Universidad de las Ciencias Informáticas
Facultad 3

Componente para la obtención de reportes en el Sistema de Registro de Entidades para la Oficina Nacional de Estadística e Información

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

Autor:

Carlos Félix Pentón Martínez

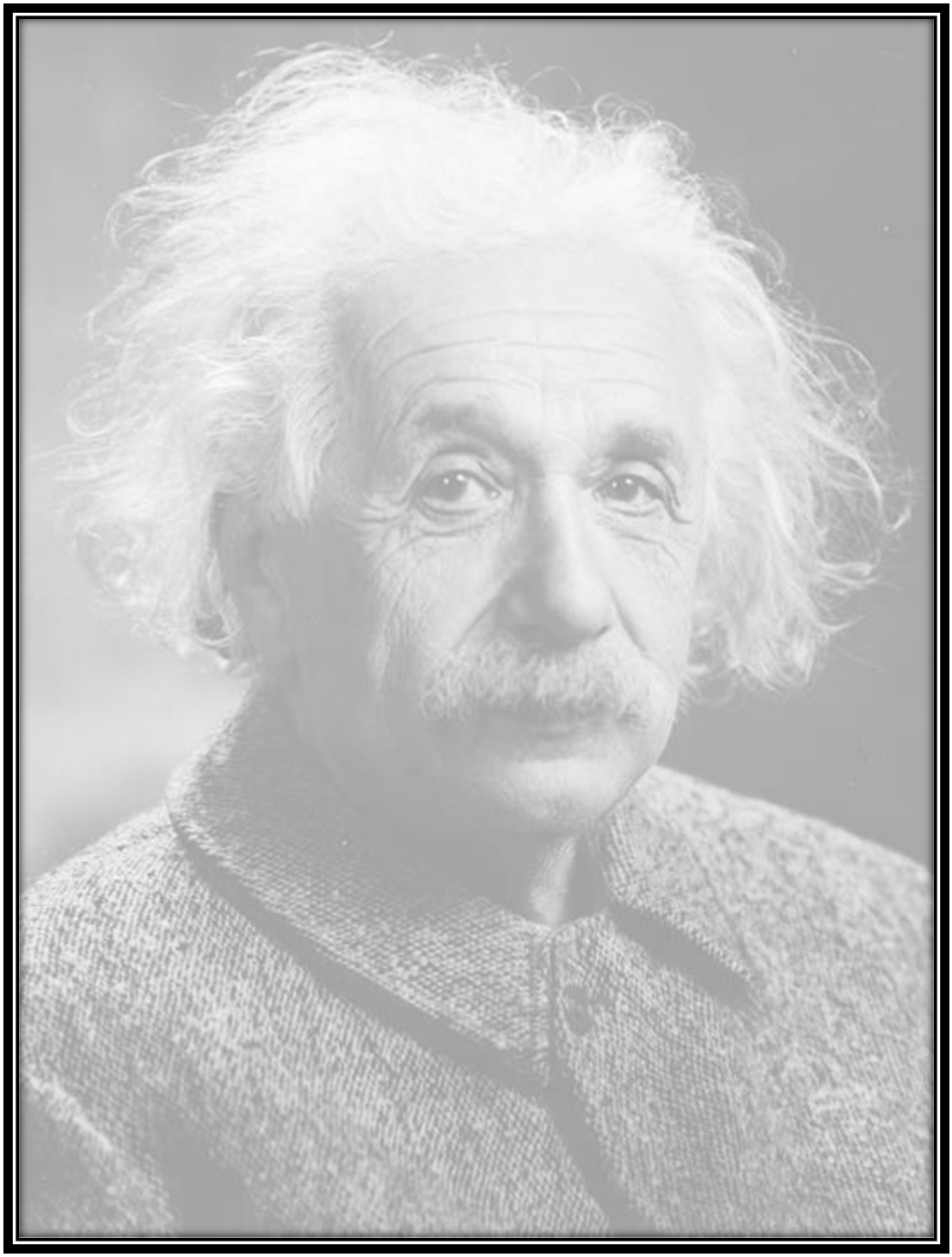
Tutores:

MSc. Yordanis García Leiva

Ing. Ángel Miguel Morciego Lezcano

La Habana, 2019

"Año 61 de la Revolución"



"El MUNDO COMO LO HEMOS CREADO ES UN PROCESO DE NUESTRO PENSAMIENTO. NO PUEDE SER CAMBIADO SIN CAMBIAR NUESTRO PENSAMIENTO"

ALBERT EINSTEIN
1879-1955

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor de este trabajo y autorizo 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 2019.

_____

Firma del autor

Carlos Félix Pentón Martínez

_____ _____

Firma del tutor **Firma del tutor**

MSc. Yordanis García Leiva Ing. Ángel Miguel Morciego Lezcano

AGRADECIMIENTOS

A mi madre Margarita.

A mis abuelas Xiomara, Ofelia, Mercedes.

A mis hermanos Omar Alejandro, Daniel Alejandro, Erika, José Carlos.

A mis tutores Yordanis, Angel Miguel, Rolando, Reinier.

A mis grandes amigos de la universidad.

A mis compañeros de aula.

DEDICATORIA

A mi madre Margarita y a mis abuelas Xiomara, Ofelia y Mercedes.

A mis hermanos Omar Alejandro, Daniel Alejandro, Erika y José Carlos.

RESUMEN

El Sistema de Registro de Entidades constituye una aplicación web para la Oficina Nacional de Estadística e Información. Su propósito es informatizar los procesos que realiza la dirección de metodología de la información en esta oficina, relacionados con el registro de las Entidades estatales y no estatales que existen en Cuba. La aplicación cuenta con un conjunto de funcionalidades que viabilizan el trabajo de los especialistas de esta dirección, pero aún no brinda la posibilidad de obtener información oportuna sobre las estadísticas que generan los registros realizados a través de este software. La presente investigación tiene como objetivo desarrollar un componente que se integre al software antes descrito y brinde la posibilidad a los usuarios de generar reportes con datos estadísticos sobre las Entidades registradas. Por otra parte, se debe poder visualizar en tiempo real estos datos mediante una pizarra de control, que orienten al usuario sobre el comportamiento del registro de Entidades. 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 fue validado aplicando pruebas de software en los niveles de unidad, integración y sistema. Además, se verifica el cumplimiento del objetivo general de la investigación a partir de la definición de un conjunto de indicadores que permiten evaluar la relación causa-efecto de la variable independiente sobre las variables dependientes de la investigación.

PALABRAS CLAVE: componente, Entidades, registro, reportes

ÍNDICE 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. Esta contribuye a satisfacer las necesidades informativas relacionadas con los objetivos y planes del Gobierno en todos los niveles de dirección, en los ámbitos: económico, social, demográfico, y medioambiental (ONEI, 2019). La ONEI tiene entre sus objetivos garantizar los servicios de la información de interés para el Gobierno, organismos internacionales y personas jurídicas. En correspondencia con este objetivo la oficina presenta la Dirección de Metodología de la Información (DMI), área en la cual se gestionan todos los Registros de Unidades Institucionales y Establecimientos existentes en el país.

La DMI cuenta con un sistema informático que gestiona cada uno de estos registros. Sin embargo, a pesar de las facilidades que brinda este software, no cuenta con niveles de usuario que regulen el acceso a la información que se gestiona. También carece de mecanismos que permitan controlar los datos de las personas que introducen cambios en el sistema y dar seguimiento al historial de acciones que realicen. Con el propósito de hacer frente a las limitantes antes descritas, un grupo de especialistas del Centro de Gobierno Electrónico de la Universidad de las Ciencias Informáticas se ha dado a la tarea de desarrollar un nuevo software para esta dirección de la ONEI. La solución se denomina Sistema de Registro de Entidades (SIRES) y tiene entre sus objetivos: gestionar la inscripción de los sujetos que desarrollan actividades económicas en el país. Así como, anotar y mantener actualizados los actos y circunstancias relacionadas con los movimientos organizativos de cada registro realizado en la DMI.

El SIRES a pesar de las funcionalidades que presenta, carece de mecanismos que garanticen a la DMI la obtención de información oportuna relacionada con las estadísticas generadas en el proceso de gestión de Entidades. En la actualidad los datos estadísticos que se generan en este proceso se calculan de forma manual. Lo anterior dificulta la entrega en tiempo de informaciones solicitadas por ministerios y Entidades nacionales, interesados en conocer sobre el comportamiento del proceso de registros estatales y no estatales. Estas solicitudes en la mayoría de los casos vienen acompañadas de varias especificaciones. Como el procesamiento de la información se realiza manualmente, la obtención de esta puede tardar un poco. Además, el resultado final está propenso a errores de cálculos o cambio de algún dígito, que alteran la veracidad en la información obtenida.

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

¿Cómo obtener información desde el SIRES de forma tal que se reduzca el tiempo y la posibilidad de errores, en el procesamiento de los datos estadísticos que genera el proceso de gestión de Entidades en la ONEI?

Tomando en cuenta el problema antes propuesto se define como **objeto de estudio:** soluciones para la generación de reportes.

Determinándose como **objetivo general:** desarrollar un componente para la obtención de reportes desde el SIRES, que reduzca el tiempo y la posibilidad de errores en el procesamiento de los datos estadísticos que genera el proceso de gestión de Entidades en la ONEI.

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 componentes informáticos dirigidos a la obtención de reportes estadísticos.
- Realizar la identificación de los requisitos, análisis, diseño e implementación del componente propuesto.
- Validar el diseño y el funcionamiento del componente propuesto 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:** componente para la generación de reportes relacionados con el proceso de gestión de Entidades en la ONEI.

Definiéndose como **idea a defender:** si se desarrolla un componente para la obtención de reportes desde el SIRES, se reducirá el tiempo y la posibilidad de errores en el procesamiento de los datos estadísticos que genera el proceso de gestión de Entidades en la ONEI.

Métodos teóricos:

Histórico-Lógico: permitió realizar un estudio sobre la evolución de soluciones web destinadas al procesamiento, análisis y publicación de datos estadísticos, en el ámbito nacional e internacional. El estudio se desarrolló con el propósito de obtener algunas de las características de estas soluciones e incorporarlas a la solución propuesta.

Analítico-Sintético: se aplicó en la realización del diseño teórico de la investigación, extrayéndose los elementos más importantes a tener en cuenta para desarrollar el componente propuesto, durante el análisis de la bibliografía estudiada.

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:

Entrevista: fue utilizado para comprender las necesidades del cliente, capturar los requisitos correspondientes al componente y obtener información que apoye la realización de la investigación. En el Anexo 1, se encuentra la guía de preguntas utilizada en el desarrollo de la entrevista.

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

Capítulo 1: Fundamentación teórica. En este capítulo se describen una serie de conceptos relacionados con el objeto de estudio de la presente investigación, así como las características fundamentales de los sistemas para la gestión estadística que existen en diversas regiones del mundo incluyendo Cuba. Además, se describe la metodología seleccionada para guiar el proceso de desarrollo de la presente investigación, junto a los lenguajes y herramientas utilizadas, así como los patrones de diseño.

Capítulo 2: Descripción de la solución propuesta. En este capítulo se realiza una descripción de las principales características del componente para la obtención de reportes en el SIRES. En el mismo se describe el diseño de la solución propuesta, a partir de las disciplinas de la metodología definida para guiar el desarrollo de esta. Entre los contenidos analizados se encuentra el proceso de captura de los requisitos funcionales (RF) y no funcionales (RnF) con los que debe cumplir el componente propuesto. Además, se describe la arquitectura de la solución, el diagrama de clases del diseño, el modelo de datos y los patrones de diseño utilizados. Por otra parte, se exponen los estándares de codificación empleados en la disciplina de implementación y se realiza una descripción de las principales interfaces de la solución obtenida.

Capítulo 3: Validación y verificación de la solución propuesta. En el capítulo se muestran los resultados obtenidos luego de aplicar las técnicas de validación de requisitos, así como las métricas para la validación del diseño de la solución propuesta. En cada caso se documentan los resultados obtenidos. 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

que en cada disciplina se realicen pruebas a nivel de unidad, integración y sistema, con la aplicación de sus respectivos métodos y técnicas. El capítulo concluye con la verificación del cumplimiento del objetivo general de la investigación a partir de la definición de un conjunto de indicadores que permiten evaluar a través de un antes y un después al desarrollo del componente, la relación causa-efecto de la variable independiente sobre las variables dependientes de la investigación.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1. Introducción

En este capítulo se describen una serie de conceptos relacionados con el objeto de estudio de la presente investigación, así como las características fundamentales de los sistemas para la gestión estadística que existen en el ámbito nacional e internacional. Además, se describe la metodología seleccionada para guiar el proceso de desarrollo de la presente investigación, junto a los lenguajes y herramientas utilizadas, así como los patrones de diseño.

1.2. Conceptos fundamentales asociados al dominio del problema

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

Reporte estadístico: documento que analiza a través de cuadros y gráficos estadísticos la evolución de las principales variables de una institución, con el propósito de ayudar a la toma de decisiones empresariales (SNI, 2019).

Pizarra de control (*Dashboard*): es una representación gráfica de los principales indicadores que intervienen en la consecución de los objetivos de negocio. Está orientado a la toma de decisiones para optimizar la estrategia de la empresa. Un *dashboard* debe transformar los datos en información y esta, en conocimiento para el negocio (Klipfolio Inc, 2019).

El autor de la presenta investigación asume las definiciones anteriores teniendo en cuenta que, del estudio bibliográfico realizado, estas son las que están en correspondencia con el aporte práctico que se realiza en la tesis.

1.3. Valoración de sistemas homólogos

Con el propósito de obtener experiencia en cuanto a la organización y estructuración de software que manipulan información estadística, se estudiaron soluciones a nivel nacional e internacional. Este estudio se realizó en los portales: Instituto Nacional de Estadística e Informática (INEI) del Perú, Instituto Nacional de Estadística, Geografía e Informática (INEGI) de México, el portal web CEPALSTAT de acceso a la información estadística de los países de América Latina y el Caribe, así como el Sitio web de la ONEI. A continuación, se describen cada una de estas:

El Instituto Nacional de Estadística e Informática (INEI): es el órgano rector del Sistema Estadístico Nacional en el Perú, norma, planea, dirige, coordina, evalúa y supervisa las actividades estadísticas

oficiales del país. Produce y difunde información estadística oficial en forma integrada, coordinada, racionalizada y bajo una normatividad técnica común, con el propósito de contribuir al diseño, monitoreo y evaluación de políticas públicas y al proceso de toma de decisiones. Perú cuenta con un ágil y eficiente sistema nacional de coordinación, producción y difusión de información estadística confiable, oportuna y de calidad, con cobertura de datos desagregada a todo nivel político-administrativo, que contribuye eficazmente al diseño, implementación y evaluación de políticas públicas, programas y proyectos de desarrollo que impactan en el crecimiento económico, reducción de la pobreza y conservación ambiental. Este sistema satisface plenamente los requerimientos de los usuarios del sector público y privado. La información estadística es de fácil acceso y su producción y difusión se realiza con el uso intensivo de la tecnología de información más avanzada. (deperu, 2019)

Instituto Nacional de Estadística y Geografía (INEGI) de México: se creó el 25 de enero de 1983 por decreto presidencial, que desde sus inicios integró en su estructura a:

- La Dirección General de Estadística, en funciones desde 1882, cuando pertenecía a la Secretaria de Fomento, Colonización, Industria y Comercio.
- La Dirección General de Geografía, establecida en 1968 y que estaba adscrita a la Secretaría de la Presidencia.
- La Dirección General de Política Informática.
- La Dirección General de Integración y Análisis de la Información.

Con su creación, el INEGI modernizó la valiosa tradición que tenía el país en materia de captación, procesamiento y difusión de información acerca del territorio, la población y la economía. Unificó en una sola institución la responsabilidad de generar la información estadística y geográfica (inegi, 2019).

CEPALSTAT: el portal web está organizado en cuatro grandes temas de manera similar a las clasificaciones internacionales de actividades estadísticas, incluyendo algunos temas transversales los cuales, dada su relevancia en la coyuntura actual, ameritan una visibilidad mayor para el usuario interno y externo.

- Estadísticas e indicadores demográficos y sociales.
- Estadísticas e indicadores económicos.
- Estadísticas e indicadores ambientales
- Estadísticas e indicadores de temas transversales (objetivos del desarrollo del milenio de las Naciones Unidas, género cohesión social, poblaciones y pueblos indígenas y afrodescendientes, desarrollo sostenible, tecnologías de información y comunicación, sector agropecuario de Centroamérica y México.)

El portal web CEPALSTAT permite acceder a la información disponible a través de diversos mecanismos de procesamiento en línea, de naturaleza variada y con características propias de las Divisiones de la Comisión Económica para América Latina (CEPAL) responsables de su generación y actualización. Una vez seleccionado un tema en particular, el usuario podrá identificar el área de su interés y generar el resultado de la consulta realizada siguiendo las instrucciones y opciones presentadas en el menú respectivo (eclac, 2019).

Sitio web de la Oficina Nacional de Estadística e Información: en el caso de Cuba, se encuentra el sitio web de la Oficina Nacional de Estadística e Información, sitio que tiene como principal desventaja que no cuenta con funcionalidades de generación de reportes estadísticos que viabilicen el trabajo a especialistas en esta área.

Las características de estos sistemas demuestran la necesidad de desarrollar el componente propuesto en la presente tesis, teniendo en cuenta que ninguno de estos puede utilizarse para dar respuesta a la problemática que dio origen a la investigación. Además la mayoría no presenta funcionalidades que permitan obtener información oportuna para la generación de reportes estadísticos.

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 define 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).

Como metodología de desarrollo de software, se decide utilizar la definida en la UCI para guiar el proceso productivo. Esta es una variación del Proceso Unificado Ágil, adaptada al ciclo de vida de los proyectos en la UCI. A continuación, se describen los principales elementos de esta metodología:

1.4.1. Metodología AUP-UCI

Dentro de las metodologías ágiles se encuentra el Proceso Unificado Ágil (*AUP*, por sus siglas en inglés), la cual consiste en una versión simplificada de la metodología de desarrollo tradicional Proceso Racional Unificado (RUP). *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.

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 elaboran 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.

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

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. Los flujos de trabajos: modelado de negocio, requisitos y análisis y diseño en *AUP* están unidos en la disciplina modelo. 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

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

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.

A partir del análisis realizado sobre la variación de la metodología *AUP* para la UCI, el autor de la presente tesis decide organizar el proceso de desarrollo del componente que da cumplimiento al objetivo general de la investigación a partir de las fases y disciplinas que propone esta variación de la metodología. La decisión se adopta teniendo en cuenta que el desarrollo del componente responde a un proyecto de la red de centros de la universidad.

El escenario a utilizar es el 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. Herramienta de Ingeniería del Software Asistida por Computadora

Las herramientas *CASE* (por sus siglas en inglés *Computer Aided 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 15.1

Es una herramienta que utiliza lenguaje unificado de modelado (*UML*, por sus siglas en inglés) que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Permite modelar todos los tipos de 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 el autor de la presente tesis decide 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 Universidad de las Ciencias Informáticas cuenta con una licencia para su uso.

1.6. Marcos de trabajo

Desde el punto de vista del desarrollo de software, un marco de trabajo es una estructura de soporte definida, en la cual otro proyecto de software puede ser organizado y desarrollado (Alegsa, 2019). Para el desarrollo de la solución propuesta se hace necesario utilizar los marcos de trabajo *Angular* y *Spring Boot* en sus versiones 5.0 y 1.5 respectivamente, con el propósito de lograr la compatibilidad del componente con el Sistema de Registro de Entidades, el cual está implementado sobre estas tecnologías.

1.6.1. Angular 5.0

Es un marco de trabajo de *JavaScript* de código abierto desarrollado por Google**.** Contiene un conjunto de librerías útiles para el desarrollo de aplicaciones web y propone una serie de patrones de diseño para llevarlas a cabo. Además, contiene todas las herramientas que los creadores ofrecen para que los desarrolladores sean capaces de crear un *HTML* (Lenguaje de Marcados para Hipertextos, por sus siglas en inglés *HyperText Markup Language)* enriquecido (Google, 2019).

1.6.2. Spring Boot 1.5

Spring Boot es una infraestructura ligera que elimina la mayor parte del trabajo de configurar las aplicaciones basadas en *Spring*, el mismo facilita la creación de proyectos con *framework Spring* eliminando la necesidad de crear largos archivos de configuración *XML* (siglas en inglés de *eXtensible Markup Language*). Además, provee configuraciones por defecto para Spring y otras librerías. También provee un modelo de programación parecido a las aplicaciones *java* tradicionales que se inician en el método principal "*main*" (Pivotal, 2019).

1.7. Herramienta para la generación de reportes

Los reportes constituyen informes que organizan y exhiben la información contenida en una base de datos. Su función es aplicar un formato determinado a los datos para mostrarlos por medio de un diseño atractivo y fácil de interpretar por los usuarios (JasperSoft, 2019).

En la actualidad el diseño e implementación de reportes se viabiliza con el uso tecnologías destinadas a estos fines. Para el desarrollo de la presente investigación se emplea la herramienta *JasperReports*. El autor decide utilizar la misma, teniendo en cuenta que es la tecnología definida para el diseño y

generación de reportes en el Departamento Desarrollo de Componentes del Centro de Gobierno Electrónico (CEGEL), al cual responde el desarrollo de la solución.

1.7.1 JasperReports 6.0

JasperReports es una biblioteca de creación de informes que tiene la habilidad de entregar contenido enriquecido al monitor, a la impresora o a ficheros en formato *PDF*, *HTML*, *XLS*, *CSV* y *XML*. Está escrito completamente en *Java* y puede ser usado en gran variedad de aplicaciones de *Java*, incluyendo *J2EE* o aplicaciones web, para generar contenido dinámico.

Las principales características de *JasperReports* son:

- Permitir una diagramación flexible de los reportes: los reportes se pueden dividir en secciones opcionales que son, el título del reporte y el encabezado de página, una sección para los detalles del reporte, el pie de página y una sección de resumen que aparece al final del reporte.
- Permite que los desarrolladores le surtan datos en varias formas, es decir, los desarrolladores permitan el paso de los datos a los reportes por medios distintos parámetros. Estos parámetros de reportes pueden ser instancia de cualquier clase de *Java*. Los reportes son capaces de presentar los datos de manera textual o a través de gráficos, no sólo son capaces de mostrar los datos que le son pasados, sino que pueden generar o calcular con esos datos otros datos de forma dinámica y mostrarlos.
- No es una herramienta por sí sola, por lo que no se puede instalar. Para utilizar *JasperReports* es necesario añadirlo a las aplicaciones *Java* por medio de la inclusión de su librería al *classpath* (por su traducción al español, ruta de la clase) de la aplicación. En cuanto a las especificaciones de sistemas operativos se puede utilizar en cualquier entorno, siempre y cuando exista una implementación de la máquina virtual de *Java*. Su licencia se distribuye bajo los términos de la Licencia Pública para Librerías *GNU*[2], por lo que es software libre y está respaldado por una gran comunidad internacional de desarrollo (JasperSoft, 2019).

1.8. Lenguajes de programación

De los lenguajes de programación que existen para desarrollar aplicaciones orientadas a la web se encuentran dos grupos fundamentales, 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***.** Los lenguajes de programación del lado servidor son reconocidos, ejecutados e interpretados por el propio servidor, el que se encarga de brindar

[2] Es una licencia de software creada por la *Free Software Foundation* que pretende garantizar la libertad de compartir y modificar el software cubierto por ella, asegurando que el software es libre para todos sus usuarios.

información al cliente en un formato comprensible para él. Para el desarrollo de la solución propuesta se hace necesario utilizar los lenguajes que a continuación se describen, teniendo en cuenta que responden a las tecnologías y lenguajes en que se desarrolla el Sistema de Registro de Entidades:

1.8.1. Java SE11

Java (desarrollado por *Sun Microsystems* y posteriormente adquirida por la compañía *Oracle*) es un lenguaje de programación de propósito general, concurrente, orientado a objetos y multiplataforma que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. El mismo permite desarrollar e implementar aplicaciones *Java* en equipos de escritorio y servidores. Proporciona una colección de clases para su uso en aplicaciones de red, que permite establecer y aceptar conexiones con servidores o clientes remotos, facilitando así la creación de aplicaciones distribuidas (ORACLE, 2019).

1.8.2. JavaScript ES6

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 son empleados en los sitios web, para realizar acciones en el lado del cliente, estando centrado en el código fuente de la página web (Mozilla, 2019).

1.8.3. TypeScript 2.9

Es un lenguaje de programación de código abierto desarrollado por Microsoft, el cual cuenta con herramientas de programación orientada a objetos, muy favorable si se tienen proyectos grandes. *TypeScript* convierte su código en *javascript* común. Es llamado también *superset*[3] de *javascript*, lo que significa que si el navegador está basado en *javascript*, este nunca llegará a comprender que el código original fue realizado con *typescript* y ejecutará el *javascript* como lenguaje original (Typescript, 2019).

1.8.4. HTML 5.0

HTML (*HyperText Markup Language* por sus siglas en inglés) es el lenguaje que se utiliza para crear las páginas web de internet. Es reconocido universalmente y permite publicar información de forma global. Define una estructura básica y un código *HTML* para la definición de contenido de una página

[3] Se trata de un lenguaje escrito sobre otro lenguaje.

web, como texto, imágenes, entre otros y se basa en la referenciación por hipertextos o enlaces entre páginas (Mozilla, 2019).

1.8.5. CSS 3.0

Hojas de Estilo en Cascada (por sus siglas en inglés *Cascading Style Sheets)* es un lenguaje para controlar la presentación de los documentos electrónicos definidos con *HTML* y *XHTML* (Lenguaje de Marcado para Hipertextos Extensible por sus siglas en inglés *eXtensible HyperText Markup Language)*. *CSS* es la mejor forma de separar los contenidos de su presentación y es imprescindible para la creación de páginas web complejas. Mejora la accesibilidad del documento, reduce la complejidad de su mantenimiento y permite visualizar el mismo documento en infinidad de dispositivos diferentes.
Se utiliza para definir el aspecto de todos los contenidos, como: el color, tamaño y tipo de letra de los párrafos de texto, la separación entre titulares y párrafos, la tabulación con la que se muestran los elementos de una lista. Esta forma de descripción de estilos ofrece a los desarrolladores el control total sobre estilo y formato de sus documentos. Funciona a base de reglas para las declaraciones sobre el estilo de uno o más elementos (Mozilla, 2019).

1.9. Entornos de Desarrollo Integrado (IDE)

Un *IDE*, es 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 (Rouse, 2019).
Para el desarrollo de la solución propuesta, se utilizan como *IDE* las herramientas *IntelliJ IDEA* en su versión 2018.2 para la comunidad y *WebStorm* en su versión 2018.2, teniendo en cuenta que ambas son libres, de código abierto y se integran perfectamente al lenguaje de programación a utilizar. Además, el componente propuesto debe ser integrado sobre una solución que también está implementada sobre estos entornos de desarrollo.

1.9.1 IntelliJ IDEA 2018.2

Es un *IDE* desarrollado por *JetBrains* y está disponible en dos ediciones: una edición para la comunidad (código abierto) y una edición comercial (de pago). Es soportado por varios sistemas operativos: *Windows*, *Linux* o *Mac OS*. Su versión gratuita soporta lenguajes como *Java*, *Groovy*, *XML/XSL*, *Kotlin* y *Python*, *Clojure*, *Dart*, *Erlang*, *Go*, *Haxe*, *Perl*, *Scala*, *Haskell*, *Lua*, estos últimos vía *plugin*. La versión de pago además de soportar todos los lenguajes antes mencionados soporta *JavaScript*, *HTML*, *CSS*, *SQL*, *Ruby*, así como *PHP* vía *plugin* (JetBrains, 2019).

1.9.2. WebStorm 2018.2

Es un entorno de desarrollo inteligente, entiende el proyecto y ayuda a producir código de alta calidad de manera más 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. *WebStorm* agiliza el flujo de trabajo mediante la integración con todo lo necesario para el desarrollo productivo. Además, desde el *IDE* se pueden utilizar varias herramientas como el depurador y terminales (JetBrains, 2019).

1.10. Sistema Gestor de Bases de Datos

Un Sistema Gestor de Bases de Datos (SGBD) es un conjunto de programas no visibles que administran y gestionan la información que contiene una base de datos. A través de él se maneja todo acceso a la base de datos con el objetivo de servir de interfaz entre ésta, el usuario y las aplicaciones (Rouse, 2019). Para el desarrollo de la solución propuesta se hace necesario utilizar el PostgreSQL como sistema gestor de bases de datos teniendo en cuenta la compatibilidad de este con el software SIRES.

1.10.1 PostgreSQL 9.6

Es un potente sistema de base de datos objeto-relacional de código abierto. Cuenta con más de 15 años de desarrollo activo y una arquitectura probada con una alta fiabilidad e integridad de datos. 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* (The PostgreSQL Global Development Group, 2019).

1.11. 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 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).
Objetivos de estos patrones:

- Proporcionar catálogos de elementos reusables en el diseño de sistemas software.

- Evitar la reiteración en la búsqueda de soluciones a problemas ya conocidos y solucionados anteriormente.
- Formalizar un vocabulario común entre diseñadores.
- Estandarizar el modo en que se realiza el diseño.
- Facilitar el aprendizaje de las nuevas generaciones de diseñadores condensando conocimiento ya existente.

Así mismo, no pretenden:

- Imponer ciertas alternativas de diseño frente a otras.
- Eliminar la creatividad inherente al proceso de diseño.
- No es obligatorio utilizar los patrones siempre, sólo en el caso de tener el mismo problema o similar que soluciona el patrón, siempre teniendo en cuenta que en un caso particular puede no ser aplicable. Abusar o forzar el uso de los patrones puede ser un error (Rouse, 2019).

Se pueden agrupar en dos grandes grupos; los GRASP (*General Responsibility Assignment Software Patterns*, en inglés), que son patrones generales de software para asignación de responsabilidades y los GOF (*Gang of Four*, en inglés), 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.

1.12. Conclusiones parciales

El desarrollo del marco teórico referencial relacionado con los términos reportes estadísticos y pizarra del control, facilita una mejor comprensión del objeto de estudio de la presente investigación. Además, el estudio de soluciones nacionales y extranjeras destinadas al procesamiento de información estadística, contribuyó a la definición de algunas de las funcionalidades del componente propuesto, entre las que se encentra, la visualización de información a través de una pizarra de control. Por otra parte, 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 del componente propuesto, en correspondencia con las características de este. El estudio de los patrones de diseño posibilitó identificar cuáles emplear en el desarrollo del componente propuesto

CAPÍTULO 2: DESCRIPCIÓN DE LA SOLUCIÓN PROPUESTA

2.1. Introducción

En este capítulo se realiza una descripción de las principales características del componente para la obtención de reportes en el SIRES. En el mismo se describe el diseño de la solución propuesta, a partir de las fases de la metodología definida para guiar el desarrollo de esta. Entre los contenidos analizados se encuentran el proceso de captura y validación de los requisitos funcionales (RF) y no funcionales (RnF) con los que debe cumplir el componente propuesto. Además, se describe la arquitectura de la solución, el diagrama de clases del diseño, el modelo de datos y los patrones de diseño utilizados. Por otra parte, se exponen los estándares de codificación empleados en la disciplina de implementación y se realiza una descripción de las principales interfaces de la solución obtenida.

2.2. Descripción del negocio

La solución propuesta se centra en la creación de un componente capaz de reducir el tiempo y la posibilidad de errores en el procesamiento de las estadísticas que genera la información registrada en el SIRES. El componente debe dar la posibilidad de generar reportes en formato *excel* o *pdf* utilizando la herramienta *JasperReport*. La solución debe permitir al usuario seleccionar una fecha inicial y una final, así como el tipo de reporte que desea obtener y luego generar el mismo en el formato seleccionado. Para el caso de los reportes de tipo notificaciones además de los rangos de fecha, se debe especificar el código de la Entidad. Por otra parte, se garantiza la visualización permanente de las estadísticas del proceso de registro de Entidades en el SIRES a través de una pizarra de control, que también permite el acceso rápido a los detalles de la información mostrada.

2.3. Disciplina de Requisitos

El esfuerzo principal en la disciplina Requisitos es desarrollar un modelo del sistema que se va a construir y comprende la administración de los requisitos funcionales y no funcionales del producto (Sánchez, 2015). Teniendo en cuenta que el componente que responde al cumplimiento del objetivo general de la presente investigación, tiene un negocio bien definido y fácil de comprender, sin necesidad de realizar un modelado de este, se decide, según la metodología seleccionada, utilizar en la disciplina de requisitos el escenario número cuatro (HU). En este epígrafe se presentan las técnicas para la identificación de los requisitos, la especificación de los mismos, así como las historias de usuario obtenidas.

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 del componente propuesto 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, 2011). Esta técnica fue aplicada a los trabajadores de la ONEI. En el Anexo 1, se encuentra la guía de preguntas utilizada en el desarrollo de la entrevista.
- **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. En el anexo 2 del presente documento se relacionan los acuerdos tomados durante la tormenta de ideas.

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 ofrecidos por el sistema (Sommerville, 2011).

En la presente investigación, una vez realizado el levantamiento de información e identificadas las necesidades del cliente, se identificaron un total de 13 requisitos funcionales (RF). A continuación, en la tabla 1 se presentan cada uno de estos y sus respectivas descripciones.

Tabla 1: requisitos funcionales

Nº	Nombre	Descripción
RF1.	Generar reporte	Permite generar el reporte seleccionado con los datos especificados.
RF2.	Generar reporte de las notificaciones	Permite generar un reporte de las notificaciones, introduciéndose el código de la Entidad a la cual corresponde el reporte, así como seleccionar una fecha inicial y una fecha final, y el tipo de formato en el cual se desea generar el reporte.
RF3.	Generar Maestro	Permite generar un reporte de tipo maestro, una vez seleccionada una fecha inicial y una fecha final, y el tipo de formato en el cual se desea generar el reporte.
RF4.	Generar reporte Organización Institucional, principales Entidades	Permite generar un reporte de tipo Organización Institucional, principales Entidades, una vez seleccionada una fecha inicial y una fecha final, y el tipo de formato en el cual se desea generar el reporte.
RF5.	Generar reporte servicio estadístico	Permite generar un reporte de tipo servicio estadístico, una vez seleccionada una fecha inicial y una fecha final, y el tipo de formato en el cual se desea generar el reporte.
RF6.	Generar reportes de movimientos institucionales	Permite generar un reporte de tipo movimientos institucionales, una vez seleccionada una fecha inicial y una fecha final, y el tipo de formato en el cual se desea generar el reporte.
RF7.	Generar actualización del Registro Estatal de Entidades Agropecuarias No Estatales (REEANE) y del Registro Estatal de Unidades Básicas de Producción Cooperativa (REUCO)	Permite generar un reporte de tipo actualización del Registro Estatal de Entidades Agropecuarias No Estatales (REEANE) y del Registro Estatal de Unidades Básicas de Producción Cooperativa (REUCO), una vez seleccionada una fecha inicial y una fecha final, y el tipo de formato en el cual se desea generar el reporte.

RF8.	Generar actualización del Registro Estatal de Empresas y Unidades Presupuestadas (REEUP)	Permite generar un reporte de tipo actualización del Registro Estatal de Empresas y Unidades Presupuestadas (REEUP), una vez seleccionada una fecha inicial y una fecha final, y el tipo de formato en el cual se desea generar el reporte.
RF9.	Visualizar datos al inicio del sistema	Permite mostrar en la página de inicio del sistema un *dashboard* dividido en cuatro cuadrantes, se visualiza en cada uno de estos las cantidades de Entidades registradas, Entidades en término, Entidades al límite e irregularidades. Además se puede acceder desde el *dashboard* a las listas de las clasificaciones antes mencionadas según las cantidades.
RF10.	Listar Entidades registradas	Permite listar las Entidades registradas con su respectiva información dividida en código, descripción y tipo de Entidad. Además, se pueden organizar según cada tipo de información y hacer filtros de búsqueda.
RF11.	Listar Entidades en término	Permite listar las Entidades registradas que se encuentran en término según la fecha definida para la entrega de la documentación requerida para concluir el registro, cada una con su respectiva información dividida en código, descripción y tipo de Entidad. Además, se pueden organizar según cada tipo de información y hacer filtros de búsqueda.
RF12.	Listar Entidades al límite	Permite listar las Entidades registradas que se encuentran al límite según la fecha definida para la entrega de la documentación requerida para concluir el registro, cada una con su respectiva información dividida en código, descripción y tipo de Entidad. Además, se pueden organizar según cada tipo de información y hacer filtros de búsqueda.
RF13.	Listar irregularidades	Permite listar las irregularidades surgidas en el proceso de registro de Entidades, cada una con su respectiva información dividida en código, descripción y tipo de Entidad. Además, se pueden organizar según cada tipo de información y hacer filtros de búsqueda.

Fuente: elaboración propia

En el caso de los requisitos no funcionales (RnF) se definieron un total de 9, los cuales son clasificados y descritos a continuación.

Usabilidad

RnF1. La interfaz del componente a desarrollar debe ser sencilla.
RnF2. El componente debe mostrar facilidad para interactuar y entender las actividades que realiza el usuario.
RnF3. La cantidad de clic para acceder a una funcionalidad debe ser como máximo 5.

Portabilidad

RnF4. El componente debe funcionar sobre las plataformas Windows y Linux.

Confiabilidad

RnF5. El componente debe contar con campos obligatorios para garantizar un manejo adecuado de la información introducida por el usuario.
RnF6. El componente no debe permitir la entrada de datos incorrectos y debe notificar los errores.

Mantenibilidad

RnF7. Todo el código del componente debe estar comentado haciéndolo fácil de analizar.

Seguridad

RnF8. Los usuarios para acceder al componente primero deben haberse autenticados en el sistema.

Eficiencia

RNF 9. El componente en condiciones de carga máxima, cuando se ejecuten tareas que realizan consultas a las bases de datos y mecanismos de comunicación entre componentes debe ser capaz de ofrecer tiempos de repuestas mínimos o iguales a 5 segundos.

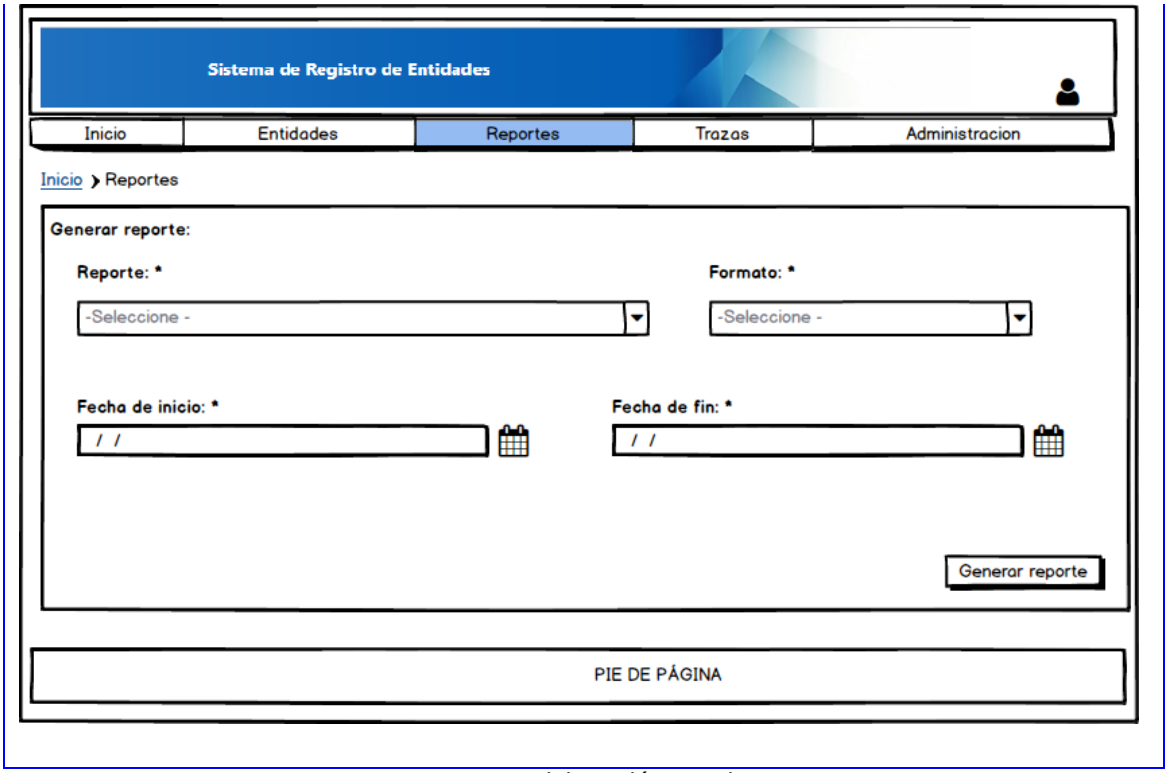
2.3.3. Historias de usuario

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 del componente propuesto se definieron un total de 13 HU, una por cada RF. A continuación, se describe la HU correspondiente al RF1 generar reporte, teniendo en cuenta que este

es uno de los RF bases para el desarrollo del componente y de mayor prioridad para el cliente. El resto de las HU se pueden consultar en el Anexo 3.

Tabla 2: HU generar reporte

Historia de usuario	
Número: 1	**Requisito:** generar reporte
Programador: Carlos Felix Pentón Martínez	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** 40h
Riesgo en Desarrollo:	**Tiempo Real:** 40h
Descripción: se exporta el reporte seleccionado con los datos especificados en los siguientes campos: - Reporte (campo de selección obligatorio para especificar el tipo de reporte a generar) - Fecha de inicio (campo de selección obligatorio para especificar la fecha desde donde se desea recopilar la información para el reporte) - Fecha de fin (campo de selección obligatorio para especificar la fecha hasta donde se desea recoger la información para el reporte) - Formato (campo de selección obligatorio para especificar el tipo de formato en que desea sea el reporte a generar) **Opción**: - Generar reporte (exporta el reporte en formato *excel* o en la plantilla determinada para el tipo de reporte)	
Observaciones:	
Prototipo de interfaz gráfica de usuario:	



Fuente: elaboración propia

2.4. Disciplina de análisis y diseño

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

2.4.1. 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, 2011).

La solución propuesta tiene como base la arquitectura del SIRES. En la figura 1 se representan los elementos de esta arquitectura y las partes donde incide el componente propuesto. Es importante especificar que el SIRES utiliza una arquitectura separada en *frontend* y *backend*. El *frontend* es la

parte del software que interactúa con los usuarios y el *backend* es la parte que procesa la entrada desde el *frontend*. La idea general es que el *frontend* sea el responsable de recolectar los datos de entrada del usuario, que pueden ser de muchas y variadas formas, y los transforma ajustándolos a las especificaciones que demanda el *backend* para poder procesarlos, devolviendo generalmente una respuesta que el *frontend* recibe y expone al usuario de una forma entendible para este. Para la solución propuesta la conexión entre ambas partes se realiza a través del protocolo *HTTP* (Protocolo de transferencia de hipertexto, por sus siglas en inglés *Hypertext Transfer Protocol*) intercambiando datos en formato *JSON* (Notación de objetos *JavaScript*, por sus siglas en inglés *JavaScript Object Notation*).

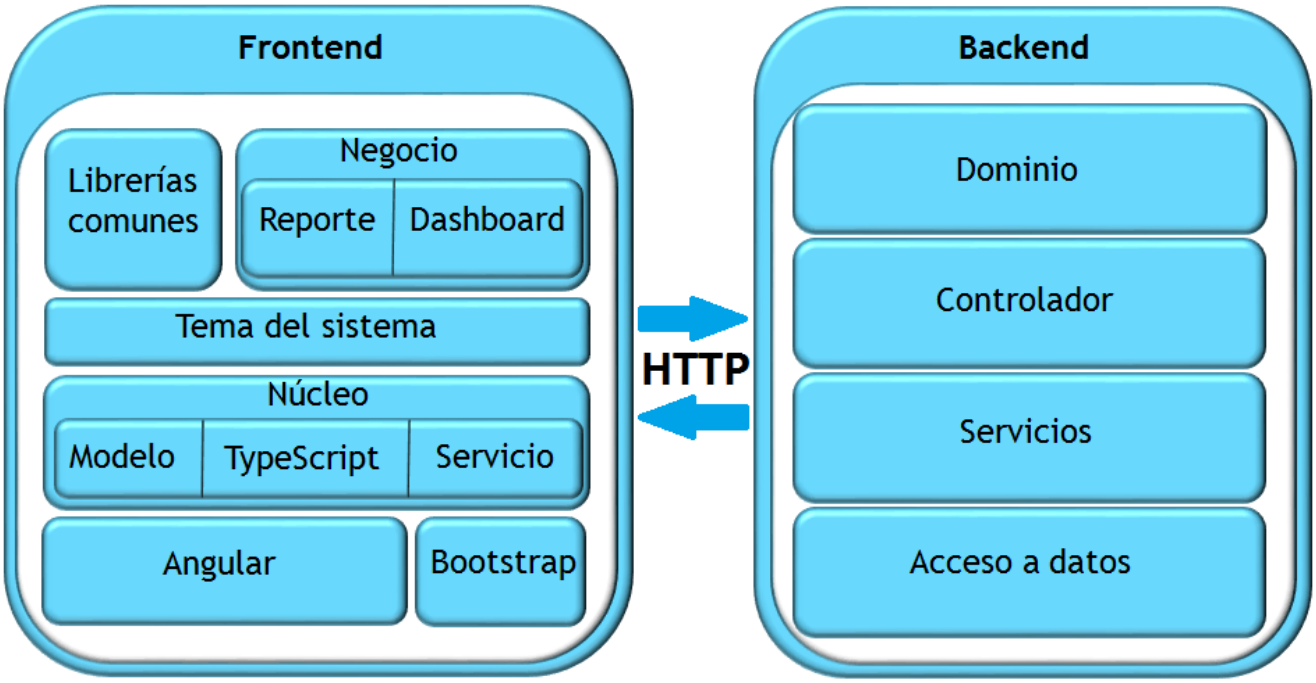


Figura 1: arquitectura SIRES
Fuente: elaboración propia

A partir de la figura anterior, en la parte del *frontend* el componente propuesto incide en los elementos del negocio. Este componente es implementado sobre la arquitectura *frontend* del SIRES a través del uso del patrón arquitectónico Modelo Vista Controlador (MVC).

El patrón MVC separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos. Se trata de un modelo maduro y que ha demostrado su validez a lo largo de los años en todo tipo de aplicaciones, y sobre multitud de lenguajes y plataformas de desarrollo. El modelo contiene una representación de los datos que maneja el sistema, su lógica de negocio, y sus mecanismos de persistencia. La vista, o interfaz de usuario, compone la información que se envía al cliente y los mecanismos de interacción con éste. El controlador, actúa como intermediario entre el

Modelo y la Vista, gestionando el flujo de información entre ellos y las transformaciones para adaptar los datos a las necesidades de cada uno (UA, 2019).

En la figura 2 se muestra una descripción más detallada de cómo se representa en el *frontend* del SIRES la arquitectura del componente propuesto, para el caso de la HU generar reporte, a partir del uso del patrón arquitectónico MVC. En la figura el modelo está compuesto por el archivo *reporte.ts*, el cual contiene la información que se le solicita al usuario para confeccionar el reporte, ejemplo: fecha inicio y fecha fin. Por otra parte, la vista se compone por el archivo *gestionar-reporte.component.html*, este representa la información visible al usuario e interactúa con funciones específicas del controlador. El controlador está compuesto por el archivo *gestionar-reporte.component.ts*, encargado de manipular el modelo y mostrar la información a la vista.

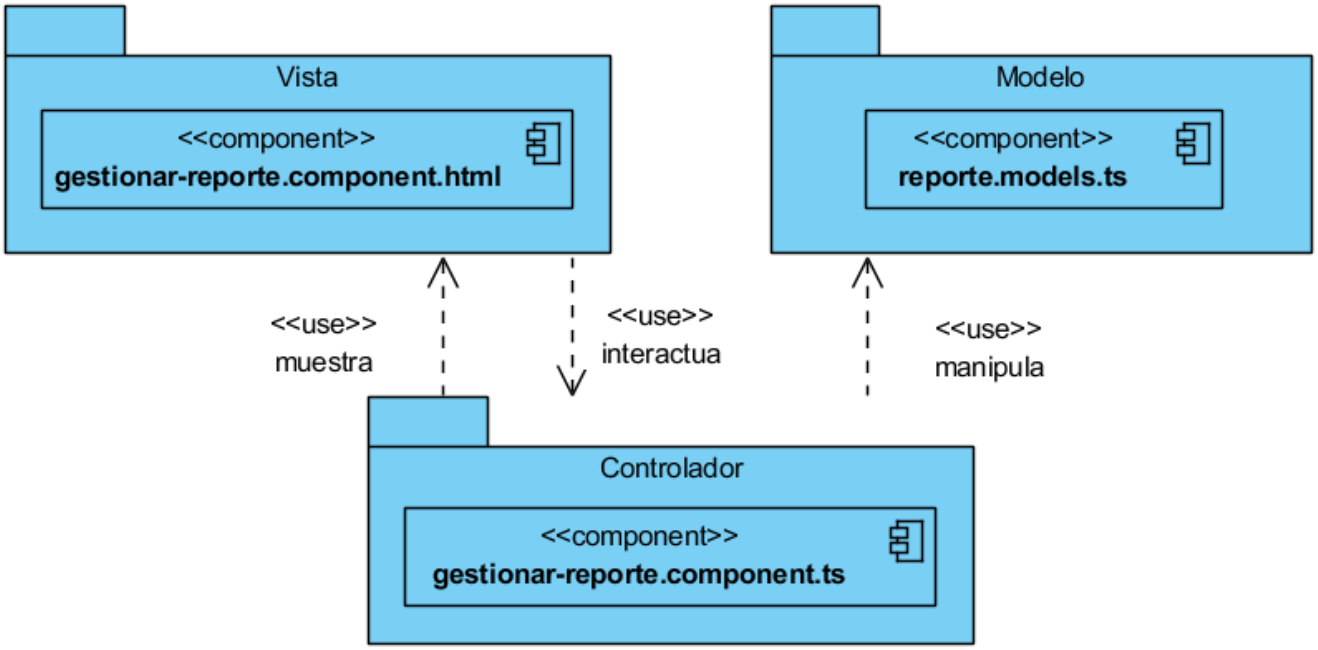


Figura 2: arquitectura *frontend* del componente propuesto
Fuente: elaboración propia

A partir del diagrama de la figura 1, en la parte del *backend* el componente propuesto incide en las capas controladora, servicios y acceso a datos. Este componente es implementado sobre la arquitectura *backend* del SIRES a través del uso del patrón arquitectónico N Capas.

El patrón arquitectónico N Capas es una extensión del patrón Capas tradicional[4]. 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 (Safari, 2019).

En la figura 3 se muestra una descripción más detallada de cómo se representa en el *backend* del SIRES la arquitectura del componente propuesto de manera genérica, a partir del uso del patrón arquitectónico N Capas. En la figura el componente *ReportesController.java* de la capa Controlador usa al componente *ReportesService.java* de la capa Servicios, el cual usa al componente de la capa de Acceso a Datos *ReportesRepository.java* y los tres a la vez están orientados a la capa Dominio, la cual es el núcleo de la aplicación. La misma y su lógica de negocio definen el comportamiento y las restricciones de la aplicación, en cuanto a la forma en que se van a comunicar las demás capas con esta. Lo anterior hace que una aplicación sea diferente de otras en cuanto a la función que realiza.

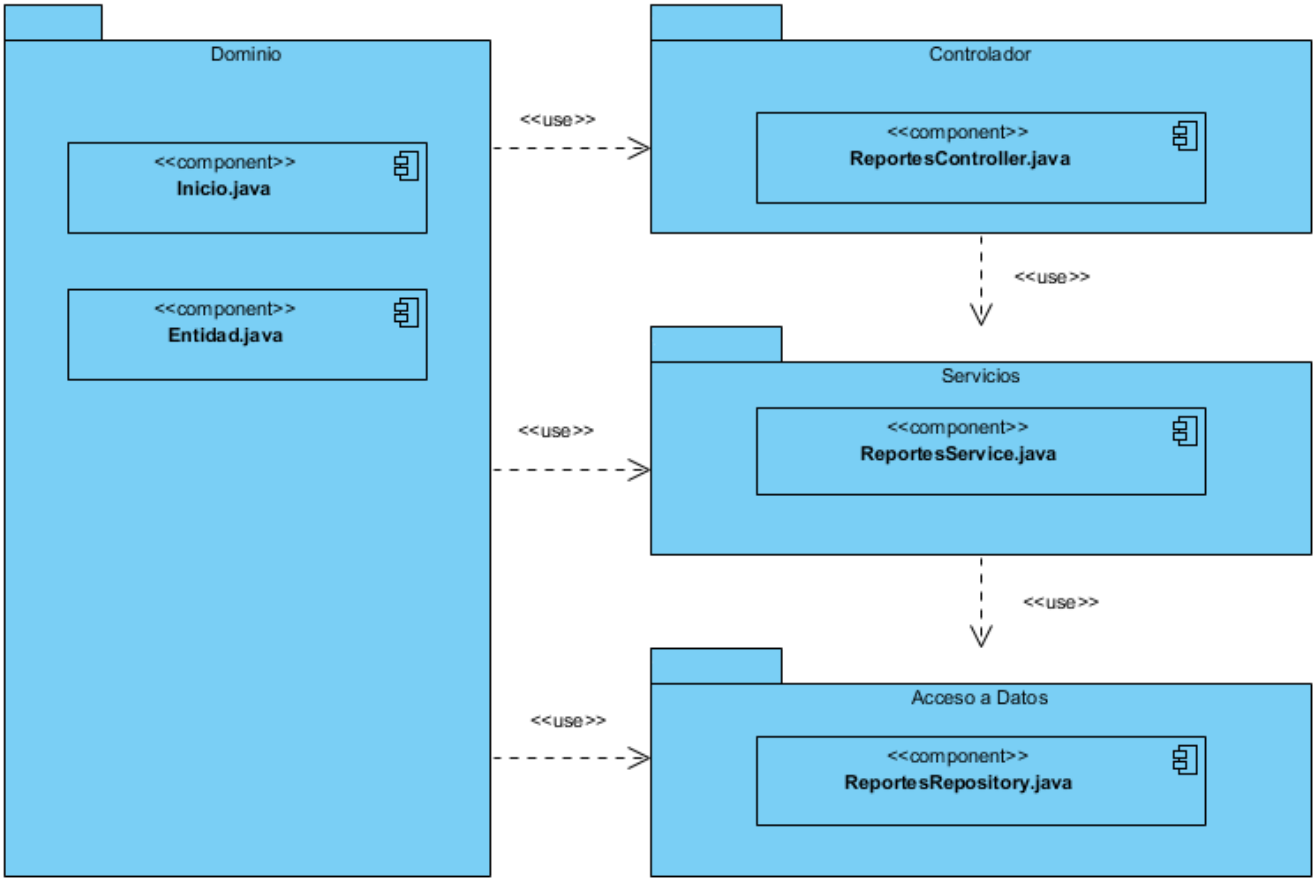


Figura 3: arquitectura *backend* del componente propuesto
Fuente: elaboración propia

[4] El patrón Capas tradicional 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

2.4.2. Patrones de diseño

Para diseñar el componente 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[5]**

Los Patrones Generales de Software para la Asignación de Responsabilidades (GRASP por sus siglas en inglés) 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.

Experto: el patrón experto en información se utiliza con frecuencia en la asignación de responsabilidades; es un principio de guía básico que se utiliza continuamente en el diseño de objetos. Este permite conservar el encapsulamiento, ya que los objetos se valen de su propia información para llevar a cabo las tareas (Larman, 2016).
En la presente investigación el uso de este patrón se evidencia en las clases entidades. Estas clases son las expertas en información y las encargadas de manejar la lógica del negocio. En la siguiente figura se muestra un fragmento del código de la clase *Entidad.java*, la cual es la experta en información para el tratamiento de las Entidades.

[5] General Responsibility Assignment Software Patterns (Patrones de software de asignación de responsabilidad general)

```java
public class  Entidad implements Serializable {

    private static final long serialVersionUID = 1L;
    @Id
    @NotNull
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "dentidad_seq")
    @SequenceGenerator(name = "dentidad_seq", sequenceName = "dentidad_id_seq", initialValue = 1, allocationSize = 1)
    @Basic(optional = false)
    @Column(name = "id")
    protected Long id;
    @Column(name = "codigo")
    private String codigo;
    @Column(name = "descripcion")
    private String descripcion;
    @Column(name = "activo")
    private Boolean activo;
    @Column(name = "siglas")
    private String siglas;
    @Column(name = "corta")
    private String corta;
    @Column(name = "nit")
    private String nit;
    @Column(name = "fechacreacion")
    @Convert(converter = FechaConverter.class)
    private LocalDate fechacreacion;
    @Column(name = "fechafin")
    @Convert(converter = FechaConverter.class)
```

Figura 4: aplicación del patrón Experto en la clase Entidad.
Fuente: elaboración propia

Creador: este patrón guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas desarrollados a partir de un paradigma orientados a objetos. El propósito fundamental de este patrón es encontrar un creador que se debe conectar con el objeto producido en cualquier evento. Al escogerlo como creador, se da soporte al bajo acoplamiento lo cual supone menos dependencias respecto al mantenimiento y mejores oportunidades de reutilización (Larman, 2016).

En la presente investigación el uso de este patrón se evidencia en la clase *EntidadServiceImpl.java.* El patrón se aplica en el momento de crear nuevas instancias de la clase inicio, tal y como ilustra en el área señalada en la figura 5.

```
@Override
public Inicio datosInicio() {
    Inicio inicio = new Inicio();
    inicio.setTotal_irregularidades(Long.parseLong(String.valueOf(irregularidadService.listarIrregularidades().size())));
    List<Entidad> entidadList = entidadRepository.findAllByActivoTrueAndPadreIsNull();
    List<Entidad> registradas = entidadList.stream().filter(entidad -> entidad.getEstadoentidad()
            .getId() != nomencladorService.nomencladorDadodescripcionAndTipo(TipoNomenclador.NESTADO_ENTIDAD.
            PendienteRegistrar(), TipoNomenclador.NESTADO_ENTIDAD).getId()).collect(Collectors.toList());
    inicio.setTotal_registradas(Long.parseLong(String.valueOf(registradas.size())));
    inicio.setTotal_termino(Long.parseLong(String.valueOf(entidadRepository
            .findAllByActivoTrueAndFechaterminoIsNotNullAndFechaterminoIsGreaterThan(LocalDate.now()).size())));
    inicio.setTotal_limites(Long.parseLong(String.valueOf(entidadRepository
            .findAllByActivoTrueAndFechaterminoIsNotNullAndFechaterminoIsLessThanEqual(LocalDate.now()).size())));
    return inicio;
}
```

Figura 5: aplicación del patrón Creador en la clase *EntidadServiceImpl.java*
Fuente: elaboración propia

Controlador: el objetivo de este patrón es asignar la responsabilidad de controlar el flujo de eventos del sistema, a clases específicas. Esta patrón delega la responsabilidad en otras clases con las que mantiene un modelo de alta cohesión (Larman, 2016).

En la presente investigación el uso del patrón se evidencia en las clases controladoras presentes en la arquitectura del *backend*. En la figura 6 se muestran las clases *EntidadController.java*, *IrregularidadController.java* y *ReportesController.java* en las cuales se aplica el patrón controlador.

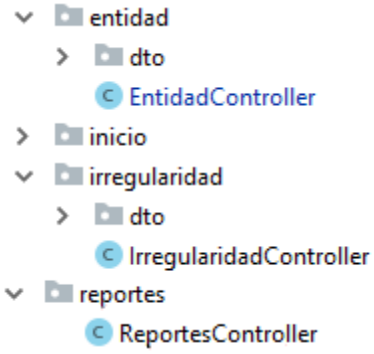


Figura 6: aplicación del patrón Controlador en clases del *backend*.
Fuente: elaboración propia

Bajo acoplamiento: este patrón permite tener las clases lo menos relacionadas entre sí, para que, en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión en el resto de las clases. Esta característica permite potenciar la reutilización y disminuye la dependencia entre las clases. Su principal característica es mantener las clases más independientes entre sí y con la menor cantidad de relaciones; la cual posibilita que, en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión posible en el resto de las clases, potenciando la reutilización y disminuyendo la dependencia entre las clases (Larman, 2016).

En la presente investigación el uso de este patrón se evidencia en el 63 % de las clases del diseño, las cuales tienen una baja dependencia una de otras, tal y como se demuestra en el Capítulo 3 con los resultados de la validación del diseño a partir de la aplicación de métricas.

Alta cohesión: en la perspectiva del diseño orientado a objetos, la cohesión es una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase. Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme (Larman, 2016). La principal función de la aplicación de este patrón es asignar responsabilidades de modo que la cohesión siga siendo alta. La información que almacena una clase debe de ser coherente y debe estar en la medida de lo posible relacionada con la clase.

En la presente investigación el uso de este patrón se evidencia en el 84 % de las clases del diseño, las cuales tienen una baja sobrecarga de responsabilidades, tal y como se demuestra en el Capítulo 3 con los resultados de la validación del diseño a partir de la aplicación de métricas.

- **Patrones GoF[6]**

Patrones Banda de los Cuatro, por sus siglas en inglés de *Gang of Four,* describen soluciones simples y eficaces a problemas específicos en el diseño de software orientado a objetos (Larman, 2016). En el caso de la presente investigación de aplicaron los siguientes patrones GoF.

Patrón Decorador: permite añadir responsabilidades a objetos concretos de manera dinámica y transparente sin afectar a otros objetos. Este patrón brinda más flexibilidad que la herencia estática y evita que las clases más altas en la jerarquía estén demasiado cargadas de funcionalidad y sean complejas (Larman, 2016).

En la presente investigación el uso de este patrón se evidencia en la clase *gestionar-reportes.component.ts.* El patrón se aplica en el momento de hacer uso del decorador *@Component*, el cual es típico de las clases *Component* en Angular. En el área señalada en la figura 7 se especifica el uso de este patrón.

[6] Gang Of Four (Pandilla de cuatro)

```
@Component({
    selector: 'app-gestionar-reporte',
    templateUrl: './gestionar-reporte.component.html',
    styleUrls: ['./gestionar-reporte.component.scss']
})
```

Figura 7: aplicación del patrón decorador en la clase *gestionar-reportes.component.ts*
Fuente: elaboración propia

Patrón Observador: permite observar los cambios producidos por un objeto, de esta forma, cada cambio que afecte el estado del objeto observado lanza una notificación a los observadores; a esto se le conoce como Publicador-Suscriptor. Es uno de los principales patrones de diseño utilizados en interfaces gráficas de usuario (GUI), ya que permite desacoplar al componente gráfico de la acción a realizar (Google, 2019).

En la presente investigación el uso de este patrón se evidencia en varias partes de la implementación de los componentes en el *frontend.* Por ejemplo, al enviar los datos de la vista al controlador, este último hace la petición de un servicio a través de una inyección y se queda a la espera de la respuesta que este servicio debe proveer para así emitir o no un evento. En la figura 8 se especifica a través del segundo recuadro un ejemplo de la aplicación de este patrón en la clase *dashboardService.ts.*

- **Otros patrones**

Inyección de dependencias (ID): este es un patrón de diseño orientado a objetos, que permite el paso de objetos como dependencias, los cuales a su vez pueden pasar a componentes y estar disponibles en toda la aplicación. El propósito de este patrón es contener la lógica del negocio, clases para acceso a datos o utilidades de infraestructura.

Angular tiene su propio marco ID, que se utiliza normalmente en el diseño de aplicaciones angulares para aumentar su eficiencia y modularidad. Las dependencias son servicios u objetos que una clase necesita para realizar su función. ID es un patrón de codificación en el que una clase solicita dependencias de fuentes externas en lugar de crearlas. En Angular, el marco ID proporciona dependencias declaradas a una clase cuando se crea una instancia de esa clase (Google, 2019).

Teniendo en cuenta que la solución propuesta está desarrollada utilizando este marco de trabajo, el patrón ID se aplica en correspondencia con las adecuaciones para Angular. En la figura 8 se señala en el primer recuadro la aplicación del patrón al crear la instancia del *ApiService.*

```
import {Injectable} from '@angular/core';
import {Observable} from 'rxjs/Rx';
import 'rxjs/add/operator/map';
import 'rxjs/add/operator/catch';
import {ApiService} from '../../../../shared/services';

@Injectable()
export class DashboardService {
  private urlResource = '/entidad/home';

  constructor(private apiService: ApiService) {}

  getResource(): Observable<Response> {
    return this.apiService.use( url: 'API_RESOURCE_URL').get(this.urlResource);
  }
}
```

Figura 8: aplicación de los patrones Observador e ID en la clase *dashboardService.ts*.
Fuente: elaboración propia

Repositorio (*Repository*): el patrón repositorio o *repository pattern* está estrechamente relacionado con el acceso a datos y permite tener una abstracción de la implementación del acceso a datos en cualquier aplicación, de modo que la lógica de negocio no conozca ni esté acoplada a la fuente de datos. En pocas palabras, el repositorio actúa como un intermediario entre la lógica del negocio y la lógica de acceso a datos para que se centralice en un solo punto, y de esta forma se logre evitar redundancia de código. Al ser este una abstracción del acceso a datos permite desacoplar y testear de una forma más sencilla el código, ya que al estar desacoplado se pueden generar pruebas unitarias con mayor facilidad (Janssen, 2019). Un ejemplo de la práctica de este patrón en la propuesta de solución se muestra en la siguiente figura cuando se obtienen las Entidades del repositorio.

```
@Override
public List<Entidad> obtenerEntidades(List<Long> entidades) {
    return entidadRepository.obtenerEntidadesIn(entidades);
}
```

Figura 9: aplicación del patrón *Repository*. Clase *EntidadServiceImpl.java*
Fuente: elaboración propia

Como se puede observar en la figura 9, la lógica de negocio del cliente no interactúa directamente con la fuente de datos, ya que con este patrón la lógica de negocio no tiene conocimiento de dónde provienen ni como se obtuvieron los datos. Además, se puede apreciar que se comunica directamente con el repositorio, el cual se encarga de hacer los mapeos necesarios y de comunicarse con la fuente de datos.

2.4.3. Diagrama de clases

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 de los sistemas, donde se crea una vista lógica de la información que se maneja en el sistema, los componentes que se encargarán del funcionamiento y la relación entre uno y otro (Larman, 2016).

A continuación, se muestra una parte del diagrama de clases del diseño, en el cual se encuentran las clases correspondientes a la HU generar reporte, teniendo en cuenta que la misma responde al requisito anteriormente seleccionado en todo el documento. El diagrama de clases del diseño con la totalidad de las clases se encuentra entre los artefactos entregables de la tesis.

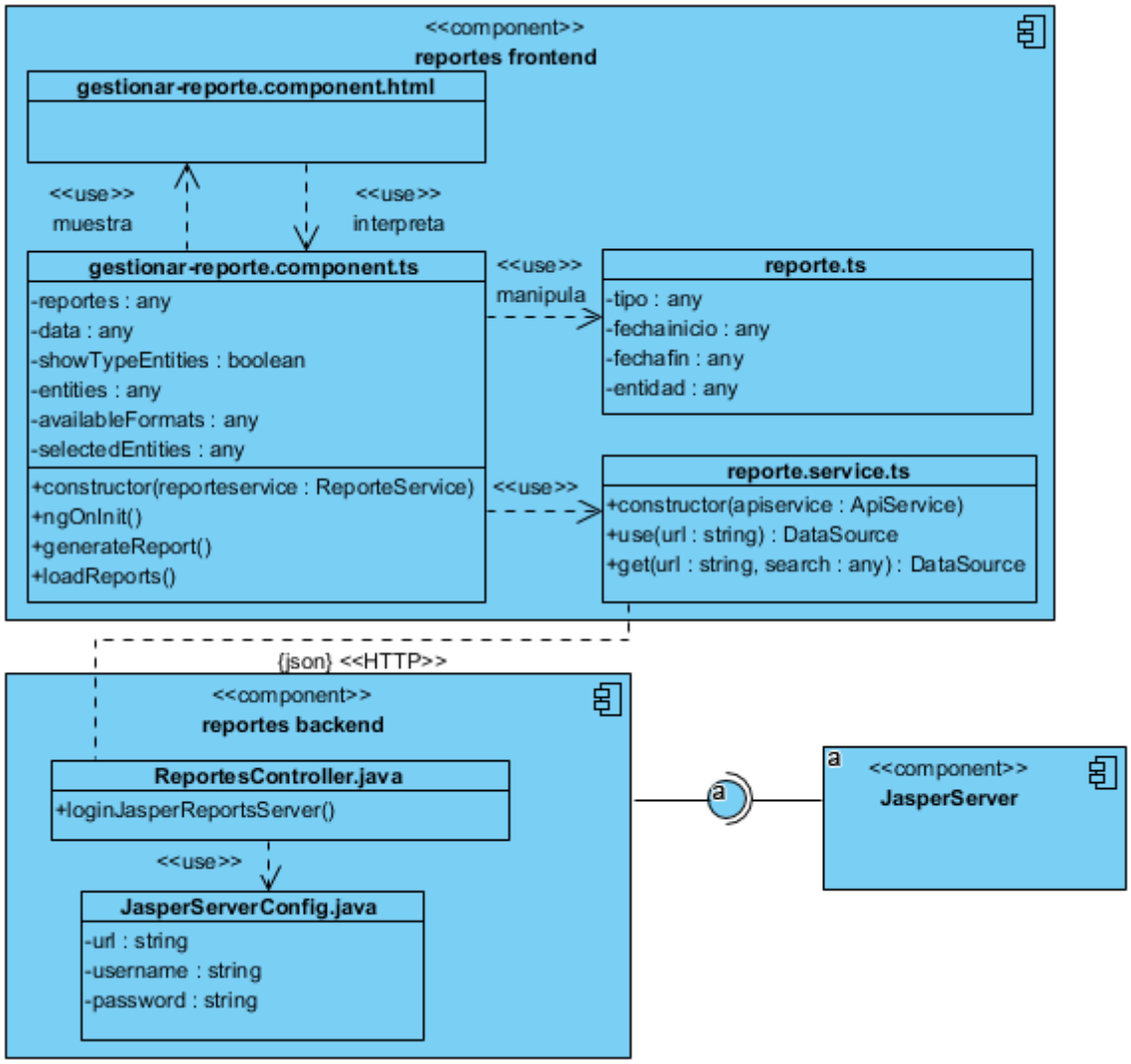


Figura 10: diagrama de clases de diseño para la HU generar reporte.

Fuente: elaboración propia

En el diagrama de clases del diseño se evidencia cómo la vista interactúa con la controladora, esta a su vez manipula al modelo y se nutre de un servicio ejecutado mediante una petición realizada al *backend* a través del protocolo HTTP. Esta petición llega al controlador *ReporteController.java*, que a su vez consume servicios del servidor de *JasperReports,* el cual es el encargado de manipular las clases entidades mapeadas desde la base de datos.

2.4.4. Modelo de base de datos

En la solución propuesta se utiliza el mismo modelo de datos del SIRES. Este modelo se muestra en la figura 11 a través del diagrama Entidad-Relación, mostrando las entidades, atributos y relaciones entre estas.

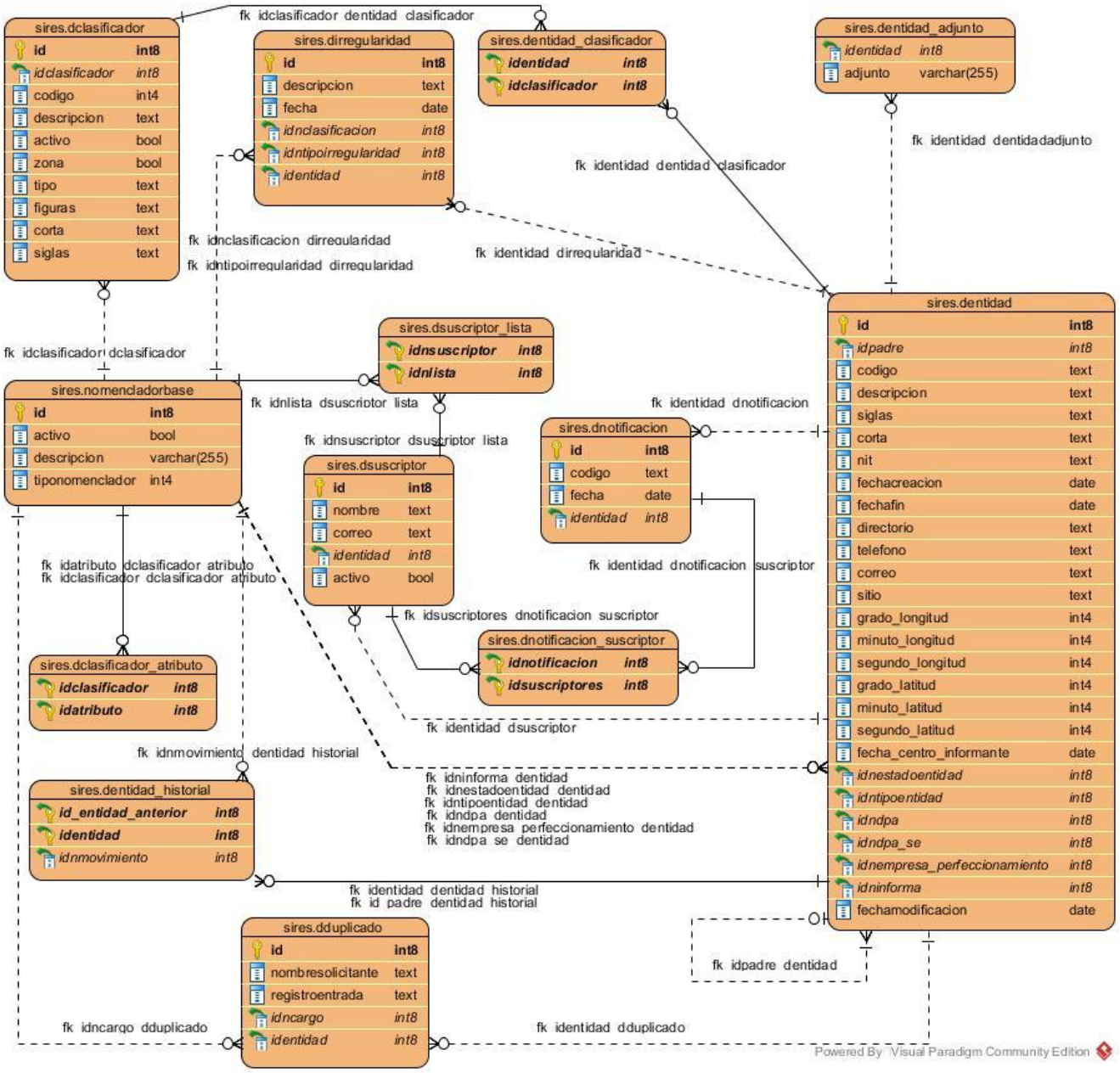


Figura 11: modelo de datos.
Fuente: elaboración propia

El modelo de datos representado en la figura 11 cuenta con un total de 13 tablas, de ellas una es un nomenclador base, dentro del cual se gestionan tanto los elementos del negocio como los de la arquitectura. Esta tabla también es la encargada de agrupar un conjunto de 10 nomencladores con atributos en común, por ejemplo, id, activo y descripción, los cuales se identifican por el atributo tipo. Las otras 12 tablas del modelo son las encargadas de gestionar conceptos específicos del negocio del sistema en general, por ejemplo en la tabla *sires.dentidad* es donde se guarda el nombre y la descripción de las Entidades que se registran en el sistema.

2.5. Disciplina de 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 SIRES.

2.5.1. 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 se escriben con mayúscula, en caso de ser un nombre compuesto las siguientes palabras se escribirán de igual forma, en la figura 12 se evidencia el uso de este estándar en la clase *GestionarReportesComponent.*

```
class GestionarReportesComponent implements OnInit {
```

Figura 12: representación del estándar de codificación para el nombre de las clases.
Fuente: elaboración propia

- Los nombres de los métodos se escriben con minúscula, en caso de ser un nombre compuesto las siguientes palabras inician con mayúscula, ver figura 13.

```
public ResponseEntity<Map<String, String>> loginJasperReportsServer() {
```

Figura 13: representación del estándar de codificación para el nombre de los métodos.
Fuente: elaboración propia

- Los identificadores para las variables y los parámetros se escriben con letras en minúsculas y en caso de ser un nombre compuesto las siguientes palabras se escriben de igual forma, en la figura 14 se evidencia el uso de este estándar de codificación.

[7] Error de software que desencadena un resultado indeseado

```
@Value("http://10.58.13.33:8080/jasperserver/rest/login")
String url;
@Value("jasperadmin")
String username;
@Value("jasperadmin")
String password;
```

Figura 14: representación del estándar de codificación para las variables y parámetros.
Fuente: elaboración propia

- **Sentencias *package* e *import***

La primera línea no-comentario de los ficheros fuente *Java* es la sentencia *package*. Después de esta, pueden seguir varias sentencias *import*. Por ejemplo:

```
package cu.uci.cegel.sires.web.reportes;

import cu.uci.cegel.sires.infrastructure.reportes.JasperServerConfig;
import org.springframework.http.HttpHeaders;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
```

Figura 15: representación del estándar de para sentencias *package* e *import*.
Fuente: elaboración propia

- Todas las funciones deben tener comentarios, que describan su propósito. En la figura 16, se evidencia el uso de este estándar en el método *loginJasperReportsServer*.

```
/**
 * Login on JasperServer
 */
@GetMapping(value = ENTITY_URI + "/jasperserver/login")
public ResponseEntity<Map<String, String>> loginJasperReportsServer() {
```

Figura 16: representación del estándar de codificación para los comentarios en las funciones.
Fuente: elaboración propia

- Los nombres de variables o funciones deben ser lo suficientemente descriptivos, sin exceder de 30 caracteres
- Evitar líneas de más de 80 caracteres, pues no son bien interpretadas por terminales y herramientas.

2.6. Descripción del sistema

Una vez concluida la disciplina de implementación se logra un componente para la obtención de reportes desde el SIRES con cada una de las funcionalidades descritas en el epígrafe 2.3.2 del presente capítulo.

En la figura 17 se muestra la funcionalidad generar reporte, desde la cual se obtienen los reportes definidos en los RF del componente. Por otra parte, la figura 18 muestra una vista del reporte Actualización del Registro Estatal de Entidades Agropecuarias No Estatales (REEANE) y del Registro Estatal de Unidades Básicas de Producción Cooperativa (REUCO), generado a partir del componente propuesto.

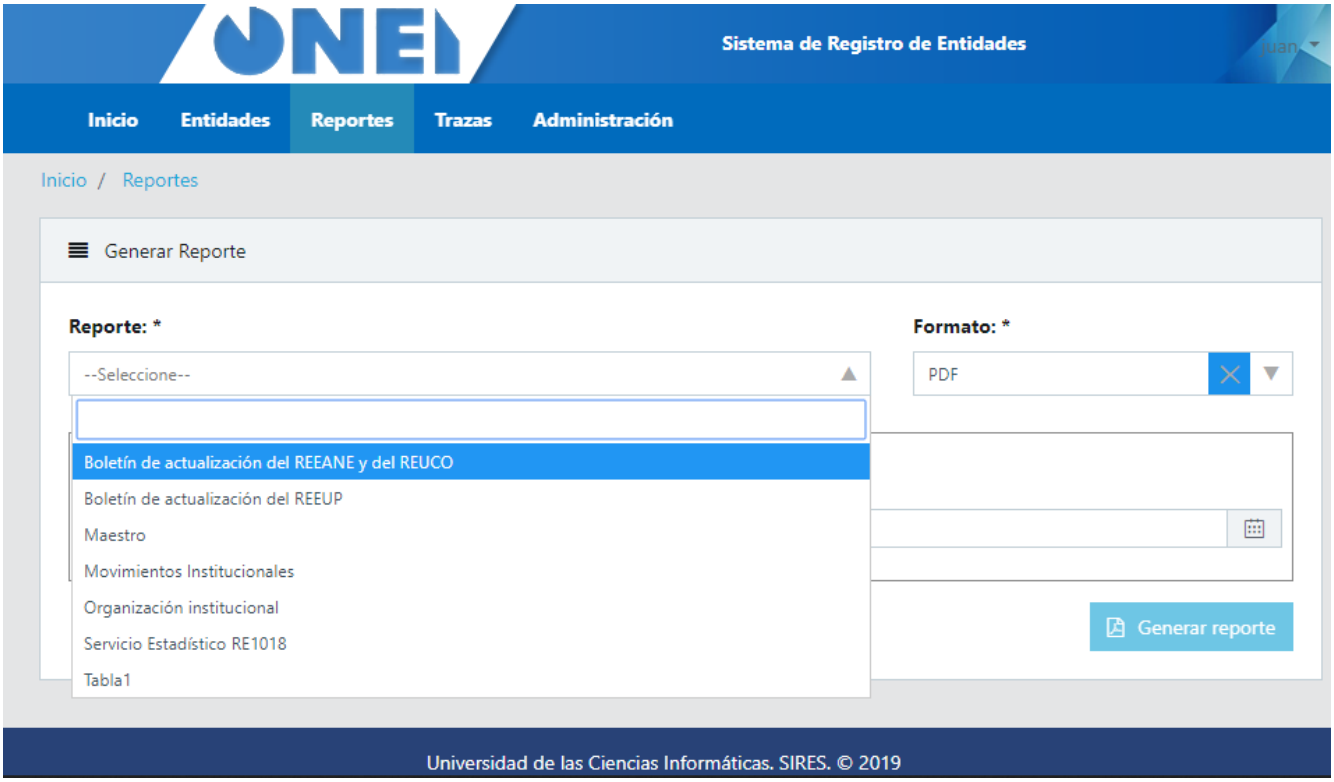


Figura 17: componente para generar reportes.
Fuente: elaboración propia

OFICINA NACIONAL DE ESTADÍSTICA E INFORMACIÓN
República de Cuba

Boletín de Actualización del REEANE y del REUCO

Boletín de actualización del REEANE y del REUCO

Fecha: 31-12-16 a 08-05-19

CÓDIGO	DESCRIPCIÓN	DIRECCIÓN	DPA	ORGANISMO	NAE	CULTIVO	TIPO
Incluir							
31856	5 DE SEPTIEMBRE	AVENIDA INDEPE	2205	171	420	5	CPA
14705	MANUEL BRITO MORALES	CARRETERA CE	2205	175	420	6	CPA
14707	SABINO PUPO	AVENIDA 168 NO	2207	156	430	7	UBPC
31872	JUAN GONZALEZ	CALLE LOMA NC	2205	154	430	6	UBPC
14702	COOPERATIVA PIZZETA DE OBISPC	CALLE PERLA N	2205	145	420	5	CPA
39082	JOSE MARTI	CALLE MAYÍA RC	2205	134	420	5	CPA
70500	COOPERATIVA HATUEY	AVENIDA ACOST	2205	166	430	6	UBPC
70499	COOPERATIVA EL TESORO	CALLE 1RA. NÚM	2205	170	430	6	UBPC

Boletín de actualización del RE

Figura 18: reporte actualización del REEANE y el REUCO en formato *excel*
Fuente: elaboración propia

En la figura 19 se muestra una imagen del *dashboard* que da cumplimiento al desarrollo del RF 9, a través del cual se visualiza de manera permanente el comportamiento de las estadísticas del proceso de registro de Entidades a través del SIRES.

Figura 19: pizarra de control (*dashboard*).
Fuente: elaboración propia

2.7. Conclusiones parciales

El empleo de la metodología *AUP* en su variación para la UCI en función de describir el proceso de desarrollo del componente propuesto, permitió organizar el desarrollo de la solución y generar los artefactos necesarios. Por otra parte, la aplicación de los patrones arquitectónicos MVC para la arquitectura del *frontend* y N-Capas para la arquitectura del *backend*, así como el uso de patrones de diseño, permitieron obtener una solución con poca dependencia entre clases, flexible al mantenimiento y a la introducción de cambios. De igual forma se emplearon estándares de codificación que contribuyen a la mantenibilidad del componente. Con la implementación del componente se dio cumplimiento a cada uno de los RF y RnF identificados en la disciplina de requisitos.

CAPÍTULO 3: VALIDACIÓN Y VERIFICACIÓN DE LA SOLUCIÓN PROPUESTA

3.1. Introducción

En el presente capítulo se muestran los resultados obtenidos luego de aplicar las técnicas de validación de requisitos, y las métricas para la validación del diseño de la solución propuesta, documentándose los resultados obtenidos en cada caso. Por otra parte, se define la estrategia de prueba aplicada a partir de las disciplinas establecidas por la metodología *AUP* variación UCI para el desarrollo de las pruebas. En la estrategia se establece que en cada disciplina se realicen pruebas a nivel de unidad, integración y sistema, con la aplicación de sus respectivos métodos y técnicas. El capítulo concluye con la verificación del cumplimiento del objetivo general de la investigación a partir de la definición de un conjunto de indicadores que permiten evaluar a través de un antes y un después al desarrollo del componente, la relación causa-efecto de la variable independiente sobre las variables dependientes de la investigación.

3.2. Validación de los requisitos

Con el objetivo de asegurar que el componente a desarrollar se corresponde con las necesidades del cliente, cada uno de los requisitos identificados fueron validados antes de llegar a la disciplina de análisis y diseño. Para la validación de estos se utilizaron las siguientes técnicas:

- **Generación de casos de prueba:** los casos de pruebas que responden a los RF del componente propuesto fueron fáciles de diseñar, lo que significa que cada uno de los RF se interpretan de forma correcta y pueden ser comprendidos satisfactoriamente por los desarrolladores en la disciplina de implementación. En el caso de la presente investigación se generaron un total de 8 descripciones de casos de prueba (DCP), en la tabla 3 se describe la DCP correspondiente al RF: generar reporte, el resto de las DCP generadas para la validación del componente se encuentran entre los artefactos entregables de la tesis.

Tabla 3: casos de prueba de la HU generar reporte

Escenario (EC)	Reporte	Formato	Fecha inicio	Fecha fin	Respuesta del sistema	Flujo central
EC 1.1: generar reporte con	V Boletín Actualización del REEUP	V *PDF*	V 15/01/2019	V 30/01/2019	El sistema permite generar el	

campos correctos					reporte correctamente.	
EC 1.2: Generar reporte con campos incompletos	I	V	V	V	El sistema muestra un mensaje indicando que: "Este campo es obligatorio" y no habilita el botón Generar reporte.	Reportes/Generar reporte
	(Vacío)	*PDF*	15/1/2019	30/01/2019		
	V	I	V	V	El sistema muestra un mensaje indicando que: "Este campo es obligatorio" y no habilita el botón Generar reporte.	
	Boletín Actualización del REEUP	(Vacío)	15/1/2019	30/01/2019		
	V	V	N/A	V	El sistema habilita el botón Generar reporte, permitiendo generarlo correctamente.	
	Boletín Actualización del REEUP	*PDF*	(Vacío)	30/01/2019		
	V	V	V	N/A	El sistema habilita el botón Generar reporte, permitiendo generarlo correctamente.	
	Boletín Actualización del REEUP	*PDF*	15/1/2019	(Vacío)		

Las celdas de la tabla contienen V, I, o N/A. V indica válido, I indica inválido, y N/A que no es necesario proporcionar un valor del dato en este caso, ya que es irrelevante.

Fuente: elaboración propia

- **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.

- **Construcción de prototipos de interfaz de usuario:** esta técnica permite hacer simulaciones del componente a implementar y brinda la posibilidad a los especialistas de tener una idea de cómo serán las interfaces del componente una vez desarrollado. En la tabla 2 del epígrafe 2.3.3 se muestra el prototipo no funcional correspondiente al RF: generar reporte.

3.3. Validación del diseño

Con el objetivo de comprobar la calidad del diseño se emplearon las métricas de diseño tamaño operacional de clases (TOC) y relaciones entre clases (RC).

3.3.1. Tamaño operacional de clases

La métrica TOC se aplica a cada una de las clases del diseño con el objetivo de medir la calidad de las mismas con respecto a su grado de responsabilidad, complejidad de implementación y reutilización (Pressman, 2010).
A continuación, se describen los pasos que se llevaron a cabo para aplicar la 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.
- Teniendo en cuenta los valores antes obtenidos, se determina la incidencia de los atributos de calidad en cada una de las clases, según los criterios expuestos en la tabla 4.

Tabla 4: rango de valores para medir la afectación de los atributos de calidad (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)

En las tablas 5 y 6 se muestran los datos obtenidos una vez aplicada la métrica TOC sobre cada una de las clases del diseño del componente propuesto. En la tabla 6 se utilizan las siguientes abreviaturas:

- **Resp:** para el atributo de Responsabilidad
- **Comp. Imp:** para el atributo de Complejidad de implementación
- **Reut:** para el atributo de Reutilización

Tabla 5: total de clases y promedio de procedimientos con la aplicación de la métrica TOC.

Total de clases	19
Promedio de procedimientos	4.526315789

Fuente: elaboración propia

En la figura 20 se muestran los resultados en por cientos obtenidos a partir del análisis de los datos mostrados en las tablas 5 y 6, los cuales permiten evaluar los atributos: responsabilidad, complejidad de implementación y reutilización con la aplicación de la métrica TOC.

Tabla 6: resultados obtenidos por clases luego de aplicada la métrica TOC.

No.	Clase	Cantidad de procedimientos	Resp	Comp. Imp	Reut
1	*JasperServerConfig.java*	1	Baja	Baja	Alta
2	*ReportesController.java*	1	Baja	Baja	Alta

3	*EntidadController.java*	16	Media	Media	Media
4	*InicioResourceAssambler.java*	1	Baja	Baja	Alta
5	*EntidadServiceImp.java*	15	Media	Media	Media
6	*IrregularidadServiceImpl.java*	6	Baja	Baja	Alta
7	*Entidad.java*	15	Media	Media	Media
8	*EntidadResourceAssambler.java*	4	Baja	Baja	Alta
9	*IrreglaridadController.java*	6	Baja	Baja	Alta
10	*Irregularidad.java*	1	Baja	Baja	Alta
11	*IrregularidadResourceAssambler.java*	2	Baja	Baja	Alta
12	*gestionar-reporte.component.ts*	8	Baja	Baja	Alta
13	*notificacion.component.ts*	2	Baja	Baja	Alta
14	*reporte.ts*	1	Baja	Baja	Alta
15	*reporte.service.ts*	1	Baja	Baja	Alta
16	*dash-board.component.ts*	3	Baja	Baja	Alta
17	*dashboarditem.component.ts*	1	Baja	Baja	Alta
18	*dashboard.service.ts*	1	Baja	Baja	Alta
19	*dashboarditem.ts*	1	Baja	Baja	Alta

Fuente: elaboración propia

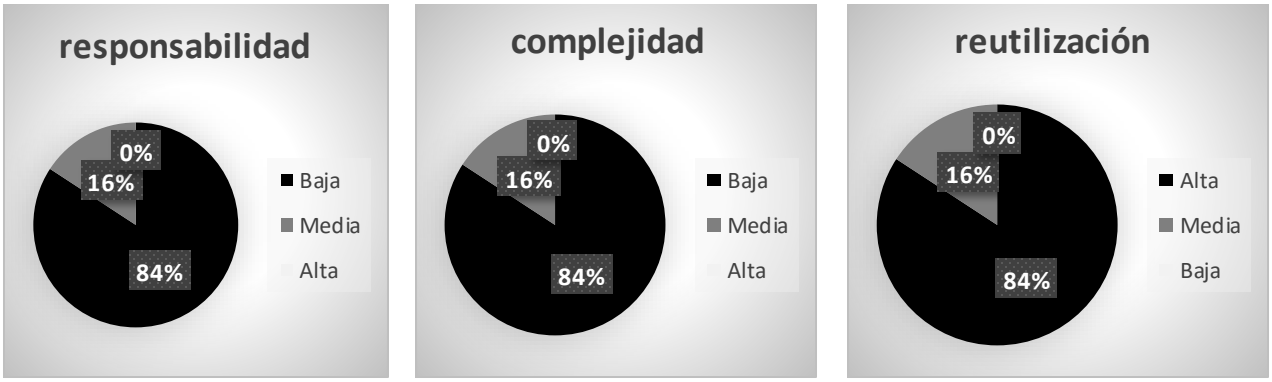


Figura 20: representación en (%) de los resultados de la aplicación de la métrica TOC.
Fuente: elaboración propia

Con la aplicación de la métrica TOC se obtienen los siguientes resultados:

- Responsabilidad: los resultados fueron satisfactorios, teniendo en cuenta que el 84 % de las clases tienen una responsabilidad baja.

- Complejidad de implementación: los resultados fueron positivos, pues se demostró que el 84 % de las clases tienen una complejidad baja.

- Reutilización: los resultados obtenidos fueron satisfactorios, demostrándose que ninguna clase tiene una baja reutilización, mientras el 84 % tiene una reutilización alta.

Haciendo un análisis de los resultados obtenidos con la aplicación de la métrica TOC se concluye que el diseño del componente propuesto tiene una baja responsabilidad y complejidad de implementación de las clases que lo componen, así como un alto grado de reutilización de estas. Todo lo anterior facilita la obtención de una solución informática con poca dependencia entre clases, flexible al mantenimiento y a la introducción de cambios.

3.3.2. Relaciones entre clases

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).
A continuación, 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.
- Teniendo en cuenta los valores antes obtenidos se determina la incidencia de los atributos de calidad en cada una de las clases, según los criterios expuestos en la tabla 7.

Tabla 7: 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 las tablas 9 y 10 se muestran los datos obtenidos una vez aplicada la métrica RC sobre cada una de las clases del diseño del componente propuesto. En la tabla 10 se utilizan las siguientes abreviaturas:

- **Acop:** para el atributo de acoplamiento.
- **Comp. Mant:** para el atributo de complejidad de mantenimiento.
- **Reut:** para el atributo de reutilización.
- **Cant. Prueb:** para el atributo cantidad de pruebas.

Tabla 8: total de clases y promedio de asociaciones de uso con la aplicación de la métrica RC.

Total de clases	19
Promedio de asociaciones de uso	1.578947368

Fuente: elaboración propia

Tabla 9: resultados obtenidos por clases luego de aplicada la métrica RC.

No.	Clase	Cantidad de Relaciones de Uso	Acop.	Comp. Mant.	Reut.	Cant. Prueb.
1	*JasperServerConfig.java*	1	Bajo	Baja	Alta	Baja
2	*ReportesController.java*	1	Bajo	Baja	Alta	Baja
3	*EntidadController.java*	3	Alto	Media	Media	Media

4	*InicioResourceAssambler.java*	2	Medio	Media	Media	Media
5	*EntidadServiceImp.java*	1	Bajo	Baja	Alta	Baja
6	*IrregularidadServiceImpl.java*	1	Bajo	Baja	Alta	Baja
7	*Entidad.java*	2	Medio	Media	Media	Media
8	*EntidadResourceAssambler.java*	1	Bajo	Baja	Alta	Baja
9	*IrreglaridadController.java*	1	Bajo	Baja	Alta	Baja
10	*Irregularidad.java*	1	Bajo	Baja	Alta	Baja
11	*IrregularidadResourceAssambler.java*	2	Medio	Media	Media	Media
12	*gestionar-reporte.component.ts*	2	Medio	Media	Media	Media
13	*notificacion.component.ts*	2	Medio	Media	Media	Media
14	*reporte.ts*	1	Bajo	Baja	Alta	Baja
15	*reporte.service.ts*	2	Medio	Media	Media	Media
16	*dash-board.component.ts*	3	Alto	Media	Media	Media
17	*dashboarditem.component.ts*	1	Bajo	Baja	Alta	Baja
18	*dashboard.service.ts*	2	Medio	Media	Media	Media
19	*dashboarditem.ts*	1	Bajo	Baja	Alta	Baja

Fuente: elaboración propia

En la figura 21 se muestran los resultados en por cientos obtenidos a partir del análisis de los datos mostrados en las tablas 9 y 10, los cuales permiten evaluar los atributos: acoplamiento, complejidad de mantenimiento, reutilización y cantidad de pruebas, con la aplicación de la métrica RC.

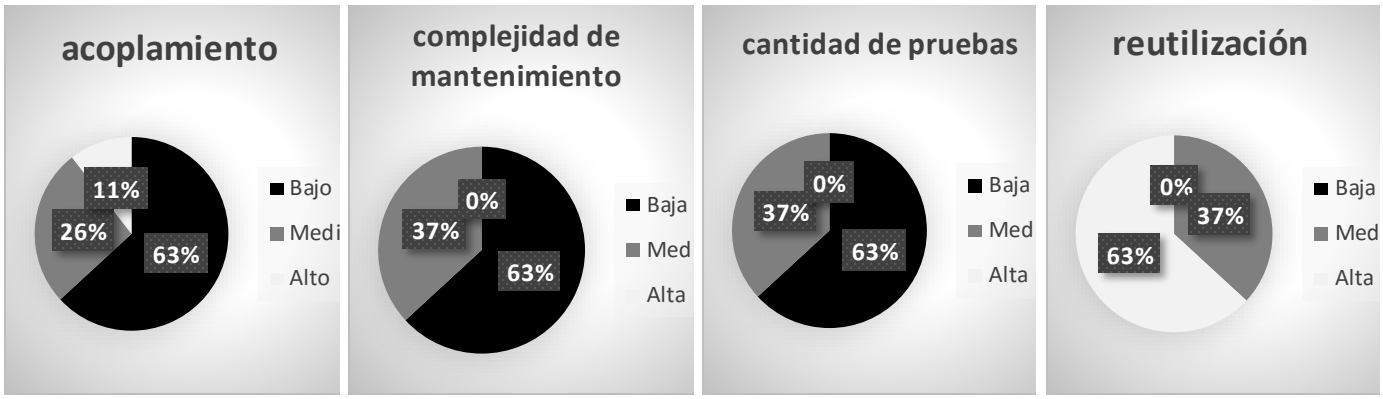


Figura 21: representación en (%) de los resultados de la aplicación de la técnica RC.
Fuente: elaboración propia

Con la aplicación de la métrica RC se obtienen los siguientes resultados:

- Acoplamiento: los resultados obtenidos son positivos teniendo en cuenta que el 63 % de las clases poseen un bajo acoplamiento.

- Complejidad de mantenimiento: los resultados mostrados en la figura anterior, demuestran que el 63 % de las clases del componente propuesto presentan una complejidad de mantenimiento baja, facilitando así las futuras actividades de soporte sobre el mismo.

- Cantidad de pruebas: los resultados demuestran que el 63 % de las clases poseen un bajo grado de esfuerzo a la hora de realizar cambios, rectificaciones o pruebas de software sobre ellas.

- Reutilización: los resultados obtenidos fueron satisfactorios, demostrándose que el 63 % de las clases tienen una reutilización alta.

Con los resultados obtenidos una vez aplicada la métrica RC se concluye que el diseño del componente propuesto tiene 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.

3.4. Pruebas 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 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 del componente propuesto en la presente investigación se define una estrategia de prueba de software aplicada a partir de las disciplinas establecidas para el desarrollo de las pruebas por la metodología *AUP* variación UCI. En la estrategia se define que en cada disciplina se realicen pruebas a nivel de unidad, integración y sistema, con la aplicación de sus respectivos métodos y técnicas. Las pruebas a nivel de aceptación no se realizan teniendo en cuenta que el alcance de la tesis solo comprende el desarrollo del componente para ser integrado al SIRES, software que aún no se ha terminado de aceptar con el cliente.

3.4.1. 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 del componente propuesto las pruebas internas se ejecutan a nivel de unidad, integración y sistema. A continuación, se detalla cómo se realizó el desarrollo de las mismas.

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 tesis en este nivel de prueba se aplicaron los métodos de caja blanca y caja negra.

Método de caja blanca:

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

La técnica de ruta básica tiene como objetivo comprobar que cada ruta 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 ruta independiente sea ejecutada 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 estos agrupan las principales funcionalidades de la solución. A continuación, se describe la aplicación del método de caja blanca sobre la funcionalidad *obtenerEntidadesPage* (ver figura 22) de la clase *EntidadController.java*. Se toma como muestra esta funcionalidad teniendo en cuenta que es una de las de mayor complejidad, en la cual se definen un grupo de condicionales que posibilitan la obtención de varias rutas como resultado de la ejecución de la técnica aplicada.

```
1  public ResponseEntity<PagedResources<EntidadResource>> obtenerEntidadesPage(Pageable pageable, @RequestBody EntidadForm entidadForm) {
       try {
           Page<Entidad> entidadPage = entidadService.listarEntidadesPage(entidadForm, pageable, entidadForm.esEstidades());
2          if (entidadForm.getCodigo() != null
3              || entidadForm.getIdntipoentidad() != null
4              || entidadForm.getDescripcion() != null) {
5              if (entidadPage.getTotalElements() != 0)
6                  return ResponseEntity.ok()
                           .body(pagedResourcesAssembler.toResource(entidadPage, entidadResourceAssembler));
               else
7                  return ResponseEntity.ok()
                           .headers(HeaderUtil.infoAlert( mensaje: "La búsqueda no arrojó resultados."))
                           .body(pagedResourcesAssembler.toResource(entidadPage, entidadResourceAssembler));
           } else
8              return ResponseEntity.ok()
                       .body(pagedResourcesAssembler.toResource(entidadPage, entidadResourceAssembler));
       } catch (Exception ex) {
9          return ResponseEntity.badRequest()
               .headers(HeaderUtil.badRequestAlert( mensaje: "Error obteniendo las " + ENTITY_NAME + "es en el servidor.")).build();
       }
10 }
```

Figura 22: método *obtenerEntidadesPage*
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 del diagrama de flujo.
- Regiones: son las áreas delimitadas por aristas y nodos.

En la siguiente figura se muestra el grafo de flujo obtenido:

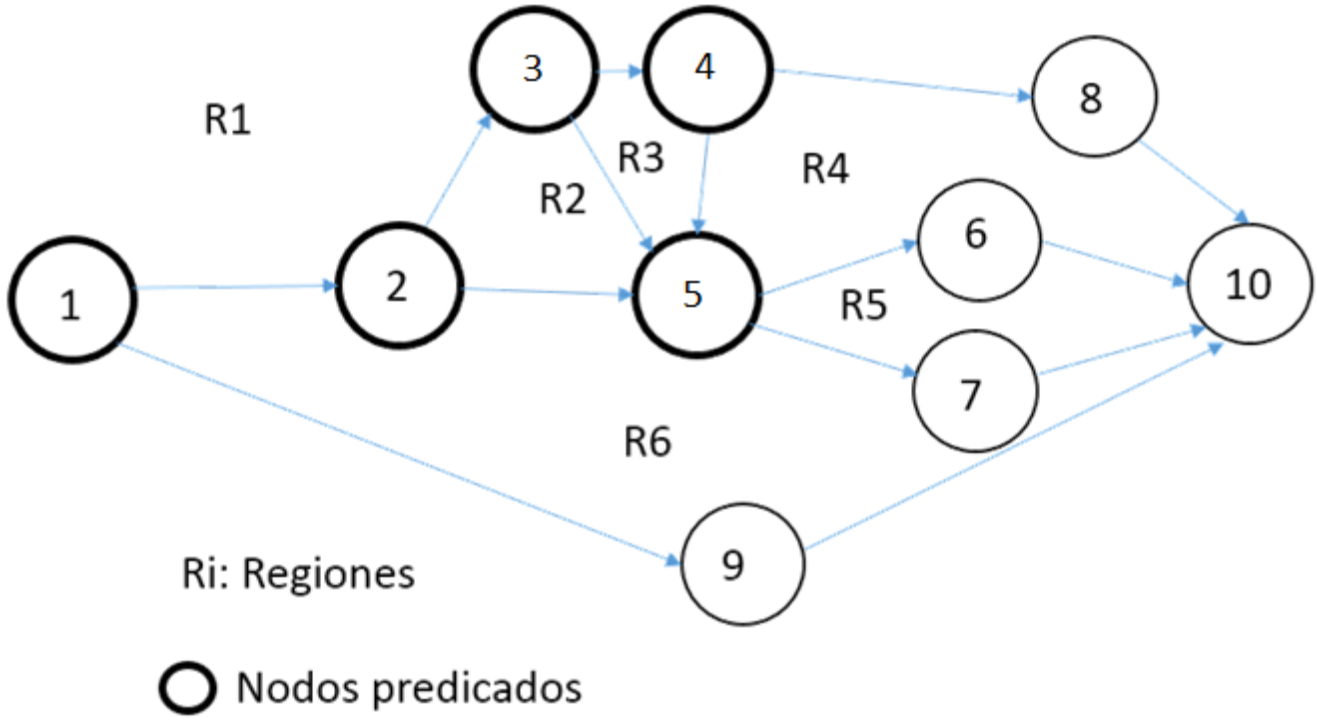


Figura 23: grafo de flujo del método *obtenerEntidadesPage*
Fuente: elaboración propia

2) Calcular la complejidad ciclomática:

El valor calculado por la complejidad ciclomática define el número de rutas independientes del conjunto básico 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 una vez (Pressman, 2010).

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

- El número de regiones del grafo de flujo corresponde a la complejidad ciclomática.

- 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 del gráfico de flujo.
- La complejidad ciclomática V(G) para un grafo de flujo G también se define como
  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 23, la complejidad ciclomática puede calcularse usando cada una de las vías anteriormente descritas:

1. El grafo de flujo tiene 6 regiones, por tanto, V(G) = 6
2. V(G) = (14 aristas - 10 nodos) +2 = 6
3. V(G) = 5 nodos predicado + 1= 6

Por tanto, la complejidad ciclomática del grafo de flujo de la figura 23 es 6.

3) Determinar un conjunto básico de rutas linealmente independientes:

El valor de V(G) proporciona la cota superior sobre el número de rutas linealmente independientes a través de la estructura de control del programa (Pressman, 2010). En el caso de la funcionalidad *obtenerEntidadesPage*, se definen 6 rutas básicas:

Ruta básica # 1: 1, 2, 5, 6, 10
Ruta básica # 2: 1, 2, 5, 7, 10
Ruta básica # 3: 1, 2, 3, 4, 8,10
Ruta básica # 4: 1, 9, 10
Ruta básica # 5: 1, 2, 3, 5, 6, 10
Ruta básica # 6: 1, 2, 3, 4, 5, 6, 10

4) Obtención de casos de prueba (CP):

Una vez definidas las rutas, se procede a diseñar los casos de prueba para cada una de las rutas básicas obtenidas. A continuación, en la tabla 10 se presenta el caso de prueba definido para la ruta básica # 2.

Tabla 10: caso de prueba de la ruta básica # 2.

Descripción	Permite conocer cuando no existe una determinada Entidad o irregularidad luego de ser listada desde el *dashboard* .
Condición de ejecución	La Entidad o irregularidad que se desea obtener no se encuentre entre las listadas.
Entradas	Se introduce el código, el tipo o la descripción de una Entidad o irregularidad que no se haya listado desde el *dashboard*.
Resultados esperados	Se debe mostrar una alerta a través del mensaje "La búsqueda no arrojó resultados", especificando que la Entidad o irregularidad buscada no se encuentra entre las listadas desde el *dashboard*.

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 se demuestra que todas las funcionalidades del componente se ejecutan satisfactoriamente, quedando libres de código repetido o innecesario.

Método de caja negra:

El método de caja negra se centra en los requisitos funcionales del software. Es decir, permite al ingeniero de software derivar conjuntos de condiciones de entrada que ejercitarán por completo todos los requisitos funcionales de un programa. El método de caja negra no es una opción frente a caja blanca. Es, en cambio, un enfoque complementario que tiene probabilidades de describir una clase diferente de errores de los que se identifican con los métodos de caja blanca (Pressman, 2010).

El método de caja negra se ejecuta a partir del desarrollo de pruebas funcionales, con la intención de identificar errores en las siguientes categorías (Pressman, 2010):

- Funciones incorrectas o faltantes.
- Errores de interfaz.
- Errores en estructuras de datos o en acceso a bases de datos externas.
- Errores de comportamiento o desempeño.
- Errores de inicialización y término.

Para desarrollar el método de caja negra se encuentra el uso de las técnicas (Pressman, 2010):

- Partición de equivalencia: divide el campo de entrada en clases de datos que tienden a ejercitar determinadas funciones del software.
- Análisis de valores límites: prueba la habilidad del programa para manejar datos que se encuentran en los límites aceptables.

Con el propósito de evaluar a nivel de equipo de desarrollo el correcto funcionamiento y diseño del componente propuesto, se realizaron pruebas funcionales a nivel de unidad. Estas pruebas permitieron comprobar que el componente responde a cada uno de los RF y RNF sobre los cuales fue implementado. El desarrollo de estas pruebas se realizó aplicando el método de caja negra con el uso de las técnicas partición de equivalencia y análisis de valores límites. Como herramientas de apoyo se utilizaron para el caso de las pruebas funcionales las DCP. En la tabla 3 se puede consultar la DCP correspondiente al RF: generar reporte. El resto de las DCP generadas para la validación del componente se encuentran entre los artefactos entregables de la tesis. A continuación, en la figura 24 se muestran los resultados de las pruebas funcionales a nivel de unidad.

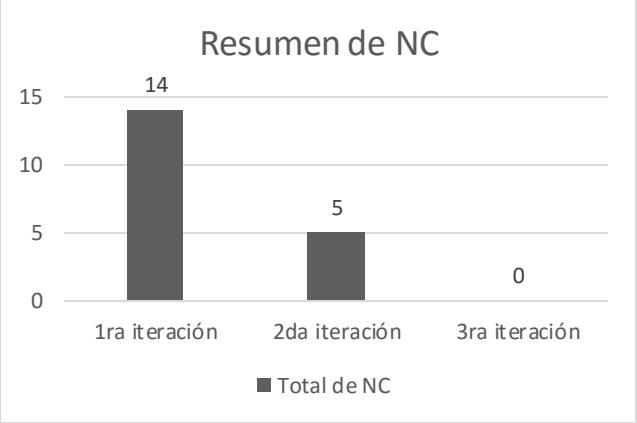


Figura 24: no conformidades detectadas al aplicar el método de caja negra a nivel de unidad
Fuente: elaboración propia

Como muestra la figura 24, en la primera iteración se detectaron un total de 14 No Conformidades (NC). Las NC se clasifican en 4 de ortografía, 3 de redacción, 4 de funcionalidad y 3 de validación. Luego en una segunda iteración se identificaron 5 nuevas NC, distribuidas en 2 de validación, 1 de funcionalidad, así como 2 de consistencia y estándares. En la tercera iteración los resultados fueron satisfactorios, obteniéndose cero NC. Este resultado demuestra que las funcionalidades del componente cumplen con cada uno de los RF y RNF.

Pruebas a nivel de integración y de sistema

Las pruebas a nivel de integración tienen como objetivo identificar errores introducidos por la combinación de programas o componentes probados unitariamente, para asegurar que la comunicación, enlaces y los datos compartidos ocurran apropiadamente. Se diseñan para descubrir errores o completitud en las especificaciones de las interfaces (Pressman, 2010).

Las pruebas a nivel de sistema tienen como objetivo verificar que se han integrado adecuadamente todos los elementos del sistema y que realizan las operaciones apropiadas funcionando como un todo. Es similar a la prueba de integración, pero con un alcance más amplio (Pressman, 2010).

Con el propósito de evaluar a nivel de equipo de desarrollo la integración del componente propuesto con el SIRES y su correcto funcionamiento una vez acoplado a este software, se realizaron pruebas funcionales a nivel de integración y de sistema, aplicando el método de caja negra con el uso de las técnicas partición de equivalencia y análisis de valores límites, utilizando las mismas DCP usadas en las pruebas a nivel de unidad. A continuación, en la figura 25 se muestran los resultados de la aplicación del método.

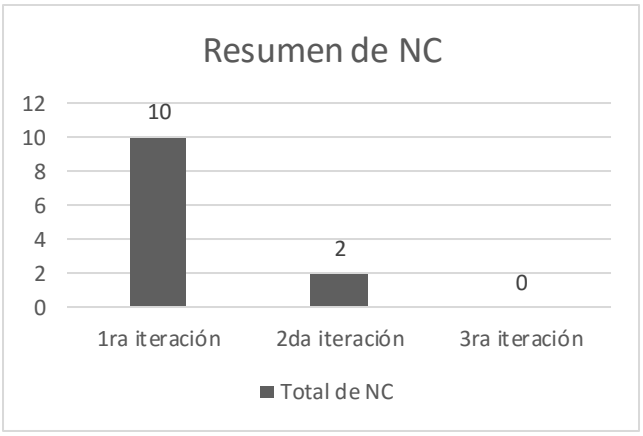


Figura 25: no conformidades detectadas al aplicar el método de caja negra a nivel de integración y sistema
Fuente: elaboración propia

Como muestra la figura 25, en la primera iteración se detectaron un total de 10 No Conformidades (NC), clasificadas 7 de funcionalidad y 3 de validación. Luego en una segunda iteración surgieron 2 de consistencia y estándares. En la tercera iteración los resultados fueron satisfactorios, obteniéndose cero NC. Este resultado demuestra que el componente se integró de forma satisfactoria con el SIRES y que su funcionamiento es exitoso una vez acoplado a este software.

3.4.2. 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).

Con el propósito de evaluar la integración del componente propuesto con el SIRES y su correcto funcionamiento una vez acoplado a este software, se realizaron pruebas de libración de tipo funcionales a nivel de integración y de sistema, utilizando el método de caja negra con el empleo de las técnicas partición de equivalencia y análisis de valores límites, aplicando las mismas DCP usadas en las pruebas internas. Estas pruebas fueron realizadas por el equipo de calidad de CEGEL. A continuación, en la figura 26 se muestran los resultados obtenidos con el desarrollo de estas pruebas.

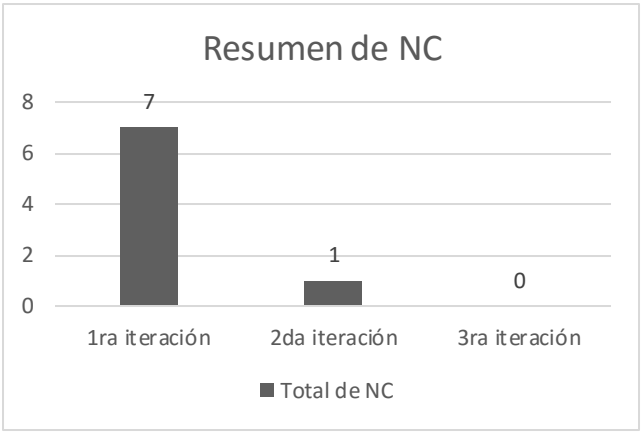


Figura 26: no conformidades detectadas en las pruebas de liberación
Fuente: elaboración propia

Como muestra la figura 26, en la primera iteración se detectaron un total de 7 No Conformidades (NC), clasificadas en 1 de redacción, 2 de funcionalidad, 2 de estética y diseño, así como 2 de consistencia y estándares. Luego en una segunda iteración persistió 1 de estética y diseño. En la tercera iteración los resultados fueron satisfactorios, obteniéndose cero NC, generándose así el Acta de Liberación por parte de la Asesora de Calidad de CEGEL (ver Anexo 4).

3.5. Verificación de los resultados de la investigación

Teniendo en cuenta que en la investigación realizada se define como idea a defender “si se desarrolla un componente para la obtención de reportes desde el SIRES se reducirá el tiempo y la posibilidad de errores en el procesamiento de los datos estadísticos que genera el proceso de gestión de Entidades en la ONEI; para analizar la relación causa efecto entre la variable independiente “si se desarrolla un

componente" y las variables dependientes "reducirá el tiempo y la posibilidad de errores en el procesamiento de los datos estadísticos que genera el proceso de gestión de Entidades en la ONEI. El autor de la presente investigación, define un conjunto de criterios de medida que permiten verificar cómo a través del componente se logra la relación entre ambas variables. La obtención de estos criterios se realiza a partir de las principales deficiencias identificadas en la situación problemática que dan lugar al desarrollo de la solución propuesta.

Para el caso de la evaluación de la variable tiempo se definen los criterios:

- Información oportuna: para el dominio de la investigación el autor define como información oportuna, la inmediatez con la cual los usuarios acceden a las estadísticas que genera el proceso gestión de Entidades.
- Proceso de generación de reportes estadísticos: describe el procedimiento que los especialistas de la DMI de la ONEI deben seguir para generar los diferentes reportes estadísticos que le son solicitados a esta dirección.

Por otra parte, para la evaluación de la variable posibilidad de errores se define el criterio:

- Proceso de cálculo de estadísticas: describe el procedimiento que los especialistas de la DMI de la ONEI deben seguir para realizar el cálculo de las estadísticas generadas durante el proceso de gestión de Entidades.

A continuación, en la tabla 11, se evalúan cada uno de los criterios de medida definidos anteriormente. La evaluación se realiza estableciendo un antes del desarrollo del componente y un después de obtenido el componente, con el propósito de verificar cómo mediante la solución propuesta se agilizan los análisis estadísticos y se evitan errores a la hora de analizar cada registro. Los datos tenidos en cuenta en la comparación fueron tomados a partir de un piloto realizado en la ONEI.

Tabla 11: evaluación de los criterios de medidas definidos.

Criterios de medida	Antes del componente	Después del componente
Información oportuna	Los especialistas de la DMI de la ONEI plantean que en la actualidad realizan manualmente el cálculo de las estadísticas generadas durante el proceso gestión de	El proceso de obtención de información estadística relacionada con el proceso de gestión de Entidades a través del componente desarrollado se reduce al tiempo que tarde

	Entidades. El proceso se realiza analizando el expediente en formato duro generado en cada registro. Esto trae como consecuencia que la información a obtener de estos análisis estadísticos puedan tardar como mínimo de 15 a 20 minutos.	el software en mostrar la información solicitada, el cual oscila entre unos 3 a 5 segundos como máximo, comprobados a partir del uso de la herramienta JMeter.
Proceso de generación de reportes estadísticos	Los especialistas de la DMI plantean que en la actualidad realizan los reportes de forma manual, teniendo que cuantificar los datos estadísticos manualmente, siguiendo el procedimiento descrito en el primer indicador, proceso que tardaba de 15 a 20 minutos y con riesgo de cometerse errores en el cálculo de algún indicador.	Con la aplicación del componente desarrollado el proceso de obtención de reportes estadísticos se reduce a solo tener que acceder a la funcionalidad generar reporte, definir el rango de fecha para el cual se desea obtener el reporte y generar el mismo de forma automática, sin riesgo de equivocación en los cálculos, pues estos ya son procesados por el propio sistema.

Proceso de cálculo de estadísticas	Los especialistas de la DMI plantean que en la actualidad realizan los cálculos de forma manual, teniendo que analizar cada expediente en formato duro generado de cada registro, volviéndose esta tarea propensa a errores de cálculo y a la omisión o cambio de algún datos que afecte a la veracidad de la información.	Con el uso del componente desarrollado el proceso de cálculo de estadísticas se reduce a solo tener que acceder a la funcionalidad generar reporte y obtener el reporte deseado de forma automática con cada uno de los cálculos ya procesados por el propio sistema.

Fuente: elaboración propia

La comparación realizada en la tabla anterior, a través de los criterios de medida antes definidos, demuestra que, utilizando el componente propuesto en la presente investigación, se reduce el tiempo y la posibilidad de errores en el procesamiento de los datos estadísticos que genera el proceso de gestión de Entidades en la ONEI.

3.6. Conclusiones parciales

La aplicación de técnicas para la validación de los requisitos garantizó llegar a las disciplinas de análisis y diseño e implementación, con requisitos libres de ambigüedades, que describen cada una de las necesidades del cliente. Por otra parte, el uso de métricas de validación del diseño certifica la obtención de un componente flexible al mantenimiento y a la introducción de cambios. De igual forma, la realización de pruebas internas y de liberación en los niveles de unidad, integración y sistema, con el empleo de sus respectivos métodos y técnicas, certifican que el componente obtenido cumple con cada uno de los RF y RNF que dieron lugar a su implementación, avalado por el acta de liberación emitida por el grupo de calidad de CEGEL. Con la verificación del resultado de la investigación, se demuestra el cumplimiento de la relación causa efecto de la variable independiente “si se desarrolla un componente” sobre las variables dependientes “se reduce el tiempo y la posibilidad de errores en el procesamiento de los datos estadísticos que genera el proceso de gestión de Entidades en la ONEI”.

CONCLUSIONES GENERALES

- El estudio de los referentes teóricos vinculados con soluciones informáticas para la gestión de datos estadísticos, permitió obtener características como la forma de generar reportes estadísticos, con el propósito de ser incorporadas al desarrollo del componente propuesto.
- El empleo de patrones de diseño y arquitectónicos, así como el uso de estándares de codificación en la implementación de la solución propuesta, permitieron obtener un componente, flexible al mantenimiento y a la introducción de cambios, que posibilita generar reportes estadísticos desde el SIRES.
- El desarrollo de pruebas de internas y de liberación en los niveles de unidad, integración y sistema, permitieron obtener un acta de liberación emitida por el grupo de calidad de CEGEL que corrobora el correcto funcionamiento del componente para la obtención de reportes desde el SIRES.
- Con la verificación de la relación causa efecto de la variable independiente sobre la variable dependiente de la investigación, se demuestra que con el desarrollo del componente propuesto se reduce el tiempo y la posibilidad de errores en el procesamiento de los datos estadísticos que genera el proceso de gestión de Entidades en la ONEI.

.

RECOMENDACIONES

- Incorporar al componente nuevos reportes que enriquezcan las funcionalidades del SIRES.
- Desarrollar mejoras en la arquitectura del componente que permitan el desarrollo de reportes dinámicos.

BIBLIOGRAFÍA REFERENCIADA

Alegsa. 2019. [En línea] 2019. http://www.alegsa.com.ar/Dic/framework.php.

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

deperu. 2019. deperu. [En línea] 2019. www.deperu.com.

eclac. 2019. Economic Commission for Latin America and the Caribbean. [En línea] 2019. www.cepal.org/.

Google. 2019. Angular. [En línea] 2019. angular.io/guide/dependency-injection.

—. 2019. Angular. [En línea] 2019. https://angular.io/guide/observables-in-angular.

—. 2019. Angular. [En línea] 2019. https://angular.io/docs.

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

inegi. 2019. [En línea] 2019. en.www.inegi.org.mx/Default.html.

Janssen, Thorben. 2019. Thoughts on JAVA. *Implementing the Repository pattern with JPA and Hibernate.* [En línea] 2019. https://thoughts-on-java.org/implementing-the-repository-pattern-with-jpa-and-hibernate/.

JasperSoft. 2019. JasperSoft Community. [En línea] 2019. community.jaspersoft.com/project/jasperreports-library.

JetBrains. 2019. JetBrains. [En línea] 2019. https://www.jetbrains.com/idea/.

—. 2019. JetBrains. [En línea] 2019. https://www.jetbrains.com/webstorm/.

Klipfolio Inc. 2019. Klipfolio. [En línea] 2019. https://www.klipfolio.com/resources/articles/what-is-data-dashboard.

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

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

Mozilla. 2019. MDN web docs mozilla. [En línea] 2019. developer.mozilla.org/en-US/docs/Web/JavaScript.

—. 2019. MDN web docs mozilla. [En línea] 2019. developer.mozilla.org/en-US/docs/Web/CSS/CSS3.

—. 2019. MDN web docs mozilla. [En línea] 2019. developer.mozilla.org/en-US/docs/Web/HTML.

ONEI. 2019. Portal Web de la ONEI. *Portal Web de la ONEI.* [En línea] 05 17, 2019. http://www.onei.cu/queeslaone.htm.

ORACLE. 2019. ORACLE. [En línea] 2019. https://docs.oracle.com/en/java/.

Pivotal. 2019. Spring. [En línea] 2019. spring.io/projects/spring-boot.

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

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

Rouse, Margaret. 2019. TeachTarget. *Database management system (DBMS).* [En línea] 2019. https://searchsqlserver.techtarget.com/definition/database-management-system.

—. 2019. TechTarget. *Integrated Development Environment (IDE).* [En línea] 2019. searchsoftwarequality.techtarget.com/definition/integrated-development-environment.

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

—. 2019. TechTarget. *pattern (design pattern).* [En línea] 2019. https://searchsoftwarequality.techtarget.com/definition/pattern.

Safari. 2019. Safari Books Online. *Oreilly.* [En línea] 2019. www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html.

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

SNI. 2019. [En línea] 2019. http://www.sni.org.pe/?page_id=872.

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

The PostgreSQL Global Development Group. 2019. PostgreSQL. *The World's Most Advanced Open Source Relational Database.* [En línea] 2019. https://www.postgresql.org/.

Typescript. 2019. Typescript. [En línea] 2019. www.typescriptlang.org.

UA. 2019. Universidad de Alicante. [En línea] 2019. https://si.ua.es/es/documentacion/asp-net-mvc-3/.

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

W3C. 2019. W3C. [En línea] 2019. https://www.w3.org/html/.