analistas, arquitectos y programadores. Aunque este aspecto no es un elemento importante para determinar la aplicación del concepto en la práctica ofrece una idea general de que la mayoría conoce en qué consiste.
- De los encuestados, el 79% consideran que en el Centro no se trabaja orientado al proceso de reutilización, aunque sí reconocen actividades que se realizan en el mismo encaminadas a la reutilización fundamentalmente la implementación (98 %), el análisis y el diseño (53 %).
- No se tiene dominio del concepto de activo de software, esto se evidencia debido a que el72% de los encuestados presentan un conocimiento incompleto del mismo.
- El 61.9% de los encuestados plantea que a veces los activos de software son tomados en cuenta cuando se va a desarrolla un nuevo proyecto y el 34.2% que frecuentemente. Esto quiere decir que el proceso de desarrollo de software en el Centro no está orientado al uso de activos de software.

- El 100% de los encuestados consideran que es necesario un sistema para el control de los activos de software y proponen alguna de las principales funcionalidades que deberían incluirse en el mismo.

De manera general la problemática se evidencia a partir de los elementos antes descritos, consolidados a partir de los resultados obtenidos con la aplicación de la encuesta en CEGEL. Dando paso a la identificación del siguiente **problema a resolver:** ¿Cómo contribuir a la gestión de los activos de software de manera que se obtengan niveles de reutilización aceptables en el proceso de desarrollo de CEGEL?

El **objeto de estudio** es la Ingeniería y Administración de software centrada en la reutilización. El **objetivo general** es desarrollar un Sistema para el Control de Activos de Software que contribuya a la gestión de los mismos durante el proceso de desarrollo de software. Consecuentemente se identifica el **campo de acción** como la Ingeniería y Administración de activos de software en el proceso de desarrollo de CEGEL.

Para lograr el objetivo general de la investigación se plantean los siguientes **objetivos específicos**:

- Elaborar el marco teórico de la investigación.
- Desarrollar la propuesta de solución.
- Validar las etapas del proceso de desarrollo de software a partir de la validación de los artefactos de Requisitos, Diseño e Implementación.

Para guiar la investigación se plantea la siguiente **idea a defender**:

El desarrollo de un Sistema para el Control de Activos de Software contribuirá a mejorar los niveles de gestión de los mismos durante el proceso de desarrollo de software.

A través de las siguientes **tareas** se le dará cumplimiento a los objetivos específicos trazados:

- Caracterización de los principios de la Ingeniería y Administración de software centrada en la reutilización.
- Caracterización de soluciones existentes similares para el control de activos de software.
- Elaboración de historias de usuario para describir las funcionalidades del sistema.
- Especificación de requisitos funcionales y no funcionales del sistema.
- Descripción de las Tarjetas CRC.
- Implementación de los componentes de software.
- Validación de la solución a partir del diseño y aplicación de pruebas unitarias a las funcionalidades desarrolladas y pruebas de integración.

Los métodos científicos teóricos empleados son los siguientes:

- **Histórico - Lógico**: Se utiliza para el estudio dela evolución de las funcionalidades de los sistemas de control de activos de software, sus características, mejoras y optimización de implementación.
- **Analítico - sintético:** Se utiliza en el análisis y síntesis de la documentación utilizada como bibliografía para la fundamentación teórica de la investigación.

Los métodos científicos empíricos utilizados para darle solución a las tareas de la investigación son:

- **Encuesta:** Es una técnica de recogida de información por medio de preguntas escritas organizadas preguntas a profesionales que dominan la problemática en la que está enmarcada la investigación**.**

El presente trabajo se estructura en los siguientes capítulos:

Capítulo 1. Fundamentación teórica: En este capítulo se plantean los principales conceptos asociados a la Ingeniería y Administración de software centrada en la reutilización de activos. Se realiza un estudio del estado del arte de los sistemas de activos de software destacando las principales características, ventajas y desventajas que inciden en el estudio de factibilidad de desarrollar la propuesta de solución presentada. Se definen las tecnologías y metodología durante el desarrollo del sistema.

Capítulo 2.Características del sistema: En este capítulo se describen las características del sistema a desarrollar, se elaboran las historias de usuario, se describen las Tarjetas CRC y se realiza una descripción de la arquitectura. Además se obtienen los elementos que servirán de base para la implementación del sistema.

Capítulo 3. Implementación y Prueba: En este capítulo se describe el modelo de despliegue y los resultados alcanzados a partir de la implementación de los elementos del diseño obtenidos (Tarjetas CRC). Se describen y analizan documentan los resultados de las pruebas aplicadas que muestran la calidad de la solución alcanzada.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

Introducción

En este capítulo se abordan los elementos relacionados al marco teórico conceptual asociado a la problemática a resolver. Se presentan los conceptos y fundamentos teóricos necesarios del objeto de estudio para abordar la problemática. Se presenta un estudio de las características de los sistemas de control de activos de software, tanto en el ámbito internacional como en el nacional. Además, se realiza el estudio de las tecnologías, metodologías y herramientas de desarrollo de software con el objetivo de fundamentar la selección de las que serán utilizadas para la implementación del sistema.

1.1 Reutilización de software

La Reutilización es el uso de un activo para la solución de diferentes problemas. “Un activo reutilizable es un elemento tal como diseños, especificaciones, código fuente, documentación, unidades de prueba y manuales de procedimientos”, los cuales fueron diseñados para su uso en múltiples contextos [6].

Activos de software

En la bibliografía se encuentran varias definiciones de activos de software. Se destacan Bastarrica y María Cecilia que definen un activo de software como todos los artefactos producidos durante el desarrollo de un software y que son potencialmente reutilizables [4]. Montilva los clasifica además como una colección de partes de software (requisitos, diseños, componentes, casos de prueba, etc.) que se configuran y componen de una manera prescrita para producir los productos de una línea de productos [5].

Para los intereses de esta investigación se considera que un activo de software puede ser un modelo de dominio, código, esquemas de bases de datos, patrones de diseño, manuales de usuario, planes de producción, bases de datos, documentación, casos de prueba, planes de prueba, etc. [1]

Repositorios de activos de software

El concepto de repositorio de activo de software es bastante amplio. Comprende desde sencillos sistemas de almacenamiento hasta complejos entornos que incorporan además de los sistemas de almacenamiento, un conjunto de herramientas de ayuda al proceso de reutilización, por lo que se puede afirmar que es un soporte al proceso de reutilización [6]. Vainer y Rahman lo definen como una base de datos de información acerca de artefactos de ingeniería, producidos o utilizados por una empresa. Ejemplos

de tales artefactos incluyen software, documentos, mapas, sistemas de información y discretos fabricados [8].

"Un repositorio debe asegurar el almacenamiento y disponibilidad de los componentes para permitir su reutilización, garantizar la concentración de un gran volumen de datos y a su vez, la seguridad y protección de los mismos" [1].

Resumiendo los principales elementos relacionados anteriormente se considera como repositorio de activos de software una base de datos que posee información sobre los mismos y que permita la gestión de estos, así como su protección y mantenimiento.

Es frecuente encontrar el concepto repositorio como sinónimo de catálogo, sin embargo se reconocen diferencias conceptuales entre los dos términos:

"Un catálogo de reutilización es un conjunto de descripciones de activos con una referencia o apuntador de donde el activo es actualizado, almacenado o hay información acerca de cómo puede ser adquirido" [10].

"Un repositorio digital, es una base de datos con ciertas características adicionales: ubicación y enfoque institucional, centrados en productos de investigación, visibilidad vía Web, disponibilidad de documentos completos (para descargas según políticas de autenticación), información estructurada a partir de metadatos y sostenibilidad y administración garantizada del sistema a largo plazo" [14].

Después de haber estudiado los dos conceptos (catálogo y repositorio) se decidió utilizar para la propuesta de solución un repositorio de reutilización ya que permite descargar los activos de software a diferencia de los catálogos que solo permiten la localización de los recursos.

1.2 El control de activos de software en función de la reutilización

Existen numerosos factores que inciden en la disminución de los niveles de reutilización en una empresa de software:

- El desconocimiento sobre la existencia de activos de software, muchas veces influenciados por la comunicación inadecuada de equipos de desarrollo en este sentido.
- Escasa documentación de los activos de software que se generan lo que incide negativamente en la agilización de la búsqueda de los mismos.
- Descentralización del almacenamiento de los activos generados durante la actividad productiva de proyectos de software que propicia la falta de control y deteriora los niveles de gestión sobre los mismos.

El control efectivo de activos de software necesita del éxito de los siguientes procesos:

Registro de los activos de software: Este proceso permite el registro de un activo y sus metadatos asociados, como nombre, tipo de contenido, título, descripción, fecha de creación, autor, tamaño, control de versiones y dependencias hacia otros activos de software. El registro de esta información permite la localización de activos de software a través de la búsqueda por palabras claves y metadatos.

Actualización de activos de software: Este proceso permite la actualización de los metadatos de un activo de software en función de rectificar errores registrados previamente o registrar nuevas actualizaciones del mismo.

Búsqueda de activos de software: Este proceso satisface las necesidades de búsqueda y localización de activos de software. Fundamentalmente a través de palabras claves y requisitos funcionales y no funcionales.

Cuando se comienza un nuevo proyecto de software, la consulta a un sistema de control de activos es necesaria durante todo el ciclo de vida del mismo. Durante el proceso de Identificación de componentes es importante realizar la búsqueda a partir de requisitos funcionales y no funcionales de lo contrario se debe seguir el método de palabras claves.

1.3 Sistemas para el control de activos de software

Ámbito internacional

WindowsClient.Net, es un catálogo fue creado por la empresa Microsoft Corporation, posee patrones de desarrollo para la construcción de aplicaciones en plataformas Windows [15].

DoFactory es un marco de trabajo que comprende un completo repositorio digital de patrones de diseño de software desarrollado en tecnología .NET, específicamente C# y Visual Basic [16].

Quince es un catálogo desarrollado por Infragistics, Inc., provee patrones de experiencia e interfaz de usuario, entendiendo por experiencia de usuario como la forma en que un cliente percibe la facilidad de uso y eficiencia de un producto software o sistema [17].

Weliepatterns, es un catálogo que provee una gran cantidad de patrones de diseño de interacción [18].

Welicki [20], propone un modelo de meta-especificación y catalogación de patrones y conceptos como respuesta a las necesidades de gestión del conocimiento en este ámbito de la ingeniería del software. La arquitectura general de la solución propuesta se compone de un lenguaje de meta-especificación para describir a los patrones a un alto nivel de abstracción, un catálogo de patrones creados con ese lenguaje, una

infraestructura de catalogación y una herramienta de explotación del catálogo. Para verificar la factibilidad de la solución propuesta, el autor ha creado y evaluado un prototipo respecto a otras soluciones y enfoques existentes para demostrar que el modelo propuesto supera las dificultades recurrentes encontradas en otros enfoques.

En [21], se presenta una propuesta que soporta el diseño e implementación de aplicaciones basadas en componentes y aspectos. Este trabajo describe un entorno integrado de desarrollo (IDE, por sus siglas en inglés) para aplicaciones basadas en componentes y aspectos y una serie de herramientas para dar soporte al mismo. El entorno incluye una plataforma de gestión del repositorio de componentes y aspectos denominada **DAOP**, cuya característica principal es la composición dinámica de componentes y aspectos. Por otro lado se presenta un lenguaje de definición de arquitecturas (ADL o Architecture Definition Language), denominado DAOP-ADL, que sirve para la especificación de la arquitectura de una aplicación y que la plataforma usa para componer componentes y aspectos. El IDE incluye además, un repositorio de componentes y aspectos (RCA).

El trabajo presentado por López, Laguna y Marqués en [22], propone una aproximación para incorporar el proceso de reutilización desde las etapas iníciales del ciclo de vida del software. La propuesta se basa en generalizar escenarios del problema y compararlos con escenarios genéricos. Los primeros se obtienen con una herramienta automatizada que está en fase de desarrollo. La generalización se realiza mediante reducción de redes de Petri. Los escenarios genéricos se almacenan en un repositorio asociados a estructuras complejas de reutilización denominadas mecanos que enlazan elementos de análisis, diseño e implementación. La comparación se realiza mediante analogía. De este modo, con una estrategia para reutilizar software desde los requisitos funcionales, se ofrece una alternativa para acelerar el desarrollo de soluciones software con reutilización a la vez que se eleva el nivel de abstracción de los elementos reutilizables. La motivación principal del artículo es que los requisitos representan el conocimiento más abstracto del dominio y un alto porcentaje de ellos son reutilizables.

En el desarrollo del proyecto **UNC-Corpus Corpus** de diagramas UML para la solución de problemas de completitud en ingeniería de software se hace uso de repositorios de diagramas con el fin de optimizar la gestión de los activos de modelado que allí utilizan además de complementarlo con herramientas para la verificación de consistencia y creación de diagramas [23].

De manera general cuando se analizan estas soluciones se identifican que la mayoría de ellas son especializadas en un tipo específico de activo destacándose los

documentos y patrones de diseño, además son privativas lo que supone la inversión de costo para su adquisición.

En Cuba: Universidad de las Ciencias Informáticas

En Cuba se dan pasos importantes en vías de la reutilización de activos de software. La Universidad de las Ciencias Informáticas es la principal precursora de este movimiento mediante Comunidades.uci.cu, el cual es un Entorno Virtual de Desarrollo Colaborativo, espacio para la creación de comunidades virtuales orientadas al desarrollo de componentes y herramientas claves para la producción e idas de impacto que pueden estar disponibles a todo la comunidad universitaria así como a los graduados dispersos a lo largo y ancho del país. En la misma se pueden compartir activos de software para su utilización. Sin embargo está orientada fundamentalmente a fomentar proyectos de investigación y colaboración ofreciendo limitantes para el registro y control de activos de software que es el principal objetivo que persigue este trabajo.

1.4 Selección de tecnologías para el desarrollo

Las herramientas de desarrollo, lenguajes de modelado y programación así como la metodología de desarrollo de software a utilizar durante la elaboración de la propuesta de solución se describen a continuación:

PHP 5.2.5

PHP (acrónimo de "PHP: Hypertext Preprocessor") es un lenguaje interpretado de alto nivel embebido en páginas HTML y ejecutado en el servidor. Al nivel más básico, PHP puede hacer cualquier cosa que se pueda hacer con un script CGI, como procesar la información de formularios, generar páginas con contenidos dinámicos, o enviar y recibir *cookies*.

Una de las características más potentes y destacables de PHP es su soporte para acceso a un gran número de bases de datos. Además soporta el uso de otros servicios que usen protocolos como IMAP, SNMP, NNTP, POP3, HTTP y derivados. También se pueden abrir sockets de red directos (conocidos por su término en inglés raw sockets) e interactuar con otros protocolos. [24]

Entre sus características principales se encuentran:

- Orientado al desarrollo de aplicaciones web dinámicas con acceso a información almacenada en una base de datos.
- Es considerado un lenguaje fácil de aprender, ya que en su desarrollo se simplificaron distintas especificaciones, como es el caso de la definición de las variables primitivas, ejemplo que se hace evidente en el uso de *phparrays*.

- El código fuente escrito en PHP es invisible al navegador web y al cliente, ya que es el servidor el que se encarga de ejecutar el código y enviar su resultado HTML al navegador. Esto hace que la programación en PHP sea más segura y confiable.
- Capacidad de conexión con la mayoría de los motores de base de datos que se utilizan en la actualidad destacándose su conectividad con MySQL y PostgreSQL.
- Capacidad de expandir su potencial utilizando módulos (llamados ext's o extensiones).
- Posee una amplia documentación en su sitio web oficial, entre la cual se destaca la descripción de todas las funcionalidades del lenguaje y ejemplificadas en un único archivo de ayuda.
- Es libre, por lo que se presenta como una alternativa de fácil acceso para todos los desarrolladores.
- Permite aplicar técnicas de programación orientada a objetos. Incluso aplicaciones como Zend Framework, empresa que desarrolla PHP, están totalmente desarrolladas mediante esta metodología.
- No requiere definición de tipos de variables aunque sus variables se pueden evaluar también por el tipo que estén manejando en tiempo de ejecución.
- Proporciona ventajas para el manejo de excepciones (desde PHP5).

Si bien PHP no obliga a quien lo usa a seguir una determinada metodología durante el desarrollo, aun haciéndolo, el programador puede aplicar en su trabajo cualquier técnica de programación que le permita escribir código ordenado, estructurado y manejable. Un ejemplo representativo son los desarrollos que en PHP se han hecho del diseño Modelo (MVC), que permiten separar el tratamiento y acceso a los datos, la lógica de control y la interfaz de usuario en tres componentes independientes.

Como es un lenguaje que se interpreta en ejecución, para ciertos usos puede resultar un inconveniente que el código fuente no pueda ser ocultado. La ofuscación es una técnica que puede dificultar la lectura del código pero no necesariamente impide que el código sea examinado.

Debido a que es un lenguaje interpretado, un script en PHP suele funcionar considerablemente más lento que su equivalente en un lenguaje de bajo nivel, sin embargo este inconveniente se puede minimizar con técnicas de cache tanto en archivos como en memoria.

Las variables al no ser tipificadas dificulta a los diferentes entornos integrados de desarrollo (IDE) para ofrecer asistencias para el completamiento del código, aunque esto no es realmente un inconveniente del lenguaje en sí. Esto es solventado por Zend Studio añadiendo un comentario con el tipo a la declaración de la variable. [33]
PHP constituye uno de los lenguajes de programación web más difundidos y utilizados actualmente, principalmente debido a las potencialidades descritas anteriormente, lo que determina que es una tecnología adecuada para el desarrollo de la solución propuesta frente a otros lenguajes de programación similares.

BOOTSTRAP2.0

Bootstrap es un marco de trabajo que simplifica el proceso de creación de diseños web combinando CSS y JavaScript. Ha sido desarrollado por Twitter[2] que recientemente liberó su versión 2.0. La mayor ventaja es que permite crear interfaces que se adapten a los distintos navegadores (técnica conocida por su término en inglés como responsive design) con el apoyo de un marco de trabajo potente con numerosos componentes webs que minimizan esfuerzo y tiempo.
Entre las características principales que hicieron de Bootstrap una buena elección para el desarrollo de la propuesta de solución, se encuentran las siguientes:

- Bootstrap ofrece una serie de plantillas CSS y ficheros JavaScript que permiten integrar el marco de trabajo de forma sencilla y potente en proyectos con tecnología web.
- Permite crear interfaces que se adapten a los diferentes navegadores, tanto de escritorio como tablets y móviles a distintas escalas y resoluciones.
- Se integra perfectamente con las principales bibliotecas de Java, por ejemplo JQuery.
- Ofrece un diseño sólido usando LESS y estándares como CSS3/HTML5.
- Es un marco de trabajo ligero que se integra de forma limpia al proyecto actual.
- Funciona con todos los navegadores, incluido Internet Explorer usando HTML Shim para que reconozca los tags HTML5.
- Dispone de distintos layout predefinidos con estructuras fijas a 940 píxeles de distintas columnas o diseños fluidos.

[2]Twitter actualmente es considerada como una de las redes sociales más difundidas en el ciberespacio con más de 500 millones de usuarios https://**twitter**.com

Bootstrap ofrece distintos componentes que se pueden usar con unos estilos predefinidos y fáciles de configurar. Además de los distintos plugins integrados con JQuery que están disponibles en la versión 2.0 de Bootstrap.

Botones: se pueden aplicar tanto a enlaces como a etiquetas button/input simplemente usando la clase btn. Así se puede distinguir el propósito de cada botón con los distintos estilos prefijados o variar su tamaño.

Dropdown: Permiten asociar un menú desplegable a un botón de forma sencilla agrupando distintas opciones.

Formularios: Bootstrap cuenta con distintos *layouts* adaptables a las principales necesidades. Además de incluir distintos elementos para remarcar distintas acciones sobre los formularios: *focused, disabled, control-group,* entre otros.

Plugin de JQuery: Bootstrap cuenta con distintos plugins que permiten crear ventanas modales, o crear el típico tooltip sobre algún elemento de la página. [34]

SYMFONY 2.0

Symfony es un marco de trabajo diseñado para mejorar el desarrollo de las aplicaciones web basado en el patrón Modelo Vista Controlador. Para empezar, separa la lógica de negocio, la lógica de servidor y la presentación de la aplicación web. Simplifica el desarrollo de aplicaciones mediante la automatización de varios patrones empleados para un propósito dado. Además se integra al código, lo que provoca que el desarrollador escriba mejor, más legible y un código más estable. En última instancia, provoca que se programe más fácil, ya que empaqueta operaciones complejas en simples declaraciones.

Sinfonía es un marco de trabajo diseñado para optimizar el desarrollo de aplicaciones web por medio de varias características claves. Contiene numerosas herramientas y clases para acortar el tiempo de desarrollo de una aplicación web compleja. Además, automatiza tareas comunes de manera que el desarrollador puede concentrarse por completo en los detalles específicos de una aplicación. Además, automatiza las tareas más comunes, permitiendo al desarrollador dedicarse por completo a los aspectos específicos de cada aplicación. El resultado de todas estas ventajas es que no se debe reinventar la rueda cada vez que se crea una nueva aplicación web.

Symfony está desarrollado completamente en PHP. Ha sido probado en numerosos proyectos reales y se utiliza en sitios web de comercio electrónico de primer nivel. Es compatible con la mayoría de gestores de bases de datos, como MySQL, PostgreSQL, Oracle y Microsoft SQL Server. Se puede ejecutar tanto en plataformas *nix (Unix, Linux, etc.) como en plataformas Windows. Es compatible con la mayoría de los motores de

bases de datos disponibles, incluyendo MySQL, PostgreSQL, Oracle, y Microsoft SQL Server.

Symfony fue diseñado para ajustarse a los siguientes requisitos:

- Fácil de instalar y configurar en la mayoría de plataformas (y con la garantía de que funciona correctamente en los sistemas Windows y *nix estándares).
- Independiente del sistema gestor de bases de datos. Su capa de abstracción y el uso de Propel, permiten cambiar con facilidad de Sistema de Gestión de Bases de Datos (SGBD) en cualquier fase del proyecto.
- Sencillo de usar en la mayoría de casos, aunque es preferible para el desarrollo de grandes aplicaciones Web que para pequeños proyectos.
- Aunque utiliza el patrón MVC (Modelo Vista Controlador), tiene su propia forma de trabajo en este punto, con variantes del MVC clásico como la capa de abstracción de base de datos, el controlador frontal y las acciones.
- Basado en la premisa de “convenir en vez de configurar”, en la que el desarrollador sólo debe configurar aquello que no es convencional.
- Sigue la mayoría de mejores prácticas y patrones de diseño para la web.
- Preparado para aplicaciones empresariales y adaptables a las políticas y arquitecturas propias de cada empresa, además de ser lo suficientemente estable como para desarrollar aplicaciones a largo plazo.
- Código fácil de leer que incluye comentarios de phpDocumentor y que permite un mantenimiento muy sencillo.
- Fácil de extender, lo que permite su integración con las bibliotecas de otros fabricantes.
- Una potente línea de comandos que facilitan generación de código, lo cual contribuye a ahorrar tiempo de trabajo.

Las características más comunes para el desarrollo de proyectos web están automatizadas en Symfony, tales como:

- Permite la internacionalización para la traducción del texto de la interfaz, los datos y el contenido de localización.
- La presentación usa plantillas y formularios que pueden ser construidos por diseñadores de HTML que no tienen que poseer necesariamente conocimientos de la tecnología.
- Los formularios soportan la validación automática, lo cual asegura mejor calidad de los datos en las base de datos y una mejor experiencia para el usuario.
- El manejo de cache reduce el uso de banda ancha y la carga del servidor.

- La facilidad de soportar autenticación y credenciales facilita la creación de áreas restringidas y manejo de seguridad de los usuarios.
- El enrutamiento y las URLs inteligentes hacen amigables las direcciones de las páginas de la aplicación.
- Las listas son más amigables, ya que permite la paginación, clasificación y filtraje automáticos.
- Los plugins proveen un alto nivel de extensibilidad.
- La interacción con AJAX es mucho más sencilla.

Las ventajas tecnológicas de Symfony [35]:

Más rápido y menos codicioso: Symfony fue concebido desde el principio para ser rápido y para favorecer el rendimiento. A modo de comparación, Symfony es aproximadamente 3 veces más rápido que la versión 1.4 o de 1.10 Zend Framework, teniendo hasta 2 veces menos memoria.

Comunidad: Hay todo un ecosistema que ha crecido alrededor de Symfony desde su lanzamiento: la comunidad (listas de correo, IRC, etc) y las empresas de servicios de muchos otros que han invertido en el marco de trabajo. Por último, también es con miras a un desarrollo sostenible que Symfony se distribuye bajo licencia Open Source MIT, que no impone restricciones y permite el desarrollo de código abierto, así como aplicaciones propietarias.

Ampliable: Desde la más pequeña pieza de código a la base completa en sí, todo lo que se presenta como un "paquete" (o plug-in en el lenguaje Symfony) en Symfony. Cada paquete está destinado para añadir funcionalidad al marco, por supuesto, y cada paquete también puede ser reutilizado en otro proyecto o compartido con el resto de la comunidad.

Estable y sostenible: Desarrollado por los Laboratorios Sensio, las principales versiones de Symfony son soportados por 3 años por la empresa e incluso de por vida en cuanto a las cuestiones relacionadas con la seguridad. Para mayor estabilidad, las versiones menores del contrato Symfony y la interfaz también están garantizadas y la compatibilidad entre todas las versiones secundarias se llevará a cabo en el API (Interfaz de programación de aplicaciones (IPA) o *API* del inglés *Application Programming Interface*) definido por las interfaces públicas.

Entusiasmo para desarrollar: En un entorno altamente funcional, Symfony también garantiza un cierto nivel de comodidad para los desarrolladores. Al cuidar de una serie de tareas desagradables (desarrollo de funcionalidades de menor importancia, por ejemplo), permite a los desarrolladores centrarse en los aspectos más destacados

reales de una aplicación completamente, validar su trabajo y mejorar su productividad. Entre las herramientas de Symfony que facilitan el trabajo del desarrollador está la barra de herramientas de depuración web, así como soporte nativo para entornos de desarrollo, páginas de error detallados o incluso de seguridad nativa.

Facilidad de uso: Completamente flexible para satisfacer las necesidades de los profesionales y usuarios avanzados por igual, Symfony es también muy accesible. Abundante documentación, la comunidad y el apoyo profesional así como las mejores prácticas dentro del marco permiten que un principiante se sienta muy rápidamente a gusto con Symfony.

HTML 5

HTML es el lenguaje de marcado de hipertexto usado para la construcción de páginas web. Fue creado en 1986 por Tim Berners Lee; a partir de dos herramientas preexistentes: El concepto de Hipertexto, que permite conectar dos elementos entre si y el SGML (Lenguaje Estándar de Marcación General) para colocar etiquetas o marcas en un texto que indique como debe verse. HTML no presenta ningún compilador, por lo tanto algún error de sintaxis que se presente este no lo detectará y se visualizará en la forma como el navegador lo entienda. El conjunto de etiquetas que se creen para la construcción de una página web, se deben guardar con la extensión htm o html. Estos documentos pueden ser mostrados por los buscadores o "browsers" de páginas Web en Internet, como Mozilla Firefox, Internet Explorer, Netscape Navigator, Mosaic, Opera, entre otros. [27]

HTML5 es la quinta revisión importante del lenguaje básico de la World Wide Web, HTML. HTML5 especifica dos variantes de sintaxis para HTML: un «clásico» HTML (text/html), la variante conocida como HTML5 y una variante XHTML conocida como sintaxis XHTML5 que deberá ser servida como XML (XHTML) (aplicación/xhtml+xml). Esta es la primera vez que HTML y XHTML se han desarrollado en paralelo.

Todavía se encuentra en modo experimental, lo cual indica el Consorcio World Wide Web (*W3C*); aunque ya es usado por múltiples desarrolladores web por sus avances, mejoras y ventajas.

HTML5 establece una serie de nuevos elementos y atributos que reflejan el uso típico de los sitios web modernos. Algunos de ellos son técnicamente similares a las etiquetas y , pero tienen un significado semántico, como por ejemplo <nav> (bloque de navegación del sitio web) y <footer>. Otros elementos proporcionan nuevas funcionalidades a través de una interfaz estandarizada, como los elementos <audio> y <video>.Mejoras en el elemento <canvas>, capaz de renderizar en los navegadores más

importantes (Mozilla, Chrome, Opera, Safari e Internet Explorer) elementos 3D.Algunos elementos de HTML 4.01 han quedado obsoletos, incluyendo elementos puramente de presentación, como <font> y , cuyos efectos son manejados por el CSS. También hay un renovado énfasis en la importancia del scripting DOM para el comportamiento de la web.

Otras de las características de esta versión es que incorpora nuevos elementos tales como:

- Etiquetas (canvas 2D y 3D, audio, video) con codecs para mostrar los contenidos multimedia. Actualmente hay una lucha entre imponer codecs libres (WebM + VP8) o privados (H.264/MPEG-4 AVC).
- Etiquetas para manejar grandes conjuntos de datos: Datagrid, Details, Menu y Command. Permiten generar tablas dinámicas que pueden filtrar, ordenar y ocultar contenido en cliente.
- Mejoras en los formularios. Nuevos tipos de datos (email, number, url, datetime) y facilidades para validar el contenido sin Javascript.
- Visores: MathML (fórmulas matemáticas) y SVG (gráficos vectoriales). En general se deja abierto a poder interpretar otros lenguajes XML.
- Drag&Drop. Nueva funcionalidad para arrastrar objetos como imágenes.

Nuevas APIs y Javascript:

- API para hacer Drag&Drop mediante eventos.
- API para trabajar Off-Line. Permite descargar todos los contenidos necesarios y trabajar en local.
- API de geoposicionamiento para dispositivos que lo soporten.
- API Storage. Facilidad de almacenamiento persistente en local, con bases de datos (basadas en SQLite) o con almacenamiento de objetos por aplicación o por dominio Web (Local Storage y Global Storage). Se dispone de una base de datos con la posibilidad de hacer consultas SQL.
- WebSockets. API de comunicación bidireccional entre páginas. Similar a los Sockets de C.
- WebWorkers. Hilos de ejecución en paralelo.
- Estándar Futuro. System Information API. Acceso al hardware a bajo nivel: red, ficheros, CPU, Memoria, puertos USB, cámaras y micrófonos, muy interesante pero con numerosas salvedades de seguridad. [37]

NetBeans 7.2

NetBeans IDE es un software de código abierto sin restricciones de uso, escrito completamente en Java, teniendo el código fuente disponible para su reutilización bajo las licencias CDDL (Common Development and Distribution Licence) y la GPL (General Public Licence). Es multiplataforma, presenta una interfaz muy amigable e intuitiva, soporta disímiles lenguajes, además en sus versiones más recientes se le han ido acoplando nuevas funcionalidades que resultan de gran utilidad para el desarrollo web, facilitando la creación de proyectos en PHP. Por otra parte el editor de PHP es muy ágil y robusto, siendo eficiente en el completamiento de código, reconocimiento de sintaxis y presentando un potente depurador que facilita la programación. Además con NetBeans se pueden crear interfaces de usuarios debido a que brinda la posibilidad de utilizar componentes gráficos para aplicaciones de escritorio de gran calidad. Brinda la posibilidad de utilizar proyectos creados por otros Entornos Integrados de Desarrollo como es el caso de Eclipse. [30].

Otras de las posibilidades que proporciona es el desarrollo de aplicaciones multiplataforma sobre Mac OS, Windows y Linux. Incluye además las siguientes características: [32]

- Soporte para XML, HTML así como Java, C y C++.
- Permite control de versiones con CVS y SVN así como la compilación avanzada con Ant.
- Permite la creación visual de componentes gráficos para el desarrollo de interfaces de usuario.
- Dispone de todo un entorno para crear la documentación del software que se desarrolla.

PostgreSQL 9.1

PostgreSQL es un Sistema de Gestión de Base de datos Objeto-Relacional, bajo licencia BSD (Berkeley Software Distribution). Este se ha convertido en el sistema de gestión de bases de datos de código abierto más potente del mercado, contando ya con una arquitectura suficientemente probada, que se ha ganado una sólida reputación de confiabilidad, integridad de datos y corrección. PostgreSQL es multiplataforma, permite el acceso desde los lenguajes de programación más populares, incluyendo PHP.

PostgreSQL ofrece muchas ventajas respecto a otros sistemas gestores de bases de datos [30] [31]:

- Tiene una comunidad amplia de desarrollo colaborativo lo que facilita el soporte y mantenimiento de los sistemas que lo utilizan.

- Ahorros considerables en costos de operaciones.
- Estabilidad y confidencialidad legendarias.
- Multiplataforma.
- Diseñado para ambientes de alto volumen de datos.
- Herramientas gráficas para la gestión de la base de datos, el diseño y la administración.

PostgreSQL utiliza un modelo cliente/servidor y usa multiprocesos en vez de multihilos para garantizar la estabilidad del sistema. Un fallo en uno de los procesos no afectará el resto y el sistema continuará funcionando.

La última serie de producción es la 9.1. Sus características técnicas la hacen uno de los sistemas de bases de datos más potentes y robustos del mercado. Su desarrollo comenzó hace más de 16 años, y durante este tiempo, estabilidad, potencia, robustez, facilidad de administración e implementación de estándares han sido las características que más se han tenido en cuenta durante su desarrollo. PostgreSQL funciona muy bien con grandes cantidades de datos y una alta concurrencia de usuarios accediendo a la vez al sistema.

Algunas de las características más importantes y soportadas por PostgreSQL que fundamentan la selección de esta herramienta para la propuesta de solución:

- Es una base de datos 100% ACID, es decir garantiza las propiedades de atomicidad, consistencia, aislamiento y durabilidad de las transacciones.
- Integridad referencial.
- Replicación asincrónica/sincrónica / Streaming replication - Hot Standby.
- Copias de seguridad en caliente (Online/hotbackups).
- Regionalización por columna.
- Múltiples métodos de autentificación.
- Acceso encriptado vía SSL.
- Actualización in-situ integrada (pg_upgrade).
- Completa documentación.
- Disponible para Linux y UNIX en todas sus variantes (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64) y Windows 32/64bit.
- Soporta el almacenamiento de objetos binarios grandes (gráficos, videos, sonido)
- Mediante un sistema denominado Acceso concurrente multiversión (MVCC, por sus siglas en inglés) PostgreSQL permite que mientras un proceso escribe en una tabla, otros accedan a la misma tabla sin necesidad de bloqueos. Cada

usuario obtiene una visión consistente de lo último a lo que se le hizo commit. Esta estrategia es superior al uso de bloqueos por tabla o por filas común en otras bases de datos, eliminando la necesidad del uso de bloqueos explícitos.

- Disparadores (triggers): Un disparador o trigger se define como una acción específica que se realiza de acuerdo a un evento, cuando este ocurra dentro de la base de datos. En PostgreSQL esto significa la ejecución de un procedimiento almacenado basado en una determinada acción sobre una tabla específica. Este SGBD permite crear una amplia funcionalidad a través de su sistema de activación de disparadores (triggers).
- Bloques de código que se ejecutan en el servidor. Pueden ser escritos en varios lenguajes, con la potencia que cada uno de ellos va, desde las operaciones básicas de programación, tales como bifurcaciones y bucles, hasta las complejidades de la programación orientada a objetos o la programación funcional.
- PostgreSQL es extensible, o sea permite la extensión de sus funcionalidades en el servidor a través de los siguientes lenguajes:
    - Un lenguaje propio llamado PL/PgSQL.
    - C.
    - C++.
    - JavaPL/Java web.
    - PL/Perl.
    - plPHP.
    - PL/Python.

Gracias a su licencia BSD, se permite la utilización del código para ser comercializado. Uno de los casos ejemplo es la de Enterprise DB (PostgreSQL Plus), la cual incluye varios agregados y una interfaz de desarrollo basada en Java. Entre otras empresas que utilizan PostgreSQL para comercializar se encuentra CyberTech (Alemania), con su producto CyberCluster. [38]

Servidor Apache 2.0

Apache es un servidor web gratuito, potente y que ofrece un servicio estable y sencillo de mantener y configurar. Las principales características que ofrece son las siguientes:

- Es multiplataforma, aunque idealmente está preparado para funcionar en entornos Unix.

- Posee una configuración básica para usuarios estándares aunque permite configuraciones avanzadas en función del nivel de servicios que debe ofrecer el servidor de aplicaciones.
- Es una tecnología gratuita de código fuente abierta.
- Tiene integración con múltiples tecnologías como PHP, aspecto muy provechoso para el trabajo en cuestión debido a la necesaria integración con este lenguaje.
- Presenta un desarrollo modular lo que permite incorporarle nuevas funcionalidades de forma muy simple.

Apache tiene amplia aceptación en la red: desde 1996, es el servidor HTTP más usado en el mundo y alcanzó su máxima cuota de mercado en 2005 siendo el servidor empleado en el 70% de los sitios web en el mundo. [28]

UML 2.0

Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. Está consolidado como el lenguaje estándar en el análisis y diseño de sistemas de cómputo. Mediante él es posible establecer la serie de requisitos y estructuras necesarias para plasmar un sistema de software previo al proceso intensivo de escribir código. Ofrece un estándar para describir un "modelo" del sistema, incluyendo aspectos conceptuales tales como procesos de negocio, funciones del sistema y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes reutilizables. Aunque es un lenguaje, posee más características visuales que programáticas, las cuales facilitan a integrantes de un equipo multidisciplinario participar e intercomunicarse fácilmente, siendo estos integrantes: los analistas, diseñadores, especialistas de área y desde luego los programadores. Un modelo UML puede ser usado en todas las fases de desarrollo de software, lo que facilita su entendimiento y la comunicación. [41]

De acuerdo a la metodología de software utilizada, el lenguaje de modelado permitirá realizar el Diagrama de Entidad-Relación para el modelado de la base de datos y el Diagrama de Despliegue.

Visual Paradigm 8.0

Es una herramienta de modelado basada en la notación UML profesional que soporta el ciclo de vida completo del proceso de 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 todo tipo de diagramas, código inverso, generar código desde diagramas y generar documentación. [42]

Visual Paradigm constituye una poderosa herramienta de software respaldada bajo licencia gratuita y comercial, de probada utilidad para los analistas. Propicia un conjunto de ayudas para el desarrollo de programas informáticos. Permite la sincronización entre el código fuente y el modelo en tiempo real y brinda soporte para toda la notación UML. Fue diseñado para una amplia gama de usuarios interesados en la construcción de sistemas de software de forma fiable a través de la utilización de un enfoque orientado a objetos.

Se caracteriza por el diseño centrado en casos de uso y enfocado al negocio, lo que permite generar un software de mayor calidad. Usa un lenguaje estándar común a todo el equipo de trabajo facilitando la comunicación con modelo y código sincronizados en todo el ciclo de desarrollo, generación de código para Java y exportación como HTML siendo fácil de instalar y actualizar.

Metodología de desarrollo de software XP

XP es la metodología de desarrollo de software ágil más destacada en los últimos años. Se diferencia de las metodologías tradicionales principalmente en que pone más énfasis en la adaptabilidad que en la previsibilidad. Surge como respuesta y posible solución a los problemas derivados del cambio en los requisitos. En la mayoría de los casos se plantea como una metodología a emplear en los proyectos con riesgos de requisitos muy cambiantes.

Los defensores de XP consideran que los cambios de requisitos sobre la marcha son un aspecto natural, inevitable e incluso deseable del desarrollo de software y creen que ser capaces de adaptarse a estos cambios en cualquier punto de la vida del proyecto es una aproximación mejor y más realista que intentar definir todos los requisitos al comienzo del proyecto e invertir esfuerzos después en controlar los cambios en los requisitos.

En su ciclo de vida se realizan pruebas unitarias continuas, frecuentemente repetidas y automatizadas, incluyendo pruebas de regresión. Emplea la refactorización del código, es decir, se reescriben ciertas partes del código para aumentar su legibilidad pero sin modificar su comportamiento. En este sentido existen un conjunto de buenas prácticas que deben aplicarse durante el desarrollo de software con XP:

- **Pruebas unitarias continuas**: frecuentemente repetidas y automatizadas, incluyendo pruebas de regresión. Se aconseja escribir el código de la prueba antes de la codificación.

- **Corrección de todos los errores:** se realiza antes de añadir nuevas funcionalidades. Se recomienda hacer entregas frecuentes.
- **Refactorización del código:** reescribir ciertas partes del código para aumentar su legibilidad y mantenibilidad pero sin modificar su comportamiento. Las pruebas han de garantizar que en la refactorización no se ha introducido ningún fallo.
- **Propiedad del código compartida:** en vez de dividir la responsabilidad en el desarrollo de cada módulo en grupos de trabajo distintos, este método promueve el que todo el personal pueda corregir y extender cualquier parte del proyecto. Las frecuentes pruebas de regresión garantizan que los posibles errores serán detectados. [25]

XP es muy parecido a un rompecabezas debido a que hay muchas piezas pequeñas. Individualmente, las piezas no tienen sentido, pero cuando se combinan, se puede ver un panorama completo. [26]

La Metodología XP plantea cuatro fases durante el desarrollo de software como se muestra en la siguiente figura:

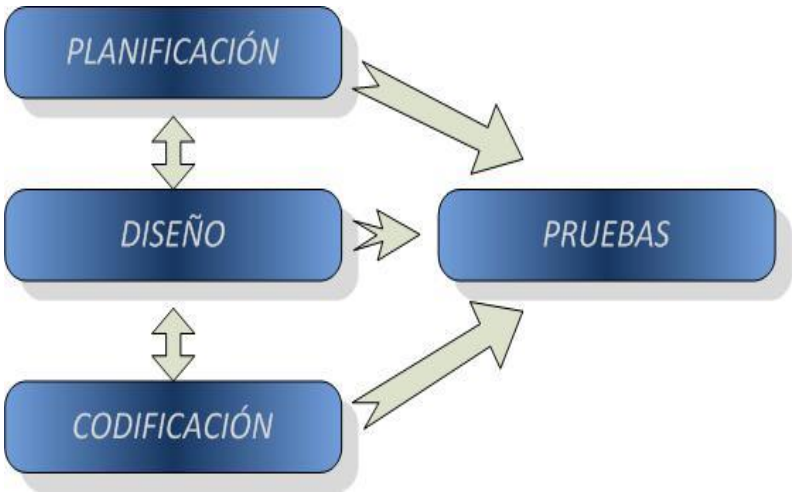

Figura 1 Fases de la Metodología XP.

1ª Fase: Planificación del proyecto

Esta fase tiene como objetivo fundamental la planificación de todo el proceso de desarrollo de software a partir de la estimación del tamaño de la solución. Comienza con una recopilación de todos los requisitos del sistema, a partir de la interacción con el usuario describiendo las Historias de Usuario. Posteriormente se debe planificarlas

próximas actividades teniendo en cuenta el alcance y los recursos humanos y materiales disponibles para lograr de forma efectiva los objetivos finales.

2ª Fase: Diseño

XP plantea que deben conseguirse diseños simples y sencillos fácilmente entendibles e implementables que a la larga costará menos tiempo y esfuerzo para desarrollarlos. En esta fase se obtienen como resultados importantes el diseño de interfaces de usuario, el diseño de bases de datos y la descripción de Tarjetas CRC que sirven de base para la implementación.

3ª Fase: Codificación

XP involucra al cliente como una parte más del equipo de desarrollo; su presencia es indispensable en las distintas fases de la metodología. A la hora de codificar una historia de usuario su presencia es aún más necesaria porque tienen un papel importante en la planificación de esta actividad y en la priorización de Historias de Usuario. Antes del desarrollo de cada historia de usuario el cliente debe especificar detalladamente lo que esta hará y también tendrá que estar presente cuando se realicen los test que verifiquen que la historia implementada cumple la funcionalidad especificada. En esta fase de la codificación o implementación los clientes y los desarrolladores del proyecto deben estar en comunicación para que los desarrolladores puedan codificar todo lo necesario para el proyecto que se requiere. El principal artefacto que se obtiene en esta etapa es el código fuente del software desarrollado y el esquema de bases de datos donde se almacenará la información.

4ª Fase: Pruebas

Uno de los pilares de la metodología XP es el uso de pruebas para comprobar el funcionamiento del código que se va generando. Para esta fase lo que se implementa es el uso de test que son pruebas que se le hacen al software o partes del software que se van obteniendo de forma iterativa. Las pruebas se aplican en todas las etapas anteriores del proceso de desarrollo lo que permite rectificar errores ocurridos de forma temprana.

Las ventajas de esta metodología están relacionadas a los siguientes elementos:

- Programación organizada.
- Menor taza de errores.
- Satisfacción del programador.
- Solución de errores de programas.
- Versiones nuevas.

- Implementa una forma de trabajo donde se adapte fácilmente a las circunstancias.

Aunque es recomendable emplearlo solo en proyectos a corto plazo. Algunas de las desventajas que presenta son:

- Altas comisiones en caso de fallar.
- Imposible prever todo antes de programar.
- Demasiado costoso.

De manera general se puede plantear que la principal ventaja de la metodología XP está en su alto grado de adaptabilidad y flexibilidad para proyectos pequeños, y su principal desventaja es su elevado costo en caso de no cumplir las metas.

Para seleccionar estas tecnologías se tuvieron en cuenta aspectos relacionados fundamentalmente al grado de conocimiento, sobre las mismas, de los autores del trabajo; su clasificación como software libre y gratuito y como elemento determinante la propuesta del Centro del uso e integración de las mismas para el desarrollo de aplicaciones de este tipo.

Conclusiones parciales

Con el desarrollo de este capítulo se arriban a las siguientes conclusiones parciales:

Los elementos que componen el objeto de estudio permitieron determinar que la solución pretenda clasificarse como un repositorio para el almacenamiento de activos de software y su catalogación, en pos de favorecer la reutilización de estos activos en el proceso de desarrollo de software de CEGEL.

El estudio de sistemas como MicrosoftClient.NET, Weliepatterns, Quince, DoFactory y Comunidades.uci.cu posibilitaron comprender los métodos de funcionamiento de los elementos fundamentales para estos tipos de solución con el objetivo de identificar una aproximación acertada en la solución que se diseña.

Fue posible identificar, el uso de la metodología ágil de desarrollo de software XP, PHP5 como lenguaje de programación, PostgreSQL 9.1 como Sistema de Gestión de Bases de Datos, NetBeans 7.2 como Entorno Integrado de Desarrollo, Bootstrap 2.0 como biblioteca para el diseño de interfaces de usuario, Symfony 2.0 como marco de trabajo integrado para el desarrollo, HTML 5 para presentación de interfaces de usuario y Apache 2.0 para servidor de aplicaciones como las herramientas que soportan el proceso de desarrollo de la solución que se persigue.

CAPÍTULO 2: CARACTERÍSTICAS DEL SISTEMA

Introducción

En el siguiente capítulo se describe la propuesta de solución, mediante el análisis y descripción de las dos primeras fases de la metodología XP; Planificación y Diseño generando de esta forma los artefactos correspondientes a dichas etapas.

2.1 Descripción del sistema

El objetivo de ese sistema es contribuir al control de activos de software en CEGEL ofreciendo facilidades para el registro y búsqueda de activos de este tipo que se produzcan en la entidad. Para acceder al sistema el usuario deberá haberse autenticado y podrá realizar determinadas operaciones en correspondencia con los permisos otorgados al mismo. Para esto se han definido los siguientes roles:

Usuario: este rol tendrá permiso a crear, modificar, eliminar y buscar activos de software.

Administrador: este rol podrá hacer todas las acciones que realizan los usuarios, además de la creación, modificación y eliminación de usuarios.

2.2 Planificación

La actividad de la planificación comienza creando una serie de historias (llamadas historias de usuario) que describen las características y la funcionalidad requeridas para el software que se construirá.

Cada historia de usuario (HU) la escribe el cliente y se coloca en una carta índice. El cliente le asigna un valor(es decir prioridad) a la historia basándose en los valores generales del negocio respecto de la característica o la función. Los miembros del equipo de la programación extrema (XP) evalúan entonces cada historia y le asignan un costo, el cual se mide en semanas de desarrollo. Si la historia requiere más de tres semanas de desarrollo, se le pide al cliente que la divida en las historias menores, y se realiza de nuevo la asignación del valor y el costo. Es importante destacar que las historias nuevas pueden escribirse en cualquier momento.

Los clientes y el equipo de desarrollo trabajan juntos para decidir cómo agrupar las historias hacia el próximo lanzamiento (el siguiente crecimiento de software) para que el equipo las desarrolle. Una vez establecido el compromiso básico (el acuerdo de las historias que se incluirán, la fecha de entrega y otras cuestiones del proyecto) para un lanzamiento, el equipo ordena las historias que se desarrollaran.

Después de que se ha entregado el primer lanzamiento del proyecto (también llamado incremento de software), el equipo calcula la velocidad del proyecto. Es decir, la

velocidad del proyecto es el número de historias de los clientes implementado durante el primer lanzamiento. Si se presenta un compromiso excesivo, el contenido de los lanzamientos se modifica o se cambian las fechas de las entregas finales. Conforme avanza el trabajo de desarrollo, el cliente puede agregar historias, cambiar el valor de la historia existente, dividir historias o eliminarlas. Entonces el equipo de XP considera de nuevo los lanzamientos restantes y modifica sus planes de acuerdo con ello. [39]

2.3 Historias de Usuarios

Una Historia de usuario es un relato acerca de qué problema debe resolver el sistema. Cada relato se escribe en una tarjeta y representa una parte de la funcionalidad que es coherente para el cliente. Son escritos por el cliente o usuario, con la ayuda de los desarrolladores, para permitir estimar los tiempos y asignar prioridades. Los clientes ayudan a asegurar que la mayoría de la funcionalidad deseada para el sistema está cubierta con las historias. Constan de 3 o 4 líneas escritas por el cliente en un lenguaje no técnico sin hacer mucho hincapié en los detalles; no se debe hablar ni de posibles algoritmos para su implementación, ni de diseños de base de datos adecuados, etc. [26] Mediante las historias de usuarios se realizó una descripción de varias características de las funcionalidades determinando la prioridad de cada una, debido al grado de importancia para el sistema de su realización. Otro de los parámetros es el Riesgo en desarrollo, que determina si es vulnerable o estable. Los puntos estimados, definidos por semanas, es la manera de medir el tiempo de desarrollo de la funcionalidad, junto a la iteración; que no es más que el grupo de desarrollo donde será implementada. Por último se encuentra la descripción de la funcionalidad, donde se muestra para qué se utiliza. Todas las Historias de Usuario elaboradas se encuentran en el Anexo 3 A continuación se describe la HU Gestionar Activos de Software considerada de gran impacto en la propuesta de solución.

Tabla 1 HU Gestionar activos de software.

Historia de Usuario	
Número: 1	Usuario: Todos los usuarios.
Nombre de historia: Gestionar activos de software.	
Prioridad en negocio: Alta.	Riesgo en desarrollo: Baja.
Puntos estimados: 2 semanas.	Iteración asignada: 1
Programador responsable: Víctor Hernández Martiartu.	

Descripción: Los usuarios tienen el permiso de adicionar activos de software, eliminarlos , así como modificar sus datos, ver sus detalles y descargarlos del sistema

Estimación de esfuerzos por historias de usuario

Una vez descritas las historias de usuario se realizó una estimación del esfuerzo que implicará la implementación de cada una de ellas. Los resultados de dicha estimación se muestran a continuación:

Tabla 2 Cantidad de semanas estimadas para implementación de Historias de Usuario.

Historia de Usuario	Punto de estimación (semanas)
Gestionar activos de software.	2
Buscar activo de software.	2
Consultar asociaciones y relaciones de dependencia de un activo de software.	1
Autenticar usuario.	1
Modificar contraseña de usuario.	1
Bloquear sesión por usuario.	1
Desbloquear sesión de usuario.	1
Cerrar sesión.	1
Consultar ayuda de usuario.	1
Gestionar roles.	1
Gestionar usuarios del sistema.	1
Gestionar roles de usuario.	1
Emitir reportes.	2

2.4 Especificación de requisitos

IEEE (Standard Glossary of Software Engineering Terminology) define los requisitos como:

- Condición o capacidad que necesita un usuario para resolver un problema o lograr un objetivo.

- Condición o capacidad que tiene que ser alcanzada o poseída por un sistema o componente de un sistema para satisfacer un contrato, estándar, u otro documento impuesto formalmente.
- Una representación en forma de documento de una condición o capacidad como las expresadas en 1 o en 2.

A partir de la aplicación de varias técnicas de especificación de requisitos como Tormenta de ideas y Entrevistas a expertos se obtuvieron los requisitos funcionales y no funcionales de la aplicación a desarrollar.

Requisitos funcionales del sistema

Los requisitos funcionales son capacidades o condiciones que el sistema debe cumplir sin alterar la funcionalidad del producto, esto quiere decir que se mantienen invariables sin importar con qué propiedades o cualidades se relacionan. Su principal tarea consiste en la generación de especificaciones correctas que describan con claridad, sin ambigüedades, en forma consistente y compacta, el comportamiento del sistema. [39]

Los requisitos funcionales del Sistema para el Control de Activos de Software se muestran a continuación agrupados por la Historia de Usuario a la que corresponden:

HU: Gestionar activo de software

RF1: Adicionar activo de software.

RF2: Eliminar activo de software.

RF3: Modificar datos de activo de software.

RF4: Descargar activo de software.

RF5: Ver detalles de activos de software.

HU: Buscar activo de software.

RF6: Buscar activo de software a través de requisitos no funcionales.

RF7: Buscar activo de software a través de palabras claves.

HU: Gestionar requisitos no funcionales

RF9: Adicionar requisito no funcional.

RF10: Modificar requisito no funcional.

HU: Autenticar usuario.

RF11: Autenticar usuario.

HU: Modificar contraseña de usuario.

RF12: Modificar contraseña de usuario.

HU: Cerrar sesión.

RF13: Cerrar sesión.

HU: Gestionar roles.

RF14: Crear rol.

RF15: Editar rol.

RF16: Eliminar rol.

RF17: Mostrar listado de roles.

HU: Gestionar usuarios del sistema.

RF18: Crear usuario.

RF19: Buscar usuario.

RF20: Editar usuario.

HU: Gestionar roles de usuario

RF21: Adicionar rol a usuario.

RF22: Eliminar rol a usuario.

HU: Emitir reportes

RF23: Reporte de búsqueda de activos.

RF24: Reporte de cantidad de activos áreas claves.

Requisitos no funcionales del sistema

Los requisitos no funcionales son propiedades o cualidades que el producto debe tener. Definiendo a las propiedades como características que hacen al producto atractivo, usable, rápido o confiable. [39]

Para el desarrollo del sistema Control de activos de software para CEGEL se identificaron 19 requisitos no funcionales (RNF) los cuales se encuentran descritos agrupados por categorías en Anexo 6.

Validación de los requisitos

La validación de los requisitos de software se realizó a partir de la aplicación de la técnica de **revisión de requisitos** y la métrica **especificidad de los requisitos,** descritas a continuación:

Revisión de requisitos: Constituye el mecanismo primario para la validación de los requisitos, consiste en la inspección o revisión de los documentos y las especificaciones buscando errores en el contenido o la interpretación, información faltante, inconsistencias, conflictos entre los requisitos o requisitos irreales. El equipo de revisión incluye ingenieros de software, clientes, usuarios y otros interesados. [42]

Métrica: Especificidad de los requisitos

Con el objetivo de medir la calidad de la especificación de los requisitos se aplicó la siguiente métrica. La especificidad de los requisitos se calcula mediante la fórmula:

$$Q_1 = \frac{n_{ui}}{n_r}$$

Donde:

Q_1: Especificidad de los requisitos.

nr: Total de requisitos definidos.

nui: Total de requisitos para los que los revisores tuvieron interpretaciones idénticas.

Cuanto más cerca de 1 esté el valor de **Q_1** menor será la ambigüedad de la especificidad de los requisitos.

Se obtiene como resultado:

Q_1= 24/24 = **1**

Como el valor de la Q_1 es 1, se puede inferir que la especificación de los requisitos no posee ambigüedad. Esta característica asegura mayor calidad en el proceso de especificación.

2.5 Plan de iteraciones

Esta fase incluye varias iteraciones sobre el sistema antes de ser entregado. El Plan de Entrega está compuesto por iteraciones de no más de tres semanas. En la primera iteración se puede intentar establecer una arquitectura del sistema que pueda ser utilizada durante el resto del proyecto. Esto se logra escogiendo las historias que fuercen la creación de esta arquitectura, sin embargo, esto no siempre es posible ya que es el cliente quien decide qué historias se implementarán en cada iteración (para maximizar el valor de negocio).

Al final de la última iteración el sistema estará listo para entrar en producción. Los elementos que deben tomarse en cuenta durante la elaboración del Plan de la Iteración son: historias de usuario no abordadas, velocidad del proyecto, pruebas de aceptación no superadas en la iteración anterior y tareas no terminadas en la iteración anterior. Todo el trabajo de la iteración es expresado en tareas de programación, cada una de ellas es asignada a un programador como responsable, pero llevadas a cabo por parejas de programadores. [26]

Iteración # 1: Para esta iteración las HU de mayor importancia se tuvieron en cuenta debido a la funcionalidad que describen cada una, es decir, aquellas HU con prioridad alta, entre las cuales se encuentran:

- HU Gestionar activos de software.
- HU Buscar activo de software.

Iteración # 2: En este caso las demás funcionalidades pasan a formar parte de esta iteración debido a que su prioridad es media, entre las que se encuentran:

- HU Autenticar usuario.
- HU Modificar contraseña de usuario.
- HU Bloquear sesión por usuario.
- HU Desbloquear sesión de usuario.
- HU Cerrar sesión.
- HU Gestionar roles.
- HU Gestionar usuarios del sistema.
- HU Gestionar roles de usuario.
- HU Emitir reportes.

Plan de duración de las iteraciones

Mediante este plan se muestran las HU que se desarrollaran en cada iteración, así como la duración estimada en semanas y el orden de implementación de cada una.

Tabla 3 Plan de duración de las iteraciones.

Nro. de iteración.	Historia de usuario a implementar.	Duración total de la iteración (semanas).
1	Gestionar activos de software. Buscar activo de software.	5
2	Autenticar usuario. Modificar contraseña de usuario. Bloquear sesión por usuario. Desbloquear sesión de usuario. Cerrar sesión. Gestionar roles. Gestionar usuarios del sistema. Gestionar roles de usuario. Emitir reportes.	11

Plan de entregas

En esta fase el cliente establece la prioridad de cada historia de usuario, y correspondientemente, los programadores realizan una estimación del esfuerzo necesario de cada una de ellas. Se toman acuerdos sobre el contenido de la primera

entrega y se determina un cronograma en conjunto con el cliente. Una entrega debería obtenerse en no más de tres meses. Esta fase dura unos pocos días.

La planificación se puede realizar basándose en el tiempo o el alcance. La velocidad del proyecto es utilizada para establecer cuántas historias se pueden implementar antes de una fecha determinada o cuánto tiempo tomará implementar un conjunto de historias. Al planificar por tiempo, se multiplica el número de iteraciones por la velocidad del proyecto, determinándose cuántos puntos se pueden completar. Al planificar según alcance del sistema, se divide la suma de puntos de las historias de usuario seleccionadas entre la velocidad del proyecto, obteniendo el número de iteraciones necesarias para su implementación. [40]

Tabla 4 Plan de entregas.

Nro. De iteración.	Historia de usuario a implementar.	Fecha de entrega.
1	Gestionar activos de software. Buscar activo de software. Consultar asociaciones y relaciones de dependencia de un activo de software.	15/03/2013
2	Autenticar usuario. Modificar contraseña de usuario. Bloquear sesión por usuario. Desbloquear sesión de usuario. Cerrar sesión. Consultar ayuda de usuario. Gestionar roles. Gestionar usuarios del sistema. Gestionar roles de usuario. Emitir reportes.	21/05/2013

2.6 Diseño

El diseño de XP sigue de manera rigurosa el principio MS (mantenerlo simple). Siempre se prefiere un diseño simple respecto de una presentación más compleja. Además, el diseño ofrece una guía de implementación para una historia como está escrita, ni más ni menos. Se desaprueba el diseño de funcionalidad extra (porque el desarrollador supone que se requerirá más tarde).

XP apoya el uso de tarjetas CRC como un mecanismo efectivo para pensar en el software en un contexto orientado a objetos. Las tarjetas CRC (colaborador-responsabilidad-clase) identifican y organizan las clases orientadas al objeto que son relevantes para el incremento del software actual. Las tarjetas CRC son el único producto de trabajo realizado como parte del proceso de XP.

Si se encuentra un problema difícil de diseño como parte del diseño de la historia, XP recomienda la creación inmediata de un prototipo operacional de esa porción del diseño. El prototipo del diseño, llamado la solución pico, se implementa y se evalúa. El propósito es reducir el riesgo cuando comience la verdadera implementación y validar las estimaciones originales en la historia que contiene el problema del diseño.

Debido a que el diseño de XP virtualmente no utiliza la notación y produce, si acaso, muy pocos productos de trabajo, distintos a las tarjetas de CRC y las soluciones pico, el diseño se considera como un artefacto que puede y debe modificarse de manera continua a medida que prosigue la construcción. El propósito de reconstruir es controlar estas modificaciones al sugerir pequeños cambios del diseño que pueden mejorar de manera radical el diseño. Sin embargo, debe notarse que el esfuerzo requerido para reconstruir puede aumentar en forma drástica a medida que crece el tamaño de la aplicación. [39]

Descripción de la arquitectura de software

Estilo arquitectónico Cliente – Servidor

La arquitectura Cliente/Servidor describe la relación de dos programas de computadoras donde un programa, el cliente, hace una llamada a otro programa, el servidor [42].

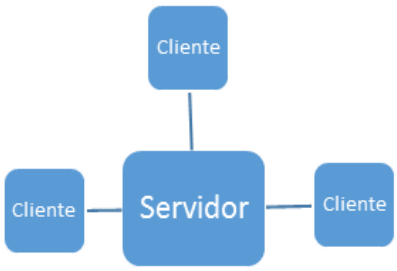

Figura 2 Arquitectura Cliente - Servidor.

El uso de la arquitectura Cliente/Servidor proporciona varias ventajas para el despliegue de aplicaciones como:

- Alta seguridad de los datos almacenados en un servidor, el cual debe ofrecer más control de la seguridad que los clientes.
- Acceso centralizado de los datos, como estos se encuentran centralizados en un solo servidor pueden ser accedidos y actualizados de mejor manera que en otros estilos arquitectónicos.
- Fácil mantenibilidad, este modelo se asegura de que se puedan mantener los sistemas sin afectar a los clientes.
- Delegación de la mayor carga de procesamiento posible hacia el servidor y mantener los clientes libres de gran parte de la lógica de negocio de las aplicaciones es una tendencia y buena práctica en estos entornos.

Otra de las ventajas es la posibilidad de escalar horizontal y verticalmente el servidor, que significa clusterizar el servidor y añadir mayor cantidad de recursos respectivamente. Esto se traduce en ofrecer mayor disponibilidad de servicios y de aplicaciones y obtener mejores índices de rendimiento.

Patrón arquitectónico Modelo-Vista-Controlador

El Modelo Vista Controlador (MVC) es un patrón de arquitectura de software que separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones.

Para ello MVC propone la construcción de tres componentes distintos que son el modelo, la vista y el controlador, es decir, por un lado define componentes para la representación de la información, y por otro lado para la interacción del usuario. Este patrón de diseño se basa en las ideas de reutilización de código y la separación de

conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento.

En el sistema el modelo es la representación específica de la información con la cual el sistema opera, por lo tanto gestiona todos los accesos a dicha información, tanto consultas como actualizaciones, el controlador responde a eventos (usualmente acciones del usuario) e invoca peticiones al 'modelo' cuando se hace alguna solicitud sobre la información (por ejemplo, editar un documento en la base de datos, la vista implementada en PHP o HTML a través de las plantillas Twig reside del lado del cliente en tiempo de ejecución.

Ventajas del MVC

- Es posible tener diferentes vistas para un mismo modelo (ej. representación de un conjunto de datos como una tabla).
- Es posible construir nuevas vistas sin necesidad de modificar el modelo subyacente.
- Proporciona un mecanismo de configuración a componentes complejos muchos más tratable que el puramente basado en eventos (el modelo puede verse como una representación estructurada del estado de la interacción). [47]

En la siguiente figura se muestra la distribución de las principales clases de cada una de las capas de la Vista, el Controlador y el Modelo en la solución de software. En la primera se destacan las clases que presentan datos al usuario como los formularios autenticarFRM, userFRM y gestionarActivosFRM. En la segunda se destacan las clases controladoras del negocio DefaultController, ldapController y paginaController. En el Modelo se muestran las clases que representan datos como Activo, Usuario y AccionUsuario.

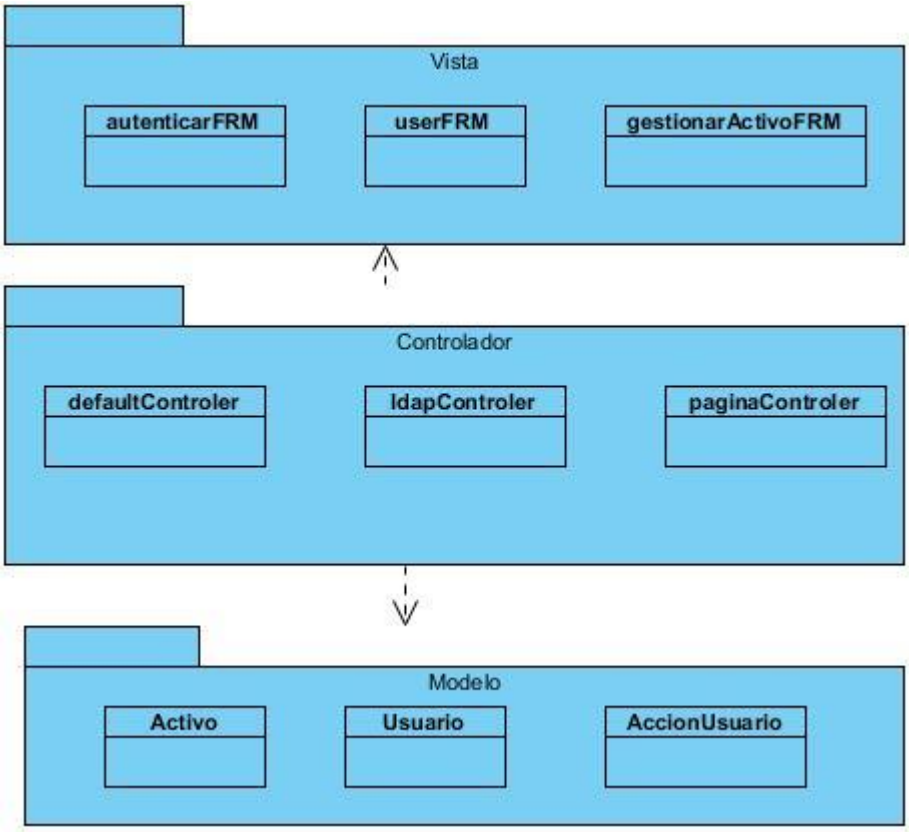

Figura 3 Patrón arquitectónico Modelo-Vista-Controlador.

Aplicación de patrones de diseño

Los patrones de diseño son un conjunto de estrategias, o buenas prácticas, que pueden facilitar el trabajo en muchas situaciones en el momento de realizar una aplicación orientada a objetos. A continuación se presentan los patrones utilizados en el desarrollo de la investigación y un fragmento de código donde se muestra la aplicación del mismo.

Controlador

El patrón controlador es un patrón que sirve de intermediario entre una interfaz específica y el algoritmo que la implementa, de tal manera que es quien recibe los datos del usuario y los envía a las distintas clases según el método llamado. Este patrón propone que la lógica de negocios debe estar separada de la capa de presentación, con el objetivo de aumentar la reutilización de código y a la vez tener un mayor control. En la siguiente figura se muestra un fragmento de código de la clase gestionarActivosController que contiene un conjunto de funcionalidades que se encargan de desarrollar la lógica de negocio para la gestión de activos de software.

```
305
306     public function adminAction() {...}
311
312     public function buscaAction() {...}
354
355 //          return $this->render('CasoBundle:Default:busca.html.twig',
356 //      }
357
358     public function gestionaractivosAction() {
359         $request = $this->getRequest();
360 //          $idSala = $this->recuperar("salaActual");
361         // if ($id == null) {
362         $entidad = new Activo();
363
364         $s = date('d/M/Y');
365         $date = date_create_from_format('d/M/Y', $s);
366         $date->getTimestamp();
367
368
369
370
371
372
373         $entidad->setFechaCreacion($date);
```

Figura 4 Fragmento de código de la clase controladora GestionarActivosController.

Bajo acoplamiento

Asigna una responsabilidad para mantener bajo acoplamiento. El acoplamiento es una medida de la fuerza con que una clase está conectada a otras clases, con que las conoce y con que recurre a ellas. Una clase con bajo (o débil) acoplamiento no depende de muchas otras. El bajo acoplamiento de las clases se muestra en los resultados de la aplicación de la métrica de diseño Relación entre clases que arroja que el 79% de las clases implementadas posee bajo o ningún acoplamiento.

Experto

Experto es un patrón que se usa más que cualquier otro en asignar responsabilidades; es un principio básico que suele utilizarse en el diseño orientado a objetos. Con él no se pretende designar una idea oscura ni extraña; expresa simplemente la "intuición" de que los objetos hacen cosas relacionadas con la información que poseen. Nótese que el cumplimiento de una responsabilidad requiere a menudo información distribuida en varias clases de objetos. Ello significa que hay muchos expertos "parciales" que colaboraron en la tarea. El patrón Experto da origen a diseño donde el objeto de software realiza las operaciones que normalmente se aplican a la cosa real que representa: a esto Peter Coad lo llama estrategia de "Hacerlo yo mismo". El patrón Experto, como tantas otras cosas en la tecnología de objeto, ofrece una analogía con el mundo real. Acostumbramos asignar responsabilidad a individuos que disponen de la información

necesaria para llevar a cabo una tarea. En la siguiente figura se muestra la declaración de la clase Usuario, ejemplo de este patrón puesto que es la encargada de realizar todas las funcionalidades concernientes a la información del usuario.

```
13  class Usuario
14  {
15      /**
16       * @var integer $idUsuario
17       *
18       * @ORM\Column(name="id_usuario", type="bigint", nullable=false)
19       * @ORM\Id
20       * @ORM\GeneratedValue(strategy="SEQUENCE")
21       * @ORM\SequenceGenerator(sequenceName="usuario_id_usuario_seq", allocat
22       */
23      private $idUsuario;
24
25      /**
26       * @var string $contrasenna
27       *
28       * @ORM\Column(name="contrasenna", type="string", nullable=true)
29       */
30      private $contrasenna;
31
32      /**
33       * @var string $nombreusuario
34       *
35       * @ORM\Column(name="nombreusuario", type="string", nullable=true)
36       */
37      private $nombreusuario;
38
39      /**
40       * @var InformacionDeRol
41       *
```

Figura 5 Declaración de la clase Usuario, experta en las funcionalidades relacionadas a la información de un usuario.

Alta cohesión

Como el patrón Bajo Acoplamiento, también Alta Cohesión es un principio que se debe tener presente en todas las decisiones de diseño: es la meta principal que ha de buscarse en todo momento. Es un patrón evaluativo que el desarrollador aplica al valorar sus decisiones de diseño.

Una clase de alta cohesión colabora con otros objetos para compartir el esfuerzo si la tarea es grande. Es útil porque es bastante fácil proporcionarle mantenimiento, entenderla y reutilizarla. Su alto grado de funcionalidad, combinada con una reducida cantidad de operaciones, también simplifica el mantenimiento y los mejoramientos. La ventaja que significa una gran funcionalidad también soporta un aumento de la capacidad de reutilización.

Este patrón se evidencia en las clases controladoras que agrupan todas las funcionalidades de la lógica de negocio y colaboran entre sí para satisfacer los requisitos de software. Un ejemplo de ellos son las clases gestionarActivosController, paginaController y defaultController.

Tarjetas CRC

El uso de las tarjetas CRC (Class, Responsibilities and Collaboration) permiten al programador centrarse y apreciar el desarrollo orientado a objetos olvidándose de los malos hábitos de la programación procedural clásica. Las tarjetas CRC representan objetos; la clase a la que pertenece el objeto se puede escribir en la parte de arriba de la tarjeta, en una columna a la izquierda se pueden escribir las responsabilidades u objetivos que debe cumplir el objeto y a la derecha, las clases que colaboran con cada responsabilidad.

Las tarjetas CRC son tarjetas de índices, una por cada clase, sobre las cuales se abrevian las responsabilidades de la clase y se anota una lista de los objetos con los que colaboran para desempeñarlas. Suelen elaborarse en una sesión de grupos pequeños donde los participantes representan papeles de las diversas clases. Cada grupo tiene las tarjetas CRC correspondientes a las clases cuyo papel este desempeñando. Las tarjetas son un método de registrar los resultados de la asignación de responsabilidades y de las colaboraciones. El registro es más satisfactorio si se utiliza este tipo de diagramas. El valor real no está constituido por las tarjetas, sino por el análisis de la asignación de responsabilidades y por la actividad relacionada con la representación de papeles.

A partir del análisis de los requisitos funcionales y no funcionales del sistema, se realizó una estimación de los componentes de software para la elaboración del diseño. De las Tarjetas CRC de mayor impacto que se construyeron para darle solución a la problemática se encuentran:

Tabla 5: Tarjeta CRC: DefaultController.

Tarjeta CRC	
Clase: *DefaultController*	
Responsabilidades	**Colaboraciones**
-Gestionar activos de software -Gestionar usuario. -Autenticar usuario. -Buscar activo por palabra clave -Buscar activo por tipo de activo -Gestionar rol de usuario	-Activos -Usuario -Ldap -Activo -Activo -Usuario

Tabla 6: Tarjeta CRC: Ldap.

Tarjeta CRC	
Clase: Repositorio	
Responsabilidades	**Colaboraciones**
Iniciar Ldap Conexión Cerrar conexión Loguear usuario	

Diseño de la Base de Datos

El modelo de datos es una abstracción que permite visualizar las entidades de la base de datos y sus relaciones. El modelo de datos asociado a las entidades persistentes del sistema se divide en dos partes: a) modelo lógico que representa las relaciones entre las entidades conceptuales del modelo; y b) modelo físico, el cual responde a las tablas de la base de datos del sistema, desplegada en el servidor.

A continuación se presenta una representación del diagrama entidad relación donde se observan las entidades y sus relaciones persistentes en el modelo de datos.

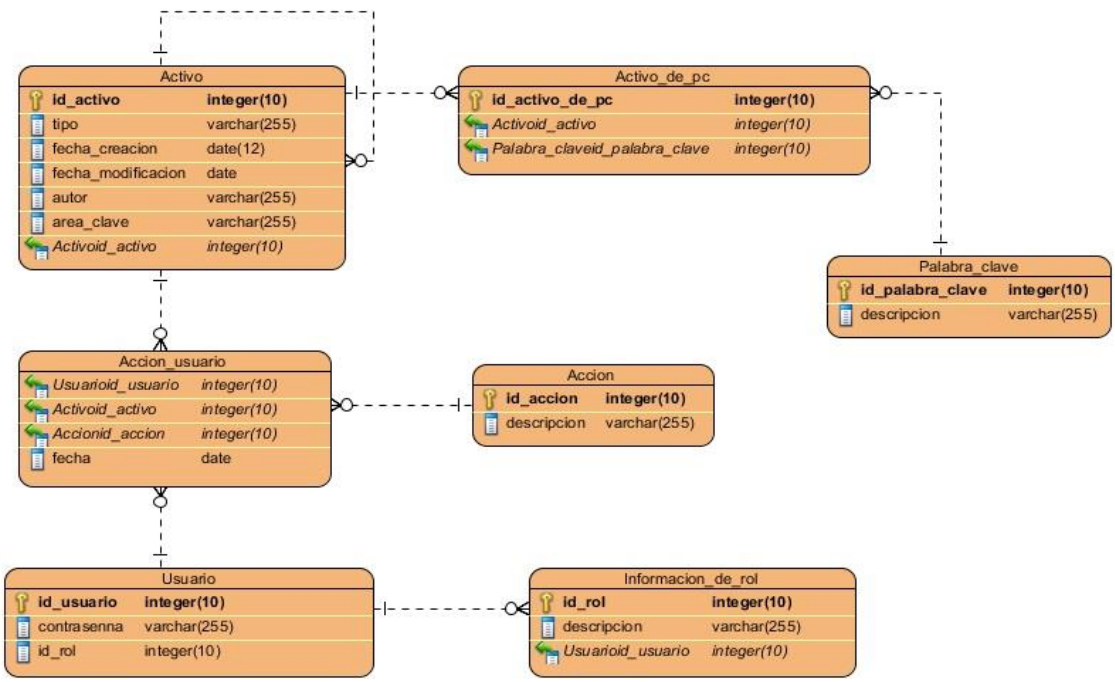

Figura 6 Diagrama Entidad - Relación para el Sistema de Control de Activos de Software.

Las entidades más importantes del modelo de datos:

Activo: representa todos activos de software registrados en el sistema.

Activo_de_pc: surge de la relación m-m entre la tabla Activo y Palabra clave.
Palabra clave: representa las clasificaciones de los activos de software por las cuales estos pueden ser localizados.
Accion_usuario: surge de la relación entre las tablas Usuario y Acción.
Acción: representa las acciones que se realiza el usuario en el sistema.

Modelo de Despliegue

Los diagramas de despliegue son utilizados para hacer una representación de nodos, conexiones, objetos y componentes [46]. Este diagrama se utiliza para modelar el hardware utilizado en la implementación del sistema y las relaciones entre sus componentes. El servidor de aplicaciones web Apache es el que despliega todos los componentes del sistema, los que interactúan con la bases de datos PostgreSQL usando el protocolo TCP/IP. Los clientes realizan las peticiones al servidor a través del protocolo HTTP. La estación de trabajo cliente puede estar conectada a una impresora para realizar operaciones de impresión de informes. A continuación se presenta el modelo de despliegue de la solución:

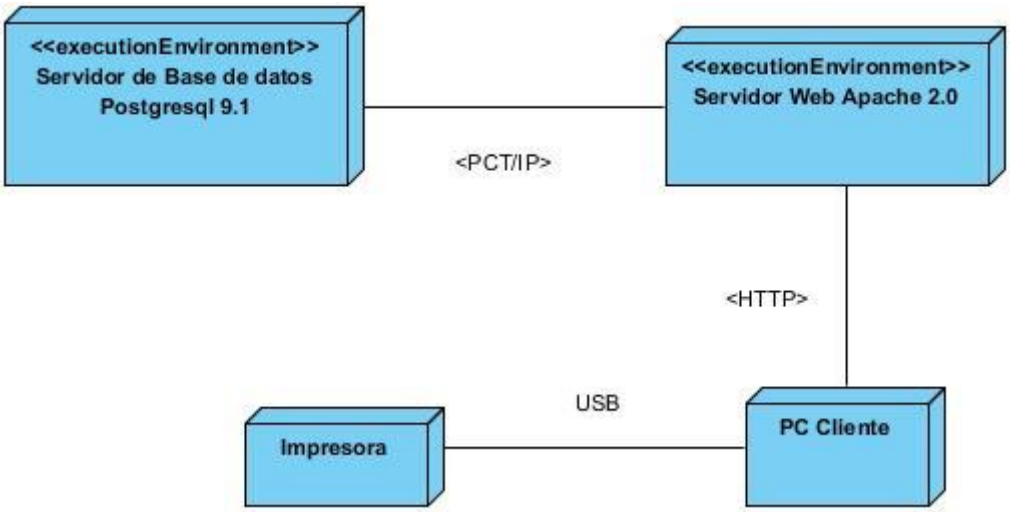

Figura 7 Diagrama de Despliegue del Sistema de Control de Activos de Software.

Validación del diseño

Para evaluar el diseño de la solución, se decide aplicar las métricas **Tamaño Operacional de Clases (TOC)** y **Relación entre Clases (RC)**.

Métrica Tamaño operacional de clase (TOC)

- **TOC:** Está dado por el número de métodos asignados a una clase.
- **Responsabilidad:** Un aumento del TOC implica un aumento de la responsabilidad asignada a la clase.

- **Complejidad de implementación:** Un aumento del TOC implica un aumento de la complejidad de implementación de la clase.
- **Reutilización:** Un aumento del TOC implica una disminución del grado de reutilización de la clase.

La métrica fue aplicada a una muestra seleccionada de 14 clases, donde se obtuvo que del total de clases que se analizaron en cuanto a responsabilidad existe un 15% alto, un 8% de medio y un 77% bajo. Atendiendo a la complejidad se obtuvo que existe un 15% alto, un 8% medio y un 77% bajo. Por último en cuanto a la reutilización los valores fueron de: 13% bajo, 25% medio y un 62% alto.

Haciendo un análisis de los resultados obtenidos en la evaluación del instrumento de medición de la métrica TOC, se puede concluir que el diseño tiene una buena calidad teniendo en cuenta que el 62% de las clases presentan una alta reutilización. Además el 77%de las clases posees avaluaciones positivas es los atributos de complejidad de implementación y reutilización, lo cual garantiza una solidez en el diseño del sistema puesto que habrá gran reutilización de las clases y su implementación no resulta complicada, ver Anexo 4.

Métrica 2: Relaciones entre clases (RC)

- **RC:** Esta dado por el número de relaciones de uso de una clase con otra.
- **Acoplamiento:** Un aumento del RC implica un aumento del Acoplamiento de la clase.
- **Complejidad de mantenimiento:** Un aumento del RC implica un aumento de la complejidad del mantenimiento de la clase.
- **Reutilización:** Un aumento del RC implica una disminución en el grado de reutilización de la clase.
- **Cantidad de pruebas:** Un aumento del RC implica un aumento de la cantidad de pruebas de unidad necesarias para probar una clase.

El resultado de la aplicación de la métrica RC está dado por el número de relaciones de uso de una clase con otras. La aplicación de dicha métrica permite evaluar atributos como el Acoplamiento, la Complejidad de mantenimiento, la Reutilización y la Cantidad de pruebas. De forma tal que mientras mayor sea las relaciones de uso entre clases mayor será el Acoplamiento, la Complejidad de Mantenimiento y la Cantidad de pruebas, mientras que su Reutilización disminuye.

Haciendo un análisis de los resultados obtenidos en la evaluación del instrumento de medición de la métrica RC, se puede concluir que el diseño diferentes sub-sistemas tiene una buena calidad teniendo en cuenta que el 80% de las clases incluidas en este

subsistema posee 3 o menos dependencias de otras clases. Esto favorece el hecho que al ocurrir un cambio de alguna de las clases, la afectación en las restantes sea de poca importancia.

El 79% de las clases posee índices de bajo o ningún acoplamiento, este es un factor es muy importante porque posibilita que se realicen modificaciones en el sistema teniendo poco impacto en el mismo y que entre las clases exista un bajo acoplamiento. Por su parte los atributos de calidad complejidad de mantenimiento, cantidad de pruebas y reutilización se comportan satisfactoriamente en un 43% de las clases, teniendo 3 o menos dependencias de otras clases, fomentando el rehúso de las mismas y facilitando el mantenimiento del sistema así como las pruebas necesarias para la comprobación del software, ver Anexo 5.

Estas métricas brindan la posibilidad medir de forma cuantitativa la calidad de los atributos internos del software, permitiendo al ingeniero evaluar la calidad durante el desarrollo del sistema.

Conclusiones parciales

En este capítulo se abordaron los elementos fundamentales correspondientes a las fases de: planificación y diseño del sistema informático que se propone, apoyándose en las técnicas relacionadas por la metodología XP para estas fases de desarrollo.

Fueron identificadas 24 funcionalidades que el sistema debe satisfacer a partir de 10 historias de usuario mientras que los requisitos no funcionales cubren clasificaciones como usabilidad, rendimiento, soporte, seguridad, software y hardware.

Con un 40% de historias de usuario de prioridad alta y un 60% de prioridad media se estima el desarrollo de la aplicación en 4 meses ideales de trabajo; dispuestos en 2 iteraciones con una duración de 5 y 11 semanas respectivamente.

La métrica para evaluar la estabilidad de los requisitos arrojó que no se han realizado, en estos, cambios significativos; por tal motivo son estables y es posible sustentar el diseño de la aplicación. Por su lado la evaluación del grado de validación de los requisitos resultó el valor máximo acorde con datos obtenidos.

La arquitectura definida responde al patrón arquitectónico Modelo-Vista-Controlador y al estilo arquitectónico Cliente-Servidor en tanto en el diseño se identifican patrones Generales de Asignación y Responsabilidad GRASP como el Controlador, Experto, Alta cohesión y Bajo acoplamiento.

En la verificación del diseño se analizaron las clases de diseño que juegan un papel fundamental de los procesos principales del sistema, a las que se les aplicó las métricas

Tamaño Operacional de Clases TOC y Relaciones entre clases obteniendo resultados satisfactorios en este sentido.

CAPÍTULO 3: IMPLEMENTACIÓN Y PRUEBA

Introducción

En el presente capítulo se describen las tareas de la ingeniería pertenecientes a las historias de usuarios de prioridad alta definidas en el capítulo anterior. Además se muestra el desarrollo de la implementación del componente durante las diferentes iteraciones definidas por la metodología de desarrollo XP. Por último para validar la solución propuesta se realizarán las pruebas correspondientes a todas las funciones implementadas durante el desarrollo de la aplicación.

3.1 Fase de implementación

La metodología XP plantea que la implementación de un software debe realizarse de forma iterativa, obteniendo al finalizar cada iteración un producto funcional que debe ser probado y mostrado al cliente. En esta fase se realiza la implementación de cada una de los HU durante el transcurso de las iteraciones, creándose las tareas de la ingeniería para ayudar a organizar la implementación exitosa de la HU, estas tareas son asignadas a los programadores que son los responsables de la implementación. Estas tareas pueden escribirse en un lenguaje técnico, al contrario de las HU que son escritas en el lenguaje del cliente. [42]

Iteraciones

Teniendo en cuenta la planificación realizada anteriormente, quedaron definidas 2 iteraciones de desarrollo sobre el sistema, con el objetivo de obtener un producto con todas las características deseadas para ser utilizado. Detallándose cada una de las iteraciones a continuación.

Desarrollo de la primera iteración

En la primera iteración se tuvieron en cuenta aquellas HU de mayor importancia en cuanto a la funcionalidad que describen cada una, es decir, aquellas HU con prioridad alta.

Tabla 7 Descripción de las Tareas de Programación por cada HU seleccionadas en la primera iteración.

Nro. de iteración.	Historia de usuario a implementar.	Duración total de la iteración (semanas).	Tareas de programación.
	Gestionar activos de software.		Desarrollar la interfaz: Gestionar

1		5	activos de software.
	Buscar activo de software.		Desarrollar la interfaz: Buscar activos de software.

HU Gestionar activos de software

Tabla 8 Tarea # 1: Desarrollar la interfaz: Gestionar activos de software.

Tarea de programación	
Número de tarea: 1	Número de historia: 1
Nombre de tarea: Desarrollar la interfaz: Gestionar activos de software.	
Tipo de tarea: Desarrollo.	Puntos estimados: 2 semanas.
Fecha de inicio:1/03/2013	Fecha de fin:15/03/2013
Programador responsable: Víctor Hernández Martiartu	
Descripción: Los usuarios tienen el permiso de adicionar activos de software, eliminarlos, así como modificar sus datos, ver sus detalles y descargarlos del sistema.	

Tabla 9 Tarea # 2: Desarrollar la interfaz: Buscar activo de software.

Tarea de programación	
Número de tarea: 2	Número de historia: 2
Nombre de tarea: Desarrollar la interfaz: Buscar activo de software.	
Tipo de tarea: Desarrollo.	Puntos estimados: 2 semanas.
Fecha de inicio:1/03/2013	Fecha de fin:15/03/2013

Programador responsable: Víctor Hernández Martiartu
Descripción: El usuario busca un activo de software mediante la descripción de requisitos y palabras claves.

Desarrollo de la segunda iteración

En la segunda iteración se tuvieron en cuenta aquellas funcionalidades del sistema con menos prioridad, es decir, aquellas HU que presentan prioridad media o baja.

Tabla 10 Descripción de las Tareas de Programación por cada HU seleccionadas en la segunda iteración.

Nro. de iteración.	Historia de usuario a implementar.	Duración total de la iteración (semanas).	Tareas de programación.
2	Autenticar usuario.	11	Desarrollar la interfaz: Autenticar usuario.
	Modificar contraseña de usuario.		Desarrollar la interfaz: Modificar contraseña de usuario.
	Cerrar sesión.		Desarrollar la interfaz: Cerrar sesión.
	Gestionar roles.		Desarrollar la interfaz: Gestionar roles.
	Gestionar usuarios del sistema.		Desarrollar la interfaz: Gestionar usuarios del sistema.

	Gestionar roles de usuario.		Desarrollar la interfaz: Gestionar roles de usuario.
	Emitir reportes.		Desarrollar la interfaz: Emitir reportes.

Tareas de programación

Las tareas de programación se realizan para detallar mejor las HU, facilitando el entendimiento en el proceso de implementación. Cada HU puede contener una o más tareas de programación en caso de necesitarla en dependencia de la complejidad de la funcionalidad a desarrollar.

Tabla 11 Tarea # 4: Desarrollar la interfaz: Autenticar usuario.

Tarea de programación	
Número de tarea: 4	Número de historia: 4
Nombre de tarea: Desarrollar la interfaz: Autenticar usuario.	
Tipo de tarea: Desarrollo.	Puntos estimados: 1 semanas.
Fecha de inicio:17/03/2013	Fecha de fin:24/03/2013
Programador responsable: Yuriel Guerra Rodríguez.	
Descripción: El usuario busca un activo de software mediante la descripción de requisitos y palabras claves.	

Tabla 12 Tarea # 5: Desarrollar la interfaz: Modificar contraseña de usuario.

Tarea de programación	
Número de tarea: 5	Número de historia: 5
Nombre de tarea: Desarrollar la interfaz: Modificar contraseña de usuario.	

Tipo de tarea: Desarrollo.	Puntos estimados: 1 semanas.
Fecha de inicio:26/03/2013	Fecha de fin:2/04/2013
Programador responsable: Yuriel Guerra Rodríguez.	
Descripción: Los usuarios podrán cambiar su contraseña para el acceso al sistema según su preferencia.	

Tabla 13 Tarea # 8: Desarrollar la interfaz: Cerrar sesión.

Tarea de programación	
Número de tarea: 8	Número de historia: 8
Nombre de tarea: Desarrollar la interfaz: Cerrar sesión.	
Tipo de tarea: Desarrollo.	Puntos estimados: 1 semanas.
Fecha de inicio:9/04/2013	Fecha de fin:16/04/2013
Programador responsable: Víctor Hernández Martiartu	
Descripción: El usuario podrá salir de la aplicación y cerrar su sección.	

Tabla 14 Tarea # 10: Desarrollar la interfaz: Gestionar roles.

Tarea de programación	
Número de tarea: 10	Número de historia: 10
Nombre de tarea: Desarrollar la interfaz: Gestionar roles.	
Tipo de tarea: Desarrollo.	Puntos estimados: 1 semanas.
Fecha de inicio:20/04/2013	Fecha de fin:27/04/2013

Programador responsable: Víctor Hernández Martiartu
Descripción: El administrador creara roles, definidos por el trabajo que se realizara en el sistema, así como editarlo según las necesidades que este necesite, eliminarlo si no es necesario su labor y mostrara un listado de los roles existentes en el sistema.

Tabla 15 Tarea # 11: Desarrollar la interfaz: Gestionar usuarios del sistema.

Tarea de programación	
Número de tarea: 11	Número de historia: 11
Nombre de tarea: Desarrollar la interfaz: Gestionar usuarios del sistema.	
Tipo de tarea: Desarrollo.	Puntos estimados: 1 semanas.
Fecha de inicio:22/04/2013	Fecha de fin:29/04/2013
Programador responsable: Víctor Hernández Martiartu	
Descripción: El administrador podrá adicionar, eliminar, modificar y buscar usuarios en la base de datos.	

Tabla 16 Tarea # 12: Desarrollar la interfaz: Gestionar roles de usuario.

Tarea de programación	
Número de tarea: 12	Número de historia: 12
Nombre de tarea: Desarrollar la interfaz: Gestionar roles de usuario.	
Tipo de tarea: Desarrollo.	Puntos estimados: 1 semanas.
Fecha de inicio:28/04/2013	Fecha de fin:5/05/2013
Programador responsable: Víctor Hernández Martiartu	

Descripción: Permitirá al administrador otorgarle un rol al usuario o eliminarlo dependiendo del trabajo que realizara en el sistema.

Tabla 17 Tarea # 13: Desarrollar la interfaz: Emitir reportes.

Tarea de programación	
Número de tarea: 13	Número de historia: 13
Nombre de tarea: Desarrollar la interfaz: Emitir reportes.	
Tipo de tarea: Desarrollo.	Puntos estimados: 2 semanas.
Fecha de inicio:6/05/2013	Fecha de fin:20/05/2013
Programador responsable: Víctor Hernández Martiartu	
Descripción: Los usuarios emitirán reportes de las búsquedas de activos de software realizadas, así como el uso de activos de software pertenecientes al sistema y las áreas claves donde influyen estos activos.	

3.2 Fase Prueba

Uno de los principios fundamentales de XP es el proceso de prueba, donde los desarrolladores realizan pruebas constantemente tanto como sea posible; de esta manera se reduce el número de errores no detectados, así como el tiempo entre la introducción de este en el sistema y su detección. Todo esto contribuye a elevar la calidad del producto desarrollado.

XP divide las pruebas en dos grupos:

- **Pruebas unitarias**, desarrolladas por los programadores, encargadas de verificar el código de forma automática.
- **Pruebas de aceptación**, las cuales son destinadas a evaluar bloques más grandes de la funcionalidad del sistema, tales como las historias de usuarios, verificando que al final de una iteración se obtuvo la funcionalidad requerida, además de que dicha funcionalidad sea la esperada por el cliente.

Pruebas Unitarias

Las pruebas unitarias permiten probar pequeñas e individuales porciones de código. A través de ellas se verifica qué módulo o funcionalidad se ejecuta dentro de los parámetros y especificaciones concretadas en documentos tales como los: casos de uso y el diseño detallado, proporcionan un contrato escrito que la porción de código debe cumplir. Permiten detectar efectivamente la inyección de defectos durante fases sucesivas de desarrollo o mantenimiento.

Las pruebas unitarias son una forma de probar el correcto funcionamiento de un componente. Luego, con las Pruebas de Integración, se podrá asegurar el correcto funcionamiento del sistema o subsistema en cuestión. El objetivo de las pruebas unitarias es aislar cada parte del programa y mostrar que las partes individuales son correctas.

Pruebas de caja blanca

Las pruebas de caja blanca realizan un seguimiento del código fuente según va ejecutando los casos de prueba, de manera que se determinan de manera concreta las instrucciones, bloques, etc., en los que existen errores. Estas pruebas se realizan sobre las funciones internas de un sistema en concreto. Entre las técnicas usadas se encuentran: la cobertura de caminos, pruebas sobre las expresiones lógico-aritméticas, pruebas de camino de datos, comprobación de bucles.

Aunque existen numerosas variantes de este tipo de pruebas, se considera factible desarrollar la prueba del camino básico porque permite obtener una medida de la complejidad lógica de un diseño y usar esta medida como guía para la definición de un conjunto básico. La idea es derivar casos de pruebas a partir de un conjunto dado de caminos independientes por los cuales puede circular el flujo de control. Para obtener el conjunto de caminos independientes se construye el Grafo de Flujo asociado y se calcula su complejidad ciclomática. Es además una de las más eficientes en cuanto a cobertura de código, pues logra que se ejecuten todos los bucles en sus límites operacionales [44].

A continuación se muestra un ejemplo descriptivo de la aplicación de esta prueba a la funcionalidad autenticar usuario. Se procede a realizar los siguientes pasos:

1. Usando el diseño o el código como base, mostrado en la siguiente figura, se dibuja el correspondiente grafo de flujo.
2. Se determina la complejidad ciclomática del grafo de flujo resultante, V (G).
3. Se determina un conjunto básico de hasta V (G) caminos linealmente independientes.

4. Se preparan los casos de prueba que forzarán la ejecución de cada camino del conjunto básico.

.

```
public function autenticarAction() {

    $request = $this->getRequest();//1

    $entidad = new Usuario();//1

    $form = $this->createForm(new AutenticarType(), $entidad);//1

    if ('POST' == $request->getMethod()) {//2
        $form->bindRequest($request);//3
        if ($form->isValid()) {//4

            $em = $this->getDoctrine()->getEntityManager();//5

            $em->persist($entidad);//5
            $em->flush();//5

            return $this->render('CasoBundle:Default:index.html.twig', array(null));//5
        } //6
        else {//7
            var_dump($form->getErrorsAsString());//8
            exit;//8
        }//9

return $this->render('CasoBundle:Default:autenticar.html.twig', array("form" => $form->createView()));//10
    }//11
 return $this->render('CasoBundle:Default:autenticar.html.twig', array("form" => $form->createView()));//12
  }  //13
```

Figura 8 Fragmento de código que representa el método autenticarAction ().

En la siguiente figura se muestra el grafo del flujo asociado al código anterior:

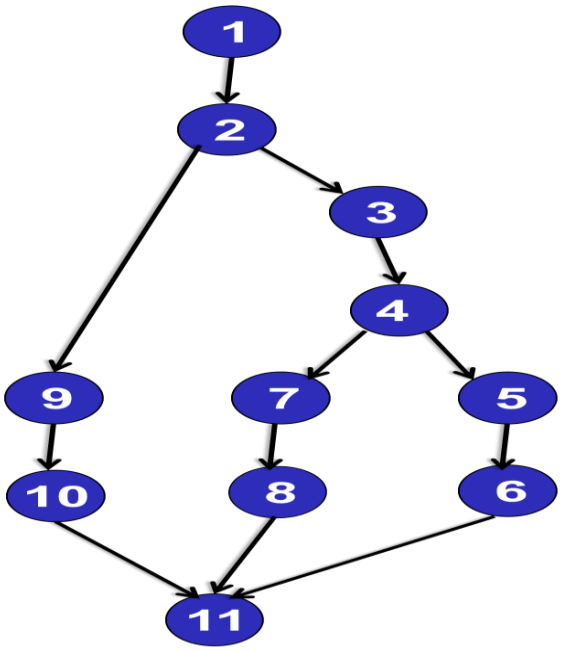

Figura 9 Grafo del código de Caso de Prueba.

La complejidad ciclomática consta de tres formas para calcularse:

- V (G) = A - N + 2 donde A: nº de arcos del grafo N: nº de nodos
- V (G) = R donde R: nº de regiones del grafo
- V (G) = P +1 donde P: nº de nodos predicado

Cálculo de la complejidad ciclomática variante 1:

V (G) = (A– N) + 2

V (G) = (12 – 11) + 2

V (G) = 3

Cálculo de la complejidad ciclomática variante 2:

V (G) = P + 1

V (G) = 2 + 1

V (G) = 10

Cálculo de la complejidad ciclomática variante 3:

V (G) = R

V (G) = 3

El cálculo efectuado mediante las tres fórmulas ha dado el mismo valor, por lo que se puede plantear que la complejidad ciclomática del código es 3, lo que significa que existen diez posibles caminos por donde el flujo puede circular, este valor representa el límite mínimo del número total de casos de pruebas para el procedimiento tratado.

Seguidamente es necesario representar los caminos básicos por los que puede recorrer el flujo.

Tabla 18 Caminos básicos por los que puede recorrer el flujo de datos.

Número	Camino básico
1	1 – 2 – 9 – 10 – 11
2	1 – 2 – 3 – 4 – 7 – 8 – 11
3	1 – 2 – 3 – 4 – 5 – 6

A cada camino básico se le realiza un caso de prueba el cual va a consistir en hacer una descripción del método y dar la condición de ejecución, luego se le pasan variables y se esperan los resultados.

Pruebas de aceptación

Las pruebas de aceptación son pruebas de caja negra que se crean a partir de las HU. Durante las iteraciones las HU seleccionadas serán traducidas a pruebas de aceptación, en ellas se especifican desde la perspectiva del cliente, los escenarios para probar que una HU ha sido implementada correctamente. Una HU puede tener todas las pruebas de aceptación que necesite para asegurar su correcto funcionamiento. El objetivo final de las mismas es garantizar que los requisitos sean cumplidos y que el sistema sea aceptado por el cliente. Una HU no se considera completa hasta que no ha pasado por sus pruebas de aceptación [43] [45].

Se elaboraron 10 casos de pruebas descritos en el Anexo 7 y fueron aplicadas a las funcionalidades del sistema en tres iteraciones. La cantidad de No conformidades detectadas se explica en la siguiente gráfica:

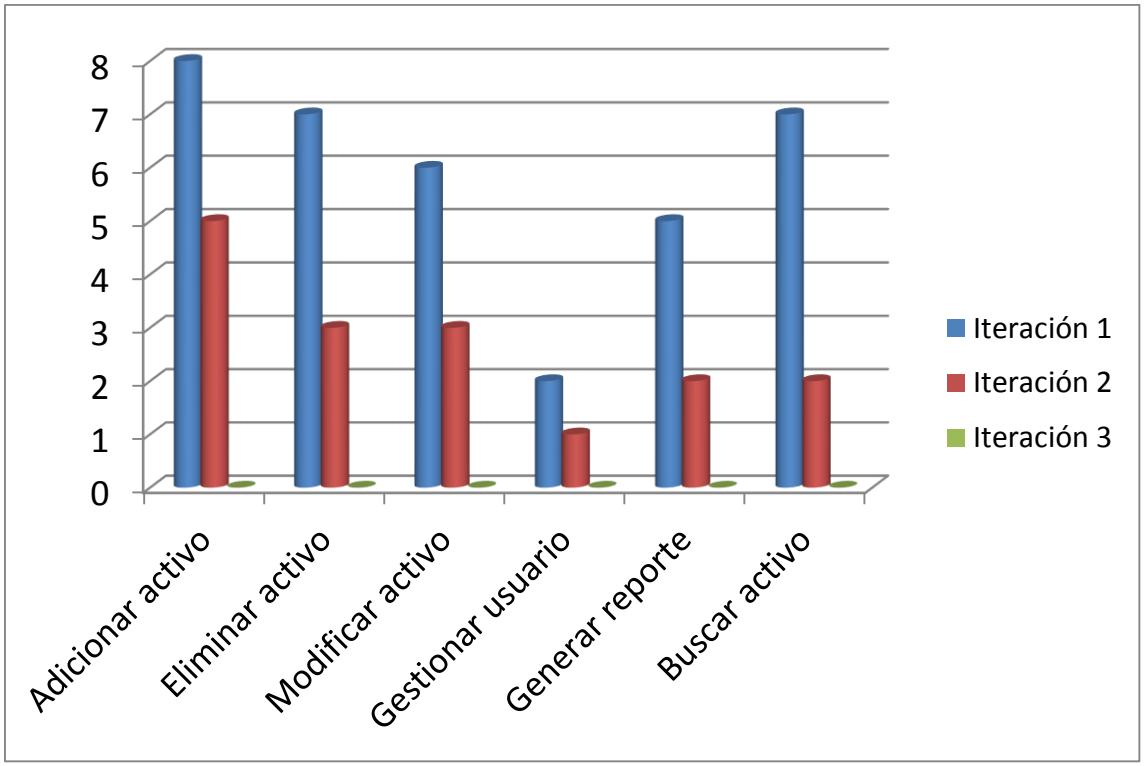

Figura 10 Relación de No conformidades detectadas.

Al finalizar la tercera iteración se habían resuelto todas las No conformidades detectadas. Las No conformidades señaladas durante el período de pruebas se muestran en el Anexo 8.

Conclusiones parciales

La realización de esta etapa permitió obtener la implementación del Sistema para el Control de Activos de Software a partir de la ejecución de las tareas de ingeniería. La aplicación de pruebas unitarias y de aceptación permitió verificar que la aplicación obtenida satisface los requisitos especificados inicialmente validados por el reconocimiento del Asesor de Tecnologías del Centro, mostrado en el Anexo **¡Error! No se encuentra el origen de la referencia.**9 sobre la calidad del sistema y su influencia en el control de activos de software en función de la reutilización en el proceso de desarrollo.

CONCLUSIONES

Al concluir el presente trabajo se puede afirmar que:

1. El análisis de los elementos que componen el objeto de estudio, el estudio de sistemas similares a la solución propuesta y la adecuada selección de las herramientas y metodología de desarrollo permitieron determinar que la solución pretenda clasificarse como un repositorio para el almacenamiento de activos de software y su catalogación, en pos de favorecer la reutilización de estos en el proceso de desarrollo de software de CEGEL.
2. El sistema logró satisfacer 24 funcionalidades comprendidas en 10 historias de usuario, de ellas un 40% con prioridad Alta y un 60% con prioridad media para su implementación; a partir de un diseño que responde a las características del patrón MVC y al estilo arquitectónico Cliente-Servidor y que incorpora soluciones aportadas por patrones de diseño previamente identificados.
3. La verificación de la calidad de los artefactos y productos de software generados, a partir de la aplicación de métricas estandarizadas y pruebas de software, permitió la validación de los procesos de Especificación de Requisitos, Diseño e Implementación de la solución respectivamente.
4. La solución informática logró satisfacer los requisitos acordados para el registro y control de activos de software en función de contribuir a la reutilización del Centro.

RECOMENDACIONES

Para mejorar el desempeño de la solución alcanzada se realizan las siguientes recomendaciones:

- Permitir, a través del sistema, la realización de reportes de información sobre asociación de activos de software.
- Permitir la construcción de mapas de asociación a través de las relaciones entre los activos de software.

REFERENCIAS BIBLIOGRÁFICAS

1. Villanueva Orta, Leidy. 2012. Análisis y diseño de un Sistema para la Administración de un Repositorio de Activos de Software aplicable al proceso de desarrollo del CEIGE. Tesis para optar por el título de Ingeniero en Ciencias Informáticas. Universidad de las Ciencias Informáticas. Ciudad de La Habana.
2. González López Pascual, Moreno Valverde Ginés, González López Ana Amelia. La Reutilización: Un camino hacia la industrialización del desarrollo de software.2011
3. Vladimir Rosales Echarri. La metodología de la Investigación Educativa para la Formación del Profesional. La Habana, 2004.
4. Bastarrica, María Cecilia. Desarrollo de Líneas de Productos de Software. Ciencias de la Computación, Universidad de Chile. Chile: s.n., 2000.
5. Montilva, Jonás A. Desarrollo de Software Basado en Líneas de Productos de Software. 2006.
6. García Peñalvo, Francisco José, y otros. Mecanos: Soporte de Diferentes Niveles de Abstracción en los elementos Software Reutilizables. España: s.n., 1998.
7. Ingeniería de software basado en componentes. http://isoftwareunesum.wordpress.com/2011/04/28/ingenieria-del-software-basado-en-componentes/. Abril, 2011.
8. Vanier, D y Rahman, S. MIIP Report: A Primer on Municipal Infrastructure Asset Management. 2004. pág. 65.
9. Alexander Álvaro Barón Salazar 2012. Pegaso, una propuesta para la gestión activos de software: universidad EAFIT facultad de ingeniería, maestría en ingeniería, Medellín, 2012.
10. IEEE COMPUTER SOCIETY. 2010. IEEE Standard for Information Technology – System and Software Life Cycle Processes – Reuse Processes, IEEE STD 1517™. New York, USA: Society IEEE Computer, 2010.
11. SOMMERVILLE, Ian. 2007. Software Engineering. 8th. United Kingdom: Pearsons Education, 2007.
12. JHA, Meena, O´BRIEN, Liam y MAHESHWARI, Piyush. 2006. Identify Issues and Concerns in Software Reuse. 2006.
13. FRAKES, William y KANG, Kyo. 2005. Software Reuse Research: Status and Future. s.l.: IEEE transactions on software engineering, 2005. Vol. 31.
14. WHITEHEAD, Derek. 2005. Repositories: What is the Target? An Arrow Perspective', New Review of Information Networking, 2005. Págs. 123-134.

15. WINDOWSCLIENT.NET COMUNITY. 2011. Microsoft WindowsClient.NET-windows forms-Windows Presentation Foundation. Citado el: 10 de marzo de 2011. Disponible en: http://www.windowsclient.net.
16. DATA & OBJECT FACTORY. 2001. Dofactory#1 in design patterns. DoFactory. Citado el: 14 de septiembre de 2011. Disponible en: http://www.dofactory.com.
17. INFRAGISTICS, INC. QUINCE. 2009. Quince. quince. Citado el: 14 de septiembre de 2011. Disponible en: http://quince.infragistics.com.
18. WELIE, Martin Van. 2008. Welie.com-Patterns in interaction Design. Citado el: 14 de septiembre de 2011. Disponible en: http://www.welie.com.
19. CARRERON, María. 2008. Construcción de un catálogo de patrones de requisitos funcionales para ERP. s.l.: Universidad Politécnica de Cataluña, 2008.
20. WELICKI, León. 2006. Meta-Especificación y Catalogación de Patrones de Software con Lenguajes de Dominio Especifico y Modelos de Objetos Adaptativos: Una vía para la Gestión del Conocimiento en la Ingeniería del Software. Tesis Doctoral. S.l.: Universidad Pontificia de Salamanca, campus Madrid., 2006.
21. FUENTES, Lidia., JIMENEZ, Daniel. y PINTO, Mónica. 2004. Hacia un entorno de desarrollo integrado basado en Componentes y Aspectos. Málaga: Departamento de Lenguajes y Ciencias para la Computación, ETSI Informática Campus de teatinos s/n., 2004.
22. LÓPEZ, Oscar, LAGUNA, Miguel Ángel y MARQUÉS, José Manuel. 2001. Reutilización del Software a partir de Requisitos Funcionales en el Modelo de Mecano: Comparación de Escenarios. s.l.: Universidad de Valladolid, 2001.
23. ZAPATA, Carlos, HERNÁNDEZ, Juan y ZULUAGA, Raúl. 2008. UNC-Corpus Corpus de diagramas UML para la solución de problemas de completitud en Ingeniería de Software. s.l.: Universidad EAFIT, 2008. Vol. 44.
24. Stig Saether Bakken, Alexander Aulbach, Egon Schmid, Jim Winstead, Lars Torben Wilson, Rasmus Lerdorf, Zeev Suraski, Andrei Zmievski, y Jouni Ahto" Manual de PHP", editado por Rafael Martínez. Publicado 15-04-2001 Copyright © 1997, 1998, 1999, 2000, 2001 por el Grupo de documentación de PHP.
25. extreme Programming explained. Consultado el 4 de febrero de 2012. Disponible en: http://extremeprogramming.host56.com
26. Extreme Programming. Consultado el 26 de enero de 2012. Disponible en http://www.extremeprogramming.org/
27. Pilgrim, Mark. HTML5 Up and Running. s.l.: O'Reilly, August 2010.
28. A Gentle Introduction to Symfony Build professional websites faster with PHP and Symfony. Symfony 1.3 & 1.4. 2011
29. González, P. Programación web II. Santiago. s.l. : I.S.C.B.M., 2009

30. NetBeans. 2010. NetBeans. Consultado el 15 de febrero de 2013. Disponible en: http://netbeans.org/index_es.html.
31. PostgreSQL, E.e.d.d.d. Manual del usuario de PostgreSQL. 2008.
32. Alberto Ruiz. Entorno de desarrollo NetBeans. Consultado el 12 de febrero de 2013. Disponible en: http://observatorio.cnice.mec.es/modules.php?op=modload&name=News&file=article&sid=720 .
33. Historia de PHP y Proyectos Relacionados. Consultado el 17 de octubre de 2012. Disponible en: http://php.net/manual/es/history.php
34. Bootstrap. Consultado en febrero 2013. Disponible en: http://twitter.github.io/bootstrap/
35. Zaninotto, F.; Potencier F. (2007) (en inglés). The Definitive Guide to Symfony (1ra edición). Apress. pp. 486. ISBN978-1-59059-786-6.
36. The technological benefits of Symfony in 6 easy lessons. Consultado en enero 2013. Disponible en: http://symfony.com/six-good-technical-reasons.
37. Ian Hickson, David Hyatt (ed.). HTML5. W3C. Consultado el 06 de octubre de 2012. Disponible en: http://dret.net/biblio/reference/sxbl.
38. PostgreSQL, Award Winning Software. Consultado en febrero 2013. Disponible http://postresql.org
39. Pressman. El Proceso Unificado de Desarrollo de Software: Desarrollo Ágil. Cap. 4
40. Letelier, P. Penadés, C. Metodologías ágiles para el desarrollo de software: XP-; Universidad Politécnica de Valencia.
41. Stevens, Pooley. El Lenguaje Unificado de Modelado. Wesley, 2002
42. Pressman, Roger S. Ingeniería del Software: Un enfoque práctico. 2002
43. Tran, Eushiuan. Validación. Carnegie Mellon University: s.n., 2004.
44. Beck, Kent. Extreme Programming Explained. Primera Edición. 1999
45. Polo, Macario. Mantenimiento Avanzado de Sistemas de Información. Ciudad Real: s.n., 2002. 13071
46. Campderrich Falgueras, Benet. Ingeniería del Software, 2003.
47. Sebastián, J. Modelo Vista Controlador- Definición y Características. 2010.

GLOSARIO DE TÉRMINOS

API: Siglas del inglés Application Programming Interface o Interfaz de programación de aplicaciones. Es el conjunto de funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

CDDL: Siglas del inglés Common Development and Distribution License (Licencia Común de Desarrollo y Distribución). Es una licencia de código abierto (OSI) y libre, producida por Sun Microsystems, basada en la Mozilla Public License o MPL, versión 1.1. La licencia CDDL fue enviada para su aprobación al Open Source Initiative el 1 de diciembre de 2004, y fue aprobada como una licencia de código abierto a mediados de enero de 2005. En el primer borrador hecho por el comité de divulgación de licencias OSI, la CDDL es una de las nueve licencias más populares, mundialmente usadas o con fuertes comunidades.

DOM: Siglas del inglés Document Object Model o Modelo de Objetos para la Representación de Documentos. Es esencialmente una interfaz de programación de aplicaciones (API) que proporciona un conjunto estándar de objetos para representar documentos HTML y XML, un modelo estándar sobre cómo pueden combinarse dichos objetos, y una interfaz estándar para acceder a ellos y manipularlos.

GPL: Siglas del inglés GNU General Public License o Licencia Pública General de GNU. Es la licencia más ampliamente usada en el mundo del software y garantiza a los usuarios finales (personas, organizaciones, compañías) la libertad de usar, estudiar, compartir (copiar) y modificar el software. Su propósito es declarar que el software cubierto por esta licencia es software libre y protegerlo de intentos de apropiación que restrinjan esas libertades a los usuarios. Esta licencia fue creada originalmente por Richard Stallman fundador de la Free Software Foundation (FSF) para el proyecto GNU (GNU Project).

XHTML: Siglas del inglés eXtensible HyperText Markup Language. XHTML es básicamente HTML expresado como XML válido.