Universidad de las Ciencias Informáticas

Facultad 3

Título: *Herramienta para evaluar el grado de implementación de las características de calidad en el proceso de desarrollo de los sistemas informáticos de Gobierno Electrónico.*

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

Autores: Sarisleidy Rodríguez Bravo
Angel Felix Carabeo Mondejar

Tutores: Ing. Angel Luis Lecuona Rodríguez
Ing. Denia Madruga Hernández

La Habana, Junio de 2014

"La lucha por la calidad del producto es una lucha revolucionaria y de vanguardia, y nunca se equivoquen en pensar que por el hecho de ser revolucionario se puede dar al pueblo un producto de mala calidad, eso sería atentar contra la Revolución"

DECLARACIÓN DE AUTORÍA

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

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

Sarisleidy Rodríguez Bravo

_____

Firma del Autor

Angel Félix Carabeo Mondejar

_____

Firma del Autor

Ing. Angel Luis Lecuona Rodríguez

_____

Firma del Tutor

Ing. Denia Madruga Hernández

_____

Firma del Tutor

DATOS DE CONTACTO

Datos de los Tutores:

Nombre: Ing. Angel Luis Lecuona Rodríguez.

Ingeniero en Ciencias Informáticas.

Correo electrónico: allecuona@uci.cu

Nombre: Ing. Denia Madruga Hernández.

Ingeniero en Ciencias Informáticas.

Correo electrónico: dmadruga@uci.cu

Datos de los Autores:

Nombre: Sarisleidy Rodríguez Bravo

Correo electrónico: srbravo@estudiantes.uci.cu

Nombre: Angel Félix Carabeo Mondejar

Correo electrónico: afcarabeo@estudiantes.uci.cu

AGRADECIMIENTOS

Sarisleidy

A mi papá por ser el mejor padre del mundo, por estar a mi lado en todos los momentos de mi vida, por ayudarme y apoyarme en todo, gracias a ti hoy soy ingeniera ya que eres mi ejemplo a seguir, gracias por todo el amor que me brindaste por permitir que este sueño fuera posible.

A mi mamá porque no tengo palabras para expresar todo por lo que te quisiera dar gracias, por tu apoyo incondicional, por los dolores de cabezas y las preocupaciones que te di, por estar a mi lado siempre. Por decirme las palabras exactas cuando creía no poder más, por ser ejemplo de fuerza, valentía, coraje y tenacidad por no dejarte vencer por los obstáculos.

A mi hermanita querida que a pesar de haber llegado cuando ya no nos lo esperábamos eres lo mejor que nos ha pasado, por ti he tratado de ser un ejemplo a seguir y por ti mis deseos de seguir superándome para demostrarte que todo lo que uno se propone lo logra.

A mis abuelos Mima y Papi por ser mis más preciados tesoros, gracias por todo el amor y el cariño por todo lo que me han dado, por mimarme y en ocasiones malcriarme los quiero mucho mis viejitos lindos.

A la mejor tía del mundo, mi tía Yaya gracias por ser como una madre para mí, por ayudarme y apoyarme en todo momento, porque sé que puedo contar siempre contigo.

A mis primos Juan Gabriel, Rosmelis y Kevin porque ustedes son como mis hermanos, gracias por estar siempre a mi lado, los quiero mucho.

A mi tío Tayo gracias por todo lo que has hecho por mí, porque no hace falta compartir la misma sangre para llamarnos familia por toda la ayuda que me has dado, para mi eres el mejor tío del mundo.

A Neymis por ser mi amiga, para mi eres parte de mi familia, gracias por todo consejo dado por escucharme y por compartir buenos momentos, por preocuparte por mí.

A Yaremis que también te adoptamos como familia gracias por estar ahí y por poder contar contigo para lo que me hiciera falta.

A mis abuelos Pupi y Claribel por todo su amor y cariño.

A mi amiga del alma Ayme, porque a pesar de la distancia nuestra amistad ha sido más fuerte y duradera.

A mi nueva familia adoptiva de Poey, Wilma y Vilma, gracias por recibirme en su familia con los brazos abiertos por brindarme todo ese amor y cariño incondicional.

Agradecimientos

A mi sister Wanda gracias por acogerme como una hermana, sabes que para mí eres muy especial en mi vida, por aguantarme y entenderme en estos 5 años que pasamos, por estar a mi lado y apoyarme en todo momento por los consejos dados y por todo el amor y el cariño que me brindaste.

A mi amiga querida Jessica por estar a mi lado en las buenas y malas, por los consejos que me distes por toda la ayuda desinteresada, por estar siempre ahí para mí gracias.

A Kenia, a mi Fragolone, al Yipi, a Raynercito, a Yoalvis, a Leandro, a Ariam, a Pedritin, a Leo, a Nicolau, gracias por ser mis amigos por ayudarme en todo momento siempre que lo necesité.

A mi grupo 3502 por ser el mejor grupo de la UCI, de veras que fuimos la unión perfecta, gracias por los momentos de fiesta, de alegría y de estudio compartidos.

A Hertico por demostrarme su amistad en los momentos en que más lo necesitaba.

A las niñas de mi apto 56102 por ser mis compañeras, por compartir momentos de alegría y de sobre todo mucho estrés a Irene, Lisy y Rosi a ustedes muchas gracias.

A mi prima Yoanis porque a pesar de conocernos en la universidad pudimos forjar ese lazo familiar que ahora nos une, gracias por todo.

A mi querido tutor Angel Luis, gracias por ser más que mi tutor eres mi amigo, por ayudarme en todas las situaciones difíciles que tuve que enfrentar, por no dejarme sola en ningún momento, por desestresarme bastante, sin ti este sueño no hubiera sido posible y siempre estaré en deuda contigo.

A la tutora Denia por el apoyo brindado.

A mi compañero de tesis que a pesar de ser el último es el más importante, gracias por ser mi compañero y mi amigo en estos 5 años, por todo el apoyo y ayuda dada, por aguantarme en mis momentos de estrés que fueron varios, porque no me imaginaba hacer mi tesis sin ti y porque supiste siempre estar ahí para mí.

Angel Felix

A mis padres por darme todo su apoyo incondicional, por estar a mi lado en las buenas y malas.

A mis abuelos por quererme y ayudarme en todo lo que me hacía falta.

A toda mi familia en general.

A todos los amigos que hice en la universidad y para toda la vida.

A mi tutor Angel Luis por darnos tu apoyo incondicional en todo momento.

DEDICATORIA

Sarisleidy

A mi hermanita que ha sido mi inspiración para seguir adelante, por ser mi motor impulsor para ser cada día mejor.
A mi mamá por ser la mejor madre del mundo y por apoyarme en todas mi decisiones.
A mi papá que es mi ejemplo a seguir.
A Mima por ser la mejor abuela del planeta.
A toda mi familia que es el más grande tesoro que tengo.

Angel Felix

A mis padres por darme todo su apoyo incondicional, por estar a mi lado en las buenas y malas.

A mis abuelos por quererme y ayudarme en todo lo que me hacía falta.

A toda mi familia en general.

RESUMEN

En la actualidad, muchas empresas se dedican a la producción de software y su principal objetivo es desarrollar productos de calidad. Para lograr el éxito en este sentido no se puede obviar la correcta evaluación de las características de calidad durante el proceso de desarrollo de software para garantizar el buen resultado del producto final.

El presente trabajo está dedicado a la realización de una herramienta para evaluar el grado de implementación de las características de calidad durante el proceso de desarrollo de software de los sistemas informáticos del Centro de Gobierno Electrónico. Para ello se realiza un estudio del estado del arte de la temática a tratar y se hace un análisis de los sistemas informáticos que evalúan dicho grado de implementación, demostrándose que estos no resuelven los problemas del Centro. Para guiar el desarrollo del sistema a implementar se escogió como metodología de desarrollo de software SXP. También se realizaron pruebas de caja negra y caja blanca para validar las funcionalidades de la aplicación y se aplicaron las métricas para la validación del diseño. Además se obtuvo como resultado una herramienta que evalúa el grado de implementación de las características de calidad durante el proceso de desarrollo de software del Centro.

PALABRAS CLAVE

Características de calidad, Calidad de software, Grado de implementación, Metodología SXP

TABLA DE CONTENIDOS

INTRODUCCIÓN

En la actualidad el mundo está experimentando un cambio debido al aumento de las Tecnologías de la Informática y las Comunicaciones (TIC). Dichas tecnologías constituyen el núcleo central de las transformaciones que afrontan la economía y la sociedad, imponiéndoles a las personas modificar no sólo sus hábitos y patrones de conducta, sino incluso, su forma de pensar. Las TIC ofrecen un amplio marco de posibilidades, fundamentalmente en el desarrollo de la industria software para la informatización de la sociedad en que se vive, trayendo como consecuencia un aumento de la productividad y una mejor gestión de las empresas y organizaciones.

Con el desarrollo de la industria del software y el aumento de la competencia en el mercado, se le exige a las aplicaciones informáticas la presencia de altos niveles de calidad y perfeccionamiento. Todo esto provoca que si el producto es mejor, mayor demanda tendrá el mismo. En este contexto uno de los procesos clave es la calidad del software, existiendo dentro de la misma características como la funcionalidad, la eficiencia, la interoperabilidad, la usabilidad, la fiabilidad, la seguridad, la mantenibilidad y la portabilidad, que son las que favorecen el buen resultado del producto final. Dado el número de procesos informatizados que existen, se requiere de un alto nivel de calidad, por lo que el proceso de seguimiento y medición de la calidad del producto no representa una ventaja a la hora de competir en el mercado del software, sino una necesidad adentrarlo en el proceso de desarrollo, desde su inicio hasta su fin, si se quiere lograr un producto eficiente que compita con éxito.

En el marco de la informatización del país existen en Cuba proyectos con el fin de lograr que el desarrollo de software y servicios informáticos tengan la mayor calidad posible. Uno de estos proyectos es la Universidad de las Ciencias Informáticas (UCI), creada en el año 2002. La UCI es una gran casa de estudios informáticos, donde se hace énfasis al desarrollo de software como parte del proceso de aprendizaje. Los productos de software elaborados en la Universidad deben poseer una alta calidad y por ende deben cumplir con las expectativas de los clientes logrando así su satisfacción.

El proceso de desarrollo de software en la Universidad agrupa los proyectos por centros productivos, contando estos con un grupo que se encarga de velar por la calidad de los productos finales desarrollados en el centro.

Actualmente el Centro de Gobierno Electrónico (CEGEL) cuenta con una serie de guías que señalan los elementos a tener en cuenta para contribuir al aseguramiento de las características de calidad durante el ciclo de vida del software pero no se podía conocer el grado de implementación que tenía cada

característica a medida que se iba desarrollando el producto, aumentando la probabilidad de que no se hubiera tomado en cuenta.
La no evaluación del grado de implementación de las características durante el proceso de desarrollo dificulta el aseguramiento de la calidad al producto. Convirtiéndose el proceso de prueba en el único momento en el cuál se evalúan las características de calidad; trayendo consigo que las No Conformidades detectadas en ese momento sean de un costo superior al que representaría su detección en etapas tempranas del ciclo de vida.

A partir de la situación planteada anteriormente se define como **problema a resolver:**

¿Cómo evaluar el grado de implementación de las características de calidad durante el proceso de desarrollo de software de los sistemas informáticos de Gobierno Electrónico para contribuir al aseguramiento de las mismas en el producto final?

Partiendo del problema anteriormente planteado se ha determinado como **objeto de estudio:** Aseguramiento de la Calidad de software y como **objetivo general:** Desarrollar un sistema que permita evaluar el grado de implementación de las características de calidad durante el proceso de desarrollo de software de los sistemas informáticos de Gobierno Electrónico para contribuir al aseguramiento de las características de calidad en el producto final.

Tomando como **campo de acción**: Aseguramiento de las características de Calidad en el proceso de desarrollo de software.

Se plantea como **idea a defender**: Con el desarrollo de una herramienta que evalúe el grado de implementación de las características de calidad durante el proceso de desarrollo de software de los sistemas informáticos de Gobierno Electrónico se contribuye al aseguramiento de las características de calidad en los productos finales.

Para un mejor desarrollo de la investigación se desglosa el objetivo general en los siguientes **objetivos específicos**:

1. Establecer el marco teórico conceptual de la investigación.
2. Elaborar una lista de chequeo para comprobar la existencia de los elementos que definen las características de calidad.

3. Realizar el análisis y diseño de la solución propuesta.
4. Implementar la solución propuesta para dar respuesta a la situación problemática existente.
5. Validar la solución propuesta.

Para dar cumplimiento a los objetivos se plantea una serie de **tareas de la investigación**:

1. Definición de los conceptos más significativos en el dominio del problema.
2. Caracterización de los procesos para el desarrollo de software en CEGEL.
3. Caracterización de las principales tecnologías, herramientas y metodología a utilizar para el desarrollo de la solución.
4. Elaboración de una lista de chequeo para comprobar la existencia de los elementos que definen las características de calidad:
   - ✓ Funcionalidad
   - ✓ Eficiencia
   - ✓ Interoperabilidad
   - ✓ Usabilidad
   - ✓ Fiabilidad
   - ✓ Seguridad
   - ✓ Mantenibilidad
   - ✓ Portabilidad
5. Elaboración de los artefactos necesarios según la metodología de desarrollo de software seleccionada.
6. Aplicación de técnicas para la validación de requisitos obtenidos y el diseño de software.
7. Implementación de las funcionalidades definidas.
8. Verificar la solución mediante la aplicación de pruebas de caja negra y caja blanca.
9. Validación de las pruebas de usuario para la aceptación del producto.
10. Validación de la propuesta de solución de la investigación frente al problema planteado.

Los métodos utilizados para cumplir con las tareas a desarrollar son:

Métodos teóricos

- ✓ *Analítico-Sintético:* se utiliza en el estudio y análisis de la bibliografía. Pudiéndose realizar una correcta selección de los conceptos y definiciones para comprender mejor el problema, darle cumplimiento a los objetivos y así lograr resultados satisfactorios en la investigación.

- ✓ *Modelación*: para la elaboración de los diferentes diagramas que ayudaron a una mejor comprensión de la lógica del proceso, así como de las funcionalidades que debe cumplir el sistema.

Métodos empíricos

- ✓ *Entrevista:* es empleado en la recopilación de información para el desarrollo de la investigación.
- ✓ *Encuesta:* se utiliza para conocer los criterios de los especialistas durante el período de validación de la propuesta de la guía.
- ✓ *Medición*: permite obtener información numérica acerca de una propiedad o cualidad del objeto, donde se comparan magnitudes medibles y conocidas. Se utiliza para evaluar la calidad del módulo, a través del uso de métricas y pruebas de calidad.

El presente trabajo de diploma está compuesto por 3 capítulos estructurados de la siguiente manera:

Capítulo 1. Fundamentación teórica.

En este capítulo se lleva a cabo el estudio del estado del arte donde se realiza un análisis de las diferentes herramientas para evaluar la calidad. Se enunciaron los principales conceptos relacionados con el tema, así como una selección de la metodología, lenguajes y herramientas usadas como apoyo para darle solución al problema planteado.

Capítulo 2. Características del sistema.

En este capítulo se realiza un estudio de una serie de guías existentes en CEGEL para evaluar el grado de implementación de las características de calidad para luego confeccionar una lista de chequeo que evalué el grado de implementación de las características en el centro. Además se elaboran los diferentes artefactos que genera la metodología a utilizar, por decisión de los autores también se generan otros artefactos para lograr una mayor claridad y profundidad en la investigación.

Capítulo 3. Implementación y validación.

El capítulo se enfoca en la solución del problema en cuestión, generando los diagramas de despliegue y componentes. Se hace un análisis de los resultados obtenidos a través de pruebas realizadas al sistema y otras técnicas de validación.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

En el presente capítulo se analizan los conceptos fundamentales para el dominio del problema. Además de un estudio detallado sobre las características de calidad a evaluar y las herramientas existentes en el mundo que evalúan el grado de implementación de las mismas. Se fundamenta la selección de la metodología de desarrollo de software empleada para la construcción del sistema, así como las principales herramientas que serán utilizadas para realizar el diseño y la implementación del sistema.

1.1. Calidad de Software

El término calidad de software juega un papel importante en el mundo de la Industria del software el cual se desarrolla día a día a un ritmo muy acelerado. Esto implica que las empresas no escatimen esfuerzos para que el término esté vigente con un alto nivel en todos sus productos logrando como resultado elevar la competitividad, influyendo positivamente en la decisión de un cliente para escoger un producto que satisfaga sus necesidades.

La Real Academia de la Lengua Española define calidad como: *"Propiedad o conjunto de propiedades inherentes a algo, que permiten apreciarla como igual, mejor o peor que las restantes de su especie"* [1].

Según Pressman: *"Concordancia con los requisitos funcionales y de rendimiento explícitamente establecidos con los estándares de desarrollo explícitamente documentados y con las características implícitas que se espera de todo software desarrollado profesionalmente"* [2].

Para la ISO 25000 calidad de software es: *"Capacidad de un producto de software de satisfacer las necesidades explícitas e implícitas cuando es utilizado en condiciones específicas"*[3].

Para la IEEE es: *"Grado con el que un sistema, componente o proceso cumple los requisitos especificados y las necesidades o expectativas del cliente o usuario"* [4].

Después de realizar un estudio de los conceptos de calidad de software antes mencionados, se determinó que la definición de La Real Academia de la Lengua Española no se enfoca en el contexto en el que se hace la investigación ya que es un concepto más general por lo que necesitan aterrizarlo en el entorno de la informática y la producción de software. Pressman y la ISO 25000 hablan solamente de la calidad del producto, en ningún momento manifiestan la importancia de llevar la calidad del proceso de desarrollo, que en gran medida es relevante para poder obtener un producto final con calidad, además de que

tampoco hablan de la importancia de tener en cuenta los requisitos no funcionales, solo Pressman habla del rendimiento. Por tanto se concluye que la definición dada por la IEEE es la más completa ya que indica elementos importantes a tener en cuenta como el sistema, el proceso de desarrollo, los requisitos especificados de manera general que pueden ser funcionales y no funcionales y las necesidades o expectativas del cliente que en ocasiones no se toman en cuenta.

1.1.1. Aseguramiento de las características de calidad

Asociado al concepto de calidad del software surgen términos como el aseguramiento de la calidad, el cual se puede definir como *"el esfuerzo total para plantear, organizar, dirigir y controlar la calidad en un sistema de producción con el objetivo de dar al cliente productos con la calidad adecuada"*. Es simplemente asegurar que la calidad sea lo que debe ser [5].

Según la Norma ISO 9000, el aseguramiento de la calidad *"es la parte de la gestión de la calidad orientada a proporcionar confianza en que se cumplirán los requisitos de la calidad"* [6].

Según Roger Pressman el aseguramiento de la calidad del software *"es un conjunto de actividades planificadas y sistemáticas necesarias para aportar la seguridad en que el producto (software) satisfará los requisitos dados de calidad"* [7].

Luego de haber analizado todos estos criterios, los autores consideran que el aseguramiento de las características de calidad es un diseño planificado de acciones o actividades a realizar a lo largo del ciclo de vida de software que se requieren para garantizar las características de calidad en el producto final.

La búsqueda de la calidad en los sistemas ha propiciado la creación de modelos para evaluar, mejorar y asegurar la calidad del software. El modelo de calidad es el centro en torno al cual se establecen los sistemas para la evaluación de la calidad del producto. En estos modelos se determinan las características de calidad que se van a tener en cuenta a la hora de evaluar las propiedades de un producto software determinado.

1.2. Modelos de Calidad a nivel de producto

Un modelo de calidad es *"un conjunto de prácticas vinculadas a los procesos de gestión y el desarrollo de proyectos. Este modelo supone una planificación para alcanzar un impacto estratégico, cumpliendo con*

los objetivos fijados en lo referente a la calidad del producto o servicio" [8].

La ISO/IEC define un estándar como *"un conjunto de reglas que se usan de manera habitual que controla cómo las personas deben desarrollar y gestionar los* recursos*, productos, servicios, tecnologías, procesos y sistemas"* [9].

1.2.1. Modelo de Calidad ISO/IEC 9126-1

ISO 9126 se publicó en 1991 con el objeto de "promover un entorno que permita la evaluación de la calidad del software". En 1994 se entendió que era necesaria una modificación y adaptación de la norma. En esta versión es dónde se introducen por primera vez los conceptos de calidad interna y calidad externa. Además se creó una nueva norma (ISO 14598) que asumía el modelo del proceso de evaluación antes incluido en la norma ISO 9126 [10].

La Norma ISO 9126 está dividida en cuatro partes:

- ISO 9126-1. Modelo de calidad.
- ISO 9126-2. Métricas externas.
- ISO 9126-3. Métricas internas.
- ISO 9126-4. Calidad en las métricas de uso.

Sólo la primera parte, ISO 9126-1, es un estándar aprobado y publicado, siendo el resto de partes de la norma, informes que se encuentran en la fase llamada *Technical Report* (TR).

ISO 9126 – 1 define un modelo de calidad basado en:

- Calidad interna y externa.
- Calidad de uso.

Calidad del software (interna y externa)					
Funcionalidad	**Fiabilidad**	**Usabilidad**	**Eficiencia**	**Mantenibilidad**	**Portabilidad**
Adecuación	Madurez	Fácil comprensión	Comportamiento frente al tiempo	Facilidad de análisis	Adaptabilidad
Exactitud	Tolerancia a fallos	Fácil aprendizaje	Uso de recursos	Capacidad para cambios	Facilidad de instalación
Interoperatividad	Capacidad de recuperación	Operatividad	Adherencia a normas	Estabilidad	Coexistencia
Seguridad	Adherencia a normas	Software atractivo		Facilidad para pruebas	Facilidad de reemplazo
Adherencia a normas		Adherencia a normas		Adherencia a normas	Adherencia a normas

Figura 1: **Modelo ISO/IEC 9126 [10].**

Valorándose la importancia que tiene dentro del desarrollo de un producto de software, las subcaracterísticas Seguridad e Interoperabilidad pasaron a ser parte de las características ya existentes, lo que redefinió el modelo de calidad propuesto por la norma, convirtiéndose en un nuevo modelo de calidad que contiene ocho características. Este modelo es propuesto en la norma SQuaRE ISO/IEC 25010.

1.2.2. Modelo de Calidad ISO/IEC 25010 SQUARE (*Software Product Quality Requeriments and Evaluation*)

El modelo SQuaRE es una revisión de ISO/IEC 9126 – 1:2001 y conserva las mismas características de calidad de software [10].

Nace con el objetivo de responder a las necesidades de los usuarios a través de un conjunto de documentos unificados que cubren los tres procesos de calidad complementarios: especificación de requisitos, medidas y evaluación. El modelo de calidad SQuaRE categoriza la calidad del software en características que se subdividen en subcaracterísticas y atributos de calidad. La ISO/IEC 25010 define un modelo de la calidad del producto compuesto por ocho características (que se dividen en subcaracterísticas) que se relacionan con propiedades estáticas de software y propiedades dinámicas del sistema informático. El modelo es aplicable en sistemas informáticos y en productos de software. La mejora en la calidad de los productos de software implica que las organizaciones deben ser creativas y

estrictas en cuanto a las actividades responsables de elevar la calidad, teniendo en cuenta la comunicación entre los sistemas implicados.

Calidad del software (interna y externa)							
Funcional-idad	**Seguridad**	**Interopera-bilidad**	**Fiabilidad**	**Usabilidad**	**Eficiencia**	**Mantenibili-dad**	**Portabilidad**
Adecuación	Acceso resistente	Compatibi-lidad OSI	Madurez	Adecuación	Tiempo de respuesta	Capacidad de análisis	Adaptabilid-ad
Exactitud	Copia resistente	Compatibili-dad Software	Tolerancia a fallos	Capacidad de aprendizaje	Utilización de recursos	Capacidad a cambios	Capacidad a instalación
Adherencia a normas	Cifrabilidad	Compatibili-dad de datos	Recuperabi-lidad	Operabilidad	Adherencia a normas	Estabilidad	Capacidad a coexistencia
	Protección resistente	Trazabilidad	Adherencia a normas	Util		Capacidad a testing	Capacidad a reemplazo
	Robustez	Adherencia a normas		Agradable		Adherencia a normas	Adherencia a normas
	Adherencia a normas			Adherencia a normas			

Figura 2: **Modelo de calidad para la calidad interna y externa para la versión ISO/IEC 25010 (Julio 2007) [11].**

A continuación se exponen brevemente cada una de las características del nuevo modelo de calidad: [11]

Funcionalidad:

Representa la capacidad del producto software para proporcionar funciones que satisfacen las necesidades declaradas e implícitas, cuando el producto se usa en las condiciones especificadas.

Seguridad:

Representa la capacidad en que la información y los datos están protegidos de modo que las personas o sistemas no autorizados no saben leer ni modificarlos y personas o sistemas autorizados no se les niega el acceso a ellos.

Interoperabilidad:

Representa la capacidad de dos o más sistemas o componentes para intercambiar información y/o llevar a cabo sus funciones requeridas cuando comparten el mismo entorno hardware o software.

Fiabilidad:

Representa la capacidad de un sistema o componente para desempeñar las funciones especificadas, cuando se usa bajo unas condiciones y periodo de tiempo determinados.

Usabilidad:

Representa la capacidad en que un producto puede ser usado por determinados usuarios para conseguir objetivos específicos con efectividad, eficiencia y satisfacción en un contexto de uso específico.

Eficiencia:

Representa la capacidad del producto de software para proporcionar el desempeño relativo a la cantidad de recursos utilizados bajo determinadas condiciones.

Mantenibilidad:

Representa la capacidad de eficacia y eficiencia con la que el producto se puede modificar.

Portabilidad:

Representa la capacidad en que un sistema o componente puede ser transferido de manera eficaz y eficiente a partir de un hardware, software u otro entorno operativo o la utilización de otro.

Se concluyó que la ISO/IEC 25010 es una revisión de la ISO/IEC 9126-1 y adquiere las mismas características de calidad del software y además añade dos nuevas características. Esta ISO es la que se va a utilizar a lo largo de la investigación.

1.3. Ciclo de vida del software según el Programa de Mejora

Como parte de la institucionalización que lleva a cabo la universidad es necesario realizar un estudio del ciclo de vida del proceso de desarrollo de software y utilizarlo como punto de partida para definir por cada una de las etapas del mismo, las actividades que se deben tener en cuenta para el aseguramiento de las características de calidad.

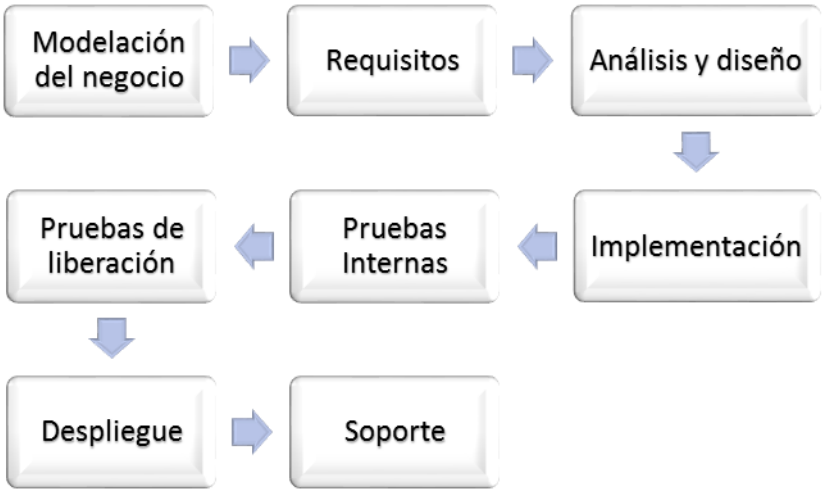


Figura 3. **Etapas del ciclo de desarrollo del software según el Programa de Mejora.**

Tabla 1. **Descripción de las etapas del ciclo de desarrollo del software según el Programa de Mejora.**

Modelado del Negocio (opcional)	El Modelado del Negocio es la disciplina destinada a comprender los procesos de negocio de una organización. Se comprende cómo funciona el negocio que se desea informatizar para tener garantías de que el software desarrollado va a cumplir su propósito.
Requisitos	El esfuerzo principal en la disciplina Requisitos es desarrollar un modelo del sistema que se va a construir. Esta disciplina comprende la administración y gestión de los requisitos funcionales y no funcionales del producto.
Análisis y Diseño	En esta disciplina, si se considera necesario, los requisitos pueden ser refinados y estructurados mediante un Modelo de Análisis para conseguir una comprensión más precisa de estos, y una descripción que sea fácil de mantener y ayude a la estructuración del sistema (incluyendo su arquitectura). Además, en esta disciplina se modela el sistema y su forma (incluida su arquitectura) para que soporte todos los requisitos, incluyendo los requisitos no funcionales. Los modelos desarrollados son más formales y específicos que el de análisis.
Implementación	En la implementación, a partir de los resultados del Análisis y Diseño se implementa el sistema.
Pruebas Internas	En esta disciplina se verifica el resultado de la implementación probando cada

	construcción, incluyendo tanto las construcciones internas como intermedias, así como las versiones finales a ser liberadas. Se deben desarrollar artefactos de prueba como: diseños de casos de prueba, listas de chequeo y de ser posibles componentes de prueba ejecutables para automatizar las pruebas.
Pruebas de Liberación	Pruebas diseñadas y ejecutadas por una entidad certificadora de la calidad externa, a todos los entregables de los proyectos antes de ser entregados al cliente para su aceptación.
Despliegue (opcional)	En esta área de desarrollo se procede a la entrega de la solución, así como a la instalación, configuración, pruebas y puesta en marcha del software en el entorno real del cliente. Estas pruebas incluyen pruebas de aceptación y pruebas pilotos. También deben realizarse en este periodo la capacitación y acompañamiento a los usuarios finales para asegurar que adquieran los conocimientos necesarios en la manipulación del software.
Soporte (opcional)	Por un tiempo limitado el proyecto ofrecerá un servicio para resolver conflictos y problemas de usabilidad y rendimiento del software entregado al cliente, suministrándole actualizaciones y correcciones a errores.

Durante cada una de las fases del proceso de desarrollo de software es de vital importancia tener en cuenta una serie de pautas que reflejen en qué grado de implementación están las características de calidad, para estos sería muy útil la elaboración de una lista de chequeo para su evaluación.

1.4. Lista de chequeo y grado de implementación de las características de calidad

Se entiende por grado que es "*Estado, valor o calidad que puede tener una persona o cosa en relación con otras y que puede ordenarse con otros estados, valores o calidades de mayor a menor o de menor a mayor*" [12].

Se entiende por implementación que es "*Es la forma y el método para llevar a cabo alguna tarea. En desarrollo de sistemas informáticos, la implementación es la etapa donde efectivamente se programa el sistema y en programación, la implementación es la programación de un determinado algoritmo en un lenguaje específico*" [13].

El grado de implementación de las características, se refiere al nivel o avance de implementación que pueden tener las características de calidad de software en un determinado proyecto y que pueden ser

medidas en cada una de las etapas de desarrollo. Se clasifican en Completamente Implementado (CI), Altamente Implementado (AI), Parcialmente Implementado (PI) o No Implementado (NI). Además el valor en el que se da se lleva a % y se ve en la escala que muestra la Tabla 2.

Tabla 2: **Grado de implementación de las características de calidad.**

Grado de implementación en %	
CI	100
AI	75,1-99,9
PI	25,1-75
NI	0-25

El grado de implementación de las características de calidad se va a calcular en una lista de chequeo.

Se entiende por lista de chequeo o *check-list*, en inglés, a *"un listado de preguntas, en forma de cuestionario que sirve para verificar el grado de cumplimiento de determinadas reglas establecidas a priori con un fin determinado*"[14]. De manera general las listas de chequeo permiten evaluar una actividad o acción.

Para llevar a cabo la evaluación del grado de implementación de las características de calidad y facilitar su desarrollo existen en el mundo varios sistemas informáticos que permiten la evaluación de las mismas y el cálculo del grado de implementación.

1.5. Valoración de las herramientas

Actualmente en el mundo existe la tendencia de elevar la calidad al máximo nivel en la industria del software para lograr que los procesos que se realizan a diario transcurran de forma positiva. La correcta evaluación de las características de calidad en los sistemas informáticos propicia que se cumpla esta tendencia con un elevado nivel de satisfacción.

Los proyectos del Centro de Gobierno Electrónico no están exentos a esta tendencia por lo que se quiere desarrollar un sistema que tribute a una correcta evaluación de las características de calidad durante todo el proceso de desarrollo de software.

Herramientas existentes en el mundo

MECHDAV

Esta herramienta es creada con la finalidad de medir en forma práctica y real la calidad de los ambientes visuales. Este modelo estandarizado para la evaluación técnica de las herramientas RAD (*Rapid Applications Development*) permite hacer un análisis de las herramientas de desarrollo de aplicaciones en ambientes visuales basado en el cumplimiento de las características de calidad de software siguientes: funcionalidad, confiabilidad, usabilidad, eficiencia, portabilidad y calidad en uso. La herramienta computacional MECHDAV permite la aplicación práctica del modelo de evaluación creado, y prueba su viabilidad. Está dirigida también a organizaciones, empresas y usuarios finales que necesiten seleccionar, de manera eficaz y fácilmente, el software más adecuado para desarrollar aplicaciones en ambientes visuales [15].

Herramienta sobre el grado de implementación de iniciativas del Plan Estratégico creado por CECAS[1]

Esta herramienta mide en qué grado se tienen implementadas las diferentes Iniciativas Estratégicas (IE) del Plan Estratégico de la mediación. El objetivo es priorizar, analizar y seleccionar las iniciativas estratégicas con las puntuaciones más bajas como punto de partida en el que trabajar. Trata de un sencillo cuestionario de 24 preguntas. Una vez contestadas, la herramienta muestra una gráfica de barras sobre el grado de implementación de cada una de las iniciativas [16].

De forma general los sistemas anteriormente analizados presentan como limitaciones:

- Trabajan solamente con 6 características de calidad.
- Trabajan con métricas, por tanto se evalúan al final del proceso de desarrollo de software.
- Trabajan con el grado de implementación pero no se adecuan a las características de calidad.

De los sistemas descritos anteriormente, aunque trabajan con algunas características de calidad, listas de chequeo y guías, se puede afirmar que no pueden ser utilizados por CEGEL ya que no poseen una forma íntegra de evaluar todas las características de calidad, además que hacen uso de modelos estandarizados y de métricas por tanto no se adecua a las necesidades del centro.

[1] Centro de Estudios del Consejo General de los Colegios de Mediadores de Seguros

De lo planteado anteriormente surge la necesidad de crear una herramienta que permita evaluar el grado de implementación de las características de calidad en el Centro de Gobierno Electrónico para que sus productos tengan la mayor calidad posible.

1.6. Selección de las tecnologías, herramientas y metodologías

En la actualidad, se hace extensivo el uso de todo tipo de soluciones informáticas para facilitar el manejo y procesamiento de diferentes informaciones. Uno de los factores de importancia es la selección de las metodologías, lenguajes y herramientas que se emplean en su creación.

1.6.1. Metodologías de Desarrollo de Software

Durante el desarrollo de software, pueden aparecer diversos contratiempos o cambios en el negocio que impactan significativamente el proceso de ejecución de todas las actividades y la creación de los artefactos. Por lo que proveer al equipo de trabajo de un conjunto de procedimientos, técnicas, herramientas y soporte documental como medios para enfrentar dichas situaciones, puede facilitar la estructuración y dirección del desarrollo en función de la obtención de un resultado más acorde a las necesidades y expectativas de los clientes, así como alineado con las normas y prácticas internacionales para la producción de software. Para alcanzar tal fin, existen diversas metodologías clasificadas en dos categorías: *Ágiles* y *Tradicionales,* de las cuales a continuación se muestra una tabla comparativa:

Tabla 3: **Diferencias entre las metodologías ágiles y las tradicionales [17].**

Metodologías Ágiles	Metodologías Tradicionales
Basadas en Heurísticas provenientes de prácticas de producción de código.	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo.
Especialmente preparadas para cambios durante el proyecto.	Poseen cierta resistencia a los cambios.
Proceso menos controlado, con pocas normativas y restricciones.	Proceso mucho más controlado, con numerosas políticas/normas.
No existe contrato tradicional o al menos es bastante flexible.	Existe contrato prefijado.
El cliente es parte del equipo de desarrollo.	El cliente interactúa con el equipo de

	desarrollo mediante reuniones.
Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio.	Grupos grandes y posiblemente distribuidos.
Debido al tamaño del equipo de desarrollo, se necesitan pocos roles.	Debido al tamaño del equipo de desarrollo, existen numerosos roles especializados.
Pocos artefactos.	Muchos artefactos.
Menos énfasis en la arquitectura del software.	La arquitectura del software es esencial y se expresa mediante modelos.

Teniendo en cuenta que para la presente investigación, el equipo de desarrollo posee solo dos integrantes y que la planificación del cronograma de ejecución establece poco tiempo para darle respuesta al problema en cuestión, la metodología de desarrollo a emplear será de tipo *Ágil*. Dentro de las distintas metodologías ágiles que existen se encuentran SCRUM (Gestión Ágil de Proyectos) y XP (Programación Extrema) debido a las características que presentan hacen que se ajusten a las necesidades para el desarrollo del software a implementar, además del estudio de una personalización de estas dos metodologías que se ha utilizado en proyectos de la Universidad de las Ciencias Informáticas con el nombre de SXP. A continuación se muestran algunas características de estas metodologías.

XP

La metodología ágil *Extreme Programing* o Programación Extrema (XP), fue desarrollada por Kent Beck. Consiste básicamente en ajustarse estrictamente a una serie de reglas que se centran en las necesidades del cliente para lograr un producto de buena calidad en poco tiempo. Es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en el desarrollo de software. Promueve el trabajo en equipo, preocupándose en todo momento del aprendizaje de los desarrolladores y estableciendo un buen clima de trabajo. Este tipo de método se basa en una retroalimentación continuada entre el cliente y el equipo de desarrollo con una comunicación fluida entre todos los participantes. Este tipo de programación es la adecuada para los proyectos con requisitos imprecisos, muy cambiantes y con un riesgo técnico excesivo. Está pensada para un grupo pequeño y muy integrado donde la comunicación sea más factible que en grupos de desarrollo grandes [18].

Scrum

SCRUM es una metodología ágil y liviana utilizada para administrar y controlar el desarrollo de un software. Está centrada en priorizar el trabajo, maximizando la utilidad de lo que se construye. Los requisitos y las prioridades se examinan y ajustan durante el proyecto en intervalos cortos y regulares. Esta metodología busca entregar un producto que resuelva las necesidades, aumentando la satisfacción del cliente.

La gestión de un proyecto en SCRUM se concentra en definir qué características debe tener el producto a construir, transformando cualquier obstáculo que pudiera impedir la tarea del equipo de desarrollo. Este proceso busca que los equipos sean los más prácticos y ágiles posibles, SCRUM posee un conjunto de reglas muy simples, basado en los principios de inspección continua, adaptación, auto-gestión e innovación. Teniendo como objetivo que el cliente se entusiasme y se comprometa con el proyecto, debido a que ve crecer el producto iteración tras iteración, localizando las herramientas para alinear el desarrollo con las metas de negocio de su empresa. Por otro lado, el equipo encuentra un ámbito propicio para desarrollar sus capacidades profesionales resultando en un incremento en la motivación de los integrantes del mismo [19].

SXP[2]

En la UCI se ha definido la metodología SXP, a partir de las metodologías XP y SCRUM. La misma ofrece una estrategia tecnológica, a partir de la introducción de procedimientos ágiles que permitan actualizar los procesos de software para el mejoramiento de la actividad productiva fomentando el desarrollo de la creatividad, aumentando el nivel de preocupación y responsabilidad de los miembros del equipo, y ayudando al líder del proyecto a tener un mejor control del mismo. Consiste en una programación rápida o extrema, cuya particularidad es tener como parte del equipo, al usuario final, pues es uno de los requisitos para llegar al éxito del proyecto [20].

La creación de SXP se centra principalmente en la utilización de SCRUM para lograr una correcta planificación y organización; mientras que XP respalda con sus prácticas todo el proceso de desarrollo y de esta forma se obtiene un proceso de software completo, caracterizándose por presentar una documentación discreta y de mayor dinamismo. SCRUM propone actividades que son adecuadas para la gestión de proyectos relativamente pequeños, sirviendo de soporte para acelerar el dinamismo identificado en XP.

[2] Siglas adoptadas a partir de la fusión de las metodologías Scrum y XP: SXP.

Teniendo en cuenta los elementos antes expuestos, referidos a las características de las metodologías XP, SCRUM y SXP, los autores de este trabajo seleccionan la metodología de desarrollo de software SXP.

SXP consta de 4 fases principales:

- **Planificación-Definición,** donde se establece la visión, se fijan las expectativas y se realiza el aseguramiento del financiamiento del proyecto.
- **Desarrollo**, donde se realiza la implementación del sistema hasta que esté listo para ser entregado.
- **Entrega**, puesta en marcha.
- **Mantenimiento**, donde se realiza el soporte para el cliente.

De cada una de ellas se despliegan 7 flujos de trabajo: concepción inicial, captura de requisitos, diseño con metáforas, implementación, prueba, entrega de la documentación, soporte e investigación, utilizándose este último por el equipo de desarrollo cuando sea necesario, es decir, es un flujo que se puede mover y utilizarlo en cualquier parte del ciclo de vida del proyecto [20].

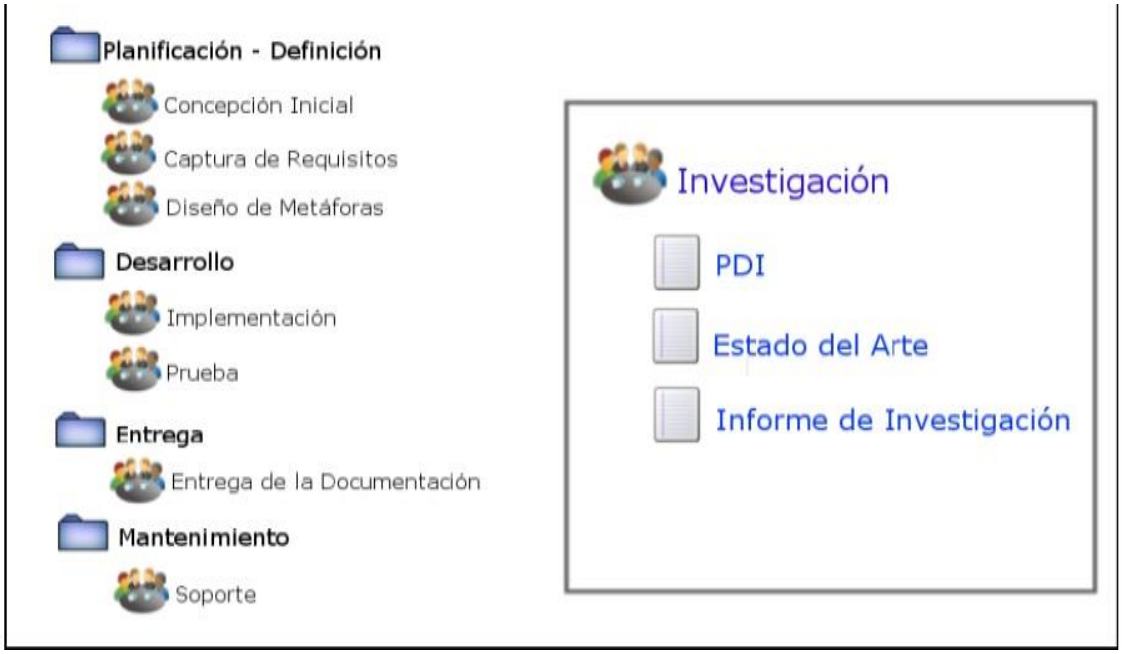


Figura 4: **Fases y flujos de trabajo de SXP [21].**

Los autores del presente trabajo elaboraron un esquema con estas fases, actividades y los artefactos que se generan en ellas, para lograr una mayor comprensión de la metodología a utilizar.

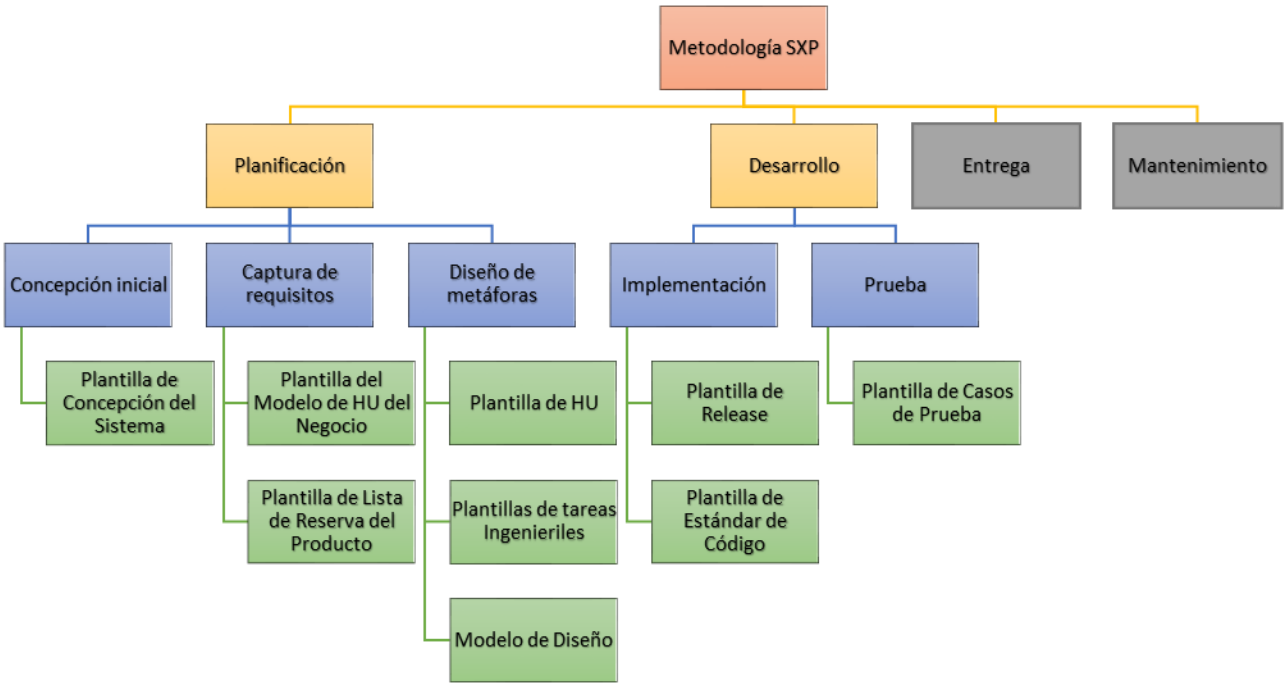


Figura 5. **Esquema de la Metodología SXP.**

Esta metodología es especialmente indicada para proyectos conformados por pequeños equipos y permite seguir de forma clara el avance de las tareas a realizar, facilitándoles a los líderes del desarrollo ver el progreso diario del trabajo. En el desarrollo de la investigación se hace uso de las fases de Planificación-Definición y de Desarrollo, generando cada uno de los artefactos definidos por la metodología en estas fases. Para hacer posible la modelación y representación de todos los elementos propuestos se hace indispensable el uso de herramientas que auxilien al equipo de desarrollo durante todas las etapas por las que transcurre.

1.6.2. Herramientas y tecnologías

1.6.2.1. Gestor de Base de Datos PostgresSQL 9.1

PostgresSQL, es un sistema de gestión de base de datos relacional orientada a objetos y libre, publicado bajo la licencia de Distribución de Software Berkeley (BSD, *Berkeley Software Distribution*, según sus siglas en inglés). PostgresSQL da la posibilidad de que mientras un proceso es escrito en una tabla, otros accedan a la misma tabla sin necesidad de bloqueos. Esta estrategia es superior al uso de bloqueos por tabla o por filas común en otras bases. Implementa el uso de retrocesos, subconsultas y transacciones, haciendo su funcionamiento mucho más eficaz. Posee la capacidad de comprobar la integridad referencial, así como también la de almacenar procedimientos en la propia base de datos [22].

Se decidió usar PostgresSQL, ya que es un sistema de gestión de base de datos relacional orientada a objetos y libre, publicado bajo la licencia de Distribución de Software Berkeley (BSD, *Berkeley Software Distribution*, según sus siglas en inglés). Además posee la capacidad de comprobar la integridad referencial, así como también la de almacenar procedimientos en la propia base de datos.

1.6.2.2. Herramientas para el modelado: Visual Paradigm for UML 8.0

Visual Paradigm es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Este software de modelado ayuda a una rápida construcción de aplicaciones de calidad, mejores y a un menor coste.

Permite dibujar todos los tipos de diagramas de clases, generar código desde diagramas, generar documentación, así como la generación de código inverso. Esta herramienta también proporciona abundantes tutoriales de UML, demostraciones interactivas de UML y proyectos UML [23].

Posee varias características importantes como son (multiplataforma, interoperabilidad, modelamiento de los requisitos, colaboración de equipo, generación de documentos, editor de detalles de casos de uso, ingeniería de código, modelado de procesos del negocio y modelamiento de bases de datos [23].

Visual Paradigm for UML se selecciona por la fuerte integración que posee con el lenguaje UML. Con esta herramienta se busca agilizar el diseño gracias a la interfaz sencilla que posee. Además, permite el trabajo en colaboración de los diseñadores mediante su propio controlador de versiones. También fue muy influyente que la universidad cuente con la licencia para su uso.

1.6.2.3. Lenguaje de modelado UML

UML (*Unified Modeling Language*) es un lenguaje que permite modelar, construir y documentar los elementos que forman un producto de software que responde a un enfoque orientado a objetos. Es un estándar internacional para definir, organizar y visualizar los elementos que configuran la arquitectura de

una aplicación orientada a objetos. UML puede considerarse como un lenguaje de modelado visual que permite una abstracción del sistema y sus componentes [24].

Con este lenguaje, se pretende unificar las experiencias acumuladas sobre técnicas de modelado e incorporar las mejores prácticas actuales en un acercamiento estándar. Se selecciona UML ya que es un lenguaje para el modelado enfocado a objetos y es soportado por la herramienta para el diseño seleccionada.

1.6.2.4. Lenguajes de Programación

Lenguajes del lado del Servidor:

PHP 5.4

PHP *(Hypertext Preprocessor)* es un lenguaje interpretado de alto nivel embebido en páginas html y ejecutado en el servidor. El uso de PHP es extremadamente simple para el principiante, pero a su vez, ofrece muchas características avanzadas para los programadores profesionales. PHP puede ser utilizado en cualquiera de los principales sistemas operativos del mercado, soporta la mayoría de los servidores web de hoy en día y ofrece soporte para varios gestores de bases de datos entre los que se destaca PostgreSQL. Su característica de ser software libre trae como consecuencia que implique menos costes y servidores más baratos [25].

Es además muy rápido, contiene una biblioteca nativa de funciones sumamente amplia e incluida, no requiere definición de tipos de variables ni manejo detallado del bajo nivel, presenta mejoras de rendimiento, es un lenguaje multiplataforma que funciona en todas las plataformas que soporten Apache. No es un lenguaje de marcas y su principal meta es permitir de forma rápida a los desarrolladores la generación dinámica de páginas. Soporta el uso de otros servicios que usen protocolos como IMAP, SNMP, NNTP, POP3, HTTP y derivados. También se pueden abrir sockets de red directos e interactuar con otros protocolos. Al ser un lenguaje libre dispone de una gran cantidad de características que lo convierten en la herramienta ideal para la creación de páginas web dinámicas. Permite la integración con varias bibliotecas externas, generar documentos en formato PDF (documentos de Acrobat Reader) y analizar código XML [25].

PHP es seleccionado por ser un lenguaje de programación del lado del servidor que posee disímiles características para facilitar el desarrollo de una aplicación web, es fácilmente integrable con el marco de trabajo escogido y permite la programación orientada a objeto.

Twig 2.1

Twig es un motor de plantillas que posee un amigable ambiente para el diseñador y desarrollador apegado a los principios de PHP, añadiendo funcionalidades a los entornos de plantillas. Posee una sintaxis corta y concisa, similar a la de otros lenguajes de programación, además, implementa un mecanismo de herencia de plantillas [26].

Twig es seleccionado por la integración que posee con el marco de trabajo escogido. Además, permite agilizar la construcción de plantillas de la vista, brindándoles estructuración y velocidad de ejecución del lado del cliente. Este lenguaje permitió compilar las plantillas hasta código PHP optimizado, lo que proporcionó rapidez durante la implementación. Brindó seguridad, pues permite evaluar el código de plantilla que no es confiable.

Lenguajes del lado del Cliente:

JavaScript 1.6

JavaScript es un lenguaje de programación utilizado para crear pequeños programas encargados de realizar acciones dentro del ámbito de una página web. Se trata de un lenguaje de programación del lado del cliente, porque es el navegador el que soporta la carga de procesamiento. El código Javascript es embebido directamente en el código html, haciendo fácil la creación de páginas web con contenido dinámico. Está diseñado para controlar la apariencia y manipular los eventos dentro de la ventana del navegador web y es soportado por la gran mayoría de los navegadores, lo que lo coloca como el lenguaje de programación web del lado del cliente más utilizado [27].

Con JavaScript se pueden crear efectos especiales en las páginas y definir interactividades con el usuario. El navegador del cliente es el encargado de interpretar las instrucciones JavaScript y ejecutarlas para realizar estos efectos e interactividades, de modo que el mayor recurso, y tal vez el único, con que cuenta este lenguaje es el propio navegador. Es un lenguaje de programación bastante sencillo y pensado para trabajar con rapidez, a veces con ligereza. Es un lenguaje con muchas posibilidades, permite la programación de pequeños scripts, pero también de programas más grandes, orientados a objetos, con funciones y estructuras de datos complejas [27].

JavaScript es seleccionado por la necesidad de contar con un lenguaje para validar el contenido del lado del cliente, así como para manejar algunos datos y eventos que son necesarios y requeridos en las funcionalidades a desarrollar en la aplicación.

HTML 5.0

HTML es el acrónimo en inglés de *HyperText Markup Language* (en español se traduce como lenguaje de marcado de hipertexto). Es un lenguaje para escritura de hipertexto, es decir, documentos de texto estructurados, incluye enlaces (*links*) que conducen a otros documentos o a otras fuentes de información y permite la inclusión de información por formularios, entre otros.

CSS 3.0

CSS es un mecanismo simple que describe cómo se va a mostrar un documento en la pantalla, o cómo se va a imprimir, o incluso cómo va a ser pronunciada la información presente en ese documento a través de un dispositivo de lectura. Esta forma de descripción de estilos ofrece a los desarrolladores el control total sobre estilo y formato de sus documentos [28].

La novedad más importante que aporta CSS3, de cara a los desarrolladores de webs, consiste en la incorporación de nuevos mecanismos para mantener un mayor control sobre el estilo con el que se muestran los elementos de las páginas, algunos de estos mecanismos son el soporte de más color y una gama más amplia de las definiciones del color, bordes con degradados, bordes con imágenes, esquinas redondeadas o bordes redondeados, cajas con sombra, múltiples imágenes de fondo, nuevos selectores, entre otras características adicionales [28].

CSS3 es seleccionado por las ventajas y facilidades que aporta a la hora de trabajar las plantillas que se van a mostrar en la aplicación, permitiendo ahorro de código y tiempo a los desarrolladores en la elaboración de varios trucos para lograr confeccionar las interfaces como son requeridas por el cliente.

1.6.2.2. Entorno de Desarrollo Integrado (IDE) NetBeans 7.4

El NetBeans 7.4 es un IDE de desarrollo que permite desarrollar rápida y fácilmente aplicaciones de escritorio, móviles y aplicaciones web, así como trabajar con HTML5, JavaScript y CSS, posibilita programar en distintos lenguajes. Ofrece un excelente entorno para programar en PHP. Cuenta con las características de multiplataforma, integración con sistemas de control de versiones, integración con Marcos de Trabajo, depurar y ejecutar programas, importar y exportar proyectos. Es gratuito y de código abierto con una gran comunidad de usuarios y desarrolladores de todo el mundo [29].

1.6.2.3. Marco de Trabajo

Symfony2

Symfony es un framework PHP de desarrollo web, que permite desarrollar aplicaciones comerciales, gratuitas y/o de software libre. Consta de una serie de herramientas, componentes de software rápidamente integrables, lo que significa que se tendrá que escribir menos código, con menos riesgo de

error y con una mayor productividad. Debe ser utilizado sobre la versión 5.3 de PHP o superior. Posee un enfoque estructurado que puede parecer restrictivo al principio, pero en realidad permite a los desarrolladores trabajar de manera eficiente y eficaz en los aspectos más complejos de una tarea [30].
Se seleccionó debido a que garantiza la seguridad, evitando la inyección SQL[3]. Reutiliza conceptos y desarrollos exitosos de terceros y los integra como librerías para luego ser utilizadas. Incluye el uso de buenas prácticas que garantizan la estabilidad, facilidad de mantenimiento y actualización de las aplicaciones que se desarrollan. Es versátil porque está basado en componentes, flexible porque cada programador puede utilizarlo como desee, interoperable pues permite crear aplicaciones que se adapten exactamente a las necesidades de cada proyecto y que además se puedan integrar entre sí.

Doctrine 2.3

Doctrines es un marco de trabajo que proporciona persistencia transparente de objetos PHP, el cual se situa en la arte superior de una poderosa capa de abstraccion de base de datos [31].
Doctrine es una librería muy completa, muy configurable viene integrada por defecto con Symfony2 y presenta entre sus caracterisiticas proncipales las siguientes:

1. Generacion automatica del modelo.
2. Posibilidades de trabajar con YAML.
3. Buscadores magico.
4. Relaciones entre entidades.
5. Lenguajes DQL.

Doctrine es seleccionado por la necesidad de utilizar un marco de trabajo que permite trabajar con los objetos de la base de datos. Ademas viene integrado por defecto al marco de trabajo Symfony2 y permite la generacion automatica del modelo logrando agilizar el trabajo.

1.6.2.4. Servidor web Apache 2.0

Apache está diseñado para ser un servidor web potente y flexible, continuamente actualizado y adaptado a los nuevos protocolos (HTTP) y puede funcionar en la más amplia variedad de plataformas y entornos. Apache se ha adaptado siempre a una gran variedad de entornos a través de su diseño modular el cual permite elegir las características del servidor seleccionando que módulos se van a cargar, ya sea al

[3] Método de infiltración de código intruso que se vale de una vulnerabilidad informática presente en una aplicación en el nivel de validación de las entradas para realizar consultas a una base de datos.

compilar o al ejecutar el servidor. Tiene capacidad para servir páginas tanto de contenido estático, como de contenido dinámico a través de otras herramientas soportadas que facilitan la actualización de los contenidos mediante bases de datos, ficheros u otras fuentes de información [32].

Es un software de libre distribución, que publica su código fuente, lo que permite que cualquiera pueda modificarlo y colaborar así a su desarrollo. Tiene interfaz con todos los sistemas de autenticación. Facilita la integración de plugins de los lenguajes de programación de páginas web dinámicas más comunes. Tiene integración en estándar del protocolo de seguridad SSL y provee interfaz a todas las bases de datos [32].

Apache es seleccionado por la necesidad de un servidor web compatible con la plataforma de desarrollo, que permita rapidez en la navegación y la seguridad e integridad de los datos con los que se trabajen. Además Apache permite configurar los módulos a utilizar en el servidor, logrando desechar elementos innecesarios para lograr mayor rapidez en la renderización de las páginas.

1.7. Conclusiones parciales

Al finalizar el presente capítulo se puede arribar a las siguientes conclusiones:

- Se expuso un marco conceptual para lograr una mayor comprensión del contexto donde se desarrolla la propuesta de solución.
- Se estudiaron las herramientas que evalúan las características de calidad con el uso de métricas y las que calculan el grado de implementación, pero ninguna evalúa el grado de implementación de las características en el proceso de desarrollo de software, quedando demostrada la necesidad de la creación de una herramienta que evalué dicho grado de implementación.
- Luego del estudio de diferentes metodologías se demuestra que la metodología SXP es adecuada para el equipo de trabajo y el tamaño de la solución.
- Las herramientas y tecnologías seleccionadas establecieron bases para el desarrollo de la propuesta de solución.

CAPÍTULO 2: CARACTERÍSTICAS DEL SISTEMA

En el presente capítulo se hace una descripción de la solución propuesta. Según la metodología seleccionada se explican los artefactos para las etapas de Planificación-Definición y Desarrollo. La implementación de la solución se basa en los principios y reglas de la metodología SXP utilizando las tecnologías y herramientas definidas. Además se generan otros artefactos que no son parte de la metodología usada pero se emplean para brindar una mayor comprensión del sistema a implementar. Se presentan también los patrones empleados durante el desarrollo así como la arquitectura con el objetivo de brindar un mayor entendimiento de la propuesta de solución.

2.1. Solución propuesta

Haciendo uso de herramientas y marcos de trabajo se propone la implementación de una herramienta para evaluar el grado de implementación de las características de calidad durante el proceso de desarrollo de software de los proyectos de CEGEL. La herramienta cumplirá con una serie de requisitos funcionales y no funcionales que lograran que la interacción del administrador y los usuarios con el sistema sea sencilla y eficiente, garantizándose así que se evalúen correctamente el grado de implementación de las características de calidad en el proceso de desarrollo.

2.2. Concepción inicial del sistema

En la etapa de concepción del sistema se genera la plantilla de concepción del sistema, en ella se especifican los aspectos generales organizativos y de concepción del sistema, su objetivo, principales involucrados y otros aspectos que permiten la posterior organización del desarrollo del proyecto. Esta plantilla se muestra en el Expediente de Proyecto.

2.3. Historias de Usuarios del Negocio

En la plantilla del modelo de historias de usuarios del negocio se describen los actores y trabajadores del negocio, además se presenta un diagrama de historias de usuarios del negocio que permite ver la relación entre los usuarios y las actividades que se realizan de ahí que se puedan obtener los requisitos. El diagrama se muestra a continuación divido en los dos trabajadores que interactúan con el negocio:

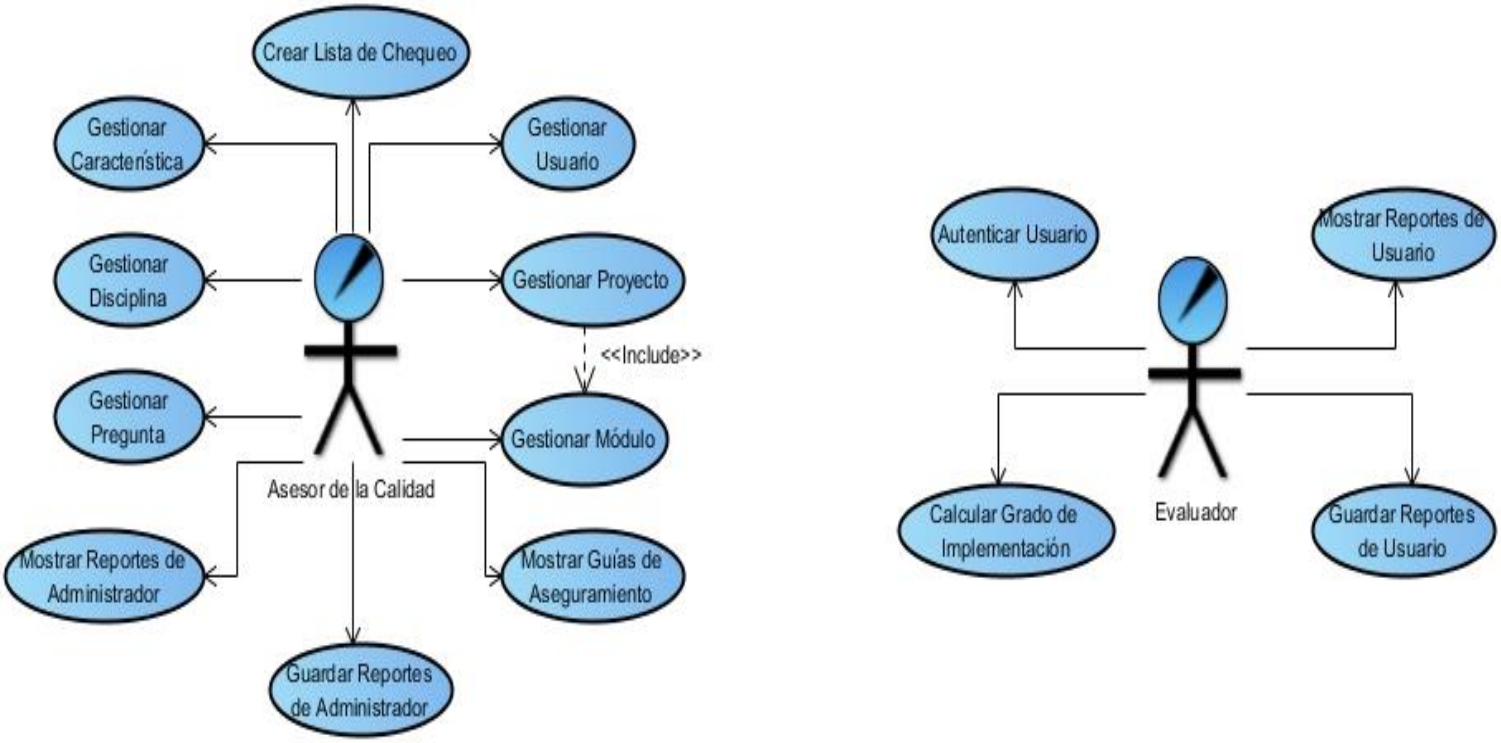


Figura 6: **Diagrama de Historias de Usuario del Negocio.**

Los restantes elementos de esta plantilla se describen en el Anexo 2.

2.4. Captura de requisitos

Los requisitos de software se encuentran clasificados en dos grupos: funcionales y no funcionales. Los requisitos funcionales son capacidades o condiciones que el sistema debe cumplir sin alterar la funcionalidad del producto y se mantienen invariables sin importar con qué propiedades o cualidades se relacionen. Los requisitos no funcionales son propiedades o cualidades que el producto debe tener; especifican propiedades como restricciones del entorno o de la implementación, rendimiento, dependencia de la plataforma, facilidad de mantenimiento, extensibilidad y fiabilidad. Los requisitos de forma general recogen las necesidades del cliente añadiendo el valor esperado por estos. Deben ser descritos de modo que sean comprendidos por los usuarios y clientes [7].

Para la adquisición de estos requisitos se emplearon técnicas que permitieron hacer este proceso de forma más adecuada y segura. De las técnicas existentes para la captura de requisitos se empleó la entrevista y la tormenta de ideas para determinar en conjunto los requisitos del sistema. Como resultado de la aplicación de estas técnicas se obtuvo la Lista de Reserva del Producto, estos elementos son descritos en los siguientes epígrafes. Una vez concluida la captura de requisitos y luego de someterse a 2 rondas de revisiones estos fueron aprobados y validados por el cliente.

2.4.1. Lista de Reserva del Producto (LRP)

En la Lista de Reserva de Producto, artefacto generado en la captura de requisitos, se describen los mismos como funcionalidades que el sistema debe cumplir en su desarrollo. Estos son descritos en la siguiente tabla:

Tabla 4: **Lista de Reserva del Producto (LRP).**

Prioridad	Ítem	Descripción	Estimación en sprint	Estimado por	Asignado a
Requisitos Funcionales Muy Alta					
	1	Autenticar Usuario	1	Programadores	Sarisleidy Rodríguez Angel Felix Carabeo
	2	Crear Lista de Chequeo	1	Programadores	Sarisleidy Rodríguez Angel Felix Carabeo
	3	Calcular Grado de Implementación	1	Programadores	Sarisleidy Rodríguez Angel Felix Carabeo
Alta					
	4	Insertar Característica	1	Programadores	Sarisleidy Rodríguez Angel Felix Carabeo
	5	Modificar Característica	1	Programadores	Sarisleidy Rodríguez Angel Felix Carabeo
	6	Eliminar Característica	1	Programadores	Sarisleidy Rodríguez Angel Felix Carabeo
	7	Mostrar Característica	1	Programadores	Sarisleidy Rodríguez Angel Felix Carabeo
	8	Insertar Disciplina	1	Programadores	Sarisleidy Rodríguez Angel Felix Carabeo
	9	Modificar Disciplina	1	Programadores	Sarisleidy Rodríguez Angel Felix Carabeo
	10	Eliminar Disciplina	1	Programadores	Sarisleidy Rodríguez Angel Felix Carabeo
	11	Mostrar Disciplina	1	Programadores	Sarisleidy Rodríguez

					Angel Felix Carabeo
	12	Insertar Pregunta	1	Programadores	Sarisleidy Rodríguez Angel Felix Carabeo
	13	Modificar Pregunta	1	Programadores	Sarisleidy Rodríguez Angel Felix Carabeo
	14	Eliminar Pregunta	1	Programadores	Sarisleidy Rodríguez Angel Felix Carabeo
	15	Mostrar Pregunta	1	Programadores	Sarisleidy Rodríguez Angel Felix Carabeo
Media					
	16	Insertar Proyecto	2	Programadores	Sarisleidy Rodríguez Angel Felix Carabeo
	17	Modificar Proyecto	2	Programadores	Sarisleidy Rodríguez Angel Felix Carabeo
	18	Eliminar Proyecto	2	Programadores	Sarisleidy Rodríguez Angel Felix Carabeo
	19	Mostrar Proyecto	2	Programadores	Sarisleidy Rodríguez Angel Felix Carabeo
	20	Insertar Módulo	2	Programadores	Sarisleidy Rodríguez Angel Felix Carabeo
	21	Modificar Módulo	2	Programadores	Sarisleidy Rodríguez Angel Felix Carabeo
	22	Eliminar Módulo	2	Programadores	Sarisleidy Rodríguez Angel Felix Carabeo
	23	Mostrar Módulo	2	Programadores	Sarisleidy Rodríguez Angel Felix Carabeo
	24	Insertar Usuario	3	Programadores	Sarisleidy Rodríguez Angel Felix Carabeo
	25	Modificar Usuario	3	Programadores	Sarisleidy Rodríguez Angel Felix Carabeo
	26	Eliminar Usuario	3	Programadores	Sarisleidy Rodríguez Angel Felix Carabeo
	27	Mostrar Usuario	3	Programadores	Sarisleidy Rodríguez Angel Felix Carabeo
	28	Mostrar Reporte del grado de implementación general del Centro	3	Programadores	Sarisleidy Rodríguez Angel Felix Carabeo
	29	Mostrar Reporte del grado de implementación por proyectos	3	Programadores	Sarisleidy Rodríguez Angel Felix Carabeo
	30	Mostrar Reporte del grado de	3	Programadores	Sarisleidy Rodríguez Angel Felix Carabeo

		implementación por características en el Centro			
	31	Mostrar Reporte del grado de implementación por características en los proyectos	3	Programadores	Sarisleidy Rodríguez Angel Felix Carabeo
	32	Mostrar Reporte de las preguntas evaluadas de NO	3	Programadores	Sarisleidy Rodríguez Angel Felix Carabeo
	33	Mostrar Reporte del grado de implementación general por módulo	3	Programadores	Sarisleidy Rodríguez Angel Felix Carabeo
	34	Mostrar Reporte del grado de implementación del módulo por características.	3	Programadores	Sarisleidy Rodríguez Angel Felix Carabeo
	35	Guardar Reporte del grado de implementación por proyectos	3	Programadores	Sarisleidy Rodríguez Angel Felix Carabeo
	36	Guardar Reporte del grado de implementación por características en el Centro	3	Programadores	Sarisleidy Rodríguez Angel Felix Carabeo
	37	Guardar Reporte del grado de implementación por características en los proyectos	3	Programadores	Sarisleidy Rodríguez Angel Felix Carabeo
	38	Guardar Reporte de las preguntas evaluadas de NO	3	Programadores	Sarisleidy Rodríguez Angel Felix Carabeo
	39	Guardar Reporte del grado de implementación del módulo por características.	3	Programadores	Sarisleidy Rodríguez Angel Felix Carabeo
Baja					
	40	Mostrar Guías de Aseguramiento de las	3	Programadores	Sarisleidy Rodríguez Angel Felix Carabeo

		Características de Calidad			
Requisitos No Funcionales					
Usabilidad	1	El sistema será usado por personal del Grupo de Calidad del Centro de Gobierno Electrónico y el personal de los proyectos que se encuentre involucrado en el proceso de Pruebas de Software.		Programadores	Sarisleidy Rodríguez Angel Felix Carabeo
	2	El funcionamiento del sistema no requerirá de grandes conocimientos para su uso.		Programadores	Sarisleidy Rodríguez Angel Felix Carabeo
	3	Deberá presentar botones organizados, de tal manera que permita al usuario una interacción cómoda con el mismo.		Diseñadores	Sarisleidy Rodríguez Angel Felix Carabeo
Confiabilidad	4	El sistema estará disponible 24 horas del día, los siete días de la semana.		Programadores	Sarisleidy Rodríguez Angel Felix Carabeo
	5	La herramienta de implementación a utilizar tiene soporte para recuperación ante fallos y errores.		Programadores	Sarisleidy Rodríguez Angel Felix Carabeo
Eficiencia	6	Tiempo de respuesta promedio de las peticiones que se realizan al servidor no deberá ser mayor de 3 segundos		Programadores	Sarisleidy Rodríguez Angel Felix Carabeo
Seguridad	7	El control de acceso se establecerá por roles que se le asignarán a los usuarios que interactúen con el sistema.		Programadores	Sarisleidy Rodríguez Angel Felix Carabeo
Interfaz	8	El sistema tiene que ofrecer una interfaz amigable, fácil de operar.		Diseñadores	Sarisleidy Rodríguez Angel Felix Carabeo
	9	Diseño sencillo, con pocas entradas, permitiendo que no sea necesario mucho entrenamiento para que los usuarios puedan utilizar el sistema		Diseñadores	Sarisleidy Rodríguez Angel Felix Carabeo

2.5. Diseño de metáforas.

Una metáfora es una historia compartida que describe cómo debería funcionar el sistema. Martin Fowler explica que la práctica de la metáfora consiste en formar un conjunto de nombres que actúen como vocabulario para hablar sobre el dominio del problema. Este conjunto de nombres ayuda al diseño y métodos del sistema [33]. Algunas de las características de las metáforas son:

- Da un contexto al equipo para entender los elementos básicos y sus relaciones.
- Proporciona integridad conceptual.

A partir de la definición de las funcionalidades descritas en la LRP es posible establecer las historias de usuarios, prototipos del sistema y las tareas ingenieriles que permiten su desarrollo siendo estas las actividades que se realizan y se describen a continuación.

2.5.1. Historia de Usuarios del Sistema

Básicamente una Historia de usuario es una lista priorizada de requisitos o funcionalidades, descritas usando la terminología del cliente. Estas se especifican en el Expediente de Proyecto en la Plantilla de Historias de Usuarios. A continuación se presenta una de ellas para dar continuidad en la explicación de las Tareas ingenieriles definidas.

Tabla 5: **Historia de Usuario Autenticar Usuario.**

Historia de Usuario	
Número: HU_1	**Nombre Historia de Usuario:** Autenticación de usuarios.
Modificación de Historia de Usuario Número: Ninguna	
Usuario: Sarisleidy Rodríguez Bravo	**Iteración Asignada:** Sprint 1
Prioridad en Negocio: Muy Alta	**Puntos Estimado:** 1
Riesgo de Desarrollo: Alto	**Puntos Reales:** 1
Descripción: El usuario accede al sistema insertando usuario y contraseña, posteriormente se procede a comprobar que estos datos son correctos, al serlo la aplicación le dará al usuario acceso a las funcionalidades a las cuales tiene permisos.	
Observaciones:	

Prototipo de interfaces:

Todas las historias de usuarios definidas se encuentran en el expediente de proyecto en la Plantilla de Historia de Usuarios.

2.5.2. Tareas Ingenieriles

Partiendo de las historias de usuarios se establecen un grupo de tareas ingenieriles para cada una de ellas, sirviendo de guía para el posterior desarrollo de la solución propuesta.

Algunas de las tareas ingenieriles están relacionadas con todas las historias de usuario preparando el ambiente de implementación, estas son:

Tabla 6: **Tarea Ingenieril 1.**

Tarea de Ingeniería	
Número Tarea:T_1	**Número Historia de Usuario:** Todas
Nombre Tarea: Generación de la base de datos.	
Tipo de Tarea: Desarrollo	**Puntos Estimados:1**
Fecha Inicio: 15-febrero-2014	**Fecha Fin:** 20-febrero-2014
Programador Responsable: Sarisleidy Rodríguez Bravo Angel Felix Carabeo Mondejar	
Descripción: Se genera la base de datos a partir del diseño realizado, esta debe implementarse sobre el sistema gestor definido para el desarrollo de la aplicación.	

Tabla 7: **Tarea Ingenieril 2.**

Tarea de Ingeniería	
Número Tarea:T_2	**Número Historia de Usuario:** Todas
Nombre Tarea: Montaje del ambiente de desarrollo con la integración de los marcos de trabajo seleccionados para el desarrollo	
Tipo de Tarea: Desarrollo	**Puntos Estimados:1**
Fecha Inicio: 15-febrero-2014	**Fecha Fin:** 5-marzo-2014
Programador Responsable: Sarisleidy Rodríguez Bravo Angel Felix Carabeo Mondejar	
Descripción: Instalar y configurar los marcos de trabajo definidos para el desarrollo de manera que se pueda comenzar la implementación de la aplicación.	

Otras están asociadas a cada historia de usuario, en particular, para poder dar cumplimiento a la implementación de las funcionalidades definidas en ellas, por ejemplo las relacionadas con la HU_1 son:

Tabla 8: **Tarea de Ingeniería 4.**

Tarea de Ingeniería	
Número Tarea: T_4	**Número Historia de Usuario:** HU_1
Nombre Tarea: Diseñar las interfaces requeridas para la funcionalidad autenticación de usuario.	
Tipo de Tarea: Desarrollo	**Puntos Estimados: 1**
Fecha Inicio: 1-abril-2014	**Fecha Fin:** 3-abril-2014
Programador Responsable: Sarisleidy Rodríguez Bravo Angel Felix Carabeo Mondejar	
Descripción: Crear las interfaces necesarias, teniendo en cuenta los prototipos definidos en la Historia de Usuario.	

Tabla 9: **Tarea de Ingeniería 5.**

Tarea de Ingeniería	
Número Tarea: T_5	**Número Historia de Usuario:** HU_1
Nombre Tarea: Implementar funcionalidad de autenticación de usuarios.	
Tipo de Tarea: Desarrollo	**Puntos Estimados: 1**
Fecha Inicio: 3-abril-2014	**Fecha Fin:** 8-abril-2014
Programador Responsable: Sarisleidy Rodríguez Bravo Angel Felix Carabeo Mondejar	
Descripción: Crear las clases y algoritmos necesarios para la implementación de la funcionalidad de autenticación de usuario.	

Todas las tareas ingenieriles definidas se encuentran en el expediente de proyecto en la Plantilla de Tareas de Ingeniería.

2.6. Arquitectura del sistema

La Arquitectura de Software es, a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se la percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema. La vista arquitectónica es una vista abstracta, aportando el más alto nivel de comprensión y la supresión o diferimiento del detalle inherente a la mayor parte de las abstracciones [34].

Un patrón arquitectónico es una descripción de un problema particular y recurrente de diseño, que aparece en un contexto específico y presenta una solución demostrada. Los patrones arquitectónicos expresan el esquema de organización estructural fundamental para sistemas de software [35].

Partiendo de la idea anterior, la arquitectura está basada en el patrón arquitectónico Modelo-Vista-Controlador (MVC), el cual separa la lógica del negocio de la interfaz de usuario, facilita la evolución por separado de ambos aspectos e incrementa la reutilización y flexibilidad. Divide una aplicación interactiva en tres partes: el modelo contiene los datos y la funcionalidad esencial, las vistas despliegan la información al usuario y los controladores manejan las entradas. Es una especialización de un modelo de capas, con la diferencia que se usa para entornos web como patrón por excelencia.

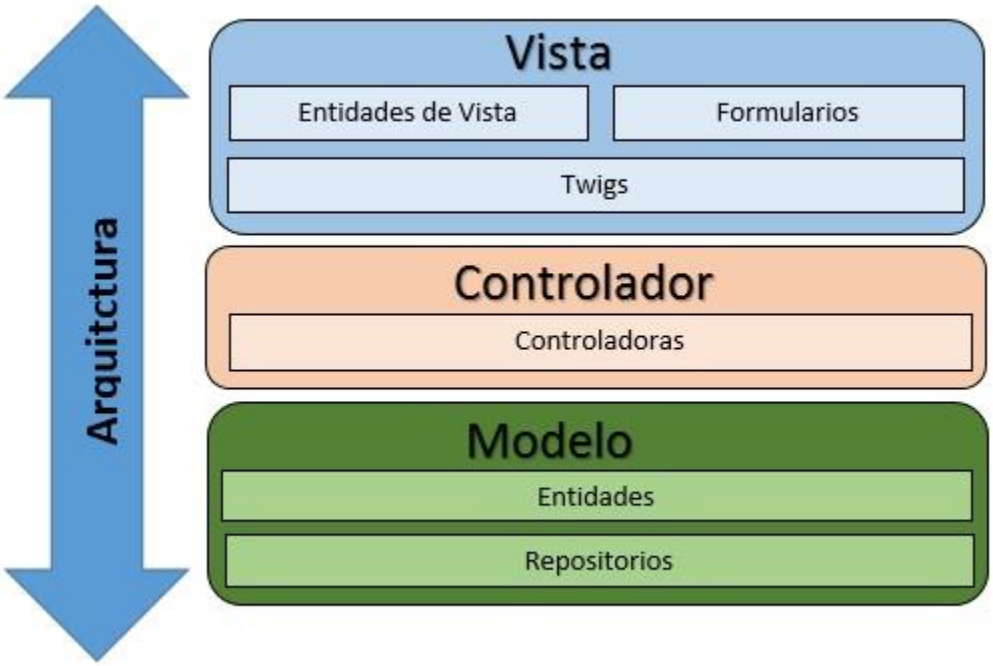


Figura 7: **Representación en capas del patrón MVC**

- **Vista:** está representada por la capa vista, esta es la capa superior que encapsula las interfaces de usuario, entidades de presentación, plantillas twig y formularios necesarios para la interacción con el cliente.
- **Controlador:** está representada por la capa controlador, esta se encarga de recibir una petición, procesar la información, hacer un pedido al modelo y devolver una respuesta a los controladores los cuales a su vez la envían a la vista. La capa controlador contiene las clases *Controller* (Controladoras) que se encarga de dar respuesta a las peticiones realizadas por el usuario.
- **Modelo:** la capa modelo es la capa inferior y contiene las clases *Entity* (Entidad) y las clase Repository (Repositorio), quienes se encargan del manejo de los datos para visualizarlos.

2.7. Modelo de Diseño

En la plantilla modelo de diseño se define un esbozo inicial del diseño del sistema sin entrar en especificaciones ni detalles solo lo que el diseñador necesita para hacer un primer entregable del sistema.

2.7.1. Diagrama de Paquetes

En la figura siguiente se muestra el diagrama de paquetes de la solución propuesta. El mismo está compuesto por 4 paquetes: Form, Controller y Repository los cuales representan la arquitectura propuesta para el desarrollo del sistema (Patrón arquitectónico Modelo-Vista-Controlador) y el paquete Entity, el cual

encapsula las clases entidades del sistema. Además se muestran las relaciones existentes entre las clases y los paquetes.

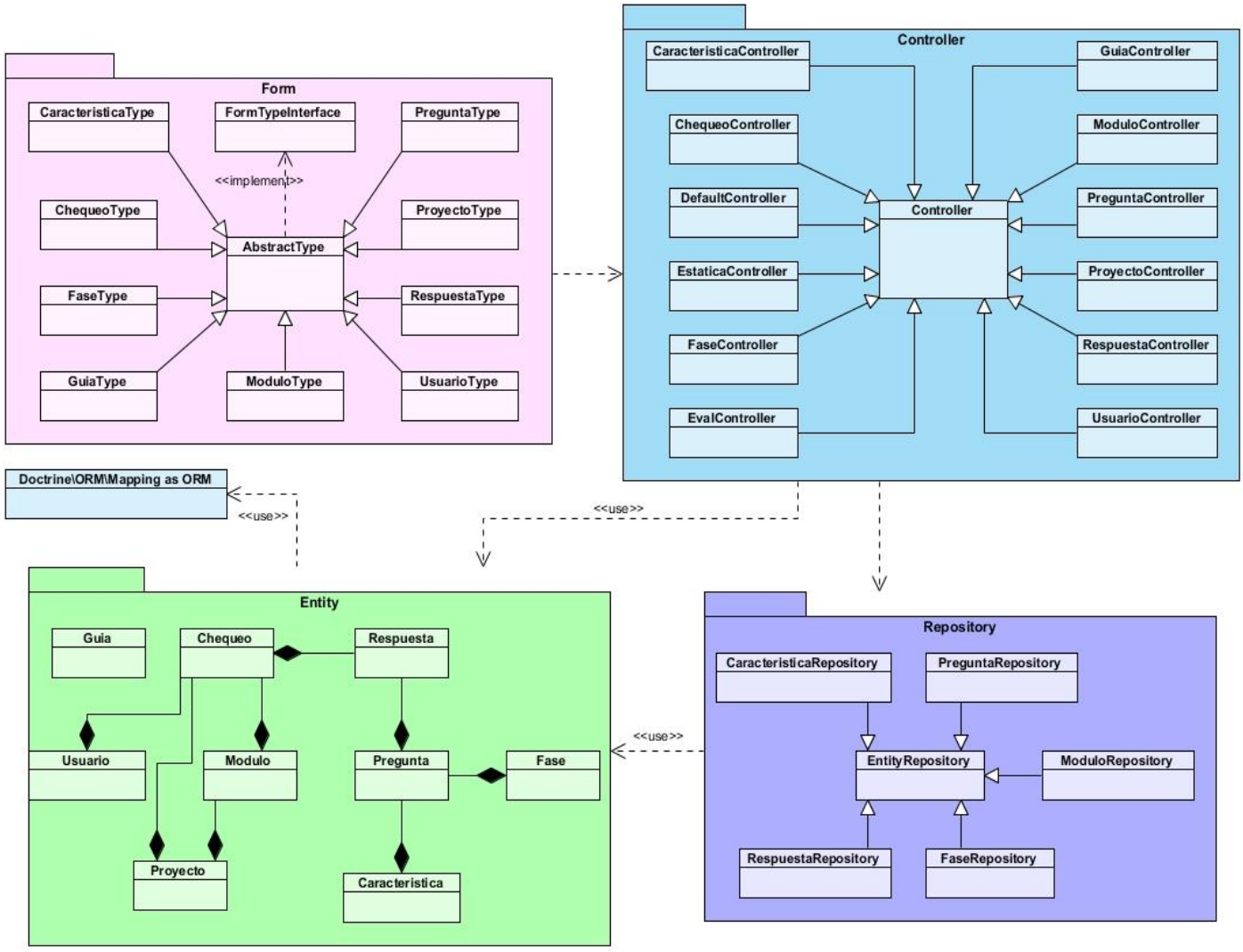


Figura 8: **Diagrama de Paquetes.**

Paquete Form: está compuesto por las clases Form que integran la aplicación, estas son las encargadas de crear los formularios (los campos con lo que van a contar los formularios) para introducirle los datos que serán procesados.

Paquete Controller: está compuesto por las clases Controller que integran la aplicación, estas son las clases controladoras donde se va a dar respuesta a todas las peticiones del usuario.

Paquete Entity: está compuesto por las clases Entity que integran la aplicación, estas son las que contienen toda la información de las clases y son las que representan las tablas de la Base de Datos.

Paquete Repository: está compuesto por las clases Repository que integran la aplicación, estas son las encargadas de obtener los datos de la Base de Datos y mostrarlos a través de vistas al usuario.

2.7.2. Patrones de diseño

Para la definición de las clases del sistema, el diseño del sistema a implementar, es importante la revisión de algunos patrones que permiten realizar un diseño adecuado y consistente. La asignación de responsabilidades es la habilidad más importante en el análisis y diseño orientado por objetos, para ello tiene suma importancia la utilización de los patrones GRASP.

En términos generales, un patrón es un conjunto de información que proporciona respuesta a un conjunto de problemas similares, es decir, un patrón es una solución a un problema en un contexto [36].

2.7.2.1. Patrones GRASP[4] (Patrones de Software para la Asignación General de Responsabilidad)

*"Los patrones **GRASP** describen los principios fundamentales de diseño de objetos para la asignación de responsabilidades. Constituyen un apoyo para la enseñanza que ayuda a entender el diseño de objeto esencial y aplica el razonamiento para el diseño de una forma sistemática, racional y explicable".* Estos patrones se describen a continuación [37]:

Bajo Acoplamiento: este patrón es el encargado de conservar el bajo acoplamiento. Se pone en práctica, puesto que las clases poseen pocas relaciones entre sí, para que en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión en las demás. Esta característica permitió potenciar la reutilización y disminuyó la dependencia entre las clases. El patrón bajo acoplamiento responde en el trabajo presentado a los principios de diseño que influyeron en la decisión de asignar responsabilidades a cada una de las clases.

[4] GRASP es un acrónimo que significa General Responsibility Asignment Software Patterns (patrones generales de software para asignar responsabilidades.

Alta cohesión: se evidencia en cada clase del diseño propuesto, que realiza una labor única dentro del sistema y colabore con las otras para llevar a cabo una tarea, como son las clases que forman parte de las capas controladora y modelo.

Experto: se pone en práctica en las clases de la entidad, que van a contener toda la información necesaria para llevar a cabo las responsabilidades que le son asignadas. Se ve reflejado en las clases *Chequeo, Respuesta, Proyecto*, entre otras.

Controlador: asigna la responsabilidad de controlar el flujo de eventos del sistema, a una clase específica, en este caso el controlador frontal (*app.php*), que es el punto de entrada único de toda la aplicación en un entorno determinado. Cuando el controlador frontal recibe una petición, utiliza el sistema de enrutamiento para asociar el nombre de una acción y el nombre de un módulo con la URL entrada por el usuario.

Creador: el patrón creador ayuda a identificar quién debe ser el responsable de la creación (o instanciación) de nuevos objetos o clases. Este patrón guía la asignación de responsabilidades relacionadas con la creación de objetos. Se ve reflejado en las clases controladoras, un ejemplo es la clase *CaracteristicaController* que es responsable de la creación de todas las instancias de la clase Caracteristica necesarias para el correcto funcionamiento de la lógica del sistema.

2.7.2.2. Patrones GoF[5] (*The Gang of Four*)

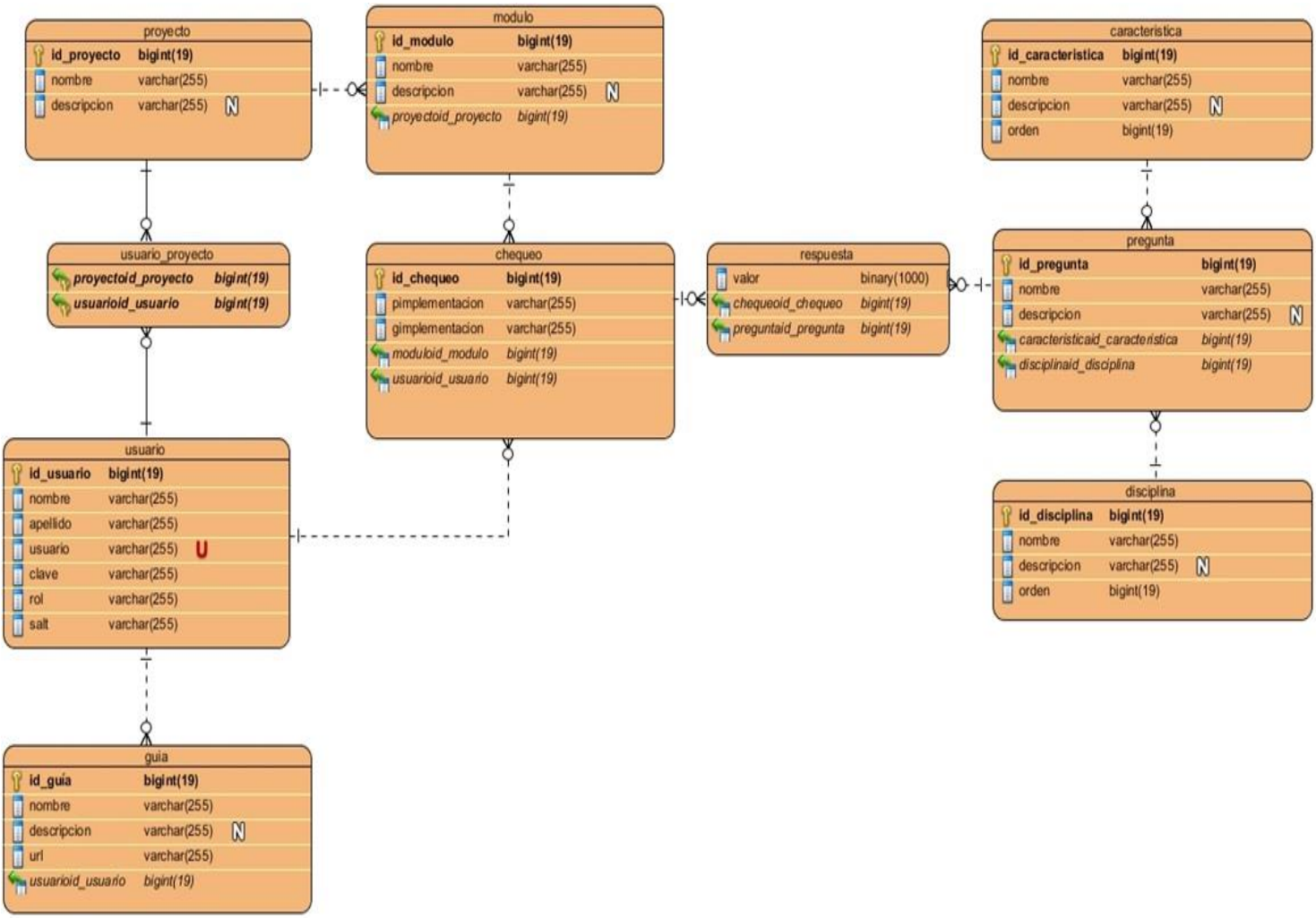
Los patrones de diseño **GoF** se clasifican en 3 grandes categorías basadas en su propósito: creacionales, estructurales y de comportamiento.

Decorador: aplicado a la generación de vistas, la solución que ofrece dicho patrón es la de añadir funcionalidad adicional a las plantillas. Por ejemplo, añadir el menú y el pie de página a las plantillas que lo requieran, se trata de decorar las plantillas con elementos adicionales reutilizables. El sistema de plantillas twig, está provisto de un mecanismo de herencia, el cual se observa en el archivo denominado *layoutAdm.html.twig* que contiene el *Layout* (plantilla global) de todas las páginas del registro. El mismo almacena el código HTML que es común a todas las páginas del registro, para no repetirlo en cada página, por lo que el *Layout* decora la plantilla.

[5] Gang of Four, en español Banda de los Cuatro, formada por Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides.

2.8. Modelo de datos

Es un conjunto de conceptos que nos permiten describir los datos, las relaciones entre ellos, la semántica



y las restricciones de consistencia [38].

Figura 9: **Modelo de Datos.**

El modelo de datos se encuentra en 3era Forma Normal (FN). Se describe el modelo de datos en el Anexo 3, cada una de las tablas y sus atributos. A continuación se presentan las descripciones de las tablas de mayor importancia para la investigación por los datos que persisten en ellas.

Tabla 10: **Tabla chequeo.**

Nombre de la Tabla: chequeo		
Descripción: Esta es la tabla donde se va almacenar toda la información de los chequeos.		
Atributos	**Tipo**	**Descripción**
id_chequeo	bigint	Es el identificador del chequeo. Es la llave primaria de la tabla.
pimplementacion	varchar	Es el por ciento de implementación.
gimplementacion	varchar	Es el grado de implementación.
id_modulo	bigint	Es la llave foránea entre la relación existente entre la tabla módulo y la tabla chequeo.
id_usuario	bigint	Es la llave foránea entre la relación existente entre la tabla usuario y la tabla chequeo.

Tabla 11: **Tabla respuesta.**

Nombre de la Tabla: respuesta		
Descripción: Esta es la tabla donde se va almacenar toda la información de las respuestas.		
Atributos	**Tipo**	**Descripción**
valor	binary	Es el valor que va a tener asociado la respuesta.
id_chequeo	bigint	Es la llave foránea entre la relación existente entre la tabla chequeo y la tabla respuesta.
id_pregunta	bigint	Es la llave foránea entre la relación existente entre la tabla pregunta y la tabla respuesta.

2.9. Conclusiones parciales

Al finalizar el presente capítulo se puede arribar a las siguientes conclusiones:

- Se realizaron los artefactos Modelo de Historia de Usuario del Negocio, Plantilla de Historia de Usuario, Plantilla de Lista de Reserva del Producto y la Plantilla de Tareas de ingeniería; preparando las bases para las fases restantes del proceso.
- Se realizó el Modelo de Datos y el Diagrama de Clases, representando las entidades relevantes del sistema y las clases involucradas en el desarrollo del mismo así como sus relaciones.

CAPÍTULO 3: IMPLEMENTACIÓN Y VALIDACIÓN

En el presente capítulo se describe cómo fue implementada la aplicación en términos de componentes. Se detalla mediante el diagrama de despliegue como quedará distribuida la aplicación. Se realizan pruebas de calidad para validar la solución implementada. Al final de este capítulo se realizará un análisis de la solución obtenida.

3.1. Implementación

3.1.1. Plan de Releases

El propósito de una iteración es transformar un conjunto de la LRP en un incremento en la funcionalidad del producto que sea potencialmente entregable a los usuarios. Para ello, previamente a cada iteración, se realiza una reunión de planificación de iteración para determinar en qué funcionalidad del producto trabajará el equipo. Las iteraciones a realizar, sus características y el orden de las historias de usuarios con su planificación estimada para ser implementadas se recogen en la plantilla Plan de Releases que se encuentra en el Anexo 4 y en el Expediente de proyecto.

3.1.2. Estándares de Codificación empleados

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

General

- Se exceptúan el uso de las tildes y la letra ñ, la que será sustituida por nn.
- En todo momento se utilizarán nombres que sean claros, concretos y libres de ambigüedades. Ejemplo: "idProyecto" y no solamente "id"
- El nombre de todas las variables y métodos comenzarán con letra minúscula y si este está compuesto por varias palabras, todas las palabras internas que lo componen comienzan con mayúscula. Ejemplo: “*actualizarCaracteristica()*”.
- El nombre de las clases siempre comienzan con mayúscula, en caso de nombre compuesto las palabras se ponen juntas y comienzan con mayúscula.

Identación

El contenido siempre se identará con tabs, nunca utilizando espacios en blanco.

Clases

Las clases controladoras tienen el nombre de la clase que representa y seguidamente Controller, las clases repositorios tienen el nombre de la clase que representa y seguidamente Repository y las clases formularios tienen el nombre de la clase que representa y seguidamente Type.

Nombre de variables

- No se utilizarán nombres de variables que puedan ser ambiguos.
- Las variables booleanas deben tener nombres que sugieran respuestas o contenidos de tipo Sí/No, por ejemplo: "*esValido*".

Estructuras de control

Se incluye if, for, foreach, while, switch, entre las estructuras de control y los paréntesis debe de existir un espacio. Se recomienda utilizar siempre llaves de apertura y cierre, incluso en situaciones en las que técnicamente son opcionales. Esto aumenta la legibilidad y disminuye la probabilidad de errores lógicos.

Buenas prácticas

Los valores booleanos y nulos siempre se escriben con mayúscula, para facilitar la legibilidad del código usar un enter antes de las estructuras de control y definición de las funciones.

3.1.3. Diagrama de Componentes

Un diagrama de componentes muestra las dependencias lógicas entre componentes software, sean estos últimos fuentes, binarios o ejecutables, los cuales ilustran las piezas del software. Los diagramas de Componentes prevalecen en el campo de la arquitectura de software pero pueden ser usados para modelar y documentar cualquier arquitectura del sistema, es decir para describir la vista de implementación estática de un sistema. Los diagramas de componentes se relacionan con los diagramas de clases, ya que un componente normalmente se corresponde con una o más clases, interfaces o colaboraciones, pero un diagrama de Componentes tiene un nivel más alto de abstracción que un diagrama de clase. Usualmente un componente se implementa por una o varias clases (u objetos) en tiempo de ejecución. Estos son bloques de construcción, como eventualmente un componente puede comprender una gran porción de un sistema [39].

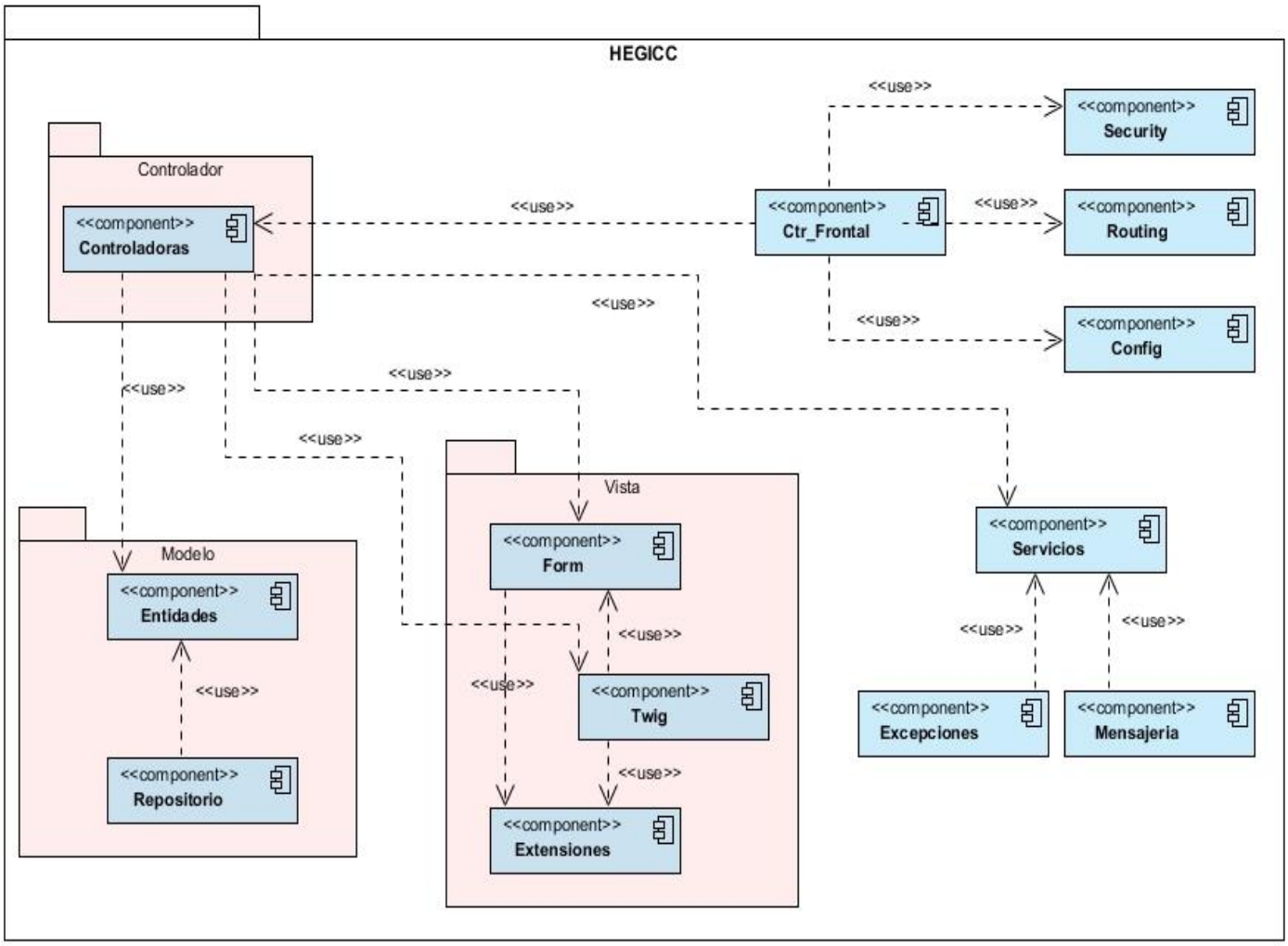


Figura 10: **Diagrama de Componentes.**

El diagrama de componentes de la herramienta HEGICC incluye los componentes Ctr_Frontal, es el controlador frontal que recibe cada petición realizada al sistema, Config (para las configuraciones), Routing (para las rutas de acceso), Security (para la seguridad), Mensajería (para la gestión de mensajes informativos y de errores) y Excepciones (para el manejo de las excepciones generadas). El sistema está compuesto por tres paquetes fundamentales, Controlador (encargado de dar respuesta a las peticiones de los usuarios), Modelo (contiene las clases entidades de la herramienta y lo relacionado con la lógica de negocio), Vista (encargado de la construcción y manejo de las interfaces de usuario).

El paquete Controlador con el componente Controladora, para dar respuesta a las peticiones realizadas por los usuarios. El componente Controladoras se relaciona con el componente Entidades, que a su vez es usado por el componente Repositorio la cual contiene las clases repositorios encargadas de realizar las consultas a la base de datos. El paquete Vista incluye los componentes Form (encargado del trabajo con los formularios), Twig (para las interfaces de usuarios y plantillas del módulo) y Extensiones (posee los JavaScript y CSS).

3.1.4. Diagrama de Despliegue

Para comprender como se ejecutará a nivel de hardware un sistema desarrollado y tener una visión clara de la estructura del sistema en ejecución y las relaciones entre los componentes que interactúan en el mismo, se realiza el diagrama de despliegue. Dicho diagrama tiene como objetivo reflejar lo mencionado anteriormente, representando la disposición de las instancias de los componentes de ejecución, en instancias de nodos conectados por enlaces de comunicación.

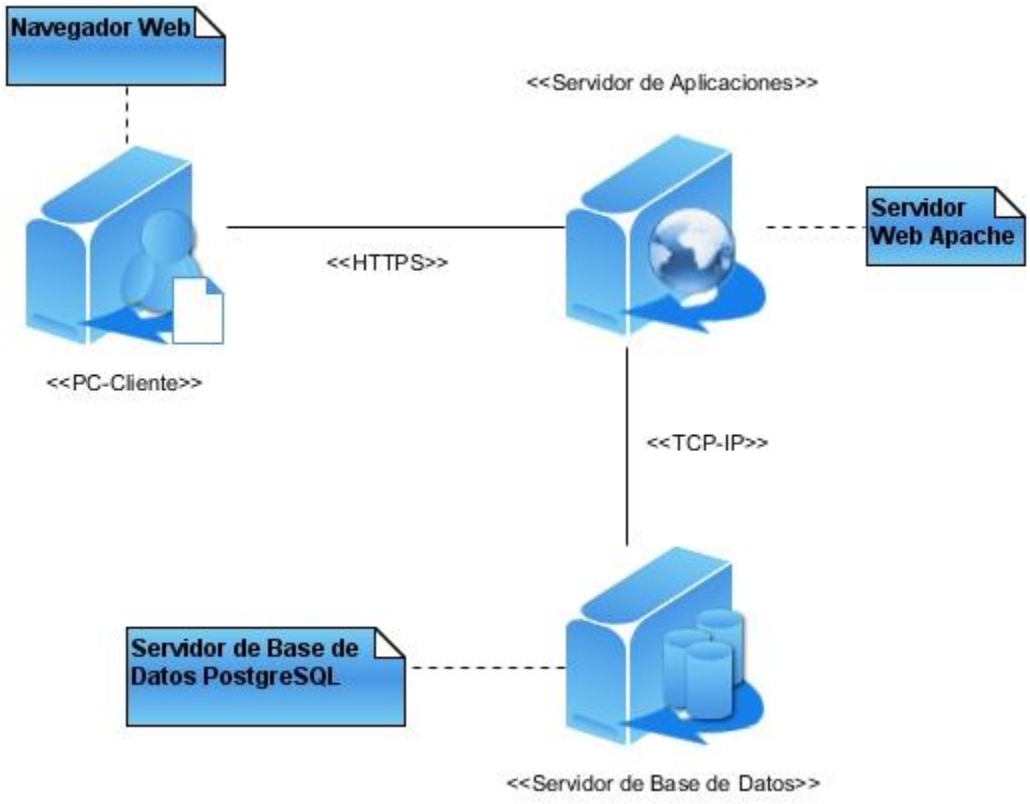


Figura 11: **Diagrama de Despliegue.**

Cada uno de los componentes que integran el diagrama de despliegue correspondiente a la aplicación tiene su relevancia para su correcto funcionamiento, definiéndolos de la siguiente manera:

- **PC_Cliente:** se encontrará el sistema operativo Linux o Windows y el navegador web Mozilla Firefox mediante el cual los clientes tendrán acceso al sistema y harán uso del mismo.
- **Servidor de Base de Datos:** se encontrará la base de datos que tendrá la información del sistema.
- **Servidor de aplicaciones:** se encontrará todo lo referente a la aplicación, donde estarán agrupados los archivos a través de los cuales el usuario logra acceder al sistema.

3.2. Aplicación de técnicas de validación de requisitos

La validación de los requisitos tiene como objetivo comprobar que estos son correctos. Esta fase debe realizarse o de lo contrario se corre el riesgo de implementar una mala especificación, con el costo que eso conlleva. Es muy importante asegurar la validez de los requisitos antes de comenzar el desarrollo del software. Para ello debe hacerse una comprobación de la correspondencia entre las descripciones iniciales y la definición de los requisitos realizada, para verificar que responden a lo que desea el usuario final. Para llevar a cabo este proceso, se aplicaron las siguientes técnicas de validación de requisitos:

- **Revisiones de requisitos:** los requisitos se analizan sistemáticamente por un revisor técnico.

Al poner en práctica estas revisiones, se detectaron en una primera iteración cinco problemas relacionados con una mala expresión de los requisitos y falta de información a la hora de definirlos. Al realizar la segunda iteración se comprobó que se habían resuelto los errores antes encontrados y que la definición efectuada era correcta.

- **Construcción de prototipos:** el prototipado de interfaz de usuario es una técnica de representación aproximada de la interfaz de usuario de un software que permite a clientes y usuarios entender fácilmente la propuesta de los ingenieros de requisitos para resolver sus problemas de negocio.

Esta técnica se aplicó diseñando prototipos de interfaz de usuario para que el cliente entendiera de qué forma iba a quedar el sistema que se comenzaba a desarrollar. Para ello se realizaron reuniones para mostrar los prototipos, en una primera reunión, el cliente no estuvo de acuerdo con tres de los prototipos realizados, luego se modificaron dichos prototipos y en la segunda reunión efectuada se consiguió la

aprobación del cliente. Además en el Anexo 5 se encuentra la carta de Aceptación del Cliente de los requisitos del software.

3.3. Resultados de la aplicación de las métricas para la validación

Las Métricas de diseño permiten medir de forma cuantitativa la calidad de los atributos internos del software. Esto permite al ingeniero evaluar la calidad durante el desarrollo del sistema [40].

La solución informática incluirá estas pruebas haciendo uso de las siguientes métricas:

Tamaño Operacional de la Clase (TOC): Está dada por el número de funcionalidades asignadas a cada una de las clases del sistema informático propuesto en la investigación evaluando los siguientes atributos de calidad [41]:

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

Relaciones entre Clases (RC): Está dado por el número de relaciones de uso de una clase con otra y evalúa los siguientes atributos de calidad [41]:

- **Acoplamiento:** Un aumento del RC implica un aumento del Acoplamiento de la clase.
- **Complejidad de mantenimiento:** Un aumento del RC implica un aumento de la complejidad del mantenimiento de la clase.
- **Reutilización:** Un aumento del RC implica una disminución en el grado de reutilización de la clase.
- **Cantidad de pruebas:** Un aumento del RC implica un aumento de la Cantidad de pruebas de unidad necesarias para probar una clase.

3.3.1. Resultados Métrica TOC

Tabla 12: **Rango de valores para la evaluación técnica de los atributos de calidad relacionados con la métrica TOC**

	Categoría	Criterio
Responsabilidad	Baja	< =Prom.
	Media	Entre Prom. y 2* Pom.

	Alta	> 2* Prom.
Complejidad implementación	Baja	< =Prom.
	Media	Entre Prom. y 2* Pom.
	Alta	> 2* Prom.
Reutilización	Baja	> 2*Prom.
	Media	Entre Prom. y 2* Pom.
	Alta	<= Prom.

A continuación se muestran las clases del sistema evaluadas en los atributos de calidad Responsabilidad, Complejidad de implementación y Reutilización.

Tabla 13: **Evaluación de las clases del sistema mediante la métrica TOC**

Clases	Cantidad de Procedimientos	Responsabilidad	Complejidad de Implementación	Reutilización
Caracteristica	9	Baja	Baja	Alta
Chequeo	15	Media	Media	Media
Fase	9	Baja	Baja	Alta
Guia	13	Media	Media	Media
Modulo	9	Baja	Baja	Alta
Pregunta	11	Media	Media	Media
Proyecto	7	Baja	Baja	Alta
Respuesta	7	Baja	Baja	Alta
Usuario	22	Alta	Alta	Baja
UsuarioProyecto	6	Baja	Baja	Alta
CaracteristicaController	11	Media	Media	Media
ChequeoController	7	Baja	Baja	Alta
DefaultController	3	Baja	Baja	Alta
EstaticaController	4	Baja	Baja	Alta
EvalController	25	Alta	Alta	Baja
FaseController	11	Media	Media	Media
GuiaController	11	Media	Media	Media
ModuloController	11	Media	Media	Media
PreguntaController	11	Media	Media	Media
ProyectoController	11	Media	Media	Media
RespuestaController	10	Baja	Baja	Alta
UsuarioController	11	Media	Media	Media

En la Figura 12 se muestra una gráfica con agrupaciones por intervalos de la cantidad de clases según la cantidad de procedimientos:

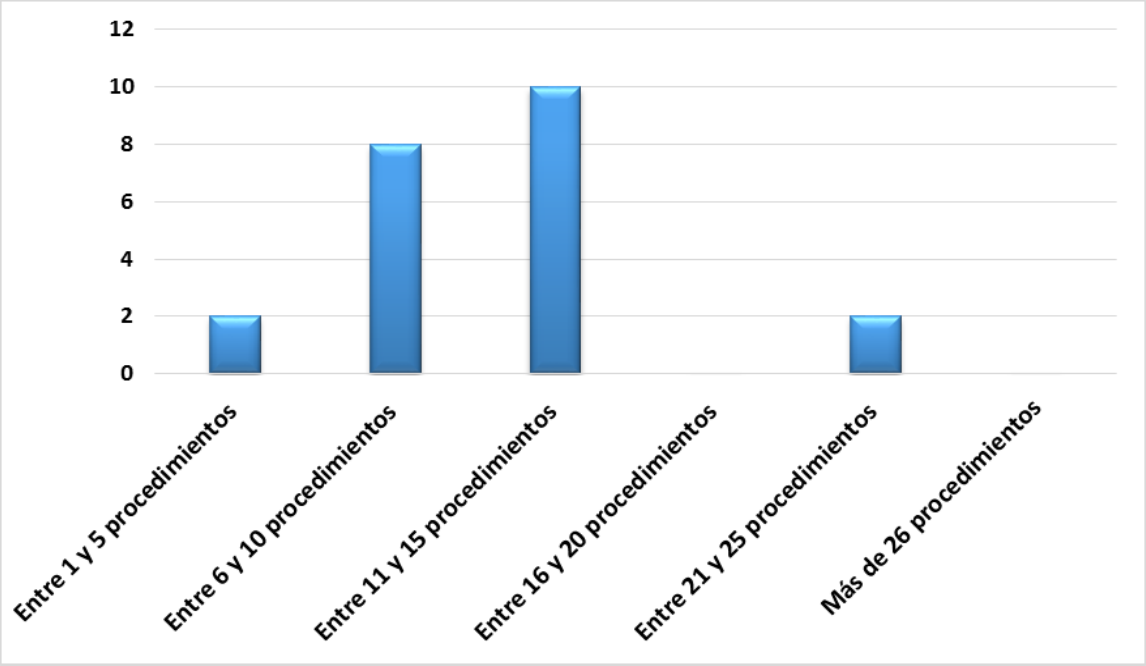


Figura 12: **Representación de la cantidad de clases agrupadas en intervalos según la cantidad de procedimientos.**

Las gráficas que corresponden a los resultados obtenidos se presentan en las siguientes figuras.

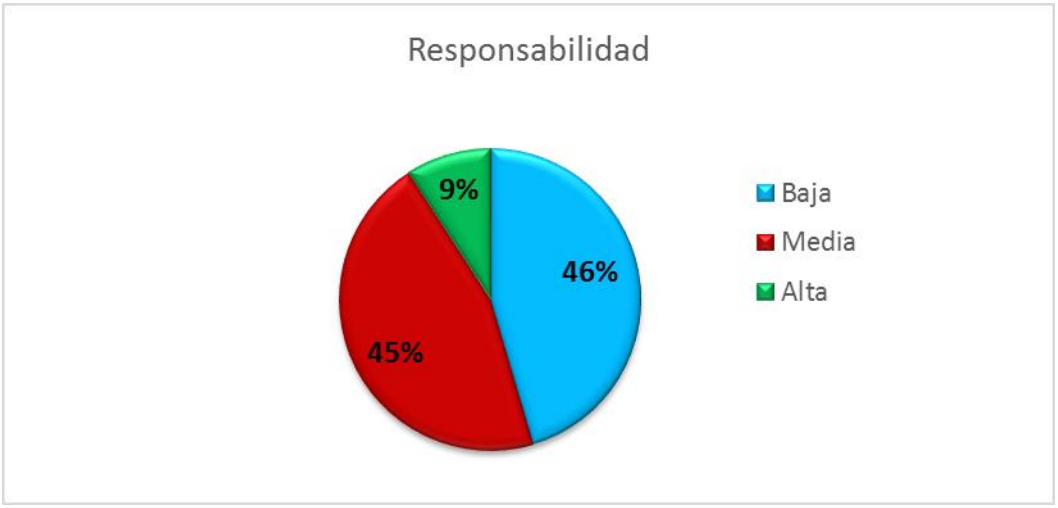


Figura 13: **Representación en por ciento (%) de los resultados obtenidos en el atributo Responsabilidad.**

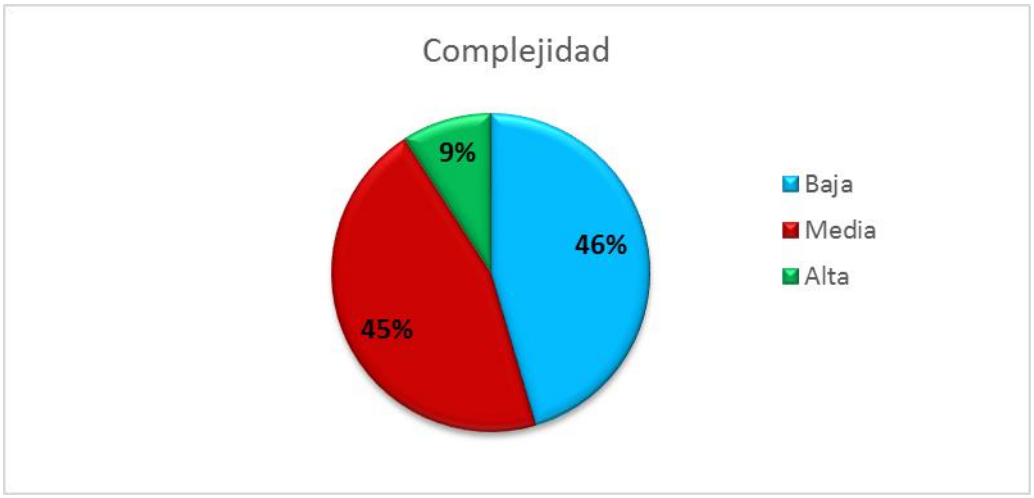


Figura 14: **Representación en por ciento (%) de los resultados obtenidos en el atributo Complejidad de implementación.**

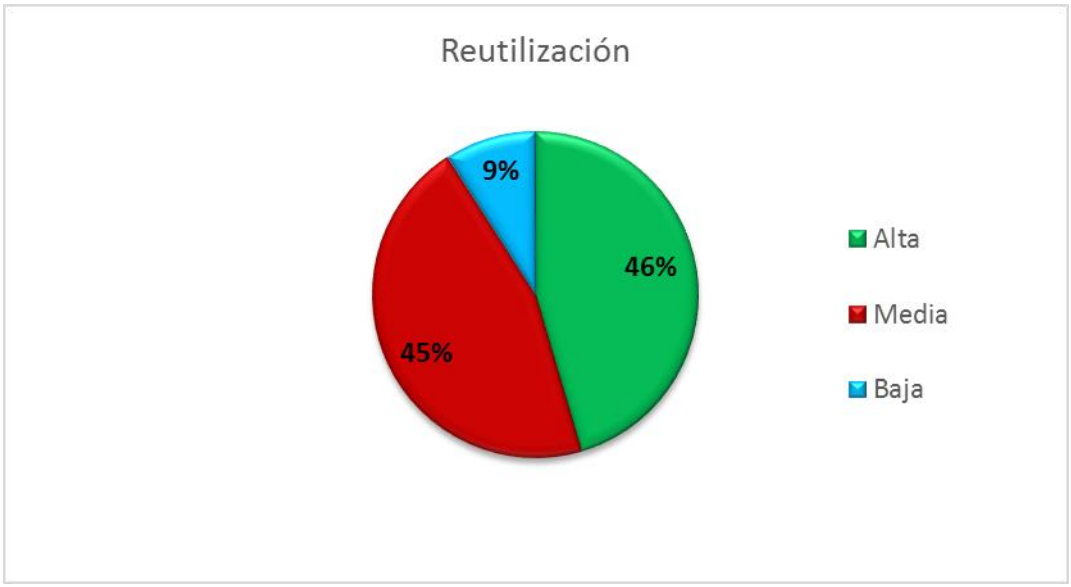


Figura 15: **Representación en por ciento (%) de los resultados obtenidos en el atributo Reutilización.**

Como resultado de la aplicación de la métrica TOC se evidencia que las clases del sistema poseen baja responsabilidad, baja complejidad y alta reutilización de implementación por lo que el diseño de las clases en cuanto a cantidad de funcionalidades por cada una es bueno.

3.3.2. Resultados Métrica RC

Tabla 14: **Rangos de valores para la evaluación técnica de los atributos de calidad relacionados con la métrica RC.**

	Categoría	Criterio
Acoplamiento	Ninguno	0
	Bajo	1
	Medio	2
	Alto	>2
Complejidad Mant.	Baja	<= Prom.
	Media	Entre Prom. y 2*Prom.
	Alta	> 2*Prom.
Reutilización	Baja	>2* Prom.
	Media	Entre Prom. y 2*Prom.
	Alta	<= Prom.
Cantidad de Pruebas	Baja	<= Prom.
	Media	Entre Prom. y 2*Prom.
	Alta	> 2*Prom.

A continuación se muestran las clases del sistema evaluadas en los atributos de calidad Acoplamiento, Complejidad del mantenimiento, Cantidad de pruebas y Reutilización.

Tabla 15: **Clases del Sistema evaluadas en los atributos de calidad según la métrica RC.**

Clase	Cantidad de Relaciones de Uso	Acoplamiento	Complejidad Mant.	Reutilización	Cantidad de Pruebas
Caracteristica	0	Ninguno	Baja	Alta	Baja
Pregunta	2	Medio	Media	Media	Media
Fase	0	Ninguno	Baja	Alta	Baja
Respuesta	2	Medio	Media	Media	Media
Chequeo	3	Alto	Alta	Baja	Alta
Proyecto	0	Ninguno	Baja	Alta	Baja
Modulo	1	Bajo	Baja	Alta	Baja
Usuario	0	Ninguno	Baja	Alta	Baja
UsuarioProyecto	2	Medio	Media	Media	Media
Guia	0	Ninguno	Baja	Alta	Baja
CaracteristicaController	1	Bajo	Baja	Alta	Baja
ChequeoController	6	Alto	Alta	Baja	Alta
DefaultController	1	Bajo	Baja	Alta	Baja
EstaticaController	5	Alto	Alta	Baja	Alta
EvalController	8	Alto	Alta	Baja	Alta
FaseController	1	Bajo	Baja	Alta	Baja
GuiaController	1	Bajo	Baja	Alta	Baja

ModuloController	3	Alto	Media	Media	Media
PreguntaController	1	Bajo	Baja	Alta	Baja
ProyectoController	2	Medio	Media	Media	Media
RespuestaController	1	Bajo	Baja	Alta	Baja
UsuarioController	1	Bajo	Baja	Alta	Baja

A continuación se muestra una gráfica con agrupaciones por intervalos de la cantidad de clases según las dependencias entre ellas.

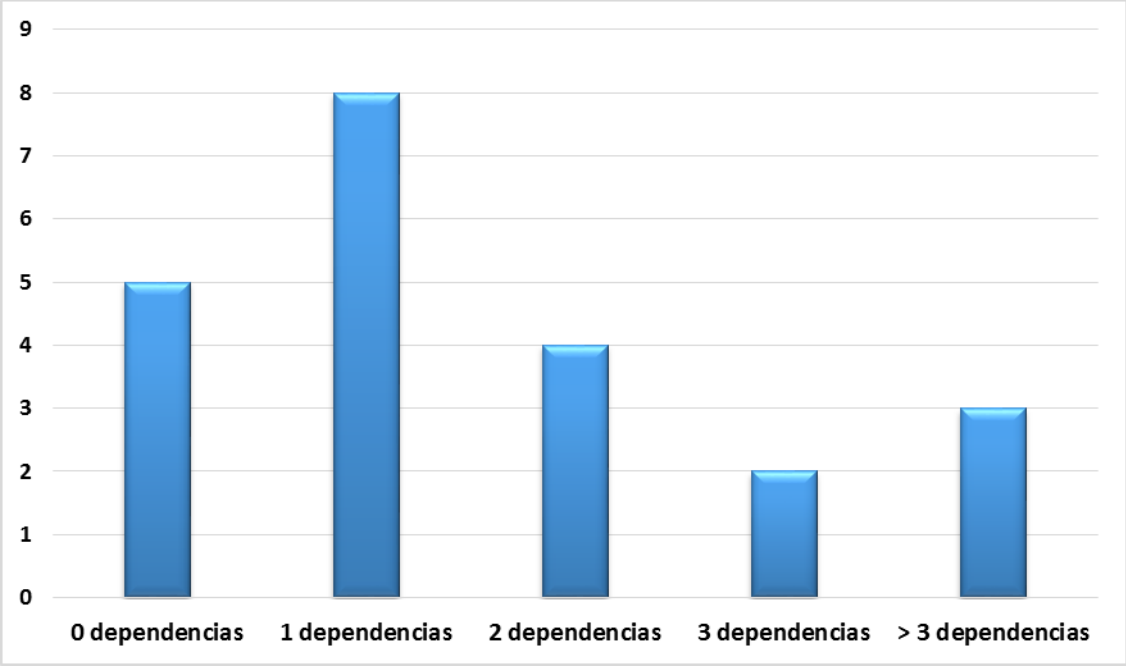


Figura 16: **Intervalos de las clases agrupadas según las dependencias entre ellas.**

La Figura 17 muestra los valores en % obtenidos al evaluar el diseño en el atributo Acoplamiento.

Figura 17: **Representación en porcientos (%) de los atributos obtenidos en el atributo Acoplamiento.**

La Figura 18 muestra los valores en % obtenidos al evaluar el diseño en el atributo Mantenimiento.

Figura 18: **Representación en porcientos (%) de los atributos obtenidos en el atributo Complejidad de Mantenimiento.**

La Figura 19 muestra los valores en % obtenidos al evaluar el diseño en el atributo Cantidad de Pruebas.

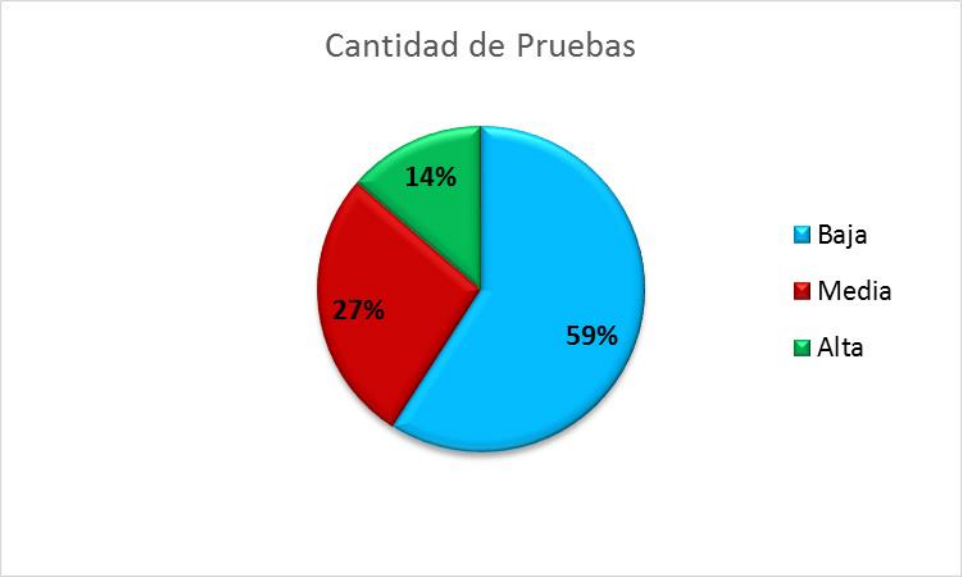


Figura 19: **Representación en porcientos (%) de los atributos obtenidos en el atributo Cantidad de Pruebas.**

La Figura 20 muestra los valores en % obtenidos al evaluar el diseño en el atributo Reutilización.

Figura 20: **Representación en porcientos (%) de los atributos obtenidos en el atributo Reutilización.**

Como resultado de aplicar la métrica RC se obtuvo bajo acoplamiento, baja complejidad de mantenimiento, baja cantidad de pruebas y alta reutilización.

3.4. Verificación del sistema

En la etapa de verificación se comprueba que el software cumple los requisitos funcionales y no funcionales de su especificación.

Pruebas unitarias

Las pruebas unitarias son aplicadas para verificar que el software cumple los requisitos funcionales y no funcionales de su especificación y también son empleadas para asegurar la calidad del código entregado. Además, son la mejor forma de detectar fallas tempranamente en el desarrollo y está demostrado que mientras más pronto se encuentren los errores, menos costará corregirlos. Para llevar a cabo las pruebas unitarias se decidió aplicar los métodos de pruebas de caja blanca y de caja negra.

3.4.1. Pruebas de Caja blanca

Las pruebas de caja blanca se basan en el conocimiento de la lógica interna del código del sistema, contemplan los distintos caminos que se pueden generar teniendo en cuenta las estructuras condicionales o los distintos estados del mismo.

Para poner en práctica este tipo de pruebas se procede a utilizar la técnica del camino básico que permite al diseñador de casos de prueba obtener una medida de la complejidad lógica de una funcionalidad y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución. Para la obtención de la complejidad ciclomática se emplea una representación del flujo de control en forma de grafo.

La complejidad ciclomática es una medida que aporta una valoración cuantitativa de la complejidad lógica de un programa. Dentro del contexto de la técnica del camino básico, representa el número de caminos independientes de un programa.

A continuación se aplica la prueba de caja blanca al método *actualizarCaracteristica()* de la clase *CaracteristicaController* utilizando el método del camino básico, para ello se procede a enumerar las sentencias del código y a partir del mismo se construye el grafo de flujo asociado.

```
public function actualizarCaracteristica(Request $request, $id) {
1   $em = $this->getDoctrine()->getManager();
    $entity = $em->getRepository('hegiccBundle:Caracteristica')->find($id);
2   if (!$entity) {
3       throw $this->createNotFoundException('Unable to find Caracteristica entity.');
    }
    $deleteForm = $this->createDeleteForm($id);
4   $editForm = $this->createEditForm($entity);
    $editForm->handleRequest($request);
    $existe = $em->getRepository("hegiccBundle:Caracteristica")->findOneByOrden($entity->getOrden());
5   if (!$existe || $existe == $entity) {
7       if ($editForm->isValid()) {
8           $em->flush();
            return $this->redirect($this->generateUrl('caracteristica', array('id' => $id)));
        }
    } else {
6       $this->get('session')->getFlashBag()->add('error', "El orden especificado ya ha sido asignado.");
    }
    return array(
9       'entity' => $entity,
        'edit_form' => $editForm->createView(),
        'delete_form' => $deleteForm->createView(),
    );
}
```

Figura 21: **Código fuente del método actualizarCaracteristica**

Se realiza el grafo de flujo partiendo del código tomado:

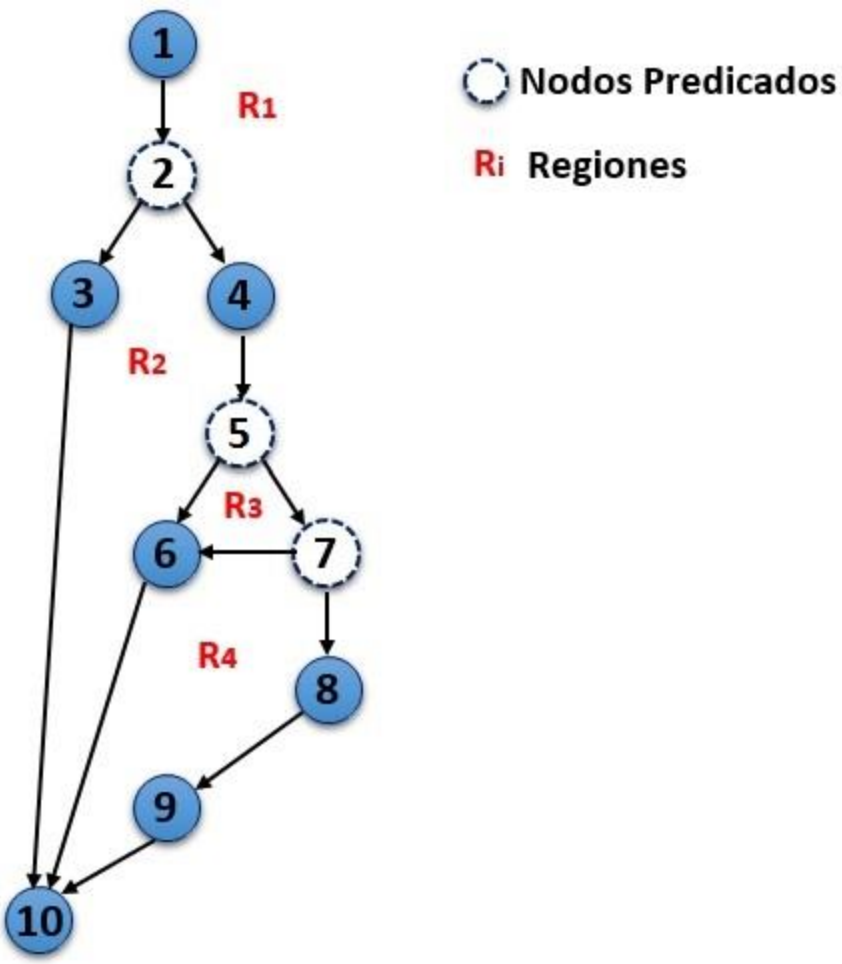


Figura 22: **Grafo de flujo del método actualizarCaracteristica**

Luego de haber realizado la construcción del grafo de flujo se procede a calcular la complejidad ciclomática mediante tres fórmulas que se describen a continuación, las cuales deben exponer el mismo resultado para asegurar que el cálculo de la complejidad es correcto.

Cada fórmula V (G) representa el valor del cálculo.

1- V (G) = (A - N) + 2

Donde **A** es el número de aristas y **N** es el número de nodos contenidos en el grafo.

V (G) = (12 - 10) + 2

V (G) = 4

2- V (G) = P + 1

Donde **P** es el número de nodos predicados contenidos en el grafo. Los nodos predicados son aquellos que parten de dos o más aristas.

V (G) = 3 + 1

V (G) = 4

3- V (G) = R

Donde **R** es la cantidad total de regiones.

V (G) = 4

El número de regiones del grafo es igual a la complejidad ciclomática.

El cálculo efectuado anteriormente dio como resultado el mismo valor en todos los casos, la complejidad ciclomática es de 4, este valor indica que existen 4 posibles caminos por donde el flujo puede circular y determina el número de casos de prueba que se deben realizar para asegurar que se ejecute cada sentencia al menos una vez. A continuación se representan los caminos básicos por los que puede transitar el flujo:

- **Camino 1:** 1 - 2 - 3 - 10
- **Camino 2:** 1 - 2 - 4 - 5 - 6 - 10
- **Camino 3:** 1 - 2 - 4 - 5 - 7 - 6 - 10
- **Camino 4:** 1 - 2 - 4 - 5 - 7 - 8 - 9 - 10

Luego de establecidos los caminos básicos se procede a realizar los casos de prueba para cada uno de ellos, de forma que los datos introducidos provoquen que se visiten las sentencias vinculadas a cada nodo del camino. A continuación se presenta un ejemplo de los 4 casos de pruebas realizados a esta funcionalidad.

Caso de prueba para el Camino 4

Tabla 16: **Caso de prueba para el Camino 4**

Entrada	Que no exista la característica que se va a editar, que el orden especificado no exista y que el formulario sea válido.
Resultados esperados	Redirecciona la página editCaracteristica.html.twig.
Condiciones	$entity==false, $existe==false, $existe == $entity, $editForm->isValid()==true

Una vez ejecutados todos los casos de pruebas obtenidos a través de la aplicación de la técnica camino básico se concluye que los mismos fueron probados satisfactoriamente demostrando que el código generado no presenta ciclos infinitos y no existe código innecesario en el sistema desarrollado.

3.4.2. Pruebas de Caja negra

Las pruebas de caja negra comprueban que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que se produce una salida correcta. Permiten detectar funcionamiento incorrecto o incompleto, errores de interfaz, errores de acceso a estructuras de datos externas, problemas de rendimiento y/o errores de inicio y terminación. Estas pruebas son llevadas a cabo sobre la interfaz del software, actuando sobre ella como una caja negra, proporcionando entradas y estudiando las salidas para ver si son o no las esperadas. Conociendo la función para la que fue diseñado, se hacen pruebas que demuestren que cada función es operativa y al mismo tiempo se buscan errores en cada una [42].
Para aplicar el método de pruebas de caja negra se va a hacer uso de la técnica de partición equivalente mediante el diseño de casos de prueba. Esta técnica se basa en la división del campo de entrada en un conjunto de clases de equivalencia, estas clases representan un grupo de datos de entrada que definen estados válidos y no válidos del sistema y se obtienen a partir de los casos de prueba.
Para la realización de las pruebas de caja negra a la solución fueron analizadas todas las funcionalidades del sistema. Se hizo una descripción de cada una de las pruebas. Se realizaron un total de 3 iteraciones para poder alcanzar los resultados satisfactorios desde el punto de vista funcional, atendiendo al correcto comportamiento del mismo ante diferentes situaciones (entradas válidas y no válidas). A continuación se realiza una descripción de las no conformidades detectadas durante la ejecución de las pruebas de caja negra. En las mismas se encontraron mayormente errores de validaciones y errores en las interfaces.

En la primera iteración se encontraron un total de 41 no conformidades (NC):

- Errores en las Interfaces 32 NC
- Errores de validación 9 NC

En la segunda iteración se encontraron un total de 5 no conformidades (NC):

- Errores en las Interfaces 3 NC
- Errores de validación 2 NC

En la tercera iteración se encontraron un total de 0 no conformidades (NC).

La Figura 23 muestra el resultado de estas pruebas según las NC detectadas durante su ejecución:

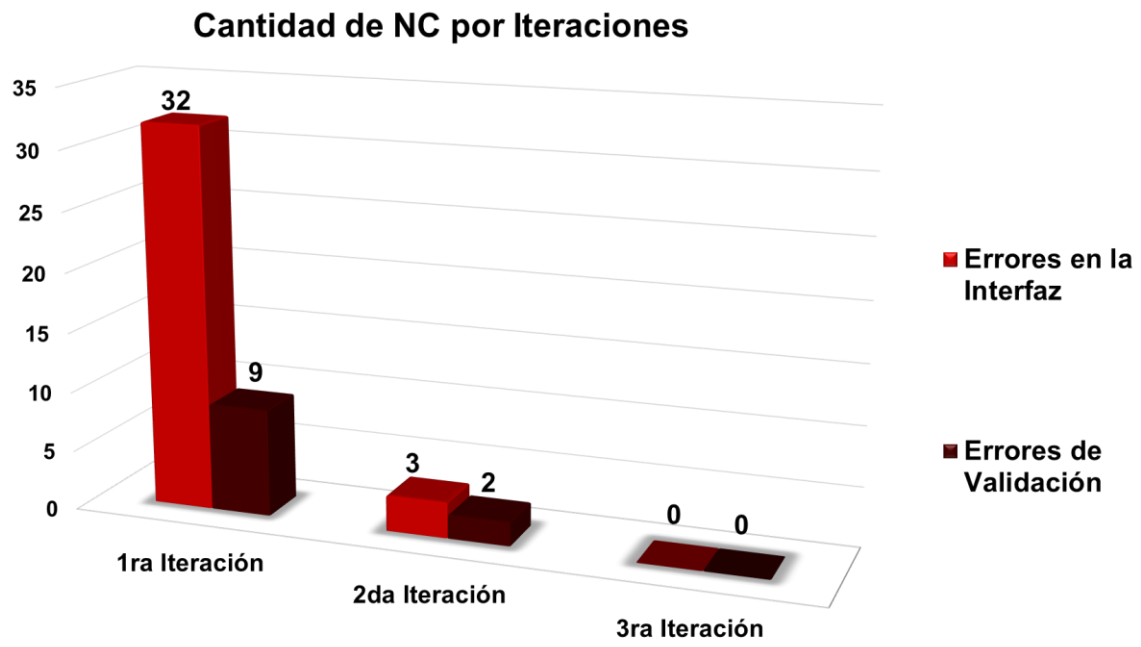


Figura 23: **Errores detectados en las pruebas funcionales.**

Los resultados alcanzados tras la aplicación de las pruebas de caja negra se pueden encontrar en el Expediente de Proyecto, específicamente en la carpeta Casos de prueba basados en Historias de Usuario. Para ver un caso de prueba aplicado, consultar Anexo 4.

3.5. Validación del Sistema

Con la validación se pretende comprobar que el software cumple las expectativas que el cliente espera. Para llevar a cabo la validación del sistema se decidió aplicar pruebas de aceptación por cada Historia de Usuario definida.

Pruebas de aceptación

Las pruebas de aceptación son destinadas a evaluar si al final de una iteración se consiguió la funcionalidad requerida por el cliente final. Estas pruebas aseguran el comportamiento del sistema y especifican los aspectos a probar cuando una Historia de Usuario ha sido correctamente implementada.

A continuación se muestra el caso de prueba de aceptación para una Historia de Usuario de prioridad alta en el negocio. Las demás se puede encontrar en el expediente de proyecto. Además en el Anexo 6 se encuentra la carta de Aceptación del Cliente.

Tabla 17: **Caso de Prueba de Aceptación Autenticación de usuarios.**

Caso de Prueba de Aceptación	
Código Caso de Prueba: M-1-1	**Nombre Historia de Usuario:** Autenticar de usuarios
Nombre de la persona que realiza la prueba: Raúl Velázquez Alvarez	
Descripción de la Prueba: Se ejecuta la prueba y se verifica que el usuario se autentique correctamente.	
Condiciones de Ejecución: • Que se inserte correctamente el usuario y la contraseña.	
Entrada / Pasos de ejecución: Se inserta un usuario y una contraseña y se verifican que sean los correctos.	
Resultado Esperado: Que el usuario acceda al sistema y se activen las funcionalidades correspondientes.	
Evaluación de la Prueba: Satisfactoria	

3.6. Validación de las variables de la investigación

Con el desarrollo del sistema, se mejora el aseguramiento de las características de calidad en los proyectos informáticos de CEGEL evaluando el grado de implementación de las mismas durante el proceso de desarrollo de software, esto se evidencia con los siguientes elementos:

Tabla 18: **Validación de las Variables.**

Antes	Después
Existía solo un conjunto de guías con elementos técnicos a tener en cuenta para implementar las características durante el proceso de desarrollo pero estas no proveían un mecanismo de evaluación.	El sistema contiene el conjunto de guías pero agrega además el mecanismo de evaluación basado en el uso de listas de chequeo que van comprobando los elementos técnicos necesarios por cada característica y en cada disciplina durante el desarrollo del producto.

Al no existir un mecanismo de evaluación no se podía conocer el grado de implementación que tenía cada característica a medida que se iba desarrollando el producto, aumentando la probabilidad de que no se hubiera tomado en cuenta.	El sistema permite evaluar cada característica por cada disciplina, utilizando los datos introducidos a partir de la aplicación de las listas de chequeo y muestra el grado de implementación a través de gráficos que permiten ver con facilidad el estado de la evaluación que puede ser: ✓ Completamente implementado ✓ Altamente implementado ✓ Parcialmente implementado ✓ No implementado
El proceso de pruebas realizado sobre el producto final era el único momento en que se podía saber el estado que tenía cada característica, trayendo consigo la detección tardía de No Conformidades, aumentando así los costos de calidad.	El sistema posibilita realizar revisiones periódicas donde se pueden detectar deficiencias desde las primeras etapas del desarrollo, dando la posibilidad al proyecto de resolver a tiempo y a un menor costo las No conformidades detectadas.

Por los elementos antes expuestos se demuestra el cumplimiento del problema de investigación planteado, constatando que con el desarrollo del sistema realizado, se contribuye la evaluación de las características de calidad en los productos informáticos de CEGEL.

3.7. Conclusiones parciales

Al finalizar el presente capítulo se puede arribar a las siguientes conclusiones:

- Se realizó el diagrama de componentes y de despliegue los cuales permitieron mostrar una vista de la aplicación a nivel de componentes y su distribución.

- Se emplearon Técnicas de Validación de los requisitos lo que permitió asegurar la validez de estos antes de comenzar el desarrollo del software.
- La aplicación de las métricas TOC y RC arrojaron resultados positivos en la validación de la aplicación propuesta.
- Se aplicaron pruebas de caja negra y caja blanca para la verificación de las funcionalidades del sistema para la evaluación de grado de implementación de las características de calidad.
- La aplicación de las pruebas de aceptación garantizó que el sistema desarrollado funciona correctamente y permitió verificar la aceptación del sistema por parte del cliente.
- Por último, se validaron las variables que forman parte del problema de la investigación, demostrando que con el sistema que se desarrolló, se mejora el aseguramiento de las características de calidad en los productos informáticos de CEGEL.

CONCLUSIONES

Durante el desarrollo de la presente investigación, se dio cumplimiento a los objetivos planteados, teniendo como conclusiones las siguientes:

- A partir del análisis de la problemática planteada y del estado del arte se entendió la necesidad de desarrollar un sistema que permita evaluar el grado de implementación de las características de calidad durante el proceso de desarrollo de software de los sistemas informáticos de CEGEL.
- El uso de la metodología SXP y la selección del conjunto de herramientas y tecnologías facilitaron el desarrollo del software, permitiendo la correcta implementación de la herramienta, lo cual quedó reflejado en todos los artefactos obtenidos a lo largo del desarrollo.
- Mediante el uso de técnicas para la validación de los requisitos se demostró una solución correcta y sin ambigüedades sobre la base de los requisitos pactados con los clientes.
- Se aplicaron métricas para la validación del diseño, la cuales arrojaron como resultado una solución estable con una alta reutilización y un bajo acoplamiento entre otros parámetros validados.
- La realización de pruebas de software ayudó a elevar la calidad y robustez del sistema, librándolo de errores y desperfectos técnicos.
- Se validaron las variables que forman parte del problema de la investigación, demostrando que con el sistema que se desarrolló, se mejora el aseguramiento de las características de calidad en los productos informáticos de CEGEL.

RECOMENDACIONES

Se recomienda poner en práctica el uso de la herramienta HEGICC para mejorar la calidad de los productos informáticos de la Universidad de las Ciencias Informáticas.

Se recomienda para una nueva versión de la herramienta que además de los dos roles que tiene se le agregue un nuevo rol para que los jefes de los proyectos puedan ver las evaluaciones y los reportes que se le realizan a sus proyectos.

BIBLIOGRAFÍA

1. DRAE. *Definicion de Calidad*. 2001; Available from: http://www.rae.es.
2. Lovelle. 1999; Available from: http://gidis.ing.unlpam.edu.ar/downloads/pdfs/Calidad_software.PDF.
3. ISO-25000, *Ingeniería de Software-Requisitos de Calidad y Evaluacion de productos de Software (SQuaRE)* 2010, La Habana, Cuba. p. 49.
4. IEEE. *Calidad de Software*. 2008 [cited 2013 Noviembre]; Available from: http://www.ieee.org/index.html.
5. Lopez, C. *Aseguramiento de las Características de Calidad*. 2001 [cited 2001 noviembre]; Available from: http://www.gestiopolis.com/canales/gerencial/articulos/27/ISO.htm.
6. ISO-9000, *Sistemas de gestión de la calidad*, in *Fundamentos y vocabulario*2000.
7. Pressman, R.S., *Software Engineering: A Practitioner´s Approach*. Séptima edición ed. 2010, New York, EEUU: McGraw-Hill. 870.
8. Modelo-Calidad. *Definición de Modelo de Calidad*. 2014; Available from: http://definicion.de/modelo-de-calidad/#ixzz2xljKzEv0.
9. ISO/IEC, *International Organization for Standardization*, 2007.
10. ISO/IEC-9126-1, *Ingeniería de Software- Calidad del Producto. Parte 1: Modelo de la Calidad*, 2004: La Habana, Cuba. p. 34.
11. ISO/IEC-25010, *Software engineering, Software product Quality Requirements and Evaluation (SQuaRE), Quality model*, 2007: Canadá. p. 46.
12. Dictionary. *Concepto de Grado*. 2014 [cited 2014; Available from: http://es.thefreedictionary.com/grado.
13. Alegsa. *Definición de Implementación (implementar)*. 2010; Available from: www.alegsaonline.com.
14. Bichachi, D.S., *Listas de Chequeo (Check-List)* 2001, 2001: Humahuaca, Argentina.
15. Vargas, L.S., Gutierrez, Agustín F., Felipe, Edgardo M. *MECHDAV: Un modelo y su Herramienta para la Evaluación Técnica de la Calidad de las Herramientas RAD para Ambientes Visuales*. 2006.
16. Cecas. *Medición PEM: Grado de implementación de las iniciativas estratégicas*. 2014 [cited 2014 21 de febrero]; Available from: http://www.cibercecas.com/medicionpem.asp.
17. Tellez, L.L. *Metodología Ágil vs Metodología Tradicional*. 2012 [cited 2014 28 de febrero]; Available from: http://es.scribd.com/doc/91676941/Metodologias-agiles-vs-tradicionales.

18. Canós, J.H., Letelier, Patricio, Panadés, M. Carmen *Metodologías ágiles en el desarrollo de software.* 2013.
19. Schwaber, K. and J. Sutherland, *La Guía Definitiva de Scrum: Las Reglas del Juego*, 2011. p. 17.
20. Peñalver, G., Meneses, A. y García, S. . *SXP, Metodología Ágil para el Desarrollo de Software*. in *1er Congreso Iberoamericano de Ingeniería de proyectos*. 2010. Chile.
21. Infante, L., *Metodología Agil - Scrum*. 2010, Bogotá, Colombia.
22. PostgreSQL. *About PostgreSQL*. 2014 [cited 2014 21 de marzo]; Available from: http://postgresql.org/about.
23. Paradigm, V. *Visual Paradigm for UML - UML tool for software application development*. 2013 [cited 2014 18 de marzo]; Available from: http://www.visual-paradigm.com/product/vpuml/.
24. Rumbaugh, J.J., Ivar; Booch, Grady, *El Lenguaje Unificado de Modelado. Manual de referencia.*, 2000, Addison, Weasley: Madrid, España. p. 552.
25. PHP. *Sitio oficial de php*. 2014 [cited 2014; Available from: http://www.php.net/manual/es/intro-whatcando.php.
26. Pacheco, N. *Manual de Twig*. 2013 [cited 2013 11 de enero]; Available from: http://gitnacho.github.io/Twig/.
27. Brandendaugh, J., *Aplicaciones JavaScript* 2000, Madrid, España: O'Reilly & Associates, Inc. 540.
28. Css3. *Características de Css3*. 2014; Available from: http://www.css3.com.
29. NetBeans-IDE. *NetBeans IDE - The Smarter and Faster Way to Code* 2013 [cited 2014 01 de abril]; Available from: https://netbeans.org/features/index.html.
30. Potencier, F.Z., François, *Guía definitiva de Symfony*. 2008, New York. EEUU. 435.
31. Doctrine. *Doctrine 2 ORM 2.0.0 documentation*. 2013 [cited 2014 28 de marzo]; Available from: http://docs.doctrine-project.org/en/2.0.x/reference/introduction.html.
32. Kabir, M., *La Biblia del Servidor Apache 2*. 2003: Anaya Multimedia. 845.
33. Fowler, M. *Is Design Dead?* 2001.
34. Clements, P. *"A Survey of Architecture Description Languages" in Proceedings of the International Workshop on Software Specification and Design*. 1996. 10.
35. Buschmann , F.M., Regine Rohnert, Hans Sornmerlad, Peter Stal, Michael *Pattern- Oriented Software Architecture: A System of Patterns*. Vol. 1. 1996, New York, EEUU: JOHN WlLEY & SONS. 467.

36. Larman, C., *UML y Patrones: Introducción al análisis y diseño orientado a objetos*. Vol. Parte VIII Temas especiales. 1999, Mexico: Prentice Hall.
37. Grosso, A. *Patrones Grasp*. 2011 [cited 2011 21 de marzo]; Available from: http://www.buenastareas.com/ensayos/Patrones-Grasp/1896730.html.
38. Aurea. *Modelo de Datos*. 2014 3 de abril]; Available from: http://gestionycontrolacademico.wordpress.com/modelos-de-datos/.
39. Arizaca, E. *Diagrama de Componentes*. 2014; Available from: http://es.scribd.com/doc/215404045/Componentes-Excelente-Explicacion.
40. Ecured. *Métricas de diseño*. 2012; Available from: http://www.ecured.cu/index.php/M%C3%A9trica_de_dise%C3%B1o.
41. *Métricas de Diseño TOC y RC*. 2014; Available from: http://www.desarrolloweb.com/articulos-copyleft/articulo-metricas-de-software.html.
42. Rojas, J.a.B., Emilio. *Pruebas de Caja Negra*. 2007; Available from: http://gemini.udistrital.edu.co/comunidad/grupos/arquisoft/fileadmin/Estudiantes/Pruebas/HTML%20-%20Pruebas%20de%20software/node26.html.