Universidad de las Ciencias Informáticas

Facultad 3

Título: Módulos Clasificadores y Variedad-Establecimiento para la Gestión del Índice de Precio de la Construcción en la Oficina Nacional de Estadística e Información

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

Autores:

Yunisleidis Montenegro Mejías

Luis Joel Cruz Castro

Tutores:

MSc. Yordanis García Leiva

MSc. Sandy Suárez Jiménez

Septiembre, 2020

"Año 62 del Triunfo de la Revolución"

DECLARACIÓN DE AUTORÍA

Declaramos que somos los únicos autores de este trabajo y autorizamos a la Facultad 3 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

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

_____ _____

Yunisleidis Montenegro Mejías Luis Joel Cruz Castro

_____ _____

MSc. Yordanis García Leiva MSc. Sandy Suárez Jiménez

AGRADECIMIENTOS

Yunisleidis

Hoy se cumple uno de mis sueños por fin después de tantos años de sacrificio y de estudio puedo decir ya soy ingeniera en estos momentos son tantas las personas que llegan a mispensamientos a los cuales de una forma u otra quisiera agradecerles

Primeramente, a mi mamá y a mi papá por estar ahí para mí en cada momento, por darme la vida por no cansarse cuando el camino se tornaba oscuro y seguir luchando junto a mí, por los consejos por los buenos y malos momentos por ser siempre la mayor razón de sus vidas por esto y por muchas otras cosas gracias los amo. A mi hermana por estar siempre ahí para mí, por ser siempre mi orgullo y mi motor que me impulsa a seguir cada día te quiero mucho. A mis abuelos paternos que, aunque no están físicamente junto a mi yo sé que donde quieran q estén están muy orgullosos de mí en estos momentos, a mis abuelos maternos por estar siempre ahí para mí y apoyarme en cada momento, A mi novio por todas las noches de desvelo junto a mí para que este sueño se realizara por eso y muchas cosasgracias.

A Vivian que ha sido como una segunda madre para mí en todos estos años muchas gracias.

A mis tutores los cuales siempre estuvieron en cada momento tanto buenos como malos por su dedicación, por los consejos, por las horas de desvelo, por sus exigencias por su guía por su apoyo cuando todo se tornaba difícil por su paciencia son una parte muy importante de este logro por todo muchas gracias

Al profe Abel Velásquez que fue una gran guía en para mí en esta universidad, a la profe Dariela por los consejos y toda su paciencia.

También quisiera agradecerle a ese grupo de amigos que me los llevo en el corazón, aunque muchos ya no están siguen aquí como Román, Oswar, Aleixi, Zule que siempre estuvieron en los malos y buenos momentos, a mi jirafa por todo lo compartido por las noches de apuros eres una gran amiga, A mi querido yandry que más que un gran amigo fuiste mi confidente en todo momento cuando más te necesite, a mis compañeras de apartamento por todas las cosas buenas y malas.

Al Batí, Bárbaro, Pang y demás por los buenos momentos.

A todos y cada uno de los que una forma y otra estuvieron junto a mí a lo largo de mi carrera muchas gracias

De Luis Joel:

Difícil tarea encontrar palabras para expresar el profundo agradecimiento que siento por todos aquellos que de cierta forma han aportado su granito de arena para ver hoy realizado mi sueño de ser Ingeniero de Ciencias Informáticas.

Gracias a los tutores Yordanis y Sandy por la confianza depositada, especialmente en los momentos difíciles, para sacar adelante la tesis, por los conocimientos, enseñanzas y experiencias para hacerla de la mejor manera posible. Al claustro de profesores de la facultad 3 que integran el tribunal que con las críticas bien recibidas en los seminarios de tesis ayudaron a darle forma.

Gracias a los compañeros de aula (Wendy, Yuni por incluirme en el equipo para los seminarios) y a los profesores que me formaron, los tendré siempre presentes.

Gracias a mis ¡Berraquitos!(Camilo Bot, Carlitos el Isla, Julito, el Dani, Ernesucristo, Mejías, el Rafa, Boris, Chino Metal, Chris) los que todavía están y los que se fueron, esos amigos que vinieron para quedarse, a los que podía molestar a las 3 de la madrugada para lo que fuera, con los que compartí momentos increíbles. Personas de distintas provincias que quizás nunca hubiese conocido, con los que forme tremendo piquete, gracias a esta familia que me dio esta linda universidad que es la UCI.

Gracias Randy por ser ese hermano que me dio la vida. ¡Nunca cambies Bro!

Gracias a mi familia por el apoyo incondicional. Gracias a la vida por permitirme celebrar este momento con mi bisabuela (de 101 años) la Juanuca, por preguntarme siempre como me va en los estudios. A mi abuela Georgina por preocuparse y ocuparse en todo lo que puede. A mi prima Isabella y a mis tíos (debería ponerlo con doble 's' para que cupieran todos) por tenerme presente en sus pensamientos a pesar de las distancias. A mi abuela Yayi, por ser esa abuela de cuentos de hadas que todos desean tener y más. A mi hermanita Sheyla mi mejor amiga, siempre a mi lado, te amo. A mis padres, Luis Antonio mi ejemplo, espero significar para alguien lo que tú para mí y Sandra la sangre en mis venas, sin duda las personas más importantes en mi vida los quiero con el alma.

DEDICATORIA

De Yunisleidis

Dedico este trabajo de diploma a mis dos grandes ejemplos a seguir, mis padres que lo han dado todo porque yo llegara a ser quien soy los quiero.

De Luis Joel:

Este trabajo de diploma está dedicado a mis padres.¡Gracias por todo!

RESUMEN

Los índices de precios constituyen una medida estadística que se calcula sobre los precios de los productos de consumo masivo en un determinado período. La Oficina Nacional de Estadística de la República de Cuba,organismo rector en el procesamiento de las estadísticas del país tiene entre sus objetivos la gestión de índices de precio. En el caso de la gestión de los índices de precio de la construcción la Oficina cuenta con una aplicación informática de escritorio con base de dato local, que dificulta la interconexión entre cada una de las sedes a nivel de país, afectandola actualización constante de la información que se procesa. Además, el sistema está implementado en tecnologías privativas que no se corresponden con la política de soberanía tecnológica que aplica el país.La presente investigación tiene como objetivo desarrollar los módulos Clasificadores y Variedad-Establecimiento para la Gestión del Índice de Precio la Construcción, de forma tal que se contribuya a la obtención de un software que soluciones las limitantes antes descritas. El desarrollo de la solución está guiado por el uso de la metodología de desarrollo de software Proceso Unificado Ágil en su variación para la Universidad de Ciencias Informáticas y la utilización de tecnologías de código abierto. El resultado de la investigación es validado aplicando pruebas de software.

PALABRASCLAVES: clasificadores, construcción, establecimiento, índice, precio, variedad

INDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

La Oficina Nacional de Estadística e Información (ONEI), fue creada a partir de lo definido en el artículo 31 del Decreto Ley No. 281 del 2 de febrero de 2011, como resultado de la organización del Sistema de Información del Gobierno. La ONEI tiene entre sus objetivos avanzar en las mediciones en los ámbitos económico, social, demográfico y medioambiental. Para el caso del análisis de temas económicos, la oficina cuenta con el Departamento de Índices de Precio, en el cual se gestionan los siguientes índices:

- Índice de Precios del Consumidor.
- Índice de Precios de la Construcción.
- Índice de Precios de la Producción Mayorista de la Industria Manufacturera.
- Índice de Precios de la Venta y Comercialización Mayorista de la Industria Manufacturera.

En el caso del Índice de Precio de la Construcción, este constituye un conjunto de indicadores que miden el cambio en los precios de los materiales utilizados en la construcción de diferentes tipos de obras. Para la definición de este tipo de índice se requiere de información sobre los insumos utilizados para la construcción y las reparaciones, según tipo de obra. Esta información se obtiene utilizando presupuestos de Obras Residenciales y Obras no Residenciales que detallan los materiales utilizados, así como los precios y cantidades usados de cada uno, llegando a mostrar un importe individual y el valor de cada uno dentro del total de la obra.

La estructura de la canasta para determinar este índice, se clasifica en tres niveles: Grupo, Elemento y Variedad. Estos niveles incluyen en primera instancia los elementos, quienes constituyen la clasificación más pequeña a formar parte en el cálculo del índice y los que contienen las variedades más específicas de cada uno.

Las variedades son desagregaciones operativas que permiten incluir en el índice diferentes presentaciones de un mismo producto. El próximo nivel es el grupo que resulta de la agregación de determinados elementos y posteriormente alcanza su lugar el índice general, que tiene una representación del cien por ciento, completado con cada una de las ponderaciones de los niveles anteriores.

En la actualidad el departamento de índices de la ONEI cuenta con un sistema informático que gestiona el índice de precio de la construcción. Este software constituye una aplicación de escritorio con base de dato local, que dificulta el logro de una interconexión entre cada una de las sedes de la ONEI a nivel de país, que permita mantener una actualización constante de la información que se procesa. Además, el sistema está implementado en las tecnologías Visual Basic y sistema gestor de base de datos Access, las cuales constituyen tecnologías privativas que no se corresponden con la política de soberanía tecnológica que aplica el país.

Otra de las características de esta aplicación es que abarca parcialmente el proceso de gestión del índice de la construcción en la ONEI, debido a que no permite gestionar los datos de los clasificadores, ni establecer relaciones entre las variedades y los establecimientos, este último proceso se denomina empadronamiento.

El sistema actual realiza la carga de los datos de los clasificadores y establecimientos desde una hoja electrónica de cálculo (documentos Excel), por tal motivo una vez que la información es importada, no existe posibilidad de hacer modificaciones, adiciones o empadronamientos. Todo lo anterior trae como consecuencia que cuando existen cambios sobre los datos de un clasificador o de un establecimiento, ejemplo: al crear nuevos empadronamientos, el sistema se ve imposibilitado de asumir los mismos y la información queda desactualizada.

A partir de la problemática antes descrita se genera la necesidad de resolver el siguiente **problema de investigación**:

¿Cómo gestionar los Clasificadores y Variedades-Establecimientos en el proceso de gestión del Índice de Precio de la Construcción de la ONEI, de forma tal que se contribuya a la actualización de la información?

Tomando en cuenta el problema antes propuesto se define como **objeto de estudio:** la gestión de datos relacionados con Clasificadores y Variedades-Establecimiento en la Gestión de Índices de Precios.

Determinándose como **objetivo general:** desarrollar los módulos Clasificadores y Variedad-Establecimiento para la Gestión del Índice de Precio la Construcción en la ONEI, de forma tal que se contribuya a la actualización de la información.

Se desglosan del objetivo general los siguientes**objetivos específicos:**

- Elaborar el marco teórico de la investigación mediante un estudio de los referentes teóricos sobre el desarrollo de soluciones informáticas dirigidas a la gestión de índice de precios.
- Realizar la identificación de los requisitos, análisis, diseño e implementación de los módulos propuestos.
- Validar el diseño y el funcionamiento de los módulos propuestos aplicando métricas y pruebas de software respectivamente.
- Verificar el cumplimiento de la relación causa efecto de la variable independiente sobre las variables dependientes de la investigación.

Para ello se identifica como **campo de acción:** la gestión de datos relacionados con Clasificadores y Variedades-Establecimiento en la Gestión del Índice de Preciode la Construcción.

Definiéndose como **idea a defender*:***Si se desarrollan los módulos Clasificadores y Variedad-Establecimiento para la Gestión del Índice de Preciode la Construcción en la ONEI, se contribuye a la actualización de la información.

Métodos Teóricos:

Histórico-Lógico: permitió realizar un estudio sobre las principales tendencias de la web con respecto a las herramientas existentes, metodologías de desarrollo y la forma en que se brindan determinados servicios en la web, sobre todo los relacionados con la gestión de índices de precios, con el fin de seleccionar las más apropiadas para el desarrollo de los módulos propuestos.

Analítico-Sintético: posibilitó la realización del estudio teórico de la investigación haciendo más sencillo el proceso de análisis de los documentos y la extracción de los elementos fundamentales a tener en cuenta para desarrollar los módulos propuestos.

Modelación: se empleó para la confección de los prototipos funcionales, la representación de los requisitos y el diseño de la base de datos.

Métodos Empíricos:

Observación: se utilizó a través del estudio realizado a diferentes aplicaciones web que existen en países latinoamericanos que gestionan los índices de precio para la

construcción, para determinar los elementos más comunes a tener en cuenta en la realización de los módulos propuestos.

Medición: permitió medir la calidad de la especificación de los requisitos y el grado de ambigüedad de estos, además de obtener una medida de la calidad del diseño para su validación.

Entrevista: fue utilizado para comprender las necesidades del cliente, capturar los requisitosy obtener información que apoye la realización de la investigación.

El contenido de este trabajo consta de tres capítulos, definidos de la siguiente forma:

Capítulo 1: Fundamentación Teórica. En este capítulo se describen conceptos relacionados con la problemática antes mencionada, las herramientas, lenguajes y tecnologías que se emplean en la construcción de la solución, así como aspectos esenciales que sirvan de soporte a la misma. Además, se realiza una descripción de los principales mecanismos de integración y patrones de diseño existentes. Se describe la metodología seleccionada para guiar el proceso de desarrollo de la presente investigación.

Capítulo 2: Descripción y diseño del sistema. En este capítulo se describe la solución propuesta para la informatización de la gestión de índice de precios de la construcción en la ONEI. Se hace referencia a las técnicas empleadas para la captura de los requisitos. Se presentan los requisitos funcionales y no funcionales con los que deben cumplir los módulos propuesto en la presente investigación. Los requisitos funcionales son agrupados en historias de usuario. En el capítulo también se muestran los diagramas de componentes referentes a la arquitectura. Por otra parte, se expone la utilización de patrones de diseño y estándares de codificación.

Capítulo 3: Validación. En el presente capítulo se muestran los elementos que forman parte de la validación de la solución. Se exponen los resultados de la aplicación de métricas para medir la calidad de los módulos propuestos, las relaciones entre clases y el tamaño operacional de las clases del diseño. Por otra parte, se define la estrategia de prueba aplicada a partir de las disciplinas establecidas para la etapa de pruebas por la metodología *AUP* variación UCI.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

En este capítulo se describenconceptos relacionados con la problemática antes mencionada, las herramientas, lenguajes y tecnologías que se emplean en la construcción de la solución, así como aspectos esenciales que sirvan de soporte a la misma. Además, se realiza una descripción de los principales mecanismos de integración y patrones de diseño existentes. Se describe la metodología seleccionada para guiar el proceso de desarrollo de la presente investigación.

1.2 Conceptos fundamentales asociados al dominio del problema

Para una mejor comprensión del tema de investigación, es necesario dominar las siguientes definiciones:

Variedad: define como la variedad espacial de un producto al conjunto de tipos o modelos de éste que una empresa ofrece al mercado en un determinado momento del tiempo. Variedades espaciales se ofrecen para satisfacer las necesidades de diferentes segmentos del mercado(Martin, 1999.).

Índice de precio: el índice de precios es un número índice, una medida estadística que se calcula sobre los precios de los productos de consumo masivo en un determinado período. Vale destacar que el más utilizado es el índice de precios al consumo, el cual mide específicamente la evolución del gasto de una familia tipo(Potes, 2015.).

Clasificadores: un clasificador, es un instrumento normativo que ordena y agrupa los recursos con que se cuenta, en categorías homogéneas definidas en función de la naturaleza y las características de las transacciones que dan origen a cada una de los recursos (Potes, 2015.).

1.3 Tendencias actuales

Con el propósito de obtener experiencia en cuanto a la organización y estructuración para la construcción de los módulos propuestos se estudiaron varios libros, revistas y artículos existentes tanto en el ámbito nacional como extranjero. Este estudio se realizó teniendo en cuenta que en la búsqueda hecha por los autores de la presente investigación no se

encontraron softwares similares a la solución propuesta, que permitan hacer una comparación y analizar sus características. Sin embargo, con la documentación analizada se identificóque el cálculo de índice para los productos de la construcción se realiza de formas distinta e individual en cada país. Entre los documentos estudiados se encuentran:

- "METODOS DE CONSTRUCCION DE INDICES DE PRECIOS DE VIVIENDA" Teoría y experiencia internacional de Julio Escobar Potes y José Vicente Romero. En este documento se analizan las principales técnicas desarrolladas para la construcción de índices de precios de vivienda. Se realiza una aproximación histórica de la evolución de estos métodos, así como un análisis los aspectos metodológicos que plantean. Todo lo anterior con el fin de servir de guía para la construcción del Índice de Precios de la Vivienda Usada (IPVU) para Colombia. Igualmente sirve de referencia para los investigadores interesados en metodologías alternativas para la construcción de índices de precios de productos no homogéneos.
- México –Índice Nacional de precios al consumidor 2018, Investigación de precio para el cálculo INPC. Este documento constituye una implementación metodológica importante atendiendo a las recomendaciones de buenas prácticas del Manual del índice de precios al consumidor: Teoría y práctica que señala, con relación al muestreo probabilístico: "La teoría moderna de muestreo estadístico se centra en el muestreo probabilístico, cuya utilización también se recomienda y se considera práctica estándar para todo tipo de encuestas estadísticas, entre ellas las encuestas económicas." ElInstituto Nacional Estadística y Gestión de la Información tomó la decisión de incorporar el muestreo probabilístico en la selección de los puntos de venta para la cotización de los precios.

Lo anterior fue posible gracias a la disponibilidad de información de los Censos Económicos, para la construcción de un marco de muestreo de unidades económicas o establecimientos. En comparación con otros países, México tiene la ventaja de contar con estos censos, cuyo levantamiento se realiza cada cinco años, así como con el Registro Estadístico de Negocios de México (RENEM), el cual proporciona información actualizada de los establecimientos donde se comercializan los bienes y servicios de consumo nacional.

Después del estudio de los documentos y la obtención de buenas prácticas para la realización del cálculo de índice, no se pudo encontrar un software para comparar y poder evaluar sus características. De esta forma los autores de la presente investigación llegan a la conclusión que es necesario realizar la implementación de los módulos propuestos, pues Cuba no cuenta con un sistema que realice las funcionalidades que se requieren en la Oficina Nacional de Estadística e Información. Además, en Cuba no se pueden utilizar algunas de las variantes empleadas en diferentes países, teniendo en cuenta que no están disponibles. Además, cada sistema es propio para cada país, pues solo son utilizados por ellos solamente y no tienen información pública disponible para ser reutilizada

1.4 Metodología de desarrollo de software

Una metodología de desarrollo de software consiste en múltiples herramientas, modelos y métodos para asistir en el proceso de desarrollo de software. En ella se definen con precisión los artefactos, roles y actividades involucradas, junto con prácticas y técnicas recomendadas, las guías de adaptación de la metodología al proyecto y las guías para uso de herramientas de apoyo (Association of Modern Technologies Professionals, 2019).

En la presente investigación se decide utilizar como metodología de desarrollo de software, una variación del Proceso Unificado Ágil (AUP, por sus siglas en inglés *Agile UnifiedProcess*) definida para los proyectos de la UCI, teniendo en cuenta que es la más acorde con las características del negocio a desarrollar, así como con las habilidades y experiencias del equipo de desarrollo. Además, la solución propuesta responde a uno de los proyectos de la Red de Centros de Desarrollo de Software de la UCI, la cual tiene establecido el uso de esta metodología en todos los proyectos de la Universidad. A continuación, se describen los principales elementos de esta metodología.

1.4.1 Metodología AUP-UCI

La metodología AUP constituye una versión simplificada de la metodología de desarrollo tradicional Proceso Racional Unificado (RUP,por sus siglas en inglés*RationalUnifiedProcess*). AUP describe la forma de desarrollar aplicaciones de software de manera fácil de entender, usando técnicas ágiles y conceptos que aún se mantienen válidos en RUP. Con el objetivo de estandarizar el proceso de desarrollo de software, la dirección de producción de la UCI tiene definida como metodología a utilizar,

una variación de AUP en unión con el modelo CMMI-DEV v1.3[1], denominada AUP-UCI. La misma establece prácticas centradas en el desarrollo de productos y servicios de calidad (Sánchez, 2015).

La metodología AUP propone organizar el proceso de desarrollo de software en cuatro fases (Inicio, Elaboración, Construcción, Transición). En el caso de la adaptación de esta metodología para los proyectos de la UCI se decide mantener la fase de inicio, pero modificando su objetivo, las tres restantes fases se unifican, quedando una sola denominada ejecución y se agrega una fase de cierre (Sánchez, 2015). A continuación, se describen cada una de las fases de la variación AUP para la UCI.

Inicio: durante el inicio del proyecto se llevan a cabo las actividades relacionadas con la planeación de este. En esta fase se realiza un estudio inicial de la organización cliente, obteniéndose información acerca del alcance del proyecto, estimaciones de tiempo, esfuerzo y costo. Todo este estudio permite decidir si se ejecuta o no el proyecto(Sánchez, 2015).

Ejecución: en esta fase se ejecutan las actividades requeridas para desarrollar el software, incluyendo el ajuste de los planes del proyecto. Durante el desarrollo se modela el negocio, obtienen los requisitos, se elabora la arquitectura y el diseño. Además, se implementa el software y se le aplican pruebas. Durante esta fase el producto es transferido al ambiente de los usuarios finales o entregado al cliente. Asimismo, en la transición se capacita a los usuarios finales sobre la utilización del software(Sánchez, 2015).

Cierre: en esta fase se analizan los resultados del proyecto relacionados con su ejecución y se realizan las actividades formales de cierre del proyecto (Sánchez, 2015).

Disciplinas de variación de AUP-UCI

[1]Modelo de Madurez de Capacidades Integrado para Desarrollo (CMMI-DEV) proporciona buenas prácticas para el desarrollo y mantenimiento de productos y servicios.

AUP propone siete disciplinas (Modelo, Implementación, Prueba, Despliegue, Gestión de configuración, Gestión de proyecto y Entorno). Para el ciclo de vida de los proyectos de la UCI se decide utilizar igual número de disciplinas, pero a un nivel más atómico.

En la metodología AUP los flujos de trabajos: modelado de negocio, requisitos y análisis y diseño están unidos en la disciplina modelo. Sin embargo, en la variación AUP para la UCI estos flujos de trabajo se consideran disciplinas independientes, además se mantiene la disciplina implementación. En el caso de las pruebas, estas se desagregan en tres disciplinas: pruebas internas, pruebas de liberación y pruebas de aceptación. Las disciplinas de AUP asociadas a la gestión de proyecto, en la variación UCI se cubren con las áreas de procesos que define CMMI-DEV v1.3 para el nivel 2 (gestión de la configuración, planeación de proyecto y monitoreo y control) (Sánchez, 2015)**.**

Escenarios para la disciplina Requisitos

A partir de que el Modelado de negocio propone tres variantes a utilizar en los proyectos (Caso de Uso del Negocio (CUN), Diagrama de Proceso del Negocio (DPN) y Modelo Conceptual (MC)). Existen tres formas de encapsular los requisitos (Caso de Uso del Sistema (CUS), Historias de Usuarios (HU), Diagrama de Requisitos por Proceso (DRP)), surgen cuatro escenarios para modelar el sistema en los proyectos, manteniendo en dos de ellos el MC, quedando de la siguiente forma:

- **Escenario No 1:** proyectos que modelen el negocio con CUN solo pueden modelar el sistema con CUS. Se aplica a los proyectos que hayan evaluado el negocio a informatizar y como resultado obtengan que puedan modelar una serie de interacciones entre los trabajadores del negocio/actores del sistema (usuario), similar a una llamada y respuesta respectivamente, donde la atención se centra en cómo el usuario va a utilizar el sistema. Es necesario que se tenga claro por el proyecto que los CUN muestran como los procesos son llevados a cabo por personas y los activos de la organización.
- **Escenario No 2:** proyectos que modelen el negocio con MC solo pueden modelar el sistema con CUS. Se aplica a los proyectos que hayan evaluado el negocio a informatizar y como resultado obtengan que no sea necesario incluir las responsabilidades de las personas que ejecutan las actividades, de esta forma modelarían exclusivamente los conceptos fundamentales del negocio. Se recomienda

este escenario para proyectos donde el objetivo primario es la gestión y presentación de información.

- **Escenario No 3:** proyectos que modelen el negocio con DPN solo pueden modelar el sistema con DRP. Se aplica a los proyectos que hayan evaluado el negocio a informatizar y como resultado obtengan un negocio con procesos muy complejos, independientes de las personas que los manejan y ejecutan, proporcionando objetividad, solidez y su continuidad. Se debe tener presente que este escenario es muy conveniente si se desea representar una gran cantidad de niveles de detalles y la relaciones entre los procesos identificados.
- **Escenario No 4:** proyectos que no modelen negocio solo pueden modelar el sistema con historias de usuario (HU). Se aplica a los proyectos que hayan evaluado el negocio a informatizar y como resultado obtengan un negocio muy bien definido. El cliente estará siempre acompañando al equipo de desarrollo para convenir los detalles de los requisitos y así poder implementarlos, probarlos y validarlos. Se recomienda en proyectos no muy extensos, ya que una HU no debe poseer demasiada información (Sánchez, 2015).

La aplicación de la variación de AUP en la UCI permite estandarizar el proceso de desarrollo de software en la universidad, dando cumplimiento a las buenas prácticas que define CMMI-DEV v1.3. Estas disciplinas se desarrollan en la fase de ejecución y pueden estar organizadas en iteraciones, con el propósito de obtener resultados incrementales.

En la presente investigación se decide utilizar el escenario No.4, ya que aplica a los proyectos que hayan evaluado el negocio a informatizar y como resultado obtengan un negocio muy bien definido, estas características se adaptan a la solución propuesta.

1.5 Herramientas de Ingeniería del Software Asistidas por Computadoras

Las herramientas *CASE* (por sus siglas en inglés *ComputerAided Software Engineering)* están destinadas a aumentar la productividad en el desarrollo del software reduciendo el coste del mismo en términos de tiempo y de dinero. Estas herramientas pueden ayudar en todos los aspectos del ciclo de vida de desarrollo del software en tareas como el diseño de proyectos, cálculo de costos, implementación de parte del código a partir del diseño dado, compilación automática y documentación o detección de errores (Visual-Paradigm, 2019). A continuación, se describe la herramienta *CASE* utilizada en las tareas ingenieriles de la presente investigación.

1.5.1 Visual Paradigm

Es una herramienta queutiliza UML(por sus siglas en inglés*UnifiedModelingLanguage*) quesoportael ciclo devidacompletodel desarrollo desoftware:análisisydiseño orientadosa objetos,construcción,pruebasydespliegue. Permitemodelar diagramas de clases, generar código desde diagramas y generar documentación. La herramienta agiliza la construcción de aplicaciones con calidad y a un menor coste de tiempo. Posibilita la generación de bases de datos, transformación de diagramas de Entidad-Relación en tablas de base de datos, así como obtener ingeniería inversa de bases de datos (Visual-Paradigm, 2019).
A partir de los elementos antes expuestos los autores de la presente tesis deciden utilizar Visual Paradigm teniendo en cuenta que esta herramienta propicia un conjunto de funcionalidades para el desarrollo de programas informáticos, desde la planificación, pasando por el análisis y el diseño, hasta la generación del código fuente de los programas y la documentación. Además, la UCI cuenta con una licencia para su uso.

1.6 Lenguajes de programación

La implementación de las clases y los métodos serán representados a través de lenguajes de programación. Es importante tener en cuenta que de los lenguajes de programación que existen para desarrollar aplicaciones orientadas a la web se encuentran dos grupos fundamentales de acuerdo con la arquitectura Cliente/Servidor, la programación del lado del servidor y la programación del lado del cliente. Esta última incluye aquellos lenguajes que son interpretados por una aplicación cliente como el navegador web, entre ellos se encuentra HTML[2].

Los lenguajes de programación del lado servidor son reconocidos, ejecutados e interpretados por el propio servidor, el que se encarga de brindar información al cliente en un formato comprensible para él. Para el desarrollo de la solución propuesta se hace necesario emplear los lenguajes Java y JavaScript, teniendo en cuenta que forman parte de las tecnologías definidas en CEGEL para el desarrollo de las aplicaciones.

[2]*Hyper Text Markup Lenguage*

1.6.1 Java

Java es un lenguaje de programación que se caracteriza por ser orientado a objetos, distribuido y dinámico, robusto, seguro, multitarea y portable. Su código es similar al lenguaje C y C++ con un modelo de objetos más sencillo (Pérez, 2018).

Java tiene la característica de ser al mismo tiempo compilado e interpretado. El compilador es el encargado de convertir el código fuente de un programa en un código intermedio llamado *bytecode* que es independiente de la plataforma en que se trabaje y es ejecutado por el intérprete de Java que forma parte de la máquina virtual de Java (Zamitiz, 2016). El lenguaje antes descrito se utiliza en el desarrollo de los módulos propuestos.

1.6.2 JavaScript

Es un lenguaje de programación ligero, interpretado por la mayoría de los navegadores y que les proporciona a las páginas web, efectos y funciones complementarias a las consideradas como estándar HTML. Este tipo de lenguaje de programación es de código abierto, por lo que cualquier persona puede utilizarlo sin comprar una licencia. Con frecuencia es empleado en los sitios web, para realizar acciones en el lado del cliente, estando centrado en el código fuente de la página web (Venemedia, 2014).

1.6.3 GraphQL

GraphQL es un lenguaje de consulta y un tiempo de ejecución del servidor para las interfaces de programación de aplicaciones (API); su función es brindar a los clientes exactamente los datos que solicitan y nada más. (Rodríguez, 2017).

Con GraphQL, las API son rápidas, flexibles y sencillas para los desarrolladores. Como alternativa a REST, GraphQL permite que los desarrolladores creen consultas para extraer datos de varias fuentes en una sola llamada a la API. (Rodríguez, 2017).

Además, GraphQL otorga a los encargados del mantenimiento de las API la flexibilidad para agregar campos o modificarlos, sin que esto afecte las consultas actuales. Los desarrolladores pueden diseñar las API con los métodos que prefieran, y la especificación

de GraphQL garantizará que funcionen de forma predecible para los clientes. (Rodríguez, 2017).

1.7 Entorno de Desarrollo Integrado

La implementación de los módulos propuestos, a partir de los lenguajes de programación antes descritos se complementa con el uso de entornos de desarrollo integrado (IDE, por sus siglas en inglés*IntegratedDevelopmentEnvironment*). Los IDE constituyen un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica. Los IDE proveen un marco de trabajo amigable para la mayoría de los lenguajes de programación (Martin Olivera, y otros, 2016).

1.7.1 IntelliJ IDEA 2018.2.4

Este IDE es soportado [3]por varios sistemas operativos: Windows, Linux o Mac OS. Su versión gratuita soporta lenguajes como: Java, Groovy, XML/XSL, Kotlin, Python, Clojure, Dart, Erlang, Go, Haxe, Perl, Scala, Haskell, Lua, estos últimos vía plugin[4]. Por otra parte su versión de pago soportar todos los lenguajes antes mencionados más JavaScript, HTML, CSS, SQL, Ruby, así como PHP vía plugin**(**JetBrains, 2018).

1.7.2 WebStorm 2018.2.4

Este es un entorno de desarrollo inteligente que ayuda a producir código de alta calidad de manera eficiente gracias al completamiento de código, se detectan errores sobre la marcha, la navegación y refactorizaciones automatizadas. Da soporte a las recientes tecnologías, funcionando de la mejor manera con las más modernas y populares para el desarrollo web, así como *AngularJS, ECMAScript 6 y Compass*. Es multiplataforma, funciona en Windows, Mac OS o Linux con una única clave de licencia. Además, WebStorm agiliza el flujo de trabajo mediante la integración con todo lo necesario para el desarrollo productivo. Desde el IDE se pueden utilizar varias herramientas como el depurador y terminales (Jetbrains, 2018).

[4]Es una aplicación que se relaciona con otra para agregarle una función nueva y generalmente muy específica.

1.8 Sistema Gestor de Bases de Datos

Un Sistema Gestor de Base de Datos (SGBD) es un sistema de software que permite la definición de bases de datos; así como la elección de las estructuras de datos necesarios para el almacenamiento y búsqueda de los datos, ya sea de forma interactiva o a través de un lenguaje de programación. Un SGBD relacional es un modelo de datos que facilita a los usuarios describir los datos que serán almacenados en la base de datos junto con un grupo de operaciones para manejar los datos (PETKOVIĆ, 2005).

Para el desarrollo de la solución propuesta se hace necesario utilizar el sistema gestor de base de datos PostgreSQL, teniendo en cuenta la compatibilidad de estos con los sistemas desarrollados en CEGEL. PostgreSQL se emplea en la administración de todos los datos del sistema y también los relacionados con la gestión de usuarios, roles y permisos.

1.8.1 PostgreSQL

Es un sistema de base de datos objeto-relacional de código abierto. Cuenta con más de 15 años de desarrollo activo y una arquitectura probada. Se ejecuta en los principales sistemas operativos que existen en la actualidad como Linux, UNIX (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64) y Windows (Microbuffer, 2011). Además, constituye un sistema de gestión de bases de datos objeto-relacional basado en Postgres. Dentro de sus principales ventajas se encuentran:

- Soporte de protocolo de comunicación encriptado por SSL5.
- Extensiones para alta disponibilidad, nuevos tipos de índices, datos espaciales y minería de datos.
- Permite la gestión de diferentes usuarios, como también los permisos asignados a cada uno de ellos (The PostgreSQL Global Development Group, 2019).

1.9 Patrones de diseño

Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o

[5]Seguridad de la capa de transporte **Fuente especificada no válida.**.

interfaces. Un patrón de diseño es una solución a un problema de diseño. Para que una solución sea considerada un patrón debe poseer ciertas características, una de ellas es que debe haber comprobado su efectividad resolviendo problemas similares en ocasiones anteriores, otra es que debe ser reusable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias (Rojas, 2010).

Los patrones de diseño se pueden agrupar en dos grandes grupos; los GRASP (por sus siglas en inglés *General ResponsibilityAssignment Software Patterns*), que son patrones generales de software para asignación de responsabilidades y los GOF (por sus siglas en inglés *GangofFour*), encargados de la inicialización, agrupación y comunicación de los objetos. En el Capítulo 2 de la presente investigación, se describen los patrones de diseño que fueron empleados en el desarrollo del componente propuesto.

Patrones GOF:

Describen soluciones simples y eficaces a problemas específicos en el diseño de software orientado a objetos (Gamma, et. al, 1994).

Son combinaciones de componentes, casi siempre clases y objetos que por experiencia se sabe que resuelven ciertos problemas de diseño comunes (Braude, 2003).

Dentro de lospatrones GOF se encuentra, el Método de fabricación.

Método de fabricación (*Factory Method*): define una interfaz para crear un objeto, pero deja a las subclases decidir qué clase instanciar (Ochoa, 2018). Es utilizado cuando:

- Una clase no puede anticipar la clase de objetos que debe crear.
- Una clase quiere que sus subclases especifiquen las clases que ella debe crear.
- Utiliza una clase constructora abstracta con unos cuantos métodos definidos y otros abstractos.
- Las clases principales en este patrón son el creador y el producto.

Otros de los patrones son:

Observador (*Observer*): consiste en definir una dependencia entre objetos de forma que cuando un objeto cambie su estado, todos los objetos dependientes son notificados y cambian su estado automáticamente(Pacheco,2017).

Inyección de dependencias (DependencyInjection): consiste en colocar dentro de un objeto otros que puedan cambiar su comportamiento, sin que esto implique volver a crear el objeto (Pacheco, 2017).

Repositorio (*Repository)*: es un mediador entre el dominio de la aplicación y los datos que le dan persistencia. Con este planteamiento se puede pensar que el usuario de este repositorio no necesitaría conocer la tecnología utilizada para acceder a los datos, sino que le bastaría con saber las operaciones que nos facilita este "mediador", el repositorio (Roche, 2013).

En el capítulo 2 se describen cómo los patrones definidos anteriormente son aplicados en el diseño de los módulos propuestos.

Patrones GRASP: es el acrónimo de General ResponsibilityAssignment Software Patterns (Patrones Generales de Software para Asignar Responsabilidades) y se encargan de describir los principios fundamentales de la asignación de responsabilidades a objetos (Larman, 1999).

Dentro de los patrones GRASP se encuentran:

Creador:Se encarga de guiar la asignación de responsabilidades relacionadas con la creación de objetos. Su intención es encontrar un creador que necesite conectarse al objeto creado en alguna situación (Integra., 2010).

Bajo acoplamiento: Este patrón expresa que entre las clases deberán existir pocas ataduras, es decir, estas estarán lo menos relacionadas posible, de forma tal que, en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión posible en el resto de las clases, incrementando la reutilización y disminuyendo la dependencia entre las clases (Larman, 2003).

Alta cohesión:Propone que la información que almacena una clase debe de ser coherente y debe estar, en la medida de lo posible, relacionada con la clase. Al realizar un cambio en una clase con alta cohesión, todos los métodos que pueden verse afectados, estarán a la vista, en el mismo archivo. Incrementa la claridad, la reutilización y la facilidad de comprensión del diseño (Larman, 2003).

1.10 Pruebas

Las pruebas de software son un conjunto de actividades que pueden ser planificadas con antelación y ejecutarse sistemáticamente durante la implementación o al finalizar el desarrollo del software. Estas se realizan para identificar posibles fallos de funcionamiento, configuración o usabilidad de un programa o aplicación. Las pruebas de software se ejecutan a partir de la aplicación de métodos y técnicas. La organización del proceso de pruebas se realiza a través de cuatro niveles: unidad, integración, sistema y aceptación (Pressman, 2010). A continuación, se describen las disciplinas definidas en la metodología AUP-UCI para la etapa de pruebas de software y la relación de estas con los niveles de prueba.

1.10.1 Pruebas internas

En la disciplina pruebas internas se verifica a nivel de equipo de desarrollo el resultado de la implementación. Para la ejecución de estas pruebas se deben desarrollar artefactos de apoyo, tales como: diseños de casos de prueba, listas de chequeo y de ser posible componentes de prueba ejecutables para automatizar el proceso (Sánchez, 2015).

Para la validación de la solución propuesta las pruebas internas se ejecutan a nivel de unidad. En el capítulo 3 del presente documento se detalla cómo se realizó el desarrollo de las mismas.

Pruebas a nivel de unidad

Las pruebas a nivel de unidad se centran en el esfuerzo de verificación de la unidad más pequeña del diseño: el componente o módulo de software. Tomando como guía la descripción del diseño a nivel de componente, se prueban importantes caminos de control para describir errores dentro de los límites del módulo. El alcance restringido que se ha determinado para las pruebas de unidad limita la relativa complejidad de las pruebas y los errores que éstas descubren (Pressman, 2010).

En el epígrafe 3.4 del presente documento se explican los métodos de pruebas utilizados a nivel de unidad.

1.10.2 Pruebas de liberación

Las pruebas de liberación son diseñadas y ejecutadas por una entidad certificadora de la calidad externa al equipo de desarrollo. Estas se realizan a todos los entregables de los proyectos antes de ser entregados al cliente para su aceptación (Sánchez, 2015).

En el capítulo 3 del presente documento se describe cómo se ejecutaron las pruebas de liberación sobre la solución propuesta.

Los autores de la presente investigación deciden organizar las pruebas para validar la solución propuesta a partir de las disciplinas definidas en la metodología AUP variación UCI para la etapa de pruebas de software. En este caso solo se aplican las disciplinas Pruebas internas y Pruebas de liberación. Las Pruebas de Aceptación no se ejecutan teniendo en cuenta que la solución forma parte de un software en desarrollo al cual en un futuro posterior a la conclusión de la investigación se realizará la integración de estos módulos con este software en desarrollo, para luego aceptar el producto cuando esté totalmente terminado. Por tal motivo en las disciplinas de pruebas internas y pruebas de liberación solo se aplicará el nivel de unidad.

1.11 Conclusiones parciales

- El desarrollo del marco teórico referencial relacionado con los términos variedad, índice de precios y clasificadores, facilita una mejor comprensión del objeto de estudio de la presente investigación.
- El estudio artículos relacionados con el cálculo de índices de precios permitió identificar que el cálculo de índice para los productos de la construcción se realiza de formas distinta e individual en cada país, por tanto, el desarrollo de la presente solución constituye una novedad en el proceso de informatización cubano.
- El análisis de las características de la variación de la metodología *AUP* para la UCI, permitió definir las disciplinas a utilizar en la organización del desarrollo de los módulos propuestos, en correspondencia con las características de este.
- La decisión de utilizar las tecnologías y herramientas que se utilizan en el centro de desarrollo de software donde se realiza la presentetesis, contribuye a disminuir la curva de aprendizaje de los autores en el trabajo con estas, así como a la compatibilidad de la solución con el proyecto al cual será integrada.

CAPÍTULO 2: ANÁLISIS, DISEÑO E IMPLEMENTACIÓN DE LA SOLUCIÓN PROPUESTA

2.1 Introducción

En este capítulo se describe la solución propuesta para la Informatización de la Gestión del Índice de Preciode la construcción en la ONEI, así como la implementación del mismo. Se hace referencia a las técnicas empleadas para la captura de los requisitos. Se presentan los requisitos funcionales y no funcionales con los que deben cumplir los módulos propuestos en la presente investigación. Los requisitos funcionales son agrupados en historias de usuario. En el capítulo también se muestran los diagramas de componentes referentes a la arquitectura. Por otra parte, se expone la utilización de los patrones de diseño y estándares de codificación durante el diseño e implementación de la solución propuesta.

2.2 Descripción del negocio

La propuesta de solución consiste en el análisis, diseño, implementación y prueba de los módulos Clasificadores y Variedad-Establecimiento, que forman parte de la Gestión del Índice de Precio de la Construcción en el SIGIP. El negocio comienza con la gestión de los Clasificadores, los cuales están organizados en tres niveles, Grupos, Elementos y Variedades. Posteriormente se crean los Establecimientos, a los cuales se vinculan o empadronan las Variedades que se vayan a comercializar en los mismos.

La automatización del Índice de Precio de la Construcción y en especial los procesos comprendidos en ambos módulos son de vital importancia para garantizar la interacción, el acceso y la visualización de la información de forma rápida entre todas las sedes de la ONEI. Una vez implementada la solución se espera reducir la posibilidad de errores y evitar la pérdi[i]da completa o parcial de los datos que se gestionan, de forma tal que se contribuya a la actualización de la información.

6

[6] Empadronar: asentar o inscribir algún documento en algún archivo oficial o no oficial.

2.3 Requisitos

El esfuerzo principal en la disciplina Requisitos es desarrollar el modelo del sistema que se va a construir y comprende la administración de los requisitos funcionales y no funcionales del producto**.**

Un requisito es simplemente una declaración abstracta de alto nivel de un servicio que debe proporcionar el sistema o una restricción de éste. En el otro extremo, es una definición detallada y formal de una función del sistema (Sommerville, 2016).La gestión de requisitos es un conjunto de actividades que ayudan al equipo del proyecto a identificar, controlar y rastrear los requisitos y los cambios a estos en cualquier momento mientras se desarrolla el proyecto (Pressman, 2010).

2.3.1 Técnicas para la identificación de requisitos

Para identificar las necesidades del cliente se utilizan técnicas que permiten determinar y documentar los requisitos para el desarrollo de la solución. Esta actividad es continua durante el ciclo de desarrollo y combina, en diferentes puntos, diversas técnicas de identificación para obtener la visión más completa de las necesidades del usuario final (Pressman, 2010). Para la captura de los requisitos de los módulos propuestos se utilizan las siguientes técnicas:

- **Entrevista:** es de gran utilidad para obtener información cualitativa, requiere seleccionar bien a los entrevistados para obtener la mayor cantidad de información en el menor tiempo posible. Es muy aceptada y permite acercarse al problema de una manera natural (Sommerville, 2016). Esta técnica fue aplicada a los trabajadores de la ONEI mediante un cuestionario de preguntas realizadas verbalmente para el descubrimiento de las necesidades existentes en esta entidad.
- **Tormenta de ideas:** es una técnica de reuniones en grupo cuyo objetivo es la generación de ideas en un ambiente libre de críticas o juicios (Pressman, 2010). Esta técnica se evidencia durante las reuniones con el cliente donde, luego de un diálogo entre ambas partes, se obtuvo como resultado un conjunto de acuerdos con el fin de refinar las necesidades del cliente.

2.3.2 Especificación de requisitos

Los requisitos constituyen un punto clave en el desarrollo de aplicaciones informáticas. Un gran número de proyectos de software fracasan debido a una mala definición, especificación o administración de requisitos. Factores tales como requisitos incompletos o mal manejo de los cambios de los requisitos, llevan a proyectos completos al fracaso total. Los requisitos se enfocan en las especificaciones de lo que se desea desarrollar y tienen dos clasificaciones: requisitos funcionales y no funcionales. Los requisitos funcionales son declaraciones de los servicios que debe proporcionar el sistema, de la manera en que éste debe reaccionar a entradas particularidades y de cómo se debe comportar en distintas situaciones y los requisitos no funcionales son restricciones de los servicios o funciones ofrecidas por el sistema (Sommerville, 2016).

En la presente investigación, una vez realizado el levantamiento de información e identificadas las necesidades del cliente, se identificaron un total de 31 requisitos funcionales (RF). A continuación, en la tabla 1 se presentan cada uno de estos y sus respectivas descripciones.

Tabla 1 : Requisitos Funcionales.

N°	***Nombre***	**Descripción**	**Prioridad para el cliente**	**Complejidad**
1	Listar relación variedad-establecimiento	Permite listar una nueva relación de la variedad con un establecimiento.	Baja	Baja
2	Modificar relación variedad-establecimiento	Permite modificar una relación de la variedad con un establecimiento existente.	Media	Media
3	Visualizar relación variedad-establecimiento	Permite ver los detalles de una relación de la variedad con un establecimiento existente.	Baja	Baja
4	Eliminar relación variedad-establecimiento	Permite eliminar una variedad de un establecimiento.	Media	Media
5	Dar baja a una variedad en un establecimiento	Permite dar baja a una variedad dentro de un establecimiento.	Media	Baja

6	Dar alta a una variedad en un establecimiento	Permite dar de alta a una variedad dentro de un establecimiento.	Media	Baja
7	Listarestablecimientos	Permite buscar un establecimiento existente en el sistema, siguiendo un conjunto de filtros de búsqueda.	Baja	Baja
8	Añadir variedad-establecimiento	Permite crear o añadir una relación variedad-establecimiento.	Baja	Media
9	Registrar establecimiento	Permite registrar un nuevo establecimiento en el sistema.	Media	Media
1	Modificar establecimiento	Permite modificar un establecimiento existente en el sistema.	Media	Media
1	Visualizar establecimiento	Permite visualizar todos los datos del establecimiento seleccionado.	Baja	Baja
1	Eliminar establecimiento	Permite eliminar un establecimiento existente en el sistema.	Media	Baja
1	Dar baja a un establecimiento	Permite dar baja a un establecimiento que se encuentre activo.	Baja	Baja
1	Dar alta a un establecimiento	Permite dar de alta a un establecimiento que se encuentre desactivado.	Baja	Baja
1	Registrar nivel en el clasificador	Permite registrar un nuevo nivel al clasificador.	Alta	Alta
1	Modificar nivel en el clasificador	Permite modificar un nivel existente en el clasificador.	Alta	Alta

1	Eliminar nivel en el clasificador	1. Permite eliminar un nivel existente en el clasificador.	Alta	Alta
1	Listar característica de la variedad	Permite adicionar una nueva característica a una variedad.	Alta	Baja
1	Adicionar característica de la variedad	Permite modificar una característica a una variedad mientras no tenga historial de captaciones.	Alta	Baja
2	Modificar característica de variedad	Permite modificar la característica de la variedad existente.	Baja	Baja
2	Visualizar característica de la variedad	Permite ver todos los detalles de la variedad	Baja	Baja
2	Activar característica de la variedad	Permite activar una característica de una variedad que estaba desactivada.	Baja	Baja
2	Desactivar característica de la variedad	Permite desactivar una característica de una variedad que estaba activada.	Baja	Baja
2	Eliminar característica de la variedad	Permite eliminar una característica de una variedad que no tenga historial de captaciones.	Baja	Baja
2	Listar especificación de la característica de la variedad	Permite buscar una especificación de característica de la variedad existente en el sistema, siguiendo un conjunto de filtros de búsqueda.	Baja	Baja
2	Adicionar especificación de la característica de la variedad	Permite adicionar una característica a una variedad mientras no tenga historial de captaciones.	Baja	Baja

2	Modificar especificación de la característica de la variedad	Permite modificar una característica a una variedad mientras no tenga historial de captaciones.	Baja	Baja
2	Visualizar especificación de la característica de la variedad	Permite visualizar todos los detalles de una característica a una variedad mientras no tenga historial de captaciones.	Baja	Baja
2	Activar especificación de la característica de la variedad	Permite activar una especificación de la característica de una variedad que estaba desactivada	Baja	Baja
3	Desactivar especificación de la característica de la variedad	Permite desactivar una especificación de la característica de una variedad que estaba desactivada	Baja	Baja
3	Eliminar especificación de la característica de la variedad	Permite eliminar una especificación de la característica de una variedad que estaba desactivada	Baja	Baja

Fuente: elaboración propia.

En el caso de los requisitos no funcionales (RnF) se definieron un total de 6, los cuales son clasificados y descritos a continuación:

Seguridad:

RNF1. El sistema debe ser capaz de asociar a los usuarios autenticados los permisos especificados.

Confiabilidad:

RNF2. El marco de trabajo permite cancelar las instancias de los procesos no deseados y las tareas innecesarias.

RNF3. El sistema debe ser capaz de crear salvas cada un intervalo de tiempo y darle al usuario la posibilidad de restablecer, de esta forma si hay falta de fluido eléctrico se garantiza que no se pierda la información introducida por el usuario.

Usabilidad:

RNF4. Para llegar a la funcionalidad debe estar a no más de 3 clics.

RNF5. El sistema provee manuales de usuario. La ayuda es sensible al contexto e incluye información sobre los flujos de trabajo.

Eficiencia:
RNF 6. El sistema de ser capaz de ofrecer tiempos de repuestas mínimos o iguales a 5s.

2.3.3 Validación de los requisitos

Con el objetivo de garantizar que los módulos a desarrollar se correspondan con las necesidades del cliente, cada uno de los requisitos identificados fueron validados antes de llegar a la disciplina de análisis y diseño.La validación de requisitos examina las especificaciones para asegurar que todos los requisitos del sistema han sido establecidos sin ambigüedad, sin inconsistencias, sin omisiones, que los errores detectados hayan sido corregidos, y que el resultado del trabajo se ajusta a los estándares establecidos. Para la validación de estos se utilizaron las siguientes técnicas:

Construcción de prototipos de interfaz de usuario: esta técnica permite hacer simulaciones de los módulos propuestos implementados y brindando la posibilidad a los especialistas de tener una idea de cómo serán las interfaces de estos una vez desarrollados. En las HU se muestran los prototipos de cómo se ilustrarán en el software las interfaces de cada uno de los requisitos implementados.

Revisiones formales de los requisitos:se realizaron revisiones formales de cada requisito, por parte del cliente y el equipo de desarrollo, quedando validado que la interpretación de cada una de las descripciones no es ambigua, ni presenta omisiones o errores que hagan que la descripción del requisito no se corresponda con las necesidades del cliente.

Métrica Calidad de la especificación:para medir la calidad de la especificación de los requisitos de software se aplicó la métrica Calidad de la Especificación (CE). El empleo de esta métrica permite obtener un alto nivel de entendimiento y precisión de los requisitos, para llegar a ello, se debe primeramente calcular el total de requisitos de software como se muestra a continuación:

Nr = Nf + Nnf

Nr: total de requisitos de software.

Nf: cantidad de requisitos funcionales.
Nnf: cantidad de requisitos no funcionales.

Sustituyendo los valores en la ecuación, se obtiene:

Nr = 31 + 6

Nr = 37

Para calcular la especificidad de los requisitos (ER) o ausencia de ambigüedad en los mismos se realiza la siguiente operación:

Q1 = Nui / Nr

Nui: número de requisitos para los cuales todos los revisores tuvieron interpretaciones idénticas.

Para sustituir los valores de las variables en la ecuación se tuvo en cuenta que, de los requisitos especificados para el desarrollo de los módulos, dos de ellos causaron contradicción en sus interpretaciones (RF 26, RF 27, RF 28 y RF 29). Por tanto, la variable Q1 obtiene el siguiente valor:

Q1 = 33 / 37

Q1 = 0.89

Es importante aclarar que mientras más cerca de 1 está el valor de Q1, menor es la ambigüedad. Teniendo en cuenta el resultado anterior, igual a 0.89, se concluye que el 89% de los requisitos es entendible. Los 4 requisitos identificados como ambiguos fueron modificados y validados para garantizar su correcta comprensión, llegando al resultado ideal de Q1=1. Este resultado demuestra que el 100 % de los requisitos son fáciles de comprender.

2.4 Historias de usuario

Las Historias de Usuario (HU) son una forma rápida de administrar los requisitos de los usuarios sin tener que elaborar gran cantidad de documentos formales y sin requerir de mucho tiempo para administrarlos. Las historias de usuario permiten responder rápidamente a los requisitos cambiantes, por lo que una historia de usuario puede tener varios cambios a lo largo de un desarrollo sin afectarse el tiempo (Rouse, 2019).

Para el desarrollo de los módulos propuestos se definieron un total de 31 HU, una por cada RF. Las cuales muestran detalladamente cada especificación de los requisitos. A continuación, se

muestra una de las historias de usuario diseñada para la elaboración de estos módulos propuestos. Se especifica la HU Registrar establecimiento teniendo en cuenta que esta constituye una funcionalidad clave en el funcionamiento de los módulos propuestos.

Tabla 2 : HU Registrar establecimiento.

<table>
<tr><td colspan="3">Historia de usuario</td></tr>
<tr><td>Número: 1</td><td colspan="2">Requisito: Registrar establecimiento</td></tr>
<tr><td colspan="2">Programador: Yunisleidis Montenegro Mejias</td><td>Iteración Asignada: 1</td></tr>
<tr><td colspan="2">Prioridad: media</td><td>Tiempo Estimado:36 h</td></tr>
<tr><td colspan="2">Riesgo en Desarrollo:</td><td>Tiempo Real: 36 h</td></tr>
<tr><td colspan="3">Descripción: Campos:
➢ DPA (campo de selección única que lista todas las provincias del país con sus códigos)
➢ Nombre(campo de texto que permite recoger el nombre del nuevo establecimiento)
➢ Dirección (campo de texto que permite recoger la dirección del nuevo establecimiento)
➢ Fecha a captar (campo de selección única que carga las posibles fechas en que se puede captar el establecimiento)
➢ Nombre(campo de texto que permite recoger el nombre del contacto)
➢ Teléfono (campo de texto que permite recoger el teléfono del contacto del nuevo establecimiento)
Opciones:
➢ Siguiente (Botón para acceder a la interfaz para añadir variedades al establecimiento</td></tr>
<tr><td colspan="3">Observaciones:</td></tr>
</table>

Prototipo de interfaz gráfica de usuario:

Fuente: elaboración propia.

2.5 Análisis y diseño

En esta disciplina los requisitos pueden ser refinados y estructurados para conseguir una comprensión más precisa de estos, y una descripción que sea fácil de mantener y ayude a la estructuración del sistema incluyendo la arquitectura. En esta disciplina se modela el sistema y su forma para que soporte todos los requisitos, incluyendo los requisitos no funcionales. (Sánchez, 2015).

2.5.1 Diseño arquitectónico

El diseño arquitectónico garantiza cómo debe organizarse un sistema y cómo tiene que diseñarse la estructura global del mismo. En el modelo del proceso de desarrollo de software, el diseño arquitectónico es la primera etapa en el proceso de diseño del software. Es el enlace crucial entre el diseño y la ingeniería de requerimientos, ya que identifica los principales componentes estructurales en un sistema y la relación entre ellos. La salida del proceso de

diseño arquitectónico consiste en un modelo que describe la forma en que se organiza el sistema como un conjunto de componentes en comunicación (Sommerville, 2016).

La solución propuesta se estructura con la API *GraphQL* de los módulos a desarrollar,dividiendo la arquitectura en dos partes esenciales *fronted* y*backend.* El*fronted* interactúa con los usuarios y el*backend* procesa las entradas desde el *frontend.*Teniendo en cuenta lo anterior, GraphQL utiliza el patrón arquitectónico Vuex para gestionar el estado de la aplicación. También se utiliza la biblioteca de *apollo-client* para tener un manejo de los datos. En la figura 1 se muestra la arquitectura definida para el*frontend*de los módulos propuestos.

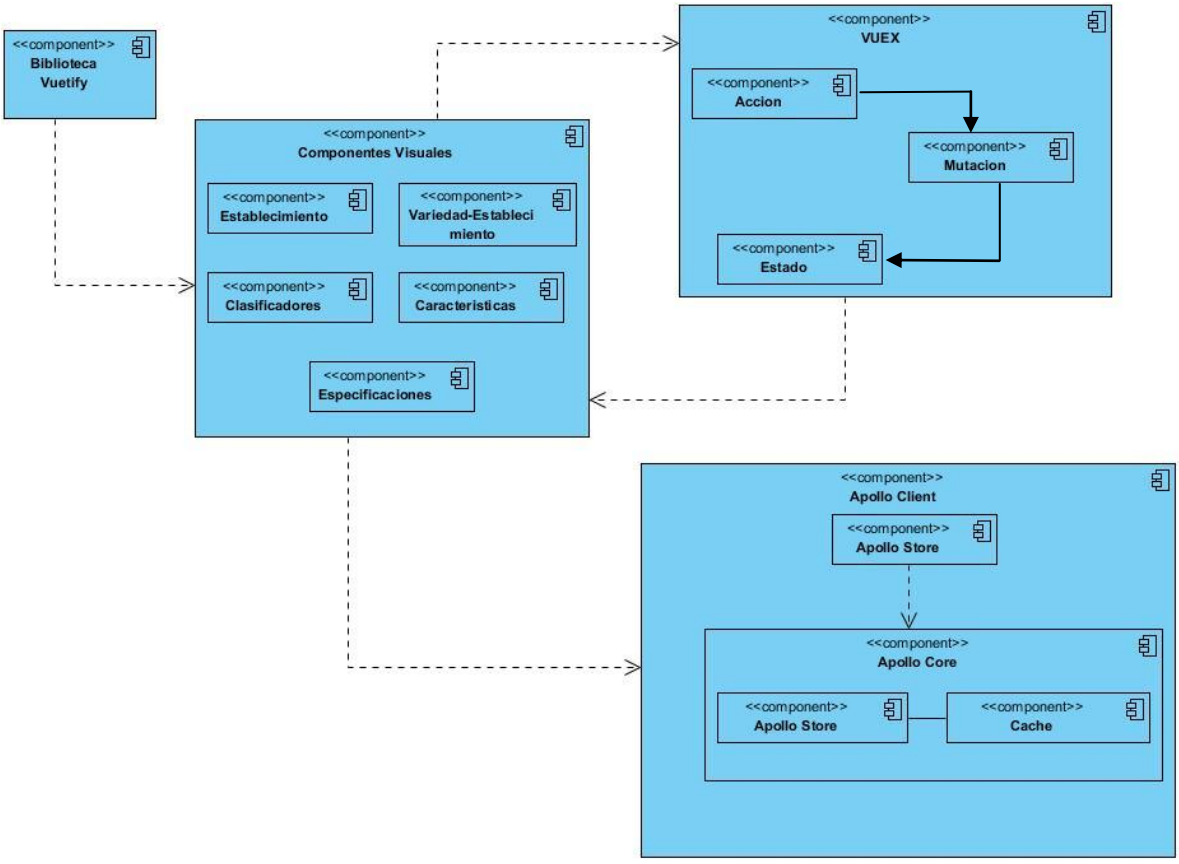

Figura 1 : Arquitectura para los módulos.

Fuente:elaboración propia.

El diagrama representado en la figura 1 muestra el marco de trabajo para las interfaces de usuario de *Vuetify*. Este se utiliza para definir los componentes visuales para los Establecimientos, Variedad-Establecimiento, Clasificadores, Características y Especificaciones. Para la gestión del estado de los módulos propuestos se emplea la biblioteca Vuex, en la cual se

definen las acciones que lanzan las mutaciones correspondientes a la misma y esta modifica el estado deseado, que a su vez es reflejado en los componentes visuales. Por otra parte, se utiliza la biblioteca *apollo-client* para la gestión del estado de los datos. Esta biblioteca define mutaciones, consultas y suscripciones que se utilizan para comunicarse con la API-GraphQL. Cuenta también con un *store* que guarda el estado de los datos para ser utilizado en los componentes visuales.*Apollo-client*también define una cache para brindar datos que no hayan cambiado durante las consultas a la API GraphQL.

Por otra parte, en la API GraphQLde los módulos propuesto se aplica el patrón arquitectónico N-Capas, definiéndose las capas web, infraestructura y dominio.

La arquitectura basada en N-Capas se enfoca principalmente en el agrupamiento de funcionalidades relacionadas dentro de una aplicación en distintas capas que son colocadas verticalmente una encima de otra. La funcionalidad dentro de cada capa se relaciona con una responsabilidad específica para cada una de las capas. El dividir en capas una aplicación, permite la separación de responsabilidades, lo que proporciona una mayor flexibilidad y un mejor mantenimiento (Muñoz Serafin, 2018).

El patrón arquitectónico N Capas es una extensión del patrón Capas tradicional (este ayuda a estructurar las aplicaciones que se pueden descomponer en grupos de subtareas en la que cada grupo de subtareas está en un nivel particular de abstracción). En el nivel más alto y abstracto, la vista de arquitectura lógica de un sistema puede considerarse como un conjunto de servicios relacionados agrupados en diversas capas(Escalante, 2014).

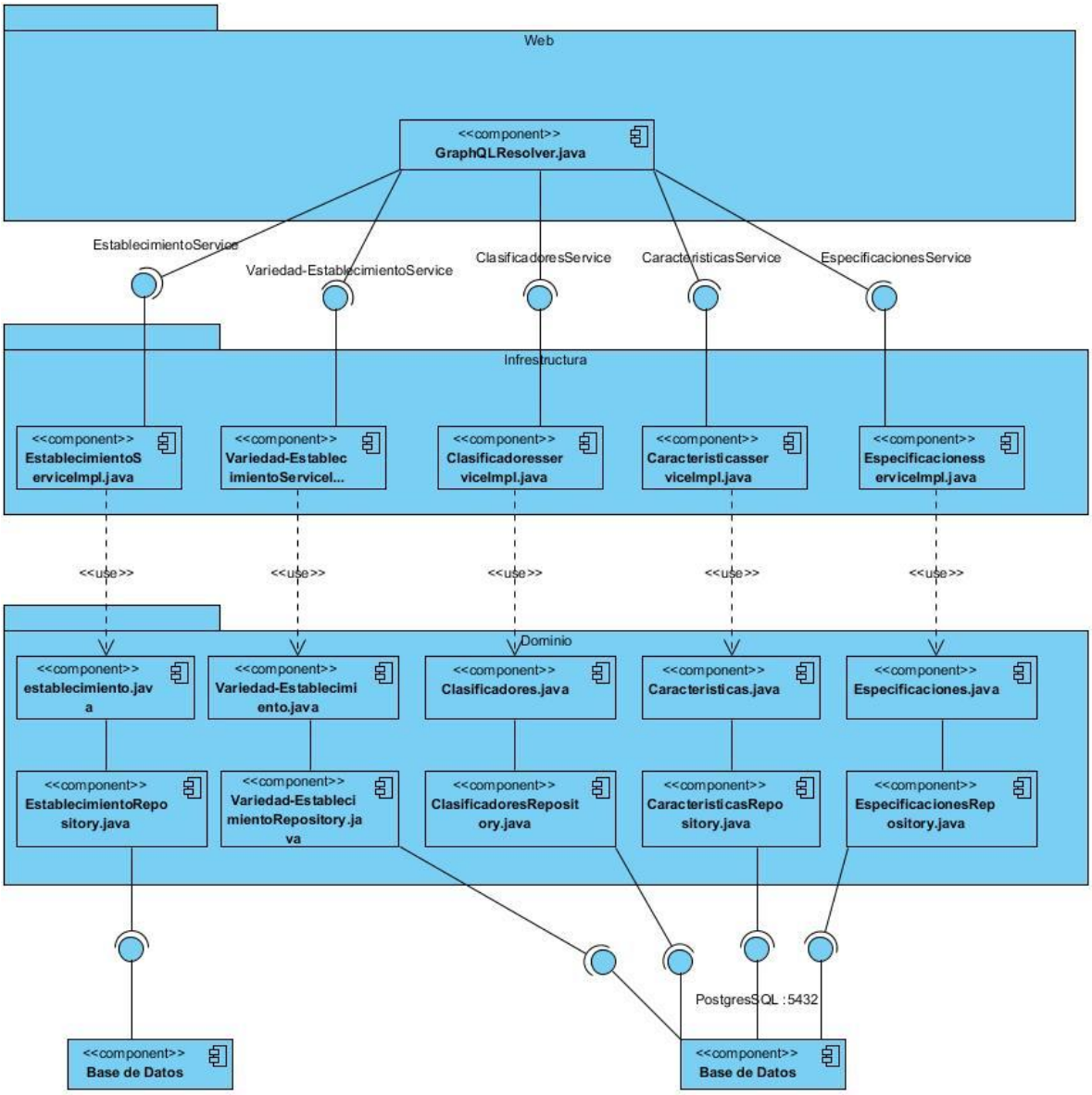

Figura 2 : Arquitectura API-GraphQL.
Fuente: elaboración propia.

En la figura 2se muestrala aplicación del patrón N-Capas en la arquitectura de la solución propuesta. La capa Web incluye el componente *GraphQLResolver* donde se encuentra el único punto de acceso de GraphQl. En la capa Dominio se encuentran las clases de persistencia, y los componentes repositorios los cuales son los encargados de la interacción con la base de datos. En la capa Infraestructura se encuentran las implementaciones de los servicios que definen las funciones necesarias para devolver los datos solicitados por el cliente GraphQL. Esta capa

facilita además todas las interfaces de los servicios, las cuales son utilizadas por *GraphQLResolver.*

En la parte del *backend*que se muestra en la figura 2 como los módulos inciden en las capas componente, servicios y acceso a datos. Estos módulos son implementados sobre la arquitectura *backend* del SIGIP a través del uso del patrón arquitectónico N Capas.

2.5.2 Patrones de diseño

Para diseñar los módulos propuestos se emplearon un conjunto de patrones de diseño, los cuales son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software. A continuación, se describen los patrones empleados:

Patrones GRASP

Los Patrones Generales de Software para la Asignación de Responsabilidades describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones (Larman, 2016). En el caso de la presente investigación se aplicaron los siguientes patrones GRASP.

Entre los patrones GRASP utilizados se encuentran:

Experto:Es utilizado para que cada objeto realice la funcionalidad de acuerdo a la información que domina. Se evidencia en las clases *Service*,*ServiceImpl* y en los *Repository*, las cuales agrupan funcionalidades y toman decisiones sobre una entidad determinada. Por ejemplo, las clases *EstablecimientoService*, *EstablecimientoServiceImpl* y *EstablecimientoRepository* solo toman decisiones relacionadas con la Entidad Establecimiento.

Creador: el uso de este patrón se evidencia en las clases *ServiceImpl*, que son las que tienen la responsabilidad de instanciar objetos para cumplir sus funciones. PorEjemplo, la clase *EstablecimientoServiceImpl*,que es donde se creanlos objetos de tipo Establecimientopara realizar las diferentes funcionalidades.

Alta cohesión: Se evidencia en el diseño de cada una de las clases de los módulos garantizando que solo existan en ellas las características y funcionalidades necesarias. Entre estas clases se encuentran las clases *Service, ServiceImpl* y los *Repository*.

Bajo acoplamiento: Al lograr una alta cohesión en el diseño de las clases del sistema, se garantiza un bajo acoplamiento ya que permite tener clases más independientes, donde la realización de cambios sea más sencillay a su vez permitan la reutilización. Este patrón se observa en las clases *Service*, *ServiceImpl*y los *Repository.*

Entre los patrones GoFse encuentran:

Instancia única (*Singleton*):Teniendo en cuenta que para el desarrollo de los módulos propuestos se utiliza el marco de trabajo SpringBoot, se asegura que todas las clases del componente mantengan una sola instancia y proporcionan un punto de acceso global a ellas durante la ejecución.

Fachada:Proporcionar una interfaz para la creación de familias de objetos interdependientes o interrelacionados, sin especificar sus clases concretas. El patrón se utilizó al diseñar interfaces con funcionalidades bien definidas que van a ser implementadas por todas las clases que las necesiten. Se evidencia en las clases *Service* que son interfaces que posteriormente son implementadas, ejemplo *EstablecimientoService* es una interfaz que posteriormente es implementada en la clase *EstablecimientoServiceImpl*.

Iterador:Proporciona una forma de acceder secuencialmente a los elementos de un objeto compuesto por agregación sin necesidad de desvelar su representación interna. Se utiliza en las clases *ServiceImpl*para recorrer las estructuras de datos utilizadas. Este patrón permite garantizar una integridad en la información.

2.5.3 Diagrama de clases del diseño

El diagrama de clases del diseño describe gráficamente las especificaciones de las clases de software y las interfaces que participan en el sistema con sus relaciones estructurales y de herencia. Los diagramas de este tipo contienen las definiciones de las entidades del software en vez de conceptos del mundo real y son utilizados durante el proceso de análisis y diseño, donde se crea una vista lógica de la información que se maneja en el sistema, los componentes que se encargan del funcionamiento y la relación entre uno y otro (Larman, 2016).

A continuación, se muestra el diagrama de clases del diseño, en el cual se encuentran las clases correspondientes, teniendo en cuenta que el mismo responde a la realización de los módulos propuestos en todo el documento.

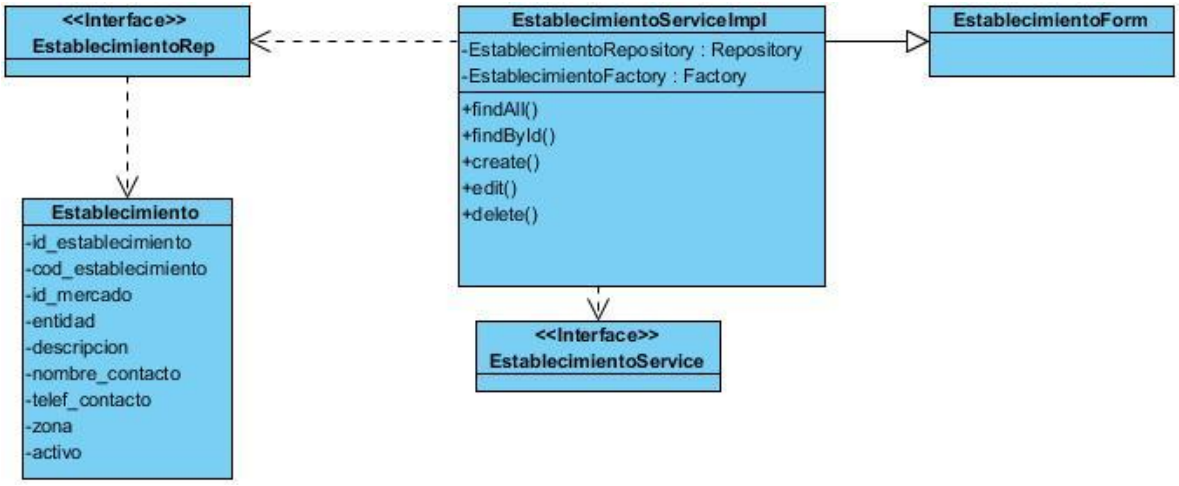

Figura 3 : Diagrama de clases de diseño.

Fuente: elaboración propia

El diagrama de clases está compuesto por las siguientes clases:

- Clase Establecimiento: es una clase de persistencia que representa los datos que contiene un establecimiento, que presenta una relación de asociación bidireccional con *EstablecimientoRepository.*
- Clase *EstablecimientoForm*: representa los parámetros de búsqueda de un Establecimiento.
- Interfaz *EstablecimientoService*: define todas las abstracciones necesarias para la gestión de los establecimientos.
- Interfaz *EstablecimientoRepository*: es la interfaz encargada del acceso a datos, la cual permite la comunicación entre la clase *EstablecimientoServiceImpl* de los módulos y la base de datos, la misma presenta una relación de asociación bidireccional con la clase Establecimiento.
- Clase *EstblecimientoServiceImpl*: implementa las abstracciones definidas en la clase *EstablecimientoService* y permite la gestión de los establecimientos, la misma presenta una relación de uso con la interfaz *EstablecimientoService,* y la

clase*EstablecimientoForm*, además presenta una relación de asociación unidireccional con la interfaz *EstablecimientoFactory* y la clase *EstablecimientoRepository.*

2.5.4 Modelo de base de datos

Un modelo de base de datos determina la estructura lógica de una base de datos y de manera fundamental determina el modo de almacenar, organizar y manipular los datos.La estructura óptima depende de la naturaleza de la organización de los datos de la aplicación y de los requisitos de esta que permitirán fiabilidad, mantenibilidad, escalabilidad y coste. Este puede ofrecer al usuario cierto control sobre la implementación física. El modelo de datos no solo es un modelo de estructura, también define el conjunto de operaciones que se realizan con los datos.(Álvarez, 2007)

En la figura 4se muestran a través del diagrama Entidad-Relación las entidades, atributos y relaciones que componen el modelo de datos de la solución propuesta.El modelo de datos representado en la figura 4cuenta con un total de 13 tablas, dentro del cual se gestionan tanto los elementos del negocio como los de la arquitectura. Este diagrama también es el encargado de agrupar un conjunto de entidades con atributos en común. Las tablas del modelo son las encargadas de gestionar conceptos específicos de los módulos en general, por ejemplo, en la tabla variedad-establecimiento es donde se guarda el id del establecimiento, el estado (valor booleano), fecha de inicio, fecha de fin y la descripción de las entidades que se registran en el sistema.

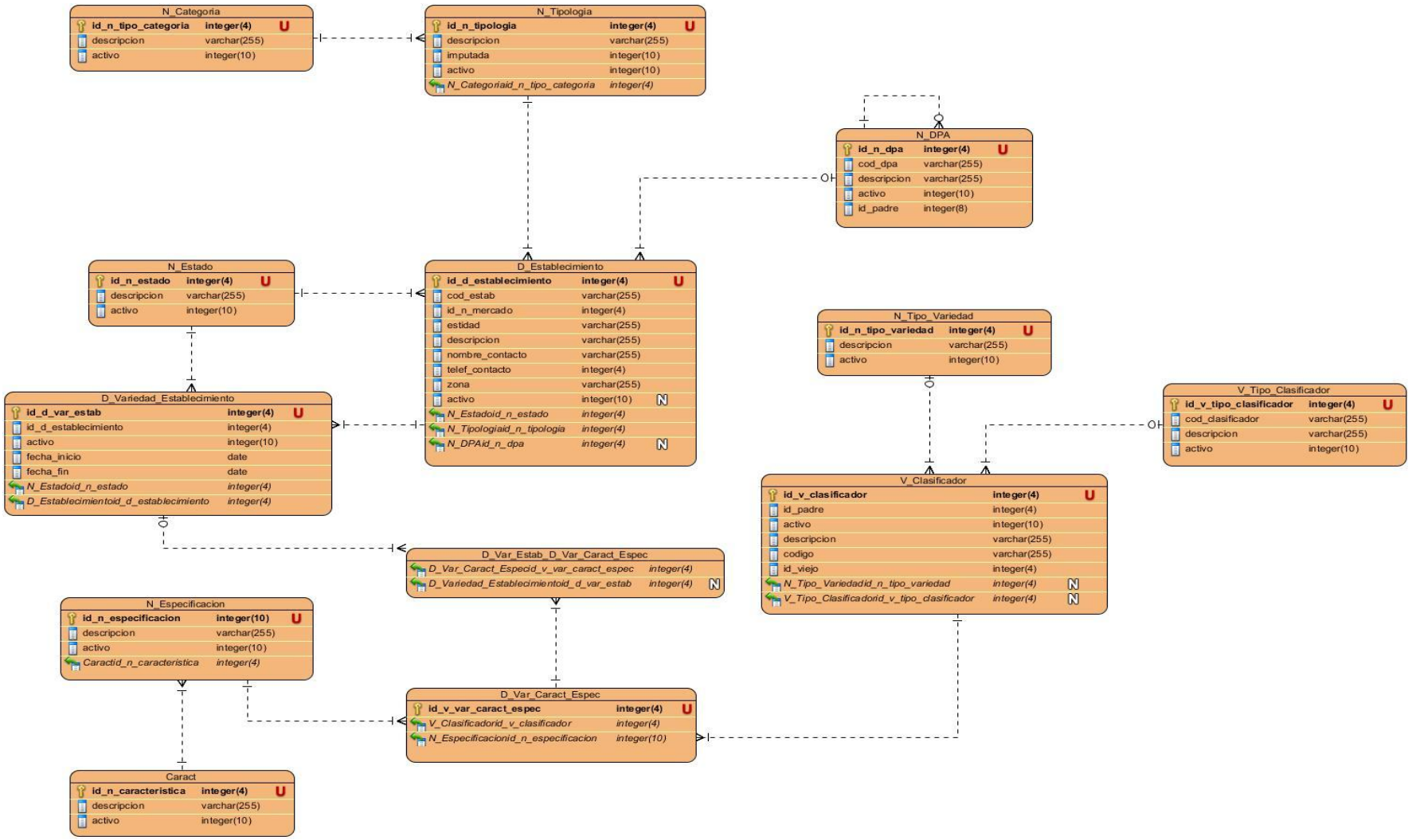
N_Categoria
id_n_tipo_categoria integer(4) U
descripcion varchar(255)
activo integer(10)
N_Tipologia
id_n_tipologia integer(4) U
descripcion varchar(255)
imputada integer(10)
activo integer(10)
N_Categoriaid_n_tipo_categoria integer(4)
N_DPA
id_n_dpa integer(4) U
cod_dpa varchar(255)
descripcion varchar(255)
activo integer(10)
id_padre integer(8)
N_Estado
id_n_estado integer(4) U
descripcion varchar(255)
activo integer(10)
D_Establecimiento
id_d_establecimiento integer(4) U
cod_estab varchar(255)
id_n_mercado integer(4)
estidad varchar(255)
descripcion varchar(255)
nombre_contacto varchar(255)
telef_contacto integer(4)
zona varchar(255)
activo integer(10) N
N_Estadoid_n_estado integer(4)
N_Tipologiaid_n_tipologia integer(4)
N_DPAid_n_dpa integer(4) N
N_Tipo_Variedad
id_n_tipo_variedad integer(4) U
descripcion varchar(255)
activo integer(10)
V_Tipo_Clasificador
id_v_tipo_clasificador integer(4) U
cod_clasificador varchar(255)
descripcion varchar(255)
activo integer(10)
D_Variedad_Establecimiento
id_d_var_estab integer(4) U
id_d_establecimiento integer(4)
activo integer(10)
fecha_inicio date
fecha_fin date
N_Estadoid_n_estado integer(4)
D_Establecimientoid_d_establecimiento integer(4)
V_Clasificador
id_v_clasificador integer(4) U
id_padre integer(4)
activo integer(10)
descripcion varchar(255)
codigo varchar(255)
id_viejo integer(4)
N_Tipo_Variedadid_n_tipo_variedad integer(4) N
V_Tipo_Clasificadorid_v_tipo_clasificador integer(4) N
D_Var_Estab_D_Var_Caract_Espec
D_Var_Caract_Especid_v_var_caract_espec integer(4)
D_Variedad_Establecimientoid_d_var_estab integer(4) N
N_Especificacion
id_n_especificacion integer(10) U
descripcion varchar(255)
activo integer(10)
Caractid_n_caracteristica integer(4)
D_Var_Caract_Espec
id_v_var_caract_espec integer(4) U
V_Clasificadorid_v_clasificador integer(4)
N_Especificacionid_n_especificacion integer(10)
Caract
id_n_caracteristica integer(4) U
descripcion varchar(255)
activo integer(10)

Figura 4 : Modelo de datos de la solución propuesta.

Fuente: elaboración propia.

2.6 Implementación

En esta disciplina a partir de los resultados del Análisis y el Diseño se construye la solución que da lugar a la presente investigación, para la cual se aplicaron un conjunto de estándares de codificación definidos por el equipo de desarrollo del SIGIP.

2.6.1 Estándares de codificación

Los estándares de codificación constituyen buenas prácticas o conjunto de reglas no formales que han ido surgiendo en las comunidades de desarrolladores de software con el paso del tiempo. Estos tienen el propósito de obtener un código fuente más legible, portable, seguro, eficiente, robusto y cohesionado, permitiendo mayor velocidad en el desarrollo, mejor coordinación entre los equipos de trabajo. Además logran que para un desarrollador sea más fácil integrarse a un proyecto ya comenzado, se eviten bugs[7] y se faciliten los procesos de mantenibilidad y revisión de código (Hommel, 2019). A continuación, se describen los estándares de codificación utilizados para el desarrollo del componente propuesto:

- Los nombres de las clases serán con mayúscula, en caso de ser un nombre compuesto las siguientes palabras se escribirán de igual forma, ejemplo: *EstablecimientoService*.
- Los nombres de los métodos serán con minúscula, en caso de ser un nombre compuesto las siguientes palabras se escribirán con mayúscula, ejemplo: registrarEstablecimiento.
- Los identificadores para las variables y los parámetros serán con letras en minúsculas y en caso de ser un nombre compuesto las siguientes palabras se escribirán con mayúscula, ejemplo: variedad y *variedadDTO*.
- Los nombres de variables o funciones deben ser lo suficientemente descriptivos, sin exceder de 30 caracteres.
- El nombre de la clase controladora terminará con la palabra *Resolver*, ejemplo: *GraphQLResolver*.
- Las interfaces de repositorio comenzarán nombrándose por el nombre de la entidad y seguido la palabra *Repository*, ejemplo: *EstablecimientoRepository*.
- Las interfaces de los servicios comenzarán nombrándose por el nombre de la entidad y seguido la palabra *Service*, ejemplo: *EstablecimientoService*.

[7] Error de software que desencadena un resultado no deseado

- Las implementaciones de las interfaces de servicios comenzarán nombrándose por el nombre de la entidad y seguido la palabra *ServiceImpl*, ejemplo: *EstablecimientoServiceImpl* y *ClasificadoresServiceImpl.*

2.7 Conclusiones parciales

- El empleo de la metodología *AUP* en su variación para la UCI en función de describir el proceso de desarrollo de los módulos propuestos, permitió organizar el desarrollo de la solución y generar los artefactos necesarios.
- La aplicación de patrones arquitectónicos y el uso de patrones de diseño, permitieron obtener una solución con poca dependencia entre clases, flexible al mantenimiento y a la introducción de cambios.
- El empleo de estándares de codificación que contribuye a la mantenibilidad de la solución.
- La implementación de los módulos da cumplimiento a cada uno de los RF y RnF identificados en la disciplina de requisitos.

CAPÍTULO 3: VALIDACIÓN DE LA SOLUCIÓN PROPUESTA

3.1 Introducción

En el presente capítulo se muestran los elementos que forman parte de la validación de la solución. Se exponen los resultados de la aplicación de métricas para medir la calidad de los módulos propuestos, las relaciones entre clases y el tamaño operacional de las clases del diseño. Por otra parte, se define la estrategia de prueba aplicada a partir de las disciplinas establecidas para la etapa de pruebas por la metodología AUP variación UCI. En la estrategia se define solo realizar la disciplina de pruebas internas a nivel de unidadcon la aplicación de sus respectivos métodos y técnicas.

3.2 Validación del diseño

Para poder verificar la calidad del diseño propuesto se emplearon las métricas de diseño relaciones entre clases (RC) y tamaño operacional de clases (TOC).

3.2.1 Métrica Tamaño Operacional de Clase (TOC)

A cada una de las clases del diseño se le aplico la métrica TOC con el objetivo de medir la calidad con respecto a la de su responsabilidad, complejidad de implementación y reutilización de los datos. Se describe a continuación los pasos seguidosal aplicar esta métrica

- Cálculo del umbral. El umbral se toma del tamaño general de una clase que se determina sumando todas las operaciones que posee el diseño.
- Calcular el promedio de los umbrales.

En la siguiente tabla se muestranlos datos a utilizar para la aplicación de la métrica TOC sobre el diseño de clases de los módulos propuestos.

Tabla 3:Métricas TOC.

Atributos de Calidad	Clasificación	Criterio
Responsabilidad	Baja	Umbral<= Promedio
	Media	Promedio < Umbral < = 2*

		Promedio
	Alta	Umbral > 2* promedio
Complejidad de Implementación	Baja	Umbral <= Promedio
	Media	Promedio < Umbral < = 2* Promedio
	Alta	Umbral > 2* promedio
Reutilización	Baja	Umbral > 2* promedio
	Media	Promedio < Umbral < = 2* Promedio
	Alta	Umbral <= Promedio

Fuente:(Lorenz, 1994)

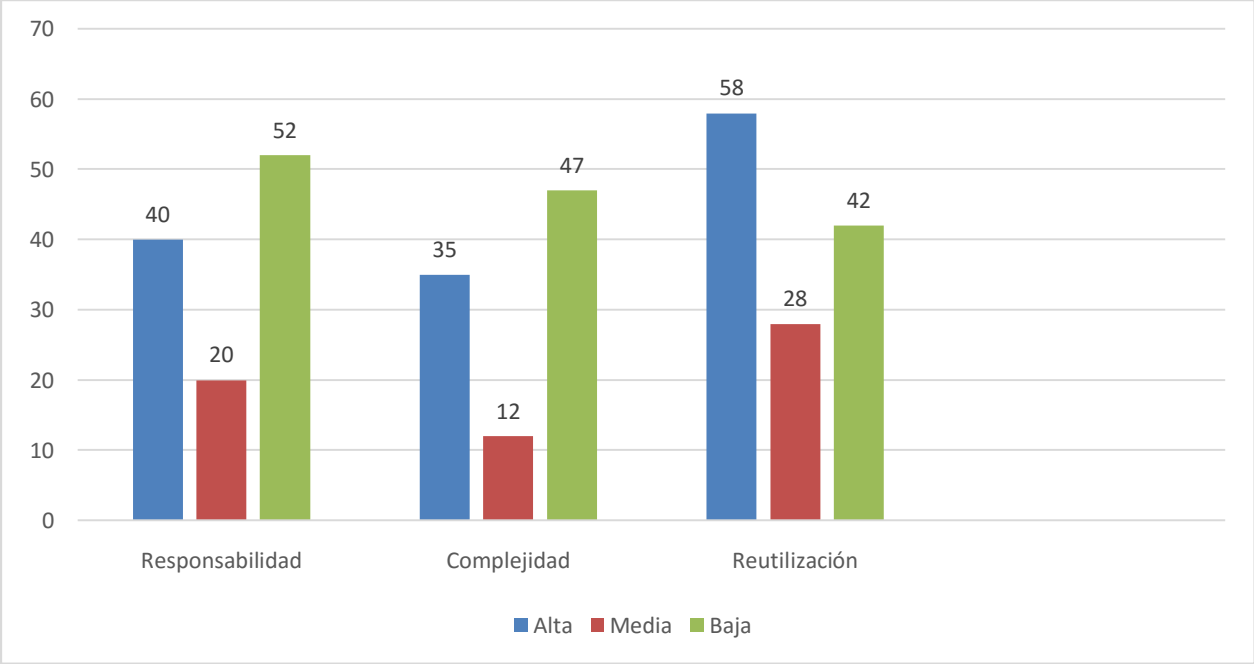

Figura 5 : Representación en (%) de los resultados de la aplicación de la métrica TOC.

Fuente: elaboración propia.

La figura anterior demuestra que con la aplicación de la métrica TOC se obtuvieron resultados positivos en relación a los siguientes atributos evaluados:

Responsabilidad: los resultados fueron satisfactorios, teniendo en cuenta que el 52 % de las clases tienen una responsabilidad Baja.

Complejidad de implementación: los resultados fueron satisfactorios, pues se demostró que el 47 % de las clases tienen una complejidad baja.

Reutilización: los resultados obtenidos fueron satisfactorios, demostrándose que el 58 % de las clases tienen una reutilización alta.

3.2.2. Métrica Relaciones entre clases (RC)

La métrica RC está dada por el número de relaciones de uso de una clase con otra. El primer paso en la aplicación de esta es evaluar los siguientes atributos de calidad: acoplamiento, complejidad de mantenimiento, reutilización de cada clase y la cantidad de pruebas que cada clase requiere (Pressman, 2010).
Se exponen los pasos que se llevaron a cabo para aplicar la métrica a todas las clases del componente propuesto en la presente investigación:

- Determinar la Cantidad de Relaciones de Uso (CRU) que poseen las clases a medir.
- Calcular el promedio de las CRU.

En la siguiente tabla se muestranlos datos a utilizar para la aplicación de la métrica RC sobre el diseño de clases de los módulos propuestos.

Tabla 4: Rango de valores para medir la afectación de los atributos de calidad (RC).

Atributos de calidad	Clasificación		Criterio
Acoplamiento	Ninguna		CRU=0
	Baja		CRU=1
	Media		CRU=2
	Alta		CRU>2
Complejidad de mantenimiento	Baja		CRU<= Promedio
	Media		Promedio <CRU< = 2* Promedio
	Alta		CRU> 2* promedio
Reutilización	Baja		CRU> 2* promedio
	Media		Promedio <CRU< = 2*

			Promedio
	Alta		CRU<= Promedio
Cantidad de pruebas	Baja		CRU<= Promedio
	Media		Promedio <CRU< = 2* Promedio
	Alta		CRU> 2* promedio

Fuente:(Lorenz, 1994)

En la figura 6se muestranlos resultados obtenidos luego de aplicar la métrica RC sobre el diseño de clases del componente propuesto.

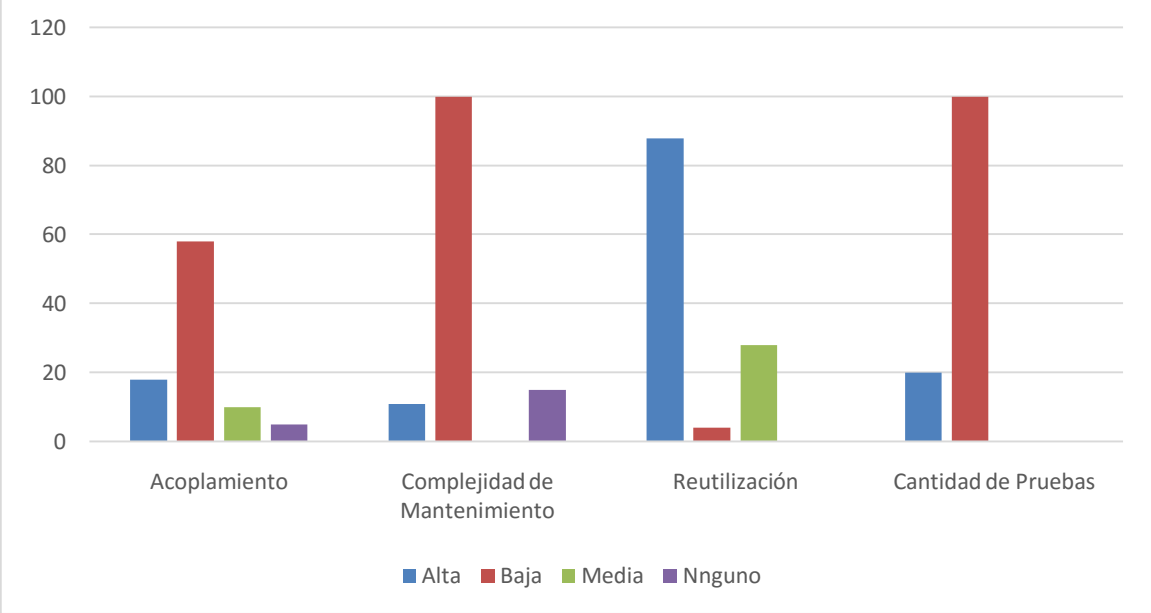

Figura 6 : Representación en (%) de los resultados de la aplicación de la métrica RC.**Fuente:**elaboración propia.

La figura anterior demuestra que con la aplicación de la métrica RC se obtuvieron resultados positivos en relación a los siguientes atributos evaluados:

Acoplamiento: los resultados obtenidos son positivos teniendo en cuenta que el 58 % de las clases no poseen acoplamiento.

Complejidad de mantenimiento: los resultados mostrados en la figura anterior, demuestran que el 100 % de las clases del componente propuesto presentan una complejidad de mantenimiento baja, facilitando así las futuras actividades de soporte sobre el mismo.

Reutilización:los resultados obtenidos fueron satisfactorios, demostrándose que el 88% de las clases tienen una reutilización alta.

Cantidad de pruebas: los resultados demuestran que el 100 % de las clases poseen un bajo grado de esfuerzo a la hora de realizar cambios, rectificaciones o pruebas sobre ellas.

Con los resultados obtenidos una vez aplicada la métrica RC se concluye que el diseño de los módulos propuestostiene una baja complejidad de mantenimiento y acoplamiento entre sus clases. Además, se requiere de un bajo grado de esfuerzos para realizar cambios sobre la mayoría de las clases y estás a su vez presentan un elevado por ciento de reutilización. Todo lo anterior facilita la obtención de una solución informática escalable, con poca dependencia entre clases, flexible al mantenimiento y a la introducción de cambios.

3.3Pruebas de Software

Las pruebas de software son un conjunto de actividades que pueden ser planificadas con antelación y ejecutarse sistemáticamente durante la implementación o al finalizar el desarrollo del software. Estas comprenden un conjunto de actividades que se realizan para identificar posibles fallos de funcionamiento, configuración o usabilidad de un programa o aplicación. Las pruebas de software se ejecutan a partir de la aplicación de métodos y técnicas. La organización del proceso de pruebas se realiza a través de cuatro niveles: unidad, integración, Sistema y aceptación (Pressman, 2010).

Para la validación de los módulos propuestos se define una estrategia de prueba de software a partir de las disciplinas establecidas por la metodología AUP variación UCI. En el caso de la presente investigación solo se realizan pruebas internas en el nivel de unidad, teniendo en cuenta que el alcance de la tesis solo comprende el desarrollo de los módulos, sin llegar a su integración con el SIGIP.

3.3.1Pruebas internas

A continuación, se detalla cómo el equipo de desarrollo ejecutó las pruebas internas a la solución propuesta, organizada en el nivel de unidad, aplicando los correspondientes métodos y tipos de pruebas.

Pruebas a nivel de unidad

Las pruebas a nivel de unidad se concentran en el esfuerzo de verificación de la unidad más pequeña del diseño, el componente o módulo de software. Tomando como guía la descripción del diseño a nivel de componente, se prueban importantes caminos de control para describir errores dentro de los límites del módulo. El alcance restringido que se ha determinado para las pruebas de unidad limita la relativa complejidad de las pruebas y los errores que éstas descubren (Pressman, 2010). En el caso de la presente investigación en este nivel de prueba se decide aplicar los métodos de caja blanca y caja negra.

Métodos de caja blanca:

El método de caja blanca posibilita el desarrollo de casos de prueba que garanticen la ejecución, al menos una vez, de los caminos independientes (Pressman, 2010). En el caso del presente trabajo de diploma el método fue ejecutado aplicando la técnica de ruta básica.

La técnica de ruta básica tiene como objetivo comprobar que cada camino se ejecute independiente de un componente o programa, obteniéndose una medida de la complejidad lógica del diseño. Esta técnica debe ser utilizada para evaluar la efectividad de los métodos asociados a una clase, con el objetivo de asegurar que cada camino independiente sea ejecutado por lo menos una vez en el sistema (Pressman, 2010).

En la validación del componente propuesto la técnica de ruta básica se aplicó a todos los métodos de las clases controladoras, teniendo en cuenta que estas agrupan las principales funcionalidades de la solución. A continuación, se describe la aplicación del método de caja blanca sobre la funcionalidad *ListarEstablecimiento*, perteneciente a la clase *EstablecimientoServiceImpl* (ver Figura 7). Se toma como muestra esta funcionalidad teniendo en cuenta que es una de las de mayor prioridad yrespondea uno de los principales requisitos funcionales de la solución.

```
1  @Override
2  public List<Establecimiento> findByDescripcionContiene(String descripcion, int paging, int size) {
3      QEstablecimiento qEstablecimiento = QEstablecimiento.establecimiento; 1
4      BooleanExpression predicate = qEstablecimiento.descripcion.toLowerCase().contains(descripcion.toLowerCase()); 2
5
6      if (size != -1) {  3
7          Pageable pageable = PageRequest.of(paging, size, Sort.by("descripcion"));  4
8          Page<Establecimiento> page = establecimientoRepository.findAll(predicate, pageable);  5
9          List<Establecimiento> establecimientoList = page.getContent();  6
10         return establecimientoList; 7
11     } else {
12         List<Establecimiento> establecimientoList = (List<Establecimiento>) establecimientoRepository.findAll(predicate, Sort.by("descripcion")); 8
13         return establecimientoList;  9
14     }
15 }  10
```

Figura 7: Funcionalidad *ListarEstablecimiento.*

Fuente: elaboración propia

Una vez definido el código sobre el cual se aplica el método, los pasos a seguir para desarrollar la técnica de ruta básica son los siguientes:

1) Confeccionar el grafo de flujo, este muestra el flujo de control lógico (Pressman, 2010). Está compuesto por los siguientes elementos:

- Nodos: son círculos que representan una o más sentencias procedimentales.
- Aristas: son flechas que representan el flujo de control y son análogas a las flechas deldiagrama de flujo.
- Regiones: son las áreas delimitadas por aristas y nodos.

En la figura 8se muestra el grafo de flujo obtenido con la ejecución del método de caja blanca sobre la funcionalidad *ListarEstablecimiento*.

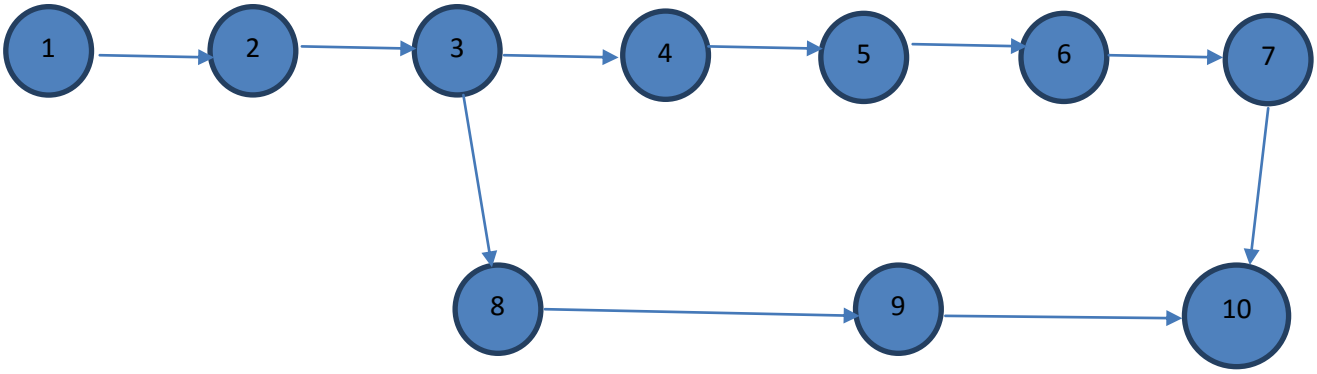

Figura 8 : Grafo de camino básico Del método *ListarEstablecimiento.*

Fuente: elaboración propia.

2) Calcular la complejidad ciclomática:

El valor calculadopor la complejidad ciclomática define el número de rutas independientes del conjuntobásico de un programa y brinda una cota superior para el número de pruebas que se deben realizar a fin de asegurar que todos los enunciados se ejecutaron al menos unavez(Pressman, 2010).

La complejidad ciclomática se calcula de tres formas diferentes, lascuales deben llegar al mismo resultado para comprobar que el cálculo es elcorrecto (Pressman, 2010).

- El número de regiones del grafo de flujo corresponde a la complejidadciclomática.
- La complejidad ciclomática V(G) para un grafo de flujo G se define como:
  V(G)=E-N+2
  Donde E es el número de aristas del gráfico de flujo y N el número de nodos delgráfico de flujo.
- La complejidad ciclomática V(G) para un grafo de flujo G también se definecomo
  V(G)=P+1
  Donde P es el número de nodos predicado (nodos de donde parten al menos dos aristas) contenidos en el grafo de flujo G.

En el grafo de flujo de la figura 8, la complejidad ciclomática puede calcularse usando cada una de las vías anteriormente descritas:

1. El grafo de flujo tiene 2 regiones, por tanto, V(G) = 2
2. V(G) = (10 aristas - 10 nodos) +2 = 2
3. V(G) = 1 nodos predicado + 1= 2

Por tanto, la complejidad ciclomática del grafo de flujo de la figura 8 es dos.

3) Determinar un conjunto básico de caminos linealmente independientes:

El valor de V(G) proporciona la cota superior sobre el número de rutas linealmenteindependientes a través de la estructura de control del programa (Pressman, 2010). En el caso de la funcionalidad *ListarEstablecimiento*, se definen 2 caminos básicos:

Camino básico #1: 1, 2, 3, 4, 5, 6, 7,10

Camino básico #2: 1, 2, 3,8,9,10

4) Obtención de casos de prueba (CP):

Una vez definidos los caminos, se procede a diseñar los casos de prueba para cada uno de los caminos básicos obtenidos. A continuación, en las tablas 5 y 6se presentanlos casos de prueba definidospara los caminos obtenidos con la técnica ruta básica.

Tabla 5 : Caso de prueba # 1.

Caso de prueba para la ruta básica # 1	
Descripción: el método recibe como parámetro una cadena de texto, unnúmero de página, untamaño. En caso de que sea el tamaño distinto de -1 devuelve una lista de establecimientos organizada o no por página	
Entrada:	Una cadena de texto, número de página, un tamaño
Resultados esperados	Una lista organizada por páginas o sin organizar.
Condiciones	

Fuente: **elaboración propia.**

Tabla 6 : Caso de prueba # 2.

Caso de prueba para la ruta básica # 2	
Descripción: el método recibe como parámetro una cadena de texto, unnúmero de página, untamaño. En caso de que sea el tamaño distinto de -1 devuelve una lista de establecimientos organizada por descripción pasada por parámetro	
Entrada:	Una cadena de texto, numero de página, un tamaño
Resultados esperados	Una lista organizada por descripción.
Condiciones	

Fuente: elaboración propia.

Una vez ejecutados todos los casos de pruebas obtenidos con la técnica empleada, se concluye que los mismos fueron probados satisfactoriamente, corrigiéndose los hallazgos surgidos en una primera iteración y comprobándose su corrección en una segunda. Al concluir la prueba sedemuestra que todas las funcionalidades de los módulos propuestosse ejecutan satisfactoriamente, quedando libres de código repetido o innecesario.

3.4 Conclusiones parciales

La aplicación de técnicas para la validación del diseño certifica la obtención de módulos flexibles al mantenimiento y a la introducción de cambios. De igual forma, la realización de pruebas internas en el nivel de unidad aplicando el método de caja blanca, corrobora la ausencia de código repetido o sentencias innecesarias. .

CONCLUSIONES GENERALES

- El estudio de los referentes teóricos vinculados con soluciones informáticas para la gestión de índices de precio, permitió incorporar algunas de sus características a la solución propuesta.
- El empleo de técnicas de captura de requisitos, patrones de diseño y arquitectónicos, así como el uso de estándares de codificación en la implementación de la solución propuesta, permitió obtener una solución, flexible al mantenimiento y a la introducción de cambios.
- El desarrollo de pruebas de internas en el nivel de unidad permitió corroborar la correcta organización y funcionamiento del código implementado, así como la depuración de este a través de la eliminación de códigos innecesarios o repetidos.

RECOMENDACIONES

- Integrar los módulos desarrollos al SIGIP.
- Extender el uso de estos módulos en otros de los subsistemas pendientes a desarrollar para el SIGIP.

BIBLIOGRAFIA REFENCIADA

Pentaho Open Source Business Intelligence. 2013. La plataforma Pentaho Open Source Business Intelligence. *Portada sobre la plataforma Pentaho Open Source Business Intelligence.* [En línea] 2013. http://pentaho.almacen-datos.com/.

Álvarez, Sara. 2007. Sistemas gestores de bases de datos. Introducción a este concepto y características especiales. [En línea] 2007. http://www.desarrolloweb.com/articulos/sistemas-gestores-bases-datos.html.

Association of Modern Technologies Professionals. 2019. It Knowledge portal. [En línea] 2019. http://www.itinfo.am/eng/software-development-methodologies/.

AVOTZ. 2013. Pruebas de Usabilidad. *AVOTZ. Web Works.* [En línea] 2013. www.avotz.com/es/component/content/article/32.html.

Ayala Catari, Iván, Romero Marca, Yecid Tomas y Serrano Urzagaste, Ronald Javier. 2010. Estudio de herramientas CASE de soporte UML y UML2. *Scribd.com.* [En línea] 2010. http://es.scribd.com/doc/25374125/Estudio-de-Herramientas-CASE-de-Soporte-a-UML-y-UML2. DOI: 25374125.

Benchimol , Daniel. 2011. *HACKING DESDE CERO: Manuales Users (Spanish Edition).* Argentina : Creative Andina Corp., 2011. 987177303X, 9789871773039.

C. Evans, Clark . The Official YAML Web Site. *Sitio web oficial de YAML.* [En línea] http://www.yaml.org/.

Carrero , Angel. 2013. Conceptos básicos de ORM (Object Relational Mapping). *Programación en castellano.* [En línea] 2013. http://www.programacion.com/articulo/conceptos_basicos_de_orm_object_relational_mapping_349.

Caswell, Brian, Beale, Jay y Baker, Andrew. 2007. *Snort Intrusion Detection and Prevention Toolkit.* Kurasa : Syngress, 2007. 978-1-59749-099-3.

Cepeda Asqui, Jessica Paulina y Tene Reino, Blanca Georgina . 2012. Investigación de la Herramienta Case para el Desarrollo del Sistema Academico Educativo en el Centro de Edcación Básica “Dr. Nicanor Larrea León, Basada en la Arquitectura .Net Framework. [En línea] 2012. http://hdl.handle.net/123456789/67. DOI: 67.

Chaffer, Jonathan. Drupal programming from an object-oriented perspective. *Drupal.* [En línea] http://drupal.org/node/547518 .

Chile, Consejo Nacional de la cultura y las artes de. 2012. Quiénes Somos. *Cultura.* [En línea] 2012. http://www.cultura.gob.cl/institucion/quienes-somos/.

Claudia. 2010. Drupal versus Joomla. *DrupalSoul.* [En línea] 2010. http://www.drupalsoul.com/blog/drupal-versus-joomla.

Collis, Ta’eed y Harley , Alexander. 2010. *How to be a Rockstar. WordPress Designer.* s.l. : Rockable Press, 2010. 1707351881.

Consejo Nacional de la cultura y las artes de Chile. Quiénes Somos. *Cultura.* [En línea] [Citado el: 20 de noviembre de 2012.] http://www.cultura.gob.cl/institucion/quienes-somos/.

Danysoft. 2013. Embarcadero ER/Studio. [En línea] 2013. http://www.codegear-shop.com/Embarcadero-ER/Studio/es.

del Castillo San Félix, Alvaro. 2000. El servidor de web Apache: Introducción práctica: Apache 1.x y 2.0 alpha. [En línea] 2000. http://acsblog.es/articulos/trunk/LinuxActual/Apache/html/x31.html.

Del Pino Valdarrama, Santiago Luis. 2005. Programación extrema en pocos minutos: planificando la transición. Cuba : Tono. Revista Técnica de la Empresa de Telecomunicaciones de Cuba, S.A., 2005. 3, págs. 41-44. 18135056.

Drupal. 2013. Hooks. *Drupal API.* [En línea] 2013. http://api.drupal.org/api/drupal/includes!module.inc/group/hooks/6.

—. 2013. Smile Open Source Solutions Iberia. *Drupal.* [En línea] 2013. http://www.smile-iberia.com/Productos/Drupal.

Editorbfb. 2011. Qué es un entorno de desarrollo integrado, IDE. *Programación Desarrollo.* [En línea] 2011. http://programaciondesarrollo.es/que-es-un-entorno-de-desarrollo-integrado-ide/.

Ercoli , Jorge . 2007. Qué es un ORM (object-relational mapping) . *Arquitectura de Sistemas.* [En línea] 2007. http://metodologiasdesistemas.blogspot.com/2007/10/que-es-un-orm-object-relational-mapping.html.

Escalante, Lain Cárdenas. 2014. *El patrón de arquitectura n-capas con orientación al dominio.* 2014.

Figueroa, Pablo. 2013. Conceptos en un Diagrama de Implementación. [En línea] 2013. http://webdocs.cs.ualberta.ca/~pfiguero/soo/uml/implementacion01.html.

Flanagan, David. 2011. *JavaScript : the definitive guide. 6th Edition.* Beijing : Sebastopol, CA : O'Reilly, 2011. 9780596805524; 0596805527.

Gauchat, Juan Diego. 2012. *El gran libro de HTML5, CSS3 y JavaScript.* Barcelona : MARCOMBO, S.A., 2012. 978-84-267-1770-2.

González Boticario, Jesús y Gaudioso Vázquez, Elena. 2001. *"Capítulo 6. JavaScript". Aprender y formar en Internet.* Madrid, España : International Thomson Editors Spain Paraninfo, S. A., 2001. 84-283-2743-2.

Guardado, Iván. 2010. Utilizando Doctrine como ORM en PHP. *Web.ontuts.* [En línea] 2010. http://web.ontuts.com/tutoriales/utilizando-doctrine-como-orm-en-php/.

Hommel, Scott. 2019. *Estandares_de_codificacion_para_Java.* 2019.

ICOM. 2013. *Código de Deontología del ICOM para los Museos.* s.l. : © ICOM, 2013. 978-92-9012-407-8.

Ing. Oré , Alexander. 2009 . UNIT TESTING - PRUEBAS UNITARIAS. *CalidadySoftware.com .* [En línea] 2009 . http://www.calidadysoftware.com/testing/pruebas_unitarias2.php .

Ing. Valdarrama del Pino, Santiago Luis . 2005. Programación extrema en pocos minutos: planificando la transición. *Revista Técnica de la Empresa de Telecomunicaciones de Cuba S.A.* Cuba : s.n., 2005. 3. 18135056. DOI: 28012845.

2017. Instinto Binario. [En línea] 2017. https://instintobinario.com/patrones-de-diseno-patron-observador/.

INTECO. 2009. Ingeniería del software: Metodologías y ciclos de vida. *Scientific Electronic Librery Online (Scielo).* [En línea] 2009. http://www.inteco.es/file/N85W1ZWFHifRgUc_oY8_Xg.

—. 2009. Revista Scielo. *INGENIERÍA DEL SOFTWARE: METODOLOGÍAS Y CICLOS DE VIDA.* [En línea] Marzo de 2009. http://www.ieee.org.sv/concapan/descargas/memoria_secciones/Jueves_10/izalc/P72.pdf.

Jacobson, Ivar , Booch, Grady y Rumbaugh, James. 2000. *El Proceso Unificado de Desarrollo de Software.* Madrid : Pearson Educación, Inc., 2000. 84-7829-036-2.

Jetbrains. 2018. [En línea] 2018. https://www.jetbrains.com/webstorm/.

JetBrains. 2018. [En línea] 2018. https://www.jetbrains.com/idea/.

Kendall, Kenneth E. y Kendall, Julie E. 2005. *Análisis y Diseño de Sistemas. Sexta Edición.* México : Pearson Educación de México, S.A. de C.V., 2005. 970-26-0577-6.

Kniberg, Henrik. 2007. *Scrum and XP from the Trenches.* Estados Unidos de América : C4Media Inc., 2007. 978-1-4303-2264-1.

Larman, Craig. 1999. *UML y Patrones. Introducción al análisis y diseño orientado a objetos.* México : PRENTICE HALL, 1999. 970-17-0261-1.

—. 2016. *UML y Patrones. Una introducción al análisis y diseño orientado a objetos y al proceso unificado.* 3ra . s.l. : Prentice-Hall, 2016.

Leffingwell, Dean y Widrig, Don. 2003. *"Using Software Engineering Techniques for Business Modeling. The Unified Modeling Language". Managing software requirements : a use case approach.* United States of America : Pearson Education, Inc., 2003. 0-321-12247-X.

Leyva Samada, Lisandra Isabel . 2009. Flujo de Investigación para la Metodología Ágil SXP. Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas. [En línea] 2009. http://repositorio_institucional.uci.cu/jspui/handle/ident/TD_2435_09. TD_2435_09.

Lorenz, M., & Kidd, J. 1994. *Object-oriented software metrics: a practical guide.* New Jersey, Prentice-Hall : s.n., 1994.

Martin. 1999. 1999.

—. 1999. 1999.

Martín Fernández, Francisco J. y Hassan Monter, Yusef. 2003. Revista No Solo Usabilidad. Qué es la Arquitectura de la Información. 2003. Vol. nº 2. 1886-8592.

Martin Olivera, Yonnys Pablo y Zamora Sanchez, Gey. 2016. *ENTORNO DE DESARROLLO INTEGRADO LIBRE Y MULTIPLATAFORMA PARA DESARROLLAR SOFTWARE EDUCATIVO EN FORMATO MULTIMEDIA.* 2016.

Martínez Illa, Santi y Mendoza, Roser . 2007. TIC y gestión de la cultura: ¿Políticas e-culturales? *Centro de Estudios y Recursos Culturales. Universidad de Alicante.* [En línea] 2007. http://www.google.com.cu/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=0CB4QFjAA&url=http%3A%2F%2Fcolombiadigital.net%2Fnewcd%2Fcomponent%2Fdocman%2Fdoc_download%2F302-tic-y-gestion-de-la-cultura-ipoliticas-e-culturales-&ei=7U-oUKfdH_SC0QHsvICYDA&usg=AFQjC.

Megías, Braulio. 2011. ¿Claves primarias naturales o subrogadas? *reThink.net.* [En línea] 2011. http://bmegias.wordpress.com/2011/01/31/%C2%BFclaves-primarias-naturales-o-subrogadas/.

Microbuffer. 2011. [En línea] 2011. https://microbuffer.wordpress.com/2011/05/04/que-es-.

Miles, Russ y Hamilton, Kim . 2006. *Learning UML 2.0.* s.l. : O'Reilly, 2006. 978-0-59-600982-3.

MinCult. 2012. Consejo Nacional de Patrimonio Cultural (CNPC). *Sitio oficial del Ministerio de Cultura de la República de Cuba.* [En línea] 2012. [Citado el: 9 de Noviembre de 2012.] http://www.min.cult.cu/loader.php?sec=instituciones&cont=cnpc.

—. 1983. *Decreto No. 118. Reglamento para la Eejecución de la Ley de Protección al Patrimonio.* La Habana : Gaceta Oficial de la República de Cuba, 1983.

—. 2009. *Ley No. 106. Ley del Sistema Nacional de Museos de la República de Cuba.* La Habana : Gaceta Oficial de la República de Cuba, 2009.

MSc. García Perdigón, Jorge R., y otros. 2009. Manual sobre el trabajo técnico de los museos adscritos al Consejo Nacional de Patrimonio Cultural. *Sitio oficial del Consejo Nacional de Patrimonio Cultural.* [En línea] 2009. http://www.cnpc.cult.cu/Portada/Manual_de_museos.pdf.

Muñoz Serafin, Miguel. 2018. *Introducción al desarrollo de aplicaciones N-Capas con tecnologías Microsoft.* 2018.

Netbeans.org. 2012. Bienvenido a NetBeans y www.netbeans.org. *NetBeans.* [En línea] 2012. http://netbeans.org/index_es.html.

Ochoa, Javier. 2018. 2018.

OMG®. 2012. Introduction to OMG's Unified Modeling Language™ (UML®). *OMG®.* [En línea] 2012. [Citado el: 8 de Enero de 2012.] http://www.omg.org/gettingstarted/what_is_uml.htm.

Pacheco, Juan Carlos Ruiz. 2017. Inyección de Dependencias. [En línea] 2017. https://msdn.microsoft.com/es-es/communitydocs/net-dev/csharp/inyeccion-de-dependencias.

Peck, Steven. 2013. Understanding Drupal. *Drupal.* [En línea] 2013. http://drupal.org/documentation/understand.

Pentaho Data Integration. 2013. Pentaho Data Integration Kettle ETL tool. *ETL-Tools.Info.* [En línea] 2013. [Citado el: 26 de Marzo de 2013.] http://etl-tools.info/en/pentaho/kettle-etl.htm.

—. 2013. Proceso ETL. *ETL-Tools.Info.* [En línea] 2013. [Citado el: 26 de Marzo de 2013.] http://etl-tools.info/es/bi/proceso_etl.htm.

Pérez, Felipe U. 2018. La Revista Informática.com. *La Revista Informática.com.* [En línea] 2018. http://www.larevistainformatica.com/Java.htm.

PETKOVIĆ, DUŠAN. 2005. *Microsoft SQL Server .* 2005.

Potes, Julio Escobar. 2015.. *“METODOS DE CONSTRUCCION DE INDICES DE PRECIOS DE VIVIENDA.* 2015.

Pressman, Roger S, Ph.D. 2010. *Ingenieria de Software. Un enfoque práctico. Septima Edición.* Mexico, D.F : The Me Graw Hill Companies, 2010.

Pressman, Roger S. 2010. *Ingeniería de Software. Un enfoque práctico.* Septima . México DF : McGraw-Hill INTERAMERICA EDITORES, 2010.

—. 2003. *Ingeniería del Software. Un enfoque práctico. Sexta Edición.* s.l. : Mc Graw Hill, 2003. 970-10-5473-3.

—. 2001. Ingeniería del Software: una tecnología estratificada. *Ingeniería del Software. Un enfoque práctico. Quinta Edición.* España : McGraw Hill, 2001.

Prieto Díaz, Vicente, y otros. 2010. Impacto de las tecnologías de la información y las comunicaciones en la educación y nuevos paradigmas del enfoque educativo. *Biblioteca Virtual en Salud.* [En línea] 2010. http://bvs.sld.cu/revistas/ems/vol25_1_11/ems09111.htm. 09111.

Puertas Ortega, Juan y Orellana Zubieta, Francisco Javier . 2011. Un-paseo-por-PHP. *Scribd.com.* [En línea] 2011. http://es.scribd.com/doc/51830143/Un-paseo-por-PHP. DOI: 51830143.

Raggett, Dave, Le Hors, Arnaud y Jacobs, Ian (Eds.). 2001. Especificación HTML 4.01. *ucistore.* [En línea] 2001. ftp://ucistore.uci.cu/documentacion/Programacion/HTML/HTML%201/html401-es.pdf.

Ramírez, T. 1999. *Proyecto de Investigación. .* Caracas: Panapo : s.n., 1999.

Roche, Emilio. 2013. /DEV/Blog. [En línea] 2013. https://barradevblog.wordpress.com/2013/04/23/el-patron-repositorio-repository-pattern-implementacion-practica-con-entity-framework/.

Rodríguez Sala, Jesús Javier . 2003. *Introducción a la programación: teoría y práctica.* s.l. : Editorial Club Universitario, 2003. 9788484542742.

Rodríguez, Fran Gil. 2012. *Experto en Drupal 7. Curso de creación y gestión de portales web con Drupal 7. Nivel Avanzado. Aprende Drupal con Forcontu.* s.l. : Forcontu S.L., 2012. 978-84-939410-5-5.

—. 2012. *Experto en Drupal 7. Curso de creación y gestión de portales web con Drupal 7. Nivel Inicial. Aprende Drupal con Forcontu.* s.l. : Forcontu S.L., 2012. 978-84-939410-3-1.

—. 2012. *Experto en Drupal 7. Curso de creación y gestión de portales web con Drupal 7. Nivel Intermedio. Aprende Drupal con Forcontu.* s.l. : Forcontu S.L., 2012. 978-84-939410-4-8.

Rodríguez, Txema. 2017. GENBETA. *GENBETA.* [En línea] diciembre de 2017. https://www.genbeta.com/desarrollo/por-que-deberiamos-abandonar-rest-y-empezar-a-usar-graphql-en-nuestras-apis.

Rojas, M. J. 2010. *Patrones de Diseño.* 2010.

Rojas, Nohemi. 2010. CMS y Framework dos conceptos distintos. [En línea] 2010. http://nohemirojas.wordpress.com/2010/03/25/cms-y-framaework-dos-conceptos-distintos/.

Romero, Gladys Marsi Peñalver. 2008. MA-GMPR-UR2 Metodología ágil para proyectos de software libre. Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas. [En línea] 2008. http://repositorio_institucional.uci.cu/jspui/handle/ident/TD_1309_08. TD_1309_08.

Rouse, Margaret. 2019. TechTarget. *User Story.* [En línea] 2019. https://searchsoftwarequality.techtarget.com/definition/user-story.

Rumbaugh, James , Jacobson, Ivar y Booch, Grady . 2007. *El Lenguaje Unificado de Modelado. Manual de Referencia.* s.l. : Addison-Wesley Iberoameri, 2007. 9788478290871.

Rumbaugh, James , Jacobson, Ivar y Booch, Grady . 2004. Unified Modeling Language Reference Manual, The (2nd Edition). [En línea] 2004. http://my.safaribooksonline.com/book/software-engineering-and-development/uml/0321245628/background/part01. 0321245628.

Sæther Bakken, Stig, Aulbach, Alexander y Schmid, Egon . 2001. Manual de PHP. *ucistore.uci.cu.* [En línea] 2001. ftp://ucistore.uci.cu/documentacion/Programacion/PHP/Manual%20de%20PHP.pdf.

Sánchez Maza, Miguel Ángel . 2012. *JavaScript.* s.l. : IC Editorial, 2012. 978-8495733184.

Sánchez, Tamara Rodríguez. 2015. *Metodología de desarrollo para la Actividad productiva de la UCI. .* La Habana. Cuba : s.n., 2015.

Schardt Chonoles, Michael Jesse y Schardt, James A. 2003. *UML 2 for Dummies.* U.S.A : Wiley Publishing, Inc., 2003. 0764526146.

Sierra, Manuel. 2012. Qué es y para qué sirve el lenguaje CSS (Cascading Style Sheets - Hojas de Estilo). *Aprenderaprogramar.com.* [En línea] 2012. http://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=546:que-es-y-para-que-sirve-el-lenguaje-css-cascading-style-sheets-hojas-de-estilo&catid=46:lenguajes-y-entornos&Itemid=163).

Software-talk.org. 2012. Netbeans vs Eclipse: An IDE comparison. *Software Talk.* [En línea] 2012. http://software-talk.org/blog/2012/01/netbeans-vs-eclipse-an-ide-comparison/.

Sommerville. 2011. *Ingeniería de Software, 9na Edición.* 2011.

Sommerville, Ian. 2005. *Ingeniería del software. Séptima Edición.* Madrid. España : Pearson Educación. S. A., 2005. 84-7829-074-5.

The PostgreSQL Global Development Group. 2019. PostgreSQL. *PostgreSQL.* [En línea] 2019. https://www.postgresql.org/docs/10/release-10-1.html.

Tinoco Gómez, Oscar, Rosales López, Pedro Pablo y Salas Bacalla, Julio . 2010. *Criterios de selección de metodologías de desarrollo de software .* Perú : Industrial Data. Revista de la Facultad de Ingeniería Industrial, 2010. 1810-9993.

Venemedia. 2014. [En línea] 2014. http://conceptodefinicion.de/javascript/.

Vera, H. Solano. 2005. Definición y diseño de la aplicación web Interfaz para el monitoreo de redes de comunicaciones mediante una aplicación web. Tesis Licenciatura Ingeniería en Sistemas Computacionales. *Departamento de Ingeniería en Sistemas*

Computacionales. [En línea] 2005. http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/solano_v_h/capitulo_4.html.

Vértice, Equipo. 2009. *Diseño Básico de Páginas Web en HTML.* España : Publicaciones Vértice S. L, 2009. 978-84-9931-034-3.

Visual-Paradigm. 2019. Visual-Paradigm. [En línea] 2019. www.visual-paradigm.com/solution/freeumltool/.

W3C. 2013. Guía Breve de Servicios Web. *W3C.es.* [En línea] 2013. http://www.w3c.es/Divulgacion/GuiasBreves/ServiciosWeb.

—. 2005. Introducción a la Accesibilidad Web. *W3C.es.* [En línea] 2005. http://www.w3c.es/Traducciones/es/WAI/intro/accessibility.

Wolanin, Peter. 2012. Is Drupal secure? *Drupal.* [En línea] 2012. http://drupal.org/documentation/is-drupal-secure .

Young, Ralph R. 2004. *Chapter 1. The Importance of Requirements: What Are Requirements and Why Are They Important? The Requirements Engineering Handbook.* Boston : Artech House Inc., 2004. 1-58053-266-7.

Zamitiz, Ing. Carlos Alberto Román. 2016. Java Basico. *Java Basico.* [En línea] 2016.

i