Universidad de las Ciencias Informáticas

Facultad 3

Título: Sistema de Gestión de Artistas e Instituciones Culturales del Consejo Nacional de las Artes Escénicas (SiGAE).

Trabajo de diploma para optar por el título de

Ingeniero en Ciencias Informáticas.

Autores:

Yuniel Mario Silva Menéndez

Arlet López Rumbaut

Tutor:

Msc. Manuel Álvarez Alonso

La Habana, Cuba

Curso: 2018-2019

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

_____	_____
Yuniel Mario Silva Menéndez	Arlet López Rumbaut
(Autor)	(Autora)

_____

Msc. Manuel Álvarez Alonso

AGRADECIMIENTOS

Agradezco a mis padres Mayra y Omar por apoyarme en todo momento, por alentarme y darme fuerzas cuando me sentía indefensa y sin fuerzas a lo largo de la carrera, por darme tanto cariño, por ser grandes ejemplos de que con sacrificio y esfuerzo se logran todas las cosas y a mi hermano Omarito por ser como es.

A mis abuelos: Oscar, Mercedita, Rafael(Pitity) quienes siempre me dan fuerza, aunque ya no estén y a Ernestina (la gallega) mi abuela querida que es ejemplo de sacrificio y tenacidad en la vida, siempre me alienta a dar todo de mí.

A mis tíos Pupi y Ana, a mi prima Isachy quien es un gran apoyo para mí y mi prima Heidy la cual ha sido como mi hermana desde pequeña y me ha apoyado siempre, y a toda mi familia les agradezco su apoyo y cariño.

A Linda Liz por ser mi apoyo más cercano, la que me ha aguantado todos mis enfados, mis llantos y ha estado a mi lado cada vez que me sentía mal, la verdad es que te debo mucho.

A mi compañero de tesis por su esfuerzo y dedicación en el desarrollo de la investigación y a Manuel, por ser muy buen tutor, y por su apoyo en todo momento.

A la profesora más preocupada y dedicada, Dariela por su apoyo desde que entre a la facultad y por ayudarme en todo momento, y a todos los profesores que me ayudaron de una forma u otra a lo largo de la carrera.

A Mercy por ser una gran amiga y decirme mis verdades a la cara y siempre hablarme con sinceridad, a Yianna por enseñarme a confiar en mí y por ayudarme a sentirme segura, a los amigos que estuvieron a lo largo de mi vida y ya no están, y a los que están y me apoyan en todo momento.

A todos muchas gracias. ***Arlet***

A lo largo de la carrera ha habido muchas personas que me han apoyado, se han preocupado en todo momento por el avance progresivo de mis estudios universitarios. He recibido palabras de ánimo, razones por las cuales debo esforzarme, la necesidad de superación como persona, en fin, cada cual ha dado su granito de arena y estas palabras no me alcanzan para agradecerles el apoyo moral y emocional que me han brindado desinteresadamente. Por estas razones, mis más sinceros agradecimientos a:

La Revolución, que sin su apoyo no estuviese en los altos estudios.

La Universidad de las Ciencias Informáticas, por su contribución a mi formación.

A todos los profesores que a lo largo de la carrera dedicaron su preciado tiempo a explicaciones, incluso en tiempo extracurricular, gracias

A mis compañeros de aula, apartamento y resto de la comunidad universitaria, de UCI que por tanto tiempo ha sido un hogar más.

Agradezco también a demás familiar que estuvieron pendiente del desarrollo y culminación de este ejercicio y por su preocupación.

A todos muchas gracias.

Yuniel

DEDICATORIA

A mis padres, a mi hermano y a mis abuelos por ser mi sustento y mi inspiración, a toda mi familia en general por el gran apoyo brindado.

A mis amigos por estar ahí cuando los necesito.

A todos muchas gracias por ayudarme de una forma u otra a llegar a este día.

Arlet

Dedico esta tesis a mis padres Natividad Susana Menéndez Mena y Mario Silva Rodríguez por haberme forjado como la persona que soy en la actualidad, muchos de mis logros se los debo a ellos, dentro de los que incluyo este. Me forjaron con reglas y con algunas libertades, pero al final, me motivaron constantemente para alcanzar mis anhelos.

No puedo dejar de mencionar a alguien con quien crecí, que me ha apoyado incondicionalmente en todo momento, contribuyendo a que me formé como un profesional de la Patria: mi hermano Yoelvy Silva Menéndez.

Yuniel

RESUMEN

El Centro de Gobierno Electrónico, adscrito a la Universidad de las Ciencias Informáticas ha trabajado desde su surgimiento con el objetivo de informatizar procesos de gestión en varias instituciones y órganos del estado cubano. En este centro se desarrolla un sistema informático para el Consejo Nacional de las Artes Escénicas (CNAE) con el objetivo de informatizar parte de sus procesos. Actualmente en el CNAE se ha detectado que se hace complicado y demorada la consulta de los datos de los artistas e instituciones adscritas a este, por el estado y el volumen de la documentación donde están. La información con la que se trabaja presenta deterioro progresivo por la constante búsqueda y utilización, el trabajo se desarrolla manualmente, la información se encuentra en formato duro. El presente trabajo de diploma tiene como objetivo principal el desarrollo del sistema para la gestión de artistas e instituciones culturales, de forma tal que contribuya a la persistencia y celeridad en el CNAE. Para el desarrollo de la presente investigación se empleó la metodología AUP en su variación UCI. Se emplearon las herramientas Visual Paradigm como herramienta CASE, como lenguaje de modelado UML, PostgreSQL como sistema gestor de bases de datos, como herramienta de prototipado Balsamiq Mockups, como entorno de desarrollo integrado el NetBeans, doctrine como mapeador de objetos relacionales. Al mismo tiempo, para garantizar la calidad del sistema se realizaron validaciones correspondientes y las pruebas pertinentes, obteniéndose como resultado el sistema para la gestión de artistas e instituciones culturales del CNAE.

Palabras claves: Consejo Nacional de las Artes Escénicas, artistas, celeridad, persistencia.

ÍNDICE

Resumen 6
Índice 7
Introducción 11
CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA 15
1.1 Conceptos asociados al dominio del problema 15
1.2 Sistemas homólogos 16
1.3 Conclusiones del establecimiento del estado del arte. 19
1.4 Metodología de desarrollo de software 19
1.5 Herramienta CASE (Computer Aided Software Engineering) 21
1.6 Marcos de trabajos 22
1.7 jQuery v3.3.1 23
1.8 Herramienta de Prototipado 23
1.9 Lenguajes de programación 24
1.10 HTML 5 25
1.11 CSS 3 26
1.12 Sistema Gestor de Bases de Datos 27
1.13 Mapeador Objeto-Relacional (ORM) 28
1.14 Servidor web 29
1.15 Entorno de desarrollo integrado 29
1.16 Control de versiones 30
1.17 Arquitectura de software 31
1.18 Patrones de diseño 32
Capítulo 2: Propuesta de solución 34
1.19 Descripción del negocio 34
1.20 Requisitos del software 34
2.2.1 Requisitos funcionales 35
2.2.2 Requisitos no funcionales 38
1.21 Historias de usuario 40
1.22 Arquitectura del sistema 46
1.23 Modelo de diseño 47
1.24 Diagrama de clases del diseño 50
1.25 Diagrama de componentes 51

1.26 Diagrama de despliegue 52
1.27 Modelo de datos 52
1.28 Estándares de codificación 53
Capítulo 3: Validación 59
3.1 Técnicas de validación de requisitos 59
3.2 Métricas aplicadas a los requisitos 59
3.3 Validación de los componentes 60
3.3.1 Tamaño Operacional de Clase(TOC) 61
3.3.2 Relaciones entre Clases (RC) 62
3.4 Pruebas de software 63
3.4.1 Pruebas internas 64
Pruebas unitarias 64
Pruebas funcionales 67
3.5 Pruebas de liberación 69
3.6 Pruebas de aceptación 69
3.7 Validación de las variables de investigación 69
Conclusiones del capítulo 71
CONCLUSIONES GENERALES 72
REFERENCIAS 73

ÍNDICE DE FIGURA

Figura 1: Representación de la carpeta Controller. 47
Figura 2:Representación del patrón experto. 48
Figura 3:Representación del patrón creador 48
Figura 4:Representación del patrón método de fabricación....49
Figura 5:Representación del patrón decorador. 50
Figura 6:Representación del patrón inyección de dependencias....50
Figura 7:Diagrama de clases del diseño del nomenclador Cargo....51
Figura 8:Diagrama de Componentes del sistema SiGAE. 52
Figura 9:Diagrama de despliegue. 52
Figura 10:Diagrama de Modelo de Datos....53
Figura 11:Declaración de clases. 54
Figura 12:Funciones y Métodos. 55
Figura 13:Definición de Constantes. 55
Figura 14:Ubicación y denominación. 56
Figura 15:Ubicación y denominación. 56
Figura 16:Definición de funciones. 57
Figura 17:Llamadas a funciones. 57
Figura 18:Llaves 57
Figura 19:No utilizar variables sin inicializar. 58
Figura 20:Representación de la cantidad de clases por cantidad de procedimientos. 61
Figura 21:Representación en porciento (%) de la cantidad de procedimiento, del nivel de responsabilidad, complejidad de la implementación y reutilización de las clases. 61
Figura 22:Representación de la cantidad de clases por cantidad de relaciones de usos que poseen. 62
Figura 23:Representación en porciento (%) del nivel de acoplamiento, complejidad de mantenimiento, cantidad de pruebas y reutilización de las clases. 63
Figura 24:Grafo del flujo del método adicionarModificarUnidadArtisticaAction. 65
Figura 25: Grafico de no conformidades. 68

ÍNDICE DE TABLAS

Tabla 1: Comparación entre los Sistemas Homólogos. 18
Tabla 2: Especificación de Requisitos funcionales del SiGAE....35
Tabla 3: HU Insertar Artista correspondiente al RF 26 40
Tabla 4: HU Insertar Unidad Artística correspondiente al RF 21 43
Tabla 5: Caso de prueba de la ruta independiente 1....66
Tabla 6: Validación de las variables de la investigación....70

INTRODUCCIÓN

Cuba apuesta con fuerza en la actualidad por tener una sociedad cada vez más informatizada como camino para mejorar el funcionamiento de las instituciones y la interacción de estas con otras instituciones y con la sociedad. Esta apuesta queda refrendada en el "Programa Rector de la Informatización de la Sociedad en Cuba" donde se apunta a introducción de la informática como medio para la mejora en las instituciones cubanas (Ministerio de Comunicaciones, 2017). El Centro de Gobierno Electrónico (CEGEL), adscrito a la Universidad de las Ciencias Informáticas (UCI) ha trabajado desde su surgimiento en línea con estas ideas y dentro de sus resultados introducido cuenta con varios sistemas registrales y de gestión en varias instituciones y órganos del estado cubano dentro de los que encontramos el Ministerio de Cultura (MINCULT) y su Consejo Nacional de Patrimonio Cultural (CNPC). Como resultado de estos trabajos previos y las relaciones resultantes del mismo, establece contacto con el Consejo Nacional de las Artes Escénicas (CNAE).

El CNAE es una Institución cultural encargada de promover el desarrollo de las manifestaciones teatrales y danzarías en Cuba. Bajo su atención se encuentran varias instituciones y más de 100 compañías de teatro, danza, canto lírico, pantomima, humor y circo. Promueve además el trabajo de la Revista Especializada Tablas, el Centro de Investigaciones de las Artes Escénicas, las Empresa Escenarte y Tecnoescena (Ministerio de Cultura República de Cuba, 2018). Dentro de sus funciones podemos encontrar entre otras:

- Contribuir al desarrollo de los movimientos teatrales y danzarios, atendiendo a criterios artísticos e ideológicos que permitan la mayor diversidad de expresión.
- Velar por el avance de la programación nacional, promover un público cada vez más diverso y numeroso para estas manifestaciones.
- Velar por la formación y superación permanente de los creadores.
- Fomentar el intercambio internacional en su esfera.

En el desarrollo de sus funciones se ha detectado que se hace complicado y demorada la consulta de los datos de los artistas e instituciones adscritas a la CNAE, por el estado y el volumen de la documentación donde están, impactando negativamente en la planificación de actividades concernientes a los artistas e instituciones y la atención de las direcciones de esta

institución a los mismos. La documentación donde se encuentra la información con la que se trabaja presenta deterioro progresivo, entre otras razones por la constante búsqueda y utilización. Debido a que el trabajo se desarrolla manualmente, la información se encuentra en formato duro, lo que dificulta la labor de los trabajadores, ya que no permite la realización de búsquedas concurrentes y el análisis de los datos derivados de estos es limitado, lo que restringe el proceso de toma de decisiones. Otro de los problemas es que existe la duplicidad de los datos, esto impacta negativamente en los tiempos de repuesta del CNAE a los procesos que maneja. La gestión de las unidades artísticas y el registro de sus integrantes presenta reservas de tiempo derivada de la gestión de la documentación referente a los mismos.

A partir de la problemática antes descrita se identifica como **problema a resolver**: ¿Cómo gestionar la información asociada a los artistas e instituciones culturales en el Consejo Nacional de las Artes Escénicas contribuyendo a la celeridad en las consultas y la persistencia de la misma?

Tomando en cuenta el problema antes propuesto se define como **objeto de estudio**: Proceso de desarrollo de aplicaciones de gestión.

De esta forma se determina como **objetivo general:** Desarrollar un sistema para gestionar a los artistas e instituciones culturales en el Consejo Nacional de las Artes Escénicas contribuyendo a la celeridad y persistencia de la información asociada.

Para ello se identifica como **campo de acción:** Aplicaciones de gestión de artistas e instituciones culturales.

Por lo tanto, **posibles resultados:** Una vez finalizado el presente trabajo, se tendrá como resultado una aplicación para la gestión de artistas en instituciones culturales que contribuya a gestionar la información asociada a los artistas e instituciones culturales en el Consejo Nacional de las Artes Escénicas contribuyendo a la celeridad en las consultas y la persistencia de la misma.

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

- Establecer los referentes teóricos en los que se sustenta la propuesta de solución.
- Realizar el análisis y diseño de la propuesta de solución como aproximación a la implementación.
- Implementar el sistema para la gestión de artistas en instituciones culturales del Consejo Nacional de las Artes Escénicas.
- Validar la solución propuesta mediante pruebas de software.

Para dar solución a los objetivos antes propuestos se definen los métodos científicos, clasificados en teóricos y empíricos. De los cuales se emplearon:

Métodos teóricos:

- **Histórico-lógico:** empleado en el estudio de los principales conceptos relacionados con el Consejo Nacional de las Artes Escénicas, además de adquirir conocimientos respecto a cómo se realizan las actividades en dicha institución.

- **Modelación**: se utilizará para la realización de los diagramas necesarios en el proceso de desarrollo de software, haciendo una representación abstracta de la solución, facilitando así el desarrollo de la misma.

- **Revisión Bibliográfica:** empleada en las actividades de búsqueda, identificación y análisis de la información existente en el CNAE sobre cómo se realizan los procesos y sobre los problemas existentes.

Métodos empíricos:

- **Entrevista**: permitió el intercambio verbal con el cliente para obtener la mayor cantidad de información posible, entender el proceso de negocio, comprender la estructura y funcionamiento de la organización, así como las deficiencias existentes que permitieron definir el problema a resolver y establecer el objeto de estudio.

- **Observación**: esta técnica permitió el intercambio visual en el ambiente de trabajo de los clientes y ayudó a una mejor comprensión de los procesos que se realizan en el CNAE.

El presente trabajo de diploma está estructurado en 3 capítulos de la siguiente forma:

Capítulo 1. Fundamentación teórica: Se aborda en detalle todo lo relacionado con la fundamentación teórica, con el objetivo de definir los elementos teóricos para la realización de la investigación. Se especifican algunos conceptos asociados a la investigación. Además, se analizarán las posibles herramientas de desarrollo y metodologías a utilizar para la elaboración del software. A partir de todos estos elementos se desarrolla la investigación, haciéndose más sencilla la comprensión de los temas que serán abordados en lo adelante.

Capítulo 2. Descripción de la solución propuesta: Se presenta la descripción de la solución propuesta a través de la identificación de los requisitos funcionales y no funcionales del sistema, obteniéndose la base de la arquitectura del sistema. A partir de los requisitos identificados se modelará Diagrama de clases del diseño, el cual permite obtener una visión más clara de la implementación del sistema.

Capítulo 3. Evaluación de la solución propuesta: Se evalúa el grado de calidad y fiabilidad de los resultados obtenidos en el desarrollo de este trabajo. Se aplican pruebas de software como son las pruebas de caja blanca y caja negra, para verificar la calidad del producto de software antes de ser entregado al cliente.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

En el presente capítulo se abordan los conceptos de interés asociados al dominio del problema, se realiza un análisis de la metodología de desarrollo de software empleada, así como del lenguaje de modelado y de la herramienta CASE seleccionada para entender y manejar los artefactos durante el desarrollo del software. También se exponen los patrones de diseño, así como el marco de trabajo y la arquitectura que se emplea en el proceso de construcción del sistema. Además, se profundiza en las características de las herramientas de implementación que fueron definidas en el SiGAE y que permiten una adecuada construcción del software.

1.1 Conceptos asociados al dominio del problema

Atendiendo a las características de la institución y el objetivo social descrito anteriormente se revisaron un conjunto de conceptos importantes para el entendimiento posterior del proyecto.

Artista: es una persona que ejercita las artes y produce obras artísticas. Persona que actúa profesionalmente en un espectáculo teatral, cinematográfico, circense, etc., interpretando ante el público (Pérez Porto & Merino , definicion.de, 2014).

Instalación Cultural o Instituciones Culturales: Es el espacio que permite participar de actividades culturales. Estas instalaciones tienen el objetivo de promover la cultura entre los habitantes de una comunidad. Ayuntamientos de ciudades y municipios, museos, fundaciones o incluso instituciones educativas, como pueden ser las universidades, son algunas de las entidades que normalmente cuentan con diversas instalaciones culturales con el claro objetivo de ofrecer una amplia gama de actividades de tipo cultural relacionadas consigo mismas (Pérez Porto, definicion.de, 2018).

Unidad Artística: Se refiere a la unión de los componentes con una cierta homogeneidad o identidad. Una unidad artística es una organización integrada por artistas pertenecientes a la misma fracción de un sistema cultural (Pérez Porto, definicion.de, 2014)**.**

Celeridad: Se trata de un término que hace referencia a la velocidad, la premura, la rapidez o la prisa (Pérez Porto & Gardey, Definicion.de, 2017).

Persistencia: Persistencia es la acción y efecto de persistir. En la informática, la persistencia es la propiedad de los datos para subsistir de una manera u otra (Pérez Porto & Merino, Definicion.de, 2014).

1.2 Sistemas homólogos

En el proceso de búsqueda de sistemas informáticos existentes con propósitos similares a los deseados se definieron los siguientes criterios a evaluar: sistemas basados en tecnología web, sistemas de gestión, búsquedas personalizadas, software privativo/software libre, disponibilidad del código, gestión de artistas, gestión de instituciones, generación de reportes y multiplataforma. Estos criterios permitieron identificar sistemas informáticos con características similares a las necesidades del CNAE, permitiendo conocer el objetivo y las funcionalidades que poseen los mismos. A continuación, se relacionan los sistemas que por sus características forman parte del estudio.

Sistema de Gestión de Inventario para galerías de artes. (España)

ITgallery es la manera más sencilla y elegante de gestionar, compartir y organizar tu inventario. Automatiza las tareas de tu colección y disfruta las ventajas de tener tu información bajo control en un mismo lugar.

Estas son las principales características del sistema:

- Tu inventario siempre accesible: La manera más sencilla de gestionar tu colección. Almacena todos tus datos y realiza un seguimiento de tus contactos, obras o facturas de una manera totalmente intuitiva.
- Genera ventas: Realiza cobros de obras. Acepta todas las tarjetas, en cualquier divisa.
- Máxima seguridad: Todos tus datos son encriptados y almacenados en servidores seguros con un nivel tan alto como el que utilizan las entidades bancarias manteniendo tu información completamente privada, confidencial y protegida.
- Genera documentos PDF: Sin necesidad de maquetar puedes generar portafolios, fichas técnicas, certificados de autenticidad, facturas, listados de obras o formularios de envío.

- Búsquedas personalizadas: localiza fácilmente cualquier información que necesites. Aplica criterios de búsqueda para localizar la obra, exposición venta o contacto que necesitas en cada momento (Zaya, 2018).

Agencia de Autores Visuales (ADAVIS)(Cuba)

ADAVIS es la institución que protege, mediante la Ley, las imágenes de los creadores de las artes visuales. La agencia gestiona, controla y negocia con los interesados las posibilidades de uso de las imágenes de las obras visuales; distribuye las recaudaciones provenientes de esa utilización a los titulares de los derechos; y ofrece a los autores Seguridad jurídica, comercial e institucional.

Para el ejercicio del derecho, la agencia ha implementado:

- Un Banco de Imágenes donde los autores pueden inscribir las obras que van a proteger.
- Un Sistema de Contratos con el que los utilizadores de imágenes pueden licenciar las obras que van a ser reproducidas.
- Un Sistema de Reclamaciones para aquellos casos en que por alguna vía se dañe la integridad de la obra o se transgredan los derechos de los autores.
- Un Consejo Asesor integrado por reconocidos artistas de la plástica que orientan y ayudan en la toma de decisiones (Consejo Nacional de las Artes Plásticas, 2011-2017).

Sistema Nacional de Información Cultural- SINIC (Colombia)

Herramienta Informática para el fortalecimiento del sector cultural del país.

Ingrese a nuestro Sistema de Información Nacional de Cultura, regístrese y obtenga los siguientes beneficios:

- Dar a conocer nuestro patrimonio y acervo de expresiones culturales frente a las comunidades nacional e internacional.
- Incentivar y consolidar la comunicación y el intercambio de la información cultural.
- Fuente de consulta a través de bases de datos y redes informáticas.
- Impulso a la creación de grupos de investigación y de trabajo.
- Acceso a programas y convenios institucionales. (Posibilidad de participar en programas y proyectos, a través de convocatorias realizadas desde el Ministerio de Cultura y sus Organismos Adscritos)

- Búsqueda de proveedores y clientes (Visibilidad ante el Ministerio, Entidades y Agentes Culturales en general, tanto municipales, regionales, nacionales e internacionales) (Gobierno de Colombia, 2018).

Sistema de Información para la Gestión (SIG)

A través del Sistema de Información para la Gestión, SIG, se administra y presenta la información organizada, relacionada con la gestión del Ministerio de Cultura para facilitar la toma de decisiones.

El SIG recoge, integra, difunde y administra la información del presupuesto, planes de acción, ejecución de los planes de acción, seguimiento al cumplimento de las metas, resultados de los planes y programas, indicadores generales del Ministerio y gestión de los proyectos presentados para ejecución con los recursos provenientes de impuestos, tasas o contribuciones al servicio de la telefonía móvil que benefician al sector Cultura en Colombia, girados a los Departamentos y el Distrito Capital y a los proyectos que se presenten al Ministerio de Cultura relacionados con infraestructura cultural (Gobierno de Colombia, 2018).

Tabla 1: Comparación entre los Sistemas Homólogos.

Sistemas Homólogos	Software Privativo/ Software Libre	Disponibilidad del Código	Gestión de artistas	Gestión de las Instituciones	Multiplataforma	Búsqueda Personalizada	Generación de Reportes
Sistema de Gestión de Inventario para galerías de artes.	Software Privativo		X		X	X	X
Agencia de Autores Visuales(ADAVIS).	Software Libre	X			X	X	
Sistema Nacional de Información	Software Privado		X	X	X		

Cultural- SINIC (Colombia)							
Sistema de Información para la gestión (SIG)	Software Privativo		X		X		

Fuente: Elaboración propia.

1.3 Conclusiones del establecimiento del estado del arte.

Después del análisis realizado de los diferentes sistemas se puede concluir que estos no son adaptables a las necesidades del Consejo Nacional de las Artes Escénicas(CNAE), los sistemas internacionales son privativos y sus funcionalidades no están acorde a los procesos del CNAE, en el caso del sistema homólogo cubano no se ajusta a la forma de gestionar los artistas e instituciones en el CNAE, de ninguno de los sistemas se conoce el código fuente por lo que no se hace factible la utilización o reutilización de estos. Por los elementos antes expuesto se hace inminente la necesidad de implementar un Sistema de Gestión de Artistas e Instituciones Culturales para el CNAE que cumpla con la totalidad de los requerimientos funcionales del proceso.

1.4 Metodología de desarrollo de software

En el desarrollo de software la necesidad de organizar o estructurar de forma correcta y disciplinada, es uno de los factores más importantes para evitar pérdidas de tiempo y recursos. Para evitar tales errores es preciso definir una estrategia para darle un orden a las tareas posibles a desarrollar, así como también llevar a cabo una guía de cómo efectuar las actividades, en fin, llevar a cabo un conjunto de procedimientos y pasos que se deben de seguir para el desarrollo de un software constituyendo los mismos una metodología de desarrollo de software (Rodríguez Sánchez, 2015).

Proceso Unificado Ágil (AUP) variación para la UCI

Al no existir una metodología de software universal, ya que toda metodología debe ser adaptada a las características de cada proyecto (equipo de desarrollo, recursos, etc.) exigiéndose así que

el proceso sea configurable, se decide hacer una variación de la metodología AUP, de forma tal que se adapte al ciclo de vida definido para la actividad productiva de la UCI. Una metodología de desarrollo de software tiene entre sus objetivos aumentar la calidad del software que se produce, de ahí la importancia de aplicar buenas prácticas, para ello nos apoyaremos en el Modelo CMMI-DEV v1.3. El cual constituye una guía para aplicar las mejores prácticas en una entidad desarrolladora. Estas prácticas se centran en el desarrollo de productos y servicios de calidad (Rodríguez Sánchez, 2015).

Con la adaptación de AUP que se propone para la actividad productiva de la UCI:

- Se logra estandarizar el proceso de desarrollo de software, dando cumplimiento además a las buenas prácticas que define CMMI-DEV v1.3.
- Se logra hablar un lenguaje común en cuanto a fases, disciplinas, roles y productos de trabajos.
- Se redujo a 1 la cantidad de metodologías que se usaban y de más de 20 roles en total que se definían se redujeron a 11 (Rodríguez Sánchez, 2015).

AUP propone 7 disciplinas (Modelo, Implementación, Prueba, Despliegue, Gestión de configuración, Gestión de proyecto y Entorno), se decide para el ciclo de vida de los proyectos de la UCI tener 7 disciplinas también, pero a un nivel más atómico que el definido en AUP: Modelado de negocio, Requisitos, Análisis y diseño, Implementación, Pruebas Internas, Pruebas de Liberación y Pruebas de Aceptación (Rodríguez Sánchez, 2015).

De igual forma se definen por parte de la metodología 4 escenarios para la disciplina de Requisitos. Para el caso del presente trabajo se selecciona el escenario 4. El mismo aplica a los proyectos que hayan evaluado el negocio a informatizar y como resultado obtengan un negocio muy bien definido. El cliente estará siempre acompañando al equipo de desarrollo para convenir los detalles de los requisitos y así poder implementarlos, probarlos y validarlos. Se recomienda en proyectos no muy extensos, ya que una HU no debe poseer demasiada información. En general, plantea que los proyectos que modelen negocio solo pueden modelar el sistema con HU (Rodríguez Sánchez, 2015).

Ilustración 1: Escenario No. 4 para encapsular los requisitos de software.

1.5 Herramienta CASE (Computer Aided Software Engineering)

Herramienta del software que automatiza (por lo menos en parte) una parte del ciclo de desarrollo de software. Constituyen diversas aplicaciones o programas informáticos destinados a aumentar el balance en el desarrollo de software reduciendo el costo de las mismas en términos de tiempo y de dinero. Estas herramientas sirven de ayuda en todos los aspectos del ciclo de vida de desarrollo del software en tareas como el diseño de proyectos, cálculo de costes, implementación de parte del código automáticamente con el diseño dado, Compilación automática, documentación o detección de errores entre otras (Algunos lineamientos útiles para trabajar y seleccionar herramientas CASE de manera ágil, 2012).

Visual Paradigm v8.0

Constituye una herramienta CASE profesional, que utiliza “UML” como lenguaje de modelado y que soporta el ciclo de vida completo del desarrollo de software, además de tener la ventaja de ser multiplataforma. La UCI tiene licencia para su uso y cumple con las políticas de migración a software libre en Cuba (Gaines, Boyd, & Copley, 2017).

Características:

- Soporta todos los diagramas de entidad-relación.
- Genera Java, C#, C++, PHP y lenguaje de definición de datos (DDL) para todas las bases de datos populares.
- Permite diseñar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. El mismo proporciona abundantes tutoriales del Lenguaje de Modelado, demostraciones interactivas de UML y proyectos UML.
- Se integra con herramientas Java, como son: Eclipse/IBM, NetBeansIDE, entre otras (Visual Paradigm, 2017).

A partir de los elementos antes expuestos se decide por parte del equipo de arquitectura del proyecto utilizar Visual Paradigm en su versión 8.0, teniendo en cuenta que esta herramienta propicia un conjunto de funcionalidades para el desarrollo de programas informáticos, desde la planificación, pasando por el análisis y el diseño, hasta la generación del código fuente de los

programas y la documentación. Además, se tuvo en cuenta que la UCI posee una licencia académica para su uso.

Lenguaje de modelado UML v2.0

UML se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software, no define un proceso de desarrollo específico, tan solo se trata de una notación. Se utiliza para detallar los artefactos en el sistema y definir un sistema de software (Pressman R. S., 2010).

El análisis de este lenguaje se realiza debido a que UML es la notación utilizada por la herramienta CASE que se emplea para la creación de los componentes. Además, es el lenguaje estándar de modelado para software que emplea la metodología AUP, la cual rige el proceso de desarrollo de software del SiGAE.

1.6 Marcos de trabajos

Es un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar (Acosta, 2017). En los epígrafes que se muestran a continuación, se describen las tecnologías que utiliza dicho marco de trabajo, así como las herramientas definidas por parte del equipo de arquitectura del proyecto y, por ende, utilizadas para el desarrollo de la propuesta de solución.

Symfony v3.4.9

Symfony es un marco de trabajo completo diseñado para optimizar el desarrollo de las aplicaciones web basado en el patrón Modelo Vista Controlador. Este marco de trabajo separa la lógica de negocio, la lógica de servidor y la presentación de la aplicación web. Proporciona varias herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación web compleja.

Para el desarrollo de la propuesta de solución se definió por parte del equipo de arquitectura la utilización de Symfony versión 3.4 debido a que es fácil de instalar y configurar en cualquier plataforma. De igual forma las aplicaciones desarrolladas con Symfony son compatibles con la mayoría de las plataformas, bibliotecas e infraestructuras que existen y se adaptan a entornos de negocio en cambio permanente, requiriendo menos esfuerzo para su mantenimiento.

Bootstrap v4.1.1

Es un framework web para diseño de sitios y aplicaciones web. Contiene plantillas de diseño con tipografía, formularios, botones, cuadros, menús de navegación y otros elementos de diseño basado en HTML y CSS, así como extensiones de JavaScript adicionales. A diferencia de muchos frameworks web, solo se ocupa del desarrollo front-end.

Bootstrap es de código abierto y está disponible en GitHub. Además, es modular y consiste esencialmente en una serie de hojas de estilo LESS que implementan la variedad de componentes de la herramienta. Los desarrolladores pueden adaptar el mismo archivo de Boostrap, seleccionando los componentes que deseen usar en su proyecto (Bootstrap Team, 2018).

1.7 jQuery v3.3.1

jQuery es una librería JavaScript open-source, que funciona en múltiples navegadores, y que es compatible con CSS3. Su objetivo principal es hacer la programación “scripting” mucho más fácil y rápida del lado del cliente. Con jQuery se pueden producir páginas dinámicas, así como animaciones parecidas a Flash en relativamente corto tiempo.

La ventaja principal de jQuery es que es mucho más fácil que sus competidores. Se agregan plugins fácilmente, traduciéndose esto en un ahorro sustancial de tiempo y esfuerzo. Otra ventaja de jQuery sobre sus competidores como Flash y puro CSS es su excelente integración con AJAX.

Una de las principales desventajas de jQuery es la gran cantidad de versiones publicadas en el corto tiempo. Query es fácil de instalar y aprender, inicialmente. Pero no es tan fácil si lo comparamos con CSS. Si jQuery es implementado inapropiadamente como un Framework, el entorno de desarrollo se puede salir de control.
Poniendo ventajas y desventajas a un lado, el futuro de jQuery se ve sumamente prometedor. Definitivamente es una de las mejores librerías en el mundo JavaScript (Capacity Information Tegnology Academy, 2016).

1.8 Herramienta de Prototipado

Balsamiq Mockups v3.5.15

A la hora de desarrollar una página web o una aplicación móvil es fundamental poder mostrar y concretar con el cliente el aspecto visual. Empezar a escribir líneas de código con el objetivo de tener un primer boceto puede ser contraproducente, y, sobre todo, implicar costes. Por ello, el

uso de herramientas como Balsamiq, pueden resultar de gran utilidad en las etapas iniciales de un proyecto de desarrollo (Upper, 2018).

La idea de esta herramienta es facilitarnos el trabajo de armado de prototipos de nuestras páginas web (aunque podría usarse también para aplicaciones de escritorio sin mayor problema). Requisito: Es necesario tener instalado el player de Adobe Air. Importación y exportación: Integración transparente con todas las versiones de maquetas, para cuando estés de nuevo en línea. Exporta a PNG o PDF: Comparte o presenta maquetas con incrustación de vínculos mediante la exportación a PDF o utiliza una herramienta de terceros para exportar a código (Upper, 2018).

1.9 Lenguajes de programación

No es más que un lenguaje artificial que puede ser usado para controlar el comportamiento de una máquina, especialmente una computadora. Estos se componen de un conjunto de reglas sintácticas y semánticas que permiten expresar instrucciones que luego serán interpretadas.
Existen muchos lenguajes de programación, para desarrollar aplicaciones tanto de escritorio como web (Wilson, 2013).

PHP v7.0.1

PHP (acrónimo de Hypertext Preprocessor) es un lenguaje "del lado del servidor" (esto significa que PHP funciona en un servidor remoto que procesa la página Web antes de que sea abierta por el navegador del usuario) especialmente creado para el desarrollo de páginas Web dinámicas. Éste puede ser incluido con facilidad dentro del código HTML, y permite una serie de funcionalidades extraordinarias (Wilson, 2013).

PHP está disponible para una gran cantidad de sistemas operativos diferentes. Puede escribir código PHP en todos los sistemas operativos gratuitos del tipo Unix, como Linux y FreeBSD, versiones comerciales de Unix, como Solaris e IRIX o en las diferentes versiones de Microsoft Windows.

Características:

- Posee gran número de funciones predefinidas. A diferencia de otros lenguajes de programación.
- Es multiplataforma, compatible con todos los navegadores web que se usan actualmente.

- Está dotado de un gran número de funciones que simplifican enormemente tareas habituales, como descargar documentos, enviar correos, trabajar con cookies y sesiones, entre otras funciones.
- Dispone de una amplia gama de librerías, esto permite que PHP pueda ser utilizado en muchas áreas diferentes, tales como encriptado, gráficos, XML y otras.
- Es un lenguaje de secuencia de comandos de servidor, diseñado específicamente para la Web. El código PHP es interpretado en el servidor Web y genera código HTML y otro contenido que el visitante verá.

JavaScript v1.8.5

Abreviado comúnmente JS, es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico.

Se utiliza principalmente en su forma del lado del cliente (client-side), implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas, aunque existe una forma de JavaScript del lado del servidor (Server-side JavaScript o SSJS). Su uso en aplicaciones externas a la web, por ejemplo, en documentos PDF, aplicaciones de escritorio (mayoritariamente widgets) es también significativo (Microsoft, 2018).

1.10 HTML 5

HTML5 (HyperText Markup Language) es la quinta revisión del lenguaje de marcado estándar que se emplea para la web. Es uno de los lenguajes de marcado más usados en todo el mundo y la razón es bastante obvia: gracias a HTML5 podemos crear la estructura de una página web. Texto, imágenes y material multimedia pueden mostrarse correctamente gracias a HTML5 (Formativa, 2016).

HTML en su versión 5 establece una serie de nuevos elementos y atributos que reflejan el uso típico de los sitios web modernos. Mejora el elemento <canvas>, capaz de renderizar elementos 3D en los navegadores, por ejemplo: Firefox, Chrome y Opera (Microsoft, 2018).

Debido a la adición de nuevas etiquetas que ayudan a nombrar partes de la estructura básica de toda página web (como <header>, por ejemplo), así como la eliminación de ciertas etiquetas, el código HTML se puede separar fácilmente entre etiquetas y contenido, permitiendo así que el

desarrollador pueda trabajar de manera más efectiva y detectar errores de manera más rápida (Formativa, 2016).

HTML5 es compatible con los navegadores móviles, de modo que cada página realizada en HTML5 que se ve en ordenadores, también se puede adaptar a los dispositivos móviles. Esta especificación para móviles puedes hacerla desde el mismo documento HTML o puedes emplear una framework especializada para mejorar tu productividad (Formativa, 2016).

1.11 CSS 3

Es un lenguaje de diseño gráfico para definir y crear la presentación de un documento estructurado escrito en un lenguaje de marcado. Es muy usado para establecer el diseño visual de los documentos web, e interfaces de usuario escritas en HTML o XHTML; el lenguaje puede ser aplicado a cualquier documento XML, incluyendo XHTML, SVG, XUL, RSS, etcétera. También permite aplicar estilos no visuales, como las hojas de estilo auditivas.
Está diseñado principalmente para marcar la separación del contenido del documento y la forma de presentación de este, características tales como las capas o layouts, los colores y las fuentes. Esta separación busca mejorar la accesibilidad del documento, proveer más flexibilidad y control en la especificación de características de presentaciones, permitir que varios documentos HTML compartan un mismo estilo usando una sola hoja de estilos separada en un archivo .css, y reducir la complejidad y la repetición de código en la estructura del documento (w3schools, 2018).

DQL (Doctrine Query Language)

DQL significa Doctrina lenguaje de consulta y es un derivado del lenguaje de consulta de objetos que es muy similar al lenguaje de consultas de Hibernate (HQL) o el lenguaje de consulta de Java Persistence (JPQL).

Un error común de los principiantes es confundir DQL por ser simplemente una forma de SQL y, por tanto, tratar de utilizar nombres de tabla y nombres de columna o unir tablas arbitrarias juntos en una consulta. Es necesario pensar en DQL como un lenguaje de consulta para su modelo de objetos, no para su esquema relacional.

DQL como un lenguaje de consulta tiene SELECT, UPDATE y DELETE construcciones que se asignan a sus correspondientes tipos de sentencias SQL. Las instrucciones INSERT no están

permitidos en DQL, porque las entidades y sus relaciones tienen que ser introducidos en el contexto de persistencia a través EntityManager#persist()de garantizar la coherencia de su modelo de objetos (doctrine-project.org, 2018).

1.12 Sistema Gestor de Bases de Datos

PostgreSQL v10.1

Es uno de los sistemas gestores de bases de datos de código abierto más avanzado del mundo. Además, constituye un sistema de gestión de bases de datos objeto-relacional (ORDBMS) basado en Postgres. Dentro de sus principales ventajas se encuentran:

- Soporte de protocolo de comunicación encriptado por SSL.
- Extensiones para alta disponibilidad, nuevos tipos de índices, datos espaciales, minería de datos, etc.
- Incorpora una estructura de datos array.
- Incorpora funciones de diversa índole: manejo de fechas, geométricas, orientadas a operaciones con redes, etc.
- Permite la declaración de funciones propias, así como la definición de disparadores.
- Soporta el uso de índices, reglas y vistas.
- Incluye herencia entre tablas (aunque no entre objetos, ya que no existen), por lo que a este gestor de bases de datos se le incluye entre los gestores objeto-relacionales.
- Permite la gestión de diferentes usuarios, como también los permisos asignados a cada uno de ellos (Marit, 2019).

El gestor de Bases de datos que será utilizado es PostgreSQL en su versión 10.1.3 es un potente motor de bases de datos, que tiene prestaciones y funcionalidades equivalentes a muchos gestores de bases de datos comerciales.

Un Sistema Gestor de Bases de Datos (SGBD) permite definir, crear y mantener las bases de datos, además permite la autonomía entre los datos y los programas de aplicación, minimizar las redundancias, garantiza la integridad, la seguridad y protección de los datos, proporciona la manipulación de la información, permite un control centralizado, también proporciona un acceso controlado a estas y un lenguaje de manejo de datos para la inserción, actualización, eliminación y consulta de información en estas bases de datos (Tutorial de SQL, 2012).

PgAdmin III v1.12.0

Es una aplicación gráfica para gestionar el gestor de bases de datos PostgreSQL, siendo la más completa y popular con licencia Open Source. Está diseñado para responder a las necesidades de todos los usuarios, desde escribir consultas SQL simples hasta desarrollar bases de datos complejas. El interfaz gráfico soporta todas las características de PostgreSQL y facilita enormemente la administración. La conexión al servidor puede hacerse mediante conexión TCP/IP o Unix Domain Sockets (en plataformas *nix), y puede encriptarse mediante SSL para mayor seguridad (PostgreSQL Global Development Group, 2018).

1.13 Mapeador Objeto-Relacional (ORM)

Permite convertir los datos de los objetos en un formato correcto para poder guardar la información en una base de datos (mapeo) creándose una base de datos virtual donde los datos que se encuentran en la aplicación, quedarán vinculados a la base de datos.
Mapeo: transformar toda la información que se recibe de la base datos, principalmente en tablas, en los objetos de tu aplicación y viceversa (Zoega, 2015).

Doctrine v2.4

Es un mapeador de objetos-relacional (ORM) escrito en PHP que proporciona una capa de persistencia para objetos PHP. Es una capa de abstracción que se sitúa justo encima de un SGBD (Sistema de gestión de bases de datos).

Características:

- Bajo nivel de configuración que necesita para empezar un proyecto. Puede generar clases a partir de una base de datos existente y después el programador puede especificar relaciones y añadir funcionalidad extra a las clases autogeneradas.
- Posibilidad de escribir consultas de base de datos utilizando un dialecto de SQL denominado DQL (Doctrine Query Language) que está inspirado en Hibernate (Java) .
- Diversos comportamientos del modelo (conjuntos anidados, internacionalización, log, índice de búsqueda)
- Una función "compilar" que combina varios archivos PHP del framework en uno solo para evitar el descenso de rendimiento que provoca incluir varios archivos PHP (Bootstrap Team, 2018).

1.14 Servidor web

Un servidor web sirve contenido estático a un navegador, carga un archivo y lo sirve a través de la red al navegador de un usuario. Este intercambio es mediado por el navegador y el servidor que hablan el uno con el otro mediante HTTP. Se pueden utilizar varias tecnologías en el servidor para aumentar su potencia más allá de su capacidad de entregar páginas HTML; éstas incluyen scripts CGI, seguridad SSL y páginas activas del servidor (ASP) entre otros (i-Concepts) (Netcraft, 2018).

Apache v2.4.23

Apache, es un servidor web hecho por excelencia, su configurabilidad, robustez y estabilidad hacen que cada vez millones de servidores reiteren su confianza en este programa.

Características:

- Multiplataforma.
- Es una tecnología gratuita de código abierto.
- Es un servidor altamente configurable de diseño modular
- Permite personalizar la respuesta ante los posibles errores que se puedan dar en el servidor.

El servidor web Apache 2.4 proporciona contenidos al cliente web o navegador como:

Páginas estáticas: es el uso más generalizado que se hace de un servidor web. De esta forma se transfieren archivos HTML, imágenes, y no se requiere un servidor muy potente en lo que al hardware se refiere.

Páginas dinámicas: la información que muestran las páginas que sirve Apache cambia ya que se obtiene a partir de consultas a bases de datos u otras fuentes de datos. Son, por tanto, páginas con contenido dinámico, cambiante (The Apache Software Fundation, 2019).

1.15 Entorno de desarrollo integrado

Un entorno de desarrollo integrado, llamado también IDE (sigla en inglés de integrated development environment), es un programa informático compuesto por un conjunto de herramientas de programación. Puede dedicarse en exclusiva a un solo lenguaje de programación o bien poder utilizarse para varios. Normalmente, un IDE consiste de un editor de código fuente, herramientas

de construcción automáticas y un depurador. La mayoría de estos tienen auto-completado inteligente de código (IntelliSense) (Dutoit, D., 2014).

Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI). Los IDEs pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes (Dutoit, D., 2014).

NetBeans v8.2

Es un entorno de desarrollo integrado libre, hecho principalmente para el lenguaje de programación Java. Existe además un número importante de módulos para extenderlo. NetBeans IDE es un producto libre y gratuito sin restricciones de uso.

Permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados módulos. Un módulo es un archivo Java que contiene clases de java escritas para interactuar con las Apis de NetBeans y un archivo especial (manifest file) que lo identifica como módulo. Las aplicaciones construidas a partir de módulos pueden ser extendidas agregándole nuevos módulos. Debido a que los módulos pueden ser desarrollados independientemente, las aplicaciones basadas en la plataforma NetBeans pueden ser extendidas fácilmente por otros desarrolladores de software (Domíngez, 2018).

1.16 Control de versiones

Se llama control de versiones a la gestión de los diversos cambios que se realizan sobre los elementos de algún producto o una configuración del mismo. Una versión, revisión o edición de un producto, es el estado en el que se encuentra el mismo en un momento dado de su desarrollo o modificación.

Aunque un sistema de control de versiones puede realizarse de forma manual, es muy aconsejable disponer de herramientas que faciliten esta gestión dando lugar a los llamados sistemas de control de versiones o VCS (del inglés Version Control System). Estos sistemas facilitan la administración de las distintas versiones de cada producto desarrollado, así como las posibles especializaciones realizadas (por ejemplo, para algún cliente específico (Hernandez, 2014).

Git v2.17

Es un software de control de versiones diseñado por Linus Torvalds. Fue creado pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente. Nos proporciona las herramientas para desarrollar un trabajo en equipo de manera inteligente y rápida y por trabajo nos referimos a algún software o página que implique código el cual necesitemos hacerlo con un grupo de personas.

Algunas de las características más importantes son:

- Rapidez en la gestión de ramas: debido a que nos dice que un cambio será fusionado mucho más frecuentemente de lo que se escribe originalmente.
- Gestión distribuida: Los cambios se importan como ramas adicionales y pueden ser fusionados de la misma manera como se hace en la rama local.
- Gestión eficiente de proyectos grandes y realmacenamiento periódico en paquetes (Robles, 2018).

1.17 Arquitectura de software

La arquitectura de un sistema es la estructura del sistema, la cual comprende los componentes del software, las propiedades de esos componentes visibles externamente y las relaciones entre ellos (Pressman R. S., 2010).

Una arquitectura de software se selecciona y diseña con base en objetivos (requisitos) y restricciones. Los objetivos son aquellos prefijados para el sistema de información, pero no solamente los de tipo funcional, también otros objetivos como la mantenibilidad, auditabilidad, flexibilidad e interacción con otros sistemas de información. Las restricciones son aquellas limitaciones derivadas de las tecnologías disponibles para implementar sistemas de información. Unas arquitecturas son más recomendables de implementar con ciertas tecnologías mientras que otras tecnologías no son aptas para determinadas arquitecturas. Por ejemplo, no es viable emplear una arquitectura de software de tres capas para implementar sistemas en tiempo real (Reynoso, 2011).

Patrón Modelo-Vista-Controlador (MVC)

Fue una de las primeras ideas en el campo de las interfaces gráficas de usuario y uno de los primeros trabajos en describir e implementar aplicaciones de software en términos de sus diferentes funciones (Nube Colectiva, 2019).

De manera genérica, los componentes de MVC se podrían definir como sigue:

Modelo: Es un objeto que representa cierta información del dominio. Constituye un objeto no visual y contiene todos los datos y comportamiento del mismo.
Vista: La vista representa la muestra del modelo mediante la interfaz visual.
Controlador: Se encarga de cambiar cualquier información existente en el modelo. Toma las entradas del usuario, manipula el modelo y actualiza la vista apropiadamente.

1.18 Patrones de diseño

Los patrones de diseño son unas técnicas para resolver problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces.

Un patrón de diseño resulta ser una solución a un problema de diseño. Para que una solución sea considerada un patrón debe poseer ciertas características. Una de ellas es que debe haber comprobado su efectividad resolviendo problemas similares en ocasiones anteriores. Otra es que debe ser reutilizable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias (Pressman R. S., 2010).

Patrones GRASP: son patrones generales de software para asignación de responsabilidades. Son una ayuda en el aprendizaje que permiten al desarrollador entender lo esencial del diseño de objetos y a aplicar el razonamiento de una forma metódica, racional y explicable (Pressman R. S., 2010).

Patrones GoF: Son soluciones concretas y técnicas basadas en la Programación Orientada a Objetos (POO). Se utilizan en situaciones frecuentes, debido a que se basan en la experiencia acumulada al resolver problemas reiterativos. Favorecen la reutilización de código. Ayudan a construir software basado en la reutilización, a construir clases reutilizables. A continuación, se presentan las formas de clasificar estos patrones según su propósito:
Creacionales: Se encargan de la creación de instancias de los objetos.

Estructurales: Son los que plantean las relaciones entre clases, las combinan y forman estructuras mayores.
Comportamiento: Plantea la interacción y cooperación entre las clases.

Conclusiones parciales

Enfocando este trabajo a las necesidades que posee el Consejo Nacional de las Artes Escénicas (CNAE), a partir del problema planteado, se realizó un estudio de las principales herramientas y tecnologías más usadas en el mundo en la realización de una aplicación web. El estudio realizado del marco teórico fundamenta la necesidad de la utilización de la herramienta informática propuesta, además de las tecnologías y herramientas que se utilizarán. De igual manera la investigación sobre los principales conceptos que se manejan, permite lograr una mejor comprensión del negocio y la comunicación con el cliente. Todo lo planteado anteriormente sienta las bases para el futuro desarrollo de la aplicación. Además, este estudio resultó de gran importancia para una mejor comprensión del objetivo del actual trabajo.

CAPÍTULO 2: PROPUESTA DE SOLUCIÓN

Introducción

El presente capítulo contiene la información relacionada con el análisis y el diseño de la propuesta de solución. Se describen los procedimientos llevados a cabo para la construcción de la aplicación, utilizando la metodología de desarrollo de software seleccionada para el desarrollo de la investigación. Se evidencian las necesidades del cliente, los requisitos que el sistema debe cumplir y se define el comportamiento del mismo, así como las restricciones del diseño, concebidas para la construcción de la aplicación. A continuación, se realiza una breve explicación del funcionamiento del negocio.

1.19 Descripción del negocio

El Consejo Nacional de Artes Escénicas (CNAE) es el centro rector de las actividades culturales que ocurren en el país. Este es el encargado de velar por las instituciones artísticas de todas las provincias, de las cuales se registran una serie de informaciones para la realización de cronogramas y carteleras culturales. En el centro existe un departamento de recursos humanos, el cual gestiona la información de los artistas y las unidades artísticas. Los artistas se dividen por categorías según la especialidad que estén avalados, y según los documentos rectores que lo acrediten. Las unidades artísticas están compuestas por artistas principalmente, aunque puede estar vinculadas a ellas otras personas como técnicos, personal de servicios, directores artísticos, generales, entre otros. Los artistas pueden estar vinculados o no a una unidad artística o en varias unidades artísticas de manera temporal.

1.20 Requisitos del software

Un requisito es simplemente una declaración abstracta de alto nivel de un servicio que debe proporcionar el sistema o una restricción de éste. En el otro extremo, es una definición detallada y formal de una función del sistema (Sanchez, 2018).

La gestión de requisitos es un conjunto de actividades que ayudan al equipo del proyecto a identificar, controlar y rastrear los requisitos y los cambios a éstos en cualquier momento mientras se desarrolla el proyecto (Pressman R. S., 2010).

2.2.1 Requisitos funcionales

Los requisitos funcionales (RF) de un sistema, son aquellos que describen cualquier actividad que este deba realizar, en otras palabras, el comportamiento o función particular de un sistema o software cuando se cumplen ciertas condiciones. Por lo general, éstos deben incluir funciones desempeñadas por pantallas específicas, descripciones de los flujos de trabajo a ser desempeñados por el sistema y otros requerimientos de negocio, cumplimiento, seguridad u otra índole (Pressman R. S., 2010).

El equipo de desarrollo identificó 53 requisitos funcionales (RF), en la Tabla 2 se muestra la especificación de estos requisitos:

Tabla 2: Especificación de Requisitos funcionales del SiGAE

No.	Nombre	Descripción
RF1	Insertar Entidad	Permitirá insertar una entidad al sistema con sus datos correspondientes.
RF2	Modificar Entidad	Permitirá modificar una entidad en el sistema.
RF3	Listar Entidad	Permitirá observar un listado de las entidades que hay en el sistema.
RF4	Activar/Desactivar entidad	Permitirá desactivar o activar la entidad según lo necesario.
RF5	Insertar Especialidad	Permitirá insertar una especialidad al sistema con sus datos correspondientes.
RF6	Modificar Especialidad	Permitirá modificar una especialidad en el sistema
RF7	Listar Especialidad	Permitirá observar un listado de las especialidades que hay en el sistema.
RF8	Activar/Desactivar especialidad	Permitirá desactivar o activar la especialidad según lo necesario.
RF9	Insertar País	Permitirá insertar una país al sistema con sus datos correspondientes.
RF10	Modificar País	Permitirá modificar un país en el sistema.

RF11	Listar País	Permitirá observar un listado de países que hay en el sistema.
RF12	Activar/Desactivar país	Permitirá desactivar o activar el país según lo necesario.
RF13	Insertar tipo de público	Permitirá insertar el tipo de público al sistema con sus datos correspondientes.
RF14	Modificar tipo de público	Permitirá modificar el tipo de público en el sistema.
RF15	Listar tipo de público	Permitirá observar un listado de los tipos de públicos que hay en el sistema.
RF16	Activar/Desactivar tipo de público	Permitirá desactivar o activar el tipo de público según lo necesario.
RF17	Insertar cargo	Permitirá insertar un cargo al sistema con sus datos correspondientes.
RF18	Modificar cargo	Permitirá modificar un cargo en el sistema.
RF19	Listar cargo	Permitirá observar un listado de cargos que hay en el sistema
RF20	Activar/Desactivar cargo	Permitirá desactivar o activar el cargo según lo necesario.
RF21	Insertar Unidad Artística	Permitirá insertar una unidad artística al sistema con sus datos correspondientes.
RF22	Modificar Unidad Artística	Permitirá modificar una unidad artística en el sistema.
RF23	Listar Unidad Artística	Permitirá observar un listado de unidades artísticas que hay en el sistema.
RF24	Visualizar detalles Unidad Artística	Permitirá conocer todos los datos de la unidad artística.
RF25	Activar/Desactivar Unidad Artística	Permitirá desactivar o activar la unidad artística según lo necesario.
RF26	Insertar Artista	Permitirá insertar un artista al sistema con sus datos correspondientes.
RF27	Modificar Artista	Permitirá modificar un artista en el sistema.

RF28	Listar Artista	Permitirá observar el listado de artistas que hay en el sistema.
RF29	Visualizar detalles Artista	Permitirá conocer todos los datos de del artista.
RF30	Activar/Desactivar Artista	Permitirá desactivar o activar el artista según lo necesario.
RF31	Insertar Instalación	Permitirá insertar una instalación al sistema con sus datos correspondientes.
RF32	Modificar Instalación	Permitirá modificar una instalación en el sistema.
RF33	Listar Instalación	Permitirá observar el listado de instalaciones que hay en el sistema.
RF34	Visualizar detalles Instalación	Permitirá conocer todos los datos de la instalación.
RF35	Activar/Desactivar Instalación	Permitirá desactivar o activar la instalación según lo necesario.
RF36	Insertar tipo de espacio	Permitirá insertar un tipo de espacio al sistema con sus datos correspondientes.
RF37	Modificar tipo de espacio	Permitirá modificar un tipo de espacio en el sistema.
RF38	Listar tipo de espacio	Permitirá observar el listado de los tipos de espacios que hay en el sistema.
RF39	Activar/Desactivar tipo de espacio	Permitirá desactivar o activar el tipo de espacio según lo necesario.
RF40	Insertar tipo de instalación	Permitirá insertar un tipo de instalación al sistema con sus datos correspondientes.
RF41	Modificar tipo de instalación	Permitirá modificar un tipo de instalación en el sistema.
RF42	Listar tipo de instalación	Permitirá observar el listado de los tipos de instalaciones que hay en el sistema.
RF43	Activar/Desactivar tipo de instalación	Permitirá desactivar o activar el tipo de instalación según lo necesario.
RF44	Buscar por Filtro	Permitirá hacer búsquedas por datos específicos.

RF45	Generar Reporte	Permitirá generar reportes sobre la cantidad de artistas por provincia y municipio, cantidad de artistas de género masculino y femenino, entre otros.
RF46	Mostrar vista previa del reporte	Permitirá mostrar una vista detallada de reporte.
RF47	Insertar Usuario	Permitirá insertar un usuario al sistema con sus datos correspondientes.
RF48	Modificar Usuario	Permitirá modificar un usuario en el sistema.
RF49	Cambiar contraseña	Permitirá cambiar la contraseña del usuario.
RF50	Activar/Desactivar usuario	Permitirá desactivar o activar un usuario según lo necesario.
RF51	Listar Usuario	Permitirá observar el listado de usuarios que están en el sistema.
RF52	Listar trazas	Permitirá observar el listado de las trazas del sistema.
RF53	Buscar trazas	Permitirá buscar las trazas del sistema.

Fuente: Elaboración propia

2.2.2 Requisitos no funcionales

Los requisitos no funcionales (RnF) son restricciones de los servicios o funciones ofrecidos por el sistema. Incluyen restricciones de tiempo sobre el proceso de desarrollo y estándares. A menudo son aplicados en su totalidad al sistema y normalmente apenas se aplican a características o servicios individuales del sistema (Pressman R. S., 2010).

Usabilidad

RnF 1. En el sistema se deben visualizar todos los mensajes en idioma español. La tipografía debe ser uniforme, de un tamaño adecuado.

RnF 2. El sistema debe facilitar la interacción con el usuario, posibilitando la utilización de combinaciones de teclas, podrán utilizarse los campos de selección en la interfaz en los casos que sea posible y se agruparán los vínculos y botones por grupos funcionales siempre que se cumpla con las pautas de diseño de las interfaces.

RnF 3. El sistema debe ofrecer una interfaz amigable, fácil de operar. Igualmente tiene que mantener la línea de diseño establecida la cual mantiene la uniformidad y representatividad de la solución. Las interfaces deben poseer un diseño sencillo, con pocas entradas, permitiendo un balance adecuado entre funcionalidad y simplicidad de tal manera que no se haga difícil para los usuarios la utilización del mismo.

Seguridad

RnF 4. Ante el fallo de una funcionalidad del sistema, el resto de las funcionalidades que no dependen de esta deberán seguir funcionando.

RnF 5. El sistema concederá acceso a cada usuario autenticado solo a las funcionalidades que le estén permitidas de acuerdo al rol especificado en el sistema.

RnF 6. Ante el fallo de una funcionalidad del sistema, en medio de una transacción el sistema mostrará un mensaje de fallo en la transacción.

Portabilidad

RnF 7. El sistema deberá adaptarse al entorno operativo siempre y cuando este cumpla con las características de la aplicación y debe coexistir con otros sistemas sin dificultad.

Eficiencia

RnF 8. El sistema debe responder en intervalos de tiempos aceptables en dependencia de la carga de operaciones y la velocidad de conexión.

Hardware

RnF 9. La estación de trabajo debe contar como mínimo con un procesador Core i3, RAM de 8GB, disco duro de 500 GB y puerto de red.

Software

RnF 10. Se recomienda que estaciones de trabajo posean un navegador web Mozilla Firefox 54.0 o una versión superior.

RnF 11. El servidor de aplicaciones web es Apache 2.4.23.

RnF 12. El servidor de base de datos deberá ser PostgreSQL v9.4. en adelante.

1.21 Historias de usuario

Las historias de usuario (HU) son una forma de administrar con rapidez los requisitos de los usuarios sin tener que elaborar gran cantidad de documentos formales y sin requerir de mucho tiempo para administrarlos. Las historias de usuario permiten responder rápidamente a los requisitos cambiantes, por lo que una historia de usuario puede tener varios cambios a lo largo de un desarrollo sin afectarse el tiempo (Palacios, 2016).

En total se definieron 49 historias de usuario de las cuales se presentan a continuación las correspondientes al RF26 y al RF21:

Tabla 3: HU Insertar Artista correspondiente al RF 26

Número: 1	**Nombre del requisito:** Insertar artista
Programador:	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** 10 días
Riesgo en Desarrollo: alto	**Tiempo Real:** 7 días

Descripción: El usuario podrá acceder en el menú, desde la opción de Registro. Esta funcionalidad permite insertar un nuevo artista de las Artes Escénicas. Del Artista se recogen los siguientes datos:

- **Nombre:** Campo de texto. Se inserta el nombre de la unidad artística, admite solo letras. Campo obligatorio.
- **Cargo:** Lista desplegable. Se selecciona el tipo de cargo que tiene el artista. Las opciones son: artístico, técnico, de apoyo. Campo obligatorio
- **Fecha de Nacimiento:** Campo de fecha, se selecciona la fecha de nacimiento del artista. El rango de fechas será desde 1/1/1920 hasta la fecha actual. Campo obligatorio.
- **Dirección:** Campo de texto, Se inserta la dirección del artista, admite cualquier tipo de caracteres. Campo obligatorio.
- **Provincia:** Campo de selección. Se inserta la provincia del artista, admite solo texto. Campo obligatorio.
- **Municipio:** Campo de selección. Se inserta el municipio en el que vive el artista, admite solo texto. Campo obligatorio.
- **Carné de Identidad:** Campo de texto. Se inserta el carné de identidad del particular. Admite solo números y 11 caracteres. Campo obligatorio.
- **Género:** Lista desplegable. Se selecciona el género del artista. Campo obligatorio.

- **Pasaporte:**
  - **Tipo de pasaporte:** Campo de selección, se selecciona el tipo de pasaporte del artista.
  - **Número:** Campo de texto, admite número, letras en mayúscula y en total 8 caracteres. Se inserta el número a continuación del tipo del pasaporte.
- **Evaluación:** Campo de tipo file. Contiene la evaluación del artista. Campo obligatorio.
- **Escolaridad:** Campo de tipo file. Contiene la escolaridad del artista. Campo obligatorio
- **Entidad:** Campo de tipo selección. Se inserta mediante una tabla que contiene todas las entidades a las que puede pertenecer un artista.
- **Experiencia Internacional:** Campo de tipo selección. Se inserta mediante una tabla, que contiene los países a los viajan los artistas.
- **Premios:** Campo de tipo selección. Se inserta mediante una tabla, que contiene todos los premios que pueden alcanzar los artistas.
- **Otros Vínculos:** Campo de tipo texto, se inserta/n el/los vínculo/s que tiene artista
- **Servicio Social:** Campo de tipo selección. Indica si el artista está o no en servicio social.

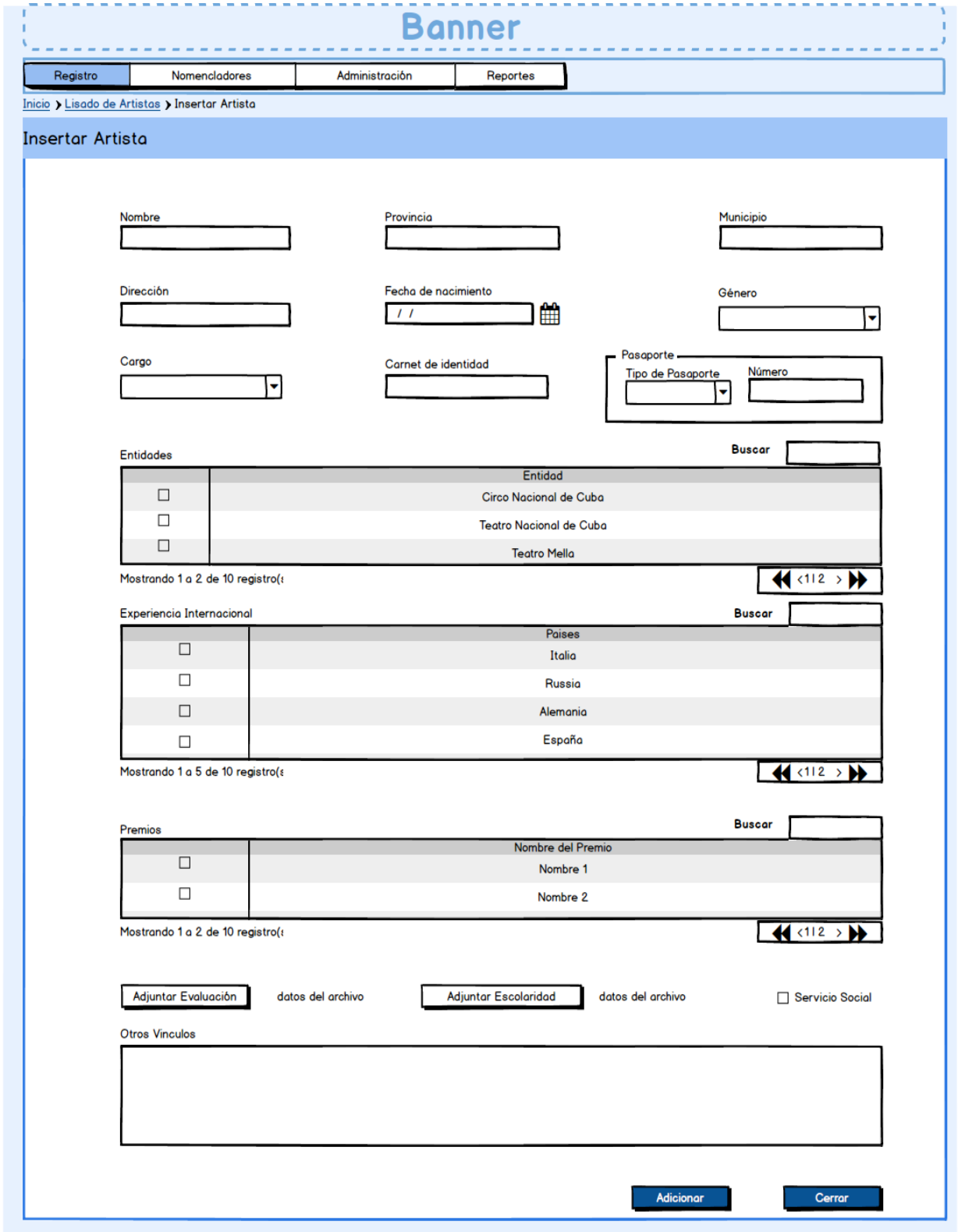
Se pueden realizar las siguientes acciones:

- **Adicionar**: Verifica que no existan campos obligatorios vacíos, guarda los datos y crea el artista.
- **Cerrar**: Redirecciona a la interfaz anterior. No guarda los datos.

Observaciones:

- Cada artista creado pertenece a una unidad artística
- Los campos de Evaluación y Escolaridad, almacenan adjuntos en formato documento (doc, docx, pdf)
- Cada artista puede pertenecer a más de una entidad.

Prototipo elemental de interfaz gráfica de usuario:



Fuente: Elaboración propia

Tabla 4: HU Insertar Unidad Artística correspondiente al RF 21

Número: 6	**Nombre del requisito:** Insertar Unidad Artística
Programador:	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** 10 días
Riesgo en Desarrollo: alto	**Tiempo Real:** 7 días

Descripción: El usuario podrá acceder en el menú, desde la opción de Registro. Permite insertar las Unidades Artísticas pertenecientes a las Entidades de las Artes Escénicas. De la Unidad Artística se recogen los siguientes datos:

- **Nombre de la unidad:** Campo de texto. Se inserta el nombre de la Unidad Artística, admite letras y números. Campo obligatorio.
- **Especialidad:** Campo de selección. Se selecciona la especialidad de la Unidad Artística. Campo obligatorio.
- **Descripción:** Campo de texto, se inserta una breve descripción del recorrido o aspectos relevantes de la Unidad Artística.
- **Fecha Fundación:** Campo de fecha. Se inserta la fecha de fundación de la Unidad Artística. El rango de fecha será desde 1/1/1900 hasta la fecha actual. Campo obligatorio.
- **Documentos de aprobación:** Campo de archivo. Se adjunta la evaluación/ aprobación de la Unidad Artística.
- **Director:** Campo de texto. Se inserta el nombre del director de la Unidad Artística. Admite solo letras.
- **Dirección:** Campo de texto. Se inserta la dirección de la Unidad Artística. Admite cualquier tipo de carácter. Campo obligatorio.
- **Provincia:** Campo de texto. Se inserta la provincia en la que se encuentra la Unidad Artística, admite solo texto. Campo obligatorio.
- **Municipio:** Campo de texto. Se inserta el municipio en el que se encuentra la Unidad Artística, admite solo texto. Campo obligatorio.
- **Página Web**: Campo de texto. Se inserta la página web del grupo. Admite solo los siguientes caracteres: letras, números", "-", "_" y "."
- **Teléfono**: Campo de texto. Se inserta el número de teléfono de la Unidad Artística. Admite solo números.

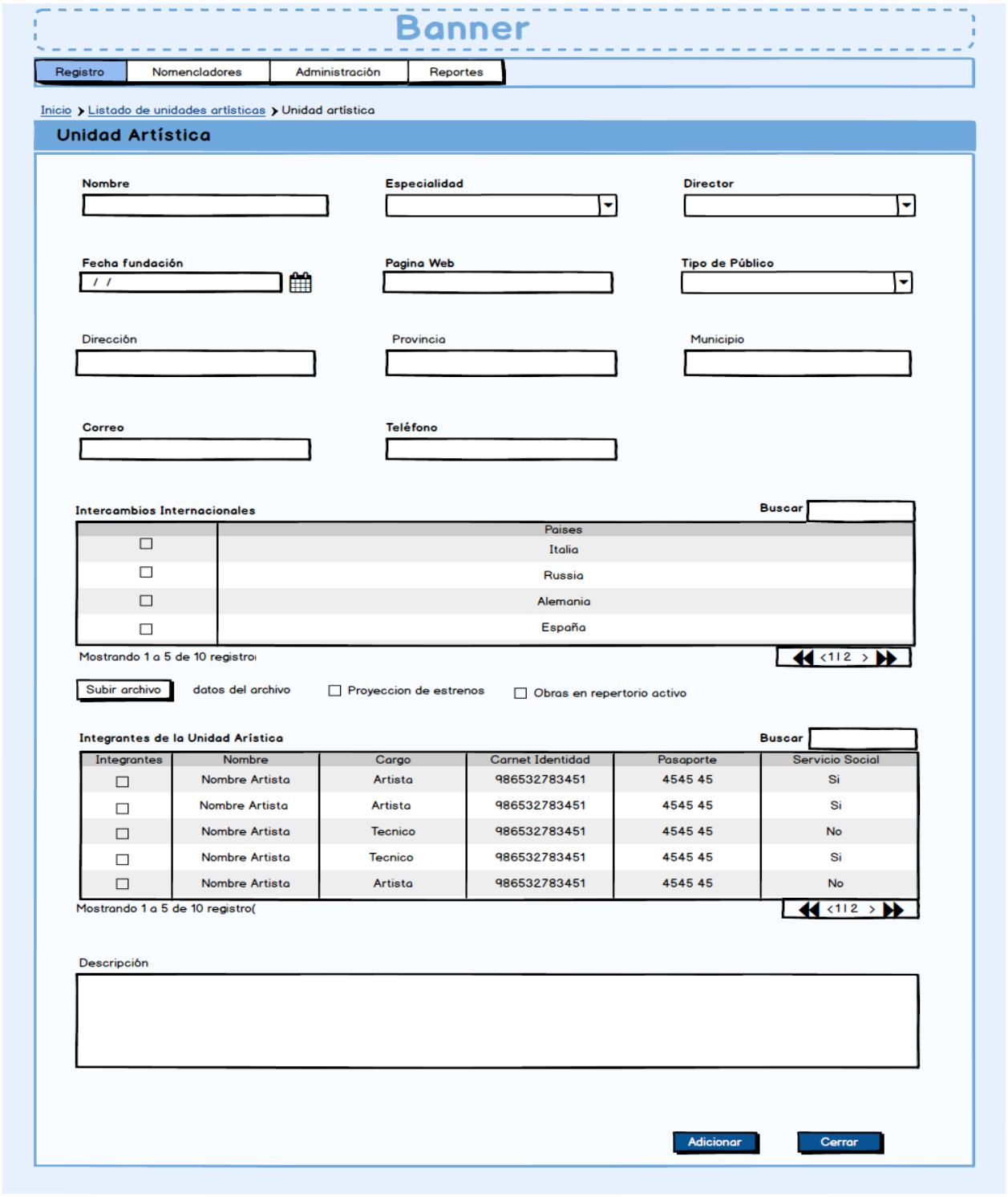
- **Correo:** Campo de texto. Se inserta el correo de la Unidad Artística. Admite solo los siguientes caracteres: letras, números, “@”, “-”, “_” y “.”
- **Intercambios internacionales:** Campo de selección. Se selecciona mediante una tabla los países en los que las Unidades Artísticas han realizado intercambios culturales.
- **Tipo de público:** Campo de selección, se selecciona el tipo de público al que va dirigido la Unidad Artística.
- **Integrantes:** Campo de selección, se selecciona mediante una tabla los integrantes que pertenecen a las Unidades Artísticas.

Se pueden realizar las siguientes acciones:

- **Adicionar**: Insertar una nueva Unidad Artística en el sistema.
- **Buscar:** Busca por cualquiera de los campos de las tablas Intercambio Internacional e Integrantes.
- **Cancelar:** Redirecciona a la interfaz anterior. No guarda los datos.

Observaciones:

Prototipo elemental de interfaz gráfica de usuario



Fuente: Elaboración propia

1.22 Arquitectura del sistema

El diseño arquitectónico es la primera etapa en el proceso de construcción del software. Constituye el enlace crucial entre el diseño y la ingeniería de requerimientos, ya que identifica los principales componentes estructurales en un sistema y la relación entre ellos. Su salida consiste en un modelo que describe la forma en que se organiza el sistema como un conjunto de componentes en comunicación (Sommerville, 2011).

Vista

En esta capa se encuentra todo lo que se refiere a la visualización de la información, el diseño, colores, estilos y la estructura visual de las páginas. En el desarrollo de la solución propuesta queda evidenciada esta capa a través del uso de los archivos JavaScript, HTML y CSS, que en conjunto con la librería de JavaScript JQuery, el marco de trabajo Bootstrap y el motor de plantillas Twig, son los encargados de crear las páginas de la aplicación. Según la estructura de carpetas que trae definida Symfony, las vistas de la aplicación se encontrarán en la carpeta "*view*" dentro de cada módulo (Center, IBM Knowledge, 2016).

Controlador

Esta capa es la responsable de procesar y mandar a mostrar los datos obtenidos en la capa de acceso a datos, es decir, trabaja de intermediaria entre la capa de la vista y la capa de acceso a datos. En la solución informática propuesta el uso de esta capa se observa a través de las clases controladoras que son las encargadas de realizar el procesamiento de los datos y lógica de negocio asociada a las peticiones realizadas desde la capa de la vista. Según la estructura de carpetas que trae definida Symfony, las clases controladoras de la aplicación se encontrarán en la carpeta "Controller" dentro de cada módulo (Center, IBM Knowledge, 2016).

Modelo

Capa de acceso a datos

Esta capa se encarga del manejo de la lógica de negocio, además sirve como intermediaria entre la capa controladora y el sistema gestor de base de datos. Las clases que componen esta capa son las gestoras y las clases repositorios, ubicadas en las carpetas Negocio y Repository respectivamente. Las clases gestoras reciben la información obtenida de la vista por medio de las controladoras y posteriormente se encargan de tratar dicha información en conjunto con las clases repositorios. Las clases repositorios contienen un conjunto de consultas para acceder y

realizar acciones sobre los datos de tablas específicas de la base de datos. En esta capa también se encuentra el ORM Doctrine que posibilita la separación de la aplicación respecto al sistema gestor de base de datos mediante su lenguaje propio de consultas DQL.

1.23 Modelo de diseño

Esta etapa de desarrollo del software representa el enlace entre los requisitos y la implementación del sistema. El objetivo de esta etapa es crear prototipos o modelos que describan cómo deben interactuar las clases, para el posterior desarrollo de la aplicación.

Patrones GRASP

Patrón Controlador: El objetivo de este patrón es asignar la responsabilidad de controlar el flujo de eventos del sistema, a clases específicas, o sea, delega la responsabilidad en otras clases con las que mantiene un modelo de alta cohesión. Este patrón se evidencia en la solución a través de las clases controladoras que se encuentran en la carpeta Controller que gestiona las acciones de cada vista. A continuación, la imagen.

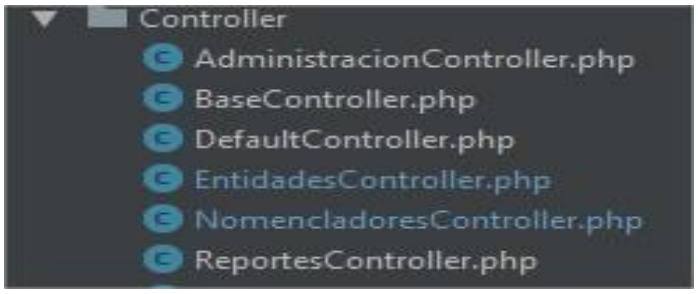


Figura 1: Representación de la carpeta Controller.

Fuente: Elaboración propia.

Patrón Experto: Este patrón define que la clase experta en información es la que debe llevar la responsabilidad, ya que esta última es la que contiene toda la información para cumplir con la responsabilidad. Este patrón se observa en la solución al crear las vistas para mostrárselas al usuario; así como en las clases Controladoras que son las expertas en información y las encargadas de crear los formularios y las vistas que se le mostrarán al usuario.

```php
class ArtistaType extends AbstractType
{
    /**
     * {@inheritdoc}
     */
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add( child: 'primerNombre',  type: ArqTextType::class, array(
                'attr' => array('maxlength' => 255,
                    'class' => 'form-control'),
                'valido' => array('letras'),
                'required' => true
            ))
            ->add( child: 'segundoNombre',  type: ArqTextType::class, array(
                'attr' => array('maxlength' => 255),
                'valido' => array('letras', 'espacio'),
                'required' => false
            ))
```

Figura 2: Representación del patrón experto.

Fuente: Elaboración propia.

Patrón Creador: En la implementación de SiGAE, este patrón se evidencia en las clases Gestoras en el momento de crear nuevas instancias de las clases Entidades. Otro ejemplo son las clases Controladoras, que son las responsables de la creación de los formularios que se mostrarán en las vistas.

```php
public function adicionarModificarCargoAction(Request $request, $id_cargo)
{
    $em = $this->getDoctrine()->getManager();
    $cargo = $this->crearCargo($id_cargo);
    $formCargo = $this->createForm( type: 'AppBundle\Form\SigaeForm\CargoType', $cargo);
```

Figura 3: Representación del patrón creador

Fuente: Elaboración propia.

Patrón Bajo acoplamiento: Este patrón plantea la baja dependencia que debe existir entre las clases, además está estrechamente relacionado con los patrones Experto o Alta Cohesión. Symfony favorece ampliamente el bajo acoplamiento de las clases en el sistema, esto ocurre porque a cada clase se le asignan solamente las responsabilidades necesarias de manera que no dependan en gran medida de otras.

Patrón Alta cohesión: Este patrón tiene como propósito asignar responsabilidades de manera que la cohesión siga siendo alta. El marco de trabajo Symfony favorece la alta cohesión asignando responsabilidades a las clases de tal manera que estas se encuentren estrechamente relacionadas entre sí y no lleguen a realizar un trabajo excesivo. Este patrón se pone de manifiesto en la interrelación que existe entre las clases controladora, las clases gestora y las funcionalidades del sistema, donde cada una de estas presenta una clase controladora y una gestora, la primera encargada de manejar la lógica de presentación y el flujo de los datos provenientes de la vista y la segunda encargada de manejar la lógica del negocio de cada funcionalidad.

Patrones GoF

Patrón Método de fabricación (Factory Method): Define una interfaz para crear un objeto, pero deja que sean las subclases quienes decidan qué clase instanciar; su objetivo es devolver una instancia de múltiples tipos de objetos, que normalmente provienen de una misma clase padre y sólo se diferencian entre ellos por algún aspecto de comportamiento. En el proyecto se puede ver que las clases controladoras provienen de una misma clase padre Controller.

```
class ArqComponentController extends Controller
```

Figura 4: Representación del patrón método de fabricación.
Fuente: Elaboración propia.

Patrón Decorador: El principal objetivo de este patrón es añadir responsabilidades a objetos concretos de manera dinámica y transparente sin afectar a otros objetos. Este patrón brinda más flexibilidad que la herencia estática y evita que las clases más altas en la jerarquía estén demasiado cargadas de funcionalidad y sean complejas. En el proyecto se ve en las clases de tipo Repository, ya que en ella se pueden agregar responsabilidades que no hay porque implementarlas directamente en la clase abstracta EntityRepository.

```php
class EntidadesRepository extends EntityRepository
{
    public function paisxartista($id_artista)
    {
        $qb = $this->getEntityManager()->createQueryBuilder();
        $qb->select( select: 'a_p.nombre')
            ->from( from: 'AppBundle:SigaeEntity\Artista', alias: 'a')
            ->innerJoin( join: 'a.pais', alias: 'a_p')
            ->andWhere($qb->expr()->eq( x: 'a.activo', y: 'true'))
            ->andWhere($qb->expr()->eq( x: 'a.id', y: ':prmArtista'))
            ->setParameter( key: 'prmArtista', $id_artista);
        $query = $qb->getQuery();
        $result = $query->getResult();
        return $result;
```

Figura 5: Representación del patrón decorador.

Fuente: Elaboración propia.

Patrón Inyección de dependencias: Este patrón es utilizado en los contenedores de servicios (o contenedores de inyección de dependencias), los cuales son objetos PHP que gestionan la creación de instancias de servicios, es decir, objetos.

```php
private function generateUrl($route, $parameters = array(), $referenceType = UrlGeneratorInterface::ABSOLUTE_PATH)
{
    return $this->container->get( id: 'router')->generate($route, $parameters, $referenceType);
}
```

Figura 6: Representación del patrón inyección de dependencias.

Fuente: Elaboración propia.

1.24 Diagrama de clases del diseño

El diagrama de clases recoge las clases de objetos y sus asociaciones. En este diagrama se representa la estructura y el comportamiento de cada uno de los objetos del sistema y sus relaciones con los demás objetos.

Con el fin de facilitar la comprensión del diagrama, se pueden incluir paquetes como elementos del mismo, donde cada uno de ellos agrupa un conjunto de clases (Cillero, Diagrama de Clases del Diseño, 2017).

A continuación, se muestra el diagrama de clases del diseño para el nomenclador “Cargo”

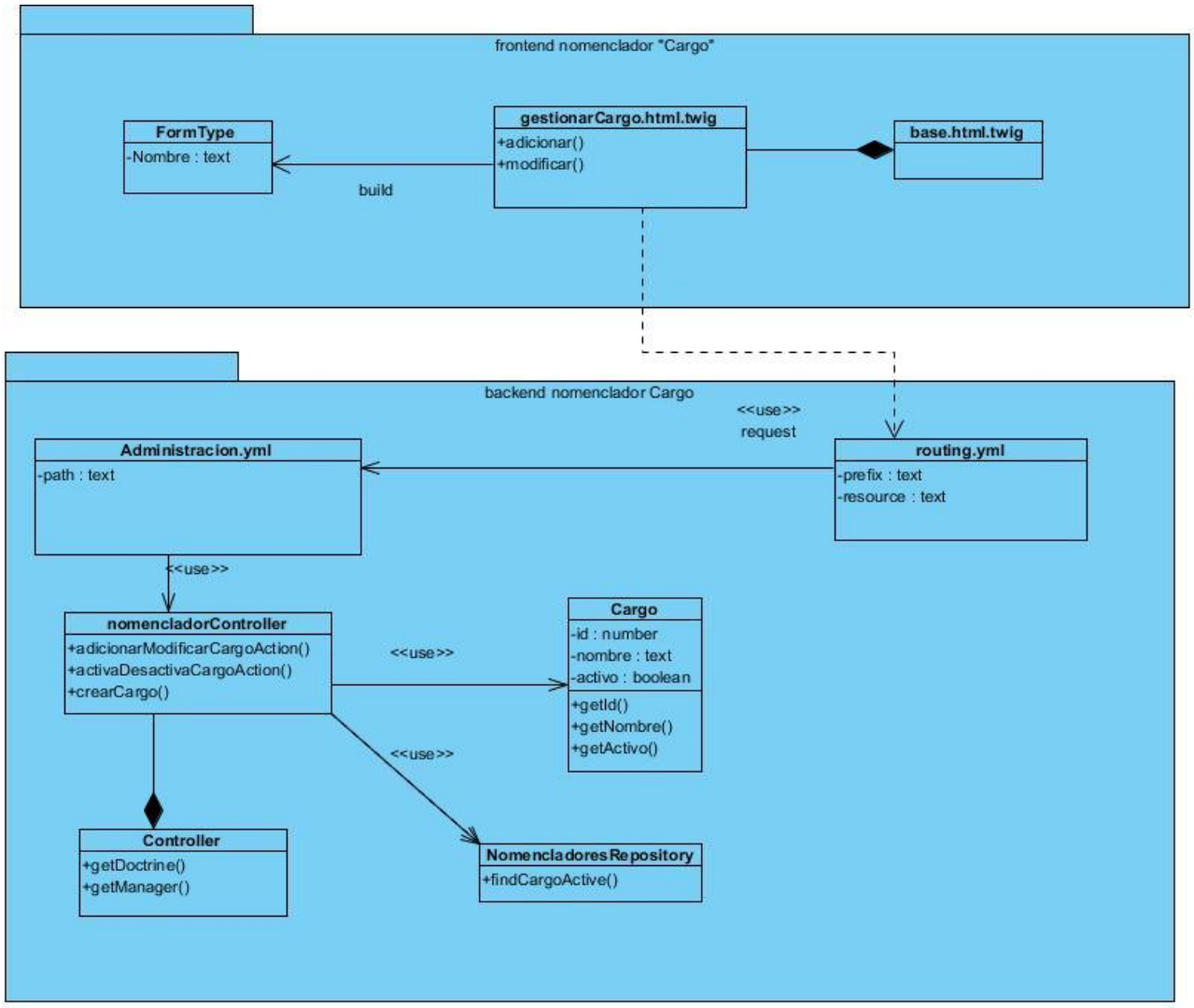


Figura 7: Diagrama de clases del diseño del nomenclador Cargo.

Fuente: Elaboración propia.

1.25 Diagrama de componentes

Un componente es un módulo de software que puede ser código fuente, código binario, un ejecutable, o una librería con una interfaz definida. Una interfaz establece las operaciones externas de un componente, las cuales determinan una parte del comportamiento del mismo. Además, se representan las dependencias entre componentes o entre un componente y la interfaz de otro, es decir uno de ellos usa los servicios o facilidades del otro.

Estos diagramas pueden incluir paquetes que permiten organizar la construcción del sistema de información en subsistemas y que recogen aspectos prácticos relacionados con la secuencia de compilación entre componentes, la agrupación de elementos en librerías, etc. (Cillero, Diagrama de Clases del Diseño, 2017).

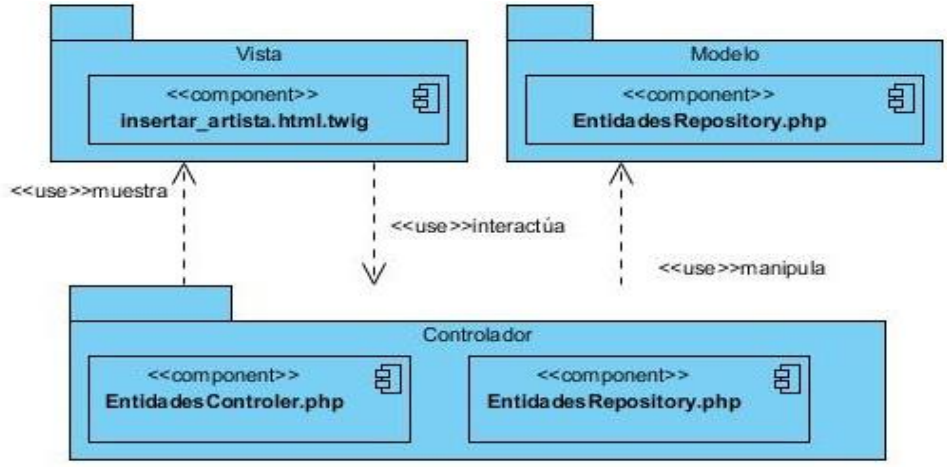


Figura 8: Diagrama de Componentes del sistema SiGAE.

Fuente: Elaboración propia.

1.26 Diagrama de despliegue

Este tipo de diagrama se realiza con el objetivo de mostrar las relaciones físicas de los distintos nodos que componen un sistema y la distribución de los componentes sobre dichos nodos. Además, se modela la arquitectura en tiempo de ejecución de un sistema (Cillero, Diagrama de despliegue, 2017).

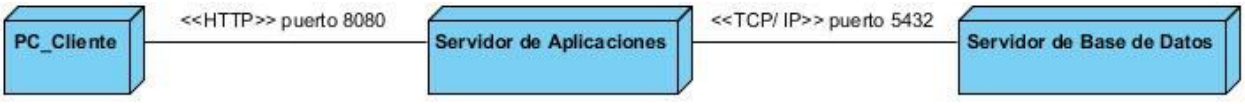


Figura 9: Diagrama de despliegue**.**

Fuente: Elaboración propia.

1.27 Modelo de datos

El modelo de datos se utiliza para describir la estructura lógica y física de la información persistente gestionada por el sistema. Se utiliza para definir la correlación entre las clases de diseño

persistentes y las estructuras de datos persistentes, y para definir las estructuras de datos persistentes (Tecnologías-Información, 2018).

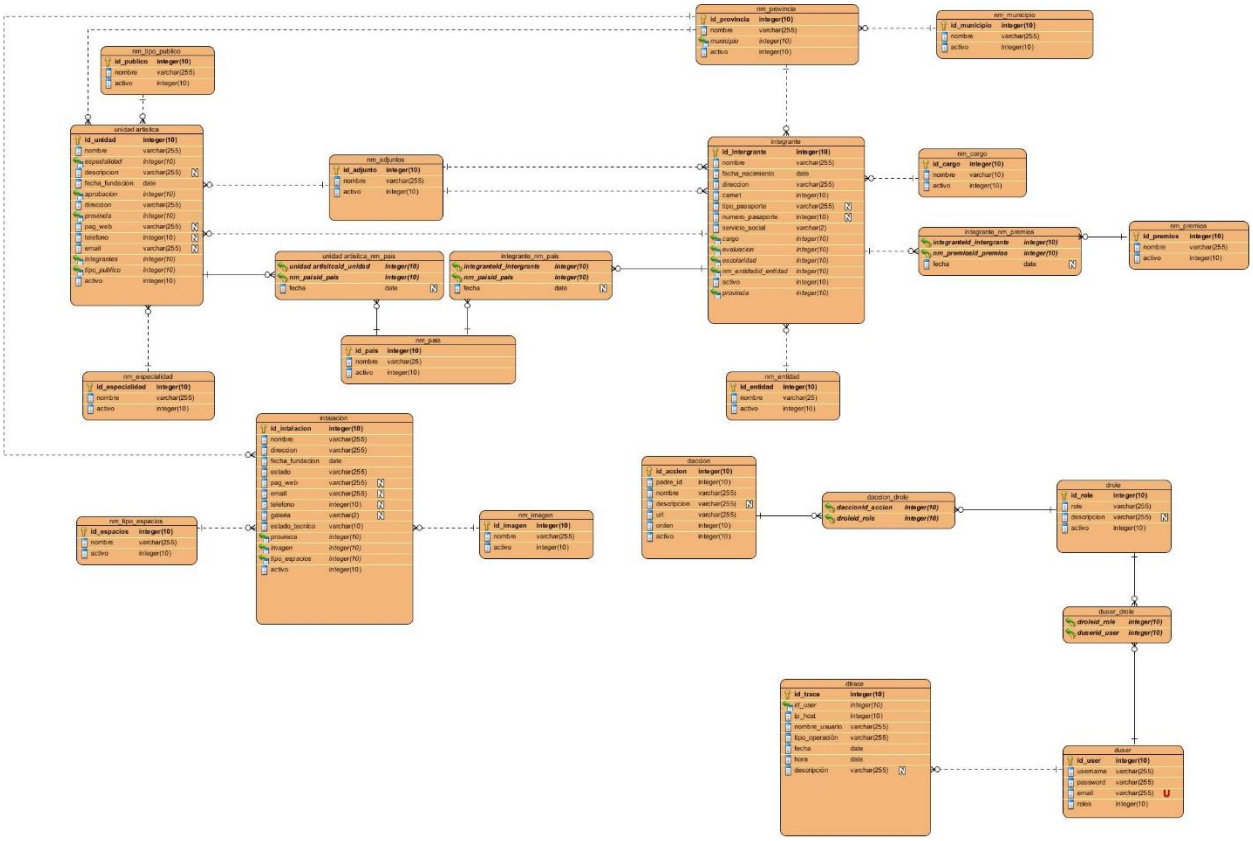


Figura 10: Diagrama de Modelo de Datos

Fuente: Elaboración propia.

El diagrama de modelo de datos del sistema SiGAE cuenta con 23 tablas, de ellos 11 son nomencladores, lo cuales son los encargados de gestionar los conceptos específicos del negocio ya definidos, por ejemplo, la tabla nm_provincia, que se relaciona con la tabla nm_municipio.

1.28 Estándares de codificación

Los estándares de codificación son normas que deben seguirse durante todo el ciclo de implementación del sistema (ohmyroot, 2017).

Los mismos fueron seleccionados por el equipo de arquitectura y diseño del proyecto, donde esta información se encuentra plasmada íntegramente en el documento CEGEL_SiGAE_Estandares_de_codificacicion_para_PHP.doc. A continuación, se describen algunos de los estándares más utilizados en la implementación del sistema SiGAE:

Declaración de clases

Las Clases deben ser nombradas de acuerdo a la convención de nombres.

La llave "{" deberá escribirse siempre en la línea debajo del nombre de la clase ("one true brace").

Todo el código contenido en una clase debe ser separado con cuatro espacios. Incluir código adicional en archivos de clase está permitido pero esta desaconsejado.

A continuación, se muestra un ejemplo de una declaración de clase que es permitida:

```
class EntidadesController extends BaseController
{

    // <editor-fold defaultstate="collapsed" desc="Artista">
    public function adicionarModificarArtistaAction(Request $request, $id_artista)
    {
        $em = $this->getDoctrine()->getManager();

        $artista = $this->crearArtista($id_artista);
        $cargos = $this->getDoctrine()->getRepository('AppBundle:SigaeEntity\Cargo')->findCargosActive();
        $premios = $this->getDoctrine()->getRepository('AppBundle:SigaeEntity\Premio')->findPremiosActive();
        $entidads = $this->getDoctrine()->getRepository('AppBundle:SigaeEntity\Entidad')->findEntidadesActive();
        $formArtista = $this->createForm( type: 'AppBundle\Form\SigaeForm\ArtistaType', $artista, array(
            'qb_cargos' => $cargos,
            'qb_premios' => $premios,
            'qb_entidads' => $entidads
        ));
```

Figura 11: Declaración de clases.

Fuente: Elaboración propia.

Funciones y Métodos

Los nombres de funciones pueden contener únicamente caracteres alfanuméricos. Los guiones bajos (_) no están permitidos. Los números están permitidos en los nombres de función, pero no se aconseja en la mayoría de los casos.

Los nombres de funciones deben empezar siempre con una letra minúscula. Cuando un nombre de función consiste en más de una palabra, la primera letra de cada nueva palabra debe estar en mayúsculas. Esto es llamado comúnmente como formato "camelCase".

Por norma general, se recomienda la elocuencia. Los nombres de función deben ser lo suficientemente elocuentes como para describir su propósito y comportamiento.

Estos son ejemplos de nombres de funciones admisibles:

```php
private function crearArtista($id_artista)
{
    $em = $this->getDoctrine()->getManager();
    if (!is_null($id_artista)) {
        $artista = $em->getRepository('AppBundle\Entity\Artista')->find($id_artista);
        if (!is_null($artista)) {
            return $artista;
        }
    }
    $artista = new Artista();
    $artista->setActivo(true);
    return $artista;
}
```

Figura 12: Funciones y Métodos.

Fuente: Elaboración propia.

Constantes

Las constantes pueden contener tanto caracteres alfanuméricos como barras bajas (_). Los números están permitidos. Todas las letras pertenecientes al nombre de una constante deben aparecer en mayúsculas. Las palabras dentro del nombre de una constante deben separarse por barras bajas (_). Por ejemplo:

```javascript
const Error = 'error';
const Success = 'success';
const Information = 'info';
const Warning = 'warning';
```

Figura 13: Definición de Constantes.

Fuente: Elaboración propia.

Comentarios en las funciones.

Todas las funciones deben tener un comentario, antes de su declaración, explicando que hacen. Ningún programador debería tener que analizar el código de una función para conocer su utilidad. Tanto el nombre como el comentario que acompañe a la función deben bastar para ello.

Ubicación y denominación de archivos.

Se ubicarán los archivos según las convenciones establecidas por Symfony3.4 o según las especificaciones del equipo de arquitectura.

Para la denominación de los archivos se seguirán las convenciones establecidas por Symfony3.4 o el equipo de arquitectura. Ejemplo:

Para las clases controladoras se usará el sufijo Controller: Ver figura 14

```
class AdministracionController extends BaseController
```

Figura 14: Ubicación y denominación.

Fuente: Elaboración propia.

Para las clases repositorio se usará el sufijo Repository:

```
class EntidadesRepository extends EntityRepository
```

Figura 15: Ubicación y denominación.

Fuente: Elaboración propia.

Estilo y reglas de escritura de código PHP

Nombres de variables.

Los nombres deben ser descriptivos y concisos. No usar ni grandes frases ni pequeñas abreviaciones para las variables. Siempre es mejor saber que hace una variable con sólo conocer su nombre. Esto aplica para los nombres de variables, funciones, argumentos de funciones y clases. Los nombres de las variables y de las funciones pueden iniciarcon letra minúscula, pero si estas tienen más de una palabra, cada nueva palabra debe iniciar con letra mayúscula. Las constantes deben de escribirse siempre en mayúsculas y tanto estas como las variables globales deben de tener como prefijo el nombre de la clase a la que pertenecen.

Las definiciones de la función

Los nombres de la función pueden contener sólo caracteres alfanuméricos. Los nombres de la función siempre deben empezar en letras minúsculas. Cuando un nombre de la función consiste de más de una palabra, la primera letra de cada nueva palabra debe capitalizarse. Ver figura 16.

```
public function paisPorUnidadartistica($id_unidadartistica)
{
    $qb = $this->getEntityManager()->createQueryBuilder();
    $qb->select( select: 'u_p.nombre')
        ->from( from: 'AppBundle:SigaeEntity\UnidadArtistica', alias: 'u')
        ->innerJoin( join: 'u.pais', alias: 'u_p')
        ->andWhere($qb->expr()->eq( x: 'u.activo', y: 'true'))
        ->andWhere($qb->expr()->eq( x: 'u.id', y: ':prmUnidad'))
        ->setParameter( key: 'prmUnidad', $id_unidadartistica);
    $query = $qb->getQuery();
    $result = $query->getResult();
    return $result;
}
```

Figura 16: Definición de funciones.

Fuente: Elaboración propia.

Llamadas a funciones

Deben llamarse las funciones sin los espacios entre el nombre de la función, el paréntesis de la apertura, y el primer parámetro; los espacios entre las comas y cada parámetro, y ningún espacial entre el último parámetro, el paréntesis del cierre, y el punto y coma. Ver figura 17

```
$cargos = $this->getDoctrine()->getRepository('AppBundle:SigaeEntity\Cargo')->findCargosActive();
$premios = $this->getDoctrine()->getRepository('AppBundle:SigaeEntity\Premio')->findPremiosActive();
```

Figura 17: Llamadas a funciones.

Fuente: Elaboración propia.

Siempre incluir las llaves.

En todo momento a la hora de codificar un bloque de instrucciones, éste debe ir encerrado entre llaves, aun cuando conste de una sola línea. Ver figura 18.

```
public function detallesArtistaAction($id_artista)
{
    $em = $this->getDoctrine()->getManager();
    $artista = $em->getRepository( className: 'AppBundle\Entity\SigaeEntity\Artista')->find($id_artista);
    $paises = $em->getRepository( className: 'AppBundle\Entity\SigaeEntity\Artista')->paisxartista($id_artista);
    $premios = $em->getRepository( className: 'AppBundle\Entity\SigaeEntity\Artista')->premioxartista($id_artista);
    $nameart = $artista->getPrimerNombre() . '_' . $artista->getPrimerAp();
    $ruta = $this->getParameter( name: 'file_directory') . $nameart . '/';
    return $this->render( view: 'AppBundle:Entidades/artista:detalles_artista.html.twig', array(
        'artista' => $artista,
        'paises' => $paises,
        'premios' => $premios,
        'ruta' => $ruta
    ));

}
```

Figura 18: Llaves

Fuente: Elaboración propia.

No utilizar variables sin inicializar.

Si no se tiene control sobre el valor de una variable, se debe verificar que esté inicializada. Esto lo permite PHP de la siguiente manera:

```
$nameart = $formArtista['primerNombre']->getData() . "_" . $formArtista['primerAp']->getData();
```

Figura 19: No utilizar variables sin inicializar.

Fuente: Elaboración propia.

Conclusiones del capítulo

El empleo de la metodología AUP en su variación para la UCI en función de describir el proceso de desarrollo del sistema propuesto, permitió organizar el desarrollo de la solución y generar los artefactos necesarios. Por otra parte, la aplicación de los patrones arquitectónicos MVC permitió un mejor entendimiento a la hora de la programación. El modelado de los procesos del negocio y la explicación detallada de estos, permitieron identificar con mayor facilidad los requisitos funcionales que el sistema debe satisfacer. Los patrones de diseño utilizados GRASP: Bajo Acoplamiento, Alta Cohesión, Experto, Controlador, Creador de los GoF, permitieron estructurar la solución respondiendo a diversos problemas en la implementación de los requisitos funcionales. El empleo de los estándares de codificación facilitó la lectura, comprensión y mantenimiento del código para los desarrolladores.

CAPÍTULO 3: VALIDACIÓN

Introducción

En este capítulo se evalúa el grado de calidad y fiabilidad de los requisitos obtenidos en el desarrollo de la investigación, donde esta evaluación se lleva a cabo a partir de la validación del diseño a través de las métricas aplicadas a las clases. Además, se aplican pruebas de software para verificar y revelar la calidad del producto realizado.

3.1 Técnicas de validación de requisitos

Con el objetivo de ratificar que los requisitos del software obtenidos definen el sistema que el cliente desea se llevó a cabo un proceso de validación de los mismos, para el cual se emplearon las siguientes técnicas:

Revisiones de los requisitos: se realizaron revisiones a cada uno de los requisitos por parte del equipo de desarrollo. Las revisiones internas generaron un total de 10 no conformidades de tipo técnicas y de ortografía, las cuales fueron corregidas satisfactoriamente en tiempo. Con el equipo de análisis y líder de proyecto se realizó la revisión en la cual se añadieron detalles a 4 requisitos funcionales y se aprobaron finalmente los mismos, generándose de este encuentro un acta de aceptación por parte del cliente.

Construcción de prototipos: se mostró al cliente una presentación que contenía los prototipos de interfaces del sistema, el cual permitió tener una visión preliminar de como se vería el sistema, se comprobó la satisfacción y aprobación del cliente, los mismos fueron aprobados satisfactoriamente.

Generación de casos de prueba: como parte del proceso de validación de los requisitos funcionales fueron diseñados casos de pruebas para cada historia de usuario.

3.2 Métricas aplicadas a los requisitos

Con el objetivo de medir la calidad de la especificación de los requisitos se aplicó una de las métricas propuestas para la metodología AUP (UCI), Calidad de la especificación (CE).

Para obtener cuán entendibles y precisos son los requisitos, primeramente, se calcula el total de requisitos de la especificación como se muestra a continuación:

Nr: el total de requisitos de especificación.

Nf: cantidad de requisitos funcionales.

Nnf: cantidad de requisitos no funcionales.

$$Nr = Nf + Nnf$$

Como resultado de la sustitución de los valores, para SiGAE se obtiene:

$$Nr = 53 + 12$$

$$Nr = 65$$

Para determinar, finalmente, la Especificidad de los Requisitos (ER) o ausencia de ambigüedad en los mismos se realiza la siguiente operación:

$$ER = Nui / Nr$$

Donde **Nui** es el número de requisitos para los cuales todos los revisores tuvieron interpretaciones idénticas. Mientras más cerca de 1 esté el valor de ER, menor será la ambigüedad.

Para el caso de los requisitos obtenidos para el SiGAE, 7 produjeron contradicción en las interpretaciones. Sustituyendo las variables se obtiene:

$$ER = 60/ 65$$

$$ER = 0.92$$

Arrojando un resultado final satisfactorio, indicando que el grado de ambigüedad de los requisitos es sumamente bajo (8%) ya que el 92% son entendibles. Los requisitos ambiguos fueron modificados y validados para garantizar una correcta interpretación.

3.3 Validación de los componentes

En función de validar el diseño realizado se aplicaron las siguientes métricas de diseño: Relaciones entre Clases (RC) y Tamaño Operacional de Clase (TOC).

3.3.1 Tamaño Operacional de Clase(TOC)

La métrica TOC permite medir la responsabilidad, la complejidad de implementación y la reutilización de las clases del diseño. Es importante destacar que, para esta métrica, la responsabilidad y la complejidad son inversamente proporcionales a la reutilización, por lo que, a mayor responsabilidad y complejidad de implementación de una clase, menor será su nivel de reutilización (Pressman R. S., 2010).

Resultados obtenidos al aplicar la métrica TOC:

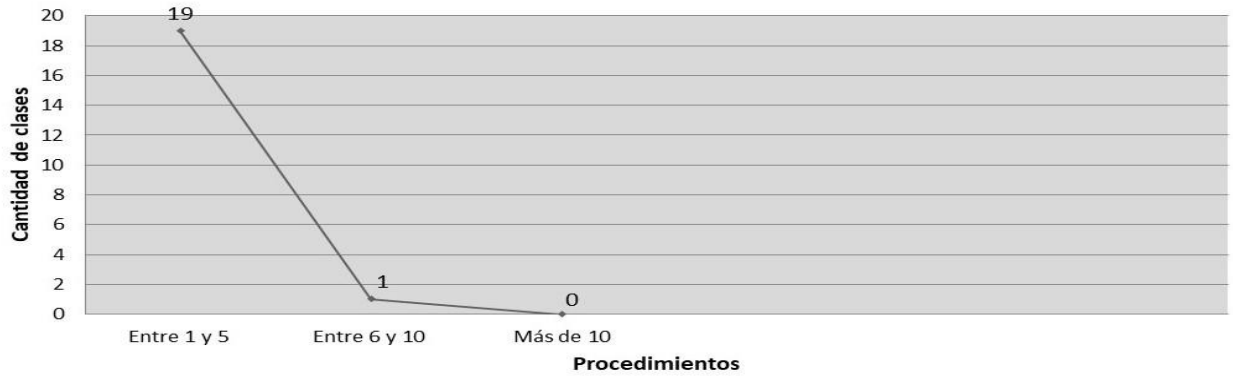


Figura 20: Representación de la cantidad de clases por cantidad de procedimientos.

Fuente: Elaboración propia.

Figura 21: Representación en porciento (%) de la cantidad de procedimiento, del nivel de responsabilidad, complejidad de la implementación y reutilización de las clases.

Fuente: Elaboración propia.

Luego de aplicada la métrica se observa que las clases del diseño del sistema no se encuentran sobrecargadas en cuanto a responsabilidades y el nivel de complejidad de las mismas no es muy alto, lo que favorece en gran medida la reutilización de estas clases.

3.3.2 Relaciones entre Clases (RC)

La métrica RC permite evaluar el acoplamiento, la complejidad de mantenimiento, la reutilización y la cantidad de pruebas de unidad necesarias para probar una clase teniendo en cuenta las relaciones existentes entre ellas.

Resultados obtenidos al aplicar la métrica TOC:

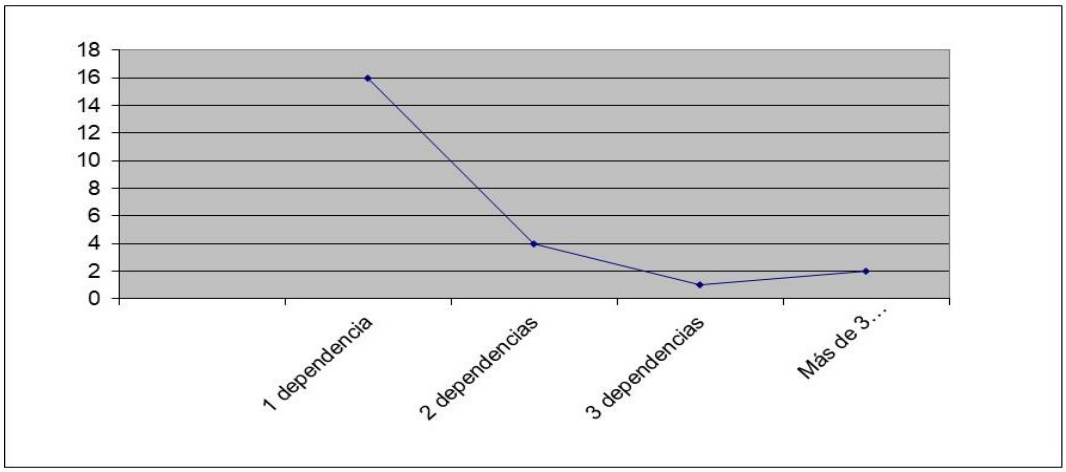


Figura 22: Representación de la cantidad de clases por cantidad de relaciones de usos que poseen.

Fuente: Elaboración propia.

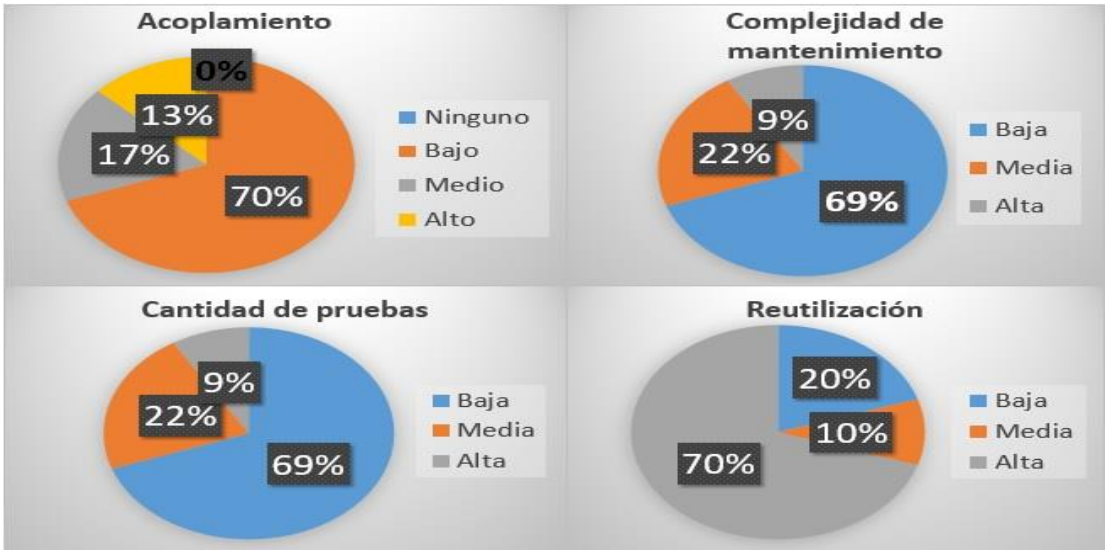


Figura 23: Representación en porciento (%) del nivel de acoplamiento, complejidad de mantenimiento, cantidad de pruebas y reutilización de las clases.

Fuente: Elaboración propia.

Una vez aplicada la métrica y teniendo en cuenta el umbral definido para validar el diseño (Bien [0.1; 0.3], Regular [0.4; 0.7] y Mal [0.8; 1]), se obtiene como resultado que las clases del diseño del portal web promueven el bajo acoplamiento, la complejidad de mantenimiento y la cantidad de pruebas no son altas y en consecuencia el grado de reutilización es mayor.

En sentido general, los resultados obtenidos de la aplicación de las métricas TOC y RC demuestran que el diseño del sistema para el CNAE no es complejo, que las clases presentan bajo acoplamiento y un alto grado de reutilización.

3.4 Pruebas de software

Según IEEE (2009), las pruebas de software "consisten en verificar el comportamiento de un programa dinámicamente a través de un grupo finito de casos de prueba, debidamente seleccionados, en relación al comportamiento esperado". Permite verificar y revelar la calidad del producto antes de ser entregado al cliente. Esto no asegura que el sistema estará limpio de errores y funcionará de acuerdo a las especificaciones (Jústiz Núñez & Gómez Suárez, 2014).

Específicamente las pruebas de software permiten evaluar las soluciones y determinar el nivel de calidad que poseen, por lo que se debe definir un proceso que se pueda emplear en el entorno de desarrollo de aplicaciones informáticas en la universidad.

3.4.1 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 posible componentes de prueba ejecutables para automatizar las pruebas (Sánchez, 2015).

Pruebas unitarias

Las pruebas unitarias se realizan sobre las funcionalidades internas de un módulo y se encargan de comprobar los caminos lógicos, ciclos (bucles) y condiciones que debe cumplir el programa (Pressman R. S., 2010).

La prueba de caja blanca del software se basa en el minucioso examen de los detalles procedimentales. Se comprueban los caminos lógicos del software proponiendo casos de prueba que ejerciten conjuntos específicos de condiciones. Se puede examinar el estado del programa en varios puntos para determinar si el estado real coincide con el estado deseado o esperado. La prueba de caja blanca es un método de diseño de casos de prueba que usa la estructura de control del diseño procedimental para obtener los casos de prueba (Pressman R. S., 2010).

Para aplicar este método se define la técnica de ruta básica.

Técnica de ruta básica

La técnica de ruta básica es empleada en el método de caja blanca, su objetivo es comprobar que cada camino se ejecute independiente de un componente o programa, obteniendo la complejidad lógica del diseño. La idea es derivar casos de prueba a partir de un conjunto dado de caminos independientes por los cuales puede circular el flujo de control. Para obtener dicho conjunto de caminos independientes se construye el Grafo de Flujo asociado y se calcula su complejidad ciclomática (Pressman R. S., 2010).

Pressman propone como estrategia para aplicar la ruta básica, realizar un análisis de la complejidad ciclomática de cada procedimiento que componen las clases del sistema; una vez concluido este paso se selecciona el método con valor de contener errores, además de que ofrece una medida del número de pruebas que deben diseñarse para validar la correcta implementación de una determinada función.

Para obtener los casos de prueba a partir de la técnica seleccionada se debe construir el grafo de flujo correspondiente al código de la función como se muestra en la Figura 24.

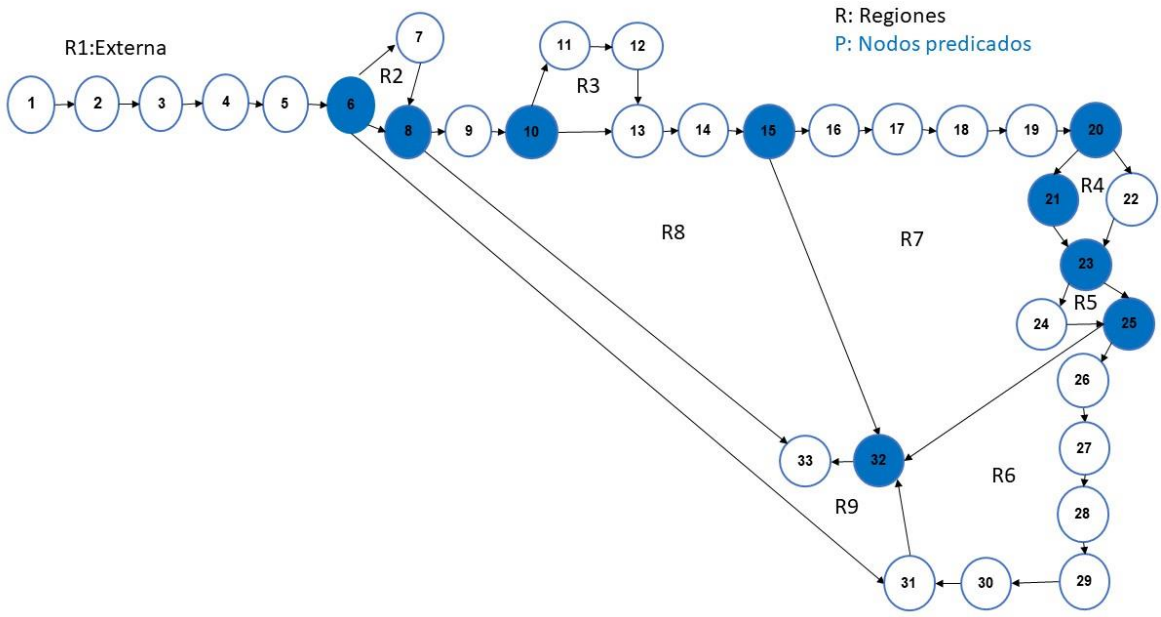


Figura 24: Grafo del flujo del método adicionarModificarUnidadArtisticaAction.

Fuente: Elaboración propia.

Luego se determina la complejidad ciclomática V(G) del grafo resultante, el cual indica el número de caminos independientes que existen en un grafo, es cualquier camino dentro del código que introduce por lo menos un nuevo conjunto de sentencias de procesos o una nueva condición. La complejidad ciclomática se puede calcular de 3 formas:

1. V (G) = a - n + 2, siendo a el número de arcos o aristas del grafo y n el número de nodos.

2. V (G) = r, siendo r el número de regiones cerradas del grafo.

3. V (G) = c + 1, siendo c el número de nodos de condición.

Realizando los cálculos correspondientes se obtiene por cualquiera de las variantes el siguiente resultado:

V (G) = a - n + 2	V (G) = r	V (G) = c + 1
V (G) = 40 - 33 + 2	V (G) = 9	V (G) = 8 + 1
V (G) = 9		V (G) = 9

El valor obtenido una vez aplicada la complejidad ciclomática, representa el límite superior de pruebas que deberán aplicarse (Pressman R. S., 2010).

Por lo que el conjunto de caminos básico sería:

Camino básico 1: 1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-32-33

Camino básico 2: 1-2-3-4-5-6-31-33

Camino básico 3: 1-2-3-4-5-6-8-9-10-13-14-15-32-33

Camino básico 4: 1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-16-17-18-19-20-21-22-23-24-25-32-33

Camino básico 5: 1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-16-17-18-19-20-22-23-25-32-33

Camino básico 6:1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-16-17-18-19-20-21-23-24-25-26-27-28-29-30-31-33

Camino básico 7: 1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-16-17-18-19-20-21-22-23-24-25-26-27-28-29-30-31-33

Camino básico 8: 1-2-3-4-5-6-8-9-10-13-14-15-16-17-18-19-20-22-23-24-25-26-27-28-29-30-31-33

Camino básico 9: 1-2-3-4-5-6-8-9-10-13-14-15-16-17-18-19-20-22-23-25-32-33

Luego se definen los casos de prueba para cada uno de los caminos básicos obtenidos. A continuación, se presenta el caso de prueba definido para la ruta independiente 1.

Tabla 5: Caso de prueba de la ruta independiente 1

Descripción	Se inserta una unidad artística asociada al CNAE.
Condición de ejecución	La unidad artística debe tener un director asociado.
Entradas	Se introduce el nombre de la unidad artística, nombre del director, fecha de creada, dirección y los aristas asociados.

Resultados esperados	Se debe mostrar una alerta especificando que la unidad artística se ha creado correctamente.

Fuente: Elaboración propia.

Resultados al aplicar la técnica de ruta básica

Esta técnica se aplicó a los métodos de las clases controladoras; dichas clases fueron seleccionadas debido a que engloban las funcionalidades del sistema. Para comprobar la fiabilidad del código fuente se realizaron dos iteraciones completas en busca de errores de codificación, como condición de parada se tuvo en cuenta el criterio de estimación anteriormente expuesto. Se realizó una primera iteración donde se obtuvo como resultado 0 cantidad de No conformidades, por lo que se puede concluir que luego de realizar la prueba de caja blanca, donde se diseñaron y ejecutaron los casos de prueba correspondientes, se logró asegurar el cumplimiento del proceso de mejora del código.

Pruebas funcionales

Las pruebas funcionales son pruebas diseñadas tomando como referencia las especificaciones funcionales de un componente o sistema (lo que se va a testear, el software o una parte de él). Se realizan para comprobar si el software cumple las funciones esperadas (Pressman R. S., 2010).

Método de Caja Negra

Las pruebas de caja negra se centran en los requisitos funcionales del software, es decir, la prueba de caja negra permite obtener conjuntos de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa. La prueba de caja negra intenta encontrar errores de las siguientes categorías (Pressman R. S., 2010):

- Funciones incorrectas o ausentes.
- Errores de interfaz.
- Errores en estructuras de datos o en accesos a las bases de datos externas.
- Errores de rendimiento.
- Errores de inicialización y de terminación.

Se utiliza la técnica de Partición equivalente para llevar a cabo el método de caja negra, a continuación, se describe.

Técnica de prueba: Partición equivalente

Esta es una técnica de prueba de Caja-Negra que divide el dominio de entrada de un programa en clases de datos de los que se pueden derivar casos de prueba. El diseño de estos casos de prueba se basa en la evaluación de las clases de equivalencia para una condición de entrada, así mismo, una condición de entrada es un valor numérico, un rango de valores, un conjunto de valores relacionados o una condición lógica. Una clase de equivalencia representa un conjunto de estados válidos y no válidos para las condiciones de entrada (Pressman R. S., 2010).

Para aplicar esta técnica, se deben primeramente realizar el Diseños de casos prueba (DCP) con el objetivo de obtener un conjunto de pruebas que tengan la mayor probabilidad de descubrir los defectos del software.

Un DCP es, en ingeniería del software, un conjunto de condiciones o variables bajo las cuales un analista determinará si una aplicación o una característica de éstos es parcial o completamente satisfactoria (Pressman R. S., 2010).

Con el objetivo de comprobar que las funcionalidades del sistema se realizaron un total de 3 iteraciones de pruebas de caja negra, obteniéndose los siguientes resultados:

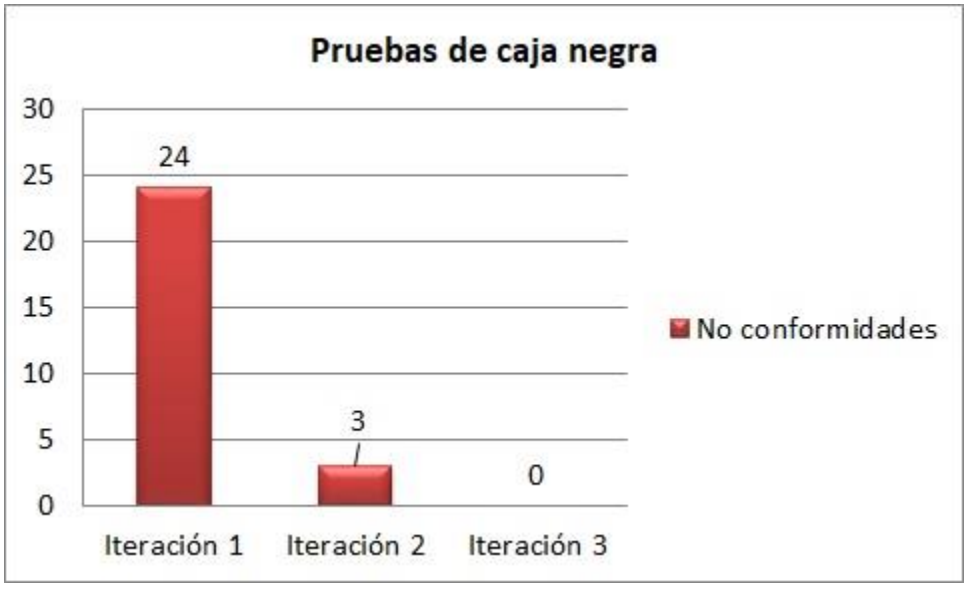


Figura 25: Gráfico de no conformidades.

Fuente: Elaboración propia.

En una primera iteración se detectaron un total de 24 no conformidades, generalmente errores ortográficos y errores de validación. En la segunda iteración se obtuvieron 3 no conformidades donde aún existían dos errores de validación y se detectó un problema de escritura. Finalmente, en una tercera iteración se obtuvieron resultados satisfactorios al no detectarse no conformidades.

3.5 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 *(Sánchez, 2015)*.

3.6 Pruebas de aceptación

Es la prueba final antes del despliegue del sistema. Su objetivo es verificar que el software está listo y que puede ser usado por usuarios finales para ejecutar aquellas funciones y tareas para las cuales el software fue construido (Sánchez, 2015).

3.7 Validación de las variables de investigación

Las deficiencias descritas en la introducción del presente trabajo de diploma afectan directamente el cumplimiento de las actividades fundamentales en el CNAE. Para demostrar la existencia del problema y aplicar los instrumentos seleccionados se definieron las siguientes variables de investigación:

Variable Dependiente: la celeridad en las consultas y la persistencia de la misma.

Variable Independiente: gestionar la información asociada a los artistas e instituciones culturales.

A partir de los resultados obtenidos mediante la entrevista realizada a la jefa de Recursos Humanos del Consejo Nacional de las Artes Escénicas (Ver Anexo 1) y la observación por parte del equipo de desarrollo (Ver Anexo 3, se verificaron las pruebas correspondientes al sistema de la presente investigación con una reducción considerable del tiempo en el registro de la información. Con el uso de los indicadores aplicados se utilizaron las mismas mediciones con la puesta en práctica del sistema. Los resultados fueron los siguientes:

Tabla 6: Validación de la variable dependiente.

Indicador	Antes	Después
1-Tiempo de búsqueda	El tiempo de búsqueda de un documento de forma manual es aproximadamente de 10-15 minutos.	El tiempo de búsqueda después de implementado el sistema es de 2.64 segundos.
2-Tiempo para crear un artista	El tiempo para crear un nuevo artista de forma manual es aproximadamente de 5-10 minutos.	El tiempo que demora el sistema en crear un artista es de 3.11 segundos.
3- Tiempo para crear una unidad artística	El tiempo para crear una nueva unidad artística de forma manual es aproximadamente de 5-15 minutos.	El tiempo que demora el sistema en crear una unidad artística es de 3.30 segundos.
4- Tiempo para crear una instalación cultural	El tiempo para crear una nueva instalación cultural de forma manual es aproximadamente de 5-15 minutos.	El tiempo que demora el sistema en crear una instalación cultural es de 2.70 segundos.
5-Tiempo para realizar un reporte	El tiempo para realizar un reporte de forma manual depende del estado de la documentación y de su ubicación, normalmente se demoran 20 minutos.	El tiempo para realizar un reporte es de 1.50 segundos.
6- Persistencia de la información.	La información se almacenaba en cajas o estantes.	Se almacena de forma digital en una base de datos.

Fuente: Elaboración propia

Después de realizada la evaluación de las variables, analizar los resultados y las características de la solución de la actual investigación se puede demostrar que el empleo del sistema SiGAE mejorará la celeridad de los servicios que realiza el CNAE lo que se verá reflejado en una mejor atención a los artistas, a los directores de las instituciones y de las unidades artísticas. Además, se garantizará la persistencia de la información gracias a la utilización de la base de datos creada

para el centro, se realizarán copias de seguridad cada 15 días, para garantizar que no se pierda información en caso de haber fallos técnicos, permitirá la búsqueda de información en menor tiempo. El sistema permitirá el trabajo concurrente, se definen niveles de seguridad mediante roles. No estará limitada la labor de los trabajadores ya que todos podrán hacer uso de sistema, facilitando las búsquedas y la agilidad en los procesos.

Conclusiones del capítulo

La aplicación de técnicas de validación de requisitos permitió ratificar la correspondencia de los mismos con las solicitudes del cliente. Además, el uso de la métrica CE proporcionó una medida cuantitativa de la calidad de la especificación de estos, quedando excluidas las ambigüedades existentes para su corrección. Por su parte, la validación del diseño mediante las métricas TOC y RC permitió obtener, de forma general, el grado de complejidad de implementación y mantenimiento, de responsabilidad, reutilización, acoplamiento y la cantidad de pruebas necesarias para realizar a las clases favoreciendo la creación de un diseño lo más sencillo posible, de fácil mantenimiento e implementación y que promoviera la reutilización. Además, las pruebas de Caja Negra permitieron comprobar que las funciones son operativas a través de la interfaz del software, que la entrada se acepta de forma adecuada y se produce un resultado correcto, manteniendo así la integridad de la información externa. Por su parte, las pruebas de Caja Blanca permitieron comprobar internamente las funciones del sistema, proporcionando la detección de no conformidades para su corrección. Se realizó una valoración del comportamiento de las variables que forman parte del problema de la investigación, demostrando que los componentes desarrollados mejoran la celeridad de los procesos que maneja el CNAE y la persistencia de la información.

CONCLUSIONES GENERALES

- Se establecen los referentes teóricos en los que se sustenta la propuesta de solución mediante el estudio y el análisis de los principales conceptos que se manejan en el CNAE, para una mejor comprensión de negocio, también en el estudio de estado del arte el cual nos brindó una noción referente a las tendencias utilizadas en el desarrollo de sitios web de gestión a nivel nacional e internacional.
- Se definió la metodología AUP en su variación para la UCI, la cual guió la realización de los procesos de análisis y diseño, constituyó la guía fundamental en el proceso de desarrollo del Sistema para la Gestión de Artistas e Instituciones Culturales para el Consejo Nacional de las Artes Escénicas y permitió generar todos los artefactos necesarios.
- Se logró una correcta implementación del Sistema para la Gestión de Artistas e Instituciones Culturales para el Consejo Nacional de las Artes Escénicas, gracias a una correcta selección de las herramientas y el uso de los estándares de codificación definidos por el equipo de desarrollo.
- La solución se logró validar correctamente a través de las métricas y pruebas de software aplicadas para garantizar la celeridad y persistencia de la información en el CNAE.

REFERENCIAS

Acosta, D. L. (2017). *Framework Design: A Role Modeling Approach.* Obtenido de https://sedici.unlp.edu.ar/handle/10915/19306

Autores, V. (2018). *Revista ENLACE*. Obtenido de https://enlacearquitectura.com/que-es-una-instalacion-artistica/

Avila, J. H. (2012). *issuu*. Obtenido de https://issuu.com/dr.marcomendozacorbetto/docs/id99-xv2_-_introducci__n_al_derecho

Bootstrap Team. (2018). *Bootstrap*. Obtenido de Bootstrap: https://getbootstrap.com/docs/4.1.1/getting-started/introduction/

Cano, J. H. (12 de noviembre de 2003). *Metodologías ágiles en el desarrollo de software.* Obtenido de http://issi.dsic.upv.es/archives/f-1069167248521/actas.pdf

***Capacity Information Tegnology Academy*. (2016).** Obtenido de http://blog.capacityacademy.com/2013/03/16/jquery-que-es-origenes-ventajas-desventajas/

Center, IBM Knowledge. (2016). *Patrón de diseño modelo-vista-controlador.* Obtenido de https://www.ibm.com/support/knowledgecenter/es/SSZLC2_8.0.0/com.ibm.commerce.developer.doc/concepts/csdmvcdespat.html

Cillero, M. (2017). Obtenido de Diagrama de Clases del Diseño: https://manuel.cillero.es/doc/metrica-3/tecnicas/diagrama-de-clases/

Cillero, M. (2017). *Diagrama de despliegue.* Obtenido de https://manuel.cillero.es/doc/metrica-3/tecnicas/diagrama-de-despliegue/

Consejo Nacional de las Artes Plásticas. (2011-2017). *Portal de las Artes Visuales en Cuba*. Obtenido de http://www.cnap.cult.cu/instituciones/agencia-de-autores-visuales

***doctrine-project.org*. (2018).** Obtenido de https://www.doctrine-project.org/projects/doctrine-orm/en/2.6/reference/dql-doctrine-query-language.html

Domíngez, D. (2018). *Todo Programación. Nº 13. Págs. 32-34.* Editorial Iberprensa (Madrid).

Dutoit, D. (2014). *sensagent*. Obtenido de http://diccionario.sensagent.com/Entorno%20de%20desarrollo%20integrado/es-es/

Formativa, A. (2016). *Aula Formativa*. Obtenido de https://blog.aulaformativa.com/definicion-usos-ventajas-lenguaje-html5/

Gaines, J., Boyd, G., & Copley, D. (2017). *Visual Paradigm Online*. Obtenido de https://online.visual-paradigm.com/es/features/

Gobierno de Colombia. (2018). *MINCULTURA*. Obtenido de http://www.mincultura.gov.co/ministerio/sistemas-de-informacion/Paginas/default.aspx

Hernandez, J. (2014). En *Análisis y Desarrollo web* (pág. 33).

Humberto Cervantes Maceda, P. V.-E. (2015). Arquitectura de software. Conceptos y ciclo.

Jústiz Núñez, D., & Gómez Suárez, D. (2014). Proceso de pruebas para productos de software en un laboratorio de calidad.

Marit, M. (2019). *Información importante sobre el DBMS "POSTGRESQL"*. Obtenido de http://postgresql-dbms.blogspot.com/p/importante-sobre-el-dbms-postgresql.html

Ministerio de Cultura República de Cuba. (15 de 11 de 2018). *Ministerio de Cultura República de Cuba*. Obtenido de http://www.ministeriodecultura.gob.cu/directorio/consejo-nacional-de-artes-escenicas-cnae/

Ministerio de Comunicaciones. (2017). *1"POLÍTICA INTEGRAL PARA EL PERFECCIONAMIENTO DE LA INFORMATIZACIÓN DE LA SOCIEDAD EN CUBA".* La Habana.

Netcraft. (2018). *web Server Survey*. Obtenido de www.webServer.com

Nube Colectiva. (2019). ***NC(Nube Colectiva)*.** Obtenido de Que es MVC (Modelo Vista Controlador) y otros detalles: http://blog.nubecolectiva.com/que-es-mvc-modelo-vista-controlador-y-otros-detalles/

ohmyroot. (12 de enero de 2017). *Estandares de codificación.* Obtenido de https://www.ohmyroot.com/buenas-practicas-legibilidad-del-codigo/

Pérez Porto, J., & Gardey, A. (2017). *Definicion.de*. Obtenido de Definicion.de: https://definicion.de/celeridad/

Pérez Porto, J., & Merino, M. (2014). *Definicion.de*. Obtenido de Definicion.de: https://definicion.de/persistencia/

Palacios, J. (2016). *Los elementos de una buena Historia de Usuario*. Obtenido de https://jeronimopalacios.com/2016/05/los-elementos-una-buena-historia-de-usuario/

Pérez Porto, J. (2014). *definicion.de*. Obtenido de https://definicion.de/unidadartistica/

Pérez Porto, J. (2018). *definicion.de*. Obtenido de https://definicion.de/instalacióncultural/

Pérez Porto, J., & Merino , M. (2014). *definicion.de*. Obtenido de https://definicion.de/artista/

***PostgreSQL Global Development Group*. (29 de 10 de 2018).** Obtenido de https://www.postgresql.org/about/licence/

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

Reynoso, C. (2011). *Introducción a la Arquietectura de Software*. Obtenido de http://carlosreynoso.com.ar/archivos/arquitectura/Arquitectura-software.pdf

Robles, V. (2018). *¿Que es Git y para que sirve?* Obtenido de https://victorroblesweb.es/2018/04/28/que-es-git-y-para-que-sirve/

Rodríguez Sánchez, T. (2015). *Metodología de desarrollo para la actividad productiva de la UCI.* La habana , Cuba.

Sanchez, R. (2 de octubre de 2018). *Especificación de Requisitos Software según el estándar de IEEE 830*. Obtenido de https://www.academia.edu/6647065/Especificaci%C3%B3n_de_Requisitos_Software_seg%C3%BAn_el_est%C3%A1ndar_de_IEEE_830

Sánchez, T. R. (2015). *Metodología de desarrollo para la Actividad productiva de la UCI. .* La Habana. Cuba.

Sommerville, I. (2011). *Software engineering.* Mexico: D.R. © 2011 por Pearson Educación de México, S.A. de C.V.

T**ecnologías-Información. (2018).** *Modelo de datos.* Obtenido de https://www.tecnologias-informacion.com/modeladodatos.html

The Apache Software Fundation. (2019). *Apache HTTP Server Project*. Obtenido de https://httpd.apache.org/docs/2.4/es/

***Tutorial de SQL*. (2012).** Obtenido de https://www.um.es/geograf/sigmur/sigpdf/temario_9.pdf

Upper. (2018). *Upper*. Obtenido de https://www.uppertechnology.com/una-herramienta-para-la-creacion-de-bocetos-balsamiq-mockups/

***w3schools*. (2018).** Obtenido de https://www.w3.org/Style/CSS/Overview.en.html

Zaya, A. R. (2018). *ITgallery*. Obtenido de https://www.itgalleryapp.com/es/caracteristicas-software-gestion-galerias-arte.html

Zoega, G. V. (2015). *Mapeador de Objeto Relacional*.