Universidad de las Ciencias Informáticas

Facultad 3

Título: Componentes visuales para la Ventanilla Única de Comercio Exterior de Cuba.

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autora: Ailin Díaz Palma

Tutor: Ing. Eddie Nelson Beltrán González

La Habana, Junio 2014

DECLARACIÓN DE AUTORÍA

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

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

Ailin Díaz Palma

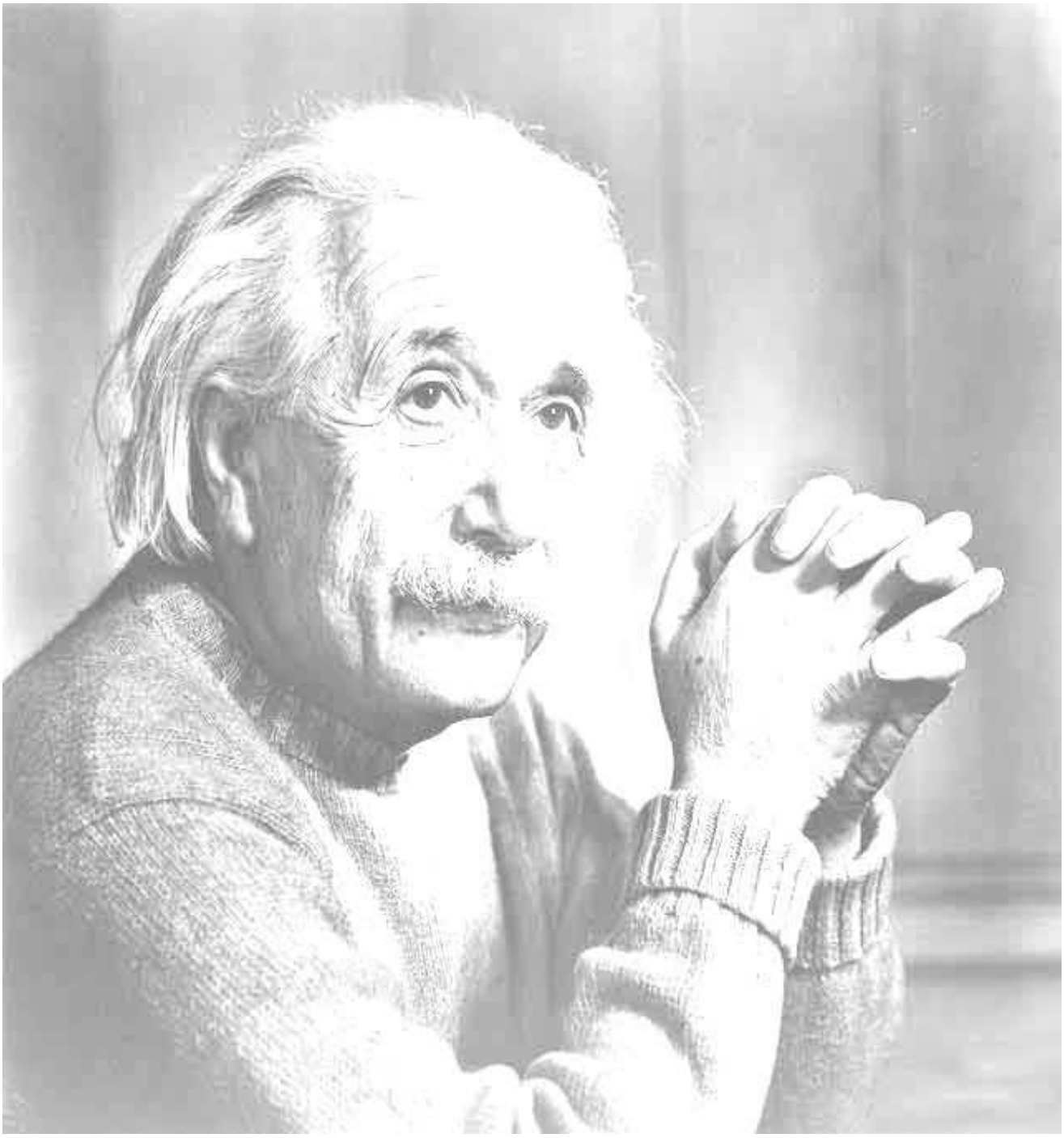
_____

Firma del Autor

Eddie Nelson Beltrán González

_____

Firma del Tutor



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

Albert Einstein

DATOS DE CONTACTO

Ingeniero en Ciencias Informáticas con 2 años de experiencia de trabajo en el desarrollo de interfaces de usuario en la VUCEC.

AGRADECIMIENTOS

A mi mamá por ser la persona más importante de mi vida, por estar junto a mí cada vez que lo he necesitado, por todo el apoyo que me ha brindado, por creer y confiar en mí. Este triunfo también es suyo y lo que soy hoy en gran parte se lo debo. Gracias por todo lo que me has dado sin condición alguna.

A mi papá por apoyarme en tantos momentos, aunque muchas veces hemos estado lejos. A ti te debo en gran medida este logro, este sueño, gracias por hacer todo lo que hiciste para que llegara hasta aquí.

A mi maravillosa familia, mi abuela Reina, mi abuelo Félix en donde quiera que esté, mi tía Danay, mi tío Basto, mis primas Amanda y Lisandra por brindarme su ayuda en todo momento y ser tan preocupados y atentos conmigo.

A mi novio Hector que aunque ahora estamos lejos siempre me ha ayudado, por tanto apoyo, amor, cariño, por confiar en mí, por hacerme crear que todo va a estar bien, por aguantarme en mis malos momentos y comprenderme.

A mi tutor Eddie por brindarme su ayuda y conocimiento en este trabajo de diploma, además de guiarme en cada paso que fui dando.

A mis amigas Yanet y Mayi por sus consejos, por transmitirme energía positiva incluso en los momentos más difíciles, por sus sonrisas, por apoyarme y entenderme siempre, por todos los momentos lindos que pasamos juntas. Gracias por alegrarme los días.

A todos los que de una forma u otra contribuyeron a la realización de este trabajo de diploma muchas gracias.

DEDICATORIA

Este trabajo está dedicado especialmente con mucho amor y cariño a la memoria de mi abuelo Félix, donde quiera que se encuentre sé que estará muy orgulloso de ver mi realización profesional. Que sepa que si algo me animó siempre a seguir hacia adelante sin importar obstáculos a lo largo de esta carrera es el sentimiento de que él siempre se encuentra conmigo en cada paso que doy en la vida, motivación suficiente para superarme cada día más y luchar para que la vida me depare un futuro mejor, que estoy segura es lo que él querría que sucediese.

RESUMEN

La Aduana General de la República (AGR) como parte de la revolución tecnológica e informática del mundo actual, desea informatizar los procesos llevados a cabo en las operaciones comerciales de Cuba, para ello solicitó los servicios de la Universidad de las Ciencias Informáticas (UCI). Esta a través del Centro de Informatización de la Gestión de Entidades (CEIGE) decidió desarrollar el proyecto de Ventanilla Única de Comercio Exterior de Cuba (VUCEC) como solución a la problemática planteada. El avance de este proyecto ha sido afectado en gran medida por las demoras en cuanto a la creación de las interfaces de la capa de presentación, debido a que se desarrollan implementando funcionalidades similares a otras ya existentes.

Por ello, en el presente trabajo de diploma se tiene como objetivo desarrollar un paquete de componentes visuales que permita mejorar la reutilización de código y reducir el tiempo de desarrollo de las interfaces de usuario en la VUCEC, contribuyendo a que sea cumplido en el tiempo establecido el cronograma de desarrollo del proyecto. Con el fin de realizar la tarea propuesta se identificaron los requerimientos de interfaz que presentan un comportamiento común, para encapsularlos en la implementación de los componentes.

PALABRAS CLAVES

Capa de presentación, requerimientos de interfaz, componente, VUCEC.

SUMMARY

The General Customs of the Republic (AGR) as part of the technological revolution and information technology of today's world, you want to computerize the processes carried out in the business operations of Cuba, for that requested the services of the University of Informatics Sciences (UCI). This through the Center of Informatization Management Entities (CEIGE) decided to develop the draft Single Window for Foreign Trade of Cuba (VUCEC) as a solution to the issues raised. The progress of this project has been greatly affected by delays in terms of creating interfaces presentation layer, because develop implementing similar functionalities that exist prior to what you want to develop.

Therefore, in this diploma work is aimed at developing a set of visual components that will improve code reuse and reduce development time of user interfaces in VUCEC, allowing it to be completed within the time established project development schedule. In order to perform the proposed interface task requirements that have a common behavior, to encapsulate these behaviors in implementing the components are identified.

KEYWORDS

Presentation layer, interface requirements, component, VUCEC.

TABLA DE CONTENIDOS

INTRODUCCIÓN

El desarrollo ascendente de las Tecnologías de la Información y las Comunicaciones (TIC) es un hecho incuestionable que forma parte de la cultura tecnológica que caracteriza al mundo contemporáneo y que se ha expandido a todos los ámbitos y estratos de la sociedad ampliando las posibilidades de desarrollo social. El conjunto de transformaciones propiciadas por esta innovación tecnológica están provocando una creciente demanda de software[1] con mayor calidad. La necesidad de producirlos en ciclos de desarrollo más cortos y con mayor retorno sobre la inversión, llevó a que los ingenieros de la rama se dieran a la tarea de desarrollar nuevas vías o alternativas que les garanticen construir un sistema en tiempo récord, cumplir con los estándares de calidad y reducir al máximo los esfuerzos humanos y económicos.

Las exigencias antes mencionadas han impulsado el surgimiento de nuevos paradigmas, tecnologías y formas de llevar a cabo el proceso de producción, donde se encuentra el Desarrollo de Software Basado en Componentes (DSBC). Esta disciplina cuenta actualmente con un creciente interés, tanto desde el punto de vista académico como industrial, en donde la demanda de estos temas es cada día mayor. [1]

En consonancia con el desarrollo y demanda del software, la Universidad de las Ciencias Informáticas (UCI) es una institución que incursiona en esta rama de la industria. Su objetivo fundamental es la formación y superación de profesionales de la informática y la producción de software y servicios asociados para la industria nacional y de exportación. Al mismo tiempo, servir de soporte al programa de informatización de la sociedad cubana. Esta institución cuenta con varios centros encargados de desarrollar diferentes líneas de investigación para lograr un óptimo nivel en la calidad de cada software que se produzca.

En la actualidad, aun cuando se ha alcanzado una mayor experiencia en el tema del DSBC, todavía son muchas las empresas del mundo que desarrollan productos implementando funcionalidades similares, empleando tiempo y recursos innecesarios debido a que existe con anterioridad lo que se desea desarrollar. La UCI en conjunto con la Aduana General de la República (AGR) no está ajena de cometer estos errores. El proyecto Ventanilla Única de Comercio Exterior de Cuba (VUCEC) en su primera versión, surge como parte de varios esfuerzos por informatizar los procesos llevados a cabo en las operaciones comerciales de Cuba. Dentro de las tecnologías definidas en la arquitectura base del mismo se

[1] Conjunto de programas y rutinas que permiten a la computadora realizar determinadas tareas.

encuentran las referentes al desarrollo de las interfaces de usuario, siendo estas el punto central del presente trabajo.

Como tecnologías base fueron definidas, JQuery como biblioteca de JavaScript y Bootstrap como marco de trabajo Hojas de Estilo en Cascada (CSS) para la presentación. Las tecnologías anteriormente especificadas suplen necesidades básicas para el desarrollo de las interfaces, sin embargo, en pos de mejorar la experiencia de los usuarios y de los desarrolladores se hace necesario el uso de componentes que permitan extender las funcionalidades, comportamientos y lógica de negocio no provistas por estas tecnologías.

La inclusión de nuevos componentes desarrollados por terceros es antecedida por un profundo estudio de las diferentes soluciones similares existentes que permita establecer ventajas, desventajas y capacidad de adecuación con las tecnologías bases definidas para la VUCEC. Varios de estos componentes requieren la implementación de un conjunto de funcionalidades no provistas por los mismos, las cuales son realizadas dentro de estos, razón que dificulta su actualización y por tanto la corrección de errores.

El uso de estos componentes varía de acuerdo a como fueron definidos por los autores, no siguiendo un estándar e introduciendo variaciones en forma de uso y configuración, razón por la cual se requiere un gran conocimiento de cada uno de estos. Si bien estos abarcan una gran parte de los escenarios, existen casos particulares del negocio de la VUCEC que los mismos no comprenden; implicando esto que el código implementado sea replicado en disímiles lugares. Todo lo anteriormente planteado conduce a una baja reutilización, alta intrusión del código fuente de los componentes y por consiguiente, retardos en la implementación de las interfaces lo cual transitivamente afecta el cronograma de desarrollo del proyecto.

Durante el análisis de la situación descrita, se identificó el siguiente **problema a resolver**: el modo en que se desarrollan las interfaces de usuario del sistema VUCEC no garantiza la reutilización de código e incrementa los tiempos de desarrollo.

Por lo antes expuesto, el **objetivo general** de este trabajo es desarrollar un paquete de componentes visuales que permita mejorar la reutilización de código y reducir el tiempo de desarrollo de las interfaces de usuario en la VUCEC.

Se plantea como **objeto de estudio**: proceso de desarrollo de interfaces de usuario en la VUCEC y como **campo de acción**: proceso de implementación de componentes visuales para la VUCEC.

Para dar cumplimiento al objetivo general de este trabajo se determinó que era necesario dar

cumplimiento a los siguientes **objetivos específicos**:

- Establecer un marco teórico de referencia.
- Identificar los componentes visuales a desarrollar, de manera que se describan las características y necesidades por cada uno de ellos.
- Implementar los componentes visuales bajo un estándar definido.
- Validar la reducción del tiempo de implementación y el aumento de la reutilización de código.

Para dar cumplimiento al objetivo general se tienen en cuenta las siguientes **tareas de la investigación**:

- Análisis y síntesis de la bibliografía recopilada, de modo que permita identificar soluciones que den respuestas al problema planteado en la investigación.
- Medir el tiempo y cantidad de código empleado en el desarrollo de interfaces sin el empleo de los componentes visuales.
- Identificación de los componentes visuales a partir de un análisis de los requerimientos de interfaz.
- Descripción de los componentes identificados, especificando funcionalidades y características.
- Identificación de componentes existentes que den soporte como base al desarrollo de los componentes descritos.
- Realizar el diseño de los componentes descritos.
- Implementación de los componentes visuales bajo la arquitectura de la VUCEC.
- Medir el tiempo y cantidad de código empleado en el desarrollo de interfaces con el empleo de los componentes implementados.
- Validar la reducción del tiempo de implementación y el aumento de la reutilización de código mediante comparación de los experimentos realizados.

Para dar cumplimiento a las tareas de la investigación y con el objetivo de obtener conocimientos necesarios y más profundos que hagan posible la materialización del objetivo general, se emplean diferentes métodos científicos de investigación como son:

Métodos Teóricos:

- Analítico-Sintético: este método permite analizar y distinguir los elementos que integran los componentes de software, para estudiar cada uno de ellos por separado y posteriormente integrarlos en una nueva unidad, partiendo de la comprensión total de la esencia de dichos elementos.
- Histórico-Lógico: este método facilita el estudio de algunos marcos de trabajo y modelos de desarrollo de componentes, así como identificar los principales patrones de diseño e implementación más eficientes en el proceso de software orientado a componentes.
- Modelación: este método posibilita la creación de modelos que representan abstracciones de los componentes para comprender su funcionamiento y realizar su posterior implementación.

Métodos Empíricos:

- Entrevista: este método posibilita obtener información luego de conversaciones planificadas con personas que tienen experiencia y conocimiento sobre el tema.
- Observación: este método permite explorar lo que sucede con la capa de presentación de la VUCEC, para así percibir de modo objetivo las necesidades reales que debe satisfacer el paquete de componentes a desarrollar.
- Experimento: este método se emplea para validar la solución mediante las pruebas de software realizadas.

Posibles resultados:

Una Interfaz de Programación de Aplicaciones (API) con la descripción de los componentes visuales que incluyan lógica de negocio, características, comportamientos dinámicos y reutilizables que permitan mejorar la reutilización de código en un nivel mínimo de 80% y reducir el tiempo de desarrollo de las interfaces de usuario.

La tesis está compuesta por introducción, tres capítulos, conclusiones, recomendaciones, referencia bibliográfica y anexos que complementan la información comprendida en ella.

Capítulo 1: está dedicado al estudio de conceptos como componente de software, así como aspectos relacionados al desarrollo de software basado en componentes; además se realiza un análisis de las tecnologías y herramientas utilizadas en esta investigación, de modo que permita lograr un entendimiento de los aspectos más importantes del domino del problema.

Capítulo 2: se realiza el análisis y la descripción de la solución que dará respuesta al problema planteado al inicio del presente trabajo de diploma.

Capítulo 3: se describe todo el proceso de implementación y pruebas realizadas a los componentes descritos con anterioridad.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

Introducción

El desarrollo de la informática y las telecomunicaciones ha proporcionado un cambio en la forma que se programa actualmente un software. Una de las causas principales es la demanda de tiempo de desarrollos más cortos, así como el costo que implica crear una aplicación. Por tal motivo se hace necesario introducir nuevas estrategias como el uso de componentes de software.

En este capítulo se estudiarán conceptos como componente de software, así como aspectos relacionados con la programación basada en componentes. Además se desglosan las herramientas, tecnologías y lenguajes para la realización de los componentes.

1.1- Desarrollo de software basado en componentes (DSBC)

1.1.1- DSBC

Julio Casal Terreros, plantea: “La complejidad de los sistemas computacionales actuales nos ha llevado a buscar la reutilización del software existente. El desarrollo de software basado en componentes permite reutilizar piezas de código pre elaborado que permiten realizar diversas tareas, conllevando a diversos beneficios como las mejoras a la calidad, la reducción del ciclo de desarrollo y el mayor retorno sobre la inversión”.[2]

Lo más importante que se debe tener en cuenta al definir el concepto de DSBC, es que hay que diferenciar las características existentes entre este y el paradigma de objetos. La Programación Orientada a Objetos (POO), es cuando el software puede ser desarrollado de acuerdo a un modelo mental de un objeto real o imaginado, mientras que el DSBC dice que el software debe ser desarrollado a partir de componentes prefabricados.

Por lo que el DSBC, no es más que una aproximación del desarrollo de software que describe, construye y utiliza técnicas de software para la elaboración de sistemas abiertos y distribuidos mediante el ensamblaje de partes software reutilizables [3]. La aproximación DSBC es utilizada para reducir los costes, tiempos y esfuerzos de desarrollo del software, a la vez que ayuda a mejorar la fiabilidad, flexibilidad y la reutilización de la aplicación final. Durante algunos años, DSBC fue referida como una filosofía conocida como “compre y no construya” promulgada por Fred Brooks en 1987 y que abogaba por la utilización de componentes prefabricados sin tener que desarrollarlos de nuevo.

1.1.2- Componente

Se entiende por Componente “una unidad de composición de aplicaciones software que posee un conjunto de requisitos incorporados al sistema y compuestos con otros componentes de forma independiente en tiempo y espacio”. [4]

Existen algunas características claves para que un elemento pueda ser catalogado como componente: [1]

- Identificable: debe tener una identificación que permita acceder fácilmente a sus servicios y que permita su clasificación.
- Auto contenido: un componente no debe requerir de la utilización de otros para finiquitar la función para la cual fue diseñado.
- Puede ser remplazado por otro componente: se puede remplazar por nuevas versiones u otro componente que lo remplace y mejore.
- Con acceso solamente a través de su interfaz: debe asegurar que estas no cambiarán a lo largo de su implementación.
- Sus servicios no varían: las funcionalidades ofrecidas en su interfaz no deben variar, pero su implementación sí.
- Bien documentado: un componente debe estar correctamente documentado para facilitar su búsqueda si se quiere actualizar, integrar con otros, adaptarlo, etc.
- Es genérico: sus servicios deben servir para varias aplicaciones.
- Reutilizado dinámicamente: puede ser cargado en tiempo de ejecución en una aplicación.
- Independiente de la plataforma: hardware, software, sistema operativo.

El paradigma de ensamblar componentes y escribir código para hacer que estos funcionen posee varios beneficios: [2]

- Reutilización del software: una de las principales ventajas del DSBC se basa en la “reutilización” de los mismos. De esta forma, los componentes se diseñan y desarrollan con el objetivo de poder ser reutilizados en otras aplicaciones reduciendo el tiempo de desarrollo, mejorando la fiabilidad del producto final (al usar componentes ya probados previamente) y siendo más competitivos en costes.

- Simplifica las pruebas: permite que las pruebas sean ejecutadas probando cada uno de los componentes antes de probar el conjunto completo de componentes ensamblados.
- Simplifica el mantenimiento del sistema: cuando existe un débil acoplamiento entre componentes, el desarrollador es libre de actualizar y/o agregar componentes según sea necesario, sin afectar otras partes del sistema.
- Mayor calidad: dado que un componente puede ser construido y luego mejorado continuamente por un experto u organización, la calidad de una aplicación basada en componentes mejorará con el paso del tiempo.

De la misma manera, el optar por comprar componentes de terceros en lugar de desarrollarlos posee algunas ventajas: [2]

- Ciclos de desarrollo más cortos: la adición de una pieza dada de funcionalidad tomará días en lugar de meses o años.
- Funcionalidad mejorada: para usar un componente que contenga una pieza de funcionalidad, solo se necesita entender su naturaleza, mas no sus detalles internos. Así, una funcionalidad que sería poco práctica de implementar en la empresa, se vuelve ahora completamente asequible.

El disponer de componentes de software no es suficiente para desarrollar aplicaciones, ya provengan estos de un mercado global o sean desarrollados a medida para la aplicación. Por lo que es necesario el modelado de desarrollo software como se explica en el acápite siguiente.

1.2- Modelo de desarrollo de software

Un aspecto crítico es la forma en la que se va a construir el producto y para ello el equipo de desarrollo de CEIGE, propone un modelo de desarrollo de software que detalla el ciclo de vida de sus proyectos.

Algunas características de este modelo son: [5]

- Centrado en la arquitectura: la arquitectura es una vista del diseño completo con las características más importantes, dejando a un lado los detalles. Ésta no sólo incluye las necesidades de los usuarios e inversores, sino también otros aspectos técnicos como el hardware, sistema operativo, sistema de gestión de base de datos, protocolos de red con los que debe coexistir el sistema. La arquitectura representa la forma del sistema, la cual va madurando en su interacción con los casos de uso hasta llegar a un equilibrio entre funcionalidad y características técnicas.

- Orientado a componentes: nos lleva a alcanzar un mayor nivel de reutilización de software, aún en contextos distintos a aquellos para los que fue diseñado. Permite que las pruebas sean ejecutadas probando cada uno de los componentes antes de probar el conjunto completo de componentes ensamblados.
- Iterativo e incremental: la alta complejidad de los sistemas actuales hace que sea factible dividir el proceso de desarrollo en varios mini-proyectos. Cada uno de estos mini-proyectos se les denomina iteración y pueden o no representar un incremento en el grado de terminación del producto completo. En cada iteración los desarrolladores seleccionan un grupo de casos de uso, los cuales se diseñan, implementan y prueban. La planificación de iteraciones hace que se reduzcan los riesgos de los costes de un solo incremento, sacar al mercado un producto en el tiempo previsto y mantener la motivación del equipo, pues puede ver avances claros a corto plazo y que el desarrollo pueda adaptarse a los cambios en los requisitos.
- Ágil y adaptable al cambio: los clientes y funcionales son involucrados en el proyecto, por lo que poseen parte de la responsabilidad del éxito del mismo. Semanalmente se concilian, discuten y aprueban los cambios. El desarrollo de las partes formaliza solo las características principales de la solución, priorizándose de esta manera los talleres y las comunicaciones.

Por cada fase deben ser generados una serie de artefactos los cuales ayudan a una mejor comprensión del proceso de desarrollo que se lleva a cabo [6]. A continuación se presentan dichos artefactos, además de otros aspectos que corresponden a cada fase y que son utilizados en el desarrollo del presente trabajo de diploma.

1.2.1- Descripción de la fase Modelado del negocio

Es la fase destinada a comprender los procesos de negocio de la organización. Se entiende cómo funciona el negocio que se desea informatizar, para tener garantías de que el software desarrollado va a cumplir su propósito.

Artefacto:

- Modelo conceptual.

1.2.2 - Descripción de la fase Requisitos

El esfuerzo principal en la fase de requisitos es desarrollar un modelo del sistema que se va a construir. Incluye un conjunto de artefactos que describen todas las interacciones que tendrán los usuarios con el software y que responden a los requisitos funcionales del sistema. Se especifican los requisitos funcionales y no funcionales.

Artefacto:

- Especificación de requisitos de software

1.2.3- Descripción de la fase de Análisis y Diseño

Durante esta fase es modelado el sistema para que soporte todos los requisitos. Esto contribuye a una arquitectura sólida y estable que se convierte en un plano para la próxima fase. Los artefactos generados en esta etapa son más formales y específicos de una implementación. En caso de llevarse a cabo la reutilización de componentes de software ya desarrollados, durante esta fase se ajusta el modelado existente a los requisitos actuales.

Artefactos:

- Diagrama de clases del diseño por componentes o por módulos
- Diagrama de componentes
- Diagrama de paquetes

1.2.4- Descripción de la fase Implementación

A partir de los resultados del análisis y diseño se implementa el sistema en términos de componentes, es decir, ficheros de código fuente, scripts, etc. Al reutilizar componentes de software ya implementados se lleva a cabo el desarrollo necesario para ajustar a los requisitos actuales y posteriormente realizar la integración de los componentes.

Artefactos:

- Ficheros de código

1.2.5- Descripción de la fase de Pruebas internas:

Durante esta fase el grupo de calidad del centro desarrolla los casos de pruebas verificando el resultado de la implementación. Permite identificar posibles errores en la documentación y el software, es decir, requisitos que el producto debería cumplir y que aún no los cumple.

1.3- Arquitectura de Software

Todo modelado de desarrollo de software está sujeto a un pensamiento basado en la arquitectura de dicho software; es primordial que se realice una arquitectura correcta de los sistemas de software antes de lanzarse a programar, como bien planteaba en el año 1968 Edsger Dijkstra de la Universidad Tecnológica en Holanda.

Existen muchas definiciones sobre arquitectura de software. Perry y Wolf en 1992 [7] la definieron como un conjunto de elementos que tienen una forma común. Garlan and Shaw en 1994 y 1996 [8] la conceptualizaron como una colección de componentes y conectores unidos con una descripción de la interacción entre esos componentes y conectores. Bass, Clements, y Kazman en 1998 [9] la definen como la estructura de las estructuras de un sistema, la cual comprende componentes de software, las propiedades visibles externas de estos componentes y las relaciones entre ellos.

La autora de la tesis se acoge a la definición oficial de arquitectura de software que brinda el documento de IEEE Std 1471-2000 [10] y que está planteada de la siguiente manera: “La Arquitectura de Software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución”.

¿Por qué es importante una arquitectura de software? Debido a que esta define la organización e interacción entre los distintos componentes del mismo, asegura que los requerimientos más importantes puedan ser evaluados e implementados. Una arquitectura de software también permite flexibilidad en el sistema pues facilita la ejecución de futuros cambios y promueve la reutilización de componentes existentes.

En síntesis, la arquitectura debería [10]:

- Mostrar la estructura del sistema pero ocultar los detalles.
- Realizar todos los casos de usos.
- Satisfacer en la medida de lo posible los intereses de los agentes.

- Ocuparse de los requisitos funcionales y de calidad.
- Determinar el tipo de sistema a desarrollar.
- Determinar los estilos arquitectónicos que se usarán.
- Tratar las principales cuestiones transversales.

1.3.1- Estilos Arquitectónicos

Los estilos arquitectónicos fundamentan la relación de cada funcionalidad de manera estructurada y jerárquica, son artefactos de ingeniería importantes porque definen clases de diseño junto con las propiedades conocidas asociadas a ellos. Ofrecen evidencia basada en la experiencia sobre la forma en que se ha utilizado históricamente cada clase, junto con razonamientos cualitativos para explicar por qué cada clase tiene esas propiedades específicas.

Para el desarrollo de los componentes se emplea un patrón de la familia de patrones encapsulados en los estilos de Llamada y Retorno. Es utilizado porque enfatizan en la disponibilidad y la escalabilidad, pues permite organizar las estructuras y componentes fundamentales de manera dinámica y porque son los estilos más generalizados en sistemas en gran escala. Son la arquitectura de programa principal y subrutina, los sistemas basados en llamadas a procedimientos remotos, los sistemas orientados a objeto y los sistemas jerárquicos en capas.

1.3.2- Patrón Arquitectónico

Los patrones arquitectónicos o de arquitectura, son patrones de diseño de software que ofrecen soluciones a problemas de arquitecturas en ingeniería de software. Dan una descripción de los elementos y el tipo de relación que tienen, junto con un conjunto de restricciones sobre cómo pueden ser usados. Un patrón arquitectónico expresa un esquema de organización estructural esencial para un sistema de software, que consta de subsistemas, sus responsabilidades e interrelaciones. En comparación con los patrones de diseño, los patrones arquitectónicos tienen un nivel de abstracción mayor.

Aunque un patrón arquitectónico comunica una imagen de un sistema, no es una arquitectura como tal. Un patrón arquitectónico es más un concepto que captura elementos esenciales de una arquitectura de software. Muchas arquitecturas diferentes pueden implementar el mismo patrón y por lo tanto compartir las mismas características. Además, los patrones son a menudo definidos como una cosa "estrictamente descrita y comúnmente disponible". Por ejemplo, la arquitectura en capas es un estilo de Llamada y

Retorno, cuando se define un estilo general para interaccionar. Cuando esto es descrito estrictamente y comúnmente disponible, es un patrón.

Uno de los aspectos más importantes de los patrones arquitectónicos es que encarnan diferentes atributos de calidad. Por ejemplo, algunos patrones representan soluciones a problemas de rendimiento y otros pueden ser utilizados con éxito en sistemas de alta disponibilidad. Al principio de la fase de diseño, un arquitecto de software escoge qué patrones arquitectónicos ofrecen la calidad deseada para el sistema.

Arquitecturas en Capas

La programación por capas es un estilo de programación, en el que el objetivo primordial es la separación de la lógica de negocios de la lógica de diseño; un ejemplo básico de esto consiste en separar la capa de datos de la capa de presentación al usuario. [11]

La ventaja principal de este estilo es que el desarrollo se puede llevar a cabo en varios niveles y en caso de que sobrevenga algún cambio, sólo se ataca al nivel requerido sin tener que revisar entre código mezclado. Un buen ejemplo de este método de programación sería el modelo de interconexión de sistemas abiertos.

Además, permite distribuir el trabajo de creación de una aplicación por niveles, de este modo, cada grupo de trabajo está totalmente abstraído del resto de niveles, de forma que basta con conocer la API que existe entre niveles.

Por las características del presente trabajo solo se trabaja en la capa de presentación de la cual se muestra a continuación una breve descripción:

- Capa de presentación: es la capa con la que interactúa el cliente o usuario (también se la denomina como "capa de usuario"); presenta las interfaces del sistema, le comunica y captura la información del usuario en un mínimo de proceso (realiza un filtrado previo para comprobar que no hay errores de formato). También es conocida como interfaz gráfica y debe tener la característica de ser "amigable" (entendible y fácil de usar) para el usuario.

1.3.3- Patrones de diseño orientados a objetos

Un patrón es una solución a un problema determinado al que se le da un nombre y que se puede aplicar a nuevos contextos; idealmente proporciona consejos sobre el modo de aplicarlo en varias circunstancias.

Durante la etapa de diseño de un software se conciben un grupo de patrones que de acuerdo a la solución que brindan cubren una necesidad determinada en el diseño.

Existen dos grupos de patrones de diseño principales, los Patrones Generales de Software para Asignación de Responsabilidades (GRASP siglas de General Responsibility Assignment Software Patterns en inglés) y los patrones correspondientes al Grupo de los Cuatro (GOF siglas de Gang of Four en inglés) que es el nombre con el que se conoce por lo general a los autores del libro Design Patterns. [12]

Los patrones GRASP describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones. Los patrones GRASP que se utilizan en la solución son:

- Bajo Acoplamiento: el bajo acoplamiento es un principio que se debe tener siempre en cuenta durante las decisiones de diseño. Es un patrón evaluativo que el diseñador aplica al juzgar sus decisiones de diseño. Soporta el diseño de clases más independientes, que reducen el impacto de los cambios, y también más reutilizables, que acrecienten la oportunidad de una mayor productividad.
- Alta cohesión: el patrón Alta Cohesión es la meta principal que ha de tenerse en cuenta en cada momento en todas las decisiones de diseño. Es un patrón evaluativo que el desarrollador aplica al valorar sus decisiones de diseño. Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme. Una clase con baja cohesión hace muchas cosas no afines o un trabajo excesivo. Colabora con otros objetos para compartir el esfuerzo si la tarea es grande.
- Creador: el patrón Creador nos ayuda a identificar quién debe ser el responsable de la creación (o instanciación) de nuevos objetos o clases. La nueva instancia deberá ser creada por la clase que:
    - Tiene la información necesaria para realizar la creación del objeto.
    - Usa directamente las instancias creadas del objeto.
    - Almacena o maneja varias instancias de la clase.
    - Contiene o agrega la clase. [13]

Una de las consecuencias de usar este patrón es la visibilidad entre la clase creada y la clase creador. Una ventaja es el bajo acoplamiento, lo cual supone facilidad de mantenimiento y reutilización. La creación de instancias es una de las actividades más comunes en un sistema orientado a objetos. En consecuencia

es útil contar con un principio general para la asignación de las responsabilidades de creación. Si se asignan bien, el diseño puede soportar un bajo acoplamiento, mayor claridad, encapsulación y reutilización.

1.4- Capa de presentación

1.4.1- Lenguajes y tecnologías del lado del cliente

Un lenguaje de programación es un mecanismo mediante el cual es posible crear programas a través de un conjunto de instrucciones, operadores y reglas que permiten establecer la comunicación entre el programador y los dispositivos, ya sean hardware o software existentes. [14] Para el desarrollo de aplicaciones web[2], existen los lenguajes que basan su procesamiento en el navegador web, es decir que se ejecutan en el navegador del usuario. Estos son conocidos como lenguajes del lado del cliente.

Lenguaje de Marcado de Hipertexto (HTML) Versión 5

Es la actualización de HTML el lenguaje que se emplea para el desarrollo de páginas web. No es propiamente un lenguaje de programación sino un sistema de etiquetas. Es un lenguaje de marcado que se diseñó con el objetivo de estructurar documentos y mostrarlos en forma de hipertexto, permite establecer relaciones unidireccionales entre ellos, es decir sirve para crear páginas web, darles estructura y contenido.

HTML 5 agrega elementos nuevos al lenguaje, a varios elementos existentes se le agregan atributos y otros cambian. Incorpora nuevas etiquetas como canvas, audio, video, header, footer, article, nav, time entre otras. Además, permite generar tablas dinámicas que pueden filtrar, ordenar y ocultar contenido en la capa de presentación. [15]

Hojas de Estilo en Cascada (CSS) Versión 3

Fueron diseñadas y desarrolladas por la World Wide Web Consortium (W3C), actualmente encargado de formular la especificación de las hojas de estilo que servirán de estándar para los agentes de usuario o navegadores. Son usadas para definir la presentación de un documento estructurado escrito en HTML o XML. Es un mecanismo simple que describe cómo se va a mostrar un documento en la pantalla, o cómo se va a imprimir, o incluso cómo va a ser pronunciada la información presente en ese documento a través de un dispositivo de lectura.

[2] Herramientas que los usuarios pueden utilizar accediendo a un servidor web a través de Internet o de una intranet mediante un navegador.

Ofrece la posibilidad de definir reglas y estilos de representación en diferentes dispositivos ya sean pantallas de equipos de escritorio, portátiles, móviles, impresoras u otros dispositivos capaces de mostrar contenidos web. Cuenta con opciones de sombreado y redondeado y funciones avanzadas de movimiento y transformación. Permite a los desarrolladores web controlar el estilo y el formato de múltiples páginas web al mismo tiempo. La idea que se encuentra detrás del desarrollo de CSS es separar la estructura de un documento de su presentación. Es imprescindible para crear páginas web. [16]

Lenguaje JavaScript

Según Mozilla Developer Network 6 (MDN), "JavaScript es un lenguaje de script multiplataforma. Es un lenguaje pequeño y ligero; no es útil como un lenguaje independiente, más bien está diseñado para una fácil incrustación en otros productos y aplicaciones, tales como los navegadores web". [17]

Principales Ventajas:

- Lenguaje de scripting seguros.
- Los scripts tienen capacidades limitadas, por razones de seguridad.
- El código JavaScript se ejecuta del lado del cliente. [18]

Principales Desventajas:

- Código visible por cualquier usuario.
- El código debe descargarse completamente.
- Puede poner en riesgo la seguridad del sitio, con el actual problema llamado XSS (significa en inglés Cross Site Scripting renombrado a XSS por su similitud con las hojas de estilo CSS). [18]

Asynchronous JavaScript and XML (Ajax)

Es una técnica de desarrollo web para crear aplicaciones interactivas. Es un conjunto de tecnologías independientes que se unen de forma nueva y sorprendente. Las tecnologías que forman AJAX son:

- XHTML y CSS para crear una presentación basada en estándares.
- DOM (Document Object Model) para la interacción y manipulación dinámica de la presentación.
- XML y JSON para el intercambio y la manipulación de la información.

- JavaScript para unir todas las demás tecnologías.

Estas aplicaciones se ejecutan en el cliente, es decir, en el navegador de los usuarios, mientras se mantiene la comunicación asincrónica con el servidor en un segundo plano. De esta forma es posible realizar cambios sobre las páginas sin la necesidad de que estas sean recargadas, lo que significa aumentar la interactividad, velocidad y usabilidad en las aplicaciones. [19]

1.4.2- Marco de Trabajo

La palabra inglesa "framework" (marco de trabajo) define, en términos generales, un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar.

En el desarrollo de software, un framework o infraestructura digital, es una estructura conceptual y tecnológica de soporte definido, normalmente con artefactos o módulos de software concretos, que puede servir de base para la organización y desarrollo de software. Típicamente, puede incluir soporte de programas, bibliotecas, y un lenguaje interpretado, entre otras herramientas, para así ayudar a desarrollar y unir los diferentes componentes de un proyecto.

Representa una arquitectura de software que modela las relaciones generales de las entidades del dominio, y provee una estructura y una especial metodología de trabajo, la cual extiende o utiliza las aplicaciones del dominio. [20]

JQuery

JQuery es un marco de trabajo JavaScript, creado inicialmente por John Resig, el cual fue presentado el 14 de enero de 2006 en el Bar Camp NYC. Es rápido, conciso y facilita la navegación de un documento HTML, manipulación de eventos, animación e interacción con la tecnología Ajax para desarrollos web rápidos, es una librería pensada para interactuar con los elementos de una web por medio del DOM. Además, permite el intercambio asíncrono de datos entre cliente y servidor de manera sencilla. [21]

JQuery presenta distintas características que hacen de esta una biblioteca muy favorable: [21]

- Consiste en un único fichero JavaScript que contiene las funcionalidades comunes de DOM, eventos, efectos y AJAX.
- Compatible con los navegadores Firefox 17.0+, Safari 3.0+, Opera 15+, Internet Explorer10.

- Permite cambiar el contenido de una página web sin necesidad de recargarla, mediante la manipulación del árbol DOM y peticiones AJAX. Para ello utiliza las funciones $() o JQuery().
- Permite crear efectos visuales y modificar los estilos CSS.
- Permite extender la librería de una forma muy simple, esto es conocido como plugins JQuery.

Principales ventajas: [21]

- Es flexible y rápido para el desarrollo web.
- La licencia open source de JQuery permite que la librería siempre cuente con soporte constante y rápido, publicándose actualizaciones de manera constante.
- Tiene una excelente comunidad de soporte, es activa y sumamente trabajadora.
- Se pueden agregar plugins fácilmente, traduciéndose esto en un ahorro substancial de tiempo y esfuerzo.
- Excelente integración con AJAX.

Principales desventajas: [21]

- Gran cantidad de versiones publicadas en corto tiempo.
- Es fácil de instalar y aprender, inicialmente. Pero no es tan fácil si se compara con CSS.
- Si JQuery es implementado inapropiadamente como un framework, el entorno de desarrollo se puede salir de control.

Ventajas de JQuery con respecto a otras alternativas:

JQuery no es el único marco de trabajo que existe en el mercado, existen varias soluciones similares que también funcionan muy bien. Cada uno de los marcos de trabajos tiene sus ventajas e inconvenientes, pero JQuery es un producto con una aceptación por parte de los programadores muy buena y un grado de penetración en el mercado muy amplio, lo que hace suponer que es una de las mejores opciones. Además, es un producto estable, bien documentado y con un gran equipo de desarrolladores a cargo de su soporte y actualización. Es interesante la amplia comunidad de creadores de plugins y componentes, lo que hace fácil encontrar soluciones ya creadas en JQuery para implementar asuntos como interfaces de usuario, galerías, votaciones, etc.

JQuery UI

JQuery UI es una librería de componentes para el marco de trabajo JQuery que provee un conjunto de plugins, efectos visuales y widgets para la creación de aplicaciones web. Se utiliza igual que cualquier otra extensión para JQuery, sólo hay que añadir los ficheros .js a la página; primero debe aparecer la biblioteca JQuery y después el resto. [22]

Características: [23]

- Cuenta con Builder (constructor), herramienta que permite a los usuarios descargar solo aquellos componentes que se necesiten; el resultado final es una biblioteca personalizada a la medida de los requisitos de la aplicación a desarrollar. A pesar de esta característica, JQuery UI no cuenta con un administrador de dependencias capaz de cargar automáticamente los archivos .js asociados a cada uno de sus widgets.
- El diseño de los widgets ha sido bien concebido, con líneas nítidas y colores agradables a la vista.
- Posee decenas de temas disponibles, además, cuenta con la herramienta ThemeRoller, la cual permite a los usuarios crear sus propios temas de manera sencilla.
- Existen además del sitio oficial, multitud de espacios donde encontrar ayuda y ejemplos de su uso.
- Es compatible con los navegadores (y sus versiones posteriores) Firefox 17.0+, Safari 3.0+, Opera 15+, Internet Explorer 10+ y Google Chrome.

Twitter Bootstrap

Bootstrap fue desarrollado por Mark Otto y Jacbod Thornton de Twitter, como un marco de trabajo para fomentar la consistencia a través de herramientas internas. Simplifica el proceso de creación de diseños web combinando CSS y JavaScript. Ofrece una serie de plantillas CSS y ficheros JavaScript que permiten integrar el marco de trabajo de forma sencilla y potente en proyectos web. [24]

Este marco de trabajo tiene 12 columnas por defecto y viene con una disposición de cuadrilla estándar de 940 píxeles de ancho. Alternativamente, el desarrollador puede usar un diseño de ancho variable. Para ambos casos, la herramienta tiene cuatro variaciones para hacer uso de distintas resoluciones y tipos de dispositivos: teléfonos móviles, formato de retratos y paisajes, tabletas y computadoras con baja y alta resolución (pantalla ancha); esto ajusta el ancho de las columnas automáticamente.

Los componentes de JavaScript para Bootstrap están basados en la librería JQuery de JavaScript. Los plugins se encuentran en la herramienta de plugin de JQuery. Proveen elementos adicionales de interfaz de usuario como diálogos, tooltips y carruseles. También extienden la funcionalidad de algunos elementos de interfaz existentes, incluyendo por ejemplo una función de auto-completar para campos de entrada (input). La versión 2.3.2 soporta los siguientes plugins de JavaScript: Modal, Dropdown, Scrollspy, Tab, Tooltip, Popover, Alert, Button, Collapse, Carousel, Affix, Transition y Typeahead.

Bootstrap permite crear interfaces que se adapten a los diferentes navegadores, tanto de escritorio como tablets y móviles a distintas escalas y resoluciones. Ofrece un diseño sólido usando estándares como CSS3/HTML5. Es un framework ligero que se integra de forma limpia en la solución propuesta. Funciona con todos los navegadores, incluido Internet Explorer. [25]

1.5- Desarrollo de componentes para el framework JQuery

El estudio del DSBC ha permitido sentar las bases para el diseño e implementación de componentes visuales para JQuery, el cual es el marco de trabajo JavaScript utilizado en la VUCEC. Aunque se suplen las necesidades básicas para la capa de presentación, explotando las características y los mecanismos que propone JQuery; se persigue desarrollar un grupo de componentes usando como base soluciones similares existentes, que extiendan las funcionalidades y comportamientos específicos de la VUCEC y que incluyan en un único componente, tanto la capa de presentación como la lógica asociada al mismo. Todo esto sin perder de vista la necesidad de mantener lo más cercano a cero si es posible el acoplamiento entre las capas, de manera tal que puedan ser desensambladas y utilizadas en el desarrollo de otros componentes con altos niveles de eficiencia.

1.6- Herramientas usadas

1.6.1- NetBeans

NetBeans, el IDE (Entorno de Desarrollo Integrado) Java de Código Abierto.

¿Qué es NetBeans?

NetBeans es un proyecto exitoso de código abierto con una gran base de usuarios, una comunidad en constante crecimiento, y con cerca de 100 socios (¡y creciendo!) en todo el mundo. Sun MicroSystems fundó el proyecto de código abierto NetBeans en junio 2000 y continúa siendo el patrocinador principal de los proyectos.

Al día de hoy hay disponibles dos productos: el NetBeans IDE y NetBeans Platform.

NetBeans IDE es un entorno de desarrollo libre y gratuito sin restricciones de uso; una herramienta para que los programadores puedan escribir, compilar, depurar y ejecutar programas. Está escrito en Java, pero puede servir para cualquier otro lenguaje de programación. Existe además un número importante de módulos para extender el NetBeans IDE.

También está disponible NetBeans Platform que al igual que NetBeans IDE es libre y gratuito; una base modular y extensible usada como estructura de integración para crear grandes aplicaciones de escritorio. Empresas independientes asociadas, especializadas en desarrollo de software, proporcionan extensiones adicionales que se integran fácilmente en la plataforma y que pueden también utilizarse para desarrollar sus propias herramientas y soluciones.

Ambos productos son de código abierto para uso tanto comercial como no comercial. El código fuente está disponible para su reutilización de acuerdo con la Common Development and Distribution License (CDDL) v1.0 y la GNU General Public License (GPL) v2. [26]

Para el desarrollo de los componentes del presente trabajo se utilizó la versión 7.0.

1.6.2- Visual Paradigm

Visual Paradigm para UML es una herramienta para desarrollo de aplicaciones utilizando modelado UML Unified Modeling Language (Lenguaje de Modelado Unificado) ideal para Ingenieros de Software, Analistas de Sistemas y Arquitectos de sistemas que están interesados en construcción de sistemas a gran escala y necesitan confiabilidad y estabilidad en el desarrollo orientado a objetos. Visual Paradigm también ofrece:

- Navegación intuitiva entre la escritura del código y su visualización.
- Potente generador de informes en formato PDF/HTML.
- Documentación automática Ad-hoc.
- Ambiente visualmente superior de modelado.
- Sofisticado diagramador automáticamente de layout.
- ¡Sincronización de código fuente en tiempo real o en demanda y mucho más!

Existen diferentes tipos de versiones que se adaptan a las necesidades del usuario:

- Visual Paradigm for UML Personal Edition (VP-UML LE): esta edición tiene diversas funcionalidades que en la versión Trial están deshabilitadas. Una marca de agua indica que la versión es para uso personal.
- Visual Paradigm for UML Modeler Edition (VP-UML ME): en esta edición no encontrará la marca de agua y también puede hacer uso de la creación de diagramas de dos clases, no permitidos en la edición anterior (VP-UML LE).
- Visual Paradigm for UML Standard Edition (VP-UML SE): tiene todas las características de Modeler Edition (VP-UML ME), agregando una serie de generadores de código que soporta más de 10 lenguajes de programación para ingeniería lineal o inversa.
- Visual Paradigm for UML Professional Edition (VP-UML PE): esta edición impresiona por la capacidad de sincronización de códigos, disminuyendo el tiempo de programación y permitiendo incluso más desarrollo a través del mapeo relacional de objetos, extremadamente intuitivo y fácil de usar.
- Visual Paradigm for UML Enterprise Edition (VP-UML EE): Es la edición top de la línea de productos, lo que representa todo lo más moderno y agrega valores en términos de modelado de datos orientado a objetos, hace posible la documentación del proyecto, mapeo relacional de objetos para Java, .NET y PHP, reduciendo costos y aumentando su productividad. Esta fue la versión usada en el modelado de la aplicación a desarrollar. [27]

Conclusiones parciales

En el presente capítulo se realizó un estudio de las principales definiciones alrededor del DSBC para sustentar la solución del problema, partiendo de que la idea principal de este es desarrollar sistemas desde componentes existentes. Esto permitió sentar las bases para el análisis, diseño e implementación de componentes visuales para JQuery. Se describieron las herramientas, lenguajes, tecnologías y metodología definida en el proyecto VUCEC para el proceso de desarrollo, con el objetivo de conocer los elementos base que dan soporte a los componentes. Para facilitar un mejor entendimiento del próximo capítulo se hace un análisis de los componentes y como serán desarrollados a partir de todos los conceptos y descripciones estudiadas.

CAPÍTULO 2: ANÁLISIS Y DESCRIPCIÓN DE LA SOLUCIÓN

Introducción

En el presente capítulo se tratan los elementos y características de las interfaces de usuario de la VUCEC, para la identificación y análisis de los componentes que se pretenden desarrollar. Se expone además el diseño que se siguió para la elaboración de los mismos, así como los patrones empleados. Igualmente se muestra la concepción de la arquitectura con el objetivo de brindar una organización de los componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución.

2.1- Análisis e identificación de los componentes visuales

Para la identificación de estos componentes se realizó una revisión de las interfaces de la VUCEC. Al cumplir estas con tal grado de similitud, se pueden agrupar las características y funcionalidades comunes para ser comprendidas en componentes, y dar la posibilidad de ser configurables y reutilizables.

Anteriormente estas interfaces han sido desarrolladas de tal modo que no permiten reutilizar el código de las mismas, sino replicarlo una y otra vez en la medida que se necesite. Por tal motivo se han enumerado las necesidades de estas y analizado sus comportamientos, para encapsular en clases los requerimientos visuales, evitando así que el desarrollador implemente una vez más la interfaz y las funcionalidades que esta brinda.

Tras desarrollado el análisis, se identificaron un conjunto de componentes visuales por la cantidad de apariciones que estos hacen hasta el momento (ver tabla #1).

Es importante destacar que el desarrollo de estos componentes está basado en el algoritmo de planificación MFU - (more frequently used) -, el más frecuentemente usado. Este algoritmo plantea que la mayor prioridad recae sobre el elemento que mayor probabilidad de ocurrencia tiene, de esta forma se desarrollarían los componentes de mayor frecuencia y los tiempos de desarrollo de las interfaces correspondientes serían menores.

Tabla #1. Ocurrencias de las necesidades identificadas como componentes.

Necesidades	Ocurrencias	Componente
Actualizar o mostrar mensajes según tipo.	106	vuMessageDialog, vuMessageNotification
Seleccionar datos a partir de una lista de selección simple o enlazada.	98	vuSelect
Manipular un formulario haciendo uso de la integración de otros. Debe permitir enviar, cargar o validar datos, entre otros.	77	vuForm
Mostrar el avance de alguna operación realizada. Este debe poder mostrar en porcientos o letras o según corresponda.	69	vuProgressBar
Manipular, insertar, modificar y eliminar datos a partir de una interfaz externa (Tabla, Formulario).	61	vuFormToGrid
La capacidad de insertar, modificar, buscar, ordenar y filtrar los datos. Debe permitir que se puedan eliminar filas o columnas según se desee.	48	vuGrid
Que este permita seleccionar una fecha o un rango de fecha con opciones para restringir los valores a introducir y facilidades para la selección de los mismos.	33	vuDatePicker
Manipular e intercambiar datos entre dos tablas.	26	vuGridToGrid
Permitir que dado un texto o palabra este sea capaz de autocompletar, permitiendo que se reduzca posible errores en la escritura del mismo y brinde mayor facilidad de uso al cliente.	19	vuTypeahead
La necesidad de seleccionar la hora con opciones para restringir los valores a introducir y facilidades para la selección de la misma.	13	vuTimePicker
La necesidad de realizar comprobaciones de negocio con los datos introducidos mediante AJAX	8	vuTextAjaxValidator

sin tener que recargar la página.		

Para la identificación de estos componentes que se presentan a continuación (ver tabla 2), se realizó un análisis diferente a las interfaces. Al hacer el estudio de estas se encontraron elementos como listas donde no se tenía visibilidad de la jerarquía existente entre las opciones y checkboxs que podían ser representados mediante un componente que mostrara los estados de manera más intuitiva. Después de la observación realizada se decidió implementar los que se presentan a continuación para que suplieran los anteriores casos planteados.

Tabla #2. Característica de los componentes vuTree y vuSwitch.

Necesidades	Componente
Transformar de manera ligera y flexible una lista no jerarquizada en un árbol expandible y plegable, ideal para las mejoras de navegación. Realizar búsquedas, filtrados, carga de datos, selección de opciones, entre otras características.	vuTree
Que sea capaz de hacer función de interruptor. Este debe poder ponerse en true/on o false/off según corresponda la petición del usuario.	vuSwitch

2.2- Descripción de la solución

Teniendo en cuenta el modelo de desarrollo de software del centro CEIGE, se realizará el desarrollo de los componentes aplicando los estándares definidos en la arquitectura del proyecto. El conjunto de componentes en cuestión, permite a los programadores elevar el nivel de abstracción de la lógica que estos abarcan y por consiguiente simplificar el trabajo.

Para una mejor comprensión del funcionamiento de los componentes se describen sus funcionalidades y características, para ello se utilizaron un total de 14 tablas. A continuación se muestra la descripción del componente vuForm:

1. Formulario.

Tabla 3. Descripción del componente Formulario.

Componente	vuForm		
Funcionamiento	El componente proporciona varios métodos para llevar a cabo acciones como insertar, editar, enviar, validar o cargar datos en el formulario. Este utiliza otros componentes de elementos de formularios para realizar estas tareas.		
Depende	form.validation.js		
Opciones	Nombre	Tipo	Descripción
	url	string	Ruta para enviar los datos.
	validate	object	Objeto con todas las validaciones requeridas.
	onError	function	Función que se ejecuta luego de haber sucedido un error en el envío.
	onSubmit	function	Función que se ejecuta luego de haberse enviado los datos correctamente.
Validar	Nombre	Tipo	Descripción
	rules	object	Todos los campos en forma { provincia:{ required:true }, textValidator:{ required:true } } que serán validados
	messages	object	Todos los mensajes en forma { textValidator:'Este campo es obligatorio' } que serán mostrados en caso de error
Métodos	Nombre	Descripción	
	.validar()	Valida el componente verificando que todos los campos sean correctos, ejecuta las reglas especificadas en el validate.	
	.setValue(campo,valor)	Inserta un nuevo valor en el campo especificado.	
	.setValues(object)	Setea los campos mediante el objeto pasado por parámetro, forma:{switchCmp:true, fecha:'10/2/2005', start:'10/2/2012', end:'10/2/2012'}.	
	.getValue(name)	Obtiene el valor del campo especificado.	

	.reset()	Resetea el formulario a su estado original.
	.clear()	Limpia todos los campos del formulario.
	.submit(object)	Envía los datos del formulario, con la configuración especificada por parámetro: {success:function(){}, error:function(){}}.
	.getFields()	Obtiene los valores del formulario contenidos en un objeto. return {campo1:'valor1',campo2:'valor2'}
	.collectData()	Obtiene los valores de cualquier tabla contenida en el formulario. return {nameTable:{header:{},rows:[]}}
Uso	Descripción	
	Llamada al Componente: $("#registro").vuForm();	

2.3- Modelo conceptual

Un modelo conceptual es un conjunto de conceptos y de reglas destinados a representar de forma global los aspectos lógicos de los diferentes tipos de elementos existentes en la realidad que están siendo analizados. Es una representación figurada de una experiencia empírica, que tiene como objetivo ayudar a comprender la realidad. [28]

A continuación se identifican y representan los conceptos relacionados con la solución propuesta.

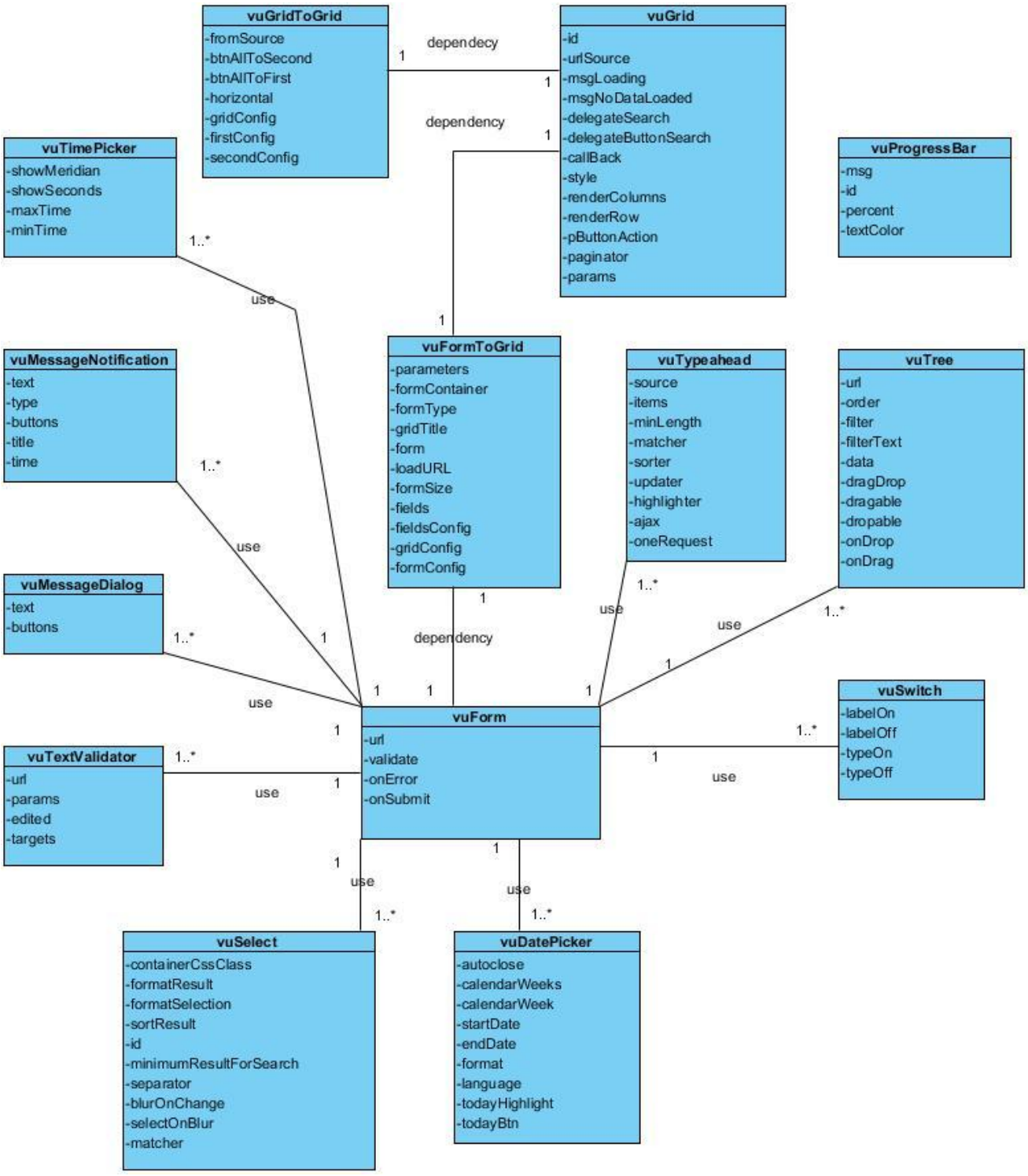


Figura 1. Modelo conceptual.

2.4- Requisitos Funcionales

El objetivo final en cualquier diseño de software es satisfacer los requisitos del usuario para el sistema. Estos requisitos pueden ser requisitos de software, requisitos de productos, o requisitos de pruebas. La meta de capturar y comprobar los requisitos del usuario es asegurar que todos los requisitos son completados, y que el diseño esté acorde con los requisitos especificados.

2.4.1- Técnicas usadas para la captura de requisitos

Por lo complejo que resulta capturar los requisitos funcionales (de modo que lo que entendemos sea lo que el cliente quería decirnos), sobre todo si el entorno de trabajo es desconocido para el equipo de analistas.

Existen algunas técnicas clásicas que pueden ayudar a realizar esta ingeniería de requisitos. Algunas de estas técnicas son: entrevistas, observación, JAD (Joint Application Development/Desarrollo conjunto de aplicaciones), brainstorming (Tormenta de ideas), concept mapping (Mapas conceptuales), sketches y storyboards (Bocetos y Guiones gráficos), entre otras. [29]

Para la captura de requisitos del presente trabajo se tomaron como técnicas las entrevistas y observación de las cuales se dan una pequeña descripción a continuación:

Entrevistas: resultan una técnica muy aceptada dentro de la ingeniería de requisitos y su uso está ampliamente extendido.

- Las entrevistas le permiten al analista tomar conocimiento del problema y comprender los objetivos de la solución buscada.
- A través de esta técnica el equipo de trabajo se acerca al problema de una forma natural. Existen muchos tipos de entrevistas y son numerosas las estructuras, las cuales abarcan estos pasos:
    - Identificación de los entrevistados.
    - Preparación de la entrevista.
    - Realización de la entrevista y documentación de los resultados (protocolo de la entrevista).

Observación: las observaciones proporcionan una manera directa de ver a las personas en su ambiente, el modo en que realizan sus trabajos o tareas y ejecutan los procesos. Son particularmente útiles para procesos detallados, cuando las personas que usan el producto tienen dificultades o se muestran renuentes para articular sus requisitos. La observación también es conocida por el término en inglés "job shadowing". Normalmente la realiza un observador externo, que mira a un experto en el negocio mientras éste ejecuta un trabajo. También puede hacerla un "observador participante", que de hecho lleva a cabo un proceso o procedimiento para experimentar cómo se hace y descubrir requisitos ocultos.

2.4.2- Requisitos funcionales del sistema

Teniendo en cuenta las técnicas descritas anteriormente, para la solución propuesta se tuvieron en cuenta los siguientes requisitos funcionales:

R 1. Gestionar componente vuForm.

R1.1. Obtener los datos de los campos del formulario.

R1.2. Insertar datos en los campos del formulario.

R1.3. Validar el formulario.

R1.4. Enviar datos del formulario al servidor.

R 2. Gestionar componente vuSelect.

R 2.1. Filtrar las opciones de forma local o remota.

R 2.2. Seleccionar una opción.

R 2.3. Establecer dependencias entre listas enlazadas.

R 3. Gestionar componente vuDatePicker.

R.3.1. Mostrar calendario de selección de fecha.

R 3.2. Restringir rango válido de fecha.

R 3.3. Obtener fecha seleccionada.

R 3.4. Insertar nueva fecha.

R 3.5. Seleccionar fecha actual.

R 4. Gestionar componente vuTimePicker.

R 4.1. Restringir rango válido de hora.

R 4.2. Obtener hora seleccionada.

R 4.3. Insertar nueva hora.

R 4.4. Seleccionar hora actual.

R 5. Gestionar componente vuGrid.

R 5.1. Realizar búsqueda en los datos.

R 5.2. Ordenar por columnas.

R 5.3. Paginar los datos.

R 5.4. Realizar selección simple o múltiple.

R 5.5. Insertar filas.

R 5.6. Modificar datos de una fila.

R 5.7. Eliminar filas.

R 6. Gestionar componentes vuMessageDialog.

R 6.1. Mostrar mensaje.

R 6.2. Ocultar mensaje.

R 6.3. Agregar botones.

R 6.4. Insertar contenido del mensaje.

R 7. Gestionar componente vuMessageNotification.

R 7.1. Mostrar mensaje.

R 7.2. Actualizar mensaje.

R 7.3. Ocultar mensaje.

R 7.4. Mostrar tipo de mensaje.

R 7.5. Establecer tipo de mensaje.

R 8. Gestionar componente vuTypeahead.

R 8.1. Mostrar coincidencias de datos.

R 9. Gestionar componente vuTextAjaxValidator.

R 9.1. Comprobar de manera remota la validez de los datos introducidos.

R 10. Gestionar componente vuTree.

R 10.1. Realizar búsqueda.

R 10.2. Realizar ordenamiento.

R 10.3. Hacer selección de datos simple y múltiple.

R 10.6. Cargar datos por demanda.

R 11. Gestionar componentes vuSwitch.

R 11.1. Activar botón.

R 11.2. Desactivar botón.

R 12. Realizar componente vuProgressBar.

R 12.1. Mostar porciento de progreso.

R 12.2. Mostar barra de progreso.

R 12.3. Mostrar texto durante el progreso.

R 13. Realizar componente vuFormToGrid.

R 13.1. Insertar datos a través de los campos del formulario en la tabla.

R 13.2. Editar datos a través de los campos del formulario en la tabla.

R 13.3. Eliminar datos a través de los campos del formulario en la tabla.

R 13.4. Permitir estructurar cómo se visualizan los elementos en el formulario.

R 13.5. Identificar filas nuevas y editadas.

R 13.6. Paginar los datos.

R 13.7. Realizar selección simple o múltiple de los datos.

R 14. Realizar componente vuGridToGrid.

R 14.1. Selección simple o múltiple de los datos de las tablas.

R 14.2. Intercambiar datos entre tablas.

R 15. Cargar datos de forma local o remota.

R 16. Mostrar indicador cuando se carguen los datos por Ajax.

R 17. Construir API.

2.4.3- Técnicas de validación de requisitos.

Es muy importante asegurar la validez de los requisitos previamente a comenzar un desarrollo de software. Para ello debe de hacerse una comprobación de la correspondencia entre las descripciones iniciales y si el modelo es capaz de responder al planteamiento inicial. Para llevar a cabo esto, se suele comprobar que el modelo obtenido responde de la misma forma deseada que la que el cliente pide por un lado, y por otro a la inversa si otras respuestas del modelo convencen al cliente. En algunos casos será necesario construir prototipos con una funcionalidad similar muy reducida para que el cliente se haga una idea aproximada del resultado.

La validación de los requisitos, obviamente tiene como objetivo comprobar que estos son correctos. Esta fase debe realizarse o de lo contrario se corre el riesgo de implementar una mala especificación, con el costo que eso conlleva. Los parámetros a validar en los requisitos son:

- Validez: no basta con preguntar al cliente, todos los potenciales clientes pueden tener puntos de vista distintos y necesitar otros requisitos.
- Consistencia: no debe haber contradicciones entre unos requisitos y otros.
- Completitud: deben estar todos los requisitos. Esto es imposible en un desarrollo iterativo, pero al menos deben estar disponibles todos los requisitos de la iteración en curso.
- Realismo: se pueden implementar con la tecnología actual.
- Verificabilidad: tiene que existir alguna forma de comprobar que cada requisito se cumple.

Uso en MADEJA

Las principales técnicas de validación de requisitos por parte de clientes y usuarios que se proponen para su uso en el área de Ingeniería de Requisitos de *Madeja* son:

- Prototipado de interfaz de usuario
- Recorrido de casos de uso

Prototipado de interfaz de usuario

El prototipado de interfaz de usuario es una técnica de representación aproximada de la interfaz de usuario de un sistema software que permite a clientes y usuarios entender más fácilmente la propuesta de los ingenieros de requisitos para resolver sus problemas de negocio. Los dos tipos principales de prototipos de interfaz de usuario son:

Desechables: se utilizan sólo para la validación de los requisitos y posteriormente se desechan. Pueden ser prototipos en papel o en software.

Evolutivos: una vez utilizados para la validación de los requisitos, se mejora su calidad y se convierten progresivamente en el producto final.

Recorrido de casos de uso

El recorrido o walkthrough es una técnica de revisión de productos tradicionalmente asociada a la inspección de código fuente. Su principal objetivo es encontrar conflictos en el producto que se revisa, de forma que puedan plantearse alternativas y los participantes aumenten su conocimiento del producto en cuestión.

Durante las sesiones de recorrido, el autor del producto recorre el producto a revisar en detalles, permitiendo que los participantes pongan de manifiesto sus opiniones sobre el mismo. Aplicado a la Especificación de Requisitos de Software (ERS), el recorrido permite a clientes y usuarios comprender el significado de cada requisito y manifestar su acuerdo o desacuerdo con los mismos. Además, aplicando esta técnica a los casos de uso se puede validar de manera natural la secuencia de pasos de un caso de uso al recorrerlos por todos los participantes. [30]

Para la validación de los requisitos del presente trabajo se utilizó la técnica de prototipo de interfaz de usuario del cual se muestra un ejemplo a continuación:

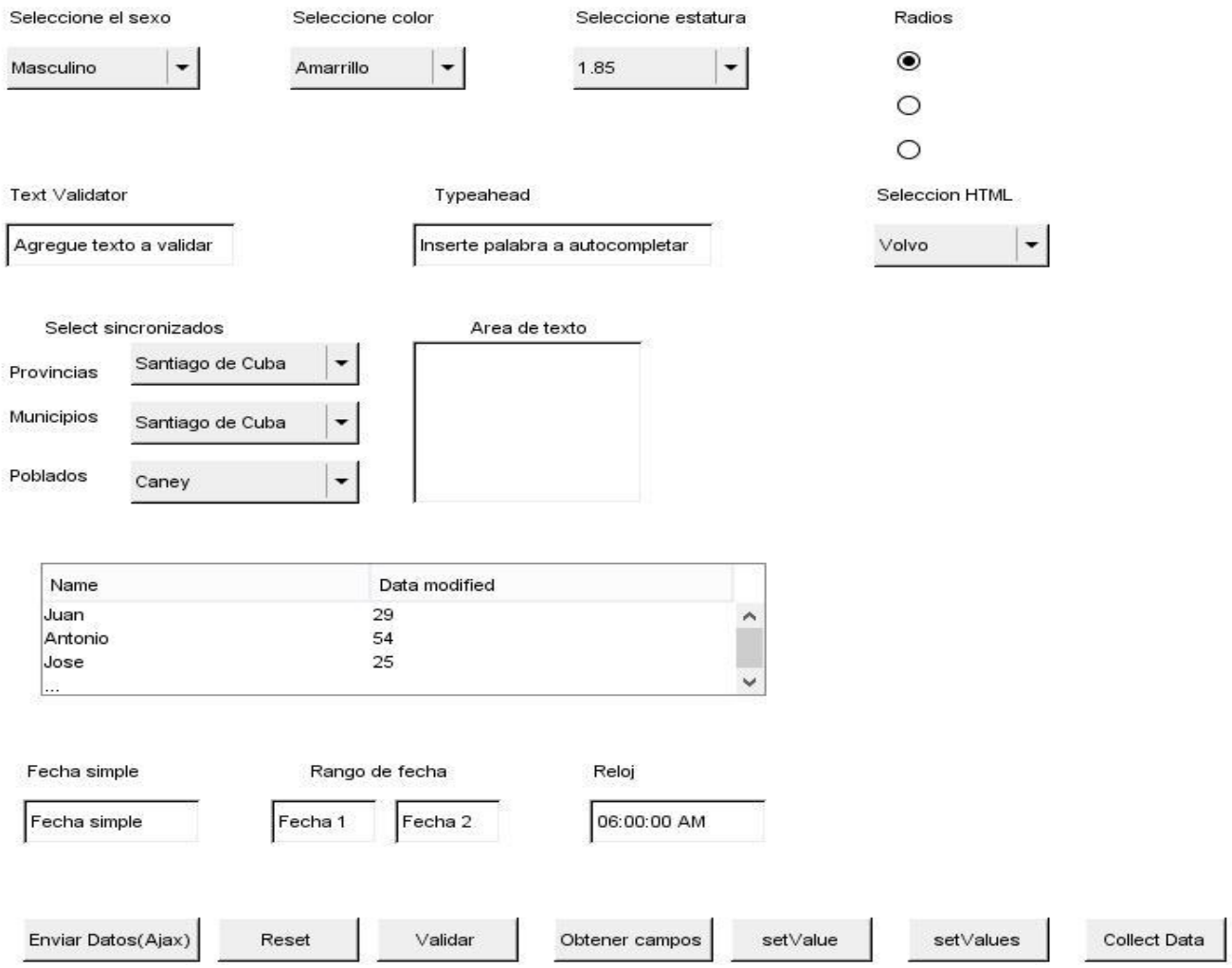


Figura 2. Prototipo de interfaz del componente vuForm.

2.5- Requisitos no funcionales

Los requisitos no funcionales especifican propiedades del sistema como restricciones de entorno, implementación, rendimiento, dependencia de la plataforma, facilidad de mantenimiento, extensibilidad, flexibilidad y fiabilidad. Por tanto, se refieren a todos los requisitos que ni describen información a guardar, ni funciones a realizar.

Algunos requisitos no funcionales hacen referencia a fenómenos del mundo real. Otros son más genéricos y no pueden relacionarse con un caso de uso en concreto o una clase del mundo real, estos se gestionarán a parte en listas de requisitos adicionales.

Requisitos adicionales: estos son requisitos no funcionales que no pueden asociarse a ningún caso de uso, sin embargo, estos tienen impacto en varios casos de uso. Estos se capturan en una lista de requisitos que luego se usan durante el análisis y diseño junto al modelo de casos de uso.

Algunos ejemplos de requisitos no funcionales típicos son los siguientes: rendimiento, disponibilidad, seguridad, accesibilidad, usabilidad, estabilidad, portabilidad, costo, operatividad, interoperabilidad, escalabilidad, concurrencia, mantenibilidad e interfaz.

En este sub-epígrafe se expondrán los requisitos no funcionales que deben cumplir los componentes propuestos, estos requisitos están definidos por el proyecto VUCEC.

Requisitos de usabilidad

- Los componentes deberán ser usados solamente por los integrantes del proyecto o por los clientes que incluyan el producto a su sistema informático.
- La información deberá estar disponible en el período de trabajo definido por los dirigentes del proyecto, limitada solamente por las restricciones de acuerdo a las políticas de seguridad definidas.

Requisitos de fiabilidad

- Los componentes deberán estar disponibles para los integrantes antes mencionados en el período de trabajo definido por los dirigentes del proyecto, solo no lo estará cuando se le esté realizando mantenimiento o suceda alguna inconveniencia externa.

Requisitos de soporte

- Las restricciones y el tiempo de soporte será establecido por el equipo de desarrollo y los clientes que utilicen el sistema, se recomienda preferentemente un día, debido a la importancia que tendrá dicho componente para el sistema.

Requisitos de hardware

- Los componentes deberán estar instalados en una computadora que tenga memoria RAM 512 MB o más.
- Velocidad de procesamiento del microprocesador 1GHz o superior.
- Un disco duro de 80 GB o más de capacidad para poder almacenar los ficheros que generen los componentes.

Requisitos de software

- Sistemas operativos: Microsoft Windows 2000/NT o superior, distribución de GNU/Linux.
- Mozilla Firefox a partir de su versión 20.

Requisitos Legales, de Derecho de Autor y otros

- Como producto, los componentes se distribuirán amparados bajo las normativas legales establecidas en el registro comercial emitido por las entidades jurídicas de la Universidad de las Ciencias Informáticas.
- Los componentes deberán regirse por la ley y los decretos establecidos que norman los diversos procesos que son automatizados.

2.6-Arquitectura

En el desarrollo de los componentes se utilizó la arquitectura del proyecto, la cual es de 3 capas. Por las características del presente trabajo solamente se desarrolló en la capa de presentación, para ello se definió un nuevo modelo en el desarrollo de los componentes basándose en los estándares del proyecto, en la figura que se muestra a continuación se evidencia este nuevo modelo, en donde se puede ver la relación entre vuComponentes, Bootstrap y JQuery.

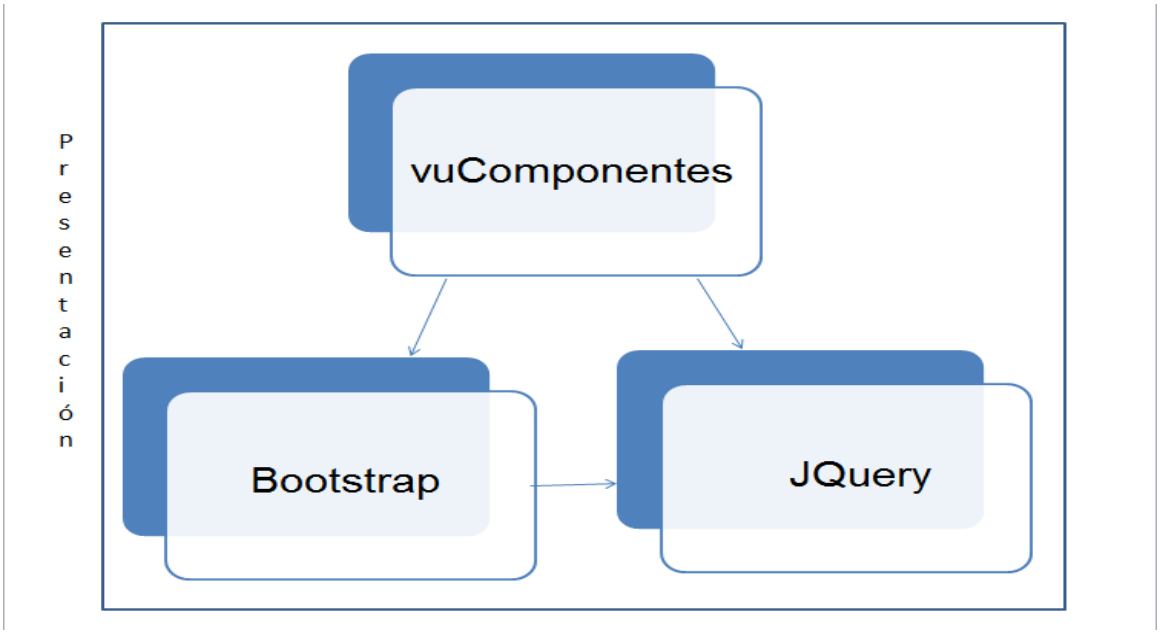


Figura 3. Arquitectura definida para el desarrollo de los componentes.

2.7- Patrones de diseño

Un patrón es una descripción de un problema y su solución, que recibe un nombre y que puede emplearse en otros contextos; en teoría, indican la manera de utilizarlo en circunstancias diversas.

Patrones Grasp:

- Creador: se aplica para la asignación de responsabilidades a las clases relacionadas con la creación de objetos, de forma tal que una instancia de un objeto sólo pueda ser creada por el objeto que contiene la información necesaria para ello. [13] Este patrón se pone de manifiesto en la clase vuForm la cual implementa las funcionalidades para un formulario usando otros componentes. Como se muestra en el segmento de código siguiente:

```
prepareComponentsVu:function(){
    var that=this;

    $(this.element).find('input').each(function(){
            if($(this).data("select2")){
                var sel=$(this).data("select2");
                if(that.options.validate)
                for( var campo in that.options.validate.rules ){
                    if(campo===sel.opts.element.attr('name'))
                        sel.setRequired();

                }

            }
            if($(this).data("switch")){
                var sel=$(this).data("switch");
                if(that.options.validate)
                for( var campo in that.options.validate.rules ){
                    if(campo===sel.options.name)
                        sel.setRequired();

                }

            }
    })

},
```

Figura 4. Segmento de código del componente vuForm.

- Alta cohesión: nos dice que la información que almacena una clase debe de ser coherente y debe estar (en la medida de lo posible) relacionada con la clase. Se aplica para realizar un diseño que evite contener clases con un alto grado de abstracción, que asuman responsabilidades que podían haber delegado a otros objetos o que tengan responsabilidades muy complejas. [13] Ejemplo de esto se manifiesta a través de la clase vuMessage como se muestra en el código siguiente, donde se define los tipos de mensajes que va a tener:

```
Message.prototype={
    constructor:Message,
    init:function(){
    // elementos relacionados con la clase
        var type = this.options.type.toUpperCase();
        var typeApply;
        var title;
        var buttons=new Array();
        switch(type){
            case 'INFO':{ typeApply = this.styleMsg.info;
                          title=VU.Msg.language.info;//'msg-info';
                          buttons.push({text:VU.Msg.language.aceptar,callback:this.close,scope:this,icon:'../../img/message/apply.png',focus:true});
                          var color={background:'#598EFD',color:'white',borderColor:'#4371D4',css:'alert alert-info'};
                          break;
            }
            case 'ERROR':{ typeApply = this.styleMsg.error;//'msg-error';
                          title=VU.Msg.language.error;
                          buttons.push({text:VU.Msg.language.aceptar,callback:this.close,scope:this,icon:'../../img/message/apply.png',focus:true});
                          var color={background:'#E34B49',color:'white',borderColor:'#B33A38',css:'alert alert-error'};
                          break;
            }
            case 'WARNING':{ typeApply = this.styleMsg.warning;//'msg-warning';
                         title=VU.Msg.language.warning;
                         buttons.push({text:VU.Msg.language.aceptar,callback:this.close,icon:'../../img/message/apply.png',scope:this,focus:true});
                         var color={background:'#DCB84F',color:'white',borderColor:'#C5A549',css:'alert alert-warning'};
                         break;
            }
            case 'SUCCESS':{ typeApply = this.styleMsg.success;//'msg-warning';
                         title=VU.Msg.language.warning;
                         buttons.push({text:VU.Msg.language.aceptar,callback:this.close,icon:'../../img/message/apply.png',scope:this,focus:true});
                         var color={background:'#43C527',color:'white',borderColor:'#359920',css:'alert alert-success'};
                         break;
            }
```

Figura 5. Segmento de código del componente vuMessage.

- Bajo acoplamiento: el acoplamiento mide la fuerza con que una clase está conectada a otra, de esta forma una clase con bajo acoplamiento debe tener un número mínimo de dependencia con otras clases. Consiste en tener las clases lo menos ligadas entre sí posible. [13] Ejemplo: la clase vuSelect solo extiende de bootstrap_select alcanzando un bajo acoplamiento entre las clases. A continuación se muestra un ejemplo a través del código usado de como se ve el bajo acoplamiento:

```
(function($){

    if($.fn.select2 == undefined)
        return;

    /**
     * Compares equality of a and b
     * @param a
     * @param b
     */
    function equal(a, b) {
        if (a === b) return true;
        if (a === undefined || b === undefined) return false;
        if (a === null || b === null) return false;
        // Check whether 'a' or 'b' is a string (primitive or object).
        // The concatenation of an empty string (+'') converts its argument to a string's primitive.
        if (a.constructor === String) return a+'' === b+''; // a+'' - in case 'a' is a String object
        if (b.constructor === String) return b+'' === a+''; // b+'' - in case 'b' is a String object
        return false;
    }

    //Functions code here
    //aquí se muestra la herencia
    window.Select2.class.abstract.prototype=$.extend(window.Select2.class.abstract.prototype,{
       /* Método para preparar las opciones
        *  Usado para activar el filtrado en la búsqueda
        *
        */
        prepararOpciones:function(){
                        this.opts.populateResults=function(container, results, query) {
                        var populate,  data, result, children, id=this.opts.id;
                        var self=this;
```

Figura 6. Segmento de código del componente vuSelect.

2.8- Diagramas de clases del diseño

Los diagramas de clases representan un conjunto de elementos del modelo que son estáticos, como las clases y los tipos, sus contenidos y las relaciones que se establecen entre ellos. Contienen la siguiente información: [31]

- Clases asociadas
- Dependencias

En la figura se muestra el diagrama de clases del diseño Gestionar componente vuForm, donde se representan las principales clases, operaciones y relaciones que se necesitan para darle cumplimiento al requisito funcional.

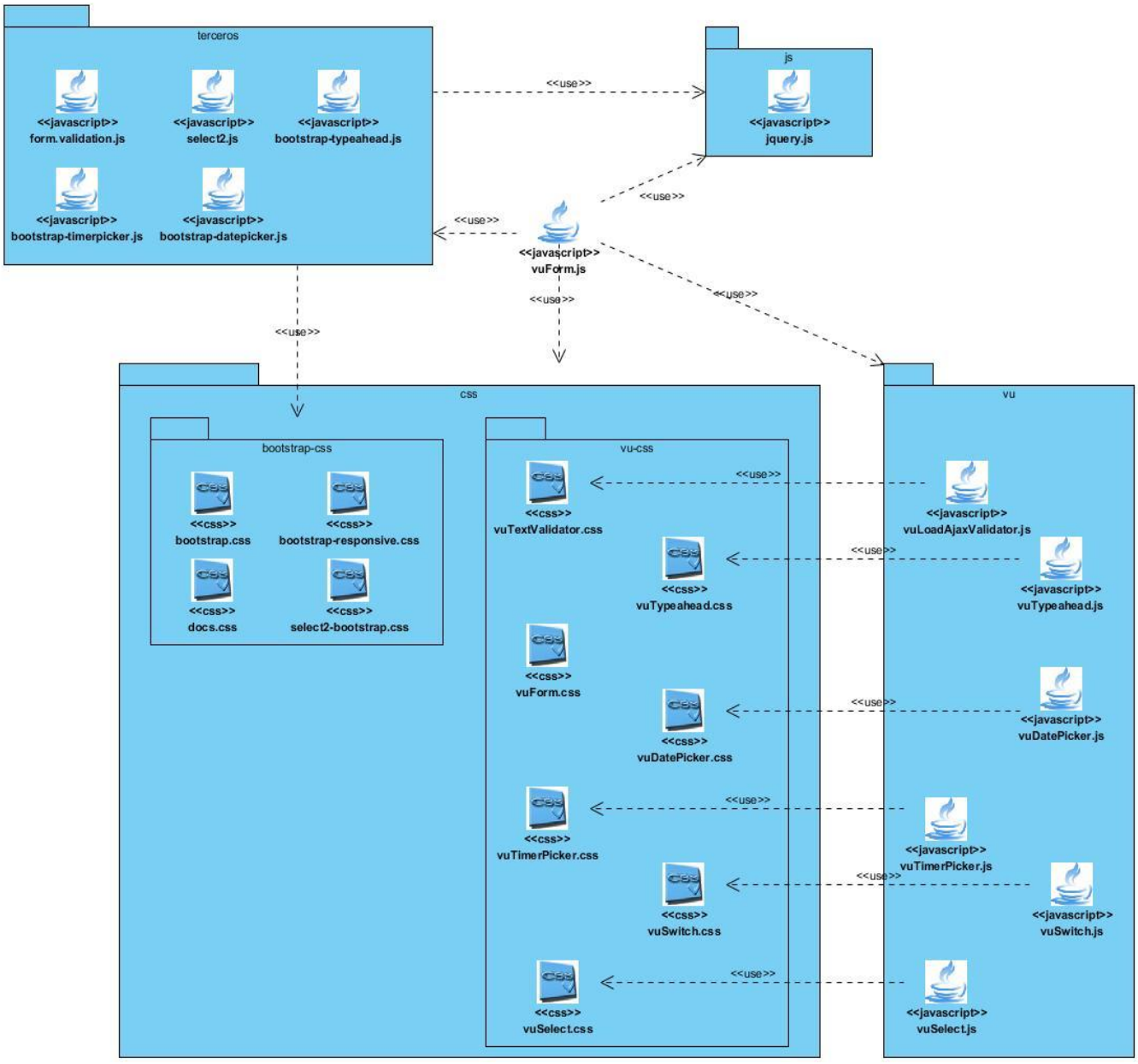


Figura 7. Diagrama de clases del diseño del componente vuForm.

2.9- Métricas para la validación del diseño

Las métricas van a ayudar a la evaluación de los modelos de análisis y de diseño, en donde proporcionarán una indicación de la complejidad de diseños procedimentales y de código fuente, y ayudaran en el diseño de pruebas más efectivas.

Permiten medir de forma cuantitativa la calidad de los atributos internos del software. Esto permite al ingeniero evaluar la calidad durante el desarrollo del sistema.

Atributos de calidad

Las métricas se centran en cuantificar tanto la complejidad, como la funcionalidad y eficiencia inmersa en el desarrollo de software. Inclina sus objetivos a mejorar la comprensión de la calidad del producto, a estimar la efectividad del proceso y mejorar la calidad del trabajo.

Las métricas empleadas están diseñadas para evaluar los siguientes atributos de calidad:

- Responsabilidad: consiste en la responsabilidad asignada a una clase en un marco de modelado de un dominio o concepto, de la problemática propuesta.
- Complejidad de implementación: consiste en el grado de dificultad que tiene implementado un diseño de clases determinado.
- Reutilización: consiste en el grado de reutilización presente en una clase o estructura de clase, dentro de un diseño de software.
- Acoplamiento: consiste en el grado de dependencia o interconexión de una clase o estructura de clase, con otras, está muy ligada a la característica de Reutilización.
- Complejidad del mantenimiento: consiste en el grado de esfuerzo necesario a realizar para desarrollar un arreglo, una mejora o una rectificación de algún error de un diseño de software. Puede influir indirecta, pero fuertemente en los costes y la planificación del proyecto.
- Cantidad de pruebas: consiste en el número o el grado de esfuerzo para realizar las pruebas de calidad (Unidad) del producto (componente, módulo, clase, conjunto de clases, etc.) diseñado.

Tamaño operacional de clase (TOC)

Está dado por el número de métodos asignados a una clase y evalúa los siguientes atributos de calidad:

- 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.

Relaciones entre clases (RC)

Está dado por el número de relaciones de uso de una clase con otra y evalúa los siguientes atributos de calidad:

- 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.

Matriz de inferencia de indicadores de calidad

También llamada matriz de cubrimiento, es una representación estructurada de los atributos de calidad y métricas utilizadas para evaluar la calidad del diseño propuesto.

Dicha matriz permite conocer si los resultados obtenidos de las relaciones atributo/métrica es positivo o no, llevando estos resultados a una escalabilidad numérica donde, si los resultados son positivos se le asigna el valor de 1, si son negativos toma valor 0 y si no existe relación es considerada como nula y es representada con un guion simple (-). Luego se puede obtener un resultado general para cada atributo promediando todas sus relaciones no nulas. [32]

Resultados de las métricas aplicando TOC

Tabla 5. Fórmulas para medir la categoría de los atributos.

Atributo	Categoría	Criterio
Responsabilidad	Baja	< =Prom.
	Media	Entre Prom. y 2* Pom.
	Alta	> 2* Prom.
Complejidad Implementación	Baja	< =Prom.
	Media	Entre Prom. y 2* Pom.
	Alta	> 2* Prom.
Reutilización	Baja	> 2* Prom.
	Media	Entre Prom. y 2*

		Pom.
	Alta	< =Prom.

Tabla 6. Clases a las que se les aplica la métrica.

No.	Clase	Cantidad de Procedimientos	Responsabilidad	Complejidad	Reutilización
1	vu.pnotify	5	Baja	Baja	Alta
2	vuChecks	13	Media	Media	Media
3	vuDatepicker	2	Baja	Baja	Alta
4	vuDialog	14	Media	Media	Media
5	vuForm	18	Media	Media	Media
6	vuGrid	13	Media	Media	Media
7	vuGridToGrid	21	Media	Media	Media
8	vuLoadAjaxData	4	Baja	Baja	Alta
9	vuMessage	12	Baja	Baja	Alta
10	vuProgessbar	16	Media	Media	Media
11	vuSelect	27	Alta	Alta	Baja
12	vuSwitch	15	Media	Media	Media
13	vuTimerPicker	11	Baja	Baja	Alta
14	vuTypeahead	6	Baja	Baja	Alta

Tabla 7. Porciento de cantidad de procedimientos por clases.

Cantidad de Procedimientos		
Criterio	Cantidad de clases	Promedio
Entre 1 y 5	3	21,42857143
Entre 6 y 10	1	7,142857143
Entre 11 y 15	6	42,85714286
Entre 16 y 21	3	21,42857143
Entre 22 y 27	1	7,142857143
Total	14	100

Tabla 8. Porciento de los atributos de calidad responsabilidad y complejidad.

Responsabilidad y Complejidad		
Criterio	Cantidad de clases	Promedio
Baja	6	42,85714286
Media	7	50
Alta	1	7,142857143
Total	14	100

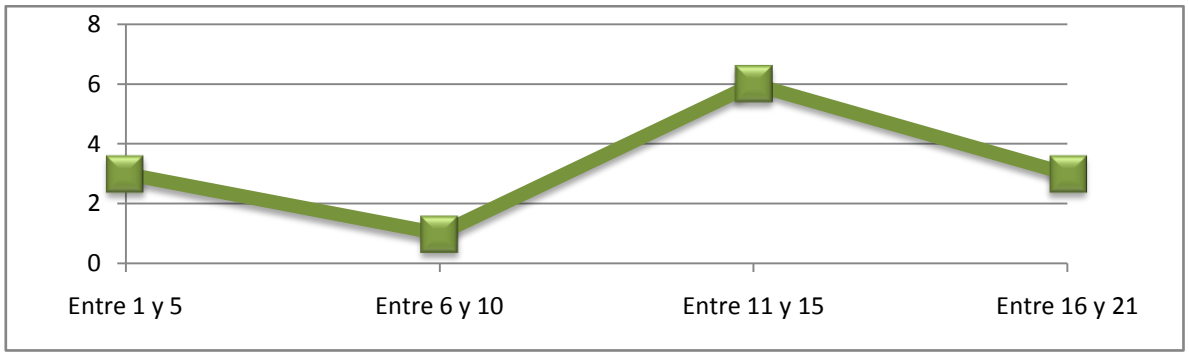


Figura 8. Gráfica para mostrar porciento de los atributos.

Tabla 9. Porciento de los atributos de calidad reutilización.

Reutilización		
Criterio	Cantidad de clases	Promedio
Baja	1	7,142857143
Media	7	50
Alta	6	42,85714286
Total	14	100

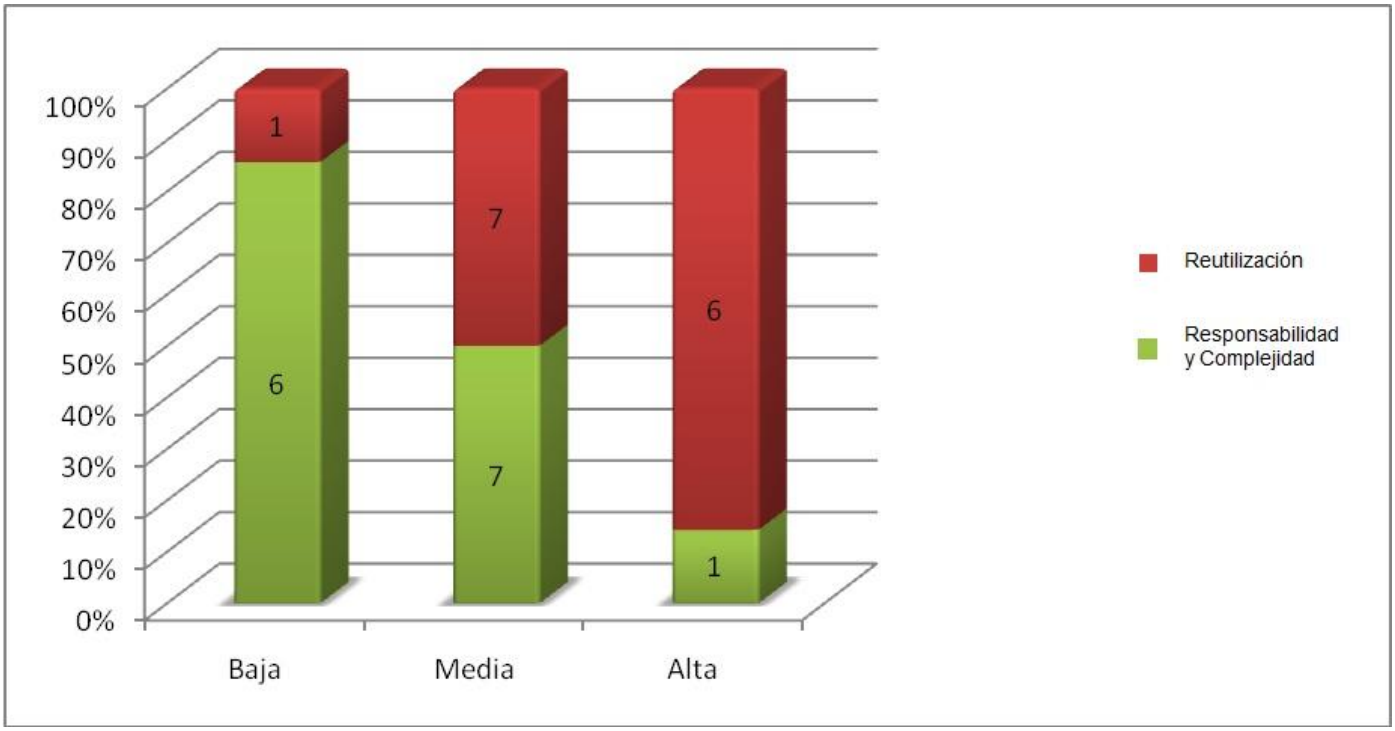


Figura 9. Gráfica para mostrar porciento del atributo de calidad reutilización.

Cuando aumenta el TOC se afectan los parámetros de calidad definidos por esta métrica. Se reduce la reutilización de las clases, la implementación se hace más compleja, las pruebas son difíciles de realizar y aumenta la responsabilidad de las clases. Al analizar los resultados obtenidos luego de aplicar el instrumento de medición de la métrica TOC, se puede concluir que el diseño propuesto para los componentes está entre los límites aceptables de calidad. El 92.85714% de las clases que conforman el sistema están dentro de las categorías de baja y media, lo que demuestra la elevada reutilización ya que supera el nivel mínimo de 80%. El resto se encuentra en la categoría de alta lo que implica baja complejidad de implementación y responsabilidad en el diseño propuesto.

2.10- Diagramas de componentes

El diagrama de componentes tiene como objetivo modelar la estructura del software, incluyendo las dependencias entre los componentes de software, los componentes de código binario, y los componentes ejecutables. En dicho diagrama se modelan componentes del sistema, a veces, agrupados por paquetes y las dependencias que existen entre componentes (y paquetes de componentes).

En la siguiente figura se muestra el diagrama de componentes que representa las relaciones entre componentes del requisito funcional Gestionar componente vuForm del sistema VUCEC.

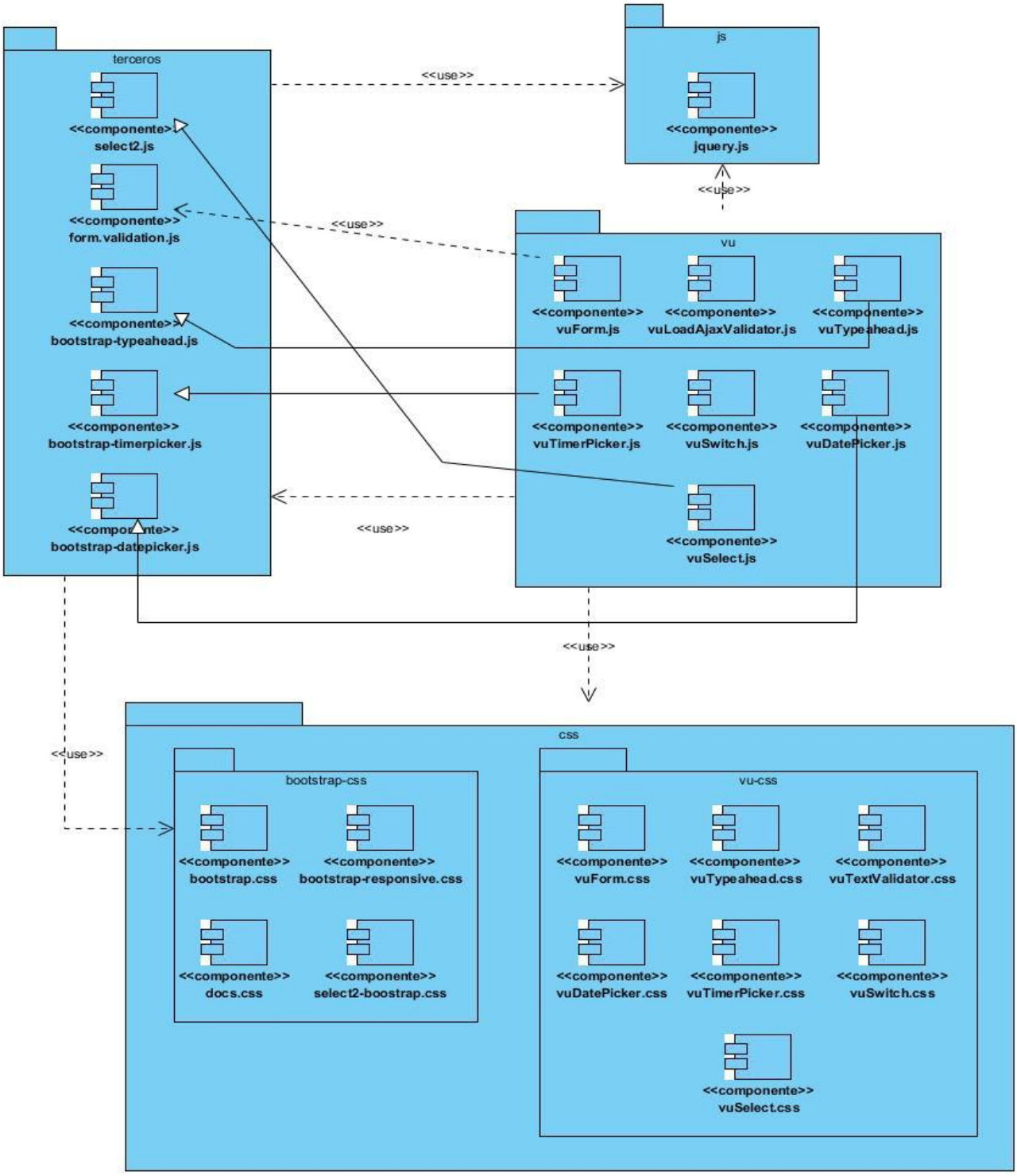


Figura 10. Diagrama de componentes “Gestionar componente vuForm”.

2.11- Diagrama de paquetes

Un diagrama de paquetes muestra cómo un sistema está dividido en agrupaciones lógicas mostrando las dependencias entre esas agrupaciones. Dado que normalmente un paquete está pensado como un directorio, los diagramas de paquetes suministran una descomposición de la jerarquía lógica de un sistema. [33]

Características:

- Los paquetes son buenos elementos de gestión.
- Se usan paquetes en un modelo de desarrollo para agrupar elementos relacionados.
- Cada uno de los paquetes se puede asignar a un individuo o a un equipo de desarrollo.

A continuación se muestra el diagrama de paquetes definido en el sistema:

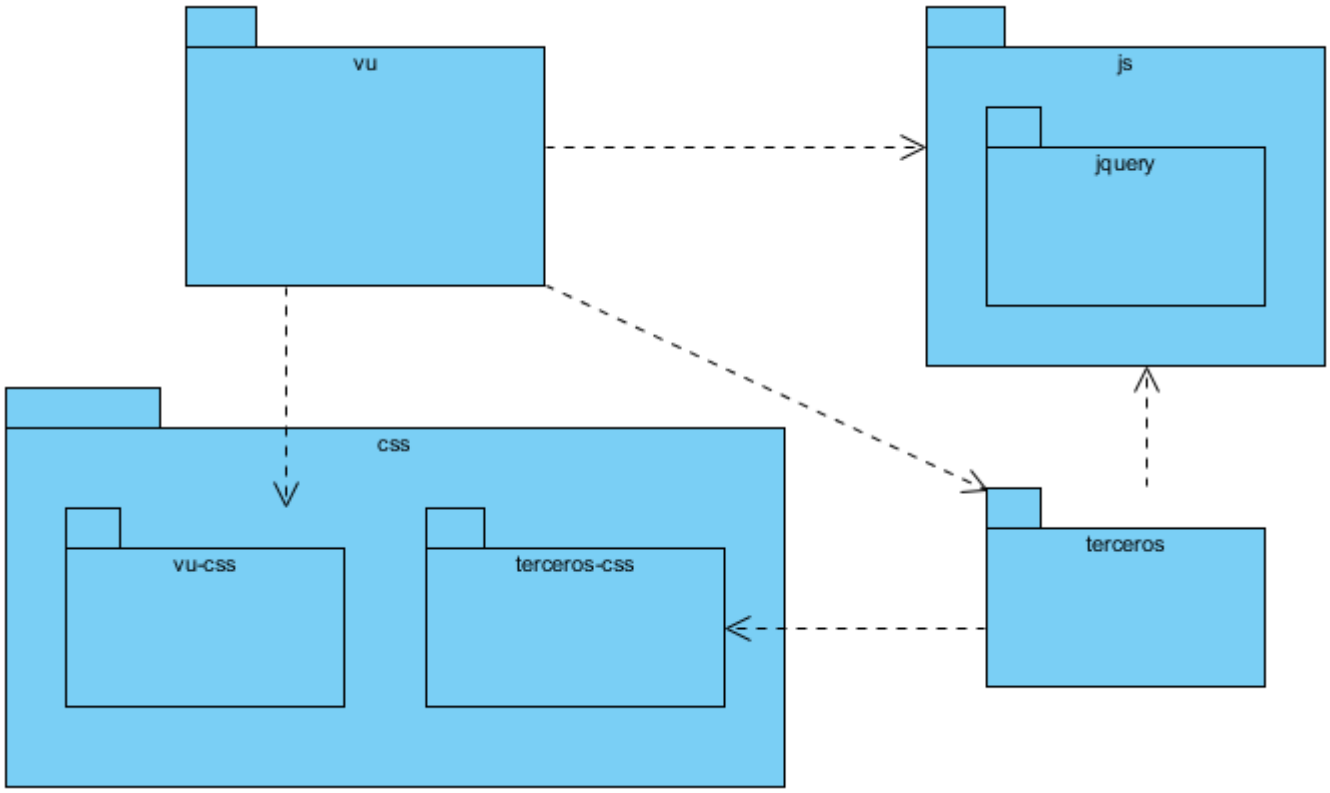


Figura 11. Diagrama de paquetes del sistema.

Descripción de los paquetes:

En el paquete js se encuentra JQuery como biblioteca de JavaScript utilizada para el desarrollo de los componentes visuales de la solución propuesta, los cuales están contenidos en el paquete vu y los implementados por el equipo de desarrollo de bootstrap y otros que están en el paquete terceros. Para apoyar el proceso se utilizó el paquete css que incluye a los terceros-css y vu-css para el estilo en cada caso.

Conclusiones parciales

Al concluir este capítulo se logró identificar los componentes visuales a desarrollar, agrupando las funcionalidades y comportamientos comunes de las interfaces de la VUCEC. Además se capturaron los requisitos funcionales y no funcionales brindando una mayor visión del producto.

También se generaron los artefactos durante el diseño pudiendo llegar a una definición más clara de las características que debe poseer el software. Se aplicó la métrica: TOC para validar y evaluar el diseño, la cual arrojó valores satisfactorios para cada uno de los indicadores correspondientes.

Además, teniendo en cuenta la arquitectura que establece el proyecto, fue posible obtener una mayor comprensión de la organización de los componentes a desarrollar, creando las bases necesarias para proceder a la fase de implementación.

CAPÍTULO 3: IMPLEMENTACIÓN Y PRUEBA

Introducción

En el presente capítulo se presenta cómo fueron desarrollados los componentes en la solución propuesta, así como las pruebas que fueron realizadas al producto para garantizar su correcto funcionamiento durante todo su ciclo de vida.

3.1- Implementación de los componentes

En este acápite se muestran los métodos necesarios para llevar a cabo la implementación de los componentes.

3.1.1- Estándares de código

Un estándar de codificación completo comprende todos los aspectos de la generación de código. Si bien los programadores deben implementar un estándar de forma prudente, éste debe tender siempre a lo práctico. Un código fuente completo debe reflejar un estilo armonioso, como si un único programador hubiera escrito todo el código de una sola vez. Al comenzar un proyecto de software, establezca un estándar de codificación para asegurarse de que todos los programadores del proyecto trabajen de forma coordinada. Cuando el proyecto de software incorpore código fuente previo, o bien cuando realice el mantenimiento de un sistema de software creado anteriormente, el estándar de codificación debería establecer cómo operar con la base de código existente.

Usar técnicas de codificación sólidas y realizar buenas prácticas de programación con vistas a generar un código de alta calidad es de gran importancia para la calidad del software y para obtener un buen rendimiento. Además, si se aplica de forma continuada un estándar de codificación bien definido, se utilizan técnicas de programación apropiadas, y, posteriormente, se efectúan revisiones del código de rutinas, caben muchas posibilidades de que un proyecto de software se convierta en un sistema de software fácil de comprender y de mantener.

Para los componentes desarrollados se siguió el estándar del proyecto VUCEC, que plantea lo siguiente:

Los nombres de las carpetas son creados según la notación Lower Camel Case (las palabras se denotan una a continuación de la otra sin separadores, y partiendo de la segunda palabra se denota la primera letra en mayúscula) (Ejemplo: vuGrid).

En el caso de los ficheros CSS y JS cuando el código se encuentra compactado (código minimizado mediante cualquier técnica para disminuir el tamaño y por consiguiente aumentar la rapidez de la carga del mismo por los navegadores) se le agrega ".min" antes de la extensión del fichero. (EJ: jquery.pnotify.min.js)

En el caso de los ficheros CSS y JS que contienen los estilos y código de la lógica de negocio correspondiente a una interfaz de usuario, la notación debe aplicar el patrón El Nombre Revela la Intención. (Ejemplo: vuTree.js)

Interfaces Bootstrap y JQuery

El desarrollo de las interfaces de usuario se lleva a cabo mediante el uso del marco de trabajo CSS Twitter Bootstrap 2.3.2 y la librería JQuery 1.9.

En los scripts se localiza la lógica implementada para las interfaces de usuario haciendo uso de la librería JQuery, estos deben poseer una estructura básica que permitirá la separación del código del ámbito global. En la figura siguiente se puede ver dicha estructura.

```
(function($){
        VU.namespace('VU.Ejemplo');
        var local = 1;
        global = 2;
        function ejemplo(){
                return "Función de ejemplo.";
        }
        VU.Ejemplo.otroEjemplo = function(){
                return "Otro ejemplo de función asignada a un espacio de nombre definido.";
        }
}) (jQuery)
```

Figura #12. Estructura de los scripts en las interfaces de usuario.

Espacios de nombre (namespace).

Los espacios de nombre (en inglés namespaces) son usados para permitir el acceso global a funciones y variables de manera que no se solapen por coincidencia en los nombres. La función para su creación es “VU.namespace” ó la abreviatura “VU.ns”. (Ejemplo: VU.namespace (‘VU.Ejemplo’) ;). Todos los espacios de nombre del proyecto deben ajustarse a las siguientes condiciones.

- Deben comenzar por “VU” indicando la pertenencia al espacio de nombre de la VUCEC.
- El separador a usar es el “.”.
- Cada nombre en el espacio de nombre debe escribirse en Upper Camel Case (Ejemplo: VU.ns (‘VU.OtroEjemplo’) ;).
- El nombre debe indicar el objetivo o área que ocupa, pasando desde lo más general a lo más específico. (Ejemplo: VU.ns (‘VU.AGR.DespachoComercial’) ;)

Prototipo de clase

La lógica reutilizable en las interfaces es implementada en clases que contienen el comportamiento deseado. JavaScript no posee clases como otros lenguajes orientados a objetos pero logra este comportamiento mediante el uso del “prototype” en las funciones, de tal manera se puede lograr el comportamiento de una clase. En la figura 12 se muestra dicha estructura.

```
(function($){
	VU.ns('VU.ClasesEjemplo');
	//Definición de la clase dentro del espacio de nombre
	VU.ClasesEjemplo.Clase = function(options){
		this.init(options);
	}
	VU.ClasesEjemplo.Clase.prototype = {
		//Propiedades de la clase
		 propiedad: 1
		,otraPropiedad: 'cadena'
		,defaults: {}
		//Se usa la propia función de la clase como constructor
		,constructor: VU.ClasesEjemplo.Clase
		//Funciones de la clase
		,init: function(options){
			//inicialización de la clase
		}
		,otra: function(param1, param2){
			return true;
		}
	};
	//Instanciación de la clase
	var objeto = new VU.ClasesEjemplo.Clase({param:true});

	//declaración del plugins
	$.fn.nombrePlugin =function(options){//Code}
})(jQuery)
```

Figura 13. Estructura de una clase para las interfaces de usuario.

3.2- Prueba de software

El desarrollo del software implica una serie de actividades de producción en las que las posibilidades de que aparezca la falibilidad humana son comunes. Los errores pueden empezar a darse desde el primer momento del proceso en el que los objetivos puedan estar especificados de forma errónea e imperfecta; así como en los posteriores pasos del diseño y desarrollo. Debido a la imposibilidad humana de trabajar y

comunicarse de forma perfecta, el desarrollo del software ha de ir acompañado de una actividad que garantice la calidad.

Las pruebas de software son una actividad en la cual un sistema o componente, es ejecutado bajo condiciones o requerimientos especificados, los resultados son observados y registrados, y una evaluación es hecha de algún aspecto del sistema o componente.

Estas son un elemento fundamental para garantizar la calidad del sistema, representan una revisión final de las especificaciones del diseño y de la codificación, ya que verifican que el software funcione como se diseñó y que los requerimientos se han cumplido, además de documentar los defectos del sistema. [34]

Nivel de Prueba

La prueba es aplicada para diferentes tipos de objetivos, en diferentes escenarios o niveles de trabajo. Se distinguen diferentes niveles de pruebas en dependencia de las características del sistema a implementar. A los componentes propuestos se le realizaron las pruebas a nivel de desarrollador para detectar posibles errores en las funcionalidades.

- Prueba de desarrollador.

Es la prueba diseñada e implementada por el equipo de desarrollo. Tradicionalmente estas pruebas han sido consideradas solo para la prueba de unidad, aunque en la actualidad en algunos casos pueden ejecutar pruebas de integración. Se recomienda que estas pruebas cubran más que las pruebas de unidad, debido a la importancia de encontrar posibles errores o respuestas inesperadas de las funcionalidades implementadas. El desarrollador tiene una labor indispensable en las mismas, proporcionando una validación a los componentes implementados. [35]

Pruebas de unidad

Las pruebas de unidad son la primera fase de las pruebas que se le aplican a cada módulo de un software de manera independiente. Su objetivo es verificar que el módulo, entendido como una unidad funcional de un programa independiente, está correctamente codificado.

Es la escala más pequeña de las pruebas, está basada en la funcionalidad de los módulos del programa, como funciones, procedimientos y módulos de clases. [36]

Es aplicable a componentes sentados en el modelo de implementación para verificar que los flujos de control y de datos están cubiertos y que ellos funcionen como se espera. Los casos de pruebas que se diseñen para este nivel de pruebas, deben descubrir errores como:

- Comparaciones entre tipos de datos distintos.
- Operadores lógicos o procedencia incorrecta.
- Igualdad esperada cuando los errores de precisión la hacen poco probable.
- Variables o comparaciones incorrectas.
- Fallo de salida cuando se encuentra una iteración divergente.

Las pruebas fueron realizadas bajo un enfoque ágil, donde a la aplicación se le realizan pruebas durante todo el desarrollo sin documentarse, se acudió al trabajo en equipo, donde el cliente forma parte del mismo y con el firebug[3] de Mozilla Firefox que brinda funcionalidades como debug para comparar variables y ver errores en la asignación de las variables.

Se realizan también, pruebas para mejorar la estructura del código, previendo el funcionamiento correcto del código. Con la aplicación de estas pruebas los errores van a estar más acotados y son más fáciles de desenmascarar.

3.3- Prueba de caja blanca

Las pruebas de caja blanca (también conocidas como pruebas de caja de cristal o pruebas estructurales) se centran en los detalles procedimentales del software, por lo que su diseño está fuertemente ligado al código fuente. El probador escoge distintos valores de entrada para examinar cada uno de los posibles flujos de ejecución del programa y cerciorarse de que se devuelven los valores de salida adecuados.

Al estar basadas en una implementación concreta, si ésta se modifica, por regla general las pruebas también deberán rediseñarse.

Aunque las pruebas de caja blanca son aplicables a varios niveles —unidad, integración y sistema—, habitualmente se aplican a las unidades de software. Su cometido es comprobar los flujos de ejecución

[3] Extensión de Firefox creada y diseñada especialmente para desarrolladores y programadores web. Es un paquete de utilidades con el que se puede analizar (revisar velocidad de carga, estructura DOM), editar, monitorizar y depurar el código fuente, CSS, HTML y JavaScript de una página web.

dentro de cada unidad (función, clase, módulo, etc.), pero, también pueden testear los flujos entre unidades durante la integración e incluso entre subsistemas durante las pruebas de sistema.

A pesar de que este enfoque permite diseñar pruebas que cubran una amplia variedad de casos de prueba, podría pasar por alto partes incompletas de la especificación o requisitos faltantes, pese a garantizar la prueba exhaustiva de todos los flujos de ejecución del código analizado. [36]

Dentro de la prueba de caja blanca se incluyen las técnicas de pruebas que serán descritas a continuación:

Prueba del camino básico: permite obtener una medida de la complejidad lógica de un diseño y usar la misma como guía para la definición de un conjunto de caminos básicos. Los casos de prueba obtenidos garantizan que durante la prueba se ejecute al menos una vez cada sentencia del programa.

Prueba de condición: ejercita las condiciones lógicas contenidas en el módulo de un programa. Garantiza la ejecución por lo menos una vez de todos los caminos independientes de cada módulo, programa o método.

Prueba de flujo de datos: se seleccionan caminos de prueba de un programa de acuerdo con la ubicación de las definiciones y los usos de las variables del programa. Garantiza que se ejerciten las estructuras internas de datos para asegurar su validez.

Prueba de bucles: método de prueba que se centra exclusivamente en la validez de las construcciones de bucles. Garantiza la ejecución de todos los bucles en sus límites operacionales. [36]

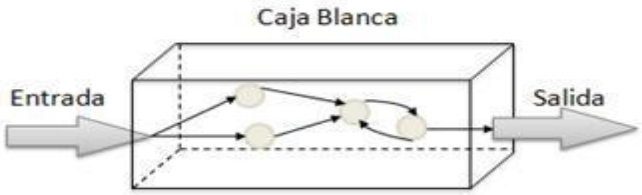


Figura 14. Modelación de la prueba de Caja Blanca.

Según la descripción de la prueba de caja blanca presentada con anterioridad y a partir de la necesidad de crear un producto de alta calidad es preciso valorar qué tan certera ha sido la implementación de los componentes y para ello es necesario aplicar una de las técnicas que esta comprende, en este caso la del camino básico. Para ello es necesario conocer el número de caminos independientes de un determinado

algoritmo mediante el cálculo de la complejidad ciclomática. Se debe comenzar por un análisis del código, posteriormente son enumeradas cada una de las instrucciones, se construye el grafo de flujo asociado y según las fórmulas pertinentes se calcula dicha complejidad:

A continuación se analizan y enumeran las sentencias de código de uno de los procedimientos contenidos en la clase vuGrid, específicamente: fnGetSelected.

```
fnGetSelected:function ( oTableLocal ){
        var aReturn = new Array();
        var aTrs = oTableLocal.fnGetNodes(); //1

        for ( var i=0 ; i<aTrs.length ; i++ ) //2
        {
            if ( $(aTrs[i]).hasClass('row_selected') ) //3
            {
                aReturn.push( aTrs[i] ); //4
            }
        }
        return aReturn; //5
}, //6
```

Figura 15. Segmento de código de la función fnGetSelected (clase vuGrid).

Después de este paso, es necesario representar los grafos de flujo asociados a los códigos antes presentados a través de nodos, aristas y regiones, en ese caso:

Nodo: círculos representados en el grafo de flujo, el cual representa una o más secuencias del procedimiento, un nodo en sí puede representar un proceso, una secuencia de procesos o una sentencia de decisión. Los nodos que no están asociados se utilizan al inicio y final del grafo.

Aristas: saetas a través de las cuales se unen los Nodos y constituyen el flujo de control del procedimiento.

Regiones: las regiones son las áreas delimitadas por las aristas y nodos.

Representación del grafo para el segmento de código vuGrid:

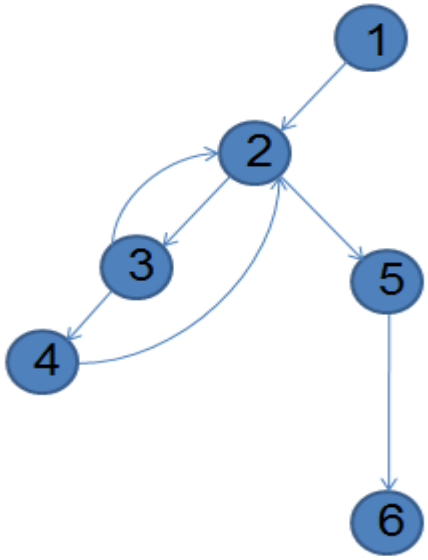


Figura 16. Diagrama de flujo asociado a la función fnGetSelected (clase vuGrid).

Una vez construido el grafo de flujo asociado al procedimiento anterior se determina la complejidad ciclomática, el cálculo es necesario efectuarlo mediante tres vías o fórmulas de manera tal que quede justificado el resultado, siendo el mismo en cada caso.

“A” la cantidad total de aristas y “N” la cantidad total de nodos.

“P” la cantidad total de nodos predicados (son los nodos de los cuales parten dos o más aristas).

“R” la cantidad total de regiones, se incluye el área exterior del grafo, contando como una región más.

Cálculo del resultado para la función fnGetSelected de la clase vuGrid:

1. V (G) = (A – N) + 2

   V (G) = (7 - 6) + 2

   V (G) = 3

2. V (G) = P + 1

   V (G) = 2 + 1

   V (G) = 3

3. V (G) = R

V (G) = 3

El cálculo efectuado mediante las fórmulas antes presentadas muestran una complejidad ciclomática de valor 3, de manera que existen tres caminos básicos para el código tomado del vuGrid, estos valores representan el número mínimo de casos de pruebas para el procedimiento tratado. Seguidamente es necesario especificar los caminos básicos que puede tomar el algoritmo durante su ejecución. En estas representaciones se subrayan los elementos de cada camino que los hacen independientes a los demás.

Tabla 10. Caminos básicos del segmento de código del vuGrid.

Nros.	Caminos básicos
1	1-2-3-4-2-5-6
2	1-2-5-6
3	1-2-3-2-5-6

3.4- Prueba de caja negra

¿Qué son las pruebas de caja negra? Las pruebas de caja negra son, ni más ni menos que, pruebas funcionales dedicadas a “mirar” en el exterior de lo que se prueba. Se denominan de varias formas, pruebas de caja “opaca”, pruebas de entrada/salida, pruebas inducidas por datos…los sinónimos son muchos y muy variados.

Estas pruebas permiten obtener un conjunto de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa. En ellas se ignora la estructura de control, concentrándose en los requisitos funcionales del sistema y ejercitándolos.

La prueba de Caja Negra no es una alternativa a las técnicas de prueba de la Caja Blanca, sino un enfoque complementario que intenta descubrir diferentes tipos de errores a los encontrados en los métodos de la Caja Blanca. Muchos autores consideran que estas pruebas permiten encontrar:

- Funciones incorrectas o ausentes.
- Errores de interfaz.

- Errores en estructuras de datos o en accesos a las Bases de Datos externas.
- Errores de rendimiento.
- Errores de inicialización y terminación.

Para preparar los casos de pruebas hacen falta un número de datos que ayuden a la ejecución de los estos casos y que permitan que el sistema se ejecute en todas sus variantes, pueden ser datos válidos o inválidos para el programa según si lo que se desea es hallar un error o probar una funcionalidad. Los datos se escogen atendiendo a las especificaciones del problema, sin importar los detalles internos del programa, a fin de verificar que el programa corra bien.

Para desarrollar la prueba de caja negra existen varias técnicas, entre ellas están:

- Partición de Equivalencia: esta técnica divide el campo de entrada en clases de datos que tienden a ejercitar determinadas funciones del software.
- Análisis de Valores Límites: esta técnica prueba la habilidad del programa para manejar datos que se encuentran en los límites aceptables.
- Grafos de Causa-Efecto: es una técnica que permite al encargado de la prueba validar complejos conjuntos de acciones y condiciones.
- Pruebas de Comparación: está técnica ejecuta versiones independientes de una aplicación con las mismas especificaciones y hace una comparación en tiempo real de los resultados [36]

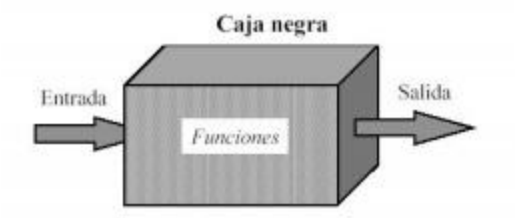


Figura 17. Modelación de la prueba de Caja Negra.

3.4.1- Prueba de aceptación

Las pruebas de aceptación conducidas por el cliente verifican que el sistema satisface los requerimientos y son similares a las pruebas de sistema, pues se basan fundamentalmente en pruebas de funcionalidad. A la hora de realizar estas pruebas, el producto está listo para implantarse en el entorno del cliente. El

usuario debe ser el que realice las pruebas, ayudado por personas del equipo de pruebas, siendo deseable, que sea el mismo usuario quien aporte los casos de prueba.

Estas pruebas se caracterizan por:

- Participación activa del usuario, que debe ejecutar los casos de prueba ayudado por miembros del equipo de pruebas.
- Están enfocadas a probar los requisitos de usuario, o mejor dicho a demostrar que no se cumplen los requisitos, los criterios de aceptación o el contrato. Si no se consigue demostrar esto el cliente deberá aceptar el producto.
- Corresponden a la fase final del proceso de desarrollo de software.

Es muy recomendable que las pruebas de aceptación se realicen en el entorno en que se va a explotar el sistema (incluido el personal que lo maneje). En caso de un producto de interés general, se realizan pruebas con varios usuarios que reportarán sus valoraciones sobre el producto. [36]

Para la generación de casos de prueba de aceptación se utilizó la técnica de caja negra prueba de comparación.

En el sub-epígrafe siguiente se muestran los resultados de la prueba de aceptación aplicada al sistema.

3.4.2- Análisis de la implementación de las interfaces utilizando los componentes visuales.

En la siguiente tabla se muestra un conjunto de datos referentes a la cantidad de líneas de código y tiempos de creación de instancias de los componentes presentados, en comparación con requerimientos visuales desarrollados anteriormente sin el uso de estos:

Tabla 11. Comparación entre interfaces creadas anteriormente y las implementadas utilizando componentes.

Requerimientos	Desarrollo Anterior/ Componentes	Líneas de Código	Tiempos de desarrollo
Cargar los datos de la tabla TC_PAIS.	Interfaz Anterior	35	30min
	vuSelect	6	5min
Cargar los datos de la tabla TC_PROVINCIA, y luego de	Interfaz Anterior	74	45min

haberse seleccionado una provincia cargar los datos correspondientes de la tabla TC_MUNICIPIO.	vuSelect	14	7min
Filtrar los documentos que hayan sido creados en un rango de fechas.	Interfaz Anterior	40	35min
	vuDatePicker	4	2min
Adicionar usuario.	Interfaz Anterior	169	120min
	vuTextAjaxValidator	120	38min
Obtener entidades a partir de un criterio.	Interfaz Anterior	12	15min
	vuTypeahead	4	2min
Completar registro de usuario.	Interfaz Anterior	420	240min
	vuDatePicker	380	206min
Completar registro de usuario.	Interfaz Anterior	420	240min
	vuMessageDialog, vuMessageNotification	323	180min
Adicionar usuario interno.	Interfaz Anterior	125	127min
	vuProgressBar	98	112min
Solicitud Realización Reconocimiento Físico.	Interfaz Anterior	1	1min
	vuTimePicker	1	1min
Listar usuarios.	Interfaz Anterior	156	65min
	vuGrid	34	12min
Asociar documentos a la Declaración de Mercancías (DM).	Interfaz Anterior	185	91min
	vuGridToGrid	57	20min
Adicionar usuario interno.	Interfaz Anterior	125	127min
	vuForm	52	23min
Gestionar funcionalidades.	Interfaz Anterior	228	165min
	vuFormToGrid	153	48min

Como se puede apreciar en la tabla antes expuesta, el desarrollo de los requerimientos de interfaz basado en los componentes presentados, agiliza considerablemente la implementación de las interfaces estandarizando tanto las funcionalidades como la forma de visualización. Además disminuye el tamaño del código propiciando la velocidad de carga de los diferentes módulos para la capa de presentación.

Conclusiones parciales

La calidad de los componentes desarrollados fue el elemento clave del capítulo que recién concluye. Motivos por los cuales se efectuaron pruebas de software en el nivel de unidad mediante casos de pruebas. Teniendo en cuenta las entradas, las salidas y los resultados esperados, se aplicó el tratamiento de errores en casos de anomalías.

Concluyendo así que el resultado del desarrollo de los componentes efectuados durante todo el presente trabajo de diploma fue satisfactorio y que se cumplieron los objetivos trazados a partir de los requerimientos capturados durante el análisis de los mismos.

CONCLUSIONES GENERALES

Una vez terminado el presente Trabajo de Diploma se puede concluir que se desarrollaron todas las tareas a fin de cumplir los objetivos propuestos, resaltando que:

- El estudio realizado en el marco teórico permitió un mayor conocimiento en el desarrollo de componentes. Además sentó las bases para la implementación de estos a través de los lenguajes, tecnologías y herramientas usadas, dando así respuesta a la problemática planteada.
- Luego del análisis de las interfaces de la VUCEC, se identificaron varias características y comportamientos comunes, traduciéndose estos en 14 componentes.
- Se presentó la descripción de los componentes visuales, así como sus atributos, métodos y eventos, para ofrecer una imagen completa de estos. Mediante estas descripciones quedó listo el escenario para el modelo conceptual y la captura de los requisitos que fueron implementados.
- La utilización de las técnicas para la captura de los requisitos funcionales, permitieron identificar los mismos, que de conjunto a la aplicación de las etapas de la Ingeniería de Requisitos responden a las necesidades reales de las interfaces de usuarios.
- Se obtuvo el diseño de los componentes correspondientes, contribuyendo a una arquitectura sólida y estable que se convierte en un plano para la próxima fase.
- Se implementaron los componentes siguiendo un estándar de codificación definido por el proyecto, obteniendo como producto final un paquete de componentes reutilizables que reducen el tiempo de desarrollo y la cantidad de líneas de código.
- Se evaluó la factibilidad de los componentes a través de pruebas de software efectuadas para el nivel de unidad, las cuales arrojaron resultados favorables posibilitando dar cumplimiento a los objetivos trazados en el presente trabajo.

RECOMENDACIONES

Después de realizadas las conclusiones de este trabajo se recomienda:

- Realizar mejoras a los componentes en función de las nuevas versiones de los lenguajes y tecnologías empleadas.
- Incluir el paquete de componentes de software en un repositorio de componentes del centro CEIGE con el objetivo de garantizar su posterior utilización en otros proyectos y que sean optimizados por la comunidad de desarrolladores de la universidad.
- Se recomienda la constante actualización de los componentes para futuras transformaciones en la medida en que las necesidades de los clientes puedan ser modificadas.

Referencias Bibliográficas

1. **Ariza Rojas Maribel, Molina Garcia Juan Carlos.** Introducción y principios básicos del Desarrollo de Software Basado en Componentes. 2004.

2. **Terreros Casal Julio.** Desarrollo de Software Basado en Componentes. [En línea] [Citado el: 8 de Diciembre de 2013.] http://msdn.microsoft.com/es-es/library/bb972268.aspx.

3. **Martínez Luis, F. Iribarne.** Modelo de medición para el Desarrollo de Software Basado en Componentes COTS. s.l. : Universidad de Almería: s.n., 2003.

4. **Clemen, Szyperski.** Component Software. Beyond Object-Oriented Programming. 1998.

5. **CEIGE.** Modelo de Desarrollo de Software. 2012.

6. **William, Obregón González Ing.** Modelo de Desarrollo de Software. 2012.

7. **Dewayne E Perry, Alexander L. Wolf.** Foundations for the Study of Software. s.l. : ACM SIGSOFT Software Engineering Notes, October 1992.

8. **Garlan David, Shaw Mary.** An Introduction to Software Arquitecture. January 1994.

9. **L Bass, P Clements, R Kazman.** Software architecture in practice. 2003.

10. **Presss, IEEE.** Recommended Pracice for Software Architecture Description of Software Intensive Systems. s.l. : IEEE., 2000.

11. **Asael, Calderon.** Programación por capas. 10 de marzo de 2013.

12. **López Estívariz José Félix, Alameda Arellano Javier.** Diseño de Software. [En línea] 12 de Diciembre de 2013. http://www.issi.uned.es/grado/swdes/index.htm.

13. **Craig, Larman.** UML y Patrones. Introducción al análisis y diseño orientado a objetos. 2004.

14. Lenguajes de Programación. [En línea] 12 de Diciembre de 2013. http://www.lenguajes-de-programacion.com/software.shtml.

15. **Vega John Freddy, Van Der Henst.** Guía HTML5. 2011.

16. **Javier, Pérez Eguíluz.** Introducción a CSS.

17. Network Developer Mozilla. [En línea] 17 de Diciembre de 2013. https://developer.mozilla.org/en-US/docs/Web/JavaScript.

18. Libros Web. [En línea] 13 de Enero de 2014. http://librosweb.es/javascript/.

19. Libros Web. [En línea] 13 de Enero de 2014. http://librosweb.es/ajax/.

20. **Riehle, Dirk.** Framework Design: ARole Modeling Approach. 2000.

21. JQuery. [En línea] 26 de Enero de 2014. http://jquery.com/.

22. **Beck, Kent.** Extreme Programming Explained: Embrace Change. . 2004.

23. JQuery. . [En línea] 29 de Enero de 2014. http://blog.jquery.com/category/jquery-ui/.

24. Bootstrap. . [En línea] 2 de Febrero de 2014. http://www.bootstrapworld.org/.

25. Bootstrap, framework de twitter. [En línea] 2 de Febrero de 2014. http://www.genbetadev.com/frameworks/bootstrap..

26. Netbeans Ide. . [En línea] 7 de Febrero de 2014. https://netbeans.org/index_es.html.

27. Visual Paradign. . [En línea] 7 de Febrero de 2014. http://www.software.com.ar/visual-paradigm-para-uml.html.

28. **Calquín Fabián .** Técnicas de recolección de datos. . [En línea] 15 de Marzo de 2014. http://es.scribd.com/doc/221061797/Modelo-Conceptual.

29. **Gracia Luis Miguel .** Técnicas para la Captura de Requisitos. [En línea] 21 de Marzo de 2014. http://unpocodejava.wordpress.com/tecnicas-para-la-captura-de-requisitos/ .

30. **Sommerville, Ian.** Software Engineering. ADDISON-WESLEY.

31. **Elisa., Lic. Arizaca.** Universidad Salesiana de Bolivia. Analisis y diseño de sistemas II. 12 de Noviembre de 2009.

32. Métrica_de_diseño. Metricas de Diseño. [En línea] 17 de Abril de 2014. http://www.ecured.cu/index.php/.

33. **López Aury, Quijada Carmen, Medina Laura, Monsalve Laura, Milano Rodolfo .** Universidad Yacambu. Gerencia Mencion: Sistemas de Informacion. . [En línea] http://www.oocities.org/es/monsalvelaura/fase2/analisis.html.

34. **Barrientos, Pablo Andrés.** Enfoque para pruebas de unidad basado en la generación aleatoria de objetos. 2006.

35. Pruebas de desarrollador . [En línea] 24 de Mayo de 2014. http://www.pmoinformatica.com/2012/12/test-driven-development-tdd-pruebas-de.html.

36. **PRESSMAN, Roger S.** Ingeniería del Software Un enfoque práctico. s.l. : 5ta. ed. La Habana: Félix Varela, ISBN: 970-105-473-3, 2005.